UNIX®

RESEARCH SYSTEM

PROGRAMMER'S MANUAL

Tenth Edition

Volume I

Computing Science Research Center
AT&T Bell Laboratories
Murray Hill, New Jersey

© 1990, American Telephone and Telegraph
Company, Bell Laboratories Division

All rights reserved. No part of this
publication may be reproduced or transmitted
in any form or by any means, electronic or
mechanical, including photocopy, recording,
or any information storage and retrieval
system, without permission in writing from
AT&T

Unix Research System Programmer's Manual, Tenth Edition Volume I
ISBN 0-03-047532-5

123 018 987654321

PREFACE TO THE TENTH EDITION

This edition of the *UNIX Programmer's Manual* appears on the twentieth anniversary of the original research system pioneered by Ken Thompson and Dennis Ritchie. It is dedicated to the memory of the late Lee McMahon, whose influence pervades the system's approaches to text processing and data networking.

As with previous editions, the manual reports on the system as it exists in its original home department at AT&T Bell Laboratories. Although related to the commercial System V and the academic BSD systems, the research system is just what its title implies. Some of the facilities have long since graduated from the lab into international commerce; others are experiments in progress. Their appearance here does not constitute an announcement of any official product.

Much has happened between the landmark seventh edition of 1979, the last edition to be published as trade book, and today. The magnitude of the changes may perhaps be most visible in the papers gathered in Volume 2, more than half of which are completely new. Even 'old stuff' has been substantially updated. For example, Volume 2 gives more up-to-date information about the popular *troff* family of typesetting software than does any other published work.

Careful examination of Volume 1 will also show enormous changes since the seventh edition. The distinctive theme is distributed computing. Ritchie's coroutine 'stream' input/output and the Datakit® virtual circuit switch realization by Lee McMahon and Bill Marshall provide the basis for networking. Peter Weinberger's network file system makes it painless. Rob Pike's software for windowed bitmap terminals extends the system right out to your fingertips, giving a multiplexed interface to an already multiplexed system. Dave Presotto's communication software brings harmony to the Babel of real-world communications. Norman Wilson's battles with entropy have made the tenth edition system trimmer than its less capable predecessors.

Besides the facilities just listed, the manual describes dozens of completely new programs in areas ranging from protocol verification to optical character recognition. Practically every page has been improved in wording if not in function. Instructive examples, often chosen to suggest nonobvious uses of the system, have been added. Sections 4, 5, and 8 have been expanded to provide more inside wisdom about the system.

Over the course of years dozens of people at Bell Laboratories have contributed to the software here described. Their work, acknowledged in previous editions, is their monument. Acknowledgement is also made to the Regents of the University of California for certain portions that descend from work in the Department of Electrical Engineering and Computer Science at Berkeley. Active contributors to this edition include, besides those already mentioned, H. S. Baird, J. L. Bentley, S. A. Browning, L. L. Cherry, W. R. Cheswick, M. A. Derr, T. D. S. Duff, C. W. Fraser, D. M. Gay, P. Glick, F. T. Grampp, E. H. Grosse, D. R. Hanson, G. J. Holzmann, A. G. Hume, B. W. Kernighan, T. Killian, A. R. Koenig, T. J. Kowalski, W. T. Marshall, M. D. McIlroy, S. C. North, S. Peeters, R. Pike, S. A. Rago, J. A. Reeds, R. Sethi, B. Stroustrup, H. W. Trickey and others. Their computing activities have been made possible by the smoothly running machines and Datakit networks managed by P. Glick, F. T. Grampp, W. T. Marshall, and E. J. Sitar. E. H. Carter consulted on book production.

A. G. Hume
M. D. McIlroy
October, 1989

TABLE OF CONTENTS

Preface 1

Contents 3

Introduction 5

Topical Index 11

Permuted Index 21

Glossary 51

Unix Software

- Commands (1) 59
- System Calls (2) 291
- Subroutines (3) 335
- Special Files (4) 447
- File Formats and Conventions (5) 479
- Language Conventions (6) 519
- Information Sources (7) 535
- Maintenance Commands and Procedures (8) 561
- Teletype 5620-related Software (9) 635

INTRODUCTION

This volume describes facilities of the version of the UNIX operating system that is used at the system's original home, the computing science research center of AT&T Bell Laboratories. Volume 2 discusses certain aspects of the system in more detail. For tutorial information, consult a standard reference such as *The Unix Programming Environment* by Kernighan and Pike.

Within the area it surveys, this volume attempts to be timely, complete and concise. Where the latter two objectives conflict, the obvious is often left unsaid in favor of brevity. It is intended that each program be described as it is, not as it should be. Inevitably, as snapshots of a living system, some of the descriptions may soon go out of date. When that happens, users may find the latest word in an on-line form of this book.

The volume is divided into sections; each section consists of a number of independent pages. (Some 'pages' spill over beyond one sheet of paper.) The name of a page is in its upper corners together with the section number. Sections 3 and 9 have further substructure as described in their respective introductions. Pages are arranged alphabetically within sections. Here and in much other documentation, the notation *man*(1) refers to the page for the *man* command in section 1. (This particular command prints selections from the on-line manual.)

There are nine printed sections and one unprinted appendix.

1. Commands
2. System calls
3. Subroutines
4. Devices, line disciplines, and file systems
5. File formats and conventions
6. Language conventions
7. Information sources
8. Maintenance commands and procedures
9. Teletype 5620-related software

A. Other facilities

Section 1 describes programs intended to be invoked directly by the user, in contradistinction to subroutines, which are intended to be called by the user's programs. Essential commands generally reside in directory `/bin` (for *bin* ary programs) or `/usr/bin`. These directories are searched automatically by command interpreters.

Section 2 describes entries into the UNIX® kernel as expressed in C, the primary implementation language of the system.

Section 3 discusses an assortment of subroutines, mostly for use with C. The primary libraries in which they are kept are described in *intro*(3). Fortran users should consult the *port*(3) mathematical library.

Section 4 discusses 'files' that actually refer to I/O devices, processing modules that modify the behavior of some devices, and 'file systems' that aren't just files on a disk.

Section 5 gives the structure of particular kinds of files, for example, the form of the output of the loader and assembler. Files internal to only one command are excluded, for example compiler intermediate files.

Section 6 lists macro packages, fonts, mail addressing conventions, etc.

Section 7 is a miscellaneous collection of information sources: library orders, news, astronomy, geography, calendar, etc.

Section 8 discusses commands and procedures not intended for use by the ordinary user. Most of the commands and files described here are kept in directory `/etc`.

Section 9 describes software intended for use with the Teletype 5620 terminal. In this category fall utilities that exploit the terminal and tools for writing code to run with or on it.

Appendix A contains information about obsolescent or specialized programs. It is not printed, but is accessible on line through *man*(1).

Not all facilities are available across all computers. Pages for programs with limited distribution are marked at the top with the name of at least one computer at the home site where they are known to reside.

All pages are based on a common format, not all parts of which necessarily appear.

The *NAME* subsection lists the exact names of the commands and subroutines covered and a very short description of their purpose.

The *SYNOPSIS* subsection summarizes the use of the program being described.

The *DESCRIPTION* subsection discusses in detail the subject at hand.

The *EXAMPLES* subsection illustrates interesting usage.

The *FILES* subsection gives the names of files which are built into the program.

The *SEE ALSO* subsection gives pointers to related information.

The *DIAGNOSTICS* subsection discusses the diagnostic indications which may be produced. Messages which are intended to be self-explanatory are not listed.

The *BUGS* subsection gives known bugs and sometimes deficiencies. Occasionally also the suggested fix is described.

The following conventions are used, particularly in section 1:

`Program font` shows literals, file names, and examples.

Italics designate parameters or arguments. In descriptions italics also name programs, as a sign that the programs can be found described in this manual, and as a reminder that the name is probably a shorthand, not a real file name.

Square brackets [ ] around an argument indicate that the argument is optional.

Ellipses '...' indicate that the previous item may be repeated.

A final convention is used by the commands themselves. An argument beginning with a minus sign − usually means a command option even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with −.

Following this introduction are a classified table of contents, a permuted index, and a glossary.

HOW TO GET STARTED

This section sketches the basic information you need to get started on UNIX: how to log in and log out, how to communicate through your terminal, and how to run a program. See Kernighan and Pike for a more complete introduction.

Logging in. You must call UNIX from an appropriate terminal. Most any ASCII terminal capable of full duplex operation and generating the entire character set can be used. You must also have a valid user name, which may be obtained, together with necessary telephone numbers or computer names, from the system administrator. After a data connection is established, the login procedure depends on what kind of terminal you are using.

Dial-up terminals: If there is a half/full-duplex switch, set it at full-duplex. When you have established a data connection, the system types out a few characters, intended to be `login:`. If they appear as gibberish, the computer is sending at the wrong baud rate. Depress the 'break' key; this is a signal that a different speed terminal is in use. The system will type `login:` at another speed. Continue depressing the break key until `login:` appears, then respond with your user name. End each line you type with a carriage return.

Hard-wired terminals. Hard-wired terminals usually begin at the right speed, up to 19200 baud; otherwise the preceding instructions apply.

Datakit® terminals. Make sure that any on-line/off-line switch is turned to on-line. If the terminal types gibberish, depress the 'break' key repeatedly until you see the prompt `Number please:` or `Destination please:`. Respond with the name of the computer you wish to connect to, followed by a carriage return. When the computer prompts `login:` proceed as above.

For all terminals, it is important that you type your name in lower case if possible. If you type upper-case letters, the system will assume that your terminal cannot generate lower-case letters and will translate all subsequent upper-case letters to lower case.

Upon logging in you may be presented with a 'message of the day'. You may also be notified of incoming mail. The final evidence that you have successfully logged in is that a shell program will type a prompt (normally `$`) to you. (The shell is described below under 'How to run a program.')

Terminals come with a bewildering variety of capabilities, which the system may have to be aware of. Get somebody who knows to help customize a `.profile` description of your terminal to set things properly upon logging in. For more information, consult *sh*(1) about profiles, *environ*(5) about environments, and *tabs*(1), *stty*(1), *tty*(4), and *ttyld*(4) about terminal settings.

Logging out. There are two ways to log out:

Type an end-of-file indication (EOT character, control-`d`) to the shell. The shell will terminate and a `login:` or `Number please:` message will appear again.

Simply hang up or turn your terminal off.

How to communicate through your terminal. When you type characters, a gnome deep in the system gathers your characters and saves them in a secret place. The characters will not be given to a program until you type a return.

UNIX terminal I/O is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is typing at you. If you type while the computer is printing, the input may appear interleaved with the output. However, whatever you type will be saved up and interpreted in correct sequence.

In typed input, the character `@` kills all the preceding characters in the line, so typing mistakes can be repaired. A backspace erases the last character typed. Successive backspaces erase characters back to, but not beyond, the beginning of the line. `@` and backspace can be transmitted to a program by preceding them with `\`. (So, to erase `\`, you need two backspaces.)

The 'delete' or 'rubout' key causes an *interrupt signal*, as does the 'break' key. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you don't want (with varying degrees of promptness). However, programs can arrange either to ignore this signal altogether, or to be notified, instead of being killed, when it happens. The editor, *ed*(1), for example, catches interrupts and stops what it is doing, instead of terminating, so that an interrupt can be used to halt an editor printout without losing the file being edited.

It is also possible to suspend output temporarily by typing control-`s` and later resume output with control-`q`.

The *quit* signal is generated by typing control-`\` or control-`|`. It not only causes a running program to terminate but also generates a file with the core image of the terminated process. Quit is useful for debugging.

How to run a program; the shell. When you have successfully logged in, a program called the shell is listening to your terminal. The shell reads typed-in lines, splits them up into a command name and arguments, and executes the command. A command is simply an executable program, which the shell finds by looking in several standard places. You can place commands in your own directory and have the shell find them there. There is nothing special about system-provided commands except where they live.

The command name is always the first word on an input line; it and its arguments are separated from one another by spaces.

When a program terminates, the shell will ordinarily regain control and type a prompt at you to indicate that it is ready for another command.

The shell (and the underlying system) has a distinctive ability to combine programs by piping the output of one into the input of another. In general, programs have been designed to be usable in pipelines as well as on their own. The shell has many other capabilities, which are described in detail in *sh*(1).

Bitmap terminals. To get the most out of the Tenth Edition system, you will probably wish to use a Teletype 5620 or 630 terminal, run with *mux*(9.1). This provides mouse control, graphics, and multiple layers, each of which acts like a separate, simultaneously running UNIX terminal, complete with its own shell. Text in a *mux*

layer can be edited on screen. The editor *sam*(9.1) and the typesetter simulator *proof*(9.1) are particularly useful programs that run only under *mux.*

The current directory. UNIX has a file system arranged in a hierarchy of directories. When you were given a user name by a system administrator, you were also given a home directory (ordinarily with the same name as your user name). When you log in, any file name you type is by default in this directory. Since you are the owner of this directory, you have full permission to read, write, alter, or destroy its contents. Permissions to have your will with other directories and files will have been granted or denied to you by their owners. As a matter of observed fact, few users protect their files from perusal by other users.

You need not do all your work in the home directory. To switch to another directory use the change directory command *cd,* described in *sh*(1).

Path names. To refer to files not in the current directory, you must use a path name. Full path names begin with `/`, the name of the 'root' of the whole file system. After the slash comes the name of a subdirectory, another slash and a sub-subdirectory, and so on until finally a file name is reached. For example, `/usr/lem/filex` refers to the file `filex` in the directory `lem`; `lem` is itself a subdirectory of `usr`; `usr` springs directly from the root.

If your current directory has subdirectories, the path names of files therein begin with the name of the subdirectory with no initial `/`.

A path name may be used anywhere a file name is required.

Important commands which modify the contents of files are *cp*(1), *mv* (described in *cp*(1)), and *rm*(1), which respectively copy, move (i.e. rename) and remove files. To find out the status of files or directories, use *ls*(1). See *mkdir*(1) for making directories.

For a fuller discussion of the file system, see Kernighan and Pike. It may also be useful to glance through section 2 of this manual, which discusses system calls, even if you don't intend to deal with the system at that level.

Environment. Many things can be changed in the environment: the already-mentioned erase character, properties of your terminal, directories from which you wish to execute commands, news notification, etc. Your `.profile` file can make these settings automatically upon login.

Writing a program. The principal language processors are *cc*(1) for C, *f77*(1) for Fortran, and *awk*(1). Processors for Basic, Pascal, Snobol, Lisp, Macsyma, ML, and others are available. For compile-before-run languages enter the text of a source program into a file, using one of the editors, *ed*(1), *sam*(9.1), or *vi*(1). By convention C programs are stored in files with names ending in `.c`, Fortran programs in `.f`. After the program text has been entered through the editor and written on a file, you can give the file to the appropriate language processor as an argument. The output of a compiler will be left in a file named `a.out` in the current directory. (If the output is precious, use *mv* to move it to a less exposed name soon.)

When you have finally gone through this entire process without provoking any diagnostics, the resulting program can be run by giving its name (`a.out` if you've not arranged otherwise) to the shell in response to the shell prompt.

Your programs can receive arguments from the command line just as system programs do, see *exec*(2).

Text processing. Almost all text is entered through the editors. The commands most often used to display text are *cat*(1), *pr*(1), and *troff*(1).

The *cat* command simply dumps text on the terminal, with no processing at all. The *pr* command paginates the text, supplies headings, and has a facility for multi-column output.

Troff prepares documents for a phototypesetter. Used naked, it requires careful forethought, but for ordinary documents it has been tamed; see *ms*(7). It was used to produce this manual. Its output can drive various devices including laser printers and graphic terminals; see *lp*(1) and *proof*(9).

Status inquiries. Who(1) prints a list of users presently logged in. *Date*(1) prints the current time and date. *Pwd*(1) tells what directory you're working in; *where*, described on the same page, tells what machine you're on. *Ls*(1) will list the files in your directory or give status information about particular files.

Surprises. Certain commands provide inter-user communication. Even if you do not plan to use them, it would be well to learn something about them, because someone else may aim them at you.

To communicate with another user currently logged in, *write*(1) may be used, but *mail*(1) is both more polite and more reliable. *Mail* leaves a message that will be announced to another user at next login, if not sooner. *Vismon*(9.1) and *sh*(1) provide ways to monitor incoming mail continuously.

Multiple machines. Tenth Edition systems have tentacles to many other machines. The file systems of other UNIX systems may be accessible through directory `/n`. Thus you may change current directory to one of these machines, although computing on your behalf is still being done on the machine you logged into. You may transmit data among such machines just as within one machine by using *cp*(1) or I/O redirection in the shell. You may also log into a remote machine to do processing there with input and output passing through the local machine; see *con*(1).

Datakit connections exist throughout Bell Laboratories. File transmission, over Datakit and over dialup lines to even more remote machines, is provided by *uucp*(1). Mail extends all over the world. See *mail*(6) for how to send mail to destinations on networks such as the ARPA Internet, Bitnet, or ACSnet (in Australia.)

INDEX BY TOPIC

Commands and Data Files

Shell and Shell Scripts

=, ==, =p, ==p − redo previous shell command . . . =(1) 61
apply, pick − repeatedly apply a command; select arguments . . . APPLY(1) 68
basename, dirname − strip filename affixes . . . BASENAME(1) 80
echo, printf − print arguments . . . ECHO(1) 126
expr − evaluate arguments as an expression . . . EXPR(1) 134
printenv − print environment . . . PRINTENV(1) 207
rc, cd, wait, whatis − command language . . . RC(1) 215
sh, cd, wait, whatis − shell, the standard command programming language . . . SH(1) 230
tee − pipe fitting . . . TEE(1) 258
test, [, newer − condition commands . . . TEST(1) 259
tr − translate characters . . . TR(1) 264
true, false − provide truth values . . . TRUE(1) 266

Manipulating Directories

chmod − change mode . . . CHMOD(1) 96
dir − format of directories . . . DIR(5) 487
dired − directory editor . . . DIRED(1) 121
find − find files . . . FIND(1) 140
idiff – interactive file comparison . . . IDIFF(1) 154
ls, lc – list contents of directory . . . LS(1) 171
mkdir – make a directory . . . MKDIR(1) 187
pwd, where – machine name and working directory . . . PWD(1) 212
rc, cd, wait, whatis − command language . . . RC(1) 215
rm − remove (unlink) files . . . RM(1) 225
sh, cd, wait, whatis − shell, the standard command programming language . . . SH(1) 230
touch, chdate − set modification or access date of a file . . . CHDATE(1) 95
worm − format of worm disks . . . WORM(5) 514

Manipulating Files

awk − pattern-directed scanning and processing language . . . AWK(1) 74
cat − catenate and print . . . CAT(1) 89
cbt − btree utilities . . . CBT(1) 91
chmod − change mode . . . CHMOD(1) 96
cmp − compare two files . . . CMP(1) 99
col, 2, 3, 4, 5, 6, mc, fold, expand − column alignment . . . COLUMN(1) 100
comm − select or reject lines common to two sorted files . . . COMM(1) 101
cp, mv, ln, reccp − copy, move, or link files . . . CP(1) 103
crypt, encrypt, decrypt − encode/decode . . . CRYPT(1) 105
cut, paste − rearrange columns of data . . . CUT(1) 109
dd, dblbuf − convert and copy a file . . . DD(1) 116
diff, diff3 − differential file comparison . . . DIFF(1) 119
file – determine file type . . . FILE(1) 139
find − find files . . . FIND(1) 140
gre, grep, egrep, fgrep – search a file for a pattern . . . GRE(1) 149
idiff – interactive file comparison . . . IDIFF(1) 154
join – relational database operator . . . JOIN(1) 156
look – find lines in a sorted list . . . LOOK(1) 168
ls, lc – list contents of directory . . . LS(1) 171
pack, unpack, pcat, compress, uncompress, zcat – compress and expand files . . . PACK(1) 195
pr – print file . . . PR(1) 203
random, fortune − sample lines from a file, return cookies . . . RANDOM(1) 214
rc, cd, wait, whatis − command language . . . RC(1) 215
rev, revpag − reverse lines or pages . . . REV(1) 224
rm − remove (unlink) files . . . RM(1) 225
sed − stream editor . . . SED(1) 226
sh, cd, wait, whatis − shell, the standard command programming language . . . SH(1) 230
sort − sort and/or merge files . . . SORT(1) 242
split, fsplit − split a file into pieces . . . SPLIT(1) 246
strings − find printable strings in a file . . . STRINGS(1) 247
sum, treesum − sum and count blocks in a file . . . SUM(1) 251
tail, readslow − deliver the last part of a file . . . TAIL(1) 253
touch, chdate − set modification or access date of a file . . . CHDATE(1) 95
tsort − topological sort . . . TSORT(1) 267

uniq − report repeated lines in a file . . . UNIQ(1) 270
vis − show invisible characters . . . VIS(1) 280
wc − word count . . . WC(1) 281
xd, od – hex, octal, decimal, or ASCII dump . . . XD(1) 286

Privacy and Security

chmod − change mode . . . CHMOD(1) 96
chown − change owner or group . . . CHOWN(8) 566
crypt, encrypt, decrypt − encode/decode . . . CRYPT(1) 105
getty − set terminal mode . . . GETTY(8) 575
getuid, id – get user identity . . . GETUID(1) 145
init − process control initialization . . . INIT(8) 578
newgrp – change to a new group . . . NEWGRP(1) 190
passwd – change login password . . . PASSWD(1) 196
passwd, group − password and group files . . . PASSWD(5) 496
su, setlog − substitute userid temporarily, become super-user . . . SU(8) 617

Editors

ed, e − text editor . . . ED(1) 127
ex, edit, vi − text editor . . . VI(1) 276
graphdraw graphpic − edit (combinatoric) graphs, convert to pic files . . . GRAPHDRAW(9) 644
icon − icon editor . . . ICON(9) 646
jf − font editor . . . JF(9) 647
ped, tped − picture editor . . . PED(9) 656
pico – graphics editor . . . PICO(1) 200
rebecca − graphics touch-up editor . . . REBECCA(9) 663
sam − screen editor with structural regular expressions . . . SAM(9) 665
sed − stream editor . . . SED(1) 226

Programming − In C

11as, 11cc, 11ld, 11nm, 11ranlib, 11reloc, 11size, 11strip − pdp11 support . . . 11(8) 561
3cc, 3as, 3ar, 3ld, 3nm, 3size, 3strip, cprs − MAC-32 C compiler . . . 3CC(9) 637
CC, cfront − C++ compiler . . . C++(1) 87
cb − C program beautifier . . . CB(1) 90
cc, lcc − C compilers . . . CC(1) 92
cin − C interpreter . . . CIN(1) 97
cpp − C language preprocessor . . . CPP(8) 569
lint, cyntax, cem − C program verifiers . . . LINT(1) 165
m4 – macro processor . . . M4(1) 173
make – maintain collections of programs . . . MAKE(1) 178
mk, mkconv, membername − maintain (make) related files . . . MK(1) 183

Programming − In Other Languages

altran, cospan, esterel, icon, lisp, macsyma, maple, ops5, pascal, ratfor, S, smp, sno, spitbol, struct, twig − languages . . . LANGS(1) 159
as − assembler . . . AS(1) 71
awk − pattern-directed scanning and processing language . . . AWK(1) 74
f2c – Convert Fortran 77 to C or C++ . . . F2C(1) 135
f77 – Fortran 77 compiler . . . F77(1) 136
lex – generator of lexical analysis programs . . . LEX(1) 164
make – maintain collections of programs . . . MAKE(1) 178
mk, mkconv, membername − maintain (make) related files . . . MK(1) 183
sml – Standard ML compiler . . . SML(1) 239
snocone − snobol with syntactic sugar . . . SNOCONE(1) 240
yacc, eyacc − yet another compiler-compiler . . . YACC(1) 287

Programming − Libraries and Object Files

32ld − bootstrap loader for 5620 . . . 32LD(9) 636
a.out − object file format . . . A.OUT(5) 479
ar − archive (library) file format . . . AR(5) 482
ar, ranlib − archive and library maintainer . . . AR(1) 70
ld – link editor or loader . . . LD(1) 162
libc − standard library functions . . . LIBC(9) 686
lorder – find ordering relation for an object library . . . LORDER(1) 169
nm – name list (symbol table) . . . NM(1) 192
size − size of an object file . . . SIZE(1) 237
stab − symbol table types . . . STAB(5) 503
strip − remove symbols and relocation bits . . . STRIP(1) 248

Programming − Debugging and Tuning

adb − debugger . . . ADB(1) 62
bigcore, coreid − permit big core images, identify source of image . . . BIGCORE(1) 85
core − format of memory image file . . . CORE(5) 485
hang – start a process in stopped state . . . HANG(1) 150
lcomp, lprint – line-by-line profiler . . . LCOMP(1) 161

pi, 3pi – process inspector . . . PI(9) 658
prof – display profile data . . . PROF(1) 208
time – time a command . . . TIME(1) 263

Writing Papers – Troff and Troff Preprocessors

cip – draw pictures for typesetting . . . CIP(9) 641
dag – preprocessor for drawing directed graphs . . . DAG(1) 110
doctype – guess command line for formatting a document . . . DOCTYPE(1) 124
eqn, neqn, checkeq – typeset mathematics . . . EQN(1) 132
eqnchar – special character definitions for eqn . . . EQNCHAR(6) 520
font – typesetter fonts . . . FONT(6) 521
grap – pic preprocessor for drawing graphs . . . GRAP(1) 146
ideal – troff preprocessor for drawing pictures . . . IDEAL(1) 152
make – maintain collections of programs . . . MAKE(1) 178
mbits – macros to typeset bitmaps . . . MBITS(6) 526
mcs – macros for formatting cover sheets . . . MCS(6) 527
mk, mkconv, membername – maintain (make) related files . . . MK(1) 183
movie, stills – algorithm animation . . . MOVIE(9) 650
mpictures – picture inclusion macros . . . MPICTURES(6) 528
mpm – macros for page makeup . . . MPM(6) 529
ms – macros for formatting manuscripts . . . MS(6) 530
ped, tped – picture editor . . . PED(9) 656
pic, tpic – troff and tex preprocessors for drawing pictures . . . PIC(1) 197
prefer, pinvert, penter, plook, pconvert – maintain and use bibliographic references . . . PREFER(1) 204
refer, lookbib, pubindex – maintain and use bibliographic references . . . REFER(1) 222
tbl – format tables for nroff or troff . . . TBL(1) 256
troff, nroff – text formatting and typesetting . . . TROFF(1) 265

Writing Papers – Other Formatters

fmt – ultra-simple text formatter . . . FMT(1) 142
latex, bibtex – tex macro package and bibliographies . . . LATEX(6) 522
monk, monksample, monkspell, monkmerge – typeset documents and letters . . . MONK(1) 188
tex, initex, virtex, dvips, dvit – text formatting and typesetting . . . TEX(1) 261

Writing Papers – Writing Aids

deroff, demonk, detex, delatex – remove formatting requests . . . DEROFF(1) 118
docgen – generate a document from a script . . . DOCGEN(1) 122
ptx – permuted index . . . PTX(1) 210
spell – find spelling errors . . . SPELL(1) 244
wc – word count . . . WC(1) 281
wwb, style, diction, punct – writer's workbench . . . WWB(1) 284

Writing Papers – Output

apsend – send troff output to phototypesetter . . . APSEND(1) 69
cb – C program beautifier . . . CB(1) 90
col, 2, 3, 4, 5, 6, mc, fold, expand – column alignment . . . COLUMN(1) 100
docsubmit – send document to library . . . DOCSUBMIT(1) 123
font – description files for troff . . . FONT(5) 492
font – typesetter fonts . . . FONT(6) 521
lp – printer output . . . LP(1) 170
proof – troff output interpreter for 5620 . . . PROOF(9) 661
psi – postscript interpreter . . . PSI(9) 662
reader – electronic retrieval of typeset documents . . . READER(9) 702
rev, revpag – reverse lines or pages . . . REV(1) 224
sendcover – send cover sheet to the library . . . SENDCOVER(8) 611

Numerical Mathematics

bc – arbitrary-precision arithmetic language . . . BC(1) 81
dc – desk calculator . . . DC(1) 113
factor, qfactor, primes – factor a number, generate large primes . . . FACTOR(1) 138
grap – pic preprocessor for drawing graphs . . . GRAP(1) 146
graph – draw a graph . . . GRAPH(1) 148
hoc – interactive floating point language . . . HOC(1) 151
matlab – interactive matrix desk calculator . . . MATLAB(1) 182
seq – print sequences of numbers . . . SEQ(1) 229

Graphics

2500 – BVH2500 videotape recorder . . . 2500(1) 60
bcp – reformat black-and-white picture files . . . BCP(1) 83
cip – draw pictures for typesetting . . . CIP(9) 641
dag – preprocessor for drawing directed graphs . . . DAG(1) 110
face, mugs – show faces, make face icons from pictures . . . FACE(9) 701
grap – pic preprocessor for drawing graphs . . . GRAP(1) 146
graph – draw a graph . . . GRAPH(1) 148
graphdraw graphpic – edit (combinatoric) graphs, convert to pic files . . . GRAPHDRAW(9) 644

icon – icon editor . . . ICON(9) 646
ideal – troff preprocessor for drawing pictures . . . IDEAL(1) 152
imscan – scan greyscale images . . . IMSCAN(1) 155
jf – font editor . . . JF(9) 647
map – digitized map formats . . . MAP(5) 494
movie, stills – algorithm animation . . . MOVIE(9) 650
ocr – optical character recognition . . . OCR(1) 193
ped, tped – picture editor . . . PED(9) 656
pic, tpic – troff and tex preprocessors for drawing pictures . . . PIC(1) 197
picfile – raster graphic image format . . . PICFILE(5) 497
pico – graphics editor . . . PICO(1) 200
plot – graphics filters . . . PLOT(1) 202
plot – graphics interface . . . PLOT(5) 499
qsnap – high resolution digital film printer . . . QSNAP(1) 213
rebecca – graphics touch-up editor . . . REBECCA(9) 663
view2d – movie of a function f(x, y, t) . . . VIEW2D(5) 512
view2d, regrid, vdata – movie of a function f(x, y, t) . . . VIEW2D(1) 279

Communication - Terminals

flicks, fflicks, preflicks, 2mux – movie graphics for 5620 . . . FLICKS(9) 642
getfont – replace terminal's default font . . . GETFONT(9) 643
jx – 5620 execution and stdio interpreter . . . JX(9) 648
lens – bitmap magnifier . . . LENS(9) 649
mouse – jerq mouse user interface . . . MOUSE(9) 693
movies – graphics movie file formats . . . FLICKFILE(9) 696
mux, ismux, invert – layer multiplexer for 5620 . . . MUX(9) 652
p, pg, more – paginate . . . P(1) 194
passwd – change login password . . . PASSWD(1) 196
ruler – measure things on the screen . . . RULER(9) 664
stty – set terminal options . . . STTY(1) 249
tabs – set terminal tabs . . . TABS(1) 252
term – nonstandard mux terminals . . . TERM(9) 671
terminals – conventional names . . . TERM(6) 532
tty, logtty – get terminal name . . . TTY(1) 268
ttys – terminal initialization data . . . TTYS(5) 509
ul – print underlines on screen terminals . . . UL(1) 269
windows, jps, reshape – create and initialize windows . . . WINDOWS(9) 674
write, mesg – write to other users, allow or forbid messages . . . WRITE(1) 283

Communication – Other Devices

2500 – BVH2500 videotape recorder . . . 2500(1) 60
apsend – send troff output to phototypesetter . . . APSEND(1) 69
dd, dblbuf – convert and copy a file . . . DD(1) 116
lp – printer output . . . LP(1) 170
null – data sink . . . NULL(4) 462
plot – graphics filters . . . PLOT(1) 202
tape, mt – identify and manipulate magnetic tape . . . TAPE(1) 254
tar – tape archiver . . . TAR(1) 255
thinkblt, think – print on thinkjet . . . THINKBLT(9) 672

Communication – Other Users

bundle – collect files for distribution . . . BUNDLE(1) 86
checknews, readnews – read netnews articles . . . READNEWS(7) 550
cpio – copy file archives in and out . . . CPIO(1) 104
cpio – format of cpio archive . . . CPIO(5) 486
mail – send or receive mail . . . MAIL(1) 176
mail – mail addresses . . . MAIL(6) 523
news – print news items . . . NEWS(7) 545
postnews – submit netnews articles . . . POSTNEWS(7) 548
vismon, sysmon, vwhois – system statistics and mail notification . . . VISMON(9) 673
who, whois, last – who is or was on the system . . . WHO(1) 282
write, mesg – write to other users, allow or forbid messages . . . WRITE(1) 283

Communication – Other Machines

con, rx – remote login and execution . . . CON(1) 102
cu, ct – call out to a terminal or another system . . . CU(1) 107
dcon, ndcon, rlogin, nrx, rsh, scriptcon – remote login and execution . . . DCON(1) 115
dkname – map system name to Datakit address . . . DKNAME(7) 540
mkpkg, inspkg, seal, unseal – package files for automatic software distribution . . . MKPKG(8) 586
push, pull, npush, npull – datakit remote file copy . . . PUSH(1) 211
rcp – remote file copy . . . RCP(1) 221
ship, shipstat – automatic software distribution . . . SHIP(8) 612
smstat – list smtp queues . . . SMSTAT(8) 615
spin – protocol analysis software . . . SPIN(1) 245
uucp, uulog, uuname – unix-to-unix remote file copy . . . UUCP(1) 271

uustat − uucp status inquiry and job control . . . . . UUSTAT(1) 273
uux − unix to unix command execution . . . . . UUX(1) 275

Process Management

getuid, id − get user identity . . . . . GETUID(1) 145
hang − start a process in stopped state . . . . . HANG(1) 150
kill − terminate a process with extreme prejudice . . . . . KILL(1) 157
newgrp − change to a new group . . . . . NEWGRP(1) 190
nice, nohup − run commands at low priority or immune to hangup . . . . . NICE(1) 191
ps − process status . . . . . PS(1) 209
rc, cd, wait, whatis − command language . . . . . RC(1) 215
sh, cd, wait, whatis − shell, the standard command programming language . . . . . SH(1) 230
sleep − suspend execution for an interval . . . . . SLEEP(1) 238
time − time a command . . . . . TIME(1) 263

Time Management

at − execute commands at a later time . . . . . AT(1) 73
cal − print calendar . . . . . CAL(7) 537
calendar − reminder service . . . . . CALENDAR(1) 88
date − print or set the date . . . . . DATE(1) 112
time − time a command . . . . . TIME(1) 263
wwv − print or set the date from accurate clock . . . . . WWV(1) 285

Instruction and Information

apnews, ap.keys − present AP wire stories . . . . . APNEWS(7) 535
ascii − interpret ASCII characters . . . . . ASCII(1) 72
ascii − map of ASCII character set . . . . . ASCII(6) 519
cal − print calendar . . . . . CAL(7) 537
date − print or set the date . . . . . DATE(1) 112
dict − look up words in English dictionaries . . . . . DICT(7) 538
dkname − map system name to Datakit address . . . . . DKNAME(7) 540
face, mugs − show faces, make face icons from pictures . . . . . FACE(9) 701
findauthor, papers, makepaper − consult database of locally authored papers . . . . . PAPERS(7) 546
games, demo − some playthings . . . . . GAMES(1) 143
library, bellcat − bell labs library services . . . . . LIBRARY(7) 541
man − macros to typeset manual . . . . . MAN(6) 524
man − print pages of this manual . . . . . MAN(1) 181
map − digitized map formats . . . . . MAP(5) 494
map − draw maps on various projections . . . . . MAP(7) 542
netnews − usenet news articles, utility files . . . . . NETNEWS(5) 495
picfile − raster graphic image format . . . . . PICFILE(5) 497
poly − polyhedron database . . . . . POLY(5) 502
polypic, polypr − database of polyhedra . . . . . POLY(7) 547
reader − electronic retrieval of typeset documents . . . . . READER(9) 702
scat − sky catalogue . . . . . SCAT(7) 552
sky − astronomical ephemeris . . . . . SKY(7) 553
tel, telno, dq − phone books . . . . . TEL(7) 554
town − gazetteer of US places . . . . . TOWN(7) 555
units − conversion program . . . . . UNITS(7) 556
weather − conditions and forecast by town . . . . . WEATHER(7) 557
whoami − computer name . . . . . WHOAMI(5) 513

Miscellany

crypt, encrypt, decrypt − encode/decode . . . . . CRYPT(1) 105
cscan − scan documents on canon scanner . . . . . CSCAN(1) 106
games, demo − some playthings . . . . . GAMES(1) 143
lab − label maker . . . . . LAB(1) 158

System Administration − Not just for su

backup − backup and recover files . . . . . BACKUP(1) 77
backup − incremental backup files . . . . . BACKUP(5) 483
du, df − disk usage . . . . . DU(1) 125
kill − terminate a process with extreme prejudice . . . . . KILL(1) 157
load − load statistics . . . . . LOAD(1) 167
passwd − change login password . . . . . PASSWD(1) 196
ps − process status . . . . . PS(1) 209
pstat − print system facts . . . . . PSTAT(8) 596
quot, findo − file system usage and hogs . . . . . QUOT(8) 599
upas, rmail, translate − mail delivery system . . . . . UPAS(8) 623
utmp, wtmp − login records . . . . . UTMP(5) 511
vismon, sysmon, vwhois − system statistics and mail notification . . . . . VISMON(9) 673
vmstat − report virtual memory statistics . . . . . VMSTAT(8) 627
who, whois, last − who is or was on the system . . . . . WHO(1) 282
wwv − print or set the date from accurate clock . . . . . WWV(1) 285

System Administration − su Only

ac − login accounting . . . . . AC(8) 562
acct − execution accounting file . . . . . ACCT(5) 481
arff − read RT11 files . . . . . ARFF(8) 563
backup − backup client administration . . . . . BACKUP(8) 565
chown − change owner or group . . . . . CHOWN(8) 566
chuck − a file system checking program . . . . . CHUCK(8) 567
clri − clear inode . . . . . CLRI(8) 568
console − VAX console interface . . . . . CONSOLE(4) 450
date − print or set the date . . . . . DATE(1) 112
dmesg − system diagnostic messages . . . . . DMESG(8) 572
fsck − file system consistency check and interactive repair . . . . . FSCK(8) 573
icheck, dcheck, ncheck − file system consistency check . . . . . ICHECK(8) 576
kmc, kdiload, kmcdump − control KMC11 input/output processors . . . . . KMC(8) 580
makekey − generate encryption key . . . . . MAKEKEY(8) 583
mkfs, mkbitfs, mklost+found − construct a disk file system . . . . . MKFS(8) 584
mknod − construct special file . . . . . MKNOD(8) 585
mount, umount − mount and dismount file system . . . . . MOUNT(8) 587
netfs − network file system . . . . . NETFS(8) 588
netstat, dkstat − show network status for internet and datakit networks . . . . . NETSTAT(8) 590
passwd, group − password and group files . . . . . PASSWD(5) 496
rarepl, rarct − replace bad blocks on MSCP disks . . . . . RAREPL(8) 600
renice − change priority of running process . . . . . RENICE(8) 604
sa, accton − system accounting . . . . . SA(8) 606
scsish − SCSI shell . . . . . SCSISH(8) 609
ship, shipstat – automatic software distribution . . . . . SHIP(8) 612
showq − status of stream input/output system . . . . . SHOWQ(8) 613
smash − rewrite bad disk sectors . . . . . SMASH(8) 614
su, setlog − substitute userid temporarily, become super-user . . . . . SU(8) 617
sync, update − update disk file systems . . . . . SYNC(8) 621
wall − write to all users . . . . . WALL(8) 628
worm, jukebox − optical disk utilities . . . . . WORM(8) 629

System Administration - Usually or Always Run Automatically

asd − automatic software distribution . . . . . ASD(8) 564
cron − clock daemon . . . . . CRON(8) 570
dkhup, dkmgr, dkzap − manage Datakit interface . . . . . DKMGR(8) 571
dpost, postdaisy, postdmd, postprint − filters to produce postscript . . . . . POSTSCRIPT(8) 594
faced − network face server . . . . . FACED(9) 695
getty − set terminal mode . . . . . GETTY(8) 575
init − process control initialization . . . . . INIT(8) 578
ipconfig, dkipconfig, udpconfig − set up DARPA Internet protocols . . . . . IPCONFIG(8) 579
ldpcs − load comet microcode . . . . . LDPCS(8) 581
login − sign on . . . . . LOGIN(8) 582
ns − name server database . . . . . NS(8) 591
postio − serial interface for postscript printers . . . . . POSTIO(8) 592
qns − query name server . . . . . QNS(7) 549
rc − boot script . . . . . RC(8) 601
reboot − bootstrapping procedures . . . . . REBOOT(8) 602
route, routed − IP gateway routing . . . . . ROUTE(8) 605
savecore − save a core image of the operating system . . . . . SAVECORE(8) 608
smtp, smtpqer, smtpd, smtpsched − handle simple mail transfer protocol . . . . . SMTP(8) 616
svcmgr − service remote computing requests . . . . . SVCMGR(8) 618
swapon − specify swapping device . . . . . SWAPON(8) 620
tcpmgr − accept and place calls via the TCP protocol . . . . . TCPMGR(8) 622
upas, rmail, translate − mail delivery system . . . . . UPAS(8) 623
uucico, uusched, uuxqt, kick, debug − uucp file transport and execution . . . . . UUCICO(8) 625
uucleanup − uucp spool directory clean-up . . . . . UUCLEANUP(8) 626

Functions, Subroutines and Data Structures

Directory Functions

access – determine accessibility of file . . . . . ACCESS(2) 295
chdir, chroot – change working or root directory . . . . . CHDIR(2) 299
chmod, fchmod, chown, fchown, utime – change file mode, owner, group, or times . . . . . CHMOD(2) 300
dirread – read from directory, hiding format . . . . . DIRREAD(2) 302
ftw − file tree walk . . . . . FTW(3) 370
getwd, getcwd – get current directory . . . . . GETWD(3) 384
mkdir, rmdir – make or remove a directory . . . . . MKDIR(2) 313
opendir, readdir, telldir, seekdir, closedir – directory operations . . . . . DIRECTORY(3) 355

File Inquiry/Create Functions

access – determine accessibility of file . . . . . ACCESS(2) 295
chmod, fchmod, chown, fchown, utime – change file mode, owner, group, or times . . . . . CHMOD(2) 300

feof, ferror, clearerr, fileno – stream status inquiries . . . FERROR(3) 361
ioctl – miscellaneous control operations . . . IOCTL(2) 310
link, symlink, readlink – link to a file . . . LINK(2) 311
mktemp, tmpnam – make a unique file name . . . MKTEMP(3) 396
stat, lstat, fstat – get file status . . . STAT(2) 323
umask – set file creation mode mask . . . UMASK(2) 330
unlink – remove directory entry . . . UNLINK(2) 331

File Open/Read/Write Functions

Finit, Frdline, Fgetc, Fread, Fseek, Fundo,Fputc, Fprint, Fwrite, Fflush, Ftie, Fclose, Fexit – fast buffered input/output . . . FIO(3) 363
dirread – read from directory, hiding format . . . DIRREAD(2) 302
dup, dup2 – duplicate an open file descriptor . . . DUP(2) 303
fd, stdin, stdout, stderr, tty – file descriptor files . . . FD(4) 455
feof, ferror, clearerr, fileno – stream status inquiries . . . FERROR(3) 361
fgets, puts, fputs, gets – string input/out on streams . . . FGETS(3) 362
fopen, freopen, fdopen, fclose, fflush – open, close, or flush a stream . . . FOPEN(3) 366
fread, fwrite – buffered binary input/output . . . FREAD(3) 367
fseek, ftell, rewind – reposition a stream . . . FSEEK(3) 369
getc, getchar, fgetc, getw, putc, putchar, fputc, putw – character- or word-at-a-time stream input/output . . . GETC(3) 374
ioctl – miscellaneous control operations . . . IOCTL(2) 310
lseek – seek, move read/write pointer . . . LSEEK(2) 312
open, creat, close – open a file for reading or writing, create file . . . OPEN(2) 316
print, fprint, sprint, fmtinstall – print formatted output . . . PRINT(3) 409
printf, fprintf, sprintf, snprintf – print formatted output . . . PRINTF(3) 411
read, write – read or write file . . . READ(2) 319
scanf, fscanf, sscanf – formatted input . . . SCANF(3) 421
select – synchronous input/output multiplexing . . . SELECT(2) 320
setbuf – assign buffering to a stream . . . SETBUF(3) 423
stdio – standard buffered input/output package . . . STDIO(3) 428
thinkchar, thinknchars, thinkflush, thinkmap, thinkabort − ThinkJet routines . . . THINKCLIENT(9) 691
ungetc – push character back into input stream . . . UNGETC(3) 441

Privacy and Security

access – determine accessibility of file . . . ACCESS(2) 295
chmod, fchmod, chown, fchown, utime – change file mode, owner, group, or times . . . CHMOD(2) 300
getgrent, getgrgid, getgrnam, setgrent, endgrent – get group file entry . . . GETGRENT(3) 379
getlogin – get login name . . . GETLOGIN(3) 380
getpass – read a password . . . GETPASS(3) 382
getpwent, getpwuid, getpwnam, setpwent, endpwent, pwdecode – get password file entry . . . GETPWENT(3) 383
getuid, getgid, geteuid, getegid, getlogname, getpid, getppid, getpgrp, setuid, setgid, setruid, setlogname, setpgrp – get or set user, group, or process identity . . . GETUID(2) 309

Manipulating Processes − Time Management

_exit, wait wait3 – terminate process, wait for child to terminate . . . EXIT(2) 306
alarm, nap, pause – schedule timing delays . . . ALARM(2) 297
ctime, localtime, gmtime, asctime, timezone – convert date and time to ASCII . . . CTIME(3) 349
nice – set program priority . . . NICE(2) 315
request, own, wait, alarm, sleep, nap, kbdchar, rcvchar, realtime, sendchar, sendnchars, kill, exit − 5620 input/output requests . . . REQUEST(9) 679
sleep – suspend execution for an interval . . . SLEEP(3) 427
time, ftime – get date and time . . . TIME(2) 328
timec, timegm, timelocal − convert ASCII to time . . . TIMEC(3) 436
times – get process times . . . TIMES(2) 329
vtimes − get usage of time, space, and paging resources . . . VTIMES(2) 332

Manipulating Processes − Memory Management

alloc, free, balloc, bfree, gcalloc, gcfree − allocate memory . . . ALLOC(9) 682
brk, sbrk – change core allocation . . . BRK(2) 298
end, etext, edata – last locations in program . . . END(3) 357
galloc, gfree, garbage – storage allocation with garbage collection . . . GALLOC(3) 371
malloc, free, realloc, calloc, cfree − memory allocator . . . MALLOC(3) 392
memccpy, memchr, memcmp, memcpy, memmove, memset – memory operations . . . MEMORY(3) 395

Manipulating Processes − Process Management

P, newproc, muxnewwind, newwindow, tolayer, debug, getproc, getproctab, putname, getname − jerq process control . . . NEWPROC(9) 677
_exit, wait wait3 – terminate process, wait for child to terminate . . . EXIT(2) 306
abort − generate a fault . . . ABORT(3) 336
environ − user environment . . . ENVIRON(5) 488
execl, execv, execle, execve, execlp, execvp, exect – execute a file . . . EXEC(2) 304
exit, onexit – terminate process . . . EXIT(3) 359
fork – spawn new process . . . FORK(2) 308
getuid, getgid, geteuid, getegid, getlogname, getpid, getppid, getpgrp, setuid, setgid, setruid, . . . GETUID(2) 309

setlogname, setpgrp – get or set user, group, or process identity
perror, sys_errlist, sys_nerr – system error messages . . . . . PERROR(3) 400
proc – process file system . . . . . PROC(4) 463
request, own, wait, alarm, sleep, nap, kbdchar, rcvchar, realtime, sendchar, sendnchars, kill, exit – 5620 input/output requests . . . . . REQUEST(9) 679
setjmp, longjmp – non-local goto . . . . . SETJMP(3) 424
signal, kill – receive and send signals . . . . . SIGNAL(2) 321
system – issue a shell command . . . . . SYSTEM(3) 432

Manipulating Processes – Tuning and Debugging

assert – assertion checking . . . . . ASSERT(3) 338
monitor – prepare execution profile . . . . . MONITOR(3) 397
nlist – get entries from name list . . . . . NLIST(3) 399
perror, sys_errlist, sys_nerr – system error messages . . . . . PERROR(3) 400
profil – execution time profile . . . . . PROFIL(2) 318

Data Manipulation/Database Management

atof, atoi, atol, strtod, strtol, strtoul – convert ASCII to numbers . . . . . ATOF(3) 339
bopen, bclose, bseek, bfirst, bkey, breclen, bread,bdelete, bwrite – compressed B-tree subroutines . . . . . CBT(3) 345
crypt, setkey, encrypt – DES encryption . . . . . CRYPT(3) 348
ctime, localtime, gmtime, asctime, timezone – convert date and time to ASCII . . . . . CTIME(3) 349
dbminit, fetch, store, delete, firstkey, nextkey – database subroutines . . . . . DBM(3) 353
ecvt, fcvt, gcvt – convert numbers to ascii . . . . . ECVT(3) 356
frexp, ldexp, modf – split into mantissa and exponent . . . . . FREXP(3) 368
getarg, iargc – command arguments to Fortran . . . . . GETARG(3) 373
getenv – value for environment name . . . . . GETENV(3) 375
getfields, getmfields, setfields – break a string into fields . . . . . GETFIELDS(3) 376
getflags – process flag arguments in argv . . . . . GETFLAGS(3) 377
getopt – get option letter from argv . . . . . GETOPT(3) 381
huff – huffman codebook/tree generator . . . . . HUFF(3) 385
isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii – character classification . . . . . CTYPE(3) 350
l3tol, ltol3 – convert between 3-byte integers and long integers . . . . . L3TOL(3) 391
qsort – quicker sort . . . . . QSORT(3) 416
re_bm, re_cw, re_re – string and pattern matching . . . . . RE(3) 418
regcomp, regexec, regsub, regerror – regular expression . . . . . REGEXP(3) 420
scanf, fscanf, sscanf – formatted input . . . . . SCANF(3) 421
strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen,strchr, strrchr, strpbrk, strspn, strcspn, strtok, strdup – string operations . . . . . STRING(3) 429
swab – swap bytes . . . . . SWAB(3) 431
timec, timegm, timelocal – convert ASCII to time . . . . . TIMEC(3) 436
tolower, toupper – force upper or lower case . . . . . TOLOWER(3) 437
types – primitive system data types . . . . . TYPES(5) 510
varargs – variable argument list . . . . . VARARGS(3) 442

C++ Data Types

Map – associative array classes . . . . . MAP(3) 393
bits – variable length bit strings . . . . . BITS(3) 341
block – adjustable arrays . . . . . BLOCK(3) 343
pool – fast memory allocation . . . . . POOL(3) 406

Mathematics

abs, sgn, gcd, lcm, min, max, labs – integer arithmetic functions:absolute value, sign, greatest common divisor, least common multiple, minimum, maximum . . . . . ARITH(3) 337
add, sub, mul, div, eqpt, eqrect, inset, muldiv, ptinrect, raddp, rsubp, rectXrect, rectclip – arithmetic on points and rectangles . . . . . ADD(9) 681
atof, atoi, atol, strtod, strtol, strtoul – convert ASCII to numbers . . . . . ATOF(3) 339
besj0, besj1, besjn, besy0, besy1, besyn – bessel functions . . . . . BESSEL(3) 340
cos, sin, atan2, sqrt, norm – integer math functions . . . . . COS(9) 685
erf, erfc – error function . . . . . ERF(3) 358
exp, log, log10, pow, sqrt – exponential, logarithm, power, square root . . . . . EXP(3) 360
fabs, fmod, floor, ceil – absolute value, remainder, floor, ceiling functions . . . . . FLOOR(3) 365
frexp, ldexp, modf – split into mantissa and exponent . . . . . FREXP(3) 368
gamma – log gamma function . . . . . GAMMA(3) 372
hypot, cabs – euclidean distance . . . . . HYPOT(3) 386
itom, mfree, madd, msub, mult, mdiv, sdiv, msqrt, mgcd, min, mout,fmin, fmout, move, mcmp, rpow, mpow – multiple precision integer arithmetic . . . . . MP(3) 398
orient, normalize – map projections . . . . . PROJ(3) 413
poly_lk, poly_read – polyhedron database routines . . . . . POLY(3) 405
port – mathematical library for Fortran . . . . . PORT(3) 408
rand, lrand, frand, nrand, srand – random number generator . . . . . RAND(3) 417
sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions . . . . . SIN(3) 425
sinh, cosh, tanh – hyperbolic functions . . . . . SINH(3) 426

Graphics

Code, addr, bitblt, point, rectf, screenswap, segment, texture − graphics functions . . . BITBLT(9) 683
Word, Point, Rectangle, Bitmap, Texture, Pt, Rect, Rpt, display, Drect, Jrect − graphics data types . . . TYPES(9) 699
chrtab – simple character bitmaps . . . CHRTAB(3) 347
circle, disc, arc, ellipse, eldisc, elarc − circle-drawing functions . . . CIRCLE(9) 684
orient, normalize − map projections . . . PROJ(3) 413
poly_lk, poly_read – polyhedron database routines . . . POLY(3) 405
view2d, moviefil – movie of a function f(x, y, t) . . . VIEW2D(3) 443

Communication − Terminals

button123, mouse, cursallow, cursinhibit, cursset, cursswitch, getrect123 − mouse control . . . BUTTON(9) 675
curses – screen functions with ‘optimal’ cursor motion . . . CURSES(3) 351
font − jerq font layouts . . . FONT(9) 697
jioctl − mux ioctl requests . . . JIOCTL(9) 692
menuhit, hmenuhit − present user with menu and get selection . . . MENUHIT(9) 687
newlayer, dellayer, downback, lbitblt, lpoint, lrectf, lsegment, ltexture, upfront − layer control and graphics . . . NEWLAYER(9) 676
pads − user interface package . . . PADS(9) 698
request, own, wait, alarm, sleep, nap, kbdchar, rcvchar, realtime, sendchar, sendnchars, kill, exit − 5620 input/output requests . . . REQUEST(9) 679
string, defont, strwidth, infont, outfont, getfont − text and font operations . . . STRING(9) 690
strinsure, strinsert, strdelete, strzero, setmuxbuf, getmuxbuf, movstring − dynamic strings in mux . . . MUXSTRING(9) 689
termcap − terminal capability file . . . TERMCAP(5) 505
tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs – device-independent terminal screen control . . . TERMCAP(3) 435
tty − serial line interface drivers . . . TTY(4) 472
tty_ld − terminal processing line discipline . . . TTYLD(4) 473
ttyname, isatty, nametty – find or set name of a terminal . . . TTYNAME(3) 438
vec, move, etc. – plot graphics interface . . . PLOT(3) 403

Communication − Between Processes

conn_ld − line discipline for unique stream connection . . . CONNLD(4) 449
fmount, funmount − mount or remove file system . . . FMOUNT(2) 307
ipccreat, ipcopen, ipclisten, ipcaccept, ipcreject, ipcexec, ipcpath , ipclogin, ipcrogin – set up connections between processes or machines . . . IPC(3) 388
pipe – create an interprocess channel . . . PIPE(2) 317
popen, ppopen, vepopen, pclose – open a pipe to/from a process . . . POPEN(3) 407
read, write – read or write file . . . READ(2) 319
select – synchronous input/output multiplexing . . . SELECT(2) 320
stream − communication channels . . . STREAM(4) 468

Communication − Outside World

buf_ld – buffering line discipline . . . BUFLD(4) 448
dialout – place call on ACU . . . DIALOUT(3) 354
dk, dkp_ld, unixp_ld, cmc_ld − Datakit interface and protocols . . . DK(4) 451
ethernet – Ethernet interface . . . ETHERNET(4) 454
in_host, in_ntoa, in_address, in_service − internet networking functions . . . INTERNET(3) 387
ip, ip_ld − DARPA internet protocol . . . IP(4) 456
mesg_ld, rmesg_ld − message line discipline modules . . . MESGLD(4) 459
mt − magtape interface . . . MT(4) 460
tcp_sock, tcp_connect, tcp_listen, tcp_accept, tcp_rcmd – tcp networking functions . . . TCP(3) 433
udp_connect, udp_listen, udp_datagram – udp networking functions . . . UDP(3) 439

System administration/Only for the Knowledgeable

acct – turn accounting on or off . . . ACCT(2) 296
conn_ld − line discipline for unique stream connection . . . CONNLD(4) 449
drum − paging device . . . DRUM(4) 453
filsys, flblk, ino − format of a disk file system . . . FILSYS(5) 489
fmount, funmount − mount or remove file system . . . FMOUNT(2) 307
fstab, mtab − information about file systems . . . FSTAB(5) 493
getfsent, getfsspec, getfsfile, setfsent, endfsent – get file system description file entry . . . GETFSENT(3) 378
mem, kmem, kUmem, mtpr, fineclock − memory and VAX processor registers . . . MEM(4) 458
mknod – make a directory or a special file . . . MKNOD(2) 314
ra − DEC MSCP disks (RA60, RA80, RA81, RA90) . . . RA(4) 465
scsi − SCSI interface . . . SCSI(4) 467
stime, biasclock – set time . . . STIME(2) 325
sync − force writing of system buffers . . . SYNC(2) 326
types − primitive system data types . . . TYPES(5) 510
uname − identify machine and kernel . . . UNAME(3) 440

Miscellany

reboot, vadvise, vlimit, vswapon, getgroups,setgroups – system calls to be avoided . . . DEPRECATED(2) 301
syscall – indirect system call . . . SYSCALL(2) 327

PERMUTED INDEX

Section (A) is an unprinted appendix. Manual pages for all sections, including the appendix, are accessible on line through *man*(1).

To save space, neighboring references to the same page have been collapsed into a single reference. This should cause no difficulty in cases like ‘atan’ and ‘atan2’, but is somewhat obscure in the case of ‘strcat’ and ‘strchr’.

	=, ==, =p, ==p − redo previous shell command	=(1)	61
11size, 11strip − pdp11 support	11as, 11cc, 11ld, 11nm, 11ranlib, 11reloc,	11(8)	561
alignment . . . col,	2, 3, 4, 5, 6, mc, fold, expand − column	COLUMN(1)	100
	2500 − BVH2500 videotape recorder	2500(1)	60
flicks, fflicks, preflicks,	2mux − movie graphics for 5620	FLICKS(9.1)	642
alignment . . . col, 2,	3, 4, 5, 6, mc, fold, expand − column	COLUMN(1)	100
300 and 300s terminals	300, 300s − handle special functions of DASI	300(A)	
	32ld − bootstrap loader for 5620	32LD(9.1)	636
l3tol, ltol3 − convert between	3-byte integers and long integers	L3TOL(3)	391
pi,	3pi − process inspector	PI(9.1)	658
3cc, 3as, 3ar, 3ld, 3nm,	3size, 3strip, cprs − MAC-32 C compiler	3CC(9.1)	637
col, 2, 3,	4, 5, 6, mc, fold, expand − column alignment	COLUMN(1)	100
tk − paginator for the Tektronix	4014	TK(A)	
terminal . . .	450 − handle special functions of the DASI 450	450(A)	
col, 2, 3, 4,	5, 6, mc, fold, expand − column alignment	COLUMN(1)	100
32ld − bootstrap loader for	5620	32LD(9.1)	636
mux, ismux, invert − layer multiplexer for	5620	MUX(9.1)	652
proof − troff output interpreter for	5620	PROOF(9.1)	661
jx −	5620 execution and stdio interpreter	JX(9.1)	648
realtime, sendchar, sendnchars, kill, exit −	5620 input/output requests . . . /kbdchar, rcvchar,	REQUEST(9.2)	679
intro − introduction to	5620-related software	INTRO(9)	635
col, 2, 3, 4, 5,	6, mc, fold, expand − column alignment	COLUMN(1)	100
	abort − generate a fault	ABORT(3)	336
arithmetic functions: absolute value, sign,/	abs, sgn, gcd, lcm, min, max, labs − integer	ARITH(3)	337
functions . . . fabs, fmod, floor, ceil −	absolute value, remainder, floor, ceiling	FLOOR(3)	365
	ac − login accounting	AC(8)	562
touch, chdate − set modification or	access date of a file	CHDATE(1)	95
	access − determine accessibility of file	ACCESS(2)	295
ac − login	accounting	AC(8)	562
	acct − execution accounting file	ACCT(5)	481
	acct − turn accounting on or off	ACCT(2)	296
sa,	accton − system accounting	SA(8)	606
sin, cos, tan, asin,	acos, atan, atan2 − trigonometric functions	SIN(3)	425
dialout − place call on	ACU	DIALOUT(3)	354
	adb − debugger	ADB(1)	62
muldiv, ptinrect, raddp, rsubp, rectXrect,/	add, sub, mul, div, eqpt, eqrect, inset,	ADD(9.3)	681
segment, texture − graphics functions . . . . . Code,	addr, bitblt, point, rectf, screenswap,	BITBLT(9.3)	683
dkname − map system name to Datakit	address	DKNAME(7)	540
mail − mail	addresses	MAIL(6)	523
block −	adjustable arrays	BLOCK(3+)	343
backup − backup client	administration	BACKUP(8)	565
dungeon-exploration games	adventure, zork, rogue, wump −	ADVENTURE(A)	
atc −	air traffic controller	ATC(A)	
	alarm, nap, pause − schedule timing delays	ALARM(2)	297
sendchar, sendnchars,/ . . . . . request, own, wait,	alarm, sleep, nap, kbdchar, rcvchar, realtime,	REQUEST(9.2)	679
altran − language for	algebraic manipulation	ALTRAN(A)	
maple − interactive symbolic	algebraic program	MAPLE(A)	
movie, stills −	algorithm animation	MOVIE(9.1)	650
allocate memory	alloc, free, balloc, bfree, gcalloc, gcfree −	ALLOC(9.3)	682
brk, sbrk − change core	allocation	BRK(2)	298
pool − fast memory	allocation	POOL(3+)	406
galloc, gfree, garbage − storage	allocation with garbage collection	GALLOC(3)	371
malloc, free, realloc, calloc, cfree − memory	allocator	MALLOC(3)	392
	altran − language for algebraic manipulation	ALTRAN(A)	

maple, ops5, pascal, ratfor, S, smp, sno,/ . . . . . altran, cospan, esterel, icon, lisp, macsyma, ... LANGS(1) 159
movie, stills − algorithm animation . . . . . MOVIE(9.1) 650
bcd, ppt, morse − convert to antique media . . . . . BCD(A)
a.out − object file format . . . . . A.OUT(5) 479
apnews, ap.keys − present AP wire stories . . . . . APNEWS(7) 535
select arguments . . . . . apply, pick − repeatedly apply a command; . . . . . APPLY(1) 68
apsend − send troff output to phototypesetter . . . . . APSEND(1) 69
ar − archive (library) file format . . . . . AR(5) 482
ar, ranlib − archive and library maintainer . . . . . AR(1) 70
functions . . . . . circle, disc, arc, ellipse, eldisc, elarc − circle-drawing . . . . . CIRCLE(9.3) 684
cpio − format of cpio archive . . . . . CPIO(5) 486
ar, ranlib − archive and library maintainer . . . . . AR(1) 70
ar − archive (library) file format . . . . . AR(5) 482
tar − tape archiver . . . . . TAR(1) 255
cpio − copy file archives in and out . . . . . CPIO(1) 104
arff − read RT11 files . . . . . ARFF(8) 563
varargs − variable argument list . . . . . VARARGS(3) 442
xargs − construct argument lists and execute command . . . . . XARGS(A)
pick − repeatedly apply a command; select arguments . . . . . apply, APPLY(1) 68
echo, printf − print arguments . . . . . ECHO(1) 126
expr − evaluate arguments as an expression . . . . . EXPR(1) 134
getflags − process flag arguments in argv . . . . . GETFLAGS(3) 377
getarg, iargc − command arguments to Fortran . . . . . GETARG(3) 373
rpow, mpow − multiple precision integer arithmetic . . . . . /min, mout, fmin, fmout, move, mcmp, MP(3) 398
arithmetic − drill in number facts . . . . . ARITHMETIC(A)
abs, sgn, gcd, lcm, min, max, labs − integer arithmetic functions: absolute value, sign,/ . . . . . ARITH(3) 337
bc − arbitrary-precision arithmetic language . . . . . BC(1) 81
ptinrect, raddp, rsubp, rectXrect, rectclip − arithmetic on points and rectangles . . . . . /muldiv, ADD(9.3) 681
Map − associative array classes . . . . . MAP(3+) 393
block − adjustable arrays . . . . . BLOCK(3+) 343
strstreambuf − streambuf specialized to arrays . . . . . SSBUF(A)
strstream − iostream specialized to arrays . . . . . STRSTREAM(A)
as − assembler . . . . . AS(1) 71
asa − interpret ASA control characters . . . . . ASA(A)
asctime, timezone − convert date and time to ASCII . . . . . ctime, localtime, gmtime, CTIME(3) 349
ecvt, fcvt, gcvt − convert numbers to ascii . . . . . ECVT(3) 356
ascii − interpret ASCII characters . . . . . ASCII(1) 72
ascii − map of ASCII character set . . . . . ASCII(6) 519
xd, od − hex, octal, decimal, or ASCII dump . . . . . XD(1) 286
atol, strtod, strtol, strtoul − convert ASCII to numbers . . . . . atof, atoi, ATOF(3) 339
timec, timegm, timelocal − convert ASCII to time . . . . . TIMEC(3) 436
ASCII . . . . . ctime, localtime, gmtime, asctime, timezone − convert date and time to . . . . . CTIME(3) 349
asd − automatic software distribution . . . . . ASD(8) 564
functions . . . . . sin, cos, tan, asin, acos, atan, atan2 − trigonometric . . . . . SIN(3) 425
as − assembler . . . . . AS(1) 71
assert − assertion checking . . . . . ASSERT(3) 338
Map − associative array classes . . . . . MAP(3+) 393
sky − astronomical ephemeris . . . . . SKY(7) 553
at − execute commands at a later time . . . . . AT(1) 73
sin, cos, tan, asin, acos, atan, atan2 − trigonometric functions . . . . . SIN(3) 425
cos, sin, atan2, sqrt, norm − integer math functions . . . . . COS(9.3) 685
atc − air traffic controller . . . . . ATC(A)
convert ASCII to numbers . . . . . atof, atoi, atol, strtod, strtol, strtoul − . . . . . ATOF(3) 339
makepaper − consult database of locally authored papers . . . . . findauthor, papers, PAPERS(7) 546
asd − automatic software distribution . . . . . ASD(8) 564
inspkg, seal, unseal − package files for automatic software distribution . . . . . mkpkg, MKPKG(8) 586
ship, shipstat − automatic software distribution . . . . . SHIP(8) 612
dist, dme, plan, path, cross − aviation navigation . . . . . AV(A)
w, fp, ft, fd, rad − aviation weather . . . . . AVW(A)
language . . . . . awk − pattern-directed scanning and processing . . . . . AWK(1) 74
back − backgammon . . . . . BACK(A)
backup − backup and recover files . . . . . BACKUP(1) 77
backup − backup client administration . . . . . BACKUP(8) 565
backup − incremental backup files . . . . . BACKUP(5) 483
rarepl, rarct − replace bad blocks on MSCP disks . . . . . RAREPL(8) 600
smash − rewrite bad disk sectors . . . . . SMASH(8) 614
memory . . . . . alloc, free, balloc, bfree, gcalloc, gcfree − allocate . . . . . ALLOC(9.3) 682
banner, rot, rnd, bigp − print in large type . . . . . BANNER(A)
basename, dirname − strip filename affixes . . . . . BASENAME(1) 80
postbgi − PostScript translator for BGI (Basic Graphical Instructions) files . . . . . POSTBGI(A)
bc − arbitrary-precision arithmetic language . . . . . BC(1) 81
bcd, ppt, morse − convert to antique media . . . . . BCD(A)
bcp − reformat black-and-white picture files . . . . . BCP(1) 83
bclose, bseek, bfirst, bkey, breclen, bread, bdelete, bwrite − compressed B-tree/ . . . . . bopen, CBT(3) 345

cb − C program beautifier . . . . . CB(1) 90
library, bellcat − bell labs library services . . . . . LIBRARY(7) 541
bessel functions . . . . . besj0, besj1, besjn, besy0, besy1, besyn − . . . . . BESSEL(3) 340
− compressed B-tree/ . . . . . bopen, bclose, bseek, bfirst, bkey, breclen, bread, bdelete, bwrite . . . . . CBT(3) 345
alloc, free, balloc, bfree, gcalloc, gcfree − allocate memory . . . . . ALLOC(9.3) 682
postbgi − PostScript translator for BGI (Basic Graphical Instructions) files . . . . . POSTBGI(A)
bianchi − espresso, steamed milk, hot water . . . . . BIANCHI(A)
stime, biasclock − set time . . . . . STIME(2) 325
penter, plook, pconvert − maintain and use bibliographic references . . . . . prefer, pinvert, PREFER(1) 204
refer, lookbib, pubindex − maintain and use bibliographic references . . . . . REFER(1) 222
latex, bibtex − tex macro package and bibliographies . . . . . LATEX(6) 522
bigcore, coreid − permit big core images, identify source of image . . . . . BIGCORE(1) 85
banner, rot, rnd, bigp − print in large type . . . . . BANNER(A)
fread, fwrite − buffered binary input/output . . . . . FREAD(3) 367
iclc − Esterel binder . . . . . ICLC(A)
replacement) . . . . . bison − GNU Project parser generator (yacc . . . . . BISON(A)
bits − variable length bit strings . . . . . BITS(3+) 341
texture − graphics functions . . . . . Code, addr, bitblt, point, rectf, screenswap, segment, ... BITBLT(9.3) 683
bitfile − format of bitmap file . . . . . BITFILE(9.5) 694
lens − bitmap magnifier . . . . . LENS(9.1) 649
Jrect − graphics data/ . . . . Word, Point, Rectangle, Bitmap, Texture, Pt, Rect, Rpt, display, Drect, ... TYPES(9.5) 699
chrtab − simple character bitmaps . . . . . CHRTAB(3) 347
mbits − macros to typeset bitmaps . . . . . MBITS(6) 526
compressed/ . . . . . bopen, bclose, bseek, bfirst, bkey, breclen, bread, bdelete, bwrite − . . . . . CBT(3) 345
image . . . . . blitblt, menudrop − save or print a screen . . . . . BLITBLT(9.1) 638
block − adjustable arrays . . . . . BLOCK(3+) 343
rc − boot script . . . . . RC(8) 601
32ld − bootstrap loader for 5620 . . . . . 32LD(9.1) 636
reboot − bootstrapping procedures . . . . . REBOOT(8) 602
getfields, getmfields, setfields − break a string into fields . . . . . GETFIELDS(3) 376
B-tree/ . . . . . bopen, bclose, bseek, bfirst, bkey, breclen, bread, bdelete, bwrite − compressed . . . . . CBT(3) 345
bridge − card game . . . . . BRIDGE(A)
brk, sbrk − change core allocation . . . . . BRK(2) 298
samuel − text editor and C browser . . . . . SAMUEL(A)
brush − painting program . . . . . BRUSH(9.1) 639
bwrite − compressed B-tree/ . . . . . bopen, bclose, bseek, bfirst, bkey, breclen, bread, bdelete, ... CBT(3) 345
cbt − btree utilities . . . . . CBT(1) 91
pipestream − iostream specialized as circular buffer . . . . . PIPESTREAM(A)
filebuf − buffer for file input/output . . . . . FILEBUF(A)
fread, fwrite − buffered binary input/output . . . . . FREAD(3) 367
Fwrite, Fflush, Ftie, Fclose, Fexit − fast buffered input/output . . . . . /Fundo, Fputc, Fprint, FIO(3) 363
stdio − standard buffered input/output package . . . . . STDIO(3) 428
streambuf − public interface of character buffering class . . . . . SBUF.PUB(A)
iostream − buffering, formatting and input/output . . . . . IOS.INTRO(A)
buf_ld − buffering line discipline . . . . . BUFLD(4) 448
setbuf − assign buffering to a stream . . . . . SETBUF(3) 423
sync − force writing of system buffers . . . . . SYNC(2) 326
buf_ld − buffering line discipline . . . . . BUFLD(4) 448
bundle − collect files for distribution . . . . . BUNDLE(1) 86
cursset, cursswitch, getrect123 − mouse/ . . . . . button123, mouse, cursallow, cursinhibit, ... BUTTON(9.2) 675
2500 − BVH2500 videotape recorder . . . . . 2500(1) 60
/bseek, bfirst, bkey, breclen, bread, bdelete, bwrite − compressed B-tree subroutines . . . . . CBT(3) 345
residency . . . . . byteyears − time-space product for file . . . . . BYTEYEARS(A)
f2c − Convert Fortran 77 to C or C++ . . . . . F2C(1) 135
samuel − text editor and C browser . . . . . SAMUEL(A)
3ar, 3ld, 3nm, 3size, 3strip, cprs − MAC-32 C compiler . . . . . 3cc, 3as, 3CC(9.1) 637
CC, cfront − C++ compiler . . . . . C++(1) 87
gcc − GNU project C Compiler . . . . . GCC(A)
mcc − MC68000 C compiler . . . . . MCC(A)
cc, lcc − C compilers . . . . . CC(1) 92
cin − C interpreter . . . . . CIN(1) 97
cpp − C language preprocessor . . . . . CPP(8) 569
f2c − Convert Fortran 77 to C or C++ . . . . . F2C(1) 135
cb − C program beautifier . . . . . CB(1) 90
pp − C program pretty printer . . . . . PP(A)
lint, cyntax, cem − C program verifiers . . . . . LINT(1) 165
hypot, cabs − euclidean distance . . . . . HYPOT(3) 386
cal − print calendar . . . . . CAL(7) 537
dc − desk calculator . . . . . DC(1) 113
matlab − interactive matrix desk calculator . . . . . MATLAB(1) 182
cal − print calendar . . . . . CAL(7) 537
calendar − reminder service . . . . . CALENDAR(1) 88
dialout − place call on ACU . . . . . DIALOUT(3) 354
cu, ct − call out to a terminal or another system . . . . . CU(1) 107

malloc, free, realloc, calloc, cfree − memory allocator . . . . . MALLOC(3) 392
tcpmgr − accept and place calls via the TCP protocol . . . . . TCPMGR(8) 622
canfield, fish − card games . . . . . CARDS(A)
cscan − scan documents on canon scanner . . . . . CSCAN(1) 106
bridge − card game . . . . . BRIDGE(A)
mille − card game . . . . . MILLE(A)
canfield, fish − card games . . . . . CARDS(A)
tolower, toupper − force upper or lower case . . . . . TOLOWER(3) 437
cat − catenate and print . . . . . CAT(1) 89
scat − sky catalogue . . . . . SCAT(7) 552
cb − C program beautifier . . . . . CB(1) 90
cbt − btree utilities . . . . . CBT(1) 91
CC, cfront − C++ compiler . . . . . C++(1) 87
cc, lcc − C compilers . . . . . CC(1) 92
rc, cd, wait, whatis − command language . . . . . RC(1) 215
programming language . . . . . sh, cd, wait, whatis − shell, the standard command . . . . . SH(1) 230
ceiling functions . . . . . fabs, fmod, floor, ceil − absolute value, remainder, floor, ... FLOOR(3) 365
lint, cyntax, cem − C program verifiers . . . . . LINT(1) 165
malloc, free, realloc, calloc, cfree − memory allocator . . . . . MALLOC(3) 392
CC, cfront − C++ compiler . . . . . C++(1) 87
brk, sbrk − change core allocation . . . . . BRK(2) 298
chmod, fchmod, chown, fchown, utime − change file mode, owner, group, or times . . . . . CHMOD(2) 300
passwd − change login password . . . . . PASSWD(1) 196
chmod − change mode . . . . . CHMOD(1) 96
chown − change owner or group . . . . . CHOWN(8) 566
renice − change priority of running process . . . . . RENICE(8) 604
newgrp − change to a new group . . . . . NEWGRP(1) 190
chdir, chroot − change working or root directory . . . . . CHDIR(2) 299
pipe − create an interprocess channel . . . . . PIPE(2) 317
stream − communication channels . . . . . STREAM(4) 468
ungetc − push character back into input stream . . . . . UNGETC(3) 441
chrtab − simple character bitmaps . . . . . CHRTAB(3) 347
streambuf − public interface of character buffering class . . . . . SBUF.PUB(A)
isprint, isgraph, iscntrl, isascii − character classification . . . . . /isspace, ispunct, CTYPE(3) 350
eqnchar − special character definitions for eqn . . . . . EQNCHAR(6) 520
fgetc, getw, putc, putchar, fputc, putw − character- or word-at-a-time stream/ . . . . . /getchar, GETC(3) 374
ocr − optical character recognition . . . . . OCR(1) 193
ascii − map of ASCII character set . . . . . ASCII(6) 519
asa − interpret ASA control characters . . . . . ASA(A)
ascii − interpret ASCII characters . . . . . ASCII(1) 72
tr − translate characters . . . . . TR(1) 264
vis − show invisible characters . . . . . VIS(1) 280
file . . . . . touch, chdate − set modification or access date of a . . . . . CHDATE(1) 95
directory . . . . . chdir, chroot − change working or root . . . . . CHDIR(2) 299
dcheck, ncheck − file system consistency check . . . . . icheck, ICHECK(8) 576
fsck − file system consistency check and interactive repair . . . . . FSCK(8) 573
eqn, neqn, checkeq − typeset mathematics . . . . . EQN(1) 132
chuck − a file system checking program . . . . . CHUCK(8) 567
checknews, readnews − read netnews articles . . . . . READNEWS(7) 550
wait wait3 − terminate process, wait for child to terminate . . . . . _exit, EXIT(2) 306
chmod − change mode . . . . . CHMOD(1) 96
chown − change owner or group . . . . . CHOWN(8) 566
owner, group, or times . . . . . chmod, fchmod, chown, fchown, utime − change file mode, ... CHMOD(2) 300
chdir, chroot − change working or root directory . . . . . CHDIR(2) 299
chrtab − simple character bitmaps . . . . . CHRTAB(3) 347
chuck − a file system checking program . . . . . CHUCK(8) 567
cin − C interpreter . . . . . CIN(1) 97
cip − draw pictures for typesetting . . . . . CIP(9.1) 641
circle-drawing functions . . . . . circle, disc, arc, ellipse, eldisc, elarc − . . . . . CIRCLE(9.3) 684
cite − process citations in a document . . . . . CITE(A)
uucleanup − uucp spool directory clean-up . . . . . UUCLEANUP(8) 626
clear − clear terminal screen . . . . . CLEAR(A)
feof, ferror, clearerr, fileno − stream status inquiries . . . . . FERROR(3) 361
backup − backup client administration . . . . . BACKUP(8) 565
wwv − print or set the date from accurate clock . . . . . WWV(1) 285
cron − clock daemon . . . . . CRON(8) 570
create file . . . . . open, creat, close − open a file for reading or writing, ... OPEN(2) 316
freopen, fdopen, fclose, fflush − open, close, or flush a stream . . . . . fopen, FOPEN(3) 366
opendir, readdir, telldir, seekdir, closedir − directory operations . . . . . DIRECTORY(3) 355
clri − clear inode . . . . . CLRI(8) 568
dk, dkp_ld, unixp_ld, cmc_ld − Datakit interface and protocols . . . . . DK(4) 451
cmp − compare two files . . . . . CMP(1) 99
segment, texture − graphics functions . . . . . Code, addr, bitblt, point, rectf, screenswap, ... BITBLT(9.3) 683
huff − huffman codebook/tree generator . . . . . HUFF(3) 385

alignment	col, 2, 3, 4, 5, 6, mc, fold, expand − column	COLUMN(1)	100
bundle −	collect files for distribution	BUNDLE(1)	86
cut, paste − rearrange	columns of data	CUT(1)	109
ldpcs − load	comet microcode	LDPCS(8)	581
sorted files	comm − select or reject lines common to two	COMM(1)	101
=, ==, =p, ==p − redo previous shell	command	=(1)	61
system − issue a shell	command	SYSTEM(3)	432
time − time a	command	TIME(1)	263
xargs − construct argument lists and execute	command	XARGS(A)	
getarg, iargc −	command arguments to Fortran	GETARG(3)	373
uux − unix to unix	command execution	UUX(1)	275
rc, cd, wait, whatis −	command language	RC(1)	215
getopt − parse	command options	GETOPT(A)	
sh, cd, wait, whatis − shell, the standard	command programming language	SH(1)	230
apply, pick − repeatedly apply a	command; select arguments	APPLY(1)	68
intro − introduction to	commands	INTRO(1)	59
test, [, newer − condition	commands	TEST(1)	259
at − execute	commands at a later time	AT(1)	73
nice, nohup − run	commands at low priority or immune to hangup	NICE(1)	191
/functions: absolute value, sign, greatest	common divisor, least common multiple, minimum,/	ARITH(3)	337
comm − select or reject lines	common to two sorted files	COMM(1)	101
stream −	communication channels	STREAM(4)	468
cmp −	compare two files	CMP(1)	99
diff, diff3 − differential file	comparison	DIFF(1)	119
idiff − interactive file	comparison	IDIFF(1)	154
3ar, 3ld, 3nm, 3size, 3strip, cprs − MAC-32 C	compiler 3cc, 3as,	3CC(9.1)	637
CC, cfront − C++	compiler	C++(1)	87
esterel − Esterel	compiler	ESTEREL(A)	
f77 − Fortran 77	compiler	F77(1)	136
gcc − GNU project C	Compiler	GCC(A)	
icont, iconc − Icon language translator and	compiler	ICONT(A)	
lcoc − Esterel	compiler	LCOC(A)	
lisp, liszt, lxref − lisp interpreter and	compiler	LISP(A)	
mcc − MC68000 C	compiler	MCC(A)	
pc − pascal language	compiler	PC(A)	
sml − Standard ML	compiler	SML(1)	239
spitbol − Snobol language	compiler	SPITBOL(A)	
yacc, eyacc − yet another	compiler-compiler	YACC(1)	287
cc, lcc − C	compilers	CC(1)	92
expand files pack, unpack, pcat,	compress, uncompress, zcat − compress and	PACK(1)	195
whoami −	computer name	WHOAMI(5)	513
	con, rx − remote login and execution	CON(1)	102
test, [, newer −	condition commands	TEST(1)	259
	config − configure a Unix kernel	CONFIG(A)	
/ipcpath , ipclogin, ipcrogin − set up	connections between processes or machines	IPC(3)	388
connection	conn_ld − line discipline for unique stream	CONNLD(4)	449
icheck, dcheck, ncheck − file system	consistency check	ICHECK(8)	576
fsck − file system	consistency check and interactive repair	FSCK(8)	573
	console − VAX console interface	CONSOLE(4)	450
mkfs, mkbitfs, mklost+found −	construct a disk file system	MKFS(8)	584
mknod −	construct special file	MKNOD(8)	585
asa − interpret ASA	control characters	ASA(A)	
ioctl − miscellaneous	control operations	IOCTL(2)	310
tcp, tcp_ld − DARPA transmission	control protocol	TCP(4)	470
units −	conversion program	UNITS(7)	556
dd, dblbuf −	convert and copy a file	DD(1)	116
number −	convert Arabic numerals to English	NUMBER(A)	
atof, atoi, atol, strtod, strtol, strtoul −	convert ASCII to numbers	ATOF(3)	339
timec, timegm, timelocal −	convert ASCII to time	TIMEC(3)	436
integers l3tol, ltol3 −	convert between 3-byte integers and long	L3TOL(3)	391
localtime, gmtime, asctime, timezone −	convert date and time to ASCII ctime,	CTIME(3)	349
f2c −	Convert Fortran 77 to C or C++	F2C(1)	135
ecvt, fcvt, gcvt −	convert numbers to ascii	ECVT(3)	356
bcd, ppt, morse −	convert to antique media	BCD(A)	
graphpic − edit (combinatoric) graphs,	convert to pic files graphdraw	GRAPHDRAW(9.1)	644
fortune − sample lines from a file, return	cookies random,	RANDOM(1)	214
pretty-printer cospan, psr −	coordination-specification analyzer and	COSPAN(A)	
push, pull, npush, npull − datakit remote file	copy	PUSH(1)	211
rcp − remote file	copy	RCP(1)	221
track − selective remote file	copy	TRACK(A)	
uucp, uulog, uuname − unix-to-unix remote file	copy	UUCP(1)	271
uupick − simplified unix-to-unix remote file	copy uuto,	UUTO(1)	274
dd, dblbuf − convert and	copy a file	DD(1)	116
cpio −	copy file archives in and out	CPIO(1)	104

cp, mv, ln, reccp −	copy, move, or link files	CP(1)	103
	core − format of memory image file	CORE(5)	485
brk, sbrk − change	core allocation	BRK(2)	298
savecore − save a	core image of the operating system	SAVECORE(8)	608
bigcore, coreid − permit big	core images, identify source of image	BIGCORE(1)	85
functions	cos, sin, atan2, sqrt, norm − integer math	COS(9.3)	685
trigonometric functions sin,	cos, tan, asin, acos, atan, atan2 −	SIN(3)	425
sinh,	cosh, tanh − hyperbolic functions	SINH(3)	426
ops5, pascal, ratfor, S, smp, sno,/ altran,	cospan, esterel, icon, lisp, macsyma, maple,	LANGS(1)	159
analyzer and pretty-printer	cospan, psr − coordination-specification	COSPAN(A)	
wc − word	count	WC(1)	281
sum, treesum − sum and	count blocks in a file	SUM(1)	251
sendcover − send	cover sheet to the library	SENDCOVER(8)	611
mcs − macros for formatting	cover sheets	MCS(6)	527
	cp, mv, ln, reccp − copy, move, or link files	CP(1)	103
	cpio − copy file archives in and out	CPIO(1)	104
	cpio − format of cpio archive	CPIO(5)	486
	cpp − C language preprocessor	CPP(8)	569
3cc, 3as, 3ar, 3ld, 3nm, 3size, 3strip,	cprs − MAC-32 C compiler	3CC(9.1)	637
	crabs − graphical marine adventure game	CRABS(A)	
	cray − run job remotely on cray-xmp	CRAY(A)	
writing, create file open,	creat, close − open a file for reading or	OPEN(2)	316
pipe −	create an interprocess channel	PIPE(2)	317
windows, jps, reshape −	create and initialize windows	WINDOWS(9.1)	674
close − open a file for reading or writing,	create file open, creat,	OPEN(2)	316
umask − set file	creation mode mask	UMASK(2)	330
	cron − clock daemon	CRON(8)	570
dist, dme, plan, path,	cross − aviation navigation	AV(A)	
pxp, pxref − pascal printer, profiler, and	cross-reference lister	PXP(A)	
	crypt, encrypt, decrypt − encode/decode	CRYPT(1)	105
	crypt, setkey, encrypt − DES encryption	CRYPT(3)	348
	cscan − scan documents on canon scanner	CSCAN(1)	106
− convert date and time to ASCII	ctime, localtime, gmtime, asctime, timezone	CTIME(3)	349
system	cu, ct − call out to a terminal or another	CU(1)	107
getwd, getcwd − get	current directory	GETWD(3)	384
cursor motion	curses − screen functions with ‘optimal’	CURSES(3)	351
button123, mouse, cursallow, cursinhibit,	cursset, cursswitch, getrect123 − mouse/	BUTTON(9.2)	675
spline − fit a	curve	SPLINE(A)	
	cut, paste − rearrange columns of data	CUT(1)	109
lint,	cyntax, cem − C program verifiers	LINT(1)	165
	d202, tc − typesetter filters	D202(A)	
cron − clock	daemon	CRON(8)	570
	dag − preprocessor for drawing directed graphs	DAG(1)	110
ip, ip_ld −	DARPA internet protocol	IP(4)	456
tcp, tcp_ld −	DARPA transmission control protocol	TCP(4)	470
300, 300s − handle special functions of	DASI 300 and 300s terminals	300(A)	
450 − handle special functions of the	DASI 450 terminal	450(A)	
cut, paste − rearrange columns of	data	CUT(1)	109
null −	data sink	NULL(4)	462
types − primitive system	data types	TYPES(5)	510
Rect, Rpt, display, Drect, Jrect − graphics	data types /Rectangle, Bitmap, Texture, Pt,	TYPES(9.5)	699
join − relational	database operator	JOIN(1)	156
fetch, store, delete, firstkey, nextkey −	database subroutines dbminit,	DBM(3)	353
dkname − map system name to	Datakit address	DKNAME(7)	540
dkhup, dkmgr, dkzap − manage	Datakit interface	DKMGR(8)	571
dk, dkp_ld, unixp_ld, cmc_ld −	Datakit interface and protocols	DK(4)	451
dkstat − show network status for internet and	datakit networks netstat,	NETSTAT(8)	590
push, pull, npush, npull −	datakit remote file copy	PUSH(1)	211
	date − print or set the date	DATE(1)	112
time, ftime − get	date and time	TIME(2)	328
gmtime, asctime, timezone − convert	date and time to ASCII ctime, localtime,	CTIME(3)	349
wwv − print or set the	date from accurate clock	WWV(1)	285
touch, chdate − set modification or access	date of a file	CHDATE(1)	95
dd,	dblbuf − convert and copy a file	DD(1)	116
nextkey − database subroutines	dbminit, fetch, store, delete, firstkey,	DBM(3)	353
	dc − desk calculator	DC(1)	113
icheck,	dcheck, ncheck − file system consistency check	ICHECK(8)	576
remote login and execution	dcon, ndcon, rlogin, nrx, rsh, scriptcon −	DCON(1)	115
	dd, dblbuf − convert and copy a file	DD(1)	116
uucico, uusched, uuxqt, kick,	debug − uucp file transport and execution	UUCICO(8)	625
P, newproc, muxnewwind, newwindow, tolayer,	debug, getproc, getproctab, putname, getname −/	NEWPROC(9.2)	677
adb −	debugger	ADB(1)	62
sdb − symbolic	debugger	SDB(A)	
xd, od − hex, octal,	decimal, or ASCII dump	XD(1)	286

crypt, encrypt, decrypt − encode/decode . . . . . CRYPT(1) 105
getfont − replace terminal's default font . . . . . GETFONT(9.1) 643
text and font operations . . . . . string, defont, strwidth, infont, outfont, getfont − . . . . . STRING(9.3) 690
deroff, demonk, detex, delatex − remove formatting requests . . . . . DEROFF(1) 118
alarm, nap, pause − schedule timing delays . . . . . ALARM(2) 297
subroutines . . . . . dbminit, fetch, store, delete, firstkey, nextkey − database . . . . . DBM(3) 353
lsegment, ltexture, upfront − layer/ . . . . . newlayer, dellayer, downback, lbitblt, lpoint, lrectf, ... NEWLAYER(9.2) 676
games, demo − some playthings . . . . . GAMES(1) 143
requests . . . . . deroff, demonk, detex, delatex − remove formatting . . . . . DEROFF(1) 118
crypt, setkey, encrypt − DES encryption . . . . . CRYPT(3) 348
dup, dup2 − duplicate an open file descriptor . . . . . DUP(2) 303
fd, stdin, stdout, stderr, tty − file descriptor files . . . . . FD(4) 455
dc − desk calculator . . . . . DC(1) 113
matlab − interactive matrix desk calculator . . . . . MATLAB(1) 182
deroff, demonk, detex, delatex − remove formatting requests . . . . . DEROFF(1) 118
drum − paging device . . . . . DRUM(4) 453
swapon − specify swapping device . . . . . SWAPON(8) 620
intro − introduction to devices, line disciplines, and file systems . . . . . INTRO(4) 447
du, df − disk usage . . . . . DU(1) 125
dmesg − system diagnostic messages . . . . . DMESG(8) 572
dialout − place call on ACU . . . . . DIALOUT(3) 354
dict − look up words in English dictionaries . . . . . DICT(7) 538
wwb, style, diction, punct − writer's workbench . . . . . WWB(1) 284
patch − a program for applying a diff file to an original . . . . . PATCH(A)
diff, diff3 − differential file comparison . . . . . DIFF(1) 119
qsnap − high resolution digital film printer . . . . . QSNAP(1) 213
map − digitized map formats . . . . . MAP(5) 494
dir − format of directories . . . . . DIR(5) 487
dag − preprocessor for drawing directed graphs . . . . . DAG(1) 110
chdir, chroot − change working or root directory . . . . . CHDIR(2) 299
getwd, getcwd − get current directory . . . . . GETWD(3) 384
ls, lc − list contents of directory . . . . . LS(1) 171
mkdir − make a directory . . . . . MKDIR(1) 187
mkdir, rmdir − make or remove a directory . . . . . MKDIR(2) 313
pwd, where − machine name and working directory . . . . . PWD(1) 212
uucleanup − uucp spool directory clean-up . . . . . UUCLEANUP(8) 626
dired − directory editor . . . . . DIRED(1) 121
unlink − remove directory entry . . . . . UNLINK(2) 331
dirread − read from directory, hiding format . . . . . DIRREAD(2) 302
readdir, telldir, seekdir, closedir − directory operations . . . . . opendir, DIRECTORY(3) 355
mknod − make a directory or a special file . . . . . MKNOD(2) 314
dired − directory editor . . . . . DIRED(1) 121
basename, dirname − strip filename affixes . . . . . BASENAME(1) 80
dirread − read from directory, hiding format . . . . . DIRREAD(2) 302
circle-drawing functions . . . . . circle, disc, arc, ellipse, eldisc, elarc − . . . . . CIRCLE(9.3) 684
filsys, flblk, ino − format of a disk file system . . . . . FILSYS(5) 489
mkfs, mkbitfs, mklost+found − construct a disk file system . . . . . MKFS(8) 584
sync, update − update disk file systems . . . . . SYNC(8) 621
smash − rewrite bad disk sectors . . . . . SMASH(8) 614
du, df − disk usage . . . . . DU(1) 125
worm, jukebox − optical disk utilities . . . . . WORM(8) 629
rarepl, rarct − replace bad blocks on MSCP disks . . . . . RAREPL(8) 600
worm − format of worm disks . . . . . WORM(5) 514
ra − DEC MSCP disks (RA60, RA80, RA81, RA90) . . . . . RA(4) 465
mount, umount − mount and dismount file system . . . . . MOUNT(8) 587
/Rectangle, Bitmap, Texture, Pt, Rect, Rpt, display, Drect, Jrect − graphics data types . . . . . TYPES(9.5) 699
rscan, pix − scan page on ricoh scanner and display on 5620 . . . . . RSCAN(A)
navigation . . . . . dist, dme, plan, path, cross − aviation . . . . . AV(A)
hypot, cabs − euclidean distance . . . . . HYPOT(3) 386
asd − automatic software distribution . . . . . ASD(8) 564
bundle − collect files for distribution . . . . . BUNDLE(1) 86
unseal − package files for automatic software distribution . . . . . mkpkg, inspkg, seal, MKPKG(8) 586
ship, shipstat − automatic software distribution . . . . . SHIP(8) 612
raddp, rsubp, rectXrect,/ . . . . . add, sub, mul, div, eqpt, eqrect, inset, muldiv, ptinrect, ... ADD(9.3) 681
maximum . . /absolute value, sign, greatest common divisor, least common multiple, minimum, ... ARITH(3) 337
interface and protocols . . . . . dk, dkp_ld, unixp_ld, cmc_ld − Datakit . . . . . DK(4) 451
protocols . . . . . ipconfig, dkipconfig, udpconfig − set up DARPA Internet . . . . . IPCONFIG(8) 579
dkhup, dkmgr, dkzap − manage Datakit interface . . . . . DKMGR(8) 571
dkname − map system name to Datakit address . . . . . DKNAME(7) 540
and protocols . . . . . dk, dkp_ld, unixp_ld, cmc_ld − Datakit interface . . . . . DK(4) 451
datakit networks . . . . . netstat, dkstat − show network status for internet and . . . . . NETSTAT(8) 590
dkhup, dkmgr, dkzap − manage Datakit interface . . . . . DKMGR(8) 571
dist, dme, plan, path, cross − aviation navigation . . . . . AV(A)
dmesg − system diagnostic messages . . . . . DMESG(8) 572

	docgen − generate a document from a script	DOCGEN(1) 122
	docsubmit − send document to library	DOCSUBMIT(1) 123
	doctor, tso − psychiatric consultation	DOCTOR(A)
document	doctype − guess command line for formatting a	DOCTYPE(1) 124
cite − process citations in a	document	CITE(A)
docgen − generate a	document from a script	DOCGEN(1) 122
docsubmit − send	document to library	DOCSUBMIT(1) 123
reader − electronic retrieval of typeset	documents	READER(9.7) 702
monksample, monkspell, monkmerge − typeset	documents and letters monk,	MONK(1) 188
cscan − scan	documents on canon scanner	CSCAN(1) 106
twid, pen −	doodle on the screen	TWID(A)
ltexture, upfront − layer/ newlayer, dellayer,	downback, lbitblt, lpoint, lrectf, lsegment,	NEWLAYER(9.2) 676
to produce postscript	dpost, postdaisy, postdmd, postprint − filters	POSTSCRIPT(8) 594
tel, telno,	dq − phone books	TEL(7) 554
graph −	draw a graph	GRAPH(1) 148
map −	draw maps on various projections	MAP(7) 542
cip −	draw pictures for typesetting	CIP(9.1) 641
paint −	draw pictures in a layer	PAINT(9.1) 654
dag − preprocessor for	drawing directed graphs	DAG(1) 110
disc, arc, ellipse, eldisc, elarc − circle-	drawing functions circle,	CIRCLE(9.3) 684
grap − pic preprocessor for	drawing graphs	GRAP(1) 146
ideal − troff preprocessor for	drawing pictures	IDEAL(1) 152
pic, tpic − troff and tex preprocessors for	drawing pictures	PIC(1) 197
Bitmap, Texture, Pt, Rect, Rpt, display,	Drect, Jrect − graphics data types /Rectangle,	TYPES(9.5) 699
tty − serial line interface	drivers	TTY(4) 472
	drum − paging device	DRUM(4) 453
	du, df − disk usage	DU(1) 125
xd, od − hex, octal, decimal, or ASCII	dump	XD(1) 286
adventure, zork, rogue, wump −	dungeon-exploration games	ADVENTURE(A)
descriptor	dup, dup2 − duplicate an open file	DUP(2) 303
tex, initex, virtex,	dvips, dvit − text formatting and typesetting	TEX(1) 261
strzero, setmuxbuf, getmuxbuf, movstring −	dynamic strings in mux /strinsert, strdelete,	MUXSTRING(9.3) 689
ed,	e − text editor	ED(1) 127
	echo, printf − print arguments	ECHO(1) 126
	ecvt, fcvt, gcvt − convert numbers to ascii	ECVT(3) 356
	ed, e − text editor	ED(1) 127
end, etext,	edata − last locations in program	END(3) 357
files graphdraw graphpic −	edit (combinatoric) graphs, convert to pic	GRAPHDRAW(9.1) 644
ex,	edit, vi − text editor	VI(1) 276
dired − directory	editor	DIRED(1) 121
ed, e − text	editor	ED(1) 127
icon − icon	editor	ICON(9.1) 646
jf − font	editor	JF(9.1) 647
jim, jim.recover − text	editor	JIM(A)
ped, tped − picture	editor	PED(9.1) 656
pico − graphics	editor	PICO(1) 200
rebecca − graphics touch-up	editor	REBECCA(9.1) 663
sed − stream	editor	SED(1) 226
ex, edit, vi − text	editor	VI(1) 276
samuel − text	editor and C browser	SAMUEL(A)
ld − link	editor or loader	LD(1) 162
sam − screen	editor with structural regular expressions	SAM(9.1) 665
	efl − extended Fortran language preprocessor	EFL(A)
gre, grep,	egrep, fgrep − search a file for a pattern	GRE(1) 149
soelim −	eliminate .so's from nroff input	SOELIM(A)
functions circle, disc, arc,	ellipse, eldisc, elarc − circle-drawing	CIRCLE(9.3) 684
crypt,	encrypt, decrypt − encode/decode	CRYPT(1) 105
crypt, setkey,	encrypt − DES encryption	CRYPT(3) 348
makekey − generate	encryption key	MAKEKEY(8) 583
program	end, etext, edata − last locations in	END(3) 357
getfsent, getfsspec, getfsfile, setfsent,	endfsent − get file system description file/	GETFSENT(3) 378
getgrent, getgrgid, getgrnam, setgrent,	endgrent − get group file entry	GETGRENT(3) 379
getpwent, getpwuid, getpwnam, setpwent,	endpwent, pwdecode − get password file entry	GETPWENT(3) 383
number − convert Arabic numerals to	English	NUMBER(A)
dict − look up words in	English dictionaries	DICT(7) 538
unlink − remove directory	entry	UNLINK(2) 331
	environ − user environment	ENVIRON(5) 488
printenv − print	environment	PRINTENV(1) 207
getenv − value for	environment name	GETENV(3) 375
sky − astronomical	ephemeris	SKY(7) 553
	eqn, neqn, checkeq − typeset mathematics	EQN(1) 132
eqn	eqnchar − special character definitions for	EQNCHAR(6) 520
rsubp, rectXrect, rectclip/ add, sub, mul, div,	eqpt, eqrect, inset, muldiv, ptinrect, raddp,	ADD(9.3) 681
	erf, erfc − error function	ERF(3) 358

error numbers . . . . . intro, errno – introduction to system calls and . . . . . INTRO(2) 291
perror, sys_errlist, sys_nerr – system error messages . . . . . PERROR(3) 400
bianchi – espresso, steamed milk, hot water . . . . . BIANCHI(A)
esterel – Esterel compiler . . . . . ESTEREL(A)
pascal, ratfor, S, smp, sno,/ . . . . . altran, cospan, esterel, icon, lisp, macsyma, maple, ops5, ... LANGS(1) 159
end, etext, edata – last locations in program . . . . . END(3) 357
ethernet – Ethernet interface . . . . . ETHERNET(4) 454
hypot, cabs – euclidean distance . . . . . HYPOT(3) 386
expr – evaluate arguments as an expression . . . . . EXPR(1) 134
ex, edit, vi – text editor . . . . . VI(1) 276
exect – execute a file . . . . . execl, execv, execle, execve, execlp, execvp, ... EXEC(2) 304
xargs – construct argument lists and execute command . . . . . XARGS(A)
at – execute commands at a later time . . . . . AT(1) 73
– execute a file . . . . . execl, execv, execle, execve, execlp, execvp, exect . . . . . EXEC(2) 304
rcvchar, realtime, sendchar, sendnchars, kill, exit – 5620 input/output requests . . . . . /kbdchar, REQUEST(9.2) 679
exit, onexit – terminate process . . . . . EXIT(3) 359
for child to terminate . . . . . _exit, wait wait3 – terminate process, wait . . . . . EXIT(2) 306
logarithm, power, square root . . . . . exp, log, log10, pow, sqrt – exponential, ... EXP(3) 360
col, 2, 3, 4, 5, 6, mc, fold, expand – column alignment . . . . . COLUMN(1) 100
frexp, ldexp, modf – split into mantissa and exponent . . . . . FREXP(3) 368
exp, log, log10, pow, sqrt – exponential, logarithm, power, square root . . . . . EXP(3) 360
expr – evaluate arguments as an expression . . . . . EXPR(1) 134
yacc, eyacc – yet another compiler-compiler . . . . . YACC(1) 287
f2c – Convert Fortran 77 to C or C++ . . . . . F2C(1) 135
f77 – Fortran 77 compiler . . . . . F77(1) 136
remainder, floor, ceiling functions . . . . . fabs, fmod, floor, ceil – absolute value, ... FLOOR(3) 365
pictures . . . . . face, mugs – show faces, make face icons from . . . . . FACE(9.7) 701
faced – network face server . . . . . FACED(9.5) 695
factor, qfactor, primes – factor a number, generate large primes . . . . . FACTOR(1) 138
true, false – provide truth values . . . . . TRUE(1) 266
Fwrite, Fflush, Ftie, Fclose, Fexit – fast buffered input/output . . . . . /Fputc, Fprint, FIO(3) 363
pool – fast memory allocation . . . . . POOL(3+) 406
abort – generate a fault . . . . . ABORT(3) 336
mode, owner, group, or times . . . . . chmod, fchmod, chown, fchown, utime – change file . . . . . CHMOD(2) 300
/Fundo, Fputc, Fprint, Fwrite, Fflush, Ftie, Fclose, Fexit – fast buffered input/output . . . . . FIO(3) 363
stream . . . . . fopen, freopen, fdopen, fclose, fflush – open, close, or flush a . . . . . FOPEN(3) 366
ecvt, fcvt, gcvt – convert numbers to ascii . . . . . ECVT(3) 356
w, fp, ft, fd, rad – aviation weather . . . . . AVW(A)
descriptor files . . . . . fd, stdin, stdout, stderr, tty – file . . . . . FD(4) 455
flush a stream . . . . . fopen, freopen, fdopen, fclose, fflush – open, close, or . . . . . FOPEN(3) 366
status inquiries . . . . . feof, ferror, clearerr, fileno – stream . . . . . FERROR(3) 361
festoon – memo writer . . . . . FESTOON(A)
database subroutines . . . . . dbminit, fetch, store, delete, firstkey, nextkey – . . . . . DBM(3) 353
5620 . . . . . flicks, fflicks, preflicks, 2mux – movie graphics for . . . . . FLICKS(9.1) 642
fopen, freopen, fdopen, fclose, fflush – open, close, or flush a stream . . . . . FOPEN(3) 366
Fwrite, Fflush, Ftie, Fclose,/ . . . . . Finit, Frdline, Fgetc, Fread, Fseek, Fundo, Fputc, Fprint, ... FIO(3) 363
character- or word-at-a-time/ . . . . . getc, getchar, fgetc, getw, putc, putchar, fputc, putw – . . . . . GETC(3) 374
on streams . . . . . fgets, puts, fputs, gets – string input/out . . . . . FGETS(3) 362
gre, grep, egrep, fgrep – search a file for a pattern . . . . . GRE(1) 149
getmfields, setfields – break a string into fields . . . . . getfields, GETFIELDS(3) 376
access – determine accessibility of file . . . . . ACCESS(2) 295
bitfile – format of bitmap file . . . . . BITFILE(9.5) 694
chdate – set modification or access date of a file . . . . . touch, CHDATE(1) 95
dd, dblbuf – convert and copy a file . . . . . DD(1) 116
execve, execlp, execvp, exect – execute a file . . . . . execl, execv, execle, EXEC(2) 304
link, symlink, readlink – link to a file . . . . . LINK(2) 311
mknod – make a directory or a special file . . . . . MKNOD(2) 314
mknod – construct special file . . . . . MKNOD(8) 585
– open a file for reading or writing, create file . . . . . open, creat, close OPEN(2) 316
pr – print file . . . . . PR(1) 203
read, write – read or write file . . . . . READ(2) 319
size – size of an object file . . . . . SIZE(1) 237
stdiobuf – iostream specialized to stdio FILE . . . . . STDIOBUF(A)
strings – find printable strings in a file . . . . . STRINGS(1) 247
sum, treesum – sum and count blocks in a file . . . . . SUM(1) 251
tail, readslow – deliver the last part of a file . . . . . TAIL(1) 253
uniq – report repeated lines in a file . . . . . UNIQ(1) 270
cpio – copy file archives in and out . . . . . CPIO(1) 104
diff, diff3 – differential file comparison . . . . . DIFF(1) 119
idiff – interactive file comparison . . . . . IDIFF(1) 154
push, pull, npush, npull – datakit remote file copy . . . . . PUSH(1) 211
rcp – remote file copy . . . . . RCP(1) 221
track – selective remote file copy . . . . . TRACK(A)
uucp, uulog, uuname – unix-to-unix remote file copy . . . . . UUCP(1) 271

uuto, uupick − simplified unix-to-unix remote . . . file copy . . . UUTO(1) 274
umask – set . . . file creation mode mask . . . UMASK(2) 330
dup, dup2 – duplicate an open . . . file descriptor . . . DUP(2) 303
fd, stdin, stdout, stderr, tty − . . . file descriptor files . . . FD(4) 455
endfsent – get file system description . . . file entry . . . /getfsspec, getfsfile, setfsent, . . . GETFSENT(3) 378
gre, grep, egrep, fgrep – search a . . . file for a pattern . . . GRE(1) 149
open, creat, close – open a . . . file for reading or writing, create file . . . OPEN(2) 316
a.out − object . . . file format . . . A.OUT(5) 479
ar − archive (library) . . . file format . . . AR(5) 482
movies − graphics movie . . . file formats . . . FLICKFILE(9.5) 696
filebuf − buffer for . . . file input/output . . . FILEBUF(A)
split, fsplit − split a . . . file into pieces . . . SPLIT(1) 246
picunpack, picpack, picerror − picture . . . file I/O . . . /picclose, picputprop, picgetprop, . . . PICFILE(3) 401
file – determine file type . . . FILE(1) 139
chmod, fchmod, chown, fchown, utime – change . . . file mode, owner, group, or times . . . CHMOD(2) 300
mktemp, tmpnam – make a unique . . . file name . . . MKTEMP(3) 396
byteyears − time-space product for . . . file residency . . . BYTEYEARS(A)
random, fortune − sample lines from a . . . file, return cookies . . . RANDOM(1) 214
stat, lstat, fstat – get . . . file status . . . STAT(2) 323
filsys, flblk, ino − format of a disk . . . file system . . . FILSYS(5) 489
fmount, funmount − mount or remove . . . file system . . . FMOUNT(2) 307
mkfs, mkbitfs, mklost+found − construct a disk . . . file system . . . MKFS(8) 584
mount, umount − mount and dismount . . . file system . . . MOUNT(8) 587
netfs − network . . . file system . . . NETFS(8) 588
proc − process . . . file system . . . PROC(4) 463
chuck − a . . . file system checking program . . . CHUCK(8) 567
icheck, dcheck, ncheck − . . . file system consistency check . . . ICHECK(8) 576
repair . . . fsck − . . . file system consistency check and interactive . . . FSCK(8) 573
getfsfile, setfsent, endfsent – get . . . file system description file entry . . . /getfsspec, . . . GETFSENT(3) 378
quot, findo − . . . file system usage and hogs . . . QUOT(8) 599
fstab, mtab − information about . . . file systems . . . FSTAB(5) 493
introduction to devices, line disciplines, and . . . file systems . . . intro − . . . INTRO(4) 447
sync, update − update disk . . . file systems . . . SYNC(8) 621
uucico, uusched, uuxqt, kick, debug − uucp . . . file transport and execution . . . UUCICO(8) 625
ftw − . . . file tree walk . . . FTW(3) 370
file – determine . . . file type . . . FILE(1) 139
filebuf − buffer for file input/output . . . FILEBUF(A)
basename, dirname − strip . . . filename affixes . . . BASENAME(1) 80
feof, ferror, clearerr, . . . fileno – stream status inquiries . . . FERROR(3) 361
backup − backup and recover . . . files . . . BACKUP(1) 77
backup − incremental backup . . . files . . . BACKUP(5) 483
bcp – reformat black-and-white picture . . . files . . . BCP(1) 83
cmp − compare two . . . files . . . CMP(1) 99
− select or reject lines common to two sorted . . . files . . . comm . . . COMM(1) 101
cp, mv, ln, reccp − copy, move, or link . . . files . . . CP(1) 103
stdin, stdout, stderr, tty − file descriptor . . . files . . . fd, . . . FD(4) 455
find − find . . . files . . . FIND(1) 140
− iostream and streambuf specialized to . . . files . . . fstream . . . FSTREAM(A)
uncompress, zcat – compress and expand . . . files . . . pack, unpack, pcat, compress, . . . PACK(1) 195
rm − remove (unlink) . . . files . . . RM(1) 225
sort − sort and/or merge . . . files . . . SORT(1) 242
mkpkg, inspkg, seal, unseal − package . . . files for automatic software distribution . . . MKPKG(8) 586
bundle − collect . . . files for distribution . . . BUNDLE(1) 86
font − description . . . files for troff . . . FONT(5) 492
qsnap – high resolution digital . . . film printer . . . QSNAP(1) 213
system . . . filsys, flblk, ino − format of a disk file . . . FILSYS(5) 489
d202, tc − typesetter . . . filters . . . D202(A)
plot – graphics . . . filters . . . PLOT(1) 202
dpost, postdaisy, postdmd, postprint − . . . filters to produce postscript . . . POSTSCRIPT(8) 594
find − find files . . . FIND(1) 140
look – . . . find lines in a sorted list . . . LOOK(1) 168
ttyname, isatty, nametty – . . . find or set name of a terminal . . . TTYNAME(3) 438
strings − . . . find printable strings in a file . . . STRINGS(1) 247
database of locally authored papers . . . findauthor, papers, makepaper − consult . . . PAPERS(7) 546
quot, . . . findo − file system usage and hogs . . . QUOT(8) 599
mem, kmem, kUmem, mtpr, . . . fineclock − memory and VAX processor registers . . . MEM(4) 458
Fputc, Fprint, Fwrite, Fflush, Ftie, Fclose,/ . . . Finit, Frdline, Fgetc, Fread, Fseek, Fundo, . . . FIO(3) 363
dbminit, fetch, store, delete, . . . firstkey, nextkey – database subroutines . . . DBM(3) 353
canfield, . . . fish − card games . . . CARDS(A)
spline − . . . fit a curve . . . SPLINE(A)
getflags – process . . . flag arguments in argv . . . GETFLAGS(3) 377
filsys, . . . flblk, ino − format of a disk file system . . . FILSYS(5) 489
graphics for 5620 . . . flicks, fflicks, preflicks, 2mux − movie . . . FLICKS(9.1) 642
hoc – interactive . . . floating point language . . . HOC(1) 151

floor, ceiling functions . . . . . fabs, fmod, — floor, ceil – absolute value, remainder, ... FLOOR(3) 365
fdopen, fclose, fflush – open, close, or — flush a stream . . . . . fopen, freopen, FOPEN(3) 366
msub, mult, mdiv, sdiv, msqrt, mgcd, min, mout, — fmin, fmout, move, mcmp, rpow, mpow –/ . . . . /madd, MP(3) 398
remainder, floor, ceiling functions . . . . . fabs, — fmod, floor, ceil – absolute value, ... FLOOR(3) 365
fmount, funmount − mount or remove file system . . . . . FMOUNT(2) 307
/mult, mdiv, sdiv, msqrt, mgcd, min, mout, fmin, — fmout, move, mcmp, rpow, mpow – multiple/ . . . . . MP(3) 398
fmt – ultra-simple text formatter . . . . . FMT(1) 142
col, 2, 3, 4, 5, 6, mc, — fold, expand − column alignment . . . . . COLUMN(1) 100
getfont − replace terminal's default — font . . . . . GETFONT(9.1) 643
font − description files for troff . . . . . FONT(5) 492
font − jerq font layouts . . . . . FONT(9.5) 697
font − typesetter fonts . . . . . FONT(6) 521
jf − — font editor . . . . . JF(9.1) 647
strwidth, infont, outfont, getfont − text and — font operations . . . . . string, defont, STRING(9.3) 690
open, close, or flush a stream . . . . . fopen, freopen, fdopen, fclose, fflush – . . . . . FOPEN(3) 366
write, mesg − write to other users, allow or — forbid messages . . . . . WRITE(1) 283
sync − — force writing of system buffers . . . . . SYNC(2) 326
weather − conditions and — forecast by town . . . . . WEATHER(7) 557
fork – spawn new process . . . . . FORK(2) 308
a.out − object file — format . . . . . A.OUT(5) 479
ar − archive (library) file — format . . . . . AR(5) 482
dirread – read from directory, hiding — format . . . . . DIRREAD(2) 302
picfile − raster graphic image — format . . . . . PICFILE(5) 497
filsys, flblk, ino − — format of a disk file system . . . . . FILSYS(5) 489
bitfile − — format of bitmap file . . . . . BITFILE(9.5) 694
cpio − — format of cpio archive . . . . . CPIO(5) 486
dir − — format of directories . . . . . DIR(5) 487
core − — format of memory image file . . . . . CORE(5) 485
worm − — format of worm disks . . . . . WORM(5) 514
tbl − — format tables for nroff or troff . . . . . TBL(1) 256
movies − graphics movie file — formats . . . . . FLICKFILE(9.5) 696
map − digitized map — formats . . . . . MAP(5) 494
istream − — formatted and unformatted input . . . . . ISTREAM(A)
ostream − — formatted and unformatted output . . . . . OSTREAM(A)
scanf, fscanf, sscanf – — formatted input . . . . . SCANF(3) 421
printf, fprintf, sprintf, snprintf – print — formatted output . . . . . PRINTF(3) 411
fmt – ultra-simple text — formatter . . . . . FMT(1) 142
ios − input/output — formatting . . . . . IOS(A)
doctype − guess command line for — formatting a document . . . . . DOCTYPE(1) 124
iostream − buffering, — formatting and input/output . . . . . IOS.INTRO(A)
tex, initex, virtex, dvips, dvit − text — formatting and typesetting . . . . . TEX(1) 261
troff, nroff − text — formatting and typesetting . . . . . TROFF(1) 265
mcs − macros for — formatting cover sheets . . . . . MCS(6) 527
ms − macros for — formatting manuscripts . . . . . MS(6) 530
deroff, demonk, detex, delatex − remove — formatting requests . . . . . DEROFF(1) 118
getarg, iargc – command arguments to — Fortran . . . . . GETARG(3) 373
port – mathematical library for — Fortran . . . . . PORT(3) 408
f77 – — Fortran 77 compiler . . . . . F77(1) 136
f2c – Convert — Fortran 77 to C or C++ . . . . . F2C(1) 135
ratfor − rational — Fortran dialect . . . . . RATFOR(A)
efl − extended — Fortran language preprocessor . . . . . EFL(A)
struct − structure — Fortran programs . . . . . STRUCT(A)
cookies . . . . . random, — fortune − sample lines from a file, return . . . . . RANDOM(1) 214
w, — fp, ft, fd, rad – aviation weather . . . . . AVW(A)
/Frdline, Fgetc, Fread, Fseek, Fundo, Fputc, — Fprint, Fwrite, Fflush, Ftie, Fclose, Fexit/ . . . . . FIO(3) 363
output . . . . . print, — fprint, sprint, fmtinstall – print formatted . . . . . PRINT(3) 409
output . . . . . printf, — fprintf, sprintf, snprintf – print formatted . . . . . PRINTF(3) 411
Finit, Frdline, Fgetc, Fread, Fseek, Fundo, — Fputc, Fprint, Fwrite, Fflush, Ftie, Fclose,/ . . . . . FIO(3) 363
getc, getchar, fgetc, getw, putc, putchar, — fputc, putw – character- or word-at-a-time/ . . . . . GETC(3) 374
fgets, puts, — fputs, gets – string input/out on streams . . . . . FGETS(3) 362
generator . . . . . rand, lrand, — frand, nrand, srand – random number . . . . . RAND(3) 417
Fprint, Fwrite, Fflush, Ftie, Fclose,/ . . . . . Finit, — Frdline, Fgetc, Fread, Fseek, Fundo, Fputc, ... FIO(3) 363
fread, fwrite – buffered binary input/output . . . . . FREAD(3) 367
allocate memory . . . . . alloc, — free, balloc, bfree, gcalloc, gcfree − . . . . . ALLOC(9.3) 682
allocator . . . . . malloc, — free, realloc, calloc, cfree − memory . . . . . MALLOC(3) 392
close, or flush a stream . . . . . fopen, — freopen, fdopen, fclose, fflush – open, ... FOPEN(3) 366
exponent . . . . . — frexp, ldexp, modf – split into mantissa and . . . . . FREXP(3) 368
scanf, — fscanf, sscanf – formatted input . . . . . SCANF(3) 421
interactive repair . . . . . — fsck − file system consistency check and . . . . . FSCK(8) 573
fseek, ftell, rewind – reposition a stream . . . . . FSEEK(3) 369
Ftie, Fclose,/ . . . . . Finit, Frdline, Fgetc, Fread, — Fseek, Fundo, Fputc, Fprint, Fwrite, Fflush, ... FIO(3) 363
split, — fsplit − split a file into pieces . . . . . SPLIT(1) 246
fstab, mtab − information about file systems . . . . . FSTAB(5) 493
stat, lstat, — fstat – get file status . . . . . STAT(2) 323

to files . . . . . fstream − iostream and streambuf specialized . . . . . FSTREAM(A)
w, fp, ft, fd, rad – aviation weather . . . . . AVW(A)
fseek, ftell, rewind – reposition a stream . . . . . FSEEK(3) 369
Fseek, Fundo, Fputc, Fprint, Fwrite, Fflush, Ftie, Fclose, Fexit – fast buffered/ . . . . . /Fread, FIO(3) 363
time, ftime – get date and time . . . . . TIME(2) 328
ftw − file tree walk . . . . . FTW(3) 370
fmount, funmount − mount or remove file system . . . . . FMOUNT(2) 307
/Fgetc, Fread, Fseek, Fundo, Fputc, Fprint, Fwrite, Fflush, Ftie, Fclose, Fexit – fast/ . . . . . FIO(3) 363
fread, fwrite – buffered binary input/output . . . . . FREAD(3) 367
with garbage collection . . . . . galloc, gfree, garbage – storage allocation . . . . . GALLOC(3) 371
bridge − card game . . . . . BRIDGE(A)
crabs − graphical marine adventure game . . . . . CRABS(A)
mille − card game . . . . . MILLE(A)
ogre − war game . . . . . OGRE(A)
redcode − assembler for mars game . . . . . REDCODE(A)
zork, rogue, wump − dungeon-exploration games . . . . . adventure, ADVENTURE(A)
canfield, fish − card games . . . . . CARDS(A)
hangman, word_clout − word games . . . . . HANGMAN(A)
snake, worm − display chase games . . . . . SNAKE(A)
warp − war games . . . . . WARP(A)
games, demo − some playthings . . . . . GAMES(1) 143
gamma – log gamma function . . . . . GAMMA(3) 372
collection . . . . . galloc, gfree, garbage – storage allocation with garbage . . . . . GALLOC(3) 371
route, routed − IP gateway routing . . . . . ROUTE(8) 605
town − gazetteer of US places . . . . . TOWN(7) 555
alloc, free, balloc, bfree, gcalloc, gcfree − allocate memory . . . . . ALLOC(9.3) 682
gcc − GNU project C Compiler . . . . . GCC(A)
arithmetic functions: absolute/ . . . . . abs, sgn, gcd, lcm, min, max, labs – integer . . . . . ARITH(3) 337
alloc, free, balloc, bfree, gcalloc, gcfree − allocate memory . . . . . ALLOC(9.3) 682
ecvt, fcvt, gcvt − convert numbers to ascii . . . . . ECVT(3) 356
gebaca, gebam − get back at corporate america . . . . . GEBACA(A)
getarg, iargc – command arguments to Fortran . . . . . GETARG(3) 373
fputc, putw – character- or word-at-a-time/ . . . . . getc, getchar, fgetc, getw, putc, putchar, ... GETC(3) 374
getwd, getcwd – get current directory . . . . . GETWD(3) 384
getenv – value for environment name . . . . . GETENV(3) 375
getpgrp, setuid, setgid,/ . . . . . getuid, getgid, geteuid, getegid, getlogname, getpid, getppid, ... GETUID(2) 309
string into fields . . . . . getfields, getfields, setfields – break a . . . . . GETFIELDS(3) 376
getflags – process flag arguments in argv . . . . . GETFLAGS(3) 377
getfont − replace terminal’s default font . . . . . GETFONT(9.1) 643
string, defont, strwidth, infont, outfont, getfont − text and font operations . . . . . STRING(9.3) 690
endfsent – get file system description file/ . . . . . getfsent, getfsspec, getfsfile, setfsent, ... GETFSENT(3) 378
getppid, getpgrp, setuid, setgid,/ . . . . . getuid, getgid, geteuid, getegid, getlogname, getpid, ... GETUID(2) 309
endgrent – get group file entry . . . . . getgrent, getgrgid, getgrnam, setgrent, ... GETGRENT(3) 379
avoided . . . . . reboot, vadvise, vlimit, vswapon, getgroups, setgroups – system calls to be . . . . . DEPRECATED(2) 301
getlogin – get login name . . . . . GETLOGIN(3) 380
setgid,/ . . . . . getuid, getgid, geteuid, getegid, getlogname, getpid, getppid, getpgrp, setuid, ... GETUID(2) 309
fields . . . . . getfields, getmfields, setfields – break a string into . . . . . GETFIELDS(3) 376
/strinsert, strdelete, strzero, setmuxbuf, getmuxbuf, movstring − dynamic strings in mux . . . . . MUXSTRING(9.3) 689
tolayer, debug, getproc, getproctab, putname, getname − jerq process control . . . . . /newwindow, NEWPROC(9.2) 677
getopt − parse command options . . . . . GETOPT(A)
getopt – get option letter from argv . . . . . GETOPT(3) 381
getpass – read a password . . . . . GETPASS(3) 382
/geteuid, getegid, getlogname, getpid, getppid, getpgrp, setuid, setgid, setruid, setlogname,/ . . . . . GETUID(2) 309
/newproc, muxnewwind, newwindow, tolayer, debug, getproc, getproctab, putname, getname − jerq/ . . . . . NEWPROC(9.2) 677
endpwent, pwdecode – get password file/ . . . . . getpwent, getpwuid, getpwnam, setpwent, ... GETPWENT(3) 383
cursallow, cursinhibit, cursset, cursswitch, getrect123 − mouse control . . . . . button123, mouse, BUTTON(9.2) 675
fgets, puts, fputs, gets – string input/out on streams . . . . . FGETS(3) 362
getty − set terminal mode . . . . . GETTY(8) 575
getpid, getppid, getpgrp, setuid, setgid,/ . . . . . getuid, getgid, geteuid, getegid, getlogname, ... GETUID(2) 309
getuid, id – get user identity . . . . . GETUID(1) 145
character- or/ . . . . . getc, getchar, fgetc, getw, putc, putchar, fputc, putw − . . . . . GETC(3) 374
getwd, getcwd – get current directory . . . . . GETWD(3) 384
garbage collection . . . . . galloc, gfree, garbage – storage allocation with . . . . . GALLOC(3) 371
time to ASCII . . . . . ctime, localtime, gmtime, asctime, timezone – convert date and . . . . . CTIME(3) 349
gcc − GNU project C Compiler . . . . . GCC(A)
bison − GNU Project parser generator (yacc replacement) . . . . . BISON(A)
setjmp, longjmp – non-local goto . . . . . SETJMP(3) 424
grap – pic preprocessor for drawing graphs . . . . . GRAP(1) 146
graph − draw a graph . . . . . GRAPH(1) 148
graphs, convert to pic files . . . . . graphdraw graphpic − edit (combinatoric) . . . . . GRAPHDRAW(9.1) 644
picfile − raster graphic image format . . . . . PICFILE(5) 497
postbgi − PostScript translator for BGI (Basic Graphical Instructions) files . . . . . POSTBGI(A)
ltexture, upfront − layer control and graphics . . . . . /lbitblt, lpoint, lrectf, lsegment, NEWLAYER(9.2) 676
Pt, Rect, Rpt, display, Drect, Jrect − graphics data types . . . . . /Bitmap, Texture, TYPES(9.5) 699

pico – graphics editor . . . . . PICO(1) 200
plot – graphics filters . . . . . PLOT(1) 202
flicks, fflicks, preflicks, 2mux − movie graphics for 5620 . . . . . FLICKS(9.1) 642
point, rectf, screenswap, segment, texture − graphics functions . . . . . Code, addr, bitblt, BITBLT(9.3) 683
vec, move, etc. – plot graphics interface . . . . . PLOT(3) 403
plot − graphics interface . . . . . PLOT(5) 499
movies − graphics movie file formats . . . . . FLICKFILE(9.5) 696
rebecca − graphics touch-up editor . . . . . REBECCA(9.1) 663
to pic files . . . . . graphdraw graphpic − edit (combinatoric) graphs, convert . . . . . GRAPHDRAW(9.1) 644
dag − preprocessor for drawing directed graphs . . . . . DAG(1) 110
grap – pic preprocessor for drawing graphs . . . . . GRAP(1) 146
graphdraw graphpic − edit (combinatoric) graphs, convert to pic files . . . . . GRAPHDRAW(9.1) 644
/arithmetic functions: absolute value, sign, greatest common divisor, least common multiple,/ . . . . . ARITH(3) 337
pattern . . . . . gre, grep, egrep, fgrep – search a file for a . . . . . GRE(1) 149
imscan – scan greyscale images . . . . . IMSCAN(1) 155
chown − change owner or group . . . . . CHOWN(8) 566
passwd, group − password and group files . . . . . PASSWD(5) 496
getgrgid, getgrnam, setgrent, endgrent – get group file entry . . . . . getgrent, GETGRENT(3) 379
setlogname, setpgrp – get or set user, group, or process identity . . . . . /setgid, setruid, GETUID(2) 309
fchown, utime – change file mode, owner, group, or times . . . . . chmod, fchmod, chown, CHMOD(2) 300
hang – start a process in stopped state . . . . . HANG(1) 150
hangman, word_clout − word games . . . . . HANGMAN(A)
– run commands at low priority or immune to hangup . . . . . nice, nohup NICE(1) 191
xd, od – hex, octal, decimal, or ASCII dump . . . . . XD(1) 286
selection . . . . . menuhit, hmenuhit − present user with menu and get . . . . . MENUHIT(9.3) 687
hoc – interactive floating point language . . . . . HOC(1) 151
quot, findo − file system usage and hogs . . . . . QUOT(8) 599
bianchi − espresso, steamed milk, hot water . . . . . BIANCHI(A)
paper − list input on HP2621P printer . . . . . PAPER(A)
huff – huffman codebook/tree generator . . . . . HUFF(3) 385
sinh, cosh, tanh – hyperbolic functions . . . . . SINH(3) 426
hypot, cabs – euclidean distance . . . . . HYPOT(3) 386
getarg, iargc – command arguments to Fortran . . . . . GETARG(3) 373
laser-printer spooler . . . . . ican, ibcan, idcan, itcan − interface to Imagen . . . . . ICAN(A)
consistency check . . . . . icheck, dcheck, ncheck − file system . . . . . ICHECK(8) 576
iclc − Esterel binder . . . . . ICLC(A)
icon − icon editor . . . . . ICON(9.1) 646
ratfor, S, smp, sno,/ . . . . . altran, cospan, esterel, icon, lisp, macsyma, maple, ops5, pascal, ... LANGS(1) 159
face, mugs − show faces, make face icons from pictures . . . . . FACE(9.7) 701
compiler . . . . . icont, iconc − Icon language translator and . . . . . ICONT(A)
getuid, id – get user identity . . . . . GETUID(1) 145
laser-printer spooler . . . . . ican, ibcan, idcan, itcan − interface to Imagen . . . . . ICAN(A)
pictures . . . . . ideal – troff preprocessor for drawing . . . . . IDEAL(1) 152
tape, mt − identify and manipulate magnetic tape . . . . . TAPE(1) 254
uname − identify machine and kernel . . . . . UNAME(3) 440
getuid, id – get user identity . . . . . GETUID(1) 145
setpgrp – get or set user, group, or process identity . . . . . /setuid, setgid, setruid, setlogname, GETUID(2) 309
idiff – interactive file comparison . . . . . IDIFF(1) 154
− permit big core images, identify source of image . . . . . bigcore, coreid BIGCORE(1) 85
blitblt, menudrop − save or print a screen image . . . . . BLITBLT(9.1) 638
picfile − raster graphic image format . . . . . PICFILE(5) 497
ican, ibcan, idcan, itcan − interface to Imagen laser-printer spooler . . . . . ICAN(A)
imp − interactive mail program . . . . . IMP(A)
imscan – scan greyscale images . . . . . IMSCAN(1) 155
functions . . . . . in_host, in_ntoa, in_address, in_service − internet networking . . . . . INTERNET(3) 387
backup − incremental backup files . . . . . BACKUP(5) 483
ptx − permuted index . . . . . PTX(1) 210
operations . . . . . string, defont, strwidth, infont, outfont, getfont − text and font . . . . . STRING(9.3) 690
internet networking functions . . . . . in_host, in_ntoa, in_address, in_service − . . . . . INTERNET(3) 387
init − process control initialization . . . . . INIT(8) 578
and typesetting . . . . . tex, initex, virtex, dvips, dvit − text formatting . . . . . TEX(1) 261
networking functions . . . . . in_host, in_ntoa, in_address, in_service − internet . . . . . INTERNET(3) 387
filsys, flblk, ino − format of a disk file system . . . . . FILSYS(5) 489
clri − clear inode . . . . . CLRI(8) 568
istream − formatted and unformatted input . . . . . ISTREAM(A)
scanf, fscanf, sscanf – formatted input . . . . . SCANF(3) 421
ungetc – push character back into input stream . . . . . UNGETC(3) 441
fgets, puts, fputs, gets – string input/out on streams . . . . . FGETS(3) 362
filebuf − buffer for file input/output . . . . . FILEBUF(A)
Fflush, Ftie, Fclose, Fexit – fast buffered input/output . . . . . /Fundo, Fputc, Fprint, Fwrite, FIO(3) 363
fread, fwrite – buffered binary input/output . . . . . FREAD(3) 367
putw – character- or word-at-a-time stream input/output . . . . . /getw, putc, putchar, fputc, GETC(3) 374
iostream − buffering, formatting and input/output . . . . . IOS.INTRO(A)
ios − input/output formatting . . . . . IOS(A)

select − synchronous	input/output multiplexing	SELECT(2)	320
stdio − standard buffered	input/output package	STDIO(3)	428
kmc, kdiload, kmcdump − control KMC11	input/output processors	KMC(8)	580
sendchar, sendnchars, kill, exit − 5620	input/output requests /rcvchar, realtime,	REQUEST(9.2)	679
showq − status of stream	input/output system	SHOWQ(8)	613
in_host, in_ntoa, in_address,	in_service − internet networking functions	INTERNET(3)	387
rectXrect,/ add, sub, mul, div, eqpt, eqrect,	inset, muldiv, ptinrect, raddp, rsubp,	ADD(9.3)	681
automatic software distribution mkpkg,	inspkg, seal, unseal − package files for	MKPKG(8)	586
netstat, dkstat − show network status for	internet and datakit networks	NETSTAT(8)	590
in_host, in_ntoa, in_address, in_service −	internet networking functions	INTERNET(3)	387
ip, ip_ld − DARPA	internet protocol	IP(4)	456
ipconfig, dkipconfig, udpconfig − set up DARPA	Internet protocols	IPCONFIG(8)	579
ascii −	interpret ASCII characters	ASCII(1)	72
pipe − create an	interprocess channel	PIPE(2)	317
intro −	introduction to 5620-related software	INTRO(9)	635
intro −	introduction to commands	INTRO(1)	59
file systems intro −	introduction to devices, line disciplines, and	INTRO(4)	447
intro −	introduction to library functions	INTRO(3)	335
intro, errno −	introduction to system calls and error numbers	INTRO(2)	291
mux, ismux,	invert − layer multiplexer for 5620	MUX(9.1)	652
vis − show	invisible characters	VIS(1)	280
picunpack, picpack, picerror − picture file	I/O /picclose, picputprop, picgetprop,	PICFILE(3)	401
	ioctl − miscellaneous control operations	IOCTL(2)	310
	ios − input/output formatting	IOS(A)	
input/output	iostream − buffering, formatting and	IOS.INTRO(A)	
fstream −	iostream and streambuf specialized to files	FSTREAM(A)	
manipulators −	iostream out of band manipulations	MANIP(A)	
pipestream −	iostream specialized as circular buffer	PIPESTREAM(A)	
strstream −	iostream specialized to arrays	STRSTREAM(A)	
stdiobuf −	iostream specialized to stdio FILE	STDIOBUF(A)	
route, routed −	IP gateway routing	ROUTE(8)	605
	ip, ip_ld − DARPA internet protocol	IP(4)	456
ipclogin,/ ipccreat, ipcopen, ipclisten,	ipcaccept, ipcreject, ipcexec, ipcpath ,	IPC(3)	388
Internet protocols	ipconfig, dkipconfig, udpconfig − set up DARPA	IPCONFIG(8)	579
ipcexec, ipcpath , ipclogin,/ ipccreat,	ipcopen, ipclisten, ipcaccept, ipcreject,	IPC(3)	388
/ipcreject, ipcexec, ipcpath , ipclogin,	ipcrogin − set up connections between/	IPC(3)	388
ip,	ip_ld − DARPA internet protocol	IP(4)	456
isspace, ispunct, isprint, isgraph, iscntrl,	isascii − character classification /isalnum,	CTYPE(3)	350
terminal ttyname,	isatty, nametty − find or set name of a	TTYNAME(3)	438
isprint, isgraph,/ isalpha, isupper, islower,	isdigit, isxdigit, isalnum, isspace, ispunct,	CTYPE(3)	350
mux,	ismux, invert − layer multiplexer for 5620	MUX(9.1)	652
	istream − formatted and unformatted input	ISTREAM(A)	
isspace, ispunct, isprint, isgraph,/ isalpha,	isupper, islower, isdigit, isxdigit, isalnum,	CTYPE(3)	350
spooler ican, ibcan, idcan,	itcan − interface to Imagen laser-printer	ICAN(A)	
msqrt, mgcd, min, mout, fmin, fmout, move,/	itom, mfree, madd, msub, mult, mdiv, sdiv,	MP(3)	398
font −	jerq font layouts	FONT(9.5)	697
mouse −	jerq mouse user interface	MOUSE(9.4)	693
debug, getproc, getproctab, putname, getname −	jerq process control /newwindow, tolayer,	NEWPROC(9.2)	677
	jf − font editor	JF(9.1)	647
	jim, jim.recover − text editor	JIM(A)	
	jioctl − mux ioctl requests	JIOCTL(9.4)	692
uustat − uucp status inquiry and	job control	UUSTAT(1)	273
	join − relational database operator	JOIN(1)	156
windows,	jps, reshape − create and initialize windows	WINDOWS(9.1)	674
Bitmap, Texture, Pt, Rect, Rpt, display, Drect,	Jrect − graphics data types /Point, Rectangle,	TYPES(9.5)	699
worm,	jukebox − optical disk utilities	WORM(8)	629
	jx − 5620 execution and stdio interpreter	JX(9.1)	648
request, own, wait, alarm, sleep, nap,	kbdchar, rcvchar, realtime, sendchar,/	REQUEST(9.2)	679
processors kmc,	kdiload, kmcdump − control KMC11 input/output	KMC(8)	580
uname − identify machine and	kernel	UNAME(3)	440
makekey − generate encryption	key	MAKEKEY(8)	583
execution uucico, uusched, uuxqt,	kick, debug − uucp file transport and	UUCICO(8)	625
/rcvchar, realtime, sendchar, sendnchars,	kill, exit − 5620 input/output requests	REQUEST(9.2)	679
signal,	kill − receive and send signals	SIGNAL(2)	321
prejudice	kill − terminate a process with extreme	KILL(1)	157
input/output processors	kmc, kdiload, kmcdump − control KMC11	KMC(8)	580
processor registers mem,	kmem, kUmem, mtpr, fineclock − memory and VAX	MEM(4)	458
integers and long integers	l3tol, ltol3 − convert between 3-byte	L3TOL(3)	391
	lab − label maker	LAB(1)	158
absolute value,/ abs, sgn, gcd, lcm, min, max,	labs − integer arithmetic functions:	ARITH(3)	337
awk − pattern-directed scanning and processing	language	AWK(1)	74
bc − arbitrary-precision arithmetic	language	BC(1)	81
hoc − interactive floating point	language	HOC(1)	151
rc, cd, wait, whatis − command	language	RC(1)	215

− shell, the standard command programming	language . . . . . sh, cd, wait, whatis	SH(1)	230
twig − tree-manipulation	language	TWIG(A)	
pc − pascal	language compiler	PC(A)	
spitbol − Snobol	language compiler	SPITBOL(A)	
altran −	language for algebraic manipulation	ALTRAN(A)	
pascal −	language interpreter	PASCAL(A)	
sno − Snobol	language interpreter	SNO(A)	
cpp − C	language preprocessor	CPP(8)	569
efl − extended Fortran	language preprocessor	EFL(A)	
icont, iconc − Icon	language translator and compiler	ICONT(A)	
ratfor, S, smp, sno, spitbol, struct, twig −	languages . . . . . /lisp, macsyma, maple, ops5, pascal,	LANGS(1)	159
bibliographies	latex, bibtex − tex macro package and	LATEX(6)	522
mux, ismux, invert −	layer multiplexer for 5620	MUX(9.1)	652
upfront − layer/ . . . . newlayer, dellayer, downback,	lbitblt, lpoint, lrectf, lsegment, ltexture, ...	NEWLAYER(9.2)	676
ls,	lc − list contents of directory	LS(1)	171
cc,	lcc − C compilers	CC(1)	92
functions: absolute value,/ . . . . . abs, sgn, gcd,	lcm, min, max, labs − integer arithmetic	ARITH(3)	337
	lcoc − Esterel compiler	LCOC(A)	
	lcomp, lprint − line-by-line profiler	LCOMP(1)	161
	ld − link editor or loader	LD(1)	162
exponent . . . . . frexp,	ldexp, modf − split into mantissa and	FREXP(3)	368
	ldpcs − load comet microcode	LDPCS(8)	581
/absolute value, sign, greatest common divisor,	least common multiple, minimum, maximum	ARITH(3)	337
	lens − bitmap magnifier	LENS(9.1)	649
monkspell, monkmerge − typeset documents and	letters . . . . . monk, monksample,	MONK(1)	188
	lex − generator of lexical analysis programs	LEX(1)	164
	libc − standard library functions	LIBC(9.3)	686
docsubmit − send document to	library	DOCSUBMIT(1)	123
− find ordering relation for an object	library . . . . . lorder	LORDER(1)	169
sendcover − send cover sheet to the	library	SENDCOVER(8)	611
	library, bellcat − bell labs library services	LIBRARY(7)	541
ar − archive (	library) file format	AR(5)	482
port − mathematical	library for Fortran	PORT(3)	408
intro − introduction to	library functions	INTRO(3)	335
libc − standard	library functions	LIBC(9.3)	686
ar, ranlib − archive and	library maintainer	AR(1)	70
buf_ld − buffering	line discipline	BUFLD(4)	448
tty_ld − terminal processing	line discipline	TTYLD(4)	473
conn_ld −	line discipline for unique stream connection	CONNLD(4)	449
mesg_ld, rmesg_ld − message	line discipline modules	MESGLD(4)	459
intro − introduction to devices,	line disciplines, and file systems	INTRO(4)	447
tty − serial	line interface drivers	TTY(4)	472
rev, revpag − reverse	lines or pages	REV(1)	224
ld −	link editor or loader	LD(1)	162
cp, mv, ln, reccp − copy, move, or	link files	CP(1)	103
	link, symlink, readlink − link to a file	LINK(2)	311
	lint, cyntax, cem − C program verifiers	LINT(1)	165
compiler	lisp, liszt, lxref − lisp interpreter and	LISP(A)	
smp, sno,/ . . . . . altran, cospan, esterel, icon,	lisp, macsyma, maple, ops5, pascal, ratfor, S, ...	LANGS(1)	159
ls, lc −	list contents of directory	LS(1)	171
smstat −	list smtp queues	SMSTAT(8)	615
nm − name	list (symbol table)	NM(1)	192
lisp,	liszt, lxref − lisp interpreter and compiler	LISP(A)	
cp, mv,	ln, reccp − copy, move, or link files	CP(1)	103
ldpcs −	load comet microcode	LDPCS(8)	581
	load − load statistics	LOAD(1)	167
ld − link editor or	loader	LD(1)	162
32ld − bootstrap	loader for 5620	32LD(9.1)	636
convert date and time to ASCII . . . . . ctime,	localtime, gmtime, asctime, timezone −	CTIME(3)	349
logarithm, power, square root . . . . . exp,	log, log10, pow, sqrt − exponential, ...	EXP(3)	360
	login − sign on	LOGIN(8)	582
ac −	login accounting	AC(8)	562
con, rx − remote	login and execution	CON(1)	102
ndcon, rlogin, nrx, rsh, scriptcon − remote	login and execution . . . . . dcon,	DCON(1)	115
getlogin − get	login name	GETLOGIN(3)	380
utmp, wtmp −	login records	UTMP(5)	511
tty,	logtty − get terminal name	TTY(1)	268
setjmp,	longjmp − non-local goto	SETJMP(3)	424
	look − find lines in a sorted list	LOOK(1)	168
dict −	look up words in English dictionaries	DICT(7)	538
bibliographic references . . . . . refer,	lookbib, pubindex − maintain and use	REFER(1)	222
object library	lorder − find ordering relation for an	LORDER(1)	169
tolower, toupper − force upper or	lower case	TOLOWER(3)	437
	lp − printer output	LP(1)	170

layer/ . . . . . newlayer, dellayer, downback, lbitblt, lpoint, lrectf, lsegment, ltexture, upfront − . . . . . NEWLAYER(9.2) 676
lcomp, lprint – line-by-line profiler . . . . . LCOMP(1) 161
generator . . . . . rand, lrand, frand, nrand, srand – random number . . . . . RAND(3) 417
newlayer, dellayer, downback, lbitblt, lpoint, lrectf, lsegment, ltexture, upfront − layer/ . . . . . NEWLAYER(9.2) 676
ls, lc – list contents of directory . . . . . LS(1) 171
lseek – seek, move read/write pointer . . . . . LSEEK(2) 312
stat, lstat, fstat – get file status . . . . . STAT(2) 323
/downback, lbitblt, lpoint, lrectf, lsegment, ltexture, upfront − layer control and graphics . . . . . NEWLAYER(9.2) 676
long integers . . . . . l3tol, ltol3 – convert between 3-byte integers and . . . . . L3TOL(3) 391
occ − Esterel and Lustre C code producer . . . . . OCC(A)
ocdebug − Esterel and Lustre Debug code producer . . . . . OCDEBUG(A)
lisp, liszt, lxref − lisp interpreter and compiler . . . . . LISP(A)
m4 – macro processor . . . . . M4(1) 173
3cc, 3as, 3ar, 3ld, 3nm, 3size, 3strip, cprs − MAC-32 C compiler . . . . . 3CC(9.1) 637
uname − identify machine and kernel . . . . . UNAME(3) 440
pwd, where – machine name and working directory . . . . . PWD(1) 212
latex, bibtex − tex macro package and bibliographies . . . . . LATEX(6) 522
m4 – macro processor . . . . . M4(1) 173
mpictures − picture inclusion macros . . . . . MPICTURES(6) 528
mcs − macros for formatting cover sheets . . . . . MCS(6) 527
ms − macros for formatting manuscripts . . . . . MS(6) 530
mpm − macros for page makeup . . . . . MPM(6) 529
mbits − macros to typeset bitmaps . . . . . MBITS(6) 526
man – macros to typeset manual . . . . . MAN(6) 524
sno,/ . . . . . altran, cospan, esterel, icon, lisp, macsyma, maple, ops5, pascal, ratfor, S, smp, ... LANGS(1) 159
mout, fmin, fmout, move, mcmp,/ . . . . itom, mfree, madd, msub, mult, mdiv, sdiv, msqrt, mgcd, min, ... MP(3) 398
tape, mt − identify and manipulate magnetic tape . . . . . TAPE(1) 254
lens − bitmap magnifier . . . . . LENS(9.1) 649
mt − magtape interface . . . . . MT(4) 460
mail − mail addresses . . . . . MAIL(6) 523
Mail − send and receive mail . . . . . MAIL(A)
upas, rmail, translate − mail delivery system . . . . . UPAS(8) 623
mail – send or receive mail . . . . . MAIL(1) 176
vismon, sysmon, vwhois − system statistics and mail notification . . . . . VISMON(9.1) 673
imp − interactive mail program . . . . . IMP(A)
smtpqer, smtpd, smtpsched − handle simple mail transfer protocol . . . . . smtp, SMTP(8) 616
prefer, pinvert, penter, plook, pconvert − maintain and use bibliographic references . . . . . PREFER(1) 204
refer, lookbib, pubindex − maintain and use bibliographic references . . . . . REFER(1) 222
make – maintain collections of programs . . . . . MAKE(1) 178
mk, mkconv, membername − maintain (make) related files . . . . . MK(1) 183
programs . . . . . usgmake − maintain, update, and regenerate groups of . . . . . USGMAKE(A)
ar, ranlib − archive and library maintainer . . . . . AR(1) 70
mkdir – make a directory . . . . . MKDIR(1) 187
mknod – make a directory or a special file . . . . . MKNOD(2) 314
face, mugs − show faces, make face icons from pictures . . . . . FACE(9.7) 701
make – maintain collections of programs . . . . . MAKE(1) 178
mkdir, rmdir – make or remove a directory . . . . . MKDIR(2) 313
mk, mkconv, membername − maintain (make) related files . . . . . MK(1) 183
makekey − generate encryption key . . . . . MAKEKEY(8) 583
authored papers . . . . . findauthor, papers, makepaper − consult database of locally . . . . . PAPERS(7) 546
mpm − macros for page makeup . . . . . MPM(6) 529
allocator . . . . . malloc, free, realloc, calloc, cfree − memory . . . . . MALLOC(3) 392
man – macros to typeset manual . . . . . MAN(6) 524
man – print pages of this manual . . . . . MAN(1) 181
manipulations . . . . . manipulators − iostream out of band . . . . . MANIP(A)
frexp, ldexp, modf – split into mantissa and exponent . . . . . FREXP(3) 368
man – print pages of this manual . . . . . MAN(1) 181
man – macros to typeset manual . . . . . MAN(6) 524
ms − macros for formatting manuscripts . . . . . MS(6) 530
Map − associative array classes . . . . . MAP(3+) 393
map − digitized map formats . . . . . MAP(5) 494
map − draw maps on various projections . . . . . MAP(7) 542
ascii − map of ASCII character set . . . . . ASCII(6) 519
orient, normalize − map projections . . . . . PROJ(3) 413
maple − interactive symbolic algebraic program . . . . . MAPLE(A)
altran, cospan, esterel, icon, lisp, macsyma, maple, ops5, pascal, ratfor, S, smp, sno,/ . . . . . LANGS(1) 159
mint − produce usage report from a maple program . . . . . MINT(A)
map − draw maps on various projections . . . . . MAP(7) 542
mars − memory array redcode simulator . . . . . MARS(A)
redcode − assembler for mars game . . . . . REDCODE(A)
umask – set file creation mode mask . . . . . UMASK(2) 330
cos, sin, atan2, sqrt, norm − integer math functions . . . . . COS(9.3) 685
port – mathematical library for Fortran . . . . . PORT(3) 408
visi − mathematical spreadsheet . . . . . VISI(A)

matlab – interactive matrix desk calculator . . . . . MATLAB(1) 182
absolute value, sign,/ . . . . abs, sgn, gcd, lcm, min, max, labs – integer arithmetic functions: . . . . . ARITH(3) 337
mbits – macros to typeset bitmaps . . . . . MBITS(6) 526
col, 2, 3, 4, 5, 6, mc, fold, expand – column alignment . . . . . COLUMN(1) 100
mcc – MC68000 C compiler . . . . . MCC(A)
msqrt, mgcd, min, mout, fmin, fmout, move, mcmp, rpow, mpow – multiple precision/ . . . . . /sdiv, MP(3) 398
mcs – macros for formatting cover sheets . . . . . MCS(6) 527
fmout, move,/ . . . . itom, mfree, madd, msub, mult, mdiv, sdiv, msqrt, mgcd, min, mout, fmin, ... MP(3) 398
ruler – measure things on the screen . . . . . RULER(9.1) 664
VAX processor registers . . . . . mem, kmem, kUmem, mtpr, fineclock – memory and . . . MEM(4) 458
mk, mkconv, membername – maintain (make) related files . . . . . MK(1) 183
memset – memory operations . . . . . memccpy, memchr, memcmp, memcpy, memmove, ... MEMORY(3) 395
festoon – memo writer . . . . . FESTOON(A)
balloc, bfree, gcalloc, gcfree – allocate memory . . . . . alloc, free, ALLOC(9.3) 682
pool – fast memory allocation . . . . . POOL(3+) 406
malloc, free, realloc, calloc, cfree – memory allocator . . . . . MALLOC(3) 392
mem, kmem, kUmem, mtpr, fineclock – memory and VAX processor registers . . . . . MEM(4) 458
core – format of memory image file . . . . . CORE(5) 485
memccpy, memchr, memcmp, memcpy, memmove, memset – memory operations . . . . . MEMORY(3) 395
blitblt, menudrop – save or print a screen image . . . . . BLITBLT(9.1) 638
get selection . . . . . menuhit, hmenuhit – present user with menu and . . . . . MENUHIT(9.3) 687
sort – sort and/or merge files . . . . . SORT(1) 242
messages . . . . . write, mesg – write to other users, allow or forbid . . . . . WRITE(1) 283
modules . . . . . mesg_ld, rmesg_ld – message line discipline . . . . . MESGLD(4) 459
mgcd, min, mout, fmin, fmout, move,/ . . . . . itom, mfree, madd, msub, mult, mdiv, sdiv, msqrt, ... MP(3) 398
ldpcs – load comet microcode . . . . . LDPCS(8) 581
bianchi – espresso, steamed milk, hot water . . . . . BIANCHI(A)
mille – card game . . . . . MILLE(A)
/madd, msub, mult, mdiv, sdiv, msqrt, mgcd, min, mout, fmin, fmout, move, mcmp, rpow, mpow/ . . . MP(3) 398
greatest common divisor, least common multiple, minimum, maximum . . . . . /absolute value, sign, ARITH(3) 337
program . . . . . mint – produce usage report from a maple . . . . . MINT(A)
system . . . . . mkfs, mkbitfs, mklost+found – construct a disk file . . . . . MKFS(8) 584
files . . . . . mk, mkconv, membername – maintain (make) related . . . . . MK(1) 183
mkdir – make a directory . . . . . MKDIR(1) 187
mkdir, rmdir – make or remove a directory . . . . . MKDIR(2) 313
file system . . . . . mkfs, mkbitfs, mklost+found – construct a disk . . . . . MKFS(8) 584
mknod – construct special file . . . . . MKNOD(8) 585
mknod – make a directory or a special file . . . . . MKNOD(2) 314
for automatic software distribution . . . . . mkpkg, inspkg, seal, unseal – package files . . . . . MKPKG(8) 586
mktemp, tmpnam – make a unique file name . . . . . MKTEMP(3) 396
sml – Standard ML compiler . . . . . SML(1) 239
chmod – change mode . . . . . CHMOD(1) 96
getty – set terminal mode . . . . . GETTY(8) 575
umask – set file creation mode mask . . . . . UMASK(2) 330
fchmod, chown, fchown, utime – change file mode, owner, group, or times . . . . . chmod, CHMOD(2) 300
tset – set terminal modes . . . . . TSET(A)
frexp, ldexp, modf – split into mantissa and exponent . . . . . FREXP(3) 368
touch, chdate – set modification or access date of a file . . . . . CHDATE(1) 95
monitor – prepare execution profile . . . . . MONITOR(3) 397
typeset documents and letters . . . . . monk, monksample, monkspell, monkmerge – . . . . . MONK(1) 188
p, pg, more – paginate . . . . . P(1) 194
bcd, ppt, morse – convert to antique media . . . . . BCD(A)
fmount, funmount – mount or remove file system . . . . . FMOUNT(2) 307
mount, umount – mount and dismount file system . . . . . MOUNT(8) 587
mouse – jerq mouse user interface . . . . . MOUSE(9.4) 693
cursswitch, getrect123 – mouse/ . . . . . button123, mouse, cursallow, cursinhibit, cursset, ... BUTTON(9.2) 675
vec, move, etc. – plot graphics interface . . . . . PLOT(3) 403
/sdiv, msqrt, mgcd, min, mout, fmin, fmout, move, mcmp, rpow, mpow – multiple precision/ . . . . . MP(3) 398
cp, mv, ln, reccp – copy, move, or link files . . . . . CP(1) 103
lseek – seek, move read/write pointer . . . . . LSEEK(2) 312
movies – graphics movie file formats . . . . . FLICKFILE(9.5) 696
flicks, fflicks, preflicks, 2mux – movie graphics for 5620 . . . . . FLICKS(9.1) 642
view2d, regrid, vdata – movie of a function f(x, y, t) . . . . . VIEW2D(1) 279
view2d – movie of a function f(x, y, t) . . . . . VIEW2D(5) 512
movie, stills – algorithm animation . . . . . MOVIE(9.1) 650
view2d, moviefil – movie of a function f(x, y, t) . . . . . VIEW2D(3) 443
movies – graphics movie file formats . . . . . FLICKFILE(9.5) 696
strdelete, strzero, setmuxbuf, getmuxbuf, movstring – dynamic strings in mux . . . . . /strinsert, MUXSTRING(9.3) 689
mpictures – picture inclusion macros . . . . . MPICTURES(6) 528
mpm – macros for page makeup . . . . . MPM(6) 529
/mgcd, min, mout, fmin, fmout, move, mcmp, rpow, mpow – multiple precision integer arithmetic . . . . . MP(3) 398
ms – macros for formatting manuscripts . . . . . MS(6) 530
rarepl, rarct – replace bad blocks on MSCP disks . . . . . RAREPL(8) 600
ra – DEC MSCP disks (RA60, RA80, RA81, RA90) . . . . . RA(4) 465

itom, mfree, madd, msub, mult, mdiv, sdiv, msqrt, mgcd, min, mout, fmin, fmout, move,/ . . . . . MP(3) 398
tape, mt − identify and manipulate magnetic tape . . . . . TAPE(1) 254
mt − magtape interface . . . . . MT(4) 460
fstab, mtab − information about file systems . . . . . FSTAB(5) 493
registers . . . . . mem, kmem, kUmem, mtpr, fineclock − memory and VAX processor . . . . . MEM(4) 458
pictures . . . . . face, mugs − show faces, make face icons from . . . . . FACE(9.7) 701
ptinrect, raddp, rsubp, rectXrect,/ . . . . . add, sub, mul, div, eqpt, eqrect, inset, muldiv, ... ADD(9.3) 681
fmout, move, mcmp,/ . . . itom, mfree, madd, msub, mult, mdiv, sdiv, msqrt, mgcd, min, mout, fmin, ... MP(3) 398
mux, ismux, invert − layer multiplexer for 5620 . . . . . MUX(9.1) 652
select − synchronous input/output multiplexing . . . . . SELECT(2) 320
jioctl − mux ioctl requests . . . . . JIOCTL(9.4) 692
5620 . . . . . mux, ismux, invert − layer multiplexer for . . . . . MUX(9.1) 652
term − nonstandard mux terminals . . . . . TERM(9.1) 671
getproctab, putname, getname −/ . . . . . P, newproc, muxnewwind, newwindow, tolayer, debug, getproc, ... NEWPROC(9.2) 677
cp, mv, ln, reccp − copy, move, or link files . . . . . CP(1) 103
getlogin − get login name . . . . . GETLOGIN(3) 380
nlist − get entries from name list . . . . . NLIST(3) 399
nm − name list (symbol table) . . . . . NM(1) 192
qns − query name server . . . . . QNS(7) 549
ns − name server database . . . . . NS(8) 591
ttyname, isatty, nametty − find or set name of a terminal . . . . . TTYNAME(3) 438
sendnchars,/ . . . . . request, own, wait, alarm, sleep, nap, kbdchar, rcvchar, realtime, sendchar, ... REQUEST(9.2) 679
alarm, nap, pause − schedule timing delays . . . . . ALARM(2) 297
dist, dme, plan, path, cross − aviation navigation . . . . . AV(A)
icheck, dcheck, ncheck − file system consistency check . . . . . ICHECK(8) 576
login and execution . . . . . dcon, ndcon, rlogin, nrx, rsh, scriptcon − remote . . . . . DCON(1) 115
eqn, neqn, checkeq − typeset mathematics . . . . . EQN(1) 132
netfs − network file system . . . . . NETFS(8) 588
netnews − usenet news articles, utility files . . . . . NETNEWS(5) 495
postnews − submit netnews articles . . . . . POSTNEWS(7) 548
checknews, readnews − read netnews articles . . . . . READNEWS(7) 550
netnews − send or receive news articles . . . . . NETNEWS(A)
internet and datakit networks . . . . . netstat, dkstat − show network status for . . . . . NETSTAT(8) 590
faced − network face server . . . . . FACED(9.5) 695
netfs − network file system . . . . . NETFS(8) 588
networks . . . . . netstat, dkstat − show network status for internet and datakit . . . . . NETSTAT(8) 590
in_ntoa, in_address, in_service − internet networking functions . . . . . in_host, INTERNET(3) 387
tcp_listen, tcp_accept, tcp_rcmd − tcp networking functions . . . . . tcp_sock, tcp_connect, TCP(3) 433
udp_connect, udp_listen, udp_datagram − udp networking functions . . . . . UDP(3) 439
newgrp − change to a new group . . . . . NEWGRP(1) 190
fork − spawn new process . . . . . FORK(2) 308
test, [, newer − condition commands . . . . . TEST(1) 259
newgrp − change to a new group . . . . . NEWGRP(1) 190
lrectf, lsegment, ltexture, upfront − layer/ . . . . . newlayer, dellayer, downback, lbitblt, lpoint, ... NEWLAYER(9.2) 676
getproc, getproctab, putname, getname −/ . . . . . P, newproc, muxnewwind, newwindow, tolayer, debug, . . . NEWPROC(9.2) 677
netnews − send or receive news articles . . . . . NETNEWS(A)
netnews − usenet news articles, utility files . . . . . NETNEWS(5) 495
news − print news items . . . . . NEWS(7) 545
putname, getname −/ . . . P, newproc, muxnewwind, newwindow, tolayer, debug, getproc, getproctab, ... NEWPROC(9.2) 677
dbminit, fetch, store, delete, firstkey, nextkey − database subroutines . . . . . DBM(3) 353
nice − set program priority . . . . . NICE(2) 315
or immune to hangup . . . . . nice, nohup − run commands at low priority . . . . . NICE(1) 191
nlist − get entries from name list . . . . . NLIST(3) 399
nm − name list (symbol table) . . . . . NM(1) 192
immune to hangup . . . . . nice, nohup − run commands at low priority or . . . . . NICE(1) 191
setjmp, longjmp − non-local goto . . . . . SETJMP(3) 424
cos, sin, atan2, sqrt, norm − integer math functions . . . . . COS(9.3) 685
orient, normalize − map projections . . . . . PROJ(3) 413
push, pull, npush, npull − datakit remote file copy . . . . . PUSH(1) 211
rand, lrand, frand, nrand, srand − random number generator . . . . . RAND(3) 417
troff, nroff − text formatting and typesetting . . . . . TROFF(1) 265
soelim − eliminate .so’s from nroff input . . . . . SOELIM(A)
tbl − format tables for nroff or troff . . . . . TBL(1) 256
execution . . . . . dcon, ndcon, rlogin, nrx, rsh, scriptcon − remote login and . . . . . DCON(1) 115
ns − name server database . . . . . NS(8) 591
null − data sink . . . . . NULL(4) 462
arithmetic − drill in number facts . . . . . ARITHMETIC(A)
number − convert Arabic numerals to English . . . . . NUMBER(A)
size − size of an object file . . . . . SIZE(1) 237
a.out − object file format . . . . . A.OUT(5) 479
lorder − find ordering relation for an object library . . . . . LORDER(1) 169
occ − Esterel and Lustre C code producer . . . . . OCC(A)
producer . . . . . ocdebug − Esterel and Lustre Debug code . . . . . OCDEBUG(A)
ocr − optical character recognition . . . . . OCR(1) 193

xd, od – hex,	octal, decimal, or ASCII dump	XD(1)	286
	ogre − war game	OGRE(A)	
exit,	onexit – terminate process	EXIT(3)	359
popen, ppopen, vepopen, pclose –	open a pipe to/from a process	POPEN(3)	407
fopen, freopen, fdopen, fclose, fflush –	open, close, or flush a stream	FOPEN(3)	366
or writing, create file	open, creat, close – open a file for reading	OPEN(2)	316
dup, dup2 – duplicate an	open file descriptor	DUP(2)	303
– directory operations	opendir, readdir, telldir, seekdir, closedir	DIRECTORY(3)	355
environment	ops5 − a rule-based production-system	OPS5(A)	
/cospan, esterel, icon, lisp, macsyma, maple,	ops5, pascal, ratfor, S, smp, sno, spitbol,/	LANGS(1)	159
ocr −	optical character recognition	OCR(1)	193
worm, jukebox −	optical disk utilities	WORM(8)	629
getopt – get	option letter from argv	GETOPT(3)	381
getopt − parse command	options	GETOPT(A)	
lorder – find	ordering relation for an object library	LORDER(1)	169
	orient, normalize − map projections	PROJ(3)	413
	ostream − formatted and unformatted output	OSTREAM(A)	
string, defont, strwidth, infont,	outfont, getfont − text and font operations	STRING(9.3)	690
lp − printer	output	LP(1)	170
ostream − formatted and unformatted	output	OSTREAM(A)	
fprint, sprint, fmtinstall – print formatted	output print,	PRINT(3)	409
fprintf, sprintf, snprintf – print formatted	output printf,	PRINTF(3)	411
realtime, sendchar, sendnchars, kill,/ request,	own, wait, alarm, sleep, nap, kbdchar, rcvchar,	REQUEST(9.2)	679
chown, fchown, utime – change file mode,	owner, group, or times chmod, fchmod,	CHMOD(2)	300
chown − change	owner or group	CHOWN(8)	566
debug, getproc, getproctab, putname, getname/	P, newproc, muxnewwind, newwindow, tolayer,	NEWPROC(9.2)	677
	p, pg, more – paginate	P(1)	194
– compress and expand files	pack, unpack, pcat, compress, uncompress, zcat	PACK(1)	195
distribution mkpkg, inspkg, seal, unseal −	package files for automatic software	MKPKG(8)	586
	pads − user interface package	PADS(9.5)	698
mpm − macros for	page makeup	MPM(6)	529
rscan, pix − scan	page on ricoh scanner and display on 5620	RSCAN(A)	
postreverse − reverse the	page order in a postscript file	POSTREVERSE(A)	
rev, revpag − reverse lines or	pages	REV(1)	224
p, pg, more –	paginate	P(1)	194
tk −	paginator for the Tektronix 4014	TK(A)	
drum −	paging device	DRUM(4)	453
vtimes − get usage of time, space, and	paging resources	VTIMES(2)	332
	paint − draw pictures in a layer	PAINT(9.1)	654
brush −	painting program	BRUSH(9.1)	639
	paper − list input on HP2621P printer	PAPER(A)	
locally authored papers findauthor,	papers, makepaper − consult database of	PAPERS(7)	546
bison − GNU Project	parser generator (yacc replacement)	BISON(A)	
pc −	pascal language compiler	PC(A)	
	pascal – language interpreter	PASCAL(A)	
lister pxp, pxref −	pascal printer, profiler, and cross-reference	PXP(A)	
/esterel, icon, lisp, macsyma, maple, ops5,	pascal, ratfor, S, smp, sno, spitbol, struct,/	LANGS(1)	159
	passwd, group − password and group files	PASSWD(5)	496
	passwd – change login password	PASSWD(1)	196
getpass – read a	password	GETPASS(3)	382
passwd, group −	password and group files	PASSWD(5)	496
getpwnam, setpwent, endpwent, pwdecode – get	password file entry getpwent, getpwuid,	GETPWENT(3)	383
cut,	paste − rearrange columns of data	CUT(1)	109
an original	patch − a program for applying a diff file to	PATCH(A)	
dist, dme, plan,	path, cross – aviation navigation	AV(A)	
grep, egrep, fgrep – search a file for a	pattern gre,	GRE(1)	149
re_bm, re_cw, re_re – string and	pattern matching	RE(3)	418
language awk −	pattern-directed scanning and processing	AWK(1)	74
alarm, nap,	pause – schedule timing delays	ALARM(2)	297
	pc − pascal language compiler	PC(A)	
and expand files pack, unpack,	pcat, compress, uncompress, zcat – compress	PACK(1)	195
popen, ppopen, vepopen,	pclose – open a pipe to/from a process	POPEN(3)	407
references prefer, pinvert, penter, plook,	pconvert − maintain and use bibliographic	PREFER(1)	204
11nm, 11ranlib, 11reloc, 11size, 11strip −	pdp11 support 11as, 11cc, 11ld,	11(8)	561
	ped, tped − picture editor	PED(9.1)	656
twid,	pen − doodle on the screen	TWID(A)	
	pengo − squash the sno-bees	PENGO(A)	
bibliographic references prefer, pinvert,	penter, plook, pconvert − maintain and use	PREFER(1)	204
ptx −	permuted index	PTX(1)	210
messages	perror, sys_errlist, sys_nerr – system error	PERROR(3)	400
p,	pg, more – paginate	P(1)	194
tel, telno, dq −	phone books	TEL(7)	554
apsend − send troff output to	phototypesetter	APSEND(1)	69
	pi, 3pi − process inspector	PI(9.1)	658

− edit (combinatoric) graphs, convert to — pic files . . . graphdraw graphpic — GRAPHDRAW(9.1) — 644
grap − — pic preprocessor for drawing graphs . . . — GRAP(1) — 146
drawing pictures . . . — pic, tpic − troff and tex preprocessors for . . . — PIC(1) — 197
picputprop, picgetprop, picunpack, picpack, — picerror − picture file I/O . . . /picclose, — PICFILE(3) — 401
— picfile − raster graphic image format . . . — PICFILE(5) — 497
arguments . . . apply, — pick − repeatedly apply a command; select . . . — APPLY(1) — 68
— pico − graphics editor . . . — PICO(1) — 200
/picclose, picputprop, picgetprop, picunpack, — picpack, picerror − picture file I/O . . . — PICFILE(3) — 401
picgetprop, picunpack,/ . . . . picopen_r, picopen_w, — picread, picwrite, picclose, picputprop, . . . — PICFILE(3) — 401
ped, tped − — picture editor . . . — PED(9.1) — 656
bcp − reformat black-and-white — picture files . . . — BCP(1) — 83
mpictures − — picture inclusion macros . . . — MPICTURES(6) — 528
face, mugs − show faces, make face icons from — pictures . . . — FACE(9.7) — 701
ideal − troff preprocessor for drawing — pictures . . . — IDEAL(1) — 152
− troff and tex preprocessors for drawing — pictures . . . pic, tpic — PIC(1) — 197
cip − draw — pictures for typesetting . . . — CIP(9.1) — 641
paint − draw — pictures in a layer . . . — PAINT(9.1) — 654
/picwrite, picclose, picputprop, picgetprop, — picunpack, picpack, picerror − picture file/ . . . — PICFILE(3) — 401
and use bibliographic references . . . prefer, — pinvert, penter, plook, pconvert − maintain . . . — PREFER(1) — 204
tee − — pipe fitting . . . — TEE(1) — 258
— pipe − create an interprocess channel . . . — PIPE(2) — 317
popen, ppopen, vepopen, pclose − open a — pipe to/from a process . . . — POPEN(3) — 407
queue . . . — pipebuf − streambuf specialized as circular . . . — PIPEBUF(A)
buffer . . . — pipestream − iostream specialized as circular . . . — PIPESTREAM(A)
on 5620 . . . rscan, — pix − scan page on ricoh scanner and display . . . — RSCAN(A)
dist, dme, — plan, path, cross − aviation navigation . . . — AV(A)
games, demo − some — playthings . . . — GAMES(1) — 143
bibliographic/ . . . prefer, pinvert, penter, — plook, pconvert − maintain and use . . . — PREFER(1) — 204
— plot − graphics interface . . . — PLOT(5) — 499
vec, move, etc. − — plot graphics interface . . . — PLOT(3) — 403
— plot − graphics filters . . . — PLOT(1) — 202
Rpt, display, Drect, Jrect − graphics/ . . . . . Word, — Point, Rectangle, Bitmap, Texture, Pt, Rect, . . . — TYPES(9.5) — 699
graphics functions . . . Code, addr, bitblt, — point, rectf, screenswap, segment, texture − . . . — BITBLT(9.3) — 683
rsubp, rectXrect, rectclip − arithmetic on — points and rectangles . . . /muldiv, ptinrect, raddp, — ADD(9.3) — 681
— poly − polyhedron database . . . — POLY(5) — 502
— polypic, polypr − database of polyhedra . . . — POLY(7) — 547
poly_lk, — poly_read − polyhedron database routines . . . — POLY(3) — 405
— pool − fast memory allocation . . . — POOL(3+) — 406
to/from a process . . . — popen, ppopen, vepopen, pclose − open a pipe . . . — POPEN(3) — 407
— port − mathematical library for Fortran . . . — PORT(3) — 408
Graphical Instructions) files . . . — postbgi − PostScript translator for BGI (Basic . . . — POSTBGI(A)
produce postscript . . . dpost, — postdaisy, postdmd, postprint − filters to . . . — POSTSCRIPT(8) — 594
printers . . . — postio − serial interface for postscript . . . — POSTIO(8) — 592
— postnews − submit netnews articles . . . — POSTNEWS(7) — 548
dpost, postdaisy, postdmd, — postprint − filters to produce postscript . . . — POSTSCRIPT(8) — 594
postscript file . . . — postreverse − reverse the page order in a . . . — POSTREVERSE(A)
psi − — postscript interpreter . . . — PSI(9.1) — 662
postio − serial interface for — postscript printers . . . — POSTIO(8) — 592
Instructions) files . . . postbgi − — PostScript translator for BGI (Basic Graphical . . . — POSTBGI(A)
square root . . . exp, log, log10, — pow, sqrt − exponential, logarithm, power, . . . — EXP(3) — 360
— pp − C program pretty printer . . . — PP(A)
to/from a process . . . popen, — ppopen, vepopen, pclose − open a pipe . . . — POPEN(3) — 407
bcd, — ppt, morse − convert to antique media . . . — BCD(A)
— pr − print file . . . — PR(1) — 203
maintain and use bibliographic references . . . — prefer, pinvert, penter, plook, pconvert − . . . — PREFER(1) — 204
flicks, fflicks, — preflicks, 2mux − movie graphics for 5620 . . . — FLICKS(9.1) — 642
cpp − C language — preprocessor . . . — CPP(8) — 569
efl − extended Fortran language — preprocessor . . . — EFL(A)
dag − — preprocessor for drawing directed graphs . . . — DAG(1) — 110
grap − pic — preprocessor for drawing graphs . . . — GRAP(1) — 146
ideal − troff — preprocessor for drawing pictures . . . — IDEAL(1) — 152
xstr − — preprocessor for sharing strings in C programs . . . — XSTR(A)
pic, tpic − troff and tex — preprocessors for drawing pictures . . . — PIC(1) — 197
pp − C program — pretty printer . . . — PP(A)
primes . . . factor, qfactor, — primes − factor a number, generate large . . . — FACTOR(1) — 138
cat − catenate and — print . . . — CAT(1) — 89
blitblt, menudrop − save or — print a screen image . . . — BLITBLT(9.1) — 638
echo, printf − — print arguments . . . — ECHO(1) — 126
pr − — print file . . . — PR(1) — 203
printf, fprintf, sprintf, snprintf − — print formatted output . . . — PRINTF(3) — 411
banner, rot, rnd, bigp − — print in large type . . . — BANNER(A)
thinkblt, think − — print on thinkjet . . . — THINKBLT(9.1) — 672
strings − find — printable strings in a file . . . — STRINGS(1) — 247
— printenv − print environment . . . — PRINTENV(1) — 207

paper − list input on HP2621P . . . printer . . . PAPER(A)
pp − C program pretty . . . printer . . . PP(A)
qsnap − high resolution digital film . . . printer . . . QSNAP(1) 213
lp − . . . printer output . . . LP(1) 170
pxp, pxref − pascal . . . printer, profiler, and cross-reference lister . . . PXP(A)
idcan, itcan − interface to Imagen laser- . . . printer spooler . . . ican, ibcan, ICAN(A)
postio − serial interface for postscript . . . printers . . . POSTIO(8) 592
echo, . . . printf − print arguments . . . ECHO(1) 126
formatted output . . . printf, fprintf, sprintf, snprintf − print . . . PRINTF(3) 411
nice − set program . . . priority . . . NICE(2) 315
renice − change . . . priority of running process . . . RENICE(8) 604
nice, nohup − run commands at low . . . priority or immune to hangup . . . NICE(1) 191
. . . proc − process file system . . . PROC(4) 463
exit, onexit − terminate . . . process . . . EXIT(3) 359
fork − spawn new . . . process . . . FORK(2) 308
getproc, getproctab, putname, getname − jerq . . . process control . . . /newwindow, tolayer, debug, NEWPROC(9.2) 677
proc − . . . process file system . . . PROC(4) 463
setpgrp − get or set user, group, or . . . process identity . . . /setgid, setruid, setlogname, GETUID(2) 309
pi, 3pi − . . . process inspector . . . PI(9.1) 658
ps − . . . process status . . . PS(1) 209
times − get . . . process times . . . TIMES(2) 329
_exit, wait wait3 − terminate . . . process, wait for child to terminate . . . EXIT(2) 306
kill − terminate a . . . process with extreme prejudice . . . KILL(1) 157
kmem, kUmem, mtpr, fineclock − memory and VAX . . . processor registers . . . mem, MEM(4) 458
kdiload, kmcdump − control KMC11 input/output . . . processors . . . kmc, KMC(8) 580
ops5 − a rule-based . . . production-system environment . . . OPS5(A)
. . . prof − display profile data . . . PROF(1) 208
. . . profil − execution time profile . . . PROFIL(2) 318
monitor − prepare execution . . . profile . . . MONITOR(3) 397
prof − display . . . profile data . . . PROF(1) 208
lcomp, lprint − line-by-line . . . profiler . . . LCOMP(1) 161
pxp, pxref − pascal printer, . . . profiler, and cross-reference lister . . . PXP(A)
cb − C . . . program beautifier . . . CB(1) 90
pp − C . . . program pretty printer . . . PP(A)
lint, cyntax, cem − C . . . program verifiers . . . LINT(1) 165
map − draw maps on various . . . projections . . . MAP(7) 542
orient, normalize − map . . . projections . . . PROJ(3) 413
. . . proof − troff output interpreter for 5620 . . . PROOF(9.1) 661
ip, ip_ld − DARPA internet . . . protocol . . . IP(4) 456
smtpd, smtpsched − handle simple mail transfer . . . protocol . . . smtp, smtpqer, SMTP(8) 616
tcp, tcp_ld − DARPA transmission control . . . protocol . . . TCP(4) 470
tcpmgr − accept and place calls via the TCP . . . protocol . . . TCPMGR(8) 622
telnet − user interface to the telnet . . . protocol . . . TELNET(A)
spin − . . . protocol analysis software . . . SPIN(1) 245
unixp_ld, cmc_ld − Datakit interface and . . . protocols . . . dk, dkp_ld, DK(4) 451
dkipconfig, udpconfig − set up DARPA Internet . . . protocols . . . ipconfig, IPCONFIG(8) 579
say − . . . proverbs . . . FORTUNE(A)
. . . ps − process status . . . PS(1) 209
. . . psi − postscript interpreter . . . PSI(9.1) 662
pretty-printer . . . cospan, . . . psr − coordination-specification analyzer and . . . COSPAN(A)
. . . pstat − print system facts . . . PSTAT(8) 596
doctor, tso − . . . psychiatric consultation . . . DOCTOR(A)
Word, Point, Rectangle, Bitmap, Texture, . . . Pt, Rect, Rpt, display, Drect, Jrect −/ . . . TYPES(9.5) 699
/sub, mul, div, eqpt, eqrect, inset, muldiv, . . . ptinrect, raddp, rsubp, rectXrect, rectclip −/ . . . ADD(9.3) 681
. . . ptx − permuted index . . . PTX(1) 210
references . . . refer, lookbib, . . . pubindex − maintain and use bibliographic . . . REFER(1) 222
push, . . . pull, npush, npull − datakit remote file copy . . . PUSH(1) 211
wwb, style, diction, . . . punct − writer's workbench . . . WWB(1) 284
ungetc − . . . push character back into input stream . . . UNGETC(3) 441
copy . . . push, pull, npush, npull − datakit remote file . . . PUSH(1) 211
word-at-a-time/ . . . getc, getchar, fgetc, getw, . . . putc, putchar, fputc, putw − character- or . . . GETC(3) 374
/newwindow, tolayer, debug, getproc, getproctab, . . . putname, getname − jerq process control . . . NEWPROC(9.2) 677
streams . . . fgets, . . . puts, fputs, gets − string input/out on . . . FGETS(3) 362
/getchar, fgetc, getw, putc, putchar, fputc, . . . putw − character- or word-at-a-time stream/ . . . GETC(3) 374
directory . . . pwd, where − machine name and working . . . PWD(1) 212
/getpwuid, getpwnam, setpwent, endpwent, . . . pwdecode − get password file entry . . . GETPWENT(3) 383
cross-reference lister . . . pxp, pxref − pascal printer, profiler, and . . . PXP(A)
large primes . . . factor, . . . qfactor, primes − factor a number, generate . . . FACTOR(1) 138
. . . qns − query name server . . . QNS(7) 549
. . . qsnap − high resolution digital film printer . . . QSNAP(1) 213
. . . qsort − quicker sort . . . QSORT(3) 416
pipebuf − streambuf specialized as circular . . . queue . . . PIPEBUF(A)
smstat − list smtp . . . queues . . . SMSTAT(8) 615
. . . quiz − test your knowledge . . . QUIZ(A)

	quot, findo − file system usage and hogs	QUOT(8)	599
	ra − DEC MSCP disks (RA60, RA80, RA81, RA90)	RA(4)	465
w, fp, ft, fd,	rad – aviation weather	AVW(A)	
on/ . . . . . /div, eqpt, eqrect, inset, muldiv, ptinrect,	raddp, rsubp, rectXrect, rectclip − arithmetic	ADD(9.3)	681
number generator	rand, lrand, frand, nrand, srand – random	RAND(3)	417
return cookies	random, fortune − sample lines from a file,	RANDOM(1)	214
ar,	ranlib − archive and library maintainer	AR(1)	70
rarepl,	rarct − replace bad blocks on MSCP disks	RAREPL(8)	600
picfile −	raster graphic image format	PICFILE(5)	497
	ratfor − rational Fortran dialect	RATFOR(A)	
/icon, lisp, macsyma, maple, ops5, pascal,	ratfor, S, smp, sno, spitbol, struct, twig −/	LANGS(1)	159
	rc − boot script	RC(8)	601
	rc, cd, wait, whatis − command language	RC(1)	215
	rcp − remote file copy	RCP(1)	221
request, own, wait, alarm, sleep, nap, kbdchar,	rcvchar, realtime, sendchar, sendnchars, kill,/	REQUEST(9.2)	679
dirread –	read from directory, hiding format	DIRREAD(2)	302
	read, write – read or write file	READ(2)	319
directory operations . . . . . opendir,	readdir, telldir, seekdir, closedir –	DIRECTORY(3)	355
documents	reader − electronic retrieval of typeset	READER(9.7)	702
link, symlink,	readlink – link to a file	LINK(2)	311
checknews,	readnews − read netnews articles	READNEWS(7)	550
tail,	readslow − deliver the last part of a file	TAIL(1)	253
malloc, free,	realloc, calloc, cfree − memory allocator	MALLOC(3)	392
/own, wait, alarm, sleep, nap, kbdchar, rcvchar,	realtime, sendchar, sendnchars, kill, exit −/	REQUEST(9.2)	679
cut, paste −	rearrange columns of data	CUT(1)	109
	rebecca − graphics touch-up editor	REBECCA(9.1)	663
matching	re_bm, re_cw, re_re – string and pattern	RE(3)	418
	reboot − bootstrapping procedures	REBOOT(8)	602
setgroups – system calls to be avoided	reboot, vadvise, vlimit, vswapon, getgroups,	DEPRECATED(2)	301
cp, mv, ln,	reccp − copy, move, or link files	CP(1)	103
handler	re_comp, re_exec – regular expression	REGEX(A)	
2500 − BVH2500 videotape	recorder	2500(1)	60
backup − backup and	recover files	BACKUP(1)	77
Word, Point, Rectangle, Bitmap, Texture, Pt,	Rect, Rpt, display, Drect, Jrect − graphics/	TYPES(9.5)	699
functions . . . . . Code, addr, bitblt, point,	rectf, screenswap, segment, texture − graphics	BITBLT(9.3)	683
/eqrect, inset, muldiv, ptinrect, raddp, rsubp,	rectXrect, rectclip − arithmetic on points and/	ADD(9.3)	681
re_bm,	re_cw, re_re – string and pattern matching	RE(3)	418
	redcode − assembler for mars game	REDCODE(A)	
mars − memory array	redcode simulator	MARS(A)	
=, ==, =p, ==p −	redo previous shell command	=(1)	61
re_comp,	re_exec – regular expression handler	REGEX(A)	
bibliographic references	refer, lookbib, pubindex − maintain and use	REFER(1)	222
pxref − pascal printer, profiler, and cross-	reference lister . . . . . pxp,	PXP(A)	
pconvert − maintain and use bibliographic	references . . . . . prefer, pinvert, penter, plook,	PREFER(1)	204
pubindex − maintain and use bibliographic	references . . . . . refer, lookbib,	REFER(1)	222
expression	regcomp, regexec, regsub, regerror – regular	REGEXP(3)	420
mtpr, fineclock − memory and VAX processor	registers . . . . . mem, kmem, kUmem,	MEM(4)	458
view2d,	regrid, vdata − movie of a function f(x, y, t)	VIEW2D(1)	279
regcomp, regexec,	regsub, regerror – regular expression	REGEXP(3)	420
re_comp, re_exec –	regular expression handler	REGEX(A)	
sam − screen editor with structural	regular expressions	SAM(9.1)	665
comm − select or	reject lines common to two sorted files	COMM(1)	101
join –	relational database operator	JOIN(1)	156
strip − remove symbols and	relocation bits	STRIP(1)	248
fabs, fmod, floor, ceil – absolute value,	remainder, floor, ceiling functions	FLOOR(3)	365
svcmgr − service	remote computing requests	SVCMGR(8)	618
push, pull, npush, npull − datakit	remote file copy	PUSH(1)	211
rcp −	remote file copy	RCP(1)	221
track − selective	remote file copy	TRACK(A)	
uucp, uulog, uuname − unix-to-unix	remote file copy	UUCP(1)	271
uuto, uupick − simplified unix-to-unix	remote file copy	UUTO(1)	274
con, rx −	remote login and execution	CON(1)	102
dcon, ndcon, rlogin, nrx, rsh, scriptcon −	remote login and execution	DCON(1)	115
cray − run job	remotely on cray-xmp	CRAY(A)	
mkdir, rmdir – make or	remove a directory	MKDIR(2)	313
unlink –	remove directory entry	UNLINK(2)	331
fmount, funmount − mount or	remove file system	FMOUNT(2)	307
rm −	remove (unlink) files	RM(1)	225
	renice − change priority of running process	RENICE(8)	604
uniq − report	repeated lines in a file	UNIQ(1)	270
rcvchar, realtime, sendchar, sendnchars, kill,/	request, own, wait, alarm, sleep, nap, kbdchar,	REQUEST(9.2)	679
re_bm, re_cw,	re_re – string and pattern matching	RE(3)	418
windows, jps,	reshape − create and initialize windows	WINDOWS(9.1)	674
vtimes − get usage of time, space, and paging	resources	VTIMES(2)	332

reader − electronic retrieval of typeset documents . . . . . READER(9.7) 702
postreverse − reverse the page order in a postscript file . . . . . POSTREVERSE(A)
rev, revpag − reverse lines or pages . . . . . REV(1) 224
fseek, ftell, rewind − reposition a stream . . . . . FSEEK(3) 369
rscan, pix − scan page on ricoh scanner and display on 5620 . . . . . RSCAN(A)
execution . . . . . dcon, ndcon, rlogin, nrx, rsh, scriptcon − remote login and . . . . . DCON(1) 115
rm − remove (unlink) files . . . . . RM(1) 225
upas, rmail, translate − mail delivery system . . . . . UPAS(8) 623
mkdir, rmdir − make or remove a directory . . . . . MKDIR(2) 313
mesg_ld, rmesg_ld − message line discipline modules . . . . . MESGLD(4) 459
banner, rot, rnd, bigp − print in large type . . . . . BANNER(A)
adventure, zork, rogue, wump − dungeon-exploration games . . . . . ADVENTURE(A)
chdir, chroot − change working or root directory . . . . . CHDIR(2) 299
banner, rot, rnd, bigp − print in large type . . . . . BANNER(A)
route, routed − IP gateway routing . . . . . ROUTE(8) 605
/mgcd, min, mout, fmin, fmout, move, mcmp, rpow, mpow − multiple precision integer/ . . . . . MP(3) 398
/Point, Rectangle, Bitmap, Texture, Pt, Rect, Rpt, display, Drect, Jrect − graphics data/ . . . . . TYPES(9.5) 699
display on 5620 . . . . . rscan, pix − scan page on ricoh scanner and . . . . . RSCAN(A)
dcon, ndcon, rlogin, nrx, rsh, scriptcon − remote login and execution . . . . . DCON(1) 115
/eqpt, eqrect, inset, muldiv, ptinrect, raddp, rsubp, rectXrect, rectclip − arithmetic on/ . . . . . ADD(9.3) 681
arff − read RT11 files . . . . . ARFF(8) 563
ruler − measure things on the screen . . . . . RULER(9.1) 664
hangup . . . . . nice, nohup − run commands at low priority or immune to . . . . . NICE(1) 191
con, rx − remote login and execution . . . . . CON(1) 102
/lisp, macsyma, maple, ops5, pascal, ratfor, S, smp, sno, spitbol, struct, twig − languages . . . . . LANGS(1) 159
sa, accton − system accounting . . . . . SA(8) 606
expressions . . . . . sam − screen editor with structural regular . . . . . SAM(9.1) 665
random, fortune − sample lines from a file, return cookies . . . . . RANDOM(1) 214
samuel − text editor and C browser . . . . . SAMUEL(A)
blitblt, menudrop − save or print a screen image . . . . . BLITBLT(9.1) 638
system . . . . . savecore − save a core image of the operating . . . . . SAVECORE(8) 608
say − proverbs . . . . . FORTUNE(A)
brk, sbrk − change core allocation . . . . . BRK(2) 298
cscan − scan documents on canon scanner . . . . . CSCAN(1) 106
imscan − scan greyscale images . . . . . IMSCAN(1) 155
rscan, pix − scan page on ricoh scanner and display on 5620 . . . . . RSCAN(A)
scanf, fscanf, sscanf − formatted input . . . . . SCANF(3) 421
scat − sky catalogue . . . . . SCAT(7) 552
clear − clear terminal screen . . . . . CLEAR(A)
ruler − measure things on the screen . . . . . RULER(9.1) 664
twid, pen − doodle on the screen . . . . . TWID(A)
tgoto, tputs − device-independent terminal screen control . . . . . /tgetnum, tgetflag, tgetstr, TERMCAP(3) 435
expressions . . . . . sam − screen editor with structural regular . . . . . SAM(9.1) 665
curses − screen functions with ‘optimal’ cursor motion . . . . . CURSES(3) 351
blitblt, menudrop − save or print a screen image . . . . . BLITBLT(9.1) 638
functions . . . . . Code, addr, bitblt, point, rectf, screenswap, segment, texture − graphics . . . . . BITBLT(9.3) 683
dcon, ndcon, rlogin, nrx, rsh, scriptcon − remote login and execution . . . . . DCON(1) 115
scsi − SCSI interface . . . . . SCSI(4) 467
scsish − SCSI shell . . . . . SCSISH(8) 609
sdb − symbolic debugger . . . . . SDB(A)
move,/ . . . . itom, mfree, madd, msub, mult, mdiv, sdiv, msqrt, mgcd, min, mout, fmin, fmout, ... MP(3) 398
software distribution . . . . . mkpkg, inspkg, seal, unseal − package files for automatic . . . . . MKPKG(8) 586
gre, grep, egrep, fgrep − search a file for a pattern . . . . . GRE(1) 149
sed − stream editor . . . . . SED(1) 226
lseek − seek, move read/write pointer . . . . . LSEEK(2) 312
opendir, readdir, telldir, seekdir, closedir − directory operations . . . . . DIRECTORY(3) 355
Code, addr, bitblt, point, rectf, screenswap, segment, texture − graphics functions . . . . . BITBLT(9.3) 683
apply, pick − repeatedly apply a command; select arguments . . . . . APPLY(1) 68
multiplexing . . . . . select − synchronous input/output . . . . . SELECT(2) 320
files . . . . . comm − select or reject lines common to two sorted . . . . . COMM(1) 101
Mail − send and receive mail . . . . . MAIL(A)
sendcover − send cover sheet to the library . . . . . SENDCOVER(8) 611
docsubmit − send document to library . . . . . DOCSUBMIT(1) 123
mail − send or receive mail . . . . . MAIL(1) 176
netnews − send or receive news articles . . . . . NETNEWS(A)
signal, kill − receive and send signals . . . . . SIGNAL(2) 321
apsend − send troff output to phototypesetter . . . . . APSEND(1) 69
sendcover − send cover sheet to the library . . . . . SENDCOVER(8) 611
/nap, kbdchar, rcvchar, realtime, sendchar, sendnchars, kill, exit − 5620 input/output/ . . . . . REQUEST(9.2) 679
seq − print sequences of numbers . . . . . SEQ(1) 229
tty − serial line interface drivers . . . . . TTY(4) 472
svcmgr − service remote computing requests . . . . . SVCMGR(8) 618
setbuf − assign buffering to a stream . . . . . SETBUF(3) 423
getfields, getmfields, setfields − break a string into fields . . . . . GETFIELDS(3) 376

description/ . . . . . getfsent, getfsspec, getfsfile,	setfsent, endfsent – get file system	GETFSENT(3) 378
/getlogname, getpid, getppid, getpgrp, setuid,	setgid, setruid, setlogname, setpgrp – get/	GETUID(2) 309
getgrent, getgrgid, getgrnam,	setgrent, endgrent – get group file entry	GETGRENT(3) 379
reboot, vadvise, vlimit, vswapon, getgroups,	setgroups – system calls to be avoided	DEPRECATED(2) 301
	setjmp, longjmp – non-local goto	SETJMP(3) 424
crypt,	setkey, encrypt – DES encryption	CRYPT(3) 348
super-user . . . . . su,	setlog − substitute userid temporarily, become	SU(8) 617
strinsure, strinsert, strdelete, strzero,	setmuxbuf, getmuxbuf, movstring − dynamic/	MUXSTRING(9.3) 689
file entry . . . . . getpwent, getpwuid, getpwnam,	setpwent, endpwent, pwdecode – get password	GETPWENT(3) 383
/getpid, getppid, getpgrp, setuid, setgid,	setruid, setlogname, setpgrp – get or set/	GETUID(2) 309
arithmetic functions: absolute value,/ . . . . . abs,	sgn, gcd, lcm, min, max, labs – integer	ARITH(3) 337
command programming language	sh, cd, wait, whatis − shell, the standard	SH(1) 230
=, ==, =p, ==p − redo previous	shell command	=(1) 61
system – issue a	shell command	SYSTEM(3) 432
language . . . . . sh, cd, wait, whatis −	shell, the standard command programming	SH(1) 230
distribution	ship, shipstat – automatic software	SHIP(8) 612
vis −	show invisible characters	VIS(1) 280
	showq − status of stream input/output system	SHOWQ(8) 613
/integer arithmetic functions: absolute value,	sign, greatest common divisor, least common/	ARITH(3) 337
	signal, kill – receive and send signals	SIGNAL(2) 321
functions . . . . . cos,	sin, atan2, sqrt, norm − integer math	COS(9.3) 685
trigonometric functions	sin, cos, tan, asin, acos, atan, atan2 –	SIN(3) 425
	sinh, cosh, tanh – hyperbolic functions	SINH(3) 426
null − data	sink	NULL(4) 462
	size − size of an object file	SIZE(1) 237
	sky − astronomical ephemeris	SKY(7) 553
scat −	sky catalogue	SCAT(7) 552
	sleep − suspend execution for an interval	SLEEP(1) 238
	sleep – suspend execution for an interval	SLEEP(3) 427
sendchar,/ . . . . . request, own, wait, alarm,	sleep, nap, kbdchar, rcvchar, realtime, ...	REQUEST(9.2) 679
	smash − rewrite bad disk sectors	SMASH(8) 614
	sml – Standard ML compiler	SML(1) 239
/lisp, macsyma, maple, ops5, pascal, ratfor, S,	smp, sno, spitbol, struct, twig − languages	LANGS(1) 159
	smstat − list smtp queues	SMSTAT(8) 615
simple mail transfer protocol	smtp, smtpqer, smtpd, smtpsched − handle	SMTP(8) 616
	snake, worm − display chase games	SNAKE(A)
	sno − Snobol language interpreter	SNO(A)
macsyma, maple, ops5, pascal, ratfor, S, smp,	sno, spitbol, struct, twig − languages . . . . . /lisp,	LANGS(1) 159
spitbol −	Snobol language compiler	SPITBOL(A)
sno −	Snobol language interpreter	SNO(A)
	snocone − snobol with syntactic sugar	SNOCONE(1) 240
printf, fprintf, sprintf,	snprintf – print formatted output	PRINTF(3) 411
	soelim − eliminate .so's from nroff input	SOELIM(A)
asd − automatic	software distribution	ASD(8) 564
seal, unseal − package files for automatic	software distribution . . . . . mkpkg, inspkg,	MKPKG(8) 586
ship, shipstat – automatic	software distribution	SHIP(8) 612
qsort – quicker	sort	QSORT(3) 416
tsort − topological	sort	TSORT(1) 267
	sort − sort and/or merge files	SORT(1) 242
comm − select or reject lines common to two	sorted files	COMM(1) 101
look – find lines in a	sorted list	LOOK(1) 168
vtimes − get usage of time,	space, and paging resources	VTIMES(2) 332
fork –	spawn new process	FORK(2) 308
eqnchar −	special character definitions for eqn	EQNCHAR(6) 520
mknod – make a directory or a	special file	MKNOD(2) 314
mknod − construct	special file	MKNOD(8) 585
	spell − find spelling errors	SPELL(1) 244
	spin – protocol analysis software	SPIN(1) 245
	spitbol − Snobol language compiler	SPITBOL(A)
maple, ops5, pascal, ratfor, S, smp, sno,	spitbol, struct, twig − languages . . . . . /macsyma,	LANGS(1) 159
	spline − fit a curve	SPLINE(A)
	split, fsplit − split a file into pieces	SPLIT(1) 246
frexp, ldexp, modf –	split into mantissa and exponent	FREXP(3) 368
itcan − interface to Imagen laser-printer	spooler . . . . . ican, ibcan, idcan,	ICAN(A)
printf, fprintf,	sprintf, snprintf – print formatted output	PRINTF(3) 411
root . . . . . exp, log, log10, pow,	sqrt – exponential, logarithm, power, square	EXP(3) 360
cos, sin, atan2,	sqrt, norm − integer math functions	COS(9.3) 685
rand, lrand, frand, nrand,	srand – random number generator	RAND(3) 417
scanf, fscanf,	sscanf – formatted input	SCANF(3) 421
	stab − symbol table types	STAB(5) 503
stdio –	standard buffered input/output package	STDIO(3) 428
libc −	standard library functions	LIBC(9.3) 686
sml –	Standard ML compiler	SML(1) 239
hang –	start a process in stopped state	HANG(1) 150

stat, lstat, fstat – get file status . . . . . STAT(2) 323
load – load statistics . . . . . LOAD(1) 167
vmstat – report virtual memory statistics . . . . . VMSTAT(8) 627
vismon, sysmon, vwhois – system statistics and mail notification . . . . . VISMON(9.1) 673
ps – process status . . . . . PS(1) 209
stat, lstat, fstat – get file status . . . . . STAT(2) 323
netstat, dkstat – show network status for internet and datakit networks . . . . . NETSTAT(8) 590
uustat – uucp status inquiry and job control . . . . . UUSTAT(1) 273
showq – status of stream input/output system . . . . . SHOWQ(8) 613
fd, stdin, stdout, stderr, tty – file descriptor files . . . . . FD(4) 455
stdiobuf – iostream specialized to stdio FILE . . . . . STDIOBUF(A)
jx – 5620 execution and stdio interpreter . . . . . JX(9.1) 648
package . . . . . stdio – standard buffered input/output . . . . . STDIO(3) 428
stdiobuf – iostream specialized to stdio FILE . . . . . STDIOBUF(A)
fd, stdin, stdout, stderr, tty – file descriptor files . . . . . FD(4) 455
movie, stills – algorithm animation . . . . . MOVIE(9.1) 650
stime, biasclock – set time . . . . . STIME(2) 325
hang – start a process in stopped state . . . . . HANG(1) 150
galloc, gfree, garbage – storage allocation with garbage collection . . . . . GALLOC(3) 371
subroutines . . . . . dbminit, fetch, store, delete, firstkey, nextkey – database . . . . . DBM(3) 353
strncpy, strlen, strchr, strrchr, strpbrk,/ . . . . . strcat, strncat, strcmp, strncmp, strcpy, ... STRING(3) 429
/strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok, strdup – string operations . . . . . STRING(3) 429
movstring – dynamic/ . . . . . strinsure, strinsert, strdelete, strzero, setmuxbuf, getmuxbuf, ... MUXSTRING(9.3) 689
strrchr, strpbrk, strspn, strcspn, strtok, strdup – string operations . . . . . /strlen, strchr, STRING(3) 429
fclose, fflush – open, close, or flush a stream . . . . . fopen, freopen, fdopen, FOPEN(3) 366
fseek, ftell, rewind – reposition a stream . . . . . FSEEK(3) 369
setbuf – assign buffering to a stream . . . . . SETBUF(3) 423
ungetc – push character back into input stream . . . . . UNGETC(3) 441
stream – communication channels . . . . . STREAM(4) 468
conn_ld – line discipline for unique stream connection . . . . . CONNLD(4) 449
sed – stream editor . . . . . SED(1) 226
fputc, putw – character- or word-at-a-time stream input/output . . . . . /getw, putc, putchar, GETC(3) 374
showq – status of stream input/output system . . . . . SHOWQ(8) 613
feof, ferror, clearerr, fileno – stream status inquiries . . . . . FERROR(3) 361
streambuf – interface for derived classes . . . . . SBUF.PROT(A)
buffering class . . . . . streambuf – public interface of character . . . . . SBUF.PUB(A)
pipebuf – streambuf specialized as circular queue . . . . . PIPEBUF(A)
strstreambuf – streambuf specialized to arrays . . . . . SSBUF(A)
fstream – iostream and streambuf specialized to files . . . . . FSTREAM(A)
puts, fputs, gets – string input/out on streams . . . . . fgets, FGETS(3) 362
re_bm, re_cw, re_re – string and pattern matching . . . . . RE(3) 418
getfont – text and font operations . . . . . string, defont, strwidth, infont, outfont, ... STRING(9.3) 690
fgets, puts, fputs, gets – string input/out on streams . . . . . FGETS(3) 362
getfields, getmfields, setfields – break a string into fields . . . . . GETFIELDS(3) 376
strpbrk, strspn, strcspn, strtok, strdup – string operations . . . . . /strlen, strchr, strrchr, STRING(3) 429
bits – variable length bit strings . . . . . BITS(3+) 341
strings – find printable strings in a file . . . . . STRINGS(1) 247
xstr – preprocessor for sharing strings in C programs . . . . . XSTR(A)
getmuxbuf, movstring – dynamic/ . . . . . strinsure, strinsert, strdelete, strzero, setmuxbuf, ... MUXSTRING(9.3) 689
strip – remove symbols and relocation bits . . . . . STRIP(1) 248
basename, dirname – strip filename affixes . . . . . BASENAME(1) 80
strlic – Esterel parser . . . . . STRLIC(A)
strlen, strchr, strrchr, strpbrk,/ . . . . . strcat, strncat, strcmp, strncmp, strcpy, strncpy, ... STRING(3) 429
/strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok, strdup – string/ . . . . . STRING(3) 429
strstream – iostream specialized to arrays . . . . . STRSTREAM(A)
strstreambuf – streambuf specialized to arrays . . . . . SSBUF(A)
strchr, strrchr, strpbrk, strspn, strcspn, strtok, strdup – string operations . . . . . /strlen, STRING(3) 429
atof, atoi, atol, strtod, strtol, strtoul – convert ASCII to numbers . . . . . ATOF(3) 339
struct – structure Fortran programs . . . . . STRUCT(A)
ops5, pascal, ratfor, S, smp, sno, spitbol, struct, twig – languages . . . . . /macsyma, maple, LANGS(1) 159
font operations . . . . . string, defont, strwidth, infont, outfont, getfont – text and . . . . . STRING(9.3) 690
dynamic/ . . . . . strinsure, strinsert, strdelete, strzero, setmuxbuf, getmuxbuf, movstring – . . . . . MUXSTRING(9.3) 689
stty – set terminal options . . . . . STTY(1) 249
wwb, style, diction, punct – writer’s workbench . . . . . WWB(1) 284
become super-user . . . . . su, setlog – substitute userid temporarily, ... SU(8) 617
ptinrect, raddp, rsubp, rectXrect,/ . . . . . add, sub, mul, div, eqpt, eqrect, inset, muldiv, ... ADD(9.3) 681
postnews – submit netnews articles . . . . . POSTNEWS(7) 548
sum, treesum – sum and count blocks in a file . . . . . SUM(1) 251
setlog – substitute userid temporarily, become super-user . . . . . su, SU(8) 617
sleep – suspend execution for an interval . . . . . SLEEP(1) 238
sleep – suspend execution for an interval . . . . . SLEEP(3) 427
svcmgr – service remote computing requests . . . . . SVCMGR(8) 618
swab – swap bytes . . . . . SWAB(3) 431
swapon – specify swapping device . . . . . SWAPON(8) 620

nm – name list (	symbol table)	NM(1)	192
stab –	symbol table types	STAB(5)	503
maple – interactive	symbolic algebraic program	MAPLE(A)	
sdb –	symbolic debugger	SDB(A)	
strip – remove	symbols and relocation bits	STRIP(1)	248
link,	symlink, readlink – link to a file	LINK(2)	311
	sync – force writing of system buffers	SYNC(2)	326
	sync, update – update disk file systems	SYNC(8)	621
select –	synchronous input/output multiplexing	SELECT(2)	320
	syscall – indirect system call	SYSCALL(2)	327
notification vismon,	sysmon, vwhois – system statistics and mail	VISMON(9.1)	673
perror, sys_errlist,	sys_nerr – system error messages	PERROR(3)	400
cu, ct – call out to a terminal or another	system	CU(1)	107
savecore – save a core image of the operating	system	SAVECORE(8)	608
showq – status of stream input/output	system	SHOWQ(8)	613
who, whois, last – who is or was on the	system	WHO(1)	282
sa, accton –	system accounting	SA(8)	606
sync – force writing of	system buffers	SYNC(2)	326
syscall – indirect	system call	SYSCALL(2)	327
intro, errno – introduction to	system calls and error numbers	INTRO(2)	291
vlimit, vswapon, getgroups, setgroups –	system calls to be avoided reboot, vadvise,	DEPRECATED(2)	301
config –	system configuration files	CONFIG(A)	
types – primitive	system data types	TYPES(5)	510
dmesg –	system diagnostic messages	DMESG(8)	572
perror, sys_errlist, sys_nerr –	system error messages	PERROR(3)	400
pstat – print	system facts	PSTAT(8)	596
	system – issue a shell command	SYSTEM(3)	432
dkname – map	system name to Datakit address	DKNAME(7)	540
vismon, sysmon, vwhois –	system statistics and mail notification	VISMON(9.1)	673
tbl – format	tables for nroff or troff	TBL(1)	256
	tabs – set terminal tabs	TABS(1)	252
file	tail, readslow – deliver the last part of a	TAIL(1)	253
functions sin, cos,	tan, asin, acos, atan, atan2 – trigonometric	SIN(3)	425
sinh, cosh,	tanh – hyperbolic functions	SINH(3)	426
tape	tape, mt – identify and manipulate magnetic	TAPE(1)	254
	tar – tape archiver	TAR(1)	255
	tbl – format tables for nroff or troff	TBL(1)	256
d202,	tc – typesetter filters	D202(A)	
tcp_listen, tcp_accept, tcp_rcmd –	tcp networking functions /tcp_connect,	TCP(3)	433
tcpmgr – accept and place calls via the	TCP protocol	TCPMGR(8)	622
tcp,	tcp_ld – DARPA transmission control protocol	TCP(4)	470
networking functions tcp_sock, tcp_connect,	tcp_listen, tcp_accept, tcp_rcmd – tcp	TCP(3)	433
protocol	tcpmgr – accept and place calls via the TCP	TCPMGR(8)	622
	tee – pipe fitting	TEE(1)	258
tk – paginator for the	Tektronix 4014	TK(A)	
	tel, telno, dq – phone books	TEL(7)	554
operations opendir, readdir,	telldir, seekdir, closedir – directory	DIRECTORY(3)	355
	telnet – user interface to the telnet protocol	TELNET(A)	
tel,	telno, dq – phone books	TEL(7)	554
	term – nonstandard mux terminals	TERM(9.1)	671
	termcap – terminal capability file	TERMCAP(5)	505
450 – handle special functions of the DASI 450	terminal	450(A)	
isatty, nametty – find or set name of a	terminal ttyname,	TTYNAME(3)	438
termcap –	terminal capability file	TERMCAP(5)	505
ttys –	terminal initialization data	TTYS(5)	509
getty – set	terminal mode	GETTY(8)	575
tset – set	terminal modes	TSET(A)	
tty, logtty – get	terminal name	TTY(1)	268
stty – set	terminal options	STTY(1)	249
cu, ct – call out to a	terminal or another system	CU(1)	107
tty_ld –	terminal processing line discipline	TTYLD(4)	473
tgetstr, tgoto, tputs – device-independent	terminal screen control /tgetnum, tgetflag,	TERMCAP(3)	435
tabs – set	terminal tabs	TABS(1)	252
handle special functions of DASI 300 and 300s	terminals 300, 300s –	300(A)	
ul – print underlines on screen	terminals	UL(1)	269
	terminals – conventional names	TERM(6)	532
kill –	terminate a process with extreme prejudice	KILL(1)	157
exit, onexit –	terminate process	EXIT(3)	359
_exit, wait wait3 –	terminate process, wait for child to terminate	EXIT(2)	306
	test, [, newer – condition commands	TEST(1)	259
quiz –	test your knowledge	QUIZ(A)	
formatting and typesetting	tex, initex, virtex, dvips, dvit – text	TEX(1)	261
latex, bibtex –	tex macro package and bibliographies	LATEX(6)	522
pic, tpic – troff and	tex preprocessors for drawing pictures	PIC(1)	197

defont, strwidth, infont, outfont, getfont − text and font operations . . . . . string, STRING(9.3) 690
ed, e − text editor . . . . . ED(1) 127
jim, jim.recover − text editor . . . . . JIM(A)
ex, edit, vi − text editor . . . . . VI(1) 276
samuel − text editor and C browser . . . . . SAMUEL(A)
fmt − ultra-simple text formatter . . . . . FMT(1) 142
tex, initex, virtex, dvips, dvit − text formatting and typesetting . . . . . TEX(1) 261
troff, nroff − text formatting and typesetting . . . . . TROFF(1) 265
bitblt, point, rectf, screenswap, segment, texture − graphics functions . . . . . Code, addr, BITBLT(9.3) 683
− graphics/ . . . . . Word, Point, Rectangle, Bitmap, Texture, Pt, Rect, Rpt, display, Drect, Jrect . . . . . TYPES(9.5) 699
tputs − device-independent terminal screen/ . . . . . tgetent, tgetnum, tgetflag, tgetstr, tgoto, ... TERMCAP(3) 435
thinkblt, think − print on thinkjet . . . . . THINKBLT(9.1) 672
thinkabort − ThinkJet routines . . . . . thinkchar, thinknchars, thinkflush, thinkmap, ... THINKCLIENT(9.3) 691
at − execute commands at a later time . . . . . AT(1) 73
stime, biasclock − set time . . . . . STIME(2) 325
timec, timegm, timelocal − convert ASCII to time . . . . . TIMEC(3) 436
time − time a command . . . . . TIME(1) 263
time, ftime − get date and time . . . . . TIME(2) 328
profil − execution time profile . . . . . PROFIL(2) 318
vtimes − get usage of time, space, and paging resources . . . . . VTIMES(2) 332
fputc, putw − character- or word-at-a- time stream input/output . . . . . /getw, putc, putchar, GETC(3) 374
gmtime, asctime, timezone − convert date and time to ASCII . . . . . ctime, localtime, CTIME(3) 349
time . . . . . timec, timegm, timelocal − convert ASCII to . . . . . TIMEC(3) 436
utime − change file mode, owner, group, or times . . . . . chmod, fchmod, chown, fchown, CHMOD(2) 300
times − get process times . . . . . TIMES(2) 329
byteyears − time-space product for file residency . . . . . BYTEYEARS(A)
ctime, localtime, gmtime, asctime, timezone − convert date and time to ASCII . . . . . CTIME(3) 349
alarm, nap, pause − schedule timing delays . . . . . ALARM(2) 297
tk − paginator for the Tektronix 4014 . . . . . TK(A)
mktemp, tmpnam − make a unique file name . . . . . MKTEMP(3) 396
getname −/ P, newproc, muxnewwind, newwindow, tolayer, debug, getproc, getproctab, putname, ... NEWPROC(9.2) 677
tolower, toupper − force upper or lower case . . . . . TOLOWER(3) 437
tsort − topological sort . . . . . TSORT(1) 267
date of a file . . . . . touch, chdate − set modification or access . . . . . CHDATE(1) 95
rebecca − graphics touch-up editor . . . . . REBECCA(9.1) 663
tolower, toupper − force upper or lower case . . . . . TOLOWER(3) 437
weather − conditions and forecast by town . . . . . WEATHER(7) 557
town − gazetteer of US places . . . . . TOWN(7) 555
ped, tped − picture editor . . . . . PED(9.1) 656
drawing pictures . . . . . pic, tpic − troff and tex preprocessors for . . . . . PIC(1) 197
tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs − device-independent terminal screen/ . . . . . TERMCAP(3) 435
tr − translate characters . . . . . TR(1) 264
track − selective remote file copy . . . . . TRACK(A)
smtpqer, smtpd, smtpsched − handle simple mail transfer protocol . . . . . smtp, SMTP(8) 616
upas, rmail, translate − mail delivery system . . . . . UPAS(8) 623
tr − translate characters . . . . . TR(1) 264
tcp, tcp_ld − DARPA transmission control protocol . . . . . TCP(4) 470
ftw − file tree walk . . . . . FTW(3) 370
twig − tree-manipulation language . . . . . TWIG(A)
sum, treesum − sum and count blocks in a file . . . . . SUM(1) 251
sin, cos, tan, asin, acos, atan, atan2 − trigonometric functions . . . . . SIN(3) 425
font − description files for troff . . . . . FONT(5) 492
tbl − format tables for nroff or troff . . . . . TBL(1) 256
pictures . . . . . pic, tpic − troff and tex preprocessors for drawing . . . . . PIC(1) 197
troff, nroff − text formatting and typesetting . . . . . TROFF(1) 265
proof − troff output interpreter for 5620 . . . . . PROOF(9.1) 661
apsend − send troff output to phototypesetter . . . . . APSEND(1) 69
ideal − troff preprocessor for drawing pictures . . . . . IDEAL(1) 152
true, false − provide truth values . . . . . TRUE(1) 266
tset − set terminal modes . . . . . TSET(A)
doctor, tso − psychiatric consultation . . . . . DOCTOR(A)
tsort − topological sort . . . . . TSORT(1) 267
fd, stdin, stdout, stderr, tty − file descriptor files . . . . . FD(4) 455
tty − serial line interface drivers . . . . . TTY(4) 472
tty, logtty − get terminal name . . . . . TTY(1) 268
tty_ld − terminal processing line discipline . . . . . TTYLD(4) 473
of a terminal . . . . . ttyname, isatty, nametty − find or set name . . . . . TTYNAME(3) 438
ttys − terminal initialization data . . . . . TTYS(5) 509
twid, pen − doodle on the screen . . . . . TWID(A)
pascal, ratfor, S, smp, sno, spitbol, struct, twig − languages . . . . . /lisp, macsyma, maple, ops5, LANGS(1) 159
twig − tree-manipulation language . . . . . TWIG(A)
Rpt, display, Drect, Jrect − graphics data types . . . . . /Rectangle, Bitmap, Texture, Pt, Rect, TYPES(9.5) 699
types − primitive system data types . . . . . TYPES(5) 510
monk, monksample, monkspell, monkmerge − typeset documents and letters . . . . . MONK(1) 188

eqn, neqn, checkeq − typeset mathematics	EQN(1)	132
d202, tc − typesetter filters	D202(A)	
font − typesetter fonts	FONT(6)	521
cip − draw pictures for typesetting	CIP(9.1)	641
virtex, dvips, dvit − text formatting and typesetting tex, initex,	TEX(1)	261
troff, nroff − text formatting and typesetting	TROFF(1)	265
ipconfig, dkipconfig, udpconfig − set up DARPA Internet protocols	IPCONFIG(8)	579
networking functions udp_connect, udp_listen, udp_datagram – udp	UDP(3)	439
ul − print underlines on screen terminals	UL(1)	269
umask – set file creation mode mask	UMASK(2)	330
mount, umount − mount and dismount file system	MOUNT(8)	587
uname − identify machine and kernel	UNAME(3)	440
pack, unpack, pcat, compress, uncompress, zcat – compress and expand files	PACK(1)	195
ul − print underlines on screen terminals	UL(1)	269
istream − formatted and unformatted input	ISTREAM(A)	
ostream − formatted and unformatted output	OSTREAM(A)	
stream ungetc – push character back into input	UNGETC(3)	441
uniq − report repeated lines in a file	UNIQ(1)	270
mktemp, tmpnam – make a unique file name	MKTEMP(3)	396
units − conversion program	UNITS(7)	556
config − configure a Unix kernel	CONFIG(A)	
protocols dk, dkp_ld, unixp_ld, cmc_ld − Datakit interface and	DK(4)	451
rm − remove (unlink) files	RM(1)	225
unlink – remove directory entry	UNLINK(2)	331
compress and expand files pack, unpack, pcat, compress, uncompress, zcat –	PACK(1)	195
distribution mkpkg, inspkg, seal, unseal − package files for automatic software	MKPKG(8)	586
upas, rmail, translate − mail delivery system	UPAS(8)	623
sync, update − update disk file systems	SYNC(8)	621
/lbitblt, lpoint, lrectf, lsegment, ltexture, upfront − layer control and graphics	NEWLAYER(9.2)	676
tolower, toupper – force upper or lower case	TOLOWER(3)	437
town − gazetteer of US places	TOWN(7)	555
quot, findo − file system usage and hogs	QUOT(8)	599
vtimes − get usage of time, space, and paging resources	VTIMES(2)	332
netnews − usenet news articles, utility files	NETNEWS(5)	495
setruid, setlogname, setpgrp – get or set user, group, or process identity /setgid,	GETUID(2)	309
getuid, id – get user identity	GETUID(1)	145
su, setlog − substitute userid temporarily, become super-user	SU(8)	617
wall − write to all users	WALL(8)	628
write, mesg − write to other users, allow or forbid messages	WRITE(1)	283
groups of programs usgmake − maintain, update, and regenerate	USGMAKE(A)	
times chmod, fchmod, chown, fchown, utime – change file mode, owner, group, or	CHMOD(2)	300
utmp, wtmp − login records	UTMP(5)	511
file transport and execution uucico, uusched, uuxqt, kick, debug − uucp	UUCICO(8)	625
uucleanup − uucp spool directory clean-up	UUCLEANUP(8)	626
uustat − uucp status inquiry and job control	UUSTAT(1)	273
copy uucp, uulog, uuname − unix-to-unix remote file	UUCP(1)	271
copy uuto, uupick − simplified unix-to-unix remote file	UUTO(1)	274
transport and execution uucico, uusched, uuxqt, kick, debug − uucp file	UUCICO(8)	625
uustat − uucp status inquiry and job control	UUSTAT(1)	273
file copy uuto, uupick − simplified unix-to-unix remote	UUTO(1)	274
uux − unix to unix command execution	UUX(1)	275
execution uucico, uusched, uuxqt, kick, debug − uucp file transport and	UUCICO(8)	625
− system calls to be avoided reboot, vadvise, vlimit, vswapon, getgroups, setgroups	DEPRECATED(2)	301
varargs – variable argument list	VARARGS(3)	442
console − VAX console interface	CONSOLE(4)	450
mem, kmem, kUmem, mtpr, fineclock − memory and VAX processor registers	MEM(4)	458
view2d, regrid, vdata − movie of a function f(x, y, t)	VIEW2D(1)	279
vec, move, etc. – plot graphics interface	PLOT(3)	403
process popen, ppopen, vepopen, pclose – open a pipe to/from a	POPEN(3)	407
lint, cyntax, cem − C program verifiers	LINT(1)	165
ex, edit, vi − text editor	VI(1)	276
2500 − BVH2500 videotape recorder	2500(1)	60
view2d − movie of a function f(x, y, t)	VIEW2D(5)	512
y, t) view2d, moviefil – movie of a function f(x,	VIEW2D(3)	443
f(x, y, t) view2d, regrid, vdata − movie of a function	VIEW2D(1)	279
typesetting tex, initex, virtex, dvips, dvit − text formatting and	TEX(1)	261
vmstat − report virtual memory statistics	VMSTAT(8)	627
vis − show invisible characters	VIS(1)	280
visi − mathematical spreadsheet	VISI(A)	
mail notification vismon, sysmon, vwhois − system statistics and	VISMON(9.1)	673
vmstat − report virtual memory statistics	VMSTAT(8)	627
to be avoided reboot, vadvise, vlimit, vswapon, getgroups, setgroups – system calls	DEPRECATED(2)	301
resources vtimes − get usage of time, space, and paging	VTIMES(2)	332
notification vismon, sysmon, vwhois − system statistics and mail	VISMON(9.1)	673

w, fp, ft, fd, rad – aviation weather . . . . . AVW(A)
realtime, sendchar, sendnchars,/ . . . . request, own, wait, alarm, sleep, nap, kbdchar, rcvchar, ... REQUEST(9.2) 679
child to terminate . . . . . _exit, wait wait3 – terminate process, wait for . . . . . EXIT(2) 306
rc, cd, wait, whatis – command language . . . . . RC(1) 215
programming language . . . . . sh, cd, wait, whatis – shell, the standard command . . . . . SH(1) 230
terminate . . . . . _exit, wait wait3 – terminate process, wait for child to . . . . . EXIT(2) 306
wall – write to all users . . . . . WALL(8) 628
ogre – war game . . . . . OGRE(A)
warp – war games . . . . . WARP(A)
bianchi – espresso, steamed milk, hot water . . . . . BIANCHI(A)
wc – word count . . . . . WC(1) 281
w, fp, ft, fd, rad – aviation weather . . . . . AVW(A)
weather – conditions and forecast by town . . . . . WEATHER(7) 557
wx - get weather information . . . . . WX(A)
rc, cd, wait, whatis – command language . . . . . RC(1) 215
programming language . . . . . sh, cd, wait, whatis – shell, the standard command . . . . . SH(1) 230
pwd, where – machine name and working directory . . . . . PWD(1) 212
whoami – computer name . . . . . WHOAMI(5) 513
who, whois, last – who is or was on the system . . . . . WHO(1) 282
windows . . . . . windows, jps, reshape – create and initialize . . . . . WINDOWS(9.1) 674
wc – word count . . . . . WC(1) 281
hangman, word_clout – word games . . . . . HANGMAN(A)
Rect, Rpt, display, Drect, Jrect – graphics/ . . . . . Word, Point, Rectangle, Bitmap, Texture, Pt, ... TYPES(9.5) 699
putc, putchar, fputc, putw – character- or word-at-a-time stream input/output . . . . . /getw, GETC(3) 374
hangman, word_clout – word games . . . . . HANGMAN(A)
dict – look up words in English dictionaries . . . . . DICT(7) 538
wwb, style, diction, punct – writer’s workbench . . . . . WWB(1) 284
pwd, where – machine name and working directory . . . . . PWD(1) 212
chdir, chroot – change working or root directory . . . . . CHDIR(2) 299
snake, worm – display chase games . . . . . SNAKE(A)
worm – format of worm disks . . . . . WORM(5) 514
worm, jukebox – optical disk utilities . . . . . WORM(8) 629
forbid messages . . . . . write, mesg – write to other users, allow or . . . . . WRITE(1) 283
read, write – read or write file . . . . . READ(2) 319
lseek – seek, move read/write pointer . . . . . LSEEK(2) 312
wall – write to all users . . . . . WALL(8) 628
write, mesg – write to other users, allow or forbid messages . . . . . WRITE(1) 283
festoon – memo writer . . . . . FESTOON(A)
wwb, style, diction, punct – writer’s workbench . . . . . WWB(1) 284
sync – force writing of system buffers . . . . . SYNC(2) 326
utmp, wtmp – login records . . . . . UTMP(5) 511
adventure, zork, rogue, wump – dungeon-exploration games . . . . . ADVENTURE(A)
workbench . . . . . wwb, style, diction, punct – writer’s . . . . . WWB(1) 284
clock . . . . . wwv – print or set the date from accurate . . . . . WWV(1) 285
wx - get weather information . . . . . WX(A)
command . . . . . xargs – construct argument lists and execute . . . . . XARGS(A)
xd, od – hex, octal, decimal, or ASCII dump . . . . . XD(1) 286
cray – run job remotely on cray-programs . . . . . xmp . . . . . CRAY(A)
xstr – preprocessor for sharing strings in C . . . . . XSTR(A)
yacc, eyacc – yet another compiler-compiler . . . . . YACC(1) 287
bison – GNU Project parser generator (yacc replacement) . . . . . BISON(A)
pack, unpack, pcat, compress, uncompress, zcat – compress and expand files . . . . . PACK(1) 195
adventure, zork, rogue, wump – dungeon-exploration games . . . . . ADVENTURE(A)

GLOSSARY

This glossary covers major terms that have special meaning for the UNIX system. It excludes ordinary terms of art such as 'ASCII', 'compiler', 'address space', or 'byte'. It also excludes most terms peculiar to a single part of UNIX, e.g. 'diversion' (*troff*), 'enumeration' (C), or 'pattern space' (*sed*).

absolute pathname same as *full pathname.*

access time the most recent *time* at which the contents of a *file* [2] were read or *executed* [2]; cf. *modification time.*

alarm a *signal* scheduled by the clock.

a.out the default name of a freshly compiled *object file,* pronounced 'A-dot-out'; historically `a.out` signified assembler output.

archive 1. a collection of data gathered from several *files* into one file. 2. especially, such a collection gathered by *ar*(1) for use as a *library.*

argument 1. a string made available to a *process* upon *executing* a *file.* 2. a string in a *command,* which the *shell* [1] will pass to the command program as an argument [1].

ASCII file same as *text file.*

automatic persistent only during the invocation of a procedure, said of data belonging to a *process;* automatic data occupies the *stack segment;* cf. *static.*

background running independently of a terminal, said of a *process;* converse of *foreground.*

basename 1. the last component of a *pathname;* the basename of `/bin/proc` is `proc`. 2. occasionally a basename [1] with conventional suffixes removed; see *basename*(1).

blit vernacular name for a prototype Teletype 5620 terminal; cf. *jerq.*

block 1. the basic unit of *buffering* [1] in a *file system* [3], 1024 or 4096 bytes in this system. 2. a unit in which memory usage is reported, usually 1024 bytes, but 512 bytes in imported code such as *cpio*(1).

block device a *device* that may contain a *file system* [3], typically a permanent storage device such as a tape or disk drive; the unit of data transfer is one *block;* cf. *character device.*

boot to start the operating system, so called because the *kernel* must bootstrap itself from secondary store into an empty machine. No *login* [3] or *process* persists across a boot.

boot block the first block of a *file system* [1], which is reserved for a *booting* program.

break 1. an out-of-band signal on an asynchronous data line arising from the 'break' or 'interrupt' key on a terminal; before *logging in* a break causes a change in baud rate; thereafter it is interpreted as an *interrupt.* 2. a control statement in C or *sh*(1). 3. the *program break.* 4. in *troff*(1), a point in running text where a new line must begin.

BSD see *UNIX.*

bss segment see *segment.*

buffer 1. a staging area for input-output where arbitrary-length transactions are collected into convenient units for system operations. 2. to use buffers.

buffer pool a region of *kernel* memory available for holding *blocks;* non-*raw* input-output for *block devices* and *file systems* [3] goes through the buffer pool to make read and write operations independent of device format.

cbreak a mode of terminal input in which every character not a *special character* becomes available to a *read*(2) operation as soon as it is typed, instead of being *buffered* up to a *newline* or *EOT character.*

change time the most recent *time* at which the contents of a *file* [2] or its *inode* properties (e.g. *owner* and *mode*) were altered; cf. *modification time.*

character 1. a unit of store, usually 8 bits; a byte. 2. a token of the ASCII code, with octal value in the range 0 to 0177.

character device a *device,* such as a terminal or the *null device,* that cannot contain a *file system* [3]; cf. *block device.*

child process see *fork.*

client a *process* that uses a service by *opening* the interface to a *server.*

close to make an *open file* unavailable for input or output; converse of *open.*

command 1. an instruction to the *shell* [1], usually to run a *program* [1] as a *child process.* 2. by extension, any *executable file,* especially a *utility program.*

command file same as *shell script.*

connection line discipline a *line discipline* used on *mounted streams* whereby each *opening* of the stream provides a new *pipe* to the stream's *server;* see *connld*(4).

connection service a collection of *mounted streams,* conventionally in *directory* `/cs`, that encapsulate generic setup protocols to reach *servers;* see *ipc*(3).

control character an ASCII character with octal code 0-037 or 0177, which does not print but may otherwise affect the behavior of a terminal; cf. *special character.*

control stream, control terminal a *stream* [2] associated with a *process* from which the process may receive *interrupt, quit,* and *hangup* signals; cf. *process group.* The control stream is conventionally `/dev/tty`, *file descriptor* 3.

cooked not *raw* [1], said of an input *stream* [2] in which *special characters* are active.

cookie a peculiar goody, token, saying, or remembrance returned by or presented to a *program* [3], cf. *file descriptor, process number.*

core file a *core image* of a terminated *process* saved for debugging; a core file is created under the name `core` in the *current directory* of the process.

core image the data of a running or terminated *process;* the data may reside physically in main store, in the *swap area,* or in a *core file,* and logically in a *process file.*

create to *open* a file for writing, bringing it into existence as a *plain file* if necessary, and discarding any data it may have contained previously; cf. *unlink.*

current directory, working directory the directory from which *relative pathnames* begin; a current directory is associated with each *process.*

daemon a *background* process, often perpetual, that performs a system-wide public function, e.g. *calendar*(1) and *cron*(8); the affected spelling is an ancient legacy; cf. *server.*

Datakit AT&T trademark for a data network based on virtual circuit switching.

data segment see *segment.*

date same as *time.*

device 1. a *file* [2] that is not a *plain file* or a *directory,* examples are tape drive, terminal, a span of *blocks* on a disk drive, or the *null device;* a *special file.* 2. a physical input-output unit.

device number a system-dependent code used in *inodes* to distinguish *devices* [1]. A device number has two parts: **major device** for the type of the device and **minor device** for an individual device of a type.

directory a *file* [2] that comprises a catalog of *filenames* [2]; the organizing principle of the *file system* [2], a directory consists of *entries* which specify further files [2], including directories, and constitutes a node of the *directory tree.*

directory entry, entry 1. an association of a name with an *inode number* appearing as an element of a *directory.* 2. the name part of such an association.

directory hierarchy, directory tree, file hierarchy the tree of all *directories,* in which each is reachable from the *root* [2] via a *path* of *subdirectories.*

driver a set of *kernel* subroutines that handle input/output for a given *major device* type.

echo to transmit characters received from a terminal back to it; the technique of echoing to a *full duplex* terminal provides acknowledgement of typed input.

effective userid see *set userid.*

end of file the condition of the *read pointer* being past the last character of an *open file.* End of file is signified by a read operation returning zero bytes, or by an EOF return from functions of the *stdio* package; it may be simulated from a terminal by typing EOT (control-D) after a *newline.*

entry see *directory entry.*

environment 1. a set of strings, distinct from the *arguments,* made available to a *process* when it *executes* [1] a *file;* the environment is usually inherited across *exec*(2) operations. 2. a specific environment [1] maintained by the *shell* [1]. 3. a nebulously identified way of doing things, as in 'interactive environment': deprecated usage, not always expunged from these manuals.

EOF 1. same as *end of file.* 2. a conventional value returned at end of file by *stdio* functions.

EOT, EOT character 1. the ASCII 'end of transmission' character, control-D, octal code 04. 2. a character that ends (and is excluded from) data obtained by *read*(2) from a terminal; EOT at the beginning of a line causes the *end of file* condition.

epoch see *time.*

erase character a *special character* [2] which, when received from a terminal, is deleted together with the character immediately preceding it on the line; usually defaulted to `#` or backspace, the erase character may be changed by *stty*(1); cf. *kill character.*

escape character 1. The ASCII character ESC, octal 033. 2. a *special character* that may cause the following character to have other than its usual meaning; the default escape character in terminal input is `\`.

executable file, object program 1. an *object file* that is ready to be copied into the address space of a *process* to run as the code of that process. 2. a file that has execute *permission,* either an *executable file* [1] or a *shell script.*

execute, exec 1. to replace the *text segment* and *data segments* of a *process* with a given *program* [1]; see *exec*(2). 2. informally, to run a *program.*

exit to terminate a *process;* exit is voluntary, in contrast to *kill.*

exit status, return code an integer value denoting the outcome of a *process,* including an indication of the cause of termination.

export 1. to include in the *environment* [1] of a *process* when it is *executed* [1], said of a string.

external known beyond the scope of a single compilation, said of names of data and functions in a program.

file 1. in general, a potential source of input or destination for output. 2. most specifically, an *inode* and/or associated contents, i.e. a *plain file,* a *special file,* or a *directory.* 3. a *directory entry* [1]; several directory entries may name the same file [2]. 4. most loosely, a *plain file.*

file descriptor a conventional integer quantity, or *cookie,* that designates an *open file;* cf. *stream* [1].

file descriptor file a *special file* that serves as a name for an *open file* [1]; see *fd*(4).

file hierarchy same as *directory hierarchy.*

filename 1. a *pathname.* 2. same as *basename.*

file pointer the number of the next byte that would normally be read or written in a *file* [2]; also called *read pointer, write pointer,* or *seek pointer.*

file system 1. a detachable part of the *directory hierarchy.* 2. the collection of all *files* [2] accessible to a computer, organized into a directory hierarchy. 3. in particular a *block device* in which *files* [2] are organized according to an *i-list* and accessed by *paths* through *directories.*

file system switch a *kernel* table that directs the interpretation of operations in *file systems* [1] and *special file systems.*

file times *access time, change time,* and *modification time.*

filter a *program* [1] that reads from the *standard input* and writes on the *standard output,* so called because it can be used as a data-transformer in a *pipeline.*

flag same as *option.*

flush to empty a *buffer,* for example to throw away unwanted input-output upon *interrupt* or to release output from the clutches of *stdio.*

foreground running under direct control of a terminal, said of a *process;* converse of *background.*

fork 1. to split one *process* into two, the **parent process** and **child process,** with separate, but initially identical,

text, data, and *stack segments.* 2. informally, same as *spawn.*

free list in a *file system* [1], the list of *blocks* that are not occupied by data.

full duplex 1. capable of carrying information simultaneously in both directions, said of a communication channel. 2. transmitting what is typed while printing what is received, said of a terminal, or of terminal communication; cf. *half duplex* and *echo.*

gid see *groupid.*

global same as *external.*

group 1. a set of *permissions* alternative to *owner* permissions for access to a *file.* 2. a set of *userids* that may assume the privileges of a group [1]. 3. the *groupid* of a *file.*

group file the file `/etc/group`, which tells which *groupids* are available to each *login name.*

groupid, gid an integer value associated with *processes* and *files* [2] used in determining *permissions.*

half duplex 1. capable of carrying information in both directions, but not simultaneously, said of a communication channel. 2. transmitting (usually typing what is transmitted) and receiving, but not simultaneously, said of a terminal; cf. *full duplex.*

hang 1. to *execute* a *program* [1] so that it stops immediately, usually for debugging; cf. *hang*(1). 2. to wait for something that won't happen; said of a process.

hangup a *signal* indicating that a user's terminal has been disconnected.

hard link 1. a *directory entry* [1] that does not designate a *symbolic link.* 2. a *link* [2] that is not a *symbolic link.*

header file same as *include file;* the name of a header file customarily ends with `.h` (dot-h).

here file in a *shell script,* literally given input data for a *command;* here files are introduced by <<.

hole a gap in a *plain file* caused by *seeking* while writing; *read*(2) takes data in holes to be zero; a *block* in a hole occupies no space in its *file system* [1].

home directory 1. the *current directory* established for each user upon *logging in.* 2. the variable `HOME` in the *environment* [2], used in particular as the default destination for the *shell* [1] `cd` command.

host process a *process* working in collaboration with a *terminal process.*

HZ alternating-current line frequency, 60 in North America, 50 in most other places.

i-list the index to a *file system* [1] listing all the *inodes* of the file system; cf. *inode number.*

image same as *core image.*

include file, header file a *file* [4], usually containing shared data declarations, that is to be copied into source programs as they are compiled.

inode an element of a *file system* [1]; an inode specifies all properties of a particular *file* [2] and locates the file's contents, if any.

inode number, i-number the position of an *inode* in the *i-list* of a *file system* [1].

Internet an informal association of data networks using protocols developed mainly under the auspices of DARPA.

interrupt 1. a *signal* that normally terminates a *process,* caused by a *break* [1] or an *interrupt character;* cf. *quit.* 2. loosely, any *signal.*

interrupt character a character (normally ASCII DEL) that, when typed on a *control terminal,* causes an *interrupt.*

io control, ioctl 1. a *system call* that changes the behavior of an *open file,* for example in interpreting *control characters.* 2. a function performed by *ioctl*(2).

jerq vernacular name for a Teletype 5620 terminal.

K 1024; 1024 bytes: a '4K file system' has 4096-byte *blocks.*

kernel the UNIX system proper; resident code that implements the *system calls.*

kernel space, kernel address space data and code that are addressable only by the *kernel.*

kill 1. a particular *signal* guaranteed to terminate a *process.* 2. by extension, to send any *signal* to a *process.* 3. the *kill character.*

kill character a *special character,* which, when received from a terminal, is deleted together with all preceding characters on the line; defaulted to `@`, the kill character may be changed by *stty*(1); cf. *erase character.*

layer a virtual terminal appearing on a screen and also capable of running *terminal processes.* The *processes* associated with each layer constitute a separate *process group.*

library an *archive* [2] of *object files* from which the *link editor* may select functions and data as needed.

line in a *text file,* a sequence of bytes terminated by a *newline.*

line discipline a module to handle protocol or data conversion for a *stream* [2]. A line discipline, unlike a *filter,* is part of the *kernel.*

link 1. to add an entry for an existing *file* [2] to a directory; converse of *unlink.* 2. particularly, any but one putatively primary directory entry for a given *inode,* either linked [1] or a *symbolic link.*

link count the number of *directory entries* [1] that pertain to an *inode;* a *file* [2] ceases to exist when its link count becomes zero and it is not *open.*

link editor, loader the utility *ld*(1), which combines separately compiled *object files* into a single object file.

listen to issue a read on a *stream,* waiting for data, usually a service request, from some other *process* or *remote* source.

loader same as *link editor.*

log in to identify one's self as a user and start a computing session.

login 1. the *program* that controls logging in. 2. the act of *logging in.* 3. by extension, the computing session that follows a login [2].

login name the name by which a person is identified when *logging in;* cf. *userid.*

magic number a *cookie* that distinguishes various flavors of *object file;* see *a.out*(5).

major device see *device number.*

makefile, mkfile a list of dependencies among files and recipes for updating them, usually by recompilation, used by *make*(1) or *mk*(1) to maintain self-consistent software; 'mkfile' rhymes with 'book file'.

manager same as *server.*

memory image same as *core image.*

message line discipline a *line discipline* used on *pipes* wherein a record of each *system call* on one end of the pipe is received as a message at the other end; see *mesgld*(4).

minor device see *device number*.

mode, file mode the *permissions* of a *file;* colloquially referred to by a 3-digit octal number, e.g. ‘a 755 file’; see *chmod*(1).

modification time the most recent *time* at which the contents of a *file* [2] were altered; cf. *access time, change time.*

mount to associate with a *directory entry* [2] a *stream* [2] or the *root* [1] of a *file system* [1], making respectively a virtual file or an extended *file system* [2]. Mounting assigns to the *inode* of a directory entry a temporary meaning, which ceases with the associated *server* [1], with explicit unmounting, or with *booting.*

mounted stream a *directory entry* [2] on which an open *stream* [2] has been *mounted.*

named pipe same as *mounted stream.*

namelist same as *symbol table.*

network file system a mechanism for cross-*mounting* the *file systems* [2] of different computers; see *netfs*(8).

newline the combined function of carriage return and line feed, represented by the ASCII character LF, octal value 012; separates *lines* in a *text file;* newline is evoked by the ‘return’ key on most terminals.

nice to place a *process* in *nice mode* [2].

nice mode 1. the scheduling priority of a *process;* see *nice*(2). 2. in particular, a low priority.

null device a *device* [1] that always yields *end of file* on reading and discards all data on writing.

null pointer in C, a pointer object with value 0.

object file a *file* [2] that contains machine language code and data; object files are produced from source programs by compilers and from other object files and *libraries* by the *link editor;* an object file that is ready to run is an *executable file* [1].

open to make a *file* [2] available for writing or reading, with the *file pointer* positioned at byte 0; converse of *close;* cf. *create.*

open file 1. the destination for input or output obtained by *opening* a *file* [2] or creating a *pipe;* a *file descriptor;* open files are shared across *forks* and persist across *executes* [2]. 2. loosely, a *file* [2] that has been opened.

option an *argument* that affects the way a *command* works; option names customarily begin with −.

ordinary file same as *plain file.*

other 1. the category of *permissions* regulating access to a *file* [2] by processes with *userid* different from the *owner* and *groupid* different from the *group* of the file. 2. the customary name of the default *group* [2] assigned upon *login* [2].

owner a *userid* associated with a *file* [2], usually that of the *process* that created it; the owner has distinctive *permissions* for a file.

page a hardware-dependent unit of virtual memory management, defined in `/usr/include/sys/types.h`; cf. *block.*

parent directory the *directory* next nearer the *root* than a given directory; the inverse of a *subdirectory.* A *directory entry* [2] `..` always refers to the parent directory.

parent process see *fork.*

password a secret word used to confirm a user’s right to *log in* under a particular *userid;* passwords are encrypted by a one-way algorithm and kept in the *password file.*

password file a record of all *login names* with the *password, userid, groupid, home directory,* and *shell* [2] for each, used to control access to the system.

path, pathname a chain of names designating a *file;* a **relative pathname** leads from the *current directory,* for example, a path to *directory* `A`, thence to directory `B`, thence to *file* [2] `C` is denoted `A/B/C`; a **full pathname** begins at the *root* [2], indicated by an initial `/`, as in `/A/B/C`.

permission a right to access a *file* [2] in a particular way: read, write, execute (or look up in, if a directory); permissions are granted separately to *owner, group,* and *others.* **permission bit** a permission, so called because each permission is encoded into one bit in an *inode.*

pid see *process id.*

pipe a nameless *stream* [2] connection between *processes,* whereby data written on an *open file* in one process may be read in another; cf. *named pipe.*

pipeline a sequence of *programs* [1] connected by *pipes.*

plain file, ordinary file a *file* [2] that is neither a *special file* nor a *directory;* plain files are the customary repository of data.

priority see *nice mode* [1].

process a connected sequence of computation; a process is characterized by a *core image* with instruction location counter, *current directory,* a set of *open files, control stream, userid,* and *groupid.*

process directory a *special file system,* `/proc`, in which all *process files* appear.

process file a *core image* of a *process,* accessible as a *file* [3].

process group a set of processes that share a *control stream* and among which *signals* may be broadcast; process groups are created upon *logging in,* creating a window, or issuing certain *io controls;* see *stream*(4).

process number, process id, pid an integer that identifies a *process.*

profile 1. an optional *shell script,* `.profile`, used by the *shell* [1] upon *logging in* to establish the *environment* [3] and other working conditions customary to a particular user. 2. to collect a histogram of values of the instruction location counter of a *process;* see *monitor*(3).

program 1. an *executable file.* 2. a *process.* 3. all the usual meanings.

program break the first address beyond the *static* data accessible to a *process;* the program break may be adjusted by *brk*(2).

quit a *signal* that normally terminates a *process,* caused by a **quit character** (normally control-\); quit differs from *interrupt* in that quit creates a *core file* for the terminated process.

random library a *library* that contains an index to *external* names; a library is made into a random library by *ranlib* see *ar*(1). Nonrandom libraries must be carefully ordered for the *link editor* to cope with cross references among the subroutines.

raw 1. a mode of terminal input in which every character typed is passed to a reading *process* and *special characters* lose their special character; converse of *cooked.* 2. said of input-output to a *raw device.*

raw device a *device* [1], read and write operations to which are not *buffered,* and are synchronized to natural records of the physical device. A raw device is usually a

character device that addresses the same physical device as some *block device.*

read ahead to fill input *buffers* in the *kernel* in advance of *read*(2) operations.

read pointer same as *file pointer.*

real userid see *set userid.*

reboot same as *boot.*

regular expression 1. an expression denoting a set of strings in a notation due to Kleene. 2. especially, a restricted and modified form of regular expression used for pattern-matching in *ed*(1) and elsewhere.

relative pathname see *pathname.*

relocation bits, relocation information information in an *object file* that tells the *link editor* how to adjust addresses when combining it with other *object files;* 'bits' is a fossilized misnomer.

return code same as *exit status.*

root 1. a distinguished directory that constitutes the origin of the *directory hierarchy* in a *file system* [1]. 2. specifically, the origin for the *file system* [2], with the conventional *pathname* `/`. 3. the conventional *login name* for the *super-user.*

schedule to assign resources − main store and CPU time − to *processes.* **scheduler** a permanent *process,* with *process number* 1, and associated *kernel* facilities that do scheduling.

search path a list of *pathnames* of *directories* that determines the meaning of a *command* [2] name. In one form of *exec,* a *relative pathname* is prefixed with members of the search path in turn until a pathname of an *executable file* [2] results.

seek to set the *file pointer* to a specified place in an *open file.*

seek pointer same as *file pointer.*

segment a contiguous range of the address space of a *process* with consistent store access capabilities; the four segments are (i) the **text segment,** occupied by executable code, (ii) the **data segment,** occupied by *static* data that is explicitly initialized, (iii) the **bss segment,** occupied by static data that is implicitly initialized to zero values (the name comes from a 1950's assembler), and (iv) the **stack segment,** occupied by *automatic* data, see *stack;* sometimes (ii), (iii), and (iv) are collectively called data segments.

server 1. a *process,* usually permanent, that serves *clients* through a *mounted* interface, either a *stream* or a *special file system.* 2. a program, often a server [1], accessed via a distinctive network address to provide some remote computing service.

set userid a special *permission* for an *executable file* [1] that causes a *process* executing it to have the access rights of the *owner* of the file; the owner's *userid* becomes the **effective userid** of the process, distinguished from the **real userid** under which the process began. **set userid bit** the associated *permission bit.*

shared text a *text segment,* one copy of which may be used simultaneously by more than one *process.*

shell 1. the program *sh*(1), which causes other programs to be executed on *command* [2]; the shell is usually started on a user's behalf upon *logging in.* 2. by analogy, any program started upon logging in.

shell script, command file a *file* [2] of *commands* taken as input to the *shell* [1].

signal an exceptional occurrence that causes a *process* to terminate or divert from the normal flow of control; cf. *interrupt, kill.*

sleep to cease activity for a specified time, or until a *signal* occurs, said of a *process.*

soft link same as *symbolic link.*

spawn to cause a *program* to be run in a *child process;* cf. *fork, execute.*

special character a character, which, when typed at a terminal, modifies the input or affects the behavior of *processes* for which that terminal is the *control terminal;* examples are the *interrupt character, erase character,* and *EOT character.*

special file an *inode* that designates a *device,* further categorized as either (i) a **block special file** describing a *block device,* or (ii) a **character special file** describing a *character device.*

special file system 1. a virtual *directory,* in which '*entries*' are constructed as needed. Examples are `/n`, the *network file system,* and `/proc`, the *process directory.*

spool to collect and serialize output from multiple *processes* competing for a single output service. **spooler** a *daemon* that spools. **spool area** a *directory* in which a spooler collects work.

stack, stack segment a *segment* of the address space into which *automatic* data and subroutine linkage information is allocated in last-in-first-out fashion; the stack occupies the largest data addresses and grows downward towards *static* data.

standard input, standard output, standard error *open files,* customarily available when a *process* begins, with *file descriptors* 0, 1, 2 and *stdio* names `stdin`, `stdout`, `stderr`. Where possible, utilities by default read from the standard input, write on the standard output, and place error comments on the standard error file.

standard input-output see *stdio.*

start/stop protocol, XON-XOFF protocol a communication flow-control method in which a received *special character,* the **stop character,** causes output to be suspended, and a received **start character** causes output to be resumed. The default start and stop characters are ASCII DC1 (octal 022, control-Q) and DC3 (octal 024, control S) respectively.

static persistent throughout a process, said of data; static data occupies the *data segment* and the *bss segment;* cf. *automatic.*

status see *exit status.*

stdio, standard input-output a collection of functions for formatted and character-by-character input-output at a higher level than the basic *read*(2), *write,* and *open*(2) operations.

stream 1. an *open file* with *buffering* superimposed by the *stdio* package. 2. a bidirectional input-output connection between a *process* and a *character device* or another process.

strip remove the *symbol table* and *relocation bits* from an *executable file* [1].

subdirectory a *directory* that appears as an *entry* [1] in another.

subshell another copy of the *shell* [1], with its own *environment* and *open files,* run as a *child process* of the shell; *shell scripts* and commands subject to `for` and

`while` or enclosed in parentheses `()` are normally run by subshells.

super-block the second *block* in a *file system* [1], which describes the allocation of space in the file system; cf. *boot block.*

super-user *userid* 0, which can access any *file* [2] regardless of *permissions* and can perform privileged *system calls,* e.g. to set the clock.

swap to move the *core image* of an executing program between main and secondary store to make room for other *processes.*

swap area the part of secondary store to which *core images* are *swapped;* the swap area is disjoint from *file systems* [1].

symbolic link an *inode* that contains the *pathname* of another. References to the symbolic link become references to the named inode.

symbol table information in an *object file* about the names of data and functions in that file; the symbol table and *relocation bits* are used by the *link editor* and by debuggers.

system call a basic operation performed by the *kernel;* see *intro*(2).

system time the accumulated time spent by a *process* in *kernel* code; cf. *user time.*

tandem a mode of the *terminal line discipline* in which the computer uses *start/stop protocol* to regulate data flow from another terminal or other computer.

terminal line discipline, teletype line discipline a *line discipline* that implements input-output features such as *erase character* and *raw mode;* see *ttyld*(4).

terminal process a program downloaded into a *layer,* often in collaboration with a *host process.* Running in a different environment, a terminal process is not a UNIX *process.*

text see *text file, segment.*

text file, ASCII file a *file,* the bytes of which are understood to be in ASCII code.

text segment see *segment.*

time the number of seconds since the **epoch**, 00:00:00 Greenwich Mean Time, January 1, 1970.

u-area a collection of data about, but not in the address space of, a *process;* the u-area is kept with the *core image* of the process.

uid see *userid.*

umask a list of *permissions* that will be denied for files *created* by a *process,* so called because the list is expressed as a *mode* bit mask.

UNIX the name of a family of systems software, not an acronym for anything; a registered trademark of AT&T, the word should be used as an adjective, e.g. 'UNIX system' or 'UNIX software'. Major genera are the research versions issued as numbered 'editions', the Berkeley software distributions (BSD), and the commercial System V.

unlink to remove an *entry* [1] from a *directory;* converse to *link* and *create.*

upas 1. a fabulous Javanese tree that poisoned everything for miles around. 2. a routing and delivery system for electronic mail.

URP universal receiver protocol used by *Datakit.*

usenet an informal, nationwide computer network based on *uucp*(1), over which 'netnews' is distributed; see *readnews*(7).

user code code that runs in a *process,* as distinct from *kernel* code.

userid, uid an integer value, usually associated with a *login name;* the userid of a *process* becomes the *owner* of files *created* by the process; it is inherited across *fork.*

user space, user address space the address space of a *process,* as distinct from *kernel space.* The address spaces of different processes are mutually disjoint.

user time the accumulated time spent by a *process* in *user code;* cf. *system time.*

utility, utility program a standard, generally useful, permanently available *program;* utilities normally inhabit directories `/bin` and `/usr/bin`.

wait to suspend running until the termination of another *process;* only a *parent process* can wait for one of its *child processes.*

window 1. same as *layer.* 2. a sublayer in *terminal processes* such as *pi*(9) or *sam*(9).

working directory same as *current directory.*

write behind to *buffer* data for writing to a *device* [2] at a convenient time without holding up the computation of a writing *process.*

write pointer same as *file pointer.*

XON-XOFF protocol same as *start/stop protocol.*

zombie a terminated *process* with a running *parent process* that has not *waited* for it.

1 COMMANDS

1 COMMANDS

NAME

intro − introduction to commands

DESCRIPTION

This section describes publicly accessible commands in alphabetic order.

The name of a particular machine at the head of the page means that the command lives there and not necessarily elsewhere. 'Local' means the same, without being specific about where.

SEE ALSO

Section (7) for databases.
Section (8) for 'hidden' commands for booting, maintenance, etc.
Section (9) for commands that involve the Teletype 5620 terminal.
How to get started, in the Introduction.

DIAGNOSTICS

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of 'normal' termination) one supplied by the program; see *exit*(2). The former byte is 0 for normal termination, the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously 'exit code', 'exit status' or 'return code', and is described only where special conventions are involved.

NAME

2500 − BVH2500 videotape recorder

SYNOPSIS

`2500` [`-lq`]

DESCRIPTION

2500 is an interpreter of commands to control a SONY BVH2500 1-inch video recorder, whose inputs and outputs have already been set up. The options are

`-l` Create a log file; useful in pursuit of bugs.

`-q` Suppress the initial status report.

Most of the commands require an intimate knowledge of the equipment. The simpler commands are described below; see the `help` command for a complete list. Times are given as `[[hrs.]min.]sec.fr` where there are 30 frames per second. The commands are

`cue` *t* Move the tape to time *t*.

`help` Produce a list of all commands.

`loop` *t0 t1 fps*
Play from *t0* through *t1* and back again at *fps* frames per second.

`play` Start playing the tape from the current frame.

`snap` *n* When in still record mode, record the current input onto the next *n* frames. A missing *n* is taken to be 1.

`status` Print some status information. The command `status status` prints all available status information.

`still` *t* Go into still record mode and cue to time *t*. The command returns before the tape transport is done; usually it must be followed by `wait`.

`still mode on|off`
Turn still mode on or off.

`stop` Stop the tape transport.

`view` *t0 t1*
Play from *t0* through *t1*. `wait` Wait for the previous tape transport command to finish.

`!` Interpret the rest of the line as a *sh*(1) command.

`#` Comment. Ignore the rest of the line.

EXAMPLES

Assuming you have already set up the video switch to feed the BVH2500, the following script will record (or rerecord) a movie starting at 2 minutes.

```
still 2.0.0
wait
!generate an image
snap 1
# repeat the last two lines as necessary
still mode off
stop
```

BUGS

The BVH2500 will misbehave if the pause between `snap`s (in still record mode) is too long, or if you record for many hours on end. The latter problem can be avoided by using scripts that run for 2 or 3 hours and sleeping for 10 minutes between scripts with the tape transport off.
Commands in the help list are (incorrectly) capitalized.

NAME

=, ==, =p, ==p − redo previous shell command

SYNOPSIS

= [*pattern*] [*substitution ...*]
== [*pattern*] [*substitution ...*]
=p [*pattern*] [*substitution ...*]
==p [*pattern*] [*substitution ...*]

DESCRIPTION

The = command provides a simple history mechanism for the shell, *sh*(1). The environment variable HISTORY, if set, names a file to which the shell appends the text of each command before execution. = searches the history file for the most recent command that matches the *pattern*, performs the *substitutions*, and executes it. The *pattern* must agree with an initial substring of the original command except for variations in spacing. If no *pattern* is specified, the most recent command is selected. If no substitution is specified, the command is executed without modification.

Substitutions have the form

old=new

specifying that the string *old* in the command is to be replaced by *new*. Substitutions are made in order and operate on the first match.

The == command is identical to =, but allows the substituted command to be edited before running. The command is printed, and a modification request is read from the terminal. Generally each character in the request specifies how to modify the character immediately above it:

`#` Delete the character.
`%` Replace the character with a space.
`^` Insert the rest of the request line before the character.
`$` Replace the characters in the command from this position on with the rest of the request line.
space or tab
Leave the character(s) unchanged.
= Must be the first and only edit character. Back up to the next most recent match in the history file and try again.
any other
This character replaces the one above it.

If the request line is longer than the command, the overhang is appended to the command.

=p and *==p* behave like = and ==, except that they print the command on their standard output instead of executing it.

NAME

adb − debugger

SYNOPSIS

`adb` [*option* ...] [*objfil* [*corfil*]]

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of *adb* cannot be used although the file can still be examined. The default for *objfil* is `a.out`. *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is `core`.

Requests to *adb* are read from the standard input and responses are to the standard output. Quit signals are ignored; interrupts cause return to the next *adb* command. The options are

`-w` Create *objfil* and *corfil* if they don't exist; open them for writing as well as reading.

`-I`*path* Directory in which to look for relative pathnames in `$<` and `$<<` commands.

In general requests to *adb* have the following form. Multiple requests on one line must be separated by `;`.

[*address*] [`,` *count*] [*command*]

If *address* is present then the current position, called 'dot', is set to *address*. Initially dot is set to 0. In general commands are repeated *count* times. Dot advances between repetitions. The default *count* is 1. *Address* and *count* are expressions.

Some formats, data sizes, and command details have different behavior on different systems. See the MACHINE DEPENDENCIES attachment for details.

Expressions

Expressions are computed with sufficient precision to address the largest possible file; generally this means a long integer. On the VAX, expressions are 32 bits; on the Cray, 64 bits.

`.` The value of dot.

`+` The value of dot incremented by the current increment.

`^` The value of dot decremented by the current increment.

`"` The last *address* typed.

integer A number in the *default radix*; see the `$d` command. Regardless of the default, the prefixes `0o` and `0O` (zero oh) force interpretation in octal radix; the prefixes `0t` and `0T` force interpretation in decimal radix; the prefixes `0x`, `0X`, and `#` force interpretation in hexadecimal radix. Thus `0o20`, `0t16`, and `#10` all represent sixteen.

integer `.` *fraction*
A floating point number.

`'`*cccc*`'` The ASCII value of one or more characters. `\` may be used to escape a `'`.

`<`*name* The value of *name*, which is either a variable name or a register name. *Adb* maintains a number of variables named by single letters or digits. The register names are those printed by the `$r` command.

symbol A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. `\` may be used to escape other characters. The value of the *symbol* is taken from the symbol table in *objfil*.

routine `.` *name*
The address of the variable *name* in the specified C routine. Both *routine* and *name* are

symbols. If *name* is omitted the value is the address of the most recently activated C stack frame corresponding to *routine*; if *routine* is omitted, the active procedure is assumed.

`(`*exp*`)` The value of the expression *exp*.

Monadic operators

`*`*exp* The contents of the location addressed by *exp* in *corfil*.

`@`*exp* The contents of the location addressed by *exp* in *objfil*.

`-`*exp* Integer negation.

`~`*exp* Bitwise complement.

`%`*exp* If *exp* is used as an address, it is in register space; see 'Addresses'.

Dyadic operators are left associative and are less binding than monadic operators.

e1`+`*e2* Integer addition.

e1`-`*e2* Integer subtraction.

e1`*`*e2* Integer multiplication.

e1`%`*e2* Integer division.

e1`&`*e2* Bitwise conjunction.

e1`|`*e2* Bitwise disjunction.

e1`#`*e2* *E1* rounded up to the next multiple of *e2*.

Commands

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands `?` and `/` may be followed by `*`; see 'Addresses' for further details.)

`?`*f* Locations starting at *address* in *objfil* are printed according to the format *f*.

`/`*f* Locations starting at *address* in *corfil* are printed according to the format *f.*

`=`*f* The value of *address* itself is printed in the styles indicated by the format *f*. (For `i` format `?` is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. If no format is given then the last format is used.

Most format letters fetch some data, print it, and advance (a local copy of) dot by the number of bytes fetched. The total number of bytes in a format becomes the *current increment.* 'Long integers' are full words, the size of an expression item: e.g. 4 bytes on the VAX, 8 bytes on the Cray. 'Short integers' are some useful shorter size: 2 byte short integers on the VAX, 2 byte parcels on the Cray.

`r`	Print short integer in the current default radix.
`R`	Print long integer in the current default radix.
`o`	Print short integer in octal.
`O`	Print long integer in octal.
`q`	Print short in signed octal.
`Q`	Print long in signed octal.
`d`	Print short in decimal.
`D`	Print long in decimal.
`x`	Print short in hexadecimal.
`X`	Print long in hexadecimal.
`u`	Print short in unsigned decimal.
`U`	Print long in unsigned decimal.
`f`	Print as a floating point number.

`F` Print double-precision floating point.
`b` Print the addressed byte in octal.
`c` Print the addressed character.
`C` Print the addressed character. Control characters are printed in the form `^`*X* and the delete character is printed as `^?`.
`s` Print the addressed characters until a zero character is reached. Advance dot by the length of the string, including the zero terminator.
`S` Print a string using the `^`*X* escape convention (see `C` above).
`Y` Print a long integer in date format (see *ctime*(3)).
`i` Print as machine instructions. This style of printing causes variables 0, (1, ...) to be set to the offset parts of the first (second, ...) operand of the instruction.
`a` Print the value of dot in symbolic form. Dot is unaffected.
`p` Print the addressed value in symbolic form. Dot is advanced by the size of a machine address (4 bytes on the VAX, 8 bytes on the Cray).
`t` When preceded by an integer tabs to the next appropriate tab stop. For example, `8t` moves to the next 8-space tab stop. Dot is unaffected.
`n` Print a newline. Dot is unaffected.
`"..."` Print the enclosed string. Dot is unaffected.
`^` Dot is decremented by the current increment. Nothing is printed.
`+` Dot is incremented by 1. Nothing is printed.
`−` Dot is decremented by 1. Nothing is printed.

newline Update dot by the current increment. Repeat the previous command with a *count* of 1.

[`?`/]`l` *value mask*
Words starting at dot are masked with *mask* and compared with *value* until a match is found. If `l` is used, the match is for a short integer; `L` matches longs. If no match is found then dot is unchanged; otherwise dot is set to the matched location. If *mask* is omitted then −1 is used.

[`?`/]`w` *value ...*
Write the short *value* into the addressed location. If the command is `W`, write a long. Option `−w` must be in effect.

[`?`/]`m` *b e f* [`?`]
New values for (*b*, *e*, *f*) in the first map entry are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. The address type (instruction or data) is unchanged in any case. If the `?` or `/` is followed by `*` then the second segment of the mapping is changed. If the list is terminated by `?` or `/` then the file (*objfil* or *corfil* respectively) is used for subsequent requests. For example, `/m?` will cause `/` to refer to *objfil*.

`>`*name* Dot is assigned to the variable or register named.

`!` A shell is called to read the rest of the line following '!'.

`$`*modifier*
Miscellaneous commands. The available *modifiers* are:

`<`*f* Read commands from the file *f*. If *f* cannot be found, try `/usr/lib/adb/`*f*. If this command is executed in a file, further commands in the file are not seen. If *f* is omitted, the current input stream is terminated. If a *count* is given, and is zero, the command will be ignored. The value of the count will be placed in variable `9` before the first command in *f* is executed.
`<<`*f* Similar to `<` except it can be used in a file of commands without causing the file to be closed. Variable `9` is saved during the execution of this command, and restored when it completes. There is a (small) limit to the number of `<<` files that can be open at once.
`>`*f* Append output to the file *f*, which is created if it does not exist. If *f* is omitted, output is returned to the terminal.

`?` Print process id, the signal which caused stopping or termination, as well as the registers. This is the default if *modifier* is omitted.

`r` Print the general registers and the instruction addressed by `pc`. Dot is set to `pc`.

`R` Like `$r`, but include miscellaneous registers such as the kernel stack pointer.

`b` Print all breakpoints and their associated counts and commands.

`c` C stack backtrace. If *address* is given then it is taken as the address of the current frame; otherwise, the current C frame pointer is used. If `C` is used then the names and (long) values of all parameters, automatic and static variables are printed for each active function. If *count* is given then only the first *count* frames are printed.

`a` Set the maximum number of arguments printed by `$c` or `$C` to *address*. The default is 20.

`d` Set the default radix to *address* and report the new value. *Address* is interpreted in the (old) current radix; `10$d` never changes the default radix. To make decimal the default radix, use `0t10$d`. A radix of zero (the initial default) is a special case; input with a leading zero is octal, that with a leading sharp-sign `#` is hexadecimal, other numbers are decimal. When the default radix is zero, the default output radix is appropriate to the machine: hexadecimal on the VAX, octal on the Cray.

`e` The names and values of all external variables are printed.

`w` Set the page width for output to *address* (default 80).

`s` Set the limit for symbol matches to *address* (default 255).

`q` Exit from *adb*.

`v` Print all non zero variables in the current radix.

`m` Print the address maps.

`k` Simulate kernel memory management.

`p` Simulate per-process memory management.

`$k` and `$p` are used for system debugging. Their details vary with machine and operating system.

: *modifier*

Manage a subprocess. Available modifiers are:

`b`*c* Set breakpoint at *address*. The breakpoint is executed *count*−1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command is omitted or sets dot to zero then the breakpoint causes a stop.

`d` Delete breakpoint at *address*.

`r` Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are enabled on entry to the subprocess.

`c`*s* The subprocess is continued. If *s* is omitted or nonzero, the subprocess is sent the signal that caused it to stop; if 0 is specified, no signal is sent. Breakpoints and single-stepping don't count as signals. Breakpoint skipping is the same as for `r`.

`s`*s* As for `c` except that the subprocess is single stepped *count* times. If a signal is sent, it is received before the first instruction is executed. If there is no current subprocess then *objfil* is run as a subprocess as for `r`. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.

`k` The current subprocess, if any, is terminated.

Variables

Adb provides a number of variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows.

`0`, `1`, ... The offset parts of the first, second, ... operands of the last instruction printed. Meaningless if the operand was a register.

`9` The count on the last `$<` or `$<<` command.

On entry the following are set from the system header in the *corfil*. If *corfil* does not appear to be a core image then these values are set from *objfil*.

`b` The base address of the data segment.
`d` The data segment size.
`e` The entry point.
`m` The 'magic' number (*a.out*(5)).
`s` The stack segment size.
`t` The text segment size.

Addresses

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by one or more quadruples (*t, b, e, f*), each mapping addresses of type *t* (instruction, data, user block) in the range *b* through *e* to the part of the file beginning at address *f*. An address *a* of type *t* is mapped to a file address by finding a quadruple of type *t*, for which $b \le a < e$; the file address is $address + f - b$. As a special case, if an instruction space address is not found, a second search is made for the same address in data space.

Typically, the text segment of a program is mapped as instruction space, the data and bss segments as data space. If *objfil* is an *a.out*, or if *corfil* is a core image or process file, maps are set accordingly. Otherwise, a single 'data space' map is set up, with *b* and *f* set to zero, and *e* set to a huge number; thus the entire file can be examined without address translation.

The `?` and `/` commands attempt to examine instruction and data space respectively. `?*` tries for data space (in *objfil*); `/*` accesses instruction space (in *corfil*).

Registers in process and core images are a special case; they live in a special 'register' address space. The contents of register 0 are located at address `%0`; register 1 at `%4` (if registers are 4 bytes long); and so on. `%` addresses are mapped to the registers for the 'current frame,' set by local variable references, and reset to the outermost frame (the 'real' registers) whenever a process runs or a stack trace is requested.

Simulated memory management translations (the `$k` and `$p` commands) are done before the mapping described above.

FILES

`a.out`
`core`
`/usr/lib/adb/*` parameter files
`/proc/*`

SEE ALSO

cin(1), *pi*(9.1), *nm*(1), *proc*(4), *a.out*(5), *bigcore*(1)
J. F. Maranzano and S. R. Bourne, 'A Tutorial Introduction to ADB' in Bell Laboratories, *UNIX Programmer's Manual*, Volume 2, Holt, Rinehart and Winston (1984)

DIAGNOSTICS

'Adb' when there is no current command or format. Exit status is 0, unless last command failed or returned nonzero status.

BUGS

Either the explanation or the implementation of register variables is too complex and arcane.

MACHINE DEPENDENCIES

PDP-11

Short integers (printed by `r` format) are 2 bytes; long integers (printed by `R` format) are 4 bytes. Addresses printed by `a` format are 2 bytes.

Register variables match the hardware in the obvious way: `r0` is at address `%0`, `r1` at `%2`, and so on.

The default output radix is octal.

`$k` and `$p` are unimplemented.

VAX

Short integers are 2 bytes, long integers are 4 bytes, addresses are 4 bytes.

Register variables match the hardware in the obvious way: r0 is at address `%0`, r1 at `%4`, and so on.

The default output radix is hexadecimal.

`$k` sets the system base register pointer to *address*. System space addresses are thereafter mapped according to the system page table starting at that physical address. An *address* of zero turns off mapping.

`$p` sets the process control block pointer to *address*; user space addresses are thereafter translated according to the user page tables described by the PCB. Kernel mapping must already be in effect. *Address* may be a physical address (that of the PCB) or the system space virtual address of a page table entry pointing to the PCB (the number stored in *p_addr*). If *address* is zero, user mapping is turned off; addresses less than 0x80000000 will be treated as physical addresses.

The command `$<crash` will initialize registers and mapping from a kernel crash dump.

Cray

Short integers are 2 bytes; long integers are 8 bytes. Addresses are 8 bytes.

Registers are funny, and yet to be described.

The default output radix is octal.

`$k` and `$p` are unimplemented.

NAME

apply, pick − repeatedly apply a command; select arguments

SYNOPSIS

`apply` [`-a`*c*] [−*n*] *command arg ...*

`pick` [*arg ...*]

DESCRIPTION

Apply runs the named *command* on each argument *arg* in turn. Normally arguments are chosen singly; the optional number *n* specifies the number of arguments to be passed to *command.* If *n* is zero, *command* is run without arguments once for each *arg.* Character sequences of the form `%`*d* in *command,* where *d* is a digit from 1 to 9, are replaced by the *d*th following unused *arg.* If any such sequences occur, *n* is ignored, and the number of arguments passed to *command* is the maximum value of *d* in *command.* The character `%` may be changed by the `-a` option.

Pick writes each argument to the standard error and reads a reply. If the reply is `y`, the argument is echoed to the standard output; if the reply is `q`, *pick* exits without reading any more arguments; there is no output for any other response. If there are no arguments, lines of the standard input are taken instead.

EXAMPLES

`apply echo *`
Time-consuming way to do `ls`.

`apply -2 cmp a1 b1 a2 b2`
Compare the ‘a’ files to the ‘b’ files.

`wc -l 'pick *.[ch]'`
Interactively select ‘.c’ and ‘.h’ files and count the lines in each.

`apply "wc -l %1" 'pick *.[ch]'`
Same, but use a separate process to count each file.

SEE ALSO

sh(1)

BUGS

There is no way to pass a literal `%2` if `%` is *apply’s* argument expansion character.

NAME

apsend − send troff output to phototypesetter

SYNOPSIS

`apsend` [*options*] [*file* ...]

DESCRIPTION

Apsend sends *troff*(1) output from the named files or from the standard input to the Murray Hill computer center for high-quality typesetting.

The options, which need only be spelled far enough to be unique, are

`account=`*xx* comp center account number (default from password file)

`bin=`*xx* comp center bin number (default from password file)

`mailto=`*xx* mailing instructions, up to 28 characters, instead of comp center bin

`comment=`*xx* up to 30 characters, for `APSLOG` file entry (default value is *file* or `pipe.end`)

`device=imagen`
Print on laser printer instead of phototypesetter.

FILES

`/usr/lib/apsend/*`
`APSLOG` record of apsend activity

SEE ALSO

troff(1), *lp*(1), *font*(5)

BUGS

Do not be misled by the historical name of this program. The correct *troff* device selection is `-Tpost`, which is fortunately the default.

NAME

ar, ranlib − archive and library maintainer

SYNOPSIS

`ar` *key* [*posname*] *afile* [*file* ...]

`ranlib` *archive* ...

DESCRIPTION

Ar maintains groups of files combined into a single archive file, *afile.* If it is not to be modified, the archive may be read from standard input, indicated by the name `.` The main use of a *ar* is to create and update library files for the loader *ld*(1). It can be used, though, for any similar purpose.

Key is one character from the set `drqtpmx,` optionally concatenated with one or more of `vuaibcl.` The *files* are constituents of the archive *afile.* The meanings of the *key* characters are:

`d` Delete *files* from the archive file.

`r` Replace *files* in the archive file. Optional modifiers are
- `u` Only replace files with modified dates later than that of the archive.
- `a` Place new files after *posname* in the archive rather than at the end.
- `b` or `i` Place new files before *posname* in the archive.

`q` Quick. Append *files* to the end of the archive without checking for duplicates. Avoids quadratic behavior in `for i in *.o; do ar r lib.a $i; done`.

`t` List a table of contents of the archive. If names are given, only those files are listed.

`p` Print the named files in the archive.

`m` Move the named files to the end or elsewhere, specified as with `r`.

`x` Extract the named files. If no names are given, all files in the archive are extracted. In neither case does `x` alter the archive file.

`v` Verbose. Give a file-by-file description of the making of a new archive file from the old archive and the constituent files. With `p`, precede each file with a name. With `t,` give a long listing of all information about the files, somewhat like a listing by *ls*(1), showing

```
mode uid/gid size date name
```

`c` Create. Normally *ar* will create a new archive when *afile* does not exist, and give a warning. Option `c` discards any old contents and suppresses the warning.

`l` Local. Normally *ar* places its temporary files in the directory `/tmp`. This option causes them to be placed in the local directory.

Ranlib makes a table-of-contents file `__.SYMDEF` for each library *archive.* With this table the loader *ld*(1) will extract files as if it were repeatedly invoked until no more subroutines can be found.

EXAMPLES

`ar cr lib.a *.o; ranlib lib.a`
Replace the contents of library `lib.a` with the object files in the current directory.

`pcat old.a.z | ar t -`
List the contents of an archived and compressed collection of old files; see *pack*(1).

FILES

`/tmp/v*` temporaries

SEE ALSO

ld(1), *ar*(5)

BUGS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.
Ld(1) warnings that a library is newer than its table of contents happen when a library is copied.

NAME

as − assembler

SYNOPSIS

`as` [*option* ...] [*name* ...]

DESCRIPTION

As assembles the named files, or the standard input if no file name is specified. The options are:

`-d`*n* Specifies the number of bytes *n* (1, 2, or 4) to be assembled for offsets which involve forward or external references, and which have sizes unspecified in the assembly language. Default is `-d4`.

`-L` Save defined labels that begin with `L`, which are normally discarded to save space in the resultant symbol table. The compilers generate such temporary labels.

`-V` Use virtual memory for intermediate storage, rather than a temporary file.

`-W` Do not complain about errors.

`-J` Use long branches to resolve jumps when byte-displacement branches are insufficient. This must be used when a compiler-generated assembly contains branches of more than 32K bytes.

`-R` Make initialized data segments read-only, by concatenating them to the text segments. This obviates the need to run editor scripts on assembly code to make initialized data read-only and shared.

`-t` Specifies a directory to receive the temporary file, other than the default `/tmp`.

`-o`*obj* Place output in file *obj*. Default is `a.out`.

All undefined symbols in the assembly are treated as global.

FILES

`/tmp/as*` default temporary file
`a.out` default object file

SEE ALSO

ld(1), *nm*(1), *adb*(1), *pi*(9.1), *a.out*(5)
J. F. Reiser and R. R. Henry ‘Assembler Reference Manual’, *Unix Programmer’s Manual, Seventh Edition, Virtual* 1980, Volume 2C (Berkeley)

BUGS

`-J` should be eliminated; the assembler should automatically choose among byte, word and long branches.

NAME

ascii − interpret ASCII characters

SYNOPSIS

`ascii` [`-oxdb`*n*] [`-nct`] [`-e`] [*text*]

DESCRIPTION

Ascii prints the ASCII values corresponding to characters and *vice versa*. The values are interpreted in a settable numeric base; `-o` specifies octal (the default), `-d` decimal, `-x` hexadecimal, and `-b`*n* base *n.*

With no arguments, *ascii* reproduces `/usr/pub/ascii` in the specified base. Characters of *text* are converted to their ASCII values, one per line. If, however, the first *text* argument is a valid number in the specified base, conversion goes the opposite way. Control characters are printed as they appear in `/usr/pub/ascii`. Other options are:

`-n` Force numeric output.

`-c` Force character output.

`-t` Convert from numbers to running text; do not interpret control characters or insert newlines.

`-e` Interpret remaining arguments as *text.*

EXAMPLES

`ascii -d`
Print the ASCII table base 10.

`ascii p`
Print the octal value of ‘p’.

`ascii 160`
Show which character is octal 160.

SEE ALSO

ascii(6)

NAME

at – execute commands at a later time

SYNOPSIS

`at` [`-r`] *time* [*day*] [*file*]

`at -l`

DESCRIPTION

At squirrels away a copy of the named *file* (standard input default) to be used as input to *sh*(1) at a specified later time. A *cd* command to the current directory is inserted at the beginning, followed by assignments to all environment variables. When the script is run, it uses the userid and groupid of the creator of the copy.

The *time* is 1 to 4 digits, with an optional following `A`, `P`, `N` or `M` for AM, PM, noon or midnight. One and two digit numbers are taken to be hours, three and four digits to be hours and minutes. If no letters follow the digits, a 24 hour clock time is understood.

The optional *day* is either a month name followed by a day number, or a day of the week; if the word `week` follows, invocation is moved seven days further off. Names of months and days may be recognizably truncated. A year number, spelled out in full, may follow the month.

The options are

`-r` Remove the specified activity.

`-l` List all activities scheduled for this user.

At programs are executed by periodic execution of `/usr/lib/atrun` from *cron*(8). The granularity of *at* depends upon how often *atrun* is executed.

The standard output and standard error files are lost unless redirected.

EXAMPLES

```
at 0800 dec 24
echo ho ho ho | mail claus
```

```
at -r `at -l`
```

Remove a scheduled activity.

FILES

`/usr/lib/atrun`	
`/usr/spool/at/`*yy*`.`*ddd*`.`*hhmm*`.*`	activity for year, day, hour
`/usr/spool/at/lasttimedone`	last *hhmm*
`/usr/spool/at/past`	activities in progress

SEE ALSO

calendar(1), *pwd*(1), *sleep*(1), *cron*(8)

BUGS

Due to the granularity of the execution of *atrun,* there may be bugs in scheduling things almost exactly 24 hours into the future.

NAME

awk − pattern-directed scanning and processing language

SYNOPSIS

`awk` [`−F`*fs*] [`−v` *var=value*] ... [`−f`] *prog* [*file* ...]

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns in *prog*, which may appear as one literal argument or as one or more file names each preceded by `−f`. With each pattern may be associated an action to be performed when a line of a *file* matches the pattern. Each line is matched against the pattern portion of every pattern-action statement in order; the associated action is performed for each matched pattern. The file name − means the standard input. Any *file* of the form *var=value* is treated as an assignment, not a filename, and is executed at the time it would have been opened if it were a filename. Option `−v` designates an assignment to be done before *prog* is executed.

An input line is made up of fields separated by white space, or by regular expression `FS`. The fields are denoted `$1`, `$2`, ..., while `$0` refers to the entire line.

A pattern-action statement has the form

pattern `{` *action* `}`

A missing `{` *action* `}` means print the line; a missing pattern always matches. Pattern-action statements are separated by newlines or semicolons.

An action is a sequence of statements. A statement can be one of the following:

```
if( expression ) statement [ else statement ]
while( expression ) statement
for( expression ; expression ; expression ) statement
for( var in array ) statement
do statement while( expression )
break
continue
{ [ statement ... ] }
expression                      # commonly var = expression
print [ expression-list ] [ > expression ]
printf format [ , expression-list ] [ > expression ]
return [ expression ]
next                            # skip remaining patterns on this input line
delete array[ expression ]      # delete an array element
exit [ expression ]             # exit immediately; status is expression
```

Statements are terminated by semicolons, newlines or right braces. An empty *expression-list* stands for `$0`. String constants are quoted `" "`, with the usual C escapes recognized within. Expressions take on string or numeric values as appropriate, and are built using the operators `+ − * / % ^` (exponentiation), and concatenation (indicated by white space). The operators `! ++ −− += −= *= /= %= ^= > >= < <= == != && || ?:` are also available in expressions. Variables may be scalars, array elements (denoted *x*`[`*i*`]`) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. Multiple subscripts such as `[i,j,k]` are permitted; the constituents are concatenated, separated by the value of `SUBSEP`.

The `print` statement prints its arguments on the standard output (or on a file if >*file* or >>*file* is present or on a pipe if |*cmd* is present), separated by the current output field separator, and terminated by the output record separator. *file* and *cmd* may be literal names or parenthesized expressions; identical string values in different statements denote the same open file. The `printf` statement formats its expression list according to the format (see *printf*(3)). The built-in function `close(`*expr*`)` closes the file or pipe *expr*.

The mathematical functions `exp`, `log`, `sqrt`, `sin`, `cos`, and `atan2` are built in. Other built-in functions:

`length` the length of its argument taken as a string, or of `$0` if no argument.
`rand` random number on (0,1)
`srand` sets seed for `rand` and returns the previous seed.
`int` truncates to an integer value
`substr(`*s*`, `*m*`, `*n*`)`
the *n*-character substring of *s* that begins at position *m* counted from 1.
`index(`*s*`, `*t*`)`
the position in *s* where the string *t* occurs, or 0 if it does not.
`match(`*s*`, `*r*`)`
the position in *s* where the regular expression *r* occurs, or 0 if it does not. The variables `RSTART` and `RLENGTH` are set to the position and length of the matched string.
`split(`*s*`, `*a*`, `*fs*`)`
splits the string *s* into array elements *a*`[1]`, *a*`[2]`, ..., *a*`[`*n*`]`, and returns *n*. The separation is done with the regular expression *fs* or with the field separator `FS` if *fs* is not given.
`sub(`*r*`, `*t*`, `*s*`)`
substitutes *t* for the first occurrence of the regular expression *r* in the string *s*. If *s* is not given, `$0` is used.
`gsub` same as `sub` except that all occurrences of the regular expression are replaced; `sub` and `gsub` return the number of replacements.
`sprintf(`*fmt*`, `*expr*`, ... )`
the string resulting from formatting *expr* ... according to the *printf*(3) format *fmt*
`system(`*cmd*`)`
executes *cmd* and returns its exit status

The ''function'' `getline` sets `$0` to the next input record from the current input file; `getline` <*file* sets `$0` to the next record from *file*. `getline` *x* sets variable *x* instead. Finally, *cmd* `|` `getline` pipes the output of *cmd* into `getline`; each call of `getline` returns the next line of output from *cmd*. In all cases, `getline` returns 1 for a successful input, 0 for end of file, and −1 for an error.

Patterns are arbitrary Boolean combinations (with `!` `||` `&&`) of regular expressions and relational expressions. Regular expressions are as in *egrep*; see *gre*(1). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions, using the operators `~` and `!~`. /*re*/ is a constant regular expression; any string (constant or variable) may be used as a regular expression, except in the position of an isolated regular expression in a pattern.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines from an occurrence of the first pattern though an occurrence of the second.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression
expression `in` *array-name*
`(`*expr*`,`*expr*`,...)` `in` *array-name*

where a relop is any of the six relational operators in C, and a matchop is either `~` (matches) or `!~` (does not match). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns `BEGIN` and `END` may be used to capture control before the first input line is read and after the last. `BEGIN` and `END` do not combine with other patterns.

Variable names with special meanings:

`FS` regular expression used to separate fields; also settable by option −`F`*fs*.

NF	number of fields in the current record
NR	ordinal number of the current record
FNR	ordinal number of the current record in the current file
FILENAME	the name of the current input file
RS	input record separator (default newline)
OFS	output field separator (default blank)
ORS	output record separator (default newline)
OFMT	output format for numbers (default %.6g)
SUBSEP	separates multiple subscripts (default 034)
ARGC	argument count, assignable
ARGV	argument array, assignable; non-null members are taken as filenames
ENVIRON	array of environment variables; subscripts are names.

Functions may be defined (at the position of a pattern-action statement) thus:

```
function foo(a, b, c) { ...; return x }
```

Parameters are passed by value if scalar and by reference if array name; functions may be called recursively. Parameters are local to the function; all other variables are global. Thus local variables may be created by providing excess parameters in the function definition.

EXAMPLES

```
length > 72
```
Print lines longer than 72 characters.

```
{ print $2, $1 }
```
Print first two fields in opposite order.

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```
Same, with input fields separated by comma and/or blanks and tabs.

```
      { s += $1 }
END   { print "sum is", s, " average is", s/NR }
```
Add up first column, print sum and average.

```
/start/, /stop/
```
Print all lines between start/stop pairs.

```
BEGIN {     # Simulate echo(1)
      for (i = 1; i < ARGC; i++) printf "%s ", ARGV[i]
      printf "\n"
      exit }
```

SEE ALSO

gre(1), *lex*(1), *sed*(1)

A. V. Aho, B. W. Kernighan, P. J. Weinberger, *The AWK Programming Language*, Addison-Wesley, 1988.

BUGS

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate "" to it.

The scope rules for variables in functions are a botch; the syntax is worse.

NAME

backup − backup and recover files

SYNOPSIS

`backup recover` [*option* ...] *file* ...

`backup grep` [*option* ...] *pattern* ...

`backup fetch` [*option* ...] [*file* ...]

`backup stats` [*option* ...]

`backup backup` [*file* ...]

`backup munge`

`backup mount` [*option* ...] *mountpt*

DESCRIPTION

The *backup* programs save and restore archival copies of files in an optical disk store on a central system (see *backup*(5)). Backup occurs automatically daily (see *backup*(8)) and upon specific request via *backup backup*. *Backup grep* shows backup copy names for specific files, and *backup fetch* restores data from specific backup copies. *Backup recover* is a combination of these two; it fetches the most recent copy. All the *backup* programs describe their options when presented with a bad option such as `-?`.

Backup recover retrieves *files* by name. The names should be full pathnames rooted at `/n/`; if not, *backup* tries to guess names that begin with `/n/`. Directories should be recovered before their contents. Regular files that are linked together will stay linked if they are recovered together. The options for *recover* are:

`-o` *dir* The argument is restored as an entry in the directory *dir*.

`-v` Verbose (enforced).

`-F` Restore directories as files containing a null-terminated list of element names.

`-r` Recursively recover any subdirectories.

`-d` Create any missing intermediate directories.

`-D`*old*`=`*new*
Replace the prefix *old* of the original filename with *new* to form the new output filename.

`-m` The names are backup copy names, as determined from *backup grep*, not original filenames.

`-f`*device*
Use *device* rather than `/dev/worm0` for the WORM. *Device* may be on another machine: *machine*`!`*device*. An initial `w` implies a WORM device; a `j` implies a jukebox. A numeric *device* means `/dev/worm`*device*.

`-e` Cause the *worm fetch* server on the backup system to terminate gracefully.

`-i` Append `.`*n* to the output name for each file where *n* is an increasing integer. This is useful for recovering multiple copies of the same file.

A diagnostic like `need disk backup2a` means you need to mount the A side of the cartridge labeled `backup2`.

Backup grep searches for names of backed up files that match the strings *patterns*. If the pattern is a literal (no `-e`) that looks like a filename, it reports the filename catenated with `//` and the time of the most recent backup copy. If the pattern is a literal that looks like the output under option `-d`, it reports the name of the corresponding backup copy. The options are:

`-d` Print file change times (`ctime`, see *stat*(2)) as integers rather than as dates.

`-e` Interpret *patterns* as regular expressions given in the notation of *regexp*(3). Warning: this option can execute extremely slowly; it is almost always better to use *gre*(1) on `/usr/backup/filenames` on the backup machine; see *backup*(5).

`-a` Print all names in the database.

`-V` Treat *pattern* as a literal filename and list all versions of the file.

`-<`*n* Only list entries with a date less than or equal to *n*. If *n* is not a simple integer date, it is interpreted as by *timec*(3).

`->`*n* Only list entries with a date greater than or equal to *n*.

`-D` Print the most recent entry for every file name starting with *pattern*, taking into account any cutoff date, but turning off option `-e`.

Backup fetch takes from its arguments or from standard input backup copy names as reported by *backup grep* (such as `v2345/987`) and restores the corresponding files. It accepts the same options as *backup recover* except `-m`; `-v` is really optional. Irrelevant prefixes are stripped from backup copy names. Thus the output of the *backup grep* command can be used directly.

Backup stats provides statistics about the files backed up. By default, it looks for all systems and all users and gives a grand total. The options are

`-i` Give information per system or user rather than a total.

`-s` *systems*

`-u` *users* With option `-i`, restrict the total to the systems or users named in comma-separated lists. The name `*` expands to all systems or all users.

`-d` Print average number of files and bytes for the last 1 day, 7 days and 30 days.

Backup backup backs up files. If no file names are given, they are taken from standard input. File names are interpreted as in *backup recover*. The files are safely on the backup system when the command exits but will normally take a day to get into the backup database.

Backup munge causes the backup system to process any received files. When this terminates (assuming no errors), the files have been put onto backup media and have been absorbed into the database.

Backup mount is an experimental way to access backed up files. The specified part of the backup files (set by `-D`*root* or `/` by default) is mounted at *mountpt*. There is one option

`-d` *date* Make the mounted hierarchy reflect the state at the given date. The mounting can be reversed with *umount*; see *mount*(8).

EXAMPLES

```
backup stats -i -s '*'
```
Get totals for all systems.

```
backup fetch `backup grep -d \`backup grep -d /n/bowell/etc/passwd\``
```
What *backup recover* does for you.

```
backup recover /n/coma/usr/rob/fortunes
cd /n/coma/usr/rob; backup recover fortunes
```
Two ways to get the latest available copy of `/n/coma/usr/rob/fortunes`.

```
backup grep -V /n/coma/usr/rob/fortunes
```
List all available copies of `/n/coma/usr/rob/fortunes` with their dates.

```
backup recover -m -o /tmp /n/wild/usr/backup/v/v919/678
backup recover -m -o /tmp v919/678
```
Two ways to recover a specific backup copy and place the result in `/tmp`. `/n/wild/usr/backup/v/v919/678` is the name of the backup copy; the file will be restored to its home machine, not to `wild`.

```
backup grep -V /n/coma/usr/rob/fortunes | backup fetch -i -o .
```
Recover all the versions of the fortunes file into `fortunes.1`, `fortunes.2`, ... in the current directory.

FILES

`/usr/lib/backup` home of all datafiles and executables (on client machines)

SEE ALSO

worm(8), *backup*(5), *backup*(8)

BUGS

Recovery via symbolic links may not work; use the non-linked pathname.

NAME

basename, dirname − strip filename affixes

SYNOPSIS

`basename` *string* [*suffix*]

`dirname` *string*

DESCRIPTION

These functions split off useful parts of a pathname; they are typically used inside substitution marks `` ` ` `` in shell scripts.

Basename deletes any prefix ending in `/` and the *suffix*, if present in *string*, from *string*, and prints the result on the standard output.

Dirname places on standard output the name of the directory in which a file named *string* would nominally be found. The calculation is syntactic and independent of the contents of the file system.

EXAMPLES

```
cc $1 -o `basename $1 .c`
```
Compile into `file`, where `$1` is `file.c` or `dir/file.c`.

```
cc $1 -o `dirname $1`/`basename $1 .c`
```
Compile `dir/file.c` into `dir/file`.

SEE ALSO

sh(1)

NAME

bc − arbitrary-precision arithmetic language

SYNOPSIS

`bc [ -c ] [ -l ] [` *file* ... `]`

DESCRIPTION

Bc is an interactive processor for a language that resembles C but provides arithmetic on numbers of arbitrary length with up to 100 digits right of the decimal point. It takes input from any files given, then reads the standard input. The `-l` argument stands for the name of an arbitrary precision math library. The following syntax for *bc* programs is like that of C; *L* means letter `a-z`, *E* means expression, *S* means statement.

Lexical

comments are enclosed in `/* */`

newlines end statements

Names

simple variables: *L*

array elements: *L* `[` *E* `]`

The words `ibase`, `obase`, and `scale`

Other operands arbitrarily long numbers with optional sign and decimal point.

`(` *E* `)`

`sqrt(` *E* `)`

`length(` *E* `)`

number of significant decimal digits

`scale(` *E* `)`

number of digits right of decimal point

L `(` *E* `,` ... `,` *E* `)`

Operators

`+ - * / % ^` (`%` is remainder; `^` is power)

`++ --` (prefix and postfix; apply to names)

`== <= >= != < >`

`= += -= *= /= %= ^=`

Statements

E

`{` *S* `;` ... `;` *S* `}`

`if (` *E* `)` *S*

`while (` *E* `)` *S*

`for (` *E* `;` *E* `;` *E* `)` *S*

null statement

`break`

`quit`

`"`text`"`

Function definitions

`define` *L* `(` *L* `,` ... `,` L `){`

`auto` *L* `,` ... `,` *L*

S `;` ... `;` *S*

`return (` *E* `)`

`}`

Functions in `-l` math library

`s(`*x*`)` sine

`c(`*x*`)` cosine

`e(`*x*`)` exponential

`l(`*x*`)` log

`a(`*x*`)` arctangent
`j(`*n*`,`*x*`)` Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Text in quotes, which may include newlines, is also printed. Either semicolons or newlines may separate statements. Assignment to `scale` influences the number of digits to be retained on arithmetic operations in the manner of *dc*(1). Assignments to `ibase` or `obase` set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. Automatic variables are pushed down during function calls. In a declaration of an array as a function argument or automatic variable empty square brackets must follow the array name.

Bc is actually a preprocessor for *dc*(1), which it invokes automatically, unless the `-c` (compile only) option is present. In this case the *dc* input is sent to the standard output instead.

EXAMPLES

Define a function to compute an approximate value of the exponential. Use it to print 10 values. (The exponential function in the library gives better answers.)

```
scale = 20
define e(x){
	auto a, b, c, i, s
	a = 1
	b = 1
	s = 1
	for(i=1; 1==1; i++){
		a = a*x
		b = b*i
		c = a/b
		if(c == 0) return(s)
		s = s+c
	}
}
for(i=1; i<=10; i++) e(i)
```

FILES

`/usr/lib/lib.b` mathematical library

SEE ALSO

dc(1), *hoc*(1)

BUGS

No `&&`, `||`, or `!` operators.
A `for` statement must have all three `E`s.
A `quit` is interpreted when read, not when executed.

NAME

bcp – reformat black-and-white picture files

SYNOPSIS

`bcp` [*option* ...] [*file1* [*file2*]]

DESCRIPTION

Bcp copies black-and-white (B&W) image *file1* to *file2*, optionally changing the file format and transforming the image. If *file1* is a directory name, then every leaf of its file tree is processed in turn; and, in this case, if *file2* also is specified, it is made the root directory of an isomorphic tree of output files. If *file2* is not specified, all output is catenated to stdout.

Bcp can copy among all the B&W *picfile*(5) formats, and some others. The default output format is `TYPE=ccitt-g4`. Image transformations include trimming, translation, scaling, and rotation, performed in that order (not in argument order).

Input files in *picfile*(5) format must begin with an ASCII `TYPE=`*type* header line. *Types* supported both as input and output are:

`dump` One byte/pixel. `NCHAN=1` is required. On input, the grey pixel values are thresholded to B&W; see option `-T`. On output, black becomes 0 and white 255.

`bitmap` One bit/pixel. Essentially Sun rasterfile format, with a *picfile*(5) header replacing the Sun binary header.

`ccitt-g4` CCITT Group 4 FAX encoding, strongly compressive on printed text. Also, `ccitt-g31` (Group 3, 1-dim) and `ccitt-g32` (Group 3, 2-dim; see `-k`).

Other supported *types* are:

`binary` One bit/pixel encoding; obsolescent, but needed for old image archives. Both input and output.

`rle` Fast run-length encoding; obsolescent, but needed for old image archives. Input or output, but not both.

`pico` Same as `dump`. Input only.

`cdf` 'Compound document format', used in AT&T FAX Connection product. Input only. Only the first of multiple pages is read.

Other formats not using a `TYPE=`*type* header, are: *bitfile*(9.5) format; tiff 5.0 format; PostScript bitmap format (output only); and Sun rasterfile format. Input tiff files may be encoded using the Group 3 or Group 4 schemes, LZW algorithm, modified Huffman encoding, Apple PackBits, or uncompressed. Sun rasterfiles may be encoded using no compression, or the byte-length encoding scheme.

The options are:

`-B[io]` Read/write *bitfile*(9.5) format (no `TYPE=`*type* header).

`-Fc` Write tiff format, compression scheme c, where c is g3 or g31 (1-dim Group 3), g32 (2-dim Group 3), g4 (Group 4), L (LZW compression), P (Apple PackBits), or N (no compression).

`-M` Write `TYPE=bitmap` format.

`-P` Write Postscript format.

`-R`*x*`,`*y* Force output resolutions to *x*,*y* (pixels/inch). If `,`*y* is missing, it is taken to be the same as *x*. Overrides `-x`*x*`,`*y*`.` Requires a `RES=`*x y* line in the header (but, see `-Z`).

`-R=` Force the output resolution to be equal to the greater of the input resolutions.

`-S` Write Sun rasterfile format (standard encoding).

`-T`*t* Threshold. When reading `TYPE=dump`, assign black to grey levels less than *t*, and white to others. Default: `-T128`.

`-Z`*x*`,`*y* Force input `RES=`*x y*.

`-b` Write `TYPE=binary` format.

`-g4`
`-4` Write `TYPE=ccitt-g4` format. Similarly, `-g31` or `-31` and `-g32` or `-32`.

`-k`*n* Set the 'k' for `ccitt-g32` encoding on output (default `-k4`).

`-o`*x*`,`*y* Offset (translate) the image by *x,y* pixels. The width and height of the picture are not changed.

`-p` Write `TYPE=dump NCHAN=1` format. Map black to 0, white to 255.

`-r` Write `TYPE=rle` format.

`-tl` Rotate the image to bring the left edge of the page to the top. Set top-left corner of the rotated image at the top-left corner of the image.

`-t`*d* Rotate the image *d* degrees counterclockwise about its center. *d* is a real number.

`-w`*l,t,r,b* Specify window (trim the image): *l,t* is the left-top corner and *r,b* the right-bottom corner measured in pixels. If the new margins are outside the original picture, the new area is set to white. An argument given as `%` leaves the edge unchanged.

`-x`*x*`,`*y* Expand/contract (scale) the image, by real factors *x* and *y*. If `,`*y* is missing, *y* is taken to be the same as *x*. May be overridden by `-R`*x*`,`*y*. Requires a `RES=`*x y* line in the header (but, see `-Z`).

SEE ALSO

cscan(1), *imscan*(1), *ocr*(1), *pico*(1), *picfile*(5)
CCITT facsimile coding standards Rec. T.4(1988) and T.6(1988).

BUGS

Concatenated pages are supported, but only if each new page has a complete header.
Scaling is accomplished by naive replication/deletion of pixels.
Rotation by small angles exhibits aliasing effects, and is slow.
Rotations `-tr` and `-tb` are unfinished.
CCITT FAX 'uncompressed' (or, 'transparent') mode is not implemented.
Postscript output is useful only for small images.
`WINDOW=`*l t r b* where *l* or *t* is non-zero may not be handled correctly for every combination of file types.
tiff LZW compression may not be working properly (input and output).
`TYPE=rle` can't be both input and output.
Should be merged with T. Duff's *pcp*.

NAME

bigcore, coreid − permit big core images, identify source of image

SYNOPSIS

`bigcore` *command*

`coreid` [*file*]

DESCRIPTION

Bigcore calls the named *command* with no restriction on the size of core images. By default, no core image will be written when a program aborts if it would be larger than a megabyte.

Coreid reads the file `core`, or the specified core image *file*, and prints on the standard output the argument list of the program that produced the core image.

FILES

`core`

SEE ALSO

core(5)

BUGS

Coreid only works for core images from C and Fortran programs.
The arguments shown are those at the time of the dump, not at invocation of the program.

NAME

bundle − collect files for distribution

SYNOPSIS

`bundle` *file* ...

DESCRIPTION

Bundle writes on its standard output a shell script for *sh*(1) that, when executed, will recreate the original *files*. Its main use is for distributing small numbers of text files by *mail*(1).

Although less refined than standard archives from *ar*(1), *cpio*(1), or *tar*(1), a *bundle* file is self-documenting and complete; little preparation is required on the receiving machine.

EXAMPLES

`bundle makefile *.[ch] | mail elsewhere!mark`

Send a makefile to Mark together with related `.c` and `.h` files. Upon receiving the mail, Mark may save the file sans postmark, say in `gift/horse`, then do

```
cd gift; sh horse; make
```

SEE ALSO

ar(1), *cpio*(1), *tar*(1), *mail*(1)

BUGS

Bundle will not create directories and is unsatisfactory for non-ASCII files.
Beware of gift horses.

NAME

CC, cfront − C++ compiler

SYNOPSIS

`CC` [*option ...*] *file ...*

`cfront` [*option ...*] *file ...*

DESCRIPTION

CC compiles and links C++ programs in the manner of *cc*(1). It handles source files with names ending in `.c`, assembler files in `.s`, and object files in `.o`. Various passes of the compiler can be substituted via environment variables listed under 'FILES'. Options include those of *cc*(1) except `−B` and `−t`, those of *ld*(1), those of *cfront*, and in addition

`−F` Run only the macro preprocessor *cpp*(8) and *cfront* on the named `.c` files, and send the result to the standard output.

`−.`*suffix*
Instead of the standard output, place `−E` and `−F` output in files whose name is that of the source with `.`*suffix* substituted for `.c`.

Cfront reads C++ code (without preprocessing) from the standard input and writes equivalent C code on the standard output. The options are

`+d` Don't expand inline functions.

`+x`*file* Take size and alignment information from *file* for cross compiling.

`+e0`
`+e1` Make external declarations (+e0) or definitions `(+e1)` for virtual function tables. These tables may appear as static data in every compilation; the options are intended to save redundant space.

`+a0` Produce classic C output (default).

`+a1` Produce ANSI C output. If this option is used with *CC*, then an ANSI C compiler such as *lcc* must be specified in environment variable `ccC`.

`+f`*name*
Use *name* to identify the source file in diagnostics.

`+L` Produce ANSI standard `#line` directives instead of `#`*number*.

FILES

`cppC=/lib/cpp`	C preprocessor
`cfrontC=/usr/bin/cfront`	C++ translator
`ccC=/bin/cc`	C compiler
`munchC=/usr/lib/munch`	linker postprocessor for static initialization
`/usr/lib/libC.a`	C++ library
`/usr/include/CC`	standard directory for C++ `#include` files

Other files as in *cc*

SEE ALSO

cc(1), *ld*(1)
B. Stroustrup, *The C++ Programming Language*, Addison-Wesley, 1986
B. Stroustrup, *C++ Reference Manual*, AT&T Bell Laboratories, May 1989

NAME

calendar − reminder service

SYNOPSIS

`calendar` [−] [*n*]

DESCRIPTION

Calendar consults calendar files and prints out lines that contain today's date or any date up through the *n*th working day hence (*n*=1 by default). Most American-style month-day dates such as `Aug. 19`, `august 19`, `8/19`, etc., are recognized, but not `19/8`. The symbol `*` denotes every month as in `*` 19 or `*/19`. A year may follow the day, as in `August 19 86`, `8/19/86`, or `Aug. 19, 1986`.

By default, the program consults the file `calendar` in directory `$HOME` (see *sh*(1)), or in the current directory if a home directory is not known. Other calendar files to be consulted may be specified by calendar lines in one of the forms

`#include` *file*
`#include` *machine*`!`*file*

where *file* is the name of some other calendar and *machine* is the name of a machine or service accessible via *con*(1).

When the optional − argument is present, *calendar* reminds all users of their calendar engagements by *mail*(1). Normally this happens daily in the wee hours under control of *cron*(8). Calendars not in home directories, or recipients not registered as users, may be registered for reminder service by placing lines of the form *calendarfile mailname* in file `/usr/lib/calendar`.

EXAMPLES

```
#include /usr/pub/btlcalendar
#include /n/coma/usr/pub/btlcalendar
#include mh/astro/coma.calendar!/usr/pub/btlcalendar
```

Ways to subscribe to a public calendar by (1) users of `mh/astro/coma`, (2) users elsewhere who have *netfs*(8) access to `coma`, and (3) users elsewhere without *netfs* access.

FILES

```
calendar
/usr/lib/calendar?
/etc/passwd
/tmp/cal*
```

SEE ALSO

at(1)

BUGS

`#includes` do not nest.
The mail reminder service doesn't work when it finds fewer than two calendars.
Your calendar must be public information for you to get reminder service.
Holidays are what the program says they are.

NAME

cat − catenate and print

SYNOPSIS

`cat` [*file* ...]

DESCRIPTION

Cat reads each *file* in sequence and writes it on the standard output. Thus

```
cat file
```

prints a file and

```
cat file1 file2 >file3
```

concatenates the first two files and places the result on the third.

If no *file* is given, or if the argument − is encountered, *cat* reads from the standard input. Output is buffered in blocks matching the input.

SEE ALSO

p(1), *pr*(1), *cp*(1)

BUGS

Beware of `cat a b >a` and `cat a b >b`, which destroy input files before reading them.

NAME

cb − C program beautifier

SYNOPSIS

`cb` [*option* ...] [*file* ...]

DESCRIPTION

Cb reads C programs either from the named *files* or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code. The options are:

`-s` Place newlines as in Kernighan and Ritchie. (Original newlines are preserved by default.)

`-j` Join split lines.

`-l` *leng*
Split lines that are longer than *leng*, 120 by default.

SEE ALSO

pr(1), *troff*(1), *lp*(1), *font*(6)
B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1988.

BUGS

Punctuation hidden in preprocessor statements causes *cb* to make indentation errors.

NAME

cbt − btree utilities

SYNOPSIS

`cbt creat` *name ...*
`cbt report` *name ...*
`cbt cat` [`−R`] *name ...*
`cbt squash` [`−o`*dest*] *name*
`cbt build` [`−R`] *name*
`cbt add` [`−R`] *name*
`cbt delete` [`−R`] *name*
`cbt look` [`−R`] *name*

DESCRIPTION

A B-tree *name* corresponds to a pair of files named *name*`.T` and *name*`.F`. *Name*`.T` contains an access tree, and *name*`.F` contains the data.

The `creat` version of the command creates empty B-trees.

The `report` version of the command scans each B-tree and reports how many records it contains.

The `cat` version of the command scans the B-tree in key-sort order, writing on its standard output. With no option, `cat` writes each key followed by a tab, followed by the record, followed by a new-line. If option `−R` (raw) is present, each key-record pair has the format

```
struct {
	short keylen;
	char key[keylen];
	short reclen;
	char rec[reclen];
};
```

Keys and records are not null-terminated and consecutive key-record pairs are not separated by new-lines. Keys may be no longer than 255 bytes.

The `squash` version of the command compresses the access tree to minimal size. Option `−o` names the squashed database *dest*, leaving the original database unaltered.

The `build` version of the command reads a sorted list of keys and records from the standard input and fills the file with them. Input is in the form produced by the corresponding `cat` option.

The `add` (`delete, look`) version of the command inserts (removes, looks up) records. Input (and output of `look`) is in the form produced by the corresponding `cat` option. The records may be unsorted. In newline-separated input, only the keys need be present.

EXAMPLES

```
sort '-t<tab>' +0 -1 inputfile | cbt build btreefile
```

<tab> denotes a tab character

FILES

`/usr/lib/btree/*`
name`.[TF]`
`TS*[TF]` temporaries for `squash`

SEE ALSO

cbt(3)

NAME

cc, lcc − C compilers

SYNOPSIS

`cc` [*option ...*] *file ...*

`lcc` [*option ...*] *file ...*

DESCRIPTION

Cc compiles the classic C language; *lcc* compiles ANSI. They are otherwise similar. In the absence of options, any named source *files* are compiled into object files and then linked, together with any named object *files*, into a single executable file named `a.out`. Compilation normally has four phases: preprocessing of `#` directives, compilation to assembly language, assembly, and linking. Suffixes of *file* names determine which phases they participate in:

`.c` C source to be preprocessed and compiled. Object code for this file is finally placed in a correspondingly named `.o` file, except when exactly one file is being compiled and linked.

`.i` C source to be compiled without preprocessing; `#` directives are ignored by *cc*, forbidden by *lcc*.

`.s` Assembler source to be assembled, producing a `.o` file.

`.o` A preexisting object file to be linked.

Both compilers accept options of *ld*(1), the most common of which are `−o` (to substitute a name for `a.out`) and `−l` (to link from a library), and in addition

`−c` Suppress the linking phase, producing `.o` files but no `a.out`.

`−g` Produce additional symbol table information for debuggers such as *pi*(9.1).

`−O` Invoke an object-code improver; superfluous in *lcc*.

`−w` Suppress warning diagnostics. In *lcc*, `#pragma ref` *variable* supplies a dummy reference to suppress an unused-variable diagnostic.

`−p` Arrange for the compiler to produce code which counts the number of times each routine is called; also, if linking takes place, replace the standard startup routine by one which arranges to gather profiling data for later examination by *prof*(1).

`−pg` Like `−p` but for *gprof* instead of *prof*(1).

`−S` Compile the named C programs, and leave the assembler-language output in `.s` files.

`−E` Run the preprocessor on the named C programs, and send the result to the standard output.

`−C` Prevent the preprocessor from eliding comments.

`−D`*name=def*
`−D`*name*
Define the *name* to the preprocessor, as if by `#define`. If no definition is given, the name is defined as `1`. *Lcc* predefines a few symbols on most machines; option `−v` exposes them.

`−U`*name*
Remove any initial definition of *name*.

`−I`*dir* `#include` files whose names do not begin with `/` are always sought first in the directory of the *file* argument, then in directories named in `−I` options, then in directories on a standard list.

These options are peculiar to *cc*:

`−P` Run the preprocessor on each `.c` file. Produce no line numbers. Place results in `.i` files.

`−R` Cause *as*(1) to make initialized variables shared and read-only.

`−B`*string*
Find substitute compiler passes in the files named *string* with the suffixes cpp, ccom and c2. If

string is empty, use a standard backup version.

`-t[p012]`
Find only the designated compiler passes in the files whose names are constructed by a `-B` option. In the absence of a `-B` option, the *string* is taken to be `/usr/c/`.

These options are peculiar to *lcc*:

`-N` Do not search standard directories for include files. Omit non-ANSI language extensions.

`-A` Warn about calls to functions without prototypes.

`-b` produce code that writes an expression-level profile into prof.out. *bprint*(1) produces an annotated listing, and `-Wf-a` uses the profile to improve register assignments.

`-d`*n* Generate jump tables for switches with density at least *n*, a floating-point constant between zero and one, 0.5 by default.

`-P` Write declarations for all defined globals on standard error.

`-n` Produce code that reports and aborts upon dereferencing a zero pointer.

`-M` Run only the preprocessor to generate *make*(1) dependencies on the standard output.

`-t` Produce code to print trace messages at function entry and exit.

`-Wp`*opt* Pass preprocessor option *opt* to the (Gnu) preprocessor. For example, `-Wp-T` allows ANSI trigraph sequences.

`-Wa`*opt*, `-Wl`*opt*, `-Wf`*opt*
Pass option *opt* to the assembler (*as*(1)), loader (*ld*(1)), or compiler proper.

`-B`*str* Use the compiler *str*`rcc` instead of the default version. *Str* usually ends with a slash.

`-v` Report compiler steps (and some version numbers) as they are executed. A second `-v` causes steps to be reported but not executed.

Lcc supports `asm(`*string*`)`. The given string constant is copied to the generated assembly language output with occurrences of `%`*name* replaced by the address or register for identifier *name* if it is visible. Otherwise, `%`*name* is simply copied to the output. Wide-character literals are treated as plain char literals; ints and long ints are the same size, as are doubles and long doubles.

EXAMPLES

`lcc -N -I/usr/include/libc file.c`
Use local include files instead of ANSI standard ones, which lack most functions of Section 2 of this manual, and often disagree (especially about `const`) with those in Section 3. See *intro*(3).

FILES

Different machines use different file names, so this list is only representative. *Lcc* option `-v` exposes the correct names.

`a.out`	linked output
`/tmp/ctm*`	temporary
`/lib/cpp`	preprocessor, *cpp*(8)
`/usr/lib/gcc-cpp`	ANSI preprocessor
`/lib/ccom`	*cc* compiler proper
`/lib/c2`	optional optimizer for *cc*
`/bin/as`	assembler, *as*(1)
`/usr/lib/rcc`	*lcc* compiler proper
`/lib/crt0.o`	runtime startoff
`/lib/mcrt0.o`	startoff for profiling
`/lib/libc.a`	standard library, see *intro*(3)
`/usr/include`	directory for *cc* `#include` files

`/usr/include/lcc` directory for ANSI standard `#include` files
`/usr/include/libc` directory for local *lcc* include files

SEE ALSO

lint(1), *ld*(1), *strip*(1), *nm*(1), *prof*(1), *bprint*(1), *cin*(1), *adb*(1), *pi*(9.1), *c++*(1)
B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd Ed., Prentice-Hall, 1988

BUGS

Cc cannot handle the `-y` flag of *ld.*
Lcc currently uses the pre-ANSI library.

MACHINE DEPENDENCIES

VAX

`-pg` is unimplemented.
Cc and *lcc* use incompatible bit-field layouts and structure return conventions.

MIPS

Lcc does not implement `-p` or `-pg`, and its `-g` supports breakpoints but not the examination of variables.
Cc and *lcc* use incompatible bit-field layouts.

Sun

Lcc options `-Bdynamic` and `-Bstatic` give the binding strategy; see *ld*(1).
Cc and lcc use incompatible bit-field layouts and structure return conventions.

NAME

touch, chdate − set modification or access date of a file

SYNOPSIS

`touch` [`-c`] *file* ...

`chdate` [`-am`] *date file* ...

DESCRIPTION

Touch attempts to set the modification time of the *files* to the current time. If a *file* does not exist, it will be created unless option `-c` is present.

Chdate sets the access and modification times of *files*. The *date* comprises two or more arguments: a month (3 letters or more), a day number, an optional time in hour:min[:sec] form, and an optional year. A missing year means a time in the last 12 months. The options are

`-a` Change the access time only.

`-m` Change the modification time only.

Chdate knows how to carry between fields of a date. Only a file's owner or the super-user can change its date.

EXAMPLES

```
chdate jul 4 12:00 1976 independence
chdate jul -3 1976 independence      # backdate one week
```

SEE ALSO

ls(1), *utime* in *chmod*(2), *stat*(2), *timec*(3), *chmod*(1), *chown*(8)

DIAGNOSTICS

Chdate returns the number of files on which the date could not be changed.

BUGS

Touch will not touch directories.

The first *file* name for *chdate* cannot begin with a digit.

NAME

chmod − change mode

SYNOPSIS

`chmod` *mode file ...*

DESCRIPTION

The mode of each named file is changed according to *mode*, which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes. (Modes that contain a 1000 bit are incompatible with other modes that have any bits among 7000.)

4000 set user ID on execution
3000 set exclusive access mode (1 writer or *n* readers)
2000 set group ID on execution
1000 set synchronized access mode (1 writer and *n* readers)
0400 read by owner
0200 write by owner
0100 execute (search in directory) by owner
0070 read, write, execute (search) by group
0007 read, write, execute (search) by others

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission ...*]

The *who* part is a combination of the letters `u` (for user's permissions), `g` (group) and `o` (other). The letter `a` stands for `ugo`. If *who* is omitted, the default is `a`.

Op can be + to add *permission* to the file's mode, − to take away *permission* and = to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters `r` (read), `w` (write), `x` (execute), `s` (set owner or group id) `e` (set exclusive access mode) and `y` (set synchronized access mode). Letters `u`, `g` or `o` indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with = to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter `s` is only useful with `u` or `g`.

Only the owner of a file (or the super-user) may change its mode.

Synchronized access guards against inconsistent updates by preventing concurrent opens for writing. Exclusive access guards against inconsistent views by preventing concurrent opens if one is for writing.

EXAMPLES

`chmod o-w file`
Deny write permission to others.

`chmod +x file`
Make file executable.

SEE ALSO

ls(1), *chmod*(2), *stat*(2), *chdate*(1), *chown*(8)

NAME

cin − C interpreter

SYNOPSIS

`cin` [*option* ...] [*file* ...] [`--` *arg* ...]

DESCRIPTION

Cin interprets a C program made of the named *files.* A single minus (−) names the standard input. Other *files* are as in *cc*(1). Arguments *arg* are passed to the interpreted program as `cin_argv[1]` ...

Options `-D`*name*`=`*def*, `-D`*name*, `-U`*name*, `-I`*name*, `-l`*x*, and `-u`*name* are as in *cc*. Options `-O` and `-f` are ignored. Other options are:

`-0`*name* Pass *name* to the interpreted program as `cin_argv[0]`.

`-V`*name*`:`*n*
Function *name* has a variable number of arguments. Check types of the first *n* arguments.

`-c` Parse files but do not execute the program.

`-d` Enable internal debugging output.

`-i` Enter interactive mode; see below.

`-o`*name* Create object code in file *name* before starting execution.

`-r` Prohibit multiply-declared global variables.

`-s` Do not catch signals.

`-t` Print file name and line number of each C statement executed.

`-v` Print the user and system times associated with loading files and executing the program, as in *time*(1).

Under option `-i`, *cin* evaluates C statements immediately using local and global variables. Non-void values are printed. Function, variable, and C pre-processor declarations are stored in the current 'view'. A function definition must include its return type. Declarations and statements may appear in any order and variables may be redeclared. The functions and variables listed below are predefined; the types of their arguments follow these conventions:

```
char *name, *string, *file;
int line;
long (*func)(), (*func_mod)(), (*func_ref)();
```

`void cin_bind(`*name*`,` *func*`);` Set an alias of *name* for the function *func*.

`int cin_break(`*name*`,` *line*`);` Set a breakpoint in function *name* at line number *line* in the current view. Return 0 if breakpoint cannot be set.

`int cin_dump(`*name*`);` Create an `a.out` and place it in the file *name*. Return 0 if *name* cannot be created.

`int cin_eval(`*string*`);` Execute the C statement *string* as if it were present in the program where the `cin_eval` is executed. Return 0 if *string* could not be parsed.

`int cin_load(`*file*`);` Load *file* into a new view, using the standard file-naming conventions for *cc*(1). Return 0 if *file* can not be loaded.

`int cin_log(`*file*`);` Record the interactive mode session in *file*. Return 0 if *file* cannot be created.

`void cin_mstats(`*string*`);` Print *cin*'s memory usage with a banner of *string*.

`void cin_return();` Return from a breakpoint.

`int cin_spy(`*name*`,` *func_mod*`,` *func_ref*`);` Call the function *func_mod* whenever the variable *name* is modified. Call the function *func_ref* whenever the variable *name* is referenced. Either *func_mod* or *func_ref* can be `(long (*)())0`. Return 0 if there are no spies active.

`void cin_system();` Enter interactive mode.

`int cin_unbreak(`*name*`, `*line*`);` Remove a breakpoint in function *name* at line number *line* in the current view. Return 0 if the breakpoint was not set.

`int cin_view(`*name*`);` Change the current view to *name*. Return 0 if the name is not found.

`void cin_views();` List the available views. The current view is starred (`*`).

`void cin_whatis(`*name*`);` Print the type of the named variable.

`void cin_where();` Print a trace of subroutine calls.

`void cin_quit();` Exit *cin*.

`extern int cin_argc;` The number of elements passed to the interpreted program.

`extern char **cin_argv;` An array of arguments passed to the interpreted program.

`extern char *cin_libpath;` A colon-separated list of libraries to search for undefined routines (by default the libraries specified on the command line plus `-lc`).

`extern char *cin_prompt;` The interactive mode prompt (by default the string `cin>`).

EXAMPLES

‘Hello world’ in interactive mode.

```
cin> printf("Hello world\n");
Hello world
(int)12
```

Setting breakpoints and tracing subroutine calls.

```
cin> int f(x) { return x <= 1 ? 1 : x * f(x-1); }
cin> cin_break("f", 1);
breakpoint in function f at line 1
(int)1
cin> f(2);
breakpoint in function 'f' at line 1 of file 'interactive mode'
cin> cin_return();
breakpoint in function 'f' at line 1 of file 'interactive mode'
cin> (void)cin_where();
cin_system()   [interactive mode:4]
f(x = 1)   [interactive mode:1]
f(x = 2)   [interactive mode:1]
cin_system()   [interactive mode:3]
cin> x;
(int)1
cin> cin_return();
(int)2
```

FILES

`/tmp/int*`	temporary
`$HOME/.cinit`	interpreter startup file

other files as in; *cc*(1)

SEE ALSO

cc(1), *lint*(1)

T. J. Kowalski, H. H. Goguen, and J. J. Puttress, ‘The C Interpreter: A Tutorial for Cin Version 0.18’, this manual, Volume 2

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978

NAME

cmp − compare two files

SYNOPSIS

`cmp` [`-lsL`] *file1 file2* [*offset1* [*offset2*]]

DESCRIPTION

The two files are compared. If the contents differ a diagnostic results, otherwise there is no output.

The options are:

`l` Print the byte number (decimal) and the differing bytes (octal) for each difference.

`s` Print nothing for differing files, but set the return code.

`L` Print the line number of the first differing byte.

If offsets are given, comparison starts at the designated byte position of the corresponding file. Offsets that begin with `0x` are hexadecimal; with `0`, octal; with anything else, decimal.

SEE ALSO

diff(1), *comm*(1)

DIAGNOSTICS

Cmp reports 'EOF' and identifies the file if one file is short. It reports the number of the first disagreeing byte if contents differ. The return code is 0 for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

NAME

col, 2, 3, 4, 5, 6, mc, fold, expand − column alignment

SYNOPSIS

`col` [`-bfx`]

`2` [*file*]

`mc` [`-`] [`-`*N*] [`-t`] [*file ...*]

`fold` [`-`*N*] [*file ...*]

`expand` [`-`*stops*] [*file ...*]

DESCRIPTION

These programs rearrange files for appearance's sake. All read the standard input and write the standard output. Some optionally read from files instead.

Col overlays lines to expunge reverse line feeds (ESC-7) and half line feeds (ESC-9 and ESC-8) as produced by *nroff* for .2C in *ms*(6) and for *tbl*(1). It normally emits only full line feeds; option `-f` (fine) allows half line feeds too. Option `-b` removes backspaces, printing just one of each pile of overstruck characters. *Col* normally converts white space to tabs; option `-x` overrides this feature. Other escaped characters and non-printing characters, except for SO and SI, are ignored.

Col should not be used for printing on an HP ThinkJet printer with *think* (*thinkblt*(9.1)), which performs the *col* function itself.

Commands *2, 3, 4, 5, 6* convert their input to 2-, 3-, 4-, 5-, or 6-column form, with consecutive input lines arranged across each row.

Fold inserts newlines after each *N* characters (default *n*=80, or *mux*(9.1) window size) of long lines.

Mc splits the input into as many columns as will fit in *N* print positions (default *N*=80). Under option `-` each input line ending in a colon `:` is printed separately (see example). On output, multiple spaces are converted to tabs; this is suppressed by option `-t`.

Expand replaces tabs by spaces. The optional *stops* argument is a comma-separated list of tab stops, counted from 0; default is every 8 columns.

EXAMPLES

`tbl file | nroff -ms | col | hp`
: Format some tables for printing on typewriters; use *col* to remove reverse line feeds and *hp* (see *ul*(1)) to do underlining, etc., on an HP terminal.

`ls directory1 directory2 | mc -`
: List files in multiple columns, separated by directory.

SEE ALSO

pr(1)

BUGS

Col can't back up more than 128 lines or handle more than 800 characters per line, and understands `VT` (013) as reverse line feed.

NAME

comm − select or reject lines common to two sorted files

SYNOPSIS

`comm` [`−123`] *file1 file2*

DESCRIPTION

Comm reads *file1* and *file2*, which are ordered in ASCII collating sequence, and produces a three column output: lines only in *file1*; lines only in *file2*; and lines in both files. The filename − means the standard input.

Flag `1`, `2`, or `3` suppresses printing of the corresponding column.

EXAMPLES

```
comm −12 file1 file2
```

Print lines common to two sorted files.

SEE ALSO

sort(1), *cmp*(1), *diff*(1), *uniq*(1), *join*(1)

NAME

con, rx − remote login and execution

SYNOPSIS

`con` [`-l`] *machine*

`rx` [`-n`] *machine* [*command-word ...*]

`/usr/bin/m/`*machine* [*command-word ...*]

DESCRIPTION

Con connects to the computer whose network address is *machine* and logs in if possible. Standard input and output go to the local machine. Option `-l` prevents automatic login; a normal login dialog ensues.

The quit signal (control-\) is a local escape. It prompts with the local machine name and >>. Legitimate responses to the prompt are

`i`	Send a quit [sic] signal to the remote machine.
`q`, `x`, or `.`	Exit.
`b`	Send a break.
`!`*command*	Execute *command* locally.

Rx executes one shell command on the remote machine as if logged in there, but with local standard input and output. Unquoted shell metacharacters in the command are interpreted locally, quoted ones remotely. The assignment `REXEC=1` appears in the remote environment. With no arguments, *rx* just diagnoses availability. Option `-n` ignores sporadic end-of-file indications on a sick network.

Network addresses for both *con* and *rx* have the form *network*`!`*host* or simply *host.* Supported networks are `dk` (Datakit) and `tcp` (TCP/IP, usually Ethernet).

Directory `/usr/bin/m` contains machine names as commands: `/usr/bin/m/`*machine* with no argument runs an appropriate flavor of *con* for the named *machine.* If given arguments, `/usr/bin/m/`*machine* runs *rx* with those arguments. If `/usr/bin/m` is in the *sh*(1) search path, the names become commands for navigating the local cluster.

EXAMPLES

```
rx overthere cat file1 >file2
```
Copy remote *file1* to local *file2*.

```
rx overthere cat file1 ">file2"
```
Copy remote *file1* to remote *file2.*

```
eqn paper | rx pipe troff -ms | rx arend lp
```
Parallel processing: do each stage of a pipeline on a different machine.

FILES

`/usr/ipc/lib/auth*`	authentication
`/usr/ipc/lib/serv*`	servers

SEE ALSO

push(1), *dcon*(1), *cu*(1), *dkmgr*(8), *svcmgr*(8), *tcpmgr*(8), *ipc*(3)
D. L. Presotto, 'Interprocess Communication in the Eighth Edition UNIX System', this manual, Volume 2

BUGS

The remote standard error and standard output are combined and go inseparably to the local standard output.
Under *rx,* a program that should behave specially towards terminals may not: *sh*(1) will not prompt, *vi*(1) will not manage the screen, etc. *Nrx* (see *dcon*(1)) avoids this trouble, but has others of its own.
Con and *rx* may not guess the right kind of connection. In case of trouble, try the programs in *dcon*(1).
The names in `/usr/bin/m` are conventions, not actual network addresses.

NAME

cp, mv, ln, reccp − copy, move, or link files

SYNOPSIS

`cp` [`-z`] *file1 file2*
`cp` [`-z`] *file ... directory*

`mv` [`-f`] *file1 file2*
`mv` [`-f`] *file ... directory*

`ln` [`-s`] *file1 file2*
`ln` [`-s`] *file ... directory*

`reccp` [`-z`] *file1 file2*
`reccp` [`-z`] *file ... directory*

DESCRIPTION

In the first form of each command, *file2* is any name except an existing directory. In the second form the command copies, moves, or links one or more *files* into a *directory* under their original filenames, as if by a sequence of commands in the first form. Thus `mv f1 f2 dir` is equivalent to `mv f1 dir/f1; mv f2 dir/f2`.

Cp copies the contents of plain *file1* to *file2*. The mode and owner of *file2* are preserved if it already exists; the mode of *file1* is used otherwise.

Mv moves *file1* to *file2*. If the two files are in the same file system, the name *file1* is simply changed to *file2*; if they are in different file systems, *file1* is copied and then removed. If *file2* already exists, it is removed before *file1* is moved. In this case the mode of *file2* is reported if it is not writable and the standard input is a terminal. Respond `y` (and newline) to permit removal.

Ln links plain *file1* and *file2*. *File2* becomes an alternate name for, and is otherwise identical to, *file1*. *File2* must be in the same file system as *file1* and must not already exist.

Reccp copies plain files like *cp*, but copies directories and their contents recursively. It attempts to duplicate linkage and dates. When run by the super-user, it preserves ownership and copies device files as device files.

The options are:

`-z` Preserve ‘holes’; see *lseek*(2).

`-f` Forcibly remove *file2* without asking.

`-s` Make symbolic links: record the (arbitrary) name *file1* in *file2*. Except in special cases, such as *rm*(1) and *lstat* (see *stat*(2)), subsequent references to *file2* are treated as references to *file1*. See *link*(2) for details.

EXAMPLES

`mkdir /usr1/ken; cp /usr/ken/* /usr1/ken`
Place in `/usr1/ken` copies of all files from `/usr/ken`.

`reccp /usr/ken /usr1`
`mkdir /usr1/ken; reccp /usr/ken/* /usr1/ken`
Two ways to duplicate in `/usr1/ken` the whole file hierarchy from `/usr/ken`.

SEE ALSO

cat(1), *link*(2), *stat*(2), *push*(1), *uucp*(1), *rcp*(1), *cpio*(1)

DIAGNOSTICS

Cp, *mv*, and *reccp* refuse to copy or move files onto themselves or directories into themselves.

BUGS

Mv to a different file system is imperfect: if *file1* is a plain file links to it are broken; if it is a directory, nothing happens.

NAME

cpio − copy file archives in and out

SYNOPSIS

`cpio −o` [*acBv*]

`cpio −i` [*Bcdmrtuvf*s*Sb6*] [*pattern* ...]

`cpio −p` [*adlmruv*] *directory*

DESCRIPTION

`Cpio −o` (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information.

`Cpio −i` (copy in) extracts files from the standard input which is assumed to be the product of a previous `cpio −o`. Only files with names that match the *patterns* are selected. *Patterns* are given in the name-generating notation of *sh*(1); the default is `*` (all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below. File ownership is preserved if possible.

`Cpio −p` (pass) reads from the standard input a list of path names of files to copy into the destination *directory*.

The options are:

`a` Reset access times of input files after they have been copied.

`B` Input/output is to be blocked 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from `/dev/rmt?`).

`d` *Directories* are to be created as needed.

`c` Write header information in ASCII character form for portability.

`r` Interactively rename files. An empty name (newline only) causes a file to be skipped.

`t` Print a table of contents of the input. No files are created.

`u` Copy unconditionally (normally an older file will not overwrite a newer file).

`v` (Verbose) List file names; `−vt` looks like `ls −l`.

`l` Whenever possible, link files rather than copying them. Usable only with the `−p` option.

`m` Retain previous file modification time. This option is ineffective on directories that are being copied.

`f` Copy in all files except those in *patterns.*

`s` Swap bytes. Use only with the `−i` option.

`S` Swap halfwords. Use only with the `−i` option.

`b` Swap both bytes and halfwords. Use only with the `−i` option.

`6` Process an old (i.e., UNIX Sixth Edition format) file. Only useful with `−i` (copy in).

EXAMPLES

```
ls | cpio −oc >/dev/mt1
```

Copy the contents of the current directory to a tape.

```
mkdir newdir
cd olddir
find . −print | cpio −pd ../newdir
```

Reproduce a directory hierarchy; `newdir` must exist.

SEE ALSO

ar(1), *bundle*(1), *tar*(1), *find*(1), *cpio*(5), *cp*(1)

BUGS

Path names are restricted to 128 characters.
Does not know about symbolic links.
If there are too many unique linked files, linking information is lost.
Only the super-user can copy special files.
The archive size is reported in archaic ‘blocks’ of 512 bytes.

NAME

crypt, encrypt, decrypt − encode/decode

SYNOPSIS

`/usr/games/crypt` [*password*]

`/usr/games/encrypt` [`-p`] [*password*]

`/usr/games/decrypt` [`-p`] [*password*]

DESCRIPTION

These commands read from the standard input and write on the standard output. The *password* is an enciphering key. If no password is given, one is demanded from the terminal; echoing is turned off while it is being typed in. *Crypt* uses a relatively simple, fast method (rotor machine) for both enciphering and deciphering. *Encrypt* and *decrypt* use a more robust, slower method (DES). Files enciphered by *crypt* are not intelligible to *encrypt/decrypt*, and vice versa.

It is prudent to supply the key from the terminal, not from the command line, and to pick a reasonably obscure and long key (6 letters for *crypt* and much longer for *encrypt*).

Under option `-p` *encrypt* enciphers into printing characters, which can be sent by *mail*(1). *Decrypt* can distinguish ciphertext from clear: it will work on a full mail message, headers and all.

FILES

`/dev/tty` for typed key

SEE ALSO

ed(1), *makekey*(8)

J. A. Reeds and P. J. Weinberger, 'File Security and the Unix Crypt Command,' *AT&T Bell Laboratories Technical Journal*, `63` (1984) 1673-1684

BUGS

Crypt is breakable by knowledgeable cryptanalysts. Its only practical use is for mildly private data transmission. *Encrypt/decrypt* gives strong protection for transmission over untrusted channels between trusted machines.

It is unwise to count on encryption of any sort for safe storage of documents.

NAME

cscan − scan documents on canon scanner

SYNOPSIS

`cscan` [*option* ...] [*file* ...]

DESCRIPTION

Cscan acquires black-and-white images using the Canon 9030 Laser Copier flatbed document scanner located in the Graphics Lab. Each *file* receives one image; multiple *file*s are scanned as fast as possible (every 12s, after a startup delay). If no *file* is specified, one image is written to the standard output.

Each image is compressed using the CCITT FAX Group 4 standard and supplied with a header as described in *picfile*(5). The default page format is 3456 by 4400 pixels (*i.e.* letter-size 8.64"×11.0" at 400 pixels/inch digitizing resolution).

The options are:

`−f`*x*,*y* Format is *x* by *y* pixels. *X* is truncated to a multiple of 32.

`−fL` Double-letter format `−f`4416,6800 or 11"×17" at 400 pixels/inch (the largest possible).

`−s`*s* Sleep between scans for an extra *s* seconds.

`−v` Verbose: announce each *file* as it arrives.

SEE ALSO

bcp(1), *ocr*(1), *picfile*(5)

BUGS

Although only 10s are required to acquire and compress each image, it can take 40s for the first file to appear, due largely to networking latencies involving *rcp*(1).
Cscan is unkillable: scanning continues and the files keep coming.
The automatic document feeder is not supported.
The resolution recorded in the header is always 400 pixels/inch, even though it is possible manually to zoom to other resolutions.

NAME

cu, ct − call out to a terminal or another system

SYNOPSIS

`cu` [`-htn`] [`-p` *parity*] [`-s` *speed*] *telno* [*service-class*]

`ct` [option ...] *phone-number* [*service-class*]

DESCRIPTION

Cu places a data call to a given telephone number and expects a computer to answer. It manages an interactive conversation with possible transfers of text files. *Telno* is the telephone number, consisting of digits with minus signs at appropriate places to indicate delay for second or subsequent dial tones. A telephone number may also be expressed symbolically. A symbolic number is looked up in the files `$HOME/lib/cunumber` and `/usr/lib/cunumber` whose lines look like this:

symbolic-number actual-number service-class comment

The actual number may be preceded by options such as `-t`. The *comment*, if present, is printed out when the connection is made.

The options are

`-n` Print the the called number but do not call it.

`-t` Tandem: use DC1/DC3 (control-`S`/control-`Q`) protocol to stop transmission from the remote system when the local terminal buffers are almost full. This argument should only be used if the remote system understands that protocol.

`-h` Half-duplex: echo locally the characters that are sent to the remote system.

`-s` *speed*
Set the line speed; `1200` means 1200 baud, etc. The default depends on service class.

`-p` *parity*
Set the parity of transmitted characters: `0`, `1`, `e`, `o` mean zero, one, even, odd parity. `0` is the default.

The service class is expressed as in *dialout*(3). A special class `direct` causes the *telno* argument to be taken as the pathname of a terminal line. *Cu* opens the file, sets line speed and other modes, and proceeds as if connected. The default line speed is 9600 baud.

An explicit service class on the command line overrides any specified in a `cunumber` file.

After making the connection, *cu* runs as two processes: the sending process reads the standard input and passes most of it to the remote system; the receiving process reads from the remote system and passes most data to the standard output. Lines beginning with ~ have special meanings.

The sending process interprets:

`~.`
~EOT Terminate the conversation.

~<*file* Send the contents of *file* to the remote system, as though typed at the terminal.

`~!` Invoke an interactive shell on the local system.

`~!`*cmd* Run the command on the local system (via `sh -c`).

`~$`*cmd* Run the command locally and send its output to the remote system.

`~b`
`~%break`
Send a break (300 ms space).

`~%take` *from* [*to*]
Copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* name is used both places.

`~%put` *from* [*to*]
Copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* name is used both places.

~~text send the line *~text* .

WARNING: Using *cu* to reach your home machine from a machine you don't trust can be hazardous to your password.

Ct places a telephone call to a remote terminal and allows a user to log in on that terminal in the normal fashion. The terminal must be equipped with an auto-answer modem.

The phone number and service class are as in *cu.* The options are

`-c` *count*
If the number doesn't answer, try *count* times before giving up (default 5).

`-w` *interval*
Space retries *interval* seconds apart (default 60).

`-h` Try to hang up the phone before placing the call. This is useful for a 'call me right back' arrangement.

FILES

`/usr/lib/cunumber`
`$HOME/lib/cunumber`

SEE ALSO

con(1), *ttyld*(4), *dialout*(3)

BUGS

Unless erase and kill characters are the same on the two machines, they will be damaged by `~%put`.
`~%take` uses `~>` at the beginning of line to synchronize transmission. This sequence can cause misfunction if it is received for any other purpose.

NAME

cut, paste − rearrange columns of data

SYNOPSIS

`cut −c`*list* [*file ...*]

`cut −f`*list* [`−d`*char*] [*file ...*]

`paste` [`−s`] [`−d`*chars*] *file ...*

DESCRIPTION

Cut selects fields from each line of the *files* (standard input default). In data base parlance, it projects a relation. The fields can be fixed length, as on a punched card (option `−c`), or be marked with a delimiter character (option `−f`).

The meanings of the options follow. A *list* is an increasing sequence of integers separated by commas, or by − to indicate a range, for example `1,3−5,7.`

`−c`*list* The *list* specifies character positions.

`−f`*list* The *list* specifies field numbers.

`−d`*char* The character is the delimiter for option `−f`. Default is tab.

`−s` Suppress lines with no delimiter characters. Normally such lines pass through untouched under option `−f`.

Paste concatenates corresponding lines of the input *files* and places the result on the standard output. The file name − refers to the standard input. Lines are glued together with characters taken circularly from the set *chars*. The set may contain the special escape sequences `\n` (newline), `\t` (tab), `\\` (backslash), and `\0` (empty string, not a null character). The options are

`−d`*chars*
The output separator characters. Default is a tab.

`−s` Paste together lines of one file, treating every line as if it came from a distinct input.

EXAMPLES

```
cut -d: -f1,3 /etc/passwd
```
Print map from login names to userids, see *passwd*(5).

```
NAME=`who am i | cut -f1 -d" "`
```
Set `NAME` to current login name (subtly different from *getuid*(1)).

```
ls | paste - - - -
ls | paste -s '-d\t\n' -
```
4-column and 2-column file listing

SEE ALSO

gre(1), *awk*(1), *sed*(1), *pr*(1), *column*(1)

BUGS

Cut should handle disordered lists under option `−f`.
In default of file names, *paste* should read the the standard input.

NAME

dag − preprocessor for drawing directed graphs

SYNOPSIS

`dag` [*option* ...] [file ...]

DESCRIPTION

Dag is a *pic*(1) or PostScript preprocessor for laying out directed graphs. It does well on acyclic graphs (dags) and other graphs that can be drawn hierarchically. Graph statements are contained between `.GD` (node ranks increase downward) or `.GR` (rightward) and `.GE`. Edges point in the direction of increasing rank if possible; the other direction is favored for edges within rank. A summary of statements follows.

`edge from` *tail* `to` *head0 edge-items*`,` `to` *head1 edge-items*`,` `to` *head2 edge-items...*`;` Create edges from the tail node to the head node(s). Nodes are created if they do not already exist. Edge-items (described below) and the noise words `edge`, `from`, `to`, and comma are optional. Node names may be quoted to protect blanks or keywords.

`ordered` *tail head0 head1 head2...*`;` Make edges with heads in given left-to-right order on the same rank. May contain noise words and edge-items.

`path` *node0 node1 node2...*`;` Make a path of edges. May contain noise words and edge-items.

`backedge` *tail head0 head1 head2...*`;` Same as edges with opposite node ranking preferred.

`backpath` *node0 node1 node2...*`;` Make a path of backedges.

`draw nodes` *node-items*`;` Set properties of subsequently created nodes. Legal *node-items*:

`as` *shape*	Known shapes are `Box`, `Circle`, `Doublecircle`, `Ellipse` (default), `Diamond`, `Square`, `Plaintext`. Other shapes may be specified within braces `{}` in the output language (e.g. *pic*) or defined; see below.
`label` "*string*"	Label with *string* instead of node name.
`pointsize` *points*	
`width` *inches*	
`height` *inches*	
`color` "*string*"	Hue-saturation-brightness triple; works only with PostScript.

`draw` *nodelist node-items*`;` Set properties of listed nodes.

`draw edges` *edge-items*`;` Set properties of subsequently created edges. Legal *edge-items*:

`dotted`, `dashed`, `solid`, `invis`	
`label` "*string*"	
`weight` *n*	High-weight edges are kept short. Default weight 1.
`color` "*string*"	
`pointsize` *points*	

`minimum rank` *nodelist*`;` Listed nodes must be on the topmost rank (leftmost with `.GR`).
`maximum rank` *nodelist*`;` Bottommost or rightmost rank.
`same rank` *nodelist*`;`

`separate ranks` *inches how* `;` Set minimum separation between ranks. The optional *how* is `exactly` or `equally`.
`separate nodes` *inches* `;`

The options are

`-O`	Place nodes ‘optimally’; practical for graphs of a few dozen nodes.
`-Tps`	Prepare output for PostScript rather than *pic*.
`-Tsimple`	Output similar to that of *graphdraw*(9.1).
`-Tcip`	Output readable by *cip*(9.1).

`-p`*width*`x`*height*`,`*marginwidth*`x`*marginheight*
Set PostScript page dimensions; *marginwidth* and/or *marginheight* may be omitted.

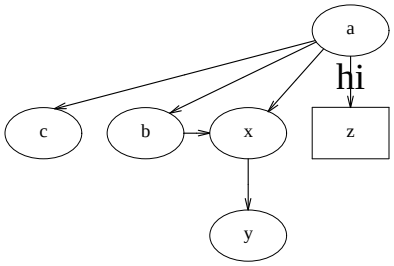
`-l` Disable automatic loading of the *dag* graphics library.

The introductory `.GD` or `.GR` line may carry optional parameters in the form `.GD` *width height* `fill`. *Width* and *height* are maximum values in inches; `fill` causes the graph to be stretched to the full dimensions.

Graphics code written in the output language (*pic* or PostScript) may be embedded between `.PS` and `.PE`. Macros defined with three arguments—label, width, and height—may be used as shape names in node-items.

EXAMPLES

```
.GD 2 2
a b c;
path a x y;
draw nodes as Box;
a z label "hi" weight 1000;
draw edges dashed;
b x;
same rank b x;
.GE
```



FILES

`$TOOLS/daglib.pic` default `TOOLS=/usr/lib`
`$TOOLS/daglib.ps`

SEE ALSO

pic(1), *lp*(1), *graphdraw*(9.1), *psi*(9.1), *troff*(1)
E. R. Gansner, S. C. North, K. P. Vo, 'DAG—A Program that Draws Directed Graphs', this manual, Volume 2

BUGS

The delimiter `.GD` is nonstandard; it may be called `.GS` in installations where *ped*(9.1) is not used.
Troff lacks dotted or dashed splines; use PostScript.
Edge labels may overlap.

NAME

date − print or set the date

SYNOPSIS

`date` [*yymmddhhmm* [`.` *ss*]]

DESCRIPTION

If no argument is given, the current date and time are printed. If an argument is given and the user is the super-user, the current date is set. *yy* is the last two digits of the year; the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; **.***ss* is optional and is the seconds. The year, month and day may be omitted, the current values being the defaults. The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight time. The options are

`-u` Set or report GMT rather than local time.

`-n` Set or report the date as the number of seconds since the epoch.

EXAMPLES

`date 10080045`

Set the date to Oct 8, 12:45 AM, local time.

FILES

`/usr/adm/wtmp` to record time-setting

SEE ALSO

utmp(5)

DIAGNOSTICS

'No permission' if a non-super user tries to change the date; 'bad conversion' if the date is invalid.

NAME

dc − desk calculator

SYNOPSIS

`dc` [*file*]

DESCRIPTION

Dc is an arbitrary precision desk calculator. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number The value of the number is pushed on the stack. A number is an unbroken string of the digits `0-9A-F`. It may be preceded by an underscore _ to input a negative number. Numbers may contain decimal points.

`+ − / * % ^`
Add `+`, subtract −, multiply `*`, divide `/`, remainder `%`, or exponentiate `^` the top two values on the stack. The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

`s`*x*
`S`*x* Pop the top of the stack and store into a register named *x*, where *x* may be any character. Under operation `S` register *x* is treated as a stack and the value is pushed on it.

`l`*x*
`L`*x* Push the value in register *x* onto the stack. The register *x* is not altered. All registers start with zero value. Under operation `L` register *x* is treated as a stack and its top value is popped onto the main stack.

`d` Duplicate the top value on the stack.

`p` Print the top value on the stack. The top value remains unchanged. `P` interprets the top of the stack as an ASCII string, removes it, and prints it.

`f` Print the values on the stack.

`q`
`Q` Exit the program. If executing a string, the recursion level is popped by two. Under operation `Q` the top value on the stack is popped and the string execution level is popped by that value.

`x` Treat the top element of the stack as a character string and execute it as a string of *dc* commands.

`X` Replace the number on the top of the stack with its scale factor.

`[ ... ]`
Put the bracketed ASCII string on the top of the stack.

<*x* >*x* =*x*
Pop and compare the top two elements of the stack. Register *x* is executed if they obey the stated relation.

`v` Replace the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

`!` Interpret the rest of the line as a UNIX command.

`c` Clear the stack.

`i` The top value on the stack is popped and used as the number base for further input.

`I` Push the input base on the top of the stack.

`o` The top value on the stack is popped and used as the number base for further output. In bases larger than 10, each 'digit' prints as a group of decimal digits.

`O` Push the output base on the top of the stack.

`k` Pop the top of the stack, and use that value as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

`z` Push the stack level is pushed onto the stack.

`Z` Replace the number on the top of the stack with its length.

`?` A line of input is taken from the input source (usually the terminal) and executed.

`; :` Used by *bc* for array operations.

The scale factor set by `k` determines how many digits are kept to the right of the decimal point. If *s* is the current scale factor, *sa* is the scale of the first operand, *sb* is the scale of the second, and *b* is the (integer) second operand, results are truncated to the following scales.

+,−	max(*sa*,*sb*)
*	min(*sa*+*sb* , max(*s*,*sa*,*sb*))
/	*s*
%	so that dividend = divisor*quotient + remainder; remainder has sign of dividend
^	min(*sa*×|*b*|, max(*s*,*sa*))
v	max(*s*,*sa*)

EXAMPLES

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

Print the first ten values of *n*!

SEE ALSO

bc(1), *hoc*(1)

DIAGNOSTICS

x `is unimplemented`, where *x* is an octal number: an internal error.
'Out of headers' for too many numbers being kept around.
'Nesting depth' for too many levels of nested execution.

BUGS

When the input base exceeds 16, there is no notation for digits greater than `F`.

NAME

dcon, ndcon, rlogin, nrx, rsh, scriptcon − remote login and execution

SYNOPSIS

`dcon` [*option* ...] *machine*

`ndcon` *machine*

`rlogin` *machine*

`nrx` *machine* [*command-word* ...]

`rsh` [*option* ...] *machine* [*command-word* ...]

`scriptcon` *machine script*

DESCRIPTION

Do not read this page unless you are familiar with *con*(1).

Dcon, ndcon, and *rlogin* are analogs (or special cases) of *con*(1) for specific kinds of network connection. They support the same local escape convention with the quit signal.

Similarly, *nrx* and *rsh* are analogs of *rx.*

Network addresses are as in *con*(1). The default networks for the various commands are

dcon, ndcon, nrx `dk`

rlogin, rsh `tcp`

Dcon connects to the remote machine, and attempts automatically to log in under the login id of the invoking user. Option `-l` turns off automatic login; the remote machine will ask for a login id and password.

Ndcon behaves like *dcon* but provides a more transparent transport protocol. In particular terminal line disciplines are preserved and it is possible to download into a *mux*(9.1) window across an *ndcon* connection.

Rlogin is like *dcon*, but uses the connection protocol found on Berkeley systems.

Rx (see *con*(1)) executes one shell command on the remote machine as if logged in there, with local standard input and output. It uses a connection protocol specific to Research machines.

Nrx is to *rx* as *ndcon* is to *dcon*: it runs a command remotely with line discipline preserved.

Rsh is to *rx* as *rlogin* is to *dcon*: it runs a command remotely using the Berkeley execution protocol.

Scriptcon provides a connection like `dcon -l`, except that the login and other initial protocol are controlled by a *script* file. The first line of the file gives a string (e.g. `login:`) expected from the remote machine; the second gives the local response, and so on in alternation. Unrecognized data from the remote machine are ignored. Warning: a script that contains a password may compromise the security of the remote system, hence *scriptcon* should be used only for restricted logins.

Con tries to connect using the protocol of *ndcon*; if that fails, it tries that of *dcon*, then that of *rlogin*. *Rx* attempts its own style of connection; if that fails, it tries that of *rsh*.

SEE ALSO

con(1), *dkmgr*(8), *svcmgr*(8), *tcpmgr*(8), *ipc*(3)
D. L. Presotto, ‘Interprocess Communication in the Eighth Edition UNIX System’, this manual, Volume 2

BUGS

If a program run by *nrx* won’t let go, for example by ignoring signals, there is no way of getting out short of hanging up.
There is no error correction or retry in a *scriptcon* script.

NAME

dd, dblbuf − convert and copy a file

SYNOPSIS

`dd` [*option=value*]

`dblbuf` [`-b` *blocksize*] [*file*]

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O. The options are

`if=`*file* Set the input file (standard input by default).

`of=`*file* Set the output file (standard output by default).

`ibs=`*n* Set input block size to *n* bytes (default 512).

`obs=`*n* Set output block size (default 512).

`bs=`*n* Set both input and output block size, superseding *ibs* and *obs*. If no conversion is specified, preserve the input block size instead of packing short blocks into the output buffer. This is particularly efficient since no in-core copy need be done.

`cbs=`*n* Set conversion buffer size.

`skip=`*n* Skip *n* input records before copying.

`iseek=`*n* Seek *n* records forward on input file before copying.

`files=`*n* Copy and concatenate *n* input files (makes sense only where input is a magnetic tape or similar device).

`oseek=`*n* Aeek *n* records from beginning of output file before copying.

`count=`*n* Copy only *n* input records.

`conv=ascii`	Convert EBCDIC to ASCII.
`ebcdic`	Convert ASCII to EBCDIC.
`ibm`	Like `ebcdic` but with a slightly different character map.
`block`	Convert variable length ASCII records to fixed length.
`unblock`	Convert fixed length ASCII records to variable length.
`lcase`	Map alphabetics to lower case.
`ucase`	Map alphabetics to upper case.
`swab`	Swap every pair of bytes.
`noerror`	Do not stop processing on an error.
`sync`	Pad every input record to *ibs* bytes.

Where sizes are specified, a number of bytes is expected. A number may end with `k`, `b`, or `w` to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by `x` to indicate a product. Multiple conversions may be specified in the style: `conv=ebcdic,ucase`.

`Cbs` is used only if `ascii`, `unblock`, `ebcdic`, `ibm`, or `block` conversion is specified. In the first two cases, *n* characters are copied into the conversion buffer, any specified character mapping is done, trailing blanks are trimmed and new-line is added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer and blanks are added to make up an output record of size *n.* If `cbs` is unspecified or zero, the `ascii`, `ebcdic`, and `ibm` options convert the character set without changing the block structure of the input file; the `unblock` and `block` options become a simple file copy.

Dblbuf copies the named *file*, or the standard input if no file is specified, to the standard output. Output is written in blocks matching the input up to the given blocksize, or 32768 bytes if not specified.

Dblbuf uses multiple processes to run faster, which is particularly useful in dealing with a device such as a streaming tape drive.

EXAMPLES

`dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase`

Read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into an ASCII file. Note the use of raw magtape to handle arbitrary record sizes.

`tar cf /dev/stdout /usr | dblbuf >/dev/rmt1`

Copy the `/usr` directory to tape on `/dev/rmt1`.

SEE ALSO

cp(1), *tar*(1), *cpio*(1)

DIAGNOSTICS

Dd reports the number of full + partial input and output blocks handled.

BUGS

The ASCII/EBCDIC conversion tables for *dd* were taken from the 256-character standard in CACM Nov, 1968. The `ibm` conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

Options `if` and `of` are verbose equivalents of < and >.

NAME

deroff, demonk, detex, delatex − remove formatting requests

SYNOPSIS

`deroff` [*option ...*] *file ...*

`demonk` [*option ...*] *file ...*

`detex` *file*

`delatex` *file*

DESCRIPTION

Deroff reads each file in sequence and removes all *nroff* and *troff*(1) requests and non-text arguments, backslash constructions, and constructs of preprocessors such as *eqn, pic,* and *tbl*(1). Remaining text is written on the standard output. *Deroff* follows files included by `.so` and `.nx` commands; if a file has already been included, a `.so` for that file is ignored and a `.nx` terminates execution. If no input file is given, *deroff* reads from standard input.

The options are

`−w` Output a word list, one 'word' (string of letters, digits, and properly embedded ampersands and apostrophes, beginning with a letter) per line. Other characters are skipped. Otherwise, the output follows the original, with the deletions mentioned above.

`−i` Ignore `.so` and `.nx` requests.

`−ms`
`−mm` Remove titles, attachments, etc., as well as ordinary *troff* constructs, from *ms*(6) or *mm* documents.

`−ml` Same as `−mm`, but remove lists as well.

Demonk removes all *monk*(1) commands and then invokes *deroff* to handle both *troff* commands and preprocessor constructs. *Demonk* follows files included by `|insert` and `|source` commands as well as *troff* `.so` and `.nx` requests. If no input file is given, *demonk* reads from standard input.

Demonk recognizes the following options and passes all options except `−b` and `−d` to *deroff* .

`−i` Ignore *monk* `|insert` and `|source` commands as well as *troff* `.so` and `.nx` requests.

`−b` Do not output blank lines resulting from the removal of *monk* commands.

`−d`*dir* Use non-standard *monk* database directory *dir.*

Detex and *delatex* do for *tex*(1) and *latex*(6) files what `deroff −w` does for *troff* files. *Delatex* largely subsumes *detex.*

SEE ALSO

troff(1), *monk*(1), *tex*(1), *spell*(1), *wwb*(1)

BUGS

These filters are not complete interpreters of *troff* or *tex.* For example, macro definitions containing `\$` cause chaos in *deroff* when the popular `$$` delimiters for *eqn* are in effect.
Text inside macros is emitted at place of definition, not place of call.

NAME

diff, diff3 − differential file comparison

SYNOPSIS

`diff` [*option ...*] *file1 file2*

`diff3` [`-ex3`] *file1 file2 file3*

DESCRIPTION

When run on regular files *diff* tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, *diff* finds a smallest sufficient set of differences. If neither file is a directory, then one may be −, meaning the standard input. If one file is a directory, then a file in that directory with basename the same as that of the other file is used.

If both files are directories, similarly named files in the two directories are compared by the method of *diff* for text files and *cmp*(1) otherwise. Options when comparing directories are:

`-r` Apply *diff* recursively to similarly named subdirectories.

`-s` Report files that are the same (normally not mentioned).

There are several options for output format; the default output format contains lines of these forms:

n1`a` *n3*`,` *n4*
n1`,` *n2*`d` *n3*
n1`,` *n2*`c` *n3*`,` *n4*

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging `a` for `d` and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs where *n1*=*n2* or *n3* = *n4* are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

`-e` Produce a script of `a`, `c`, and `d` for *ed*(1) to recreate *file2* from *file1*. When comparing directories, produce a *sh*(1) script to convert text files common to the two directories.

`-c`*n* Include *n* extra lines of context with each set of differences. The output format is modified: the output begins with identification of the files involved and their creation dates and changes are separated by lines of *'s. Lines removed from *file1* are marked with −; those added to *file2* are marked +. Lines which are changed from one file to the other are marked in both files with `!`.

`-h` Do a fast, half-hearted job, useful only when changed stretches are short and well separated, but does work on files of unlimited length.

`-b` Ignore trailing blanks (spaces and tabs) and treat other strings of blanks as if they were a single space.

`-B` Ignore all blanks.

Diff3 compares three versions of a file and publishes the various disagreeing ranges of text. One of the following indicators introduces each reported difference.

`====` All three files differ.

`====`*f* File *f* differs, where *f* is 1, 2, or 3.

Disagreeing fragments from the three files follow the `====` line, each identified by a *diff*-like range indication:

f`:` *n1*`a` File *f* lacks text that other files have; their text would be appended after line *n1*.

f`:` *n1*`,` *n2*`c`
f`:` *n1*`c` Lines *n1* through *n2* (or line *n1* only) of file *f* would have to be changed to agree with some other file. The original contents follow, unless a higher-numbered file has the same contents.

Under option −e, *diff3* publishes a script for *ed*(1) that will incorporate into *file1* all changes between *file2* and *file3*, i.e. the changes that normally would be flagged ==== and ====3. Option −x (−**3**) produces a script to incorporate only changes flagged ==== (====3).

EXAMPLES

```
(cat diff0-1 diff1-2 diff2-3; echo '1,$p') | ed - file0 >file3
```

An ancestral file0 has been kept along with a chain of version-to-version difference files made thus: diff −e file0 file1 >diff0−1. The shell command reconstructs the latest version.

```
if diff3 mod1 old mod2 | grep -s '^====$'
then :
else (diff3 -e mod1 old mod2; echo '1,$p') | ed - mod1 >new
fi
```

Compare two different modified versions with an old file. If no modifications interfere with each other (*grep* finds no ==== lines), make a new file incorporating both modifications.

FILES

/tmp/d*
/usr/lib/diffh for −h
/usr/lib/diff3

SEE ALSO

cmp(1), *comm*(1), *ed*(1), *idiff*(1)

DIAGNOSTICS

Diff yields exit status is 0 for no differences, 1 for some, 2 for trouble.

BUGS

Text lines that consist of a single '.' will defeat options −e, −x, and −3.
Superfluous output may result for files that differ only in insignificant blanks when comparing directories under option −b.
Option −c is unpardonably bizarre.

NAME

dired − directory editor

SYNOPSIS

`dired` [*option* ...] [*file*]

DESCRIPTION

Dired displays a directory listing like `ls -l` (see *ls*(1)) and allows you to prowl around the listed entries, deleting, editing, and displaying them. It requires a cursor-addressed terminal identified in environment variable `TERM`; see *term*(9.1) to simulate such terminals in *mux*(9.1).

With no *file* argument, the current directory is listed. With only one *file* argument, the argument is interpreted as a directory and it is listed. With multiple arguments, the arguments are interpreted as filenames. The options are:

`-[sr][nsrw]`
: Sort the file list by name (default), size, access time, or modification time. Ordering for `s` is increasing if by name, decreasing otherwise. Ordering is opposite for `r`.

`-w`*n*
: If *n* is `f`, use the full screen; if `h`, use half the screen (default); if a number, use *n* lines for the directory listing, reserving the rest of the screen for quick file display.

The fields of a *dired* listing are: mode, link count, owner, size, write date and name. A cursor shows the current entry.

Commands consist of single characters; arguments are prompted for at the bottom of the screen. To get a complete list, use the help command. Fuller descriptions of less obvious commands are given below.

`!`
: Prompt for a shell command. The command is executed with `%` characters in the command are replaced by the pathname of the current entry, and `#` characters by the basename.

`.`
: Repeat the previous `!` shell command, using the current entry to replace `%` or `#`.

`d`
: Mark the current entry for deletion. Deletion of a directory is recursive.

`e`
: If the current entry is a file, edit it with the editor *e*, or an editor named by the environment variable `EDITOR`. If it is a directory, invoke *dired* recursively for that directory.

`h`

`?`
: Display a help file.

`m`
: Display the current file with the pager *p*(1), or another pager named by the environment variable `PAGER`.

`q`
: Quit this directory level of *dired*. List the files marked for deletion and request confirmation before deleting them.

`t`
: Type. Display the current file. In two-window mode pause after each screenfull until you type a carriage return. The display may be interrupted.

FILES

`/usr/lib/dired` help file

SEE ALSO

ls(1)

DIAGNOSTICS

While *dired* is preparing a listing it reports ‘Reading’, and types a dot for each 10 files.

BUGS

Long lines and diagnostics can foul up the display.
Needs a command to search for a given file.
The off-line print command `p` is broken.

NAME

docgen − generate a document from a script

SYNOPSIS

`docgen` [*option* ...] [*ofile*]

DESCRIPTION

Docgen guides interactive preparation of standard documents according to canned scripts and places the output in *ofile*. The output typically takes the form of *troff*(1) input. These options invoke standard scripts:

`−mcs` (default) Bell Labs cover sheet; output (`temp.cover` by default) may be typeset thus: `troff temp.cover`

`−ms` documents using the macro package *ms*(6); output (`temp` by default) may be typeset thus: `troff −ms temp`

`−mm` similarly for the MM macro package of System V

Other options are:

`−f` *file* take script from *file*

`−v` (verbose) print document as it is generated

`−d` (debug) print information about the script as it is read

The reference tells how to construct scripts.

FILES

`/usr/lib/docgen/*` scripts
`temp.cover`
`temp`

SEE ALSO

troff(1), *ms*(6), *mcs*(6)

BUGS

Not all document types are implemented.

NAME

docsubmit − send document to library

SYNOPSIS

`docsubmit` [`-c` *cover-file*] [`-C`] [`-r` *ref-file*] [`-t`] [`-f` *copyf*] [`-d`] *file ...*

DESCRIPTION

Docsubmit sends the full text of a TM, including cover sheet, to the Bell Laboratories library network, for inclusion in the libraries' Linus database. The *files* are those that would be mentioned in a *troff* command to print the paper; files included by `.so` macros or by preprocessors such as *pic*(1) should not be mentioned. The options are

`-c` *coverfile*
: Cover sheet is in a file by itself.

`-C` : Cover sheet is in the document. One of `-c` or `-C` must be specified.

`-r` *ref-file*
: Specify a separate reference file for *refer*(1) or *prefer*(1).

`-t` : The single *file* is *tex*(1) source; only the base name, without `.tex`, should be specified.

`-f`*copyf*
: Include a copy of *copyf* in the cpio file. This flag should only be necessary for things like awk scripts executed inside the paper with .sy commands.

`-d` : Don't consided include files that can't be found as fatal errors.

Electronic submission is not a substitute for the official paper submission. For more information contact your local library or call (201)582-4840.

EXAMPLES

```
docgen cover-file
eqn cover-file file.1 file.2 | troff | lp
docsubmit -c cover-file file.1 file.2
```

FILES

`/usr/lib/docsubmit/*`

SEE ALSO

troff(1), *docgen*(1)

BUGS

Only documents that may be viewed by any AT&T employee can have their full text made available under Linus.

Documents with a complicated construction process, such as a shell script or makefile, cannot be handled directly.

NAME

doctype − guess command line for formatting a document

SYNOPSIS

`doctype` [*option* ...] [*file*]

DESCRIPTION

Doctype guesses and prints on the standard output the command line for printing a document that uses *troff*(1), related preprocessors like *eqn*(1), and the *ms*(6) and *mm* macro packages.

Option `-n` invokes *nroff* instead of *troff*. Other options are passed to *troff*.

EXAMPLES

```
eval `doctype chapter.?` | apsend
```

Typeset files named chapter.0, chapter.1, ...

SEE ALSO

troff(1), *eqn*(1), *tbl*(1), *refer*(1), *prefer*(1), *pic*(1), *ideal*(1), *grap*(1), *ped*(9.1), *mcs*(6), *ms*(6), *man*(6)

BUGS

It's pretty dumb about guessing the proper macro package.

NAME

du, df − disk usage

SYNOPSIS

`du` [`-s`] [`-a`] [*file* ...]

`df` [`-i`] [*special* ...] [*file* ...]

DESCRIPTION

Du gives the number of Kbytes allocated to data blocks of named *files* and, recursively, of files in named directories. If *name* is missing, **.** is used. The count for a directory includes the counts of the contained files and directories. The options are

`-s` Print only the grand total.

`-a` Print a count for every file in a directory. Normally counts are printed only for contained directories.

A file which has two (hard) links to it is counted only once. Symbolic links are neither counted nor followed.

Df printsthe amount of free space on the file system contained in *special*, or on the file system in which the specified *file* is contained. If no file system is specified, the free space on all of the currently mounted file systems is printed.

The reported numbers are in Kbytes, independently of the blocksize actually used on the file system. The option is

`-i` Report also on free and used inodes.

EXAMPLES

`df .`

How much space is there where I'm working?

FILES

`/etc/fstab`	only for the root device
`/etc/mtab`	list of currently mounted file systems

SEE ALSO

quot(8), *fstab*(5), *icheck*(8)

BUGS

In the absence of option `-a` non-directories given as arguments to *du* are not listed.
If there are too many distinct linked files, *du* counts the excess files multiply.
Unwritten holes in files count as if real data were present, and indirect blocks are not counted.

NAME

echo, printf − print arguments

SYNOPSIS

`echo` [`−n`] [`−e`] [*arg* ...]

`printf` [*format* [*arg...*]]

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a newline on the standard output. Option `−n` suppresses the newline.

Option `−e` enables the interpretation of C-style escape codes, `\b`, `\f`, `\n`, `\r`, `\t`, `\\`, and `\`*ddd*, where *d* is an octal digit, plus the special code `\c`, which terminates the output.

Echo is useful for producing diagnostics in shell programs and for writing constant data on pipes.

Printf behaves like the library function of the same name; each *arg* is printed on the standard output according to the corresponding %-introduced specification in the *format* string. The standard C escape sequences `\n`, `\r`, `\t`, `\b`, and `\`*digits* are recognized in *format*. The *arg* will be treated as a string if the corresponding format is `s`; otherwise it is evaluated as a C constant, with the following extensions:

A leading plus or minus is allowed.

If the leading character is a single or double quote, the value is the ASCII code of the next character.

Otherwise, if the leading character is not a digit, the value is its ASCII code.

The format string is reused as often as necessary to satisfy the *arg*'s. Any extra format specifications are evaluated with zero or the null string.

EXAMPLES

```
echo "can't open file" $1 1>&2
```

Send a message to the standard error file.

SEE ALSO

printf(3)

BUGS

Printf has no diagnostics for illegal syntax.

NAME

ed, e − text editor

SYNOPSIS

`ed` [`-`] [`-o`] [*file*]

DESCRIPTION

Ed is the standard text editor; *e* is another name for it.

If a *file* argument is given, *ed* simulates an `e` command (see below) on that file: it is read into *ed's* buffer so that it can be edited. The options are

- `-` Suppress the printing of character counts by `e`, `r`, and `w` commands and of the confirming `!` by `!` commands.
- `-o` (for output piping) Place on the standard error file all output except writing by `w` commands. If no *file* is given, make `/dev/stdout` the remembered file; see the `e` command below.

Ed operates on a 'buffer', a copy of the file it is editing; changes made in the buffer have no effect on the file until a `w` (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*.

Commands to *ed* have a simple and regular structure: zero or more *addresses* followed by a single character *command*, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Missing addresses are supplied by default.

In general, only one command may appear on a line. Certain commands allow the addition of text to the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, no commands are recognized; all input is merely collected. Input mode is left by typing a period `.` alone at the beginning of a line.

Ed supports a limited form of *regular expression* notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. In the following specification for regular expressions the word 'character' means any character but newline.

1. Any character except a special character matches itself. Special characters are the regular expression delimiter plus `\[.` and sometimes `^*$`.
2. A `.` matches any character.
3. A `\` followed by any character except a digit, `(`, or `)` matches that character.
4. A nonempty string *s* bracketed `[` *s* `]` (or `[^` *s* `]`) matches any character in (or not in) *s*. In *s*, `\` has no special meaning, and `]` may only appear as the first letter. A substring *a*`-`*b*, with *a* and *b* in ascending ASCII order, stands for the inclusive range of ASCII characters.
5. A regular expression of form 1-4 followed by `*` matches a sequence of 0 or more matches of the regular expression.
6. A regular expression, *x*, of form 1-8, bracketed `\(` *x* `\)` matches what *x* matches.
7. A `\` followed by a digit *n* matches a copy of the string that the bracketed regular expression beginning with the *n*th `\(` matched.
8. A regular expression of form 1-8, *x*, followed by a regular expression of form 1-7, *y* matches a match for *x* followed by a match for *y*, with the *x* match being as long as possible while still permitting a *y* match.
9. A regular expression of form 1-8, or a null string, preceded by `^` (and/or followed by `$`), is constrained to matches that begin at the left (and/or end at the right) end of a line.
10. A regular expression of form 1-9 picks out the longest among the leftmost matches in a line.

11. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in one command (see *s* below) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\'. This also applies to the character bounding the regular expression (often `/`) and to `\` itself.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; however, the exact effect on the current line is discussed under the description of the command. Addresses are constructed as follows.

1. The character `.`, customarily called 'dot', addresses the current line.
2. The character `$` addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. `'x` addresses the line marked with the name *x*, which must be a lower-case letter. Lines are marked with the `k` command described below.
5. A regular expression enclosed in slashes `/` addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the beginning of the buffer.
6. A regular expression enclosed in queries `?` addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the end of the buffer.
7. An address followed by a plus sign + or a minus sign − followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.
8. An address followed by + (or −) followed by a regular expression enclosed in slashes specifies the first matching line following (or preceding) that address. The search wraps around if necessary. The + may be omitted, so `0/x/` addresses the *first* line in the buffer with an `x`. Enclosing the regular expression in `?` reverses the search direction.
9. If an address begins with + or − the addition or subtraction is taken with respect to the current line; e.g. `-5` is understood to mean `.-5`.
10. If an address ends with + or −, then 1 is added (resp. subtracted). As a consequence of this rule and rule 9, the address − refers to the line before the current line. Moreover, trailing + and − characters have cumulative effect, so −− refers to the current line less 2.
11. To maintain compatibility with earlier versions of the editor, the character ^ in addresses is equivalent to −.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma `,`. They may also be separated by a semicolon `;`. In this case the current line is set to the previous address before the next address is interpreted. If no address precedes a comma or semicolon, line 1 is assumed; if no address follows, the last line of the buffer is assumed. The second address of any two-address sequence must correspond to a line following the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the default. 'Dot' means the current line.

`(.)a`

<text>
.
Read the given text and append it after the addressed line. Dot is left on the last line input, if there were any, otherwise at the addressed line. Address `0` is legal for this command; text is placed at the beginning of the buffer.

`(.,.)b[+-][`*pagesize*`][pln]`
Browse. Print a 'page', normally 20 lines. The optional + (default) or − specifies whether the next or previous page is to be printed. The optional *pagesize* is the number of lines in a page. The optional `p`, `n`, or `l` causes printing in the specified format, initially `p`. Pagesize and format are remembered between `b` commands. Dot is left at the last line displayed.

`(.,.)c`
<text>
.
Change. Delete the addressed lines, then accept input text to replace these lines. Dot is left at the last line input; if there were none, it is left at the line preceding the deleted lines.

`(.,.)d`
Delete the addressed lines from the buffer. Dot is set to the line following the last line deleted, or to the last line of the buffer if the deleted lines had no successor.

`e` *filename*
Edit. Delete the entire contents of the buffer; then read the named file into the buffer. Dot is set to the last line of the buffer. The number of characters read is typed. The file name is remembered for possible use in later `e`, `r`, or `w` commands. If *filename* is missing, the remembered name is used.

`E` *filename*
Unconditional `e`; see 'DIAGNOSTICS' below.

`f` *filename*
Print the currently remembered file name. If *filename* is given, the currently remembered file name is first changed to *filename.*

`(1,$)g/`*regular expression*`/`*command list*
`(1,$)g/`*regular expression*`/`
`(1,$)g/`*regular expression*
Global. First mark every line which matches the given *regular expression.* Then for every such line, execute the *command list* with dot initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must end with `\`. The '`.`' terminating input mode for an `a`, `i`, `c` command may be omitted if it would be on the last line of the command list. The commands `g` and `v` are not permitted in the command list. Any character other than space or newline may be used instead of `/` to delimit the regular expression. The second and third forms mean `g/`*regular expression*`/p`.

`(.)i`
<text>
.
Insert the given text before the addressed line. Dot is left at the last line input, or, if there were none, at the line before the addressed line. This command differs from the *a* command only in the placement of the text.

`(.,.+1)j`
Join the addressed lines into a single line; intermediate newlines are deleted. Dot is left at the resulting line.

`(.)k`*x* Mark the addressed line with name *x*, which must be a lower-case letter. The address form `'`*x*

then addresses this line.

(.,.)l
: List. Print the addressed lines in an unambiguous way: a tab is printed as \t, a backspace as \b, backslashes as \\, and non-printing characters are printed as a backslash followed by three octal digits. Long lines are folded, with the second and subsequent sub-lines indented one tab stop. If the last character in the line is a blank, it is followed by \n. An l may be appended, like p, to any non-I/O command.

(.,.)m*a*
: Move. Reposition the addressed lines after the line addressed by *a*. Dot is left at the last moved line.

(.,.)n
: Number. Perform p, prefixing each line with its line number and a tab. An n may be appended, like p, to any non-I/O command.

(.,.)p
: Print the addressed lines. Dot is left at the last line printed. A p appended to any non-I/O command causes the then current line to be printed after the command is executed.

(.,.)P
: This command is a synonym for p.

q
: Quit the editor. No automatic write of a file is done.

Q
: Quit unconditionally; see 'DIAGNOSTICS' below.

($)r *filename*
: Read in the given file after the addressed line. If no *filename* is given, the remembered file name is used. The file name is remembered if there were no remembered file name already. If the read is successful, the number of characters read is typed. Dot is left at the last line read in from the file.

(.,.)s*n*/*regular expression*/*replacement*/
(.,.)s*n*/*regular expression*/*replacement*/g
(.,.)s*n*/*regular expression*/*replacement*
: Substitute. Search each addressed line for an occurrence of the specified regular expression. On each line in which *n* matches are found (*n* defaults to 1 if missing), the *n*th matched string is replaced by the replacement specified. If the global replacement indicator g appears after the command, all subsequent matches on the line are also replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or newline may be used instead of / to delimit the regular expression and the replacement. Dot is left at the last line substituted. The third form means s*n*/*regular expression*/*replacement*/p. The second / may be omitted if the replacement is empty.

  An ampersand & appearing in the replacement is replaced by the string matching the regular expression. The characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression enclosed between \(and \). When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left.

  A literal &, /, \ or newline may be included in a replacement by prefixing it with \.

(.,.)t*a*
: Transfer. Copy the addressed lines after the line addressed by *a*. Dot is left at the last line of the copy.

(.,.)u
: Undo. Restore the preceding contents of the current line, which must be the last line in which a substitution was made.

(1,$)v/*regular expression*/*command list*
(1,$)v/*regular expression*/
(1,$)v/*regular expression*
: This command is the same as the global command g except that the command list is executed with dot initially set to every line *except* those matching the regular expression.

(1,$)w *filename*
: Write the addressed lines onto the given file. If the file does not exist, it is created with mode 666 (readable and writable by everyone). If no *filename* is given, the remembered file name, if any, is used. The file name is remembered if there were no remembered file name already. Dot is unchanged. If the write is successful, the number of characters written is printed.

(1,$)W *filename*
: Perform w, but append to, instead of overwriting, any existing file contents.

($)= Print the line number of the addressed line. Dot is unchanged.

!*shell command*
: Send the remainder of the line after the ! to *sh*(1) to be interpreted as a command. Dot is unchanged.

(.+1)<newline>
: An address without a command is taken as a p command. A terminal / may be omitted from the address. A blank line alone is equivalent to .+1p; it is useful for stepping through text.

If an interrupt signal (ASCII DEL) is sent, *ed* prints a ? and returns to its command level.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last newline. It refuses to read files containing non-ASCII characters.

FILES

/tmp/e*
ed.hup work is saved here if terminal hangs up

SEE ALSO

sam(9.1), *sed*(1), *vi*(1)

DIAGNOSTICS

?*name* for inaccessible file; ?TMP for temporary file overflow; ? for errors in commands or other overflows.

To protect against throwing away valuable work, a q or e command is considered to be in error if the buffer has been modified since the last w, q, or e command.

NAME

eqn, neqn, checkeq − typeset mathematics

SYNOPSIS

`eqn` [*option* ...] [*file* ...]

`neqn` [*option* ...] [*file* ...]

`checkeq` [*file* ...]

DESCRIPTION

Eqn is a *troff*(1) preprocessor for typesetting mathematics on a phototypesetter, *neqn* on terminals. Usage is almost always

```
eqn file ... | troff
neqn file ... | nroff
```

If no files are specified, these programs read from the standard input. *Eqn* prepares output for the typesetter named in the `-T`*dest* option (Mergenthaler Linotron 202 default, see *troff*(1)). When run with other preprocessor filters, *eqn* usually comes last.

A line beginning with `.EQ` marks the start of an equation; the end of an equation is marked by a line beginning with `.EN`. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; text between delimiters is also *eqn* input. Delimiters may be set to characters *x* and *y* with the option `-d`*xy* or (more commonly) with `delim` *xy* between `.EQ` and `.EN`. Left and right delimiters may be identical. (They are customarily taken to be `$$`). Delimiters are turned off by `delim off`. All text that is neither between delimiters nor between `.EQ` and `.EN` is passed through untouched.

Checkeq reports missing or unbalanced delimiters and `.EQ`/`.EN` pairs.

Tokens within *eqn* are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character like `x` could appear, a complicated construction enclosed in braces may be used instead. Tilde ~ represents a full space in the output, circumflex ^ half as much.

Subscripts and superscripts are produced with the keywords `sub` and `sup`. Thus `x sub i` makes x_i, `a sub i sup 2` produces a_i^2, and `e sup {x sup 2 + y sup 2}` gives $e^{x^2+y^2}$.

Fractions are made with `over`: `a over b` yields $\frac{a}{b}$.

`sqrt` makes square roots: `1 over sqrt {ax sup 2 +bx+c}` results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords `from` and `to` introduce lower and upper limits on arbitrary things: $\lim_{n\to\infty} \sum_{0}^{n} x_i$ is made with `lim from {n -> inf} sum from 0 to n x sub i .`

Left and right brackets, braces, etc., of the right height are made with `left` and `right`: `left [ x sup 2 + y sup 2 over alpha right ] ~=~1` produces $\left[x^2 + \frac{y^2}{\alpha}\right] = 1$. The `right` clause is optional. Legal characters after `left` and `right` are braces, brackets, bars, `c` and `f` for ceiling and floor, and `""` for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with `pile`, `lpile`, `cpile`, and `rpile`: `pile {a above b above c}` produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile. `lpile` left-justifies, `pile` and `cpile` center, with different vertical spacing, and `rpile` right justifies.

Matrices are made with `matrix`: `matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }` produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is `rcol` for a right-justified column.

Diacritical marks are made with `prime`, `dot`, `dotdot`, `hat`, `tilde`, `bar`, `under`, `vec`, `dyad`, and `under`: `x sub 0 sup prime = f(t) bar + g(t) under` is $x_0' = \overline{f(t)} + \underline{g(t)}$, and `x vec = y dyad` is $\vec{x} = \overleftrightarrow{y}$.

Sizes and font can be changed with prefix operators `size` *n*, `size` ±*n*, `fat`, `roman`, `italic`, `bold`, or `font` *n*. Size and fonts can be changed globally in a document by `gsize` *n* and `gfont` *n*, or by the command-line arguments `-s`*n* and `-f`*n*.

Normally subscripts and superscripts are reduced by 3 point sizes from the previous size; this may be changed by the command-line argument `-p`*n*.

Successive display arguments can be lined up. Place `mark` before the desired lineup point in the first equation; place `lineup` at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with `define`: `define` *thing* `%` *replacement* `%` defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The `%` may be any character that does not occur in `replacement`.

Keywords like `sum` ($\sum$) `int` ($\int$) `inf` (∞) and shorthands like `>=` ($\geq$) `->` ($\rightarrow$), and `!=` ($\neq$) are recognized. Greek letters are spelled out in the desired case, as in `alpha` or `GAMMA`. Mathematical words like `sin`, `cos`, `log` are made Roman automatically. *Troff*(1) four-character escapes like `\(lh` (☜) can be used anywhere. Strings enclosed in double quotes `" "` are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff* when all else fails.

SEE ALSO

troff(1), *tbl*(1), *ms*(6), *eqnchar*(6), *doctype*(1)
B. W. Kernighan and L. L. Cherry, 'Typesetting Mathematics—User's Guide', this manual, Volume 2
J. F. Ossanna and B. W. Kernighan, 'NROFF/TROFF User's Manual', *ibid.*

BUGS

To embolden digits, parens, etc., it is necessary to quote them, as in `bold "12.3"`.

NAME

expr − integer and string-match expression evaluator for shell scripts

SYNOPSIS

`expr` *arg ...*

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument.

The operators and keywords are listed below in order of increasing precedence, with equal precedence operators grouped.

expr1 `|` *expr2*
: Value is the value of *expr1* if that is neither empty nor 0, otherwise the value of *expr2*.

expr1 `&` *expr2*
: Value is the value of *expr1* if neither *expr1* nor *expr2* is empty or 0, otherwise 0.

expr1 relop expr2
: *Relop* is one of `< <= = != >= > .` Value is 1 if the indicated comparison is true, 0 if false. The comparison is numeric if both *expr* are integers, otherwise lexicographic.

expr1 `+` *expr2*
expr1 `−` *expr2*
: Value is the sum or difference of the (integer) values of *expr1* and *expr2*.

expr1 `*` *expr2*
expr1 `/` *expr2*
expr1 `%` *expr2*
: Value is the product, quotient, or remainder of the (integer) values of *expr1* and *expr2*.

expr `:` *regexp*
: Match the string value of *expr* with the regular expression *regexp*; regular expression syntax is the same as in *ed*(1), but matches are anchored at the left. On success a subexpression `\(...\)`, if present in *regexp*, picks out a return value from the matched string. Otherwise, the matching operator yields the number of characters matched (0 on failure).

`(` *expr* `)` Parentheses for grouping.

arg Value is the string *arg*.

EXAMPLES

```
a=`expr $a + 1`
```
Add 1 to shell variable *a*.

```
expr $a : '.*/\(.*\)' '|' $a
```
Same as `basename $a`.

SEE ALSO

sh(1), *test*(1)

DIAGNOSTICS

Expr returns exit code 0 if the expression is neither null nor 0, 1 if the expression is null or 0, 2 for invalid expressions.

NAME

f2c – convert Fortran 77 to C

SYNOPSIS

`f2c` [*option* ...] *file* ...

DESCRIPTION

F2c converts Fortran 77 source code to C. Arguments that end with `.f` or `.F` are taken to be Fortran 77 source programs; each is converted to a C source file in the current directory, with `.c` substituted for the final `.f` or `.F`. If no *file* arguments are given, then *f2c* reads Fortran 77 on its standard input and writes the resulting C on its standard output.

The options are

`-w`, `-w66`, `-onetrip`, `-u`, `-C`, `-I2`, `-U`
: Same as in *f77*(1).

`-A` Use ANSI C constructs. The default is old-style (pre-ANSI) C.

`-R` Do not promote REAL functions and operations to DOUBLE PRECISION (as *f77* does).

`-i2` Similar to `-I2`, but assume a modified *libF77* and (compiled with `-Df2c_i2`), so INTEGER and LOGICAL variables may be assigned in INQUIREs.

`-a` Give local variables automatic (rather than static) storage unless they appear in a DATA, EQUIVALENCE, or SAVE statement.

`-c` Include original Fortran as comments.

`-g` Include original line numbers as comments.

`-r8` Promote REAL to DOUBLE PRECISION, COMPLEX to DOUBLE COMPLEX.

`-z` Do not implicitly recognize DOUBLE COMPLEX.

`-w8` Suppress warnings when COMMON or EQUIVALENCE forces non-double alignment of doubles.

The resulting C code invokes routines from *f77*'s support libraries, *libI77* and *libF77*; object code should be loaded by *f77* or with *ld*(1) or *cc*(1) options `-lI77 -lF77 -lm`. Calling conventions are those of *f77*; see the reference.

FILES

`*.[fF]`
`*.c`

SEE ALSO

S. I. Feldman and P. J. Weinberger, 'A Portable Fortran 77 Compiler', this manual, Volume 2

BUGS

Floating-point constants of over 17 significant digits are rounded to 17 significant digits.

NAME

f77 – Fortran 77 compiler

SYNOPSIS

`f77` [*option* ...] *file* ...

DESCRIPTION

F77 is a Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with `.f` are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with `.o` substituted for `.f`.

Arguments whose names end with `.r` or `.e` are taken to be Ratfor or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by *f77*.

In the same way, arguments whose names end with `.c` or `.s` are taken to be C or assembly source programs and are compiled or assembled, producing a `.o` file.

The following options have the same meaning as in *cc*(1). See *ld*(1) for load-time options.

`-c` Suppress loading and produce `.o` files for each source file.

`-g` Have the compiler produce additional symbol table information for *sdb*(A) or *pi*(9.1).

`-w` Suppress all warning messages. If the option is `-w66`, only Fortran 66 compatibility warnings are suppressed.

`-p` Prepare object files for profiling, see *prof*(1).

`-O` Invoke an object-code optimizer.

`-S` Compile the named programs, and leave the assembler-language output on corresponding files suffixed `.s`. (No `.o` is created.).

`-o` *output*
Name the final output file *output* instead of `a.out`.

The following options are peculiar to *f77*.

`-onetrip`
Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

`-u` Make the default type of a variable ‘undefined’ rather than using the default Fortran rules.

`-C` Compile code to check that subscripts are within declared array bounds.

`-I2` Render INTEGER and LOGICAL as short, INTEGER*4 as long. Allow only INTEGER*4 (and no LOGICAL) variables in INQUIREs.

`-U` Honor the case of variable and external names. Fortran keywords must be in lower case.

`-F` Apply EFL and Ratfor preprocessors to relevant files, put the results in the files with the suffix changed to `.f`, but do not compile.

`-m` Apply the M4 preprocessor to each `.r` or `.e` file before transforming it with the Ratfor or EFL preprocessor.

`-E`*x* Use the string *x* as an EFL option in processing `.e` files.

`-R`*x* Use the string *x* as a Ratfor option in processing `.r` files.

Other arguments are taken to be either loader option arguments, or F77-compatible object programs, typically produced by an earlier run, or perhaps libraries of F77-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name `a.out`.

FILES

file`.[fresc]`	input file
`*.o`	object file
`a.out`	loaded output
`./fort*`	temporary
`/usr/lib/f77pass1`	compiler
`/lib/f1`	pass 2
`/lib/c2`	optional optimizer
`/usr/lib/libF77.a`	intrinsic function library
`/usr/lib/libI77.a`	Fortran I/O library
`/lib/libc.a`	C library, see section 3

SEE ALSO

prof(1), *cc*(1), *ld*(1), *efl*(A), *ratfor*(A)
S. I. Feldman and P. J. Weinberger, 'A Portable Fortran 77 Compiler', this manual, Volume 2

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

NAME

factor, qfactor, primes – factor a number, generate large primes

SYNOPSIS

`factor` [*number*]

`qfactor`

`primes` [*start* [*finish*]]

DESCRIPTION

Factor prints *number* and its prime factors, each repeated the proper number of times. The number must be positive and less than 2^{56} (about 7.2×10^{16}).

If no *number* is given, *factor* reads a stream of numbers from the standard input and factors them. It exits on any input not a positive integer. Maximum running time is proportional to $\sqrt{n}$.

Qfactor reads one number from the standard input and factors it. It will factor numbers up to about 40 digits. For large numbers it is much faster than *factor*.

Primes prints the prime numbers ranging from *start* to *finish*, where *start* and *finish* are positive numbers less than 2^{56}. If *finish* is missing, *primes* prints without end; if *start* is missing, it reads the starting number from the standard input.

NAME

file – determine file type

SYNOPSIS

`file` *file* ...

`file -f` *names*

DESCRIPTION

File performs a series of tests on a set of files in an attempt to classify their contents by language or purpose. The names may be enumerated or contained in the file specified by the `-f` option.

BUGS

It can make mistakes, for example classifying a file of decimal data, `.01`, `.02`, etc. as *troff*(1) input.

NAME

find − find files

SYNOPSIS

`find` *pathname ... expression*

DESCRIPTION

Find recursively descends the directory hierarchy for each *pathname*, seeking files that match a boolean *expression*, which consists of one or more arguments. It does not follow symbolic links. In the following descriptions of primary expressions, *n* is a decimal integer; *+n* may be written to specify more than *n* and *−n* to specify less.

`−name` *filename*
: True if the *filename* argument matches the current file name. Normal shell filename metacharacters may be used if quoted.

`−perm` *onum*
: True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more mode bits (017777, see *stat*(2)) become significant and the modes are compared: *(mode&onum)==onum*.

`−type` *c* True if the type of the file is *c*, where *c* is `b`, `c`, `d`, `f`, or `L` for block special file, character special file, directory, plain file or symbolic link.

`−links` *n* True if the file has *n* links.

`−user` *uname*
: True if the file belongs to the user *uname* (login name or numeric userid).

`−group` *gname*
: True if the file belongs to group *gname* (group name or numeric groupid).

`−size` *n* True if the file is *n* blocks long (512 bytes per block).

`−inum` *n* True if the file has inode number *n.*

`−atime` *n* True if the file has been accessed in *n* days.

`−mtime` *n* True if the file has been modified in *n* days.

`−ctime` *n* True if the inode has been changed in *n* days.

`−exec` *command*
: True if the executed command returns a zero value as exit status. The end of the command must be punctuated by an escaped semicolon. A command argument `{}` is replaced by the current pathname.

`−ok` *command*
: Like `−exec` except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response `y`.

`−print` Always true; causes the current pathname to be printed.

`−newer` *file*
: True if the file has been modified more recently than the argument *file.*

`−status` *n*
: True if *lstat* (see *stat*(2)) applied to the file yields error number *n*; see *intro*(2). Testing `−status` turns off diagnostics that errors normally produce. On ordinary systems a nonzero error number occurs when a file disappears underfoot or a file system is in trouble.

The following operators, listed in order of decreasing precedence, may be used to combine primary expressions.

`(` *expression* `)`
Group with parentheses.

`!` *expression*
Negation. True if and only if *expression* is not true.

expression expression
Conjunction. True if both expressions are true.

expression `-o` *expression*
Disjunction. True if either expression is true.

EXAMPLES

```
find / \( -name a.out -o -name '*.o' \) -atime +7 -exec rm '{}' \;
```
Remove all files named `a.out` or `*.o` that have not been accessed for a week.

FILES

`/etc/passwd`
`/etc/group`

SEE ALSO

sh(1), *test*(1), *filsys*(5)

NAME

fmt – ultra-simple text formatter

SYNOPSIS

`fmt` [*option* ...] [*file* ...]

DESCRIPTION

Fmt copies the given *files* (standard input by default) to its standard output, filling and indenting lines. The options are

`-l` *n* Output line length is *n*, including indent, (default 70).

`-i` *n* Indent *n* spaces (default 0).

Empty lines and initial white space in input lines are preserved. Empty lines are inserted between input files.

Fmt is idempotent: it leaves already formatted text unchanged.

BUGS

Words longer than 256 characters are split.

NAME

games, demo − some playthings

SYNOPSIS

`/usr/jerq/bin/demo` [*game*]

Labyrinth games
`adventure`
`zork`
`rogue`
`wump`

Card games
`fish`
`canfield`
`bridge` [*arg ...*]
`mille`

Board games
`back`

Word games
`hangman` [`-a`]
`word_clout`
`festoon` *length percent*

System games
`imp`
`tso`

War games
`mars` [`-dfhmp`] [`-cqs`*value*] *file ...*
`ogre` [*type*]
`warp`

Games of speed
`atc`
`snake`
`worm`

Educational games
`quiz` [`-i` *file*] [`-t`] [*question answer*]
`arithmetic` [`+-x/`] [*range*]

Creative games
`/usr/jerq/bin/twid`
`banner`

Sayings
`fortune` [*file*]
`doctor`
`say` [*N*]

Coding games
`bcd` *text*
`ppt`
`morse`
`/usr/bin/number`

Out-of-layer games
`/usr/jerq/bin/pen`
`/usr/jerq/bin/crabs` [`-i`] \
[`-s` *gracetime*] [`-v` *speed*] [*n*]

DESCRIPTION

Game programs exist sporadically on various machines. Manuals for many of them may be obtained by using *man*(1). For example, to see manual pages for *atc* and *twid*, type `man atc twid`. Unless shown otherwise, games live in `/usr/games`, which may be put in your shell `PATH` to reach them more conveniently. Some need a cursor-addresses terminal; see the appropriate manual pages and *term*(9.1).

Demo and other games found in `/usr/jerq/bin` need a Teletype 5620 terminal running under *mux*(9.1). *Demo* comprises many games, which are listed when it is invoked without an argument. Experiment with the mouse to find out they work. Some unobvious interactions are these:

> *Swar* is for two players, one using `asdwx` on the keyboard, the other `12350` on the keypad.
> *Pacman* is controlled by `hjkl` keys as in *vi*(1).

Here is a list of some *demo* games.

Watch the time
`clock`

War games
`swar`

Games of speed
`gebam`
`pengo`
`centipede`
`asteroids`
`pacman`

Educational games
`maxwell`

Out-of-layer games
`tracks`
`pogo`
`magnet`

Animation
EWD
road
juggle
ball
fence

Movies
horse
dodec
explode
arno

Patterns
bounce
moire
fireworks
rose
disc
lunch

NAME

getuid, id – get user identity

SYNOPSIS

`getuid` [–] [*arguments*]

`id`

DESCRIPTION

Getuid prints on its standard output information about its invoker, based on the effective user id, as presented in the password file. With no arguments, *getuid* prints the login id of its invoker. Arguments select which information to print:

`user`	login id
`group`	group id
`passwd`	encrypted password
`uid`	numerical user id
`gid`	numerical group id
`acct`	comp center account number
`bin`	comp center output bin
`home`	home directory
`shell`	default shell

If the optional – or more than one argument is present, the information is displayed in the form

user=name

as suitable for setting environment variables in the shell.

Id prints the effective userid and groupid and the login name. The userid and groupid are printed numerically and, if possible, textually.

FILES

`/etc/passwd`

`/etc/group`

SEE ALSO

who(1), *getuid*(2), *passwd*(5), *newgrp*(1)

BUGS

Getuid reports the default group for the user, not the current effective group id.
The login id reported is the first one in the password file with the correct numerical user id, not necessarily the login for the current session.

NAME

grap – pic preprocessor for drawing graphs

SYNOPSIS

`grap` [*file* ...]

DESCRIPTION

Grap is a *pic*(1) preprocessor for drawing graphs on a typesetter. Graphs are surrounded by the *troff* 'commands' `.G1` and `.G2`. Data are scaled and plotted, with tick marks supplied automatically. Commands exist to modify the frame, add labels, override the default ticks, change the plotting style, define coordinate ranges and transformations, and include data from files. In addition, *grap* provides the same loops, conditionals and macro processing that *pic* does.

`frame ht` *e* `wid` *e* `top dotted` ...: Set the frame around the graph to specified `ht` and `wid`; default is 2 by 3 (inches). The line *styles* (`dotted`, `dashed`, `invis`, `solid` (default)) of the *sides* (`top`, `bot`, `left`, `right`) of the frame can be set independently.

`label` *side* `"a label" "as a set of strings"` *adjust*: Place label on specified side; default side is bottom. *adjust* is `up` (or `down left right`) *expr* to shift default position; `width` *expr* sets the width explicitly.

`ticks` *side* `in at` *optname expr, expr, ...*: Put ticks on *side* at *expr, ...*, and label with "*expr*". If any *expr* is followed by "...", label tick with "...", and turn off all automatic labels. If "..." contains `%f`'s, they will be interpreted as `printf` formatting instructions for the tick value. Ticks point `in` or `out` (default out). Tick iterator: instead of `at` ..., use `from` *expr* `to` *expr* `by` *op expr* where *op* is optionally `+-*/` for additive or multiplicative steps. `by` can be omitted, to give steps of size 1. If no ticks are requested, they are supplied automatically; suppress this with `ticks off`. Automatic ticks normally leave a margin of 7% on each side; set this to anything by `margin` = *expr*.

`grid` *side linedesc* `at` *optname expr, expr, ...*: Draw grids perpendicular to *side* in style *linedesc* at *expr, ...*. Iterators and labels work as with ticks.

`coord` *optname* `x` *min, max* `y` *min, max* `log x log y`: Set range of coords and optional log scaling on either or both. This overrides computation of data range. Default value of *optname* is current coordinate system (each `coord` defines a new coordinate system).

`plot` "*str*" `at` *point*; "*str*" `at` *point*: Put *str* at *point*. Text position can be qualified with `rjust`, `ljust`, `above`, `below` after "...".

`line from` *point* `to` *point linedesc*: Draw line from here to there. `arrow` works in place of `line`.

`next` *optname* `at` *point linedesc*: Continue plot of data in *optname to point*; default is current.

`draw` *optname linedesc* ...: Set mode for `next`: use this style from now on, and plot "..." at each point (if given).

`new` *optname linedesc* ...: Set mode for `next`, but disconnect from previous.

A list of numbers *x y1 y2 y3* ... is treated as `plot bullet at` *x,y1*; `plot bullet at` *x,y2*; etc., or as `next at` *x,y1* etc., if `draw` is specified. Abscissae of 1,2,3,... are provided if there is only one input number per line.

A point *optname expr, expr* maps the point to the named coordinate system. A *linedesc* is one of `dot dash invis solid` optionally followed by an expression.

`define` *name* { *whatever* }*:* Define a macro. There are macros already defined for standard plotting symbols like `bullet`, `circle`, `star`, `plus`, etc., in `/usr/lib/grap.defines`, which is included if it exists.

var = *expr*: Evaluate an expression. Operators are `+ - * /` and `.` Functions are `log` and `exp` (both base 10), `sin`, `cos`, `sqrt`; `rand` returns random number on [0,1); `max(`*e*`,`*e*`)`, `min(`*e*`,`*e*`)`, `int(`*e*`)`.

`print` *expr*; `print "..."`: As a debugging aid, print *expr or string* on the standard error.

`copy "`*filename*`"`: Include this file right here.

`copy thru` *macro*: Pass rest of input (until `.G2`) through *macro*, treating each field (non-blank, or "...") as an argument. *macro* can be the name of a macro previously defined, or the body of one in place, like `/plot $1 at $2,$3/`.

`copy thru` *macro* `until "`*string*`"`: Stop copy when input is *string* (left-justified).

`pic` *remainder of line*: Copy to output with leading blanks removed.

`graph` *Name pic-position*: Start a new frame, place it at specified position, e.g., `graph Thing2 with .sw at Thing1.se + (0.1,0)`. *Name* must be capitalized to keep *pic* happy.

`.`*anything at beginning of line*: Copied verbatim.

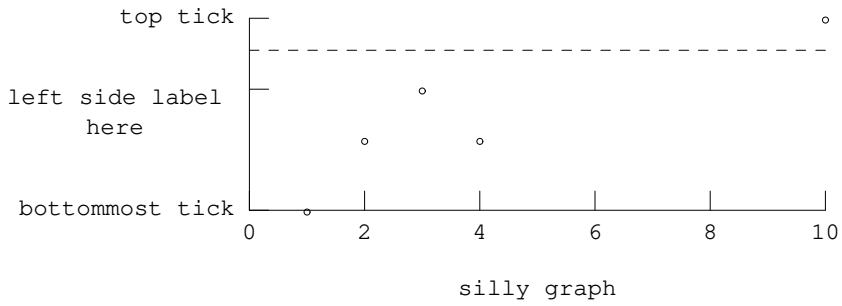
`sh %`*anything* `%`: Pass everything between the `%`'s to the shell; as with macros, `%` may be any character and *anything* may include newlines.

`#` *anything*: A comment, which is discarded.

Order is mostly irrelevant; no category is mandatory. Any arguments on the `.G1` line are placed on the generated `.PS` line for *pic*.

EXAMPLES

```
.G1
frame ht 1 top invis right invis
coord x 0, 10 y 1, 3 log y
ticks left in at 1 "bottommost tick", 2,3 "top tick"
ticks bot in from 0 to 10 by 2
label bot "silly graph"
label left "left side label" "here"
grid left dashed at 2.5
copy thru / circle at $1,$2 /
1 1
2 1.5
3 2
4 1.5
10 3
.G2
```



FILES

`/usr/lib/grap.defines` definitions of standard plotting characters, e.g., bullet

SEE ALSO

graph(1), *pic*(1), *troff*(1), *plot*(3)

J. L. Bentley and B. W. Kernighan, 'GRAP—A Language for Typesetting Graphs', this manual, Volume 2

NAME

graph − draw a graph

SYNOPSIS

`graph` [*option* ...]

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas (*x*-values) and ordinates (*y*-values) of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by *plot*(1) filters.

If the ordinate of a point is followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes `" "` in which case they may be empty or contain blanks and numbers; labels never contain newlines.

The following options are recognized, each as a separate argument.

`−a` Supply abscissas automatically; no *x*-values appear in the input. Spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0, or 1 with a log scale in *x*, or the lower limit given by `−x`).

`−b` Break (disconnect) the graph after each label in the input.

`−c` Character string given by next argument is default label for each point.

`−g` Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

`−l` Next argument is a legend to title the graph. Grid ranges are automatically printed as part of the title unless a `−s` option is present.

`−m` Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected. Some devices give distinguishable line styles for other small integers. Mode −1 (default) begins with style 1 and rotates styles for successive curves under option `−o`.

`−o` (Overlay.) The ordinates for *n* superposed curves appear in the input with each abscissa value. The next argument is *n*.

`−s` Save screen; no new page for this graph.

`−x l` If `l` is present, *x*-axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

`−y l` Similarly for *y*.

`−e` Make automatically determined *x* and *y* scales equal.

`−h` Next argument is fraction of space for height.

`−w` Similarly for width.

`−r` Next argument is fraction of space to move right before plotting.

`−u` Similarly to move up before plotting.

`−t` Transpose horizontal and vertical axes. (Option `−a` now applies to the vertical axis.)

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

plot(1), *grap*(1), *spline*(A)

BUGS

In *graph* segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.
Option `−e` actually makes automatic limits, rather than automatic scaling, equal.

NAME

gre, grep, egrep, fgrep – search a file for a pattern

SYNOPSIS

`gre` [*option* ...] *pattern* [*file* ...]

`grep` [*option* ...] *pattern* [*file* ...]

`egrep` [*option* ...] *pattern* [*file* ...]

`fgrep` [*option* ...] *strings* [*file* ...]

DESCRIPTION

Gre searches the input *files* (standard input default) for lines (with newlines excluded) that match the *pattern*, a regular expression as defined in *re*(3). A file name of − is interpreted as standard input. Normally, each line matching the pattern is 'selected', and each selected line is copied to the standard output. The options are

`−1` Print only the first selected line of each file argument.
`−b` Mark each printed line with its byte position in its file. This is sometimes useful in locating patterns in non-text files.
`−c` Print only a count of matching lines.
`−e` *pattern*
Same as a simple *pattern* argument, but useful when *pattern* begins with a −.
`−E` Simulate *egrep.*
`−f` *file* Read the pattern from *file*; there is no *pattern* argument
`−F` Simulate *fgrep.*
`−G` Simulate *grep.*
`−h` Do not print filename tags (headers) with output lines.
`−i` Ignore alphabetic case distinctions.
`−l` Print the names of files with selected lines; don't print the lines.
`−L` Print the names of files with no selected lines; the converse of `−l`.
`−n` Mark each printed line with its line number counted in its file.
`−s` Produce no output, but return status.
`−v` Reverse: print lines that do not match the pattern.
`−x` Exact match: The pattern is `^(`*pattern*`)$`.

Output lines are tagged by filename when there is more than one input file. (To force this tagging, include `/dev/null` as a filename argument.) If the output line exceeds some internal limit, a warning is given and a small block of text surrounding the match is printed.

Care should be taken when using the shell metacharacters `$*[^|()\` and newline in *pattern*; it is safest to enclose the entire expression in single quotes `'...'`.

Gre supplants three classic programs, which are still available:

Grep handles only *ed*(1)-like regular expressions. It uses `\(\)` instead of `()`.

Egrep handles the same patterns as *gre* except for back-referencing with `\1`, `\2`, ...

Fgrep handles no operators except newline (alternation).

SEE ALSO

re(3), *awk*(1), *sed*(1), *sam*(9.1), *strings*(1)

DIAGNOSTICS

Exit status is 0 if any lines are selected, 1 if none, 2 for syntax errors, inaccessible files (even if matches were found). Warnings will be given for input lines that exceed a (generous) internal limit.

BUGS

Grep, *egrep*, and *fgrep* do not support some options and print block numbers rather than byte numbers for option `−b`.

NAME

hang – start a process in stopped state

SYNOPSIS

`hang` *command*

DESCRIPTION

The given command is executed, but stopped before `main()` is called so that the process can be picked up by a debugger. To ensure that breakpoints may be set, the process does not share its text.

SEE ALSO

pi(9.1), *kill*(1)

NAME

hoc – interactive floating point language

SYNOPSIS

`hoc` [*file* ...]

DESCRIPTION

Hoc interprets a simple language for floating point arithmetic, at about the level of Basic, with C-like syntax and functions.

The named *file*s are read and interpreted in order. If no *file* is given or if *file* is – *hoc* interprets the standard input.

Hoc input consists of *expressions* and *statements*. Expressions are evaluated and their results printed. Statements, typically assignments and function or procedure definitions, produce no output unless they explicitly call *print*.

Variable names have the usual syntax, including _; the name _ by itself contains the value of the last expression evaluated. The variables `E`, `PI`, `PHI`, `GAMMA` and `DEG` are predefined; the last is 59.25..., degrees per radian.

Expressions are formed with these C-like operators, listed by decreasing precedence.

```
^       exponentiation
! - ++ --
* / %
+ -
> >= < <= == !=
&&
||
= += -= *= /= %=
```

Built in functions include `abs`, `atan` (one argument), `cos`, `exp`, `gamma`, `int`, `log`, `log10`, `sin`, and `sqrt`. The function `read(x)` reads a value into the variable `x`; the statement `print` prints a list of expressions that may include string constants such as `"hello\n"`.

Control flow statements are `if-else`, `while`, and `for`, with braces for grouping. Newline ends a statement. Backslash-newline is equivalent to a space.

Functions and procedures are introduced by the words `func` and `proc`; `return` is used to return with a value from a function. Within a function or procedure, arguments are referred to as `$1`, `$2`, etc.; all other variables are global.

EXAMPLES

```
func gcd() {
        temp = abs($1) % abs($2)
        if(temp == 0) return abs($2)
        return gcd($2, temp)
}
for(i=1; i<12; i++) print gcd(i,12)
```

SEE ALSO

bc(1), *dc*(1)

B. W. Kernighan and R. Pike, *The Unix Programming Environment*, Prentice-Hall, 1984

BUGS

Error recovery is imperfect within function and procedure definitions.

The treatment of newlines is not exactly user-friendly.

NAME

ideal – troff preprocessor for drawing pictures

SYNOPSIS

`ideal` [*option* ...] [*file* ...]

DESCRIPTION

Ideal is a constraint-based *troff*(1) preprocessor for typesetting figures in the complex plane. A line beginning `.IS` marks the start of an *ideal* program, `.IE` or `.IF` marks the end. `.IE` leaves the typesetting baseline below the bottom of the picture; `.IF` (flyback) leaves it at the top. The options are

`-T`*dev* Produce instructions for *troff*(1) device *dev*. `-a` is a synonym for `-Taps`; `-t` for `-T202`.

`-tex` Produce output for *tex*(1).

`-p` Produce *plot*(1) instructions. Erases come unbidden at every `.IS`.

`-4` Produce instructions for a Tektronix 4014 and wait at each `.IE` for an input character before erasing and starting the next picture.

`-n` Produce raw *ideal* output, which passes unharmed through *nroff*.

`-v` Print calculated values of variables on standard error.

Ideal programs are built of 'boxes'; boxes look like C functions, in that they are named and delimited by braces. They may include the following kinds of statements, each terminated by a semicolon:

`var` declares one or more complex variables local to the box. Variable names are made up of letters and digits, and start with a letter; do not use any of the following keywords as variable names: `at`, `bdlist`, `boundary`, `box`, `conn`, `construct`, `draw`, `exterior`, `interior`, `left`, `opaque`, `put`, `right`, `spline`, `text`, `to`, `using`, `var`

equation declares relative positions of significant points of the box

`conn` asks for a straight-line path through named points

`pen` asks for a box to be replicated along a line between two points

`left` left-justifies text with respect to a point

`text` centers text with respect to a point

`right` right-justifies text with respect to a point

`spline` draws a spline guided by the named points

`put` asks for an instance of a box to be drawn

`opaque` asks for a box to erase lines already in the picture that are covered by its bounding polygon

`boundary` specifies the bounding polygon for an opaque box

`construct` builds a partial picture on a separate 'sheet of paper'

`draw` adds the contents of the named constructed box to the current picture

Ideal expects all components of a picture to be specified as boxes; instructions to draw the entire picture should comprise a box called `main`. Boxes are remembered across `.IS`/`.IE` boundaries; if you won't need a box again, you can reclaim the space it requires by including the command `...forget` *boxname* on a line between any `.IS`/`.IE` pair after the last use of *boxname*. Box `main` is an exception to this rule: it is always forgotten at `.IE`.

During its first pass, *ideal* solves all the equations to determine the locations of all points it needs to know. These equations must be linear equations in complex variables, although they may include non-linear operators: *ideal* plugs in for as many variables, and does as much function evaluation, as it can before solving the linear equation. It waits until it has absolutely no hope of reducing an equation to a linear equation before complaining. *Ideal* knows about the following functions:

f`[`*z*`,` *w*`]` == *z*+*(w-z)f*, fraction *f* of the way from *z* to *w*

`re(`*z*`)` real part of complex number

`im(`*z*`)` imaginary part of complex number
`conj(`*z*`)` complex conjugate of complex number
`abs(`*z*`)` absolute value (modulus) of complex number
`cis(`*z*`)` the unit vector $cosˆx˜+˜iˆsinˆx$, where *x* = re(*z*) and *x* is measured in degrees (radians if the line `...radians` appeared more recently in the file than the line `...degrees`)
`E(`*x*`)` == cis(360 x) if *x* is measured in degrees
`angle(`*z*`)`
angle of complex number, arctan(im(*z*)/re(*z*))

During the second pass, *ideal* draws the picture.

To draw a circle, include the line `...libfile circle` between the `.IS` and `.IE` lines, and `put` the box named `circle`, giving enough information that the circle can be determined; for instance, give the center and the radius, or give three points through which the circle passes, or give the center and a point on the circle. The circle has center `center`, radius `radius`, and passes through `z1`, `z2`, and `z3`.

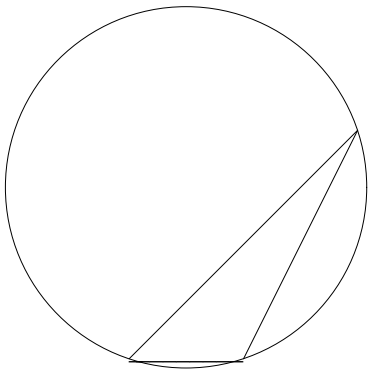
To draw an arc, include the line `...libfile arc` between the `.IS` and `.IE` lines, and `put` the box named `arc`, again giving enough information to determine the arc; for instance, give the center, radius, and starting and ending angles, or give three points on the arc--where to start, where to end, and somewhere in between. The arc has center `center`, radius `radius`, starts at point `start`, passes through point `midway` at angle `midang`, and ends at point `end` at angle `endang`. If no `midway` is specified, the arc is drawn counterclockwise from `start` to `end`.

The picture will be scaled to a default width of four inches and centered in a column of six inches. The default width can be changed by a `...width` command, which includes a number in inches. The default column width can be changed by a `...colwid` command. To defeat *ideal's* notion of the size of the picture, you can include lines of the form `...minx`, `...miny`, `...maxx`, or `...maxy`; these give the various coordinates of the bounding box of the picture in the coordinate system used by the picture.

Ideal supports both C-style comments (between `/*` and `*/` brackets — which nest), and shell-style comments (between # and newline).

EXAMPLES

```
...libfile circle
triangle {
	var z1, z2, z3;
	conn z1 to z2 to z3 to z1;
}
main {
	put T: triangle {
		z1 = 0; z2 = 1; z3 = (2,2);
	}
	put circle {
		z1 = T.z1; z2 = T.z2; z3 = T.z3;
	}
}
```



SEE ALSO

troff(1), *pic*(1), *ped*(9.1), *doctype*(1)
C. J. Van Wyk, 'IDEAL User's Manual', this manual, Volume 2

BUGS

Ideal is relatively unforgiving about syntax errors.
Bounding box computation is naive for arcs and text strings.

NAME

idiff – interactive file comparison

SYNOPSIS

`idiff` [*option*] *file1 file2*

DESCRIPTION

Idiff compares *file1* with *file2* using *diff*, then presents each set of changed lines for selection or processing. *File2* may be a directory; in that case, the basename of *file1* is appended.

For each group, legal responses are

`<` to retain the 'from' lines

`>` to retain the 'to' lines

`e` to edit both sets of lines

`d` to delete both sets

`1` to retain the rest of the 'from' file

`2` to retain the rest of the 'to' file

`!` to invoke a shell command

Lines that compare equal are copied verbatim from *file1*. Lines produced by this process, including the lines written from within the editor, are written to file *idiff.out*. Comparison may be affected by the *diff*(1) options

`-b` Ignore trailing blanks (spaces and tabs) and treat other strings of blanks as if they were a single space.

`-B` Ignore all blanks.

FILES

```
idiff.out
idiff.*
/tmp/idiff.*
```

SEE ALSO

diff(1)

BUGS

There is no way to revisit a choice.

NAME

imscan – scan greyscale images

SYNOPSIS

`imscan` [`-s`*N*] [`-l`*N*] *file*

DESCRIPTION

Imscan digitizes an image with an Imagitex grey-scale scanner and places the result in the named file in the form of *picfile*(5). The options are

`-s`*N* Set a scale factor $1 \le N \le 9$, default 4. With scale factor *N* the image is subsampled: only 1 out of every $N \times N$ pixels is stored. A larger scale factor, therefore, produces a smaller image.

`-l`*N* Use lens focal length *N*, where *N* is either 5 or 8 (default). The 8-inch lens scans images at 480 dots per inch. The 5-inch lens scans at 754 dots per inch.

SEE ALSO

cscan(1), *pico*(1), *qsnap*(1), *mugs* in *face*(9.7), *picfile*(5)

BUGS

It is hard to get more than 2000 pixels per scanline reliably. For large originals, higher scale factors work better than smaller ones.

NAME

join – relational database operator

SYNOPSIS

`join` [*options*] *file1 file2*

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If one of the file names is –, the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

Input fields are normally separated spaces or tabs; output fields by space. In this case, multiple separators count as one, and leading separators are discarded.

The following options are recognized, with Posix syntax.

`–a` *n* In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.

`–v` *n* Like `–a`, omitting output for paired lines.

`–e` *s* Replace empty output fields by string *s*.

`–1` *m*
`–2` *m* Join on the *m*th field of *file1* or *file2.*

`–j`*n m* Archaic equivalent for `–`*n* `m`.

`–o`*fields*
Each output line comprises the designated fields. The comma-separated field designators are either `0`, meaning the join field, or have the form *n.m*, where *n* is a file number and *m* is a field number. Archaic usage allows separate arguments for field designators.

`–t`*c* Use character *c* as the only separator (tab character) on input and output. Every appearance of *c* in a line is significant.

EXAMPLES

```
sort /etc/password | join -t: -a 1 -e "" - bdays
```
Add birthdays to password information, leaving unknown birthdays empty. The layout of `/etc/passwd` is given in *passwd*(5); `bdays` contains sorted lines like `ken:Feb 4`.

```
tr : ' ' </etc/passwd | sort -k 3 3 >temp
join -1 3 -2 3 -o 1.1,2.1 temp temp | awk '$1 < $2'
```
Print all pairs of users with identical userids.

SEE ALSO

sort(1), *comm*(1), *awk*(1)

BUGS

With default field separation, the collating sequence is that of `sort -b -k`*y*`,`*y*, with `-t`, the sequence is that of `sort -t`*x* `-k`*y*`,`*y*.
One of the files must be randomly accessible.

NAME

kill – terminate a process with extreme prejudice

SYNOPSIS

`kill` [−*sig*] *processid* ...
`kill −l`

DESCRIPTION

Kill sends the `SIGTERM` signal to the specified processes. If a signal name or number preceded by − is given as first argument, that signal is sent instead; see *signal*(2). The signal names are listed by `kill −l`, and are as given in `<signal.h>`.

The terminate signal will kill processes that do not catch the signal. The `SIGKILL` signal is a sure kill, since it cannot be caught. By convention, if process number 0 is specified, all members in the process group (usually processes of the current login or current *mux*(9.1) layer) are signaled. Killed processes must belong to the current user unless that is super-user.

To shut the system down and bring it up single user the super-user may send the initialization process a terminate signal by `kill 1`; see *init*(8). To force *init* to close and open terminals according to what is currently in `/etc/ttys` use `kill −SIGHUP 1`.

The process number of an asynchronous process started with `&` is reported by the shell and by *ps*(1).

EXAMPLES

`kill 7151`
: Kill process 7151 gently; the process can catch the signal.

`kill −SIGKILL 7151`
: Kill peremptorily; this signal cannot be caught.

`kill 0`
: Kill all the background processes in this process group.

SEE ALSO

ps(1), *signal*(2), *signal*(2), *init*(8)

NAME

lab – label maker

SYNOPSIS

`lab` [`-m`] [*file* ...]

DESCRIPTION

Lab causes the *files* to be queued for printing as mailing labels. If no files are named, the standard input is read. The option `-m` causes notification via *mail*(1) to be sent when the job completes.

The last line of each label is identified by ending it with one or more spaces and a hyphen. The hyphen must be the last character in the line; no white space may follow it.

FILES

`/usr/spool/lab/*`	spool area
`/usr/lib/lab`	printer daemon

SEE ALSO

pr(1)

BUGS

Queued jobs print in directory (seemingly random) order.

NAME

altran, cospan, esterel, icon, lisp, macsyma, maple, ops5, pascal, ratfor, S, smp, sno, spitbol, struct, twig − languages

SYNOPSIS

`altran` [*option* ...] *file* ...
`cospan` [*option* ...] file
`esterel`
`iconc`
`icont`
`lisp`
`/usr/lbin/macsyma`
`maple`
`/usr/lbin/ops5`
`ratfor` [*option* ...] *file* ...
`S`
`smp`
`sno` [*file* ...]
`spitbol` [*option* ...] *file* ...
`struct` [*option* ...] *file* ...
`twig` [−w*xx*] *file*

DESCRIPTION

Altran, a language for rational algebra, is described in W. S. Brown, *ALTRAN User's Manual*. For more information execute `man altran`.

Cospan, a system which analyzes concurrent programs written in the data-flow language S/R for properties defined by automata, is described in Z. Har'El and R. P. Kurshan, *COSPAN User's Guide*, 1121-871009-21TM, AT&T Bell Laboratories, 1987. For more information, execute `man cospan`.

Esterel compiles single-process implementations of programs expressed in terms of asynchronously cooperating automata. For more information, execute `man esterel`.

Icon, a general-purpose language with stream-based coroutines is described in R. E. Griswold, *The Icon Programming Language*, Prentice-Hall, 1983. For more information, execute `man icont`.

Lisp, the symbol manipulation language, is described in J. K. Foderara, 'The Franz Lisp Manual', in Unix Programmer's Manual, Seventh Edition, Virtual VAX-11 Version, 1980, Volume 2C (Berkeley)

Macsyma, another symbolic algebra language, is described in *Macsyma Reference Manual*, Laboratory for Computer Science, MIT, 1977. It breaks if the environment contains shell functions.

Maple, a third symbolic algebra language, is described in K. O. Geddes, G. H. Gonnet, and B. W. Char, *MAPLE User's Manual, Third Edition*, Research Report CS-83-41 Dept. of Computer Science, University of Waterloo, 1983. For more information execute `man maple`.

Ops5 is a production-system interpreter described in C. L. Forgy *OPS5 User's Manual*, Department of Computer Science, Carnegie-Mellon University, July, 1981. For more information execute `man ops5`.

Pascal is an interpreter and *pc* is a compiler for the well known language. For more information, type `man pascal pc pxp`.

Ratfor accepts Fortran extended with C-like control constructs and compiles into Fortran. For more information execute `man ratfor`. For a full description see B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

S, a system for data analysis and graphics, is described in R. A. Becker, J. M. Chambers, and A. R. Wilks, *The New S Language*, Wadsworth, 1988.

Smp, a fourth symbolic algebra language, is described in C. Cole and S. Wolfram, *SMP Handbook*, California Institute of Technology, 1981.

Sno is a dialect of Snobol 3. For more information execute `man sno`.

Spitbol is a dialect of Snobol 4. For more information execute `man spitbol`. The full story is in R. B. K. Dewar, A. P. McCann, R. E. Goldberg, and S. G. Duff, *Macro SPITBOL Program Reference Manual*, and R. E. Griswold, J. F. Poage, and I. P. Polonsky, *The SNOBOL4 Programming Language* Academic Press, 1968.

Struct, an inverse of *ratfor*, deduces rational control structure from pure Fortran. For more information execute `man struct`.

Twig is a language for tree manipulation, useful for code generation and other applications. For more information execute `man twig`. See also S. W. K. Tjiang, *Twig Reference Manual*, AT&T Bell Laboratories Computing Science Technical Report 120.

BUGS

These language processors are available on a haphazard collection of machines. Many are maintained in the same spirit.

NAME

lcomp, lprint – line-by-line profiler

SYNOPSIS

`lcomp` [*option* ...] *file* ...

`lprint` [*option*] [*file* ...]

DESCRIPTION

Lcomp is used in place of *cc*(1) or *f77*(1) to insert instruction-counting code into programs. It shares with those commands options whose initial letters are taken from the string `cwpDUIRdlNnz`, and accepts files whose names end in `.c`, `.f`, `.s`, or `.o`. From each source file it derives a `.o` file and a `.sL` file which *lprint* uses to correlate source lines with basic blocks.

Option `-C` declares that `.c` files (and `.o` files, if no source files are named) are C++ files. If the `-c` option is not present *lcomp* creates `a.out`. Each time `a.out` is run statistics are added to a profiling file `prof.out`.

Lprint produces on the standard output a listing (in the style of *pr*(1)) of the programs compiled by *lcomp*. Without arguments or files, each line of the listing is preceded by the number of times it was executed, as determined from the data in `prof.out`. *Lprint* interprets the following options.

`-a` Detailed listing of every machine instruction and how often it was executed.

`-b` How often each basic block was executed.

`-c` Compress the `prof.out` file, which otherwise grows with every execution of `a.out`.

`-f` Print summary information by function: instruction executions, number of invocations, source instructions, and number of instructions never executed.

`-i` Before each line of source print the number of machine instructions executed.

`-p` Before each line of source print the number of times the first basic block in that line was executed.

`-s` Summarize the counts by source file: instruction executions, source instructions, instructions never executed, basic block executions, total number of source basic blocks, and how many were never executed.

If any file names are given, the options `abip` apply only to them. If no options are given, `-p` is assumed. Any combination of options is allowed.

FILES

`prof.out`	counts
`*.sL`	for correlating with source
`/usr/lib/bb`	for finding basic blocks and inserting counting code
`/usr/lib/nexit.o`	for printing counts when `a.out` exits

SEE ALSO

cc(1), *f77*(1), *prof*(1)

BUGS

A line in the source file may be in zero, one, or more basic blocks; the count given in the listing corresponds to some particular choice of the basic block to associate with the line.
Processing the output of *yacc*(1) without removing `#line` directives will produce unsatisfactory results.
Option `-C` masks an option of *cc*(1).

NAME

ld – link editor or loader

SYNOPSIS

`ld` [*option ...*] *file ...*

DESCRIPTION

Ld combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object *files* are given, and *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the `-r` option must be given to preserve the relocation bits.) The output of *ld* is left on `a.out`. This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine (unless the `-e` option is specified).

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by *ranlib* (see *ar*(1)), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important; see *lorder*(1). The first member of a library should be a file named `__.SYMDEF`, which is understood to be a dictionary for the library as produced by *ranlib*; the dictionary is searched iteratively to satisfy as many references as possible.

The symbols `_etext`, `_edata`, and `_end` (`etext`, `edata`, and `end` in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data, respectively. It is erroneous to define these symbols.

Ld understands several options. Except for `-l` and `-o`, they should appear before the file names.

`-A` Load incrementally, so that the resulting object code may be read into an already executing program. The next argument names an object file whose symbol table will be added to. Only newly linked material will be entered into the text and data portions of `a.out`, but the new symbol table will reflect every symbol defined before and after the incremental load. `-A` must not follow any object file names.

`-D` Take the next argument as a hexadecimal number and pad the data segment with zeros to the indicated length.

`-d` Force definition of common storage even if the `-r` flag is present.

`-e` The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default.

`-l`*x* This option is an abbreviation for the library name `/lib/lib`*x*`.a`, where *x* is a string. If that does not exist, *ld* tries `/usr/lib/lib`*x*`.a` A library is searched when its name is encountered, so the placement of the option is significant.

`-M` produce a primitive load map, listing the names of the files which will be loaded.

`-N` Do not make the text portion read-only or sharable. (Use ‘magic number’ 0407.)

`-n` Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. (Use magic number 0410 and move the data segment to a 1024 byte boundary.)

`-o` The *name* argument after `-o` is used as the name of the *ld* output file, instead of `a.out`.

`-r` Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final fixing of ‘common’ symbols (uninitialized C variables or Fortran common variables), and suppresses ‘undefined symbol’ diagnostics.

`-s` Strip the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debuggers). This information can also be removed by *strip*(1).

`-S` Partially strip; remove all symbols that were not in the source.

`-T` The next argument is a hexadecimal number which sets the text segment origin. With option `-A` this origin must be a multiple of 1024. The default is 0, or `_end` with `-A`.

`-t` (trace) Print the name of each file as it is processed.

`-u` Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

`-X` Save local symbols except for those whose names begin with `L`. This option is used by *cc*(1) to discard internally-generated labels while retaining symbols local to routines.

`-x` Do not preserve local symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

`-y`*sym* Indicate each file in which *sym* appears, its type and whether the file defines or references it. Many such options may be given to trace many symbols. (It is usually necessary to begin *sym* with an underscore _, as external C, Fortran, and Pascal variables begin with underscores.)

`-z` Arrange for the process to be loaded on demand from the resulting executable file (magic number 413) rather than preloaded. This (default) output format has a 1024-byte header followed by a text and data segment each of which have size a multiple of 1024 bytes (being padded out with zeros if necessary). The first few BSS segment symbols may appear in the data segment to avoid wasting space at the end of that segment.

FILES

`/lib/lib*.a`	libraries
`/usr/lib/lib*.a`	more libraries
`a.out`	output file

SEE ALSO

as(1), *ar*(1), *cc*(1), *f77*(1), *size*(1), *nm*(1), *lorder*(1), *a.out*(5)

BUGS

There is no way to force data to be page aligned.

NAME

lex – generator of lexical analysis programs

SYNOPSIS

`lex` [`-tvfn`] [*file* ...]

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text. The input *files* (standard input default) contain regular expressions to be searched for, and actions written in C to be executed when expressions are found.

A C source program, `lex.yy.c` is generated, to be compiled thus:

```
cc lex.yy.c -ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized.

The options have the following meanings.

`-t` Place the result on the standard output instead of in file `lex.yy.c`.

`-v` Print a one-line summary of statistics of the generated analyzer.

`-n` Opposite of `-v`; `-n` is default.

`-f` 'Faster' compilation: don't bother to pack the resulting tables; limited to small programs.

EXAMPLES

This program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
%%
[A-Z] putchar(yytext[0]+'a'-'A');
[ ]+$
[ ]+  putchar(' ');
```

FILES

`lex.yy.c`

SEE ALSO

yacc(1), *sed*(1)

M. E. Lesk and E. Schmidt, 'LEX—Lexical Analyzer Generator', this manual, Volume 2

NAME

lint, cyntax, cem − C program verifiers

SYNOPSIS

`lint` [`-abchnpuvx`] [*option ...*] *file ...*

`cyntax` [*option ...*] *file ...*

`/usr/lib/cyntax/cem` [*option ...*] *file ...*

DESCRIPTION

Lint checks, more thoroughly than *cc*(1), the syntactic validity and semantic consistency of one or more C program *files*. It is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to *lint*; these libraries are referred to by a conventional name, such as `-lm`, in the style of *ld*(1).

Any number of the option letters in the following list may be used. The `-D`, `-U`, and `-I` options of *cc*(1) are also recognized as separate arguments.

`p` Attempt to check portability to some other dialects of C.
`h` Apply heuristics to intuit bugs, improve style, and reduce waste.
`b` Report *break* statements that cannot be reached.
`v` Suppress complaints about unused arguments in functions.
`x` Report variables referred to by extern declarations, but never used.
`a` Report assignments of long values to int variables.
`c` Complain about casts which have questionable portability.
`u` Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running *lint* on a subset of files out of a larger program).
`n` Do not check compatibility against the standard library.

Certain conventional comments in the C source will change the behavior of *lint*:

`/*NOTREACHED*/`
at appropriate points. Stop comments about unreachable code.

`/*VARARGS`*n*`*/`
Suppress the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

`/*PRINTFLIKE`*n*`*/`
The data types of the first *n* arguments are checked as usual. The remaining arguments are checked against the *n*th argument, which is interpreted as a *printf*(3) format string.

`/*SCANFLIKE`*n*`*/`
Similarly for *scanf*(3).

`/*NOSTRICT*/`
Shut off strict type checking in the next expression.

`/*ARGSUSED*/`
Turn on the `-v` option for the next function.

`/*LINTLIBRARY*/`
at the beginning of a file. Shut off complaints about unused functions in this file.

Cyntax makes checks similar to those of *lint*, more stringent about syntax, less observant of portability issues. It keeps type information gleaned from source files, whose names end with `.c`, in 'object files' with corresponding names ending in `.O`. If all goes well it will cross check among all the `.c` and `.O` input *files*.

Options `-D`, `-I`, `-U`, `-o`, `-l` are as in *cc*(1). Options `-O`, `-P`, `-g`, `-p`, and `-s` are ignored. Other options are:

`-c` Suppress cross checking.
`-d` Passed to *cem*.

`-h` Base object files on the basename of the source file rather than the full pathname.
`-n` Do not check compatibility against the standard library.
`-G` Change default directory of include files to `/usr/2include`, pass `-lg` to *cem* instead of `-lc`.
`-j` Change default directory of include files to `/usr/jerq/include`, pass `-lj` to *cem* instead of `-lc`.
`-k` Change default directory of include files to `/usr/ninclude`, pass `-lk` to *cem* instead of `-lc`.
`-w` Enable pedantic warning diagnostics.
`-m` equivalent to `-j -DMUX`.
`-v` Report what *cyntax* is actually doing.
`-V`*func*`:`*n*
Declare function *func* to have a variable number of arguments, the first *n* of which are to be type checked.

Cem (cemantics), the cross-checker, is normally invoked only by *cyntax*. It also has options, some of which *cyntax* can't be coerced into providing. Besides `-o` and `-l`, they are:

`-m` Don't believe file modification times. These are normally used to avert redundant type checking.
`-d` Debug: print ascii version of `.O` files on standard output.
`-p` Be pedantic about type checking.
`-t` Unconditionally include file modification times in diagnostics.
`-v` Use a verbose format for type names.

FILES

`/usr/lib/lint/lint[12]`	programs
`/usr/lib/lint/llib-lc`	declarations for standard functions
`/usr/lib/lint/llib-lport`	declarations for portable functions
`/usr/include/*`	
`/usr/lib/cyntax/ccom`	cyntax proper
`/usr/lib/cyntax/libc`	type library
`/usr/lib/cyntax/libj`	

SEE ALSO

cc(1), *cin*(1)

BUGS

Lint's understanding of the type system of C is outmoded: its handling of `void` and ®is simply wrong. The unnatural default setting of *lint* option `-b` is intended to hide the ugliness of C code produced by *yacc*(1) and *lex*(1).

NAME

load – load statistics

SYNOPSIS

`load` [*interval* [*count* [*sysfile* [*corefile*]]]]

DESCRIPTION

Load reports the number of processes ready to run averaged over the preceding 1, 5, and 15 minutes.

The optional *interval* argument causes a report once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports.

The optional arguments *sysfile* and *corefile* cause the named files to be consulted instead of the defaults, `/unix` and `/dev/kmem`.

FILES

`/dev/kmem`
`/unix`

SEE ALSO

vismon(9.1), *ps*(1), *vmstat*(8)

NAME

look – find lines in a sorted list

SYNOPSIS

`look` [`-dfnixt`*c*] [*string*] [*file*]

DESCRIPTION

Look consults a sorted *file* and prints all lines that begin with *string*. It uses binary search.

The following options are recognized. Options `dfnt`*c* affect comparisons as in *sort*(1).

`-i` Interactive. There is no *string* argument; instead *look* takes lines from the standard input as strings to be looked up.

`-x` Exact. Print only lines of the file whose key matches *string* exactly.

`-d` 'Directory' order: only letters, digits, tabs and blanks participate in comparisons.

`-f` Fold. Upper case letters compare equal to lower case.

`-n` Numeric comparison with initial string of digits, optional minus sign, and optional decimal point.

`-t`*c* 'Tab character' *c* terminates the sort key in the *file*.

If no *file* is specified, `/usr/dict/words` is assumed, with collating sequence `df`.

FILES

`/usr/dict/words`

SEE ALSO

sort(1), *gre*(1), *dict*(7)

DIAGNOSTICS

Look returns exit status 0 if *string* is found, 1 if not found, 2 for error.

NAME

lorder – find ordering relation for an object library

SYNOPSIS

`lorder` *file* ...

DESCRIPTION

The input is one or more object or library archive (see *ar*(1)) *files*. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass sequential access by *ld*(1).

EXAMPLES

`ar cr libnew.a ‘lorder *.o | tsort‘`

Build a new library from existing `.o` files.

FILES

`/tmp/*symref`
`/tmp/*symdef`

SEE ALSO

ar(1), *tsort*(1), *ld*(1)

BUGS

The names of object files, in and out of libraries, must end with `.o`; nonsense results otherwise.

NAME

lp − printer output

SYNOPSIS

`lp` [*option* ...] [*file* ...]

DESCRIPTION

Lp is a generalized output printing service. It can be used to queue files for printing, check a queue, or kill jobs in a queue. The options are:

`−d` *dest* Select the destination printer. If *dest* is `?`, list the currently available printers. In the absence of `−d`, the destination is taken from the environment variable `LPDEST`, or finally from the file `/usr/spool/lp/defdevice`. Destination `stdout` is the standard output.

`−p` *proc* The given preprocessor is invoked. The default preprocessor is `generic`, which tries to do the right thing for regular text, *troff*(1) output, or *bitfile*(9.5) bitmaps. If no preprocessing is desired (e.g. an ImPress file from `dviimp` is to be printed) `noproc` may be specified.

`−q` Print the queue for the given destination. For some devices, include printer status.

`−k` Kill the job(s) given as subsequent arguments instead of file names for the given destination.

The remaining options may be used to affect the output at a given device. These options may not be applicable to all devices.

`−c` *n* Print *n* copies.

`−f` *font* Set the font (default `CW.11`).

`−H` Suppress printing of header page.

`−i` *n* Select paper input tray *n.*

`−l` *n* Set the number of lines per page to *n.*

`−L` Print pages in landscape mode (i.e. turned 90 degrees).

`−m` *n* Set magnification to *n.*

`−n` *n* Print *n* logical pages per physical page.

`−o` *list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *n*−*m* means pages *n* through *m*; a range −*n* means from the beginning to page *n*; a range *n*− means from page *n* to the end.

`−r` Reverse the order of page printing (currently not functional).

`−x` *n* Set the horizontal offset of the print image, measured in inches.

`−y` *n* Set the vertical offset of the print image, measured in inches.

EXAMPLES

```
eqn paper | troff −ms | lp
```
Typeset and print a paper that contains equations.

```
pr −l100 file | lp −l100 −fCW.8
```
Print a file in a small font at 100 lines per page.

```
lp −dstdout −H <bitfile >postfile
```
Convert a bitmap to postscript form. Use *mpictures*(6) macros to insert the output into a *troff* document.

FILES

`/usr/spool/lp/defdevice`	default printer name
`/usr/spool/lp/devices`	printer list with interface specification
`/usr/spool/lp/process`	directory of preprocessors
`/usr/spool/lp/prob/*`	where printer jobs go when things go awry

SEE ALSO

pr(1), *blitblt*(9.1), *plot*(1), *font*(6), *postio*(8), *postscript*(8)

BUGS

Not all options work with all output devices.
Any user can kill any job.

NAME

ls, lc – list contents of directory

SYNOPSIS

`ls` [`-acdfilrstuFLR`] *name ...*

`lc` [*options*] *name ...*

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted, but file arguments appear before directories and their contents.

Lc is the same as *ls*, but prints the list in multiple columns.

There are an unbelievable number of options:

`-l` List in long format, giving mode (see below), number of links, owner, group, size in bytes, and time of last modification for each file. Symbolic links are identified by a link count marked `L`; the link count is that of the ultimate file. If the file is a special file the size field will instead contain the major and minor device numbers.

`-d` If argument is a directory, list its name, not its contents.

`-t` Sort by time modified (latest first) instead of by name, as is normal.

`-L` Under `-l` for each symbolic link give the immediate, not the ultimate, link count and append the name pointed to.

`-a` List all entries; usually `.` and `..` are suppressed.

`-c` Under `-t` sort by time of inode change; under `-l` print time of inode change.

`-f` Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off `-l`, `-t`, `-s`, and `-r`, and turns on `-a`; the order is the order in which entries appear in the directory.

`-F` Mark directories with a trailing `/` and executable files with a trailing `*`

`-i` Print i-number in first column of the report for each file listed.

`-r` Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

`-R` recursively list subdirectories encountered.

`-s` Give size in Kbytes for each entry.

`-u` Under `-t` sort by time of last access; under `-l` print time of last access.

The mode printed under the `-l` option contains 11 characters which are interpreted as follows: the first character is

`d` if the entry is a directory;
`b` if the entry is a block-type special file;
`c` if the entry is a character-type special file;
`l` if the entry is a symbolic link and option `-L` is in effect;
`-` if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, ‘execute’ permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

`r` if the file is readable;

`w` if the file is writable;
`x` if the file is executable;
`-` if the indicated permission is not granted.

The group-execute permission character is given as `s` if the file has set-group-ID mode; likewise the user-execute permission character is given as `s` if the file has set-user-ID mode.

The last character of the mode (normally a blank) indicates the type of concurrency control:

`e` if the file is set for exclusive access (1 writer or *n* readers);
`y` if the file is set for synchronized access (1 writer and *n* readers);

FILES

`/etc/passwd`
`/etc/group`

SEE ALSO

stat(2)

BUGS

Option `-s` counts unwritten holes as if they were real data.

NAME

m4 – macro processor

SYNOPSIS

`m4` [*option* ...] [*file* ...]

DESCRIPTION

M4 is a macro processor intended as a front end for C and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is –, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

`-e` Operate interactively. Interrupts are ignored and the output is unbuffered. Using this mode requires a special state of mind.

`-s` Enable line sync output for the C preprocessor, (`#line ...`)

`-B`*int* Change the size of the push-back and argument collection buffers from the default of 4,096.

`-H`*int* Change the size of the symbol table hash array from the default of 199. The size should be prime.

`-S`*int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

`-T`*int* Change the size of the token buffer from the default of 512 bytes.

The preceding options must appear before any file names or `-D` or `-U` options.

`-D`*name*[=*val*]
Defines *name* to *val* or to null if *val* is missing.

`-U`*name*
undefines *name*.

Macro calls have the form:

```
name(arg1,arg2, ..., argn)
```

The `(` must immediately follow the name of the macro. If a defined macro name is not followed by a `(`, it is deemed to have no arguments. Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

The value of a macro is obtained by replacing each occurrence of `$`*n* in the replacement text, where *n* is a digit, with the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; `$#` is replaced by the number of arguments; `$*` is replaced by a list of all the arguments separated by commas; `$@` is like `$*`, but each argument is quoted (with the current quotes).

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

`define` the second argument is installed as the replacement text of the macro whose name is the first argument.

`undefine` Remove the definition of the macro named in the argument.

`defn` Return the quoted definition of the argument(s); useful for renaming macros, especially built-ins.

`pushdef` Like *define*, but save any previous definition.

`popdef` Remove current definition of the argument(s), exposing the previous one if any.

`ifdef` If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word `unix` is predefined on UNIX versions of *m4*.

`shift` Return all but the first argument. The other arguments pushed back with commas in between and quoted to nullify the effect of the extra scan.

`changequote`
Change quote symbols to the first and second arguments. The symbols may be up to five characters long. `Changequote` without arguments restores the original values (i.e., `` ` ' ``).

`changecom` Change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.

`divert` *m4* Switch output to one of 10 streams, numbered 0-9 designated by the argument. The final output is the concatenation of the streams in numerical order; stream 0 is the current initially. Output to a stream other than 0 through 9 is discarded.

`undivert` Cause immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Once undiverted, the diverted text is no longer contained in that diversion.

`divnum` Return the name of the current output stream.

`dnl` reads and discards characters up to and including the next new-line.

`ifelse` If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if that is not present, null.

`incr` Return the value of the argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

`decr` Return the value of the argument decremented by 1.

`eval` Evaluate the argument as an arithmetic expression, using 32-bit arithmetic. C-like operators include `+-*/%`, bitwise `&|^~`; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

`len` Returns the number of characters in the argument.

`index` Return the position in the first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.

`substr` Return a substring of the first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

`translit` Transliterate the characters in the first argument from the set given by the second argument to the set given by the third, deleting characters that lack a correspondent in the

third set. There is no character-range notation.

`include` Return the contents of the file named in the argument.

`sinclude` Same, but give no diagnostic if the file is inaccessible.

`syscmd` Execute the UNIX command given in the first argument. No value is returned.

`sysval` The return code from the last call to *syscmd*.

`maketemp` Fill in a string of `X` characters in the argument with the current process id.

`m4exit` Exit immediately from *m4*. Argument 1, if given, is the exit code; the default is 0.

`m4wrap` Push the argument back at the end of the input. Example: `m4wrap(‘cleanup()’)`

`errprint` Prints the argument on the standard error file.

`dumpdef` Print current names and definitions, for the named items, or for all if no arguments are given.

`traceon` If there are no arguments, turn on tracing for all macros (including built-ins). Otherwise, turn on tracing for named macros.

`traceoff` Turn off trace globally and for any macros specified. Macros specifically traced by `traceon` can be untraced only by specific calls to `traceoff`.

EXAMPLES

```
define(fib,‘ifelse(define(‘n’,eval($1))n,0,1,n,1,1,dnl()
‘eval(fib(n-1)+fib($1-2))’)’)dnl()
fib(2*3)
```

Recursively evaluate a Fibonacci number. The inner `define` avoids some reevaluations.

NAME

mail – send or receive mail

SYNOPSIS

`mail` [`-mpren`] [`-f` *file*]

`mail` [`-#`] *person ...*

`mail`

`/usr/lib/upas/gone.fishing` [*mesg*]

DESCRIPTION

Printing Mail

When *persons* are not named, *mail* displays your incoming computer mail. The options are:

`-r`	Print mail in first-in, first-out order.
`-p`	Print all the mail messages without prompting for commands.
`-m`	Use a manual style of interface, i.e., print no messages unless directed to.
`-f` *file*	Use *file*, e.g. `mbox`, as if it were the mailbox.
`-e`	Check silently if there is anything in the mailbox; return zero (true) if so, non-zero otherwise.
`-n`	Announce mail to the control terminal when it arrives. Do not print mail now.

Mail prints a user's mail, message by message, prompting between messages. After printing a prompt *mail* reads a line from the standard input to direct disposition of the message. Commands, as in *ed*(1), are of the form '[*range*] *command* [*arguments*]'. The command is applied to each message in the (optional) range addressed by message number and/or regular expressions in the style of *regexp*(3). A regular expression in slashes searches among header (postmark) lines; an expression in backslashes searches on message content.

address	to indicate a single message header
address`,`*address*	to indicate a range of contiguous message headers
`g/`*expression*`/`	to indicate all message headers matching the regular *expression.*

The commands are:

`b`	Print the headers for the next ten messages.
`d`	Mark message for deletion on exiting mail.
`h`	Print the disposition, size in characters, and header line of the message.
`m` *person ...*	Mail the message to the named *persons.*
`M` *person ...*	Same as `m` except that lines typed on the terminal (terminated by `EOT or .`) are prepended to the message.
`p`	Print message. An interrupt stops the printing.
`r`	Reply to the sender of the message.
`R`	Like `r` but with the message appended to the reply.
`s` *file*	(Save) Append the message to the named *file* (`mbox` default, in `HOME` directory if known, see *environ*(5)).
`q`	Put undeleted mail back in the mailbox and stop.
EOT (control-D)	Same as `q`.
`w` *file*	Same as `s` with the mail header line(s) stripped.
`u`	Remove mark for deletion.
`x`	Exit, without changing the mailbox file.
`?`	Print a command summary.
`|`*command*	Run the *command* with the message as standard input.
`!`*command*	Escape to the shell to do *command.*

= Print the number of the current message.

Sending Mail

When *persons* are named, *mail* takes the standard input up to an end-of-file, or (if input is from a terminal) to a line consisting of a single `.` and adds it to each *person's* mailbox. The message is automatically postmarked with the sender's name and date. Lines that look like postmarks are prefixed with >.

Person is a login name on the local system or a network name for a remote system; see *mail*(6).

Option `-#` does not send mail, but reports instead how the mail would be sent: the sender, the next machine to handle the mail, and the recipient's address relative to that machine. The report reflects address translation; see *mail*(6) and *upas*(8).

Sh(1) and *vismon*(9.1) have mechanisms for timely notification of incoming mail.

Mailboxes

Each user owns a mailbox for incoming mail, normally `/usr/spool/mail/`*person*. *Mail* creates mailboxes as necessary, and never removes them. Mailboxes are created readable but not writable by others. For more privacy, a mailbox's owner may make it unreadable; see *chmod*(2).

If a mailbox contains the sole line

`Forward to` *name*,

mail for that mailbox is sent instead to *name*. *Name* may be a list of names. If the mailbox contains

`Pipe to` *command*

the mail is sent to the standard input of *command* instead of being appended to the mailbox. The command is run with the userid and groupid of the mailbox's owner. The command is sent (see *signal*(2)) after two minutes. (On System V machines, the set userid bit must be set.)

Mail checks centralized forwarding lists before looking in mailboxes. If you have accounts on many machines, but wish to receive mail on only one, it is usually easier to register in forwarding lists than to install `Forward to` in many mailboxes; see *upas*(8).

To use *mail* as an answering machine while you are away, replace the contents of your mailbox with a single line like

`Pipe to /usr/lib/upas/gone.fishing /usr/`*you*`/`*mesg*

The *mesg* file will be sent (just once) to everyone who sends you mail; arriving messages will be collected in `gone.mail` in your home directory. If you do not name a *mesg* file, `/usr/lib/upas/gone.msg` will be used by default.

FILES

`/usr/spool/mail/mail.log`	mail log file
`/usr/spool/mail/*`	mailboxes
`/etc/passwd`	to identify sender and locate persons
`$HOME/mbox`	saved mail
`$HOME/dead.letter`	unmailable text
`/usr/lib/upas/edmail`	the program for editing mail
`/usr/lib/upas/send`	the program for sending mail
`/bin/rmail`	a link to `/bin/mail`, used to receive remote mail
`/usr/lib/upas/gone.msg`	
`$HOME/gone.mail`	
`$HOME/gone.addrs`	list of senders answered by *gone.fishing*

SEE ALSO

write(1), *vismon*(9.1), *uucp*(1), *verify*(1), *mail*(6), *upas*(8), *smtp*(8)

BUGS

Long headers are truncated for header search.
Backslash quoting is impossible in content regular expressions.

NAME

make – maintain collections of programs

SYNOPSIS

`make` [`-f` *makefile*] [*option* ...] [*name* ...]

DESCRIPTION

Make executes recipes in *makefile* to update the target *names* (usually programs). If no target is specified, the targets of the first rule in *makefile* are updated. If no `-f` option is present, `makefile` and `Makefile` are tried in order. If *makefile* is `-`, the standard input is taken. More than one `-f` option may appear.

Make updates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist. The prerequisites are updated before the target.

The makefile comprises a sequence of rules and macro definitions. The first line of a rule is a blank-separated list of targets, then a single or double colon, then a list of prerequisite files terminated by semicolon or newline. Text following a semicolon, and all following lines that begin with a tab, are shell commands: the recipe for updating the target.

If a name appears as target in more than one single-colon rule, it depends on all of the prerequisites of those rules, but only one recipe may be specified among the rules. A target in a double-colon rule is updated by the following recipe only if it is out of date with respect to the prerequisites of that rule.

Two special forms of name are recognized. A name like *a*(*b*) means the file named *b* stored in the archive named *a*. A name like *a*((*b*)) means the file stored in archive *a* and containing the entry point *b*.

Sharp and newline surround comments.

In this makefile `pgm` depends on two files `a.o` and `b.o`, and they in turn depend on `.c` files and a common file `ab.h`:

```
pgm: a.o b.o
	cc a.o b.o -lplot -o pgm

a.o: ab.h a.c
	cc -c a.c

b.o: ab.h b.c
	cc -c b.c
```

Makefile lines of the form

string1 = *string2*

are macro definitions. Subsequent appearances of `$(`*string1*`)` are replaced by *string2*. If *string1* is a single character, the parentheses are optional; `$$` is replaced by `$`. Each entry in the environment (see *sh*(1)) of the *make* command is taken as a macro definition, as are command arguments with embedded equal signs.

Lines of the form *string1* `:=` *string2* occurring in a recipe are assignments: macro definitions that are made in the course of executing the recipe.

A target containing a single `%` introduces a pattern rule, which controls the making of names that do not occur explicitly as targets. The `%` matches an arbitrary string called the stem: *A*`%`*B* matches any string that begins with *A* and ends with *B*. A `%` in a prerequisite name stands for the stem; and the special macro `$%` stands for the stem in the recipe. A name that has no explicit recipe is matched against the target of each pattern rule. The first pattern rule for which the prerequisites exist specifies further dependencies.

The following pattern rule maintains an object library where all the C source files share a common include file `defs.h`.

```
arch.a(%.o) : %.c defs.h
      cc -c $%.c
      ar r arch.a $%.o
      rm $%.o
```

A set of default pattern rules is built in, and effectively follows the user's list of rules. Assuming these rules, which tell, among other things, how to make `.o` files from `.c` files, the first example becomes:

```
pgm: a.o b.o
      cc a.o b.o -lplot -o pgm
a.o b.o: ab.h
```

Here, greatly simplified, is a sample of the built-in rules:

```
 CC = cc
 %.o: %.c
      $(CC) $(CFLAGS) -c $%.c
 %.o: %.f
      f77 $(FFLAGS) -c $%.f
 % : %.c
      $(CC) $(CFLAGS) -o $% $%.c
```

The first rule says that a name ending in `.o` could be made if a matching name ending in `.c` were present. The second states a similar rule for files ending in `.f`. The third says that an arbitrary name can be made by compiling a file with that name suffixed by `.c`.

Macros make the builtin pattern rules flexible: `CC` names the particular C compiler, `CFLAGS` gives *cc*(1) options, `FFLAGS` for *f77*(1), `LFLAGS` for *lex*(1), `YFLAGS` for *yacc*(1), and `PFLAGS` for *pascal*(A).

An older, now disparaged, means of specifying default rules is based only on suffixes. Prerequisites are inferred according to selected suffixes listed as the 'prerequisites' for the special name `.SUFFIXES`; multiple lists accumulate; an empty list clears what came before.

The rule to create a file with suffix *s2* that depends on a similarly named file with suffix *s1* is specified as an entry for the 'target' *s1s2*. Order is significant; the first possible name for which both a file and a rule exist is inferred. An old style rule for making optimized `.o` files from `.c` files is

```
.SUFFIXES: .c .o
.c.o: ; cc -c -O -o $@ $*.c
```

The following two macros are defined for use in any rule:

`$($@)` full name of target
`$($/)` target name beginning at the last slash, if any

A number of other special macros are defined automatically in rules invoked by one of the implicit mechanisms:

`$*` target name with suffix deleted
`$@` full target name
`$<` list of prerequisites in an implicit rule
`$?` list of prerequisites that are out of date
`$^` list of all prerequisites

The following are included for consistency with System V:

`$(@D)` directory part of `$@` (up to last slash)
`$(@F)` file name part of `$@` (after last slash)
`$(*D)` directory part of `$*` (up to last slash)
`$(*F)` file name part of `$*` (after last slash)
`$(<D)` directory part of `$<` (up to last slash)

`$(<F)` file name part of `$<` (after last slash)

Recipe lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the special target `.SILENT` is in the makefile, or the first character of the command is `@`.

Commands that return nonzero status cause *make* to terminate unless the special target `.IGNORE` is in the makefile or the command begins with <tab><hyphen>.

Interrupt and quit cause the target to be deleted unless the target depends on the special name `.PRECIOUS`.

Make includes a rudimentary parallel processing ability. If the separation string is `:&` or `::&` , *make* can run the command sequences to create the prerequisites simultaneously. If two names are separated by an ampersand on the right side of a colon, those two may be created in parallel.

Other options:

`-i` Equivalent to the special entry `.IGNORE: .`

`-k` When a command returns nonzero status, abandon work on the current entry, but continue on branches that do not depend on the current entry.

`-n` Trace and print, but do not execute the commands needed to update the targets.

`-t` Touch, i.e. update the modified date of targets, without executing any commands.

`-r` Turn off built-in rules.

`-s` Equivalent to the special entry `.SILENT:`.

`-e` Environment definitions override conflicting definitions in arguments or in makefiles. Ordinary precedence is argument over makefile over environment.

`-o` Assume old style default suffix list: `.SUFFIXES: .out .o .c .e .r .f .y .l .s .p`

`-P`*n* Permit *n* command sequences to be done in parallel with `&`.

`-z` Run commands by passing them to the shell; normally simple commands are run directly by *exec*(2).

FILES

```
makefile
Makefile
```

SEE ALSO

sh(1), *touch* in *chdate*(1), *ar*(1), *mk*(1)

BUGS

Comments can’t appear on recipe lines.
Archive entries are not handled reliably.

NAME

man – print pages of this manual

SYNOPSIS

`man` [*option* ...] [*chapter*] *title* ...

`man -k` *pattern*

DESCRIPTION

Man locates and prints pages of this manual named *title* in the specified *chapter*. *Title* is given in lower case. The *chapter* number is a single digit, 1-9; pages marked (3S) or (9.1), for example, belong to chapters 3 and 9 respectively. If no *chapter* is specified, pages in all chapters, including the unprinted appendix, are printed.

On some machines a cache of preformatted pages is available. If so, and if neither option `-t` or `-n` is present, option `-q`, for quick printing, is assumed.

The options are:

`-q` Copy cached pages to the standard output or, if they are out of date or unavailable, act as `-n`. Any name from the NAME list at the top of the page will serve as a *title* in the cache. If the standard output is directed to a terminal, filter it through *ul*(1).

`-t` Use *troff*(1) to place on the standard output intermediate code to drive the typesetting devices of *lp*(1), *apsend*(1), or *proof*(9.1).

`-n` Print the pages on the standard output using *nroff*.

`-f` The *titles* are actual names of manual source files. Assume option `-t` unless `-n` is present.

`-k` Report the NAME lines of all manual pages that matches the *pattern*. The *pattern* is as in *egrep*; see *gre*(1).

Under `-t` and `-n` further options, e.g. to specify the kind of terminal you have, are passed on to *troff* or *nroff*.

From time to time a daemon updates cached pages that are out of date and supplies links for subsidiary entries.

EXAMPLES

`man man`
Reproduce this page and *man*(6).

`man -t eqn eqnchar | lp`
Format the *eqn* and *eqnchar* pages and send them to the laser printer.

`man -k 'copy.*file|file.*copy'`
Look for file-copying utilities.

FILES

`/usr/man/man?/*.?`	*troff* source for manual; this page is `/usr/man/man1/man.1`
`/usr/spool/man/man?/*.?`	manual cache
`/usr/man/man0/cache`	cache daemon; invoked by *cron*(8)

SEE ALSO

troff(1), *man*(6), *apsend*(1), *lp*(1), *proof*(9.1)

BUGS

The manual was intended to be typeset; some detail is sacrificed on terminals.
You can't ask for manual pages named 1 through 9.
Pages not available in the cache can be located by their proper titles only.
Cache entries by subsidiary names are always deemed up to date.

NAME

matlab – interactive matrix desk calculator

SYNOPSIS

`/usr/lbin/matlab`

DESCRIPTION

Matlab manipulates complex matrices interactively. Special cases include real matrices and scalars. Operations include pseudoinversion (which is inversion for square nonsingular matrices), eigendecomposition, various other factorizations, solution of linear equations, matrix products (including inner and outer products), Kronecker products, log, exponential and trigonometric functions of matrices, and rank and condition estimation.

Variables are alphanumeric strings of at most 4 characters. Case is ignored. Expressions and assignment statements are written as in Fortran. Multiple statements can be put on one line, separated by either comma or semicolon; the result of a statement is written on the standard output unless a semicolon follows the statement. Extensions to Fortran notation include:

Matrix construction from elements. Elements in a row are separated by commas; columns are separated by semicolons; matrices are surrounded by < > brackets.

Transpose is indicated by postfix prime `'`.

Consecutive integers are denoted by colons in the style 1:4 or 1:2:8 (meaning 1,3,5,7). `A(2:5)` is a subarray; `A(:,j)` is a column.

Identity matrix is denoted `eye`; its dimensions are dictated by context.

Reverse division is denoted by `\`. For example, `x = A \ b` is roughly the same as `x = inv(A) * b`, except that Gaussian elimination, if applicable, is used to compute *x*.

Some *matlab* commands:

`help`
`help` *word*
: List commands and functions, or specific information about a *word*:

ABS	ANS	ATAN	BASE	CHAR	CHOL	CHOP	CLEA	COND	CONJ	COS
DET	DIAG	DIAR	DISP	EDIT	EIG	ELSE	END	EPS	EXEC	EXIT
EXP	EYE	FILE	FLOP	FLPS	FOR	FUN	HESS	HILB	IF	IMAG
INV	KRON	LINE	LOAD	LOG	LONG	LU	MACR	MAGI	NORM	ONES
ORTH	PINV	PLOT	POLY	PRIN	PROD	QR	RAND	RANK	RCON	RAT
REAL	RETU	RREF	ROOT	ROUN	SAVE	SCHU	SHOR	SEMI	SIN	SIZE
SQRT	STOP	SUM	SVD	TRIL	TRIU	USER	WHAT	WHIL	WHO	WHY

`save('`*file*`')`
`save('`*file*[,*var*]...**')**
: Save all current variables, or just the designated variables in *file.*

`load('`*file*`')`
: Restore saved variables.

`exec('`*file*`')`
: Execute the commands in *file* before reading more commands from the standard input.

Matlab can be called as a subroutine. For details, see the reference or type `help user`.

FILES

`/usr/lib/mathelp.dac`
`/usr/lib/mathelp.idx`

SEE ALSO

Cleve Moler, *MATLAB User's Guide*, Technical Report CS81−1 (Revised), Dept. of Computer Science, University of New Mexico, 1982. (Available in troff form with the Matlab source.)

NAME

mk, mkconv, membername − maintain (make) related files

SYNOPSIS

`mk` [`-f` *mkfile*] ... [*option* ...] [*name* ...]

`mkconv` *makefile*

`membername` *aggregate* ...

DESCRIPTION

Mk is most often used to keep object files current with the source they depend on.

Mk reads *mkfile* and builds and executes dependency dags (directed acyclic graphs) for the target *names*. If no target is specified, the targets of the first non-metarule in the first *mkfile* are used. If no `-f` option is present, `mkfile` is tried. Other options are:

`-a` Assume all targets to be out of date. Thus, everything gets made.
`-d[egp]` Produce debugging output (`p` is for parsing, `g` for graph building, `e` for execution).
`-e` Explain why each target is made.
`-i` Force any missing intermediate targets to be made.
`-k` Do as much work as possible in the face of errors.
`-m` Generate an equivalent makefile on standard output. Recipes are not handled well.
`-n` Print, but do not execute, the commands needed to update the targets.
`-t` Touch (update the modified date of) non-virtual targets, without executing any recipes.
`-u` Produce a table of clock seconds spent with *n* recipes running.
`-w`*name1,name2,...*
Set the initial date stamp for each name to the current time. The names may also be separated by blanks or newlines. (Use with `-n` to find what else would need to change if the named files were modified.)

Mkconv attempts to convert a *make*(1) *makefile* to a *mkfile* on standard output. The conversion is not likely to be faithful.

The shell script *membername* extracts member names (see 'Aggregates' below) from its arguments.

Definitions

A *mkfile* consists of *assignments* (described under 'Environment') and *rules*. A rule contains *targets* and a *tail.* A target is a literal string, or *label,* and is normally a file name. The tail contains zero or more *prerequisites* and an optional *recipe,* which is a shell script.

A *metarule* has a target of the form *A*`%`*B* where *A* and *B* are (possibly empty) strings. A metarule applies to any label that matches the target with `%` replaced by an arbitrary string, called the *stem*. In interpreting a metarule, the stem is substituted for all occurrences of `%` in the prerequisite names. A metarule may be marked as using regular expressions (described under 'Syntax'). In this case, `%` has no special meaning; the target is interpreted according to *regexp*(3). The dependencies may refer to subexpressions in the normal way, using `\`*n.* The *dependency dag* for a target consists of *nodes* connected by directed *arcs*. A node consists of a label and a set of arcs leading to prerequisite nodes. The root node is labeled with an original target *name.*

Building the Dependency Dag

Read the *mkfiles* in command line order and distribute rule tails over targets to get single-target rules.

For a node *n*, for every rule *r* that matches *n*'s label generate an arc to a prerequisite node. The node *n* is then marked as done. The process is then repeated for each of the prerequisite nodes. The process stops if *n* is already done, or if *n* has no prerequisites, or if any rule would be used more than `$NREP` times on the current path in the dag. A probable node is one where the label exists as a file or is a target of a non-metarule.

After the graph is built, it is checked for cycles, and subdags containing no probable nodes are deleted. Also, for any node with arcs generated by a non-metarule with a recipe, arcs generated by a metarule

with a recipe are deleted. Disconnected subdags are deleted.

Execution

Labels have an associated date stamp. A label is *ready* if it has no prerequisites, or all its prerequisites are made. A ready label is *trivially uptodate* if it is not a target and has a nonzero date stamp, or it has a nonzero date stamp, and all its prerequisites are made and predate the ready label. A ready label is marked *made* (and given a date stamp) if it is trivially uptodate or by executing the recipe associated with the arcs leading from the node associated with the ready label. The `P` attribute can be used to generalize *mk*'s notion of determining if prerequisites predate a label. Rather than comparing date stamps, it executes a specified program and uses the exit status.

Date stamps are calculated differently for virtual labels, for labels that correspond to extant files, and for other labels. If a label is *virtual* (target of a rule with the `V` attribute), its date stamp is initially zero and upon being made is set to the most recent date stamp of its prerequisites. Otherwise, if a label is nonexistent (does not exist as a file), its date stamp is set to the most recent date stamp of its prerequisites, or zero if it has no prerequisites. Otherwise, the label is the name of a file and the label's date stamp is always that file's modification date.

Nonexistent labels which have prerequisites and are prerequisite to other label(s) are treated specially unless the `-i` flag is used. Such a label *l* is given the date stamp of its most recent prerequisite and if this causes all the labels which have *l* as a prerequisite to be trivially uptodate, *l* is considered to be trivially uptodate. Otherwise, *l* is made in the normal fashion.

Two recipes are called identical if they arose by distribution from a single rule as described above. Identical recipes may be executed only when all their prerequisite nodes are ready, and then just one instance of the identical recipes is executed to make all their target nodes.

Files may be made in any order that respects the preceding restrictions.

A recipe is executed by supplying the recipe as standard input to the command

```
/bin/sh -e
```

The environment is augmented by the following variables:

`$alltarget`	all the targets of this rule.
`$newprereq`	the prerequisites that caused this rule to execute.
`$nproc`	the process slot for this recipe. It satisfies 0≤`$nproc`<`$NPROC`, where `$NPROC` is the maximum number of recipes that may be executing simultaneously.
`$pid`	the process id for the *mk* forking the recipe.
`$prereq`	all the prerequisites for this rule.
`$stem`	if this is a metarule, `$stem` is the string that matched `%`. Otherwise, it is empty. For regular expression metarules, the variables `stem0`, ..., `stem9` are set to the corresponding subexpressions.
`$target`	the targets for this rule that need to be remade.

Unless the rule has the `Q` attribute, the recipe is printed prior to execution with recognizable shell variables expanded. To see the commands print as they execute, include a `set -x` in your rule. Commands returning nonzero status (see *intro*(1)) cause *mk* to terminate.

Aggregates

Names of the form *a*(*b*) refer to member *b* of the aggregate *a*. Currently, the only aggregates supported are *ar*(1) archives.

Environment

Rules may make use of shell (or environment) variables. A legal shell variable reference of the form `$OBJ` or `${name}` is expanded as in *sh*(1). A reference of the form `${name:`*A*`%`*B*`=`*C*`%`*D*`}`, where *A*, *B*, *C*, *D* are (possibly empty) strings, has the value formed by expanding `$name` and substituting *C* for *A* and *D* for *B* in each word in `$name` that matches pattern *A*`%`*B*.

Variables can be set by assignments of the form

var=[*attr*=]*tokens*

where *tokens* and the optional attributes are defined under 'Syntax' below. The environment is exported to recipe executions. Variable values are taken from (in increasing order of precedence) the default values below, the environment, the mkfiles, and any command line assignment. A variable assignment argument overrides the first (but not any subsequent) assignment to that variable.

```
AS=as                       FFLAGS=                     NPROC=1
CC=cc                       LEX=lex                     NREP=1
CFLAGS=                     LFLAGS=                     YACC=yacc
FC=f77                      LDFLAGS=                    YFLAGS=
BUILTINS='
%.o:   %.c
       $CC $CFLAGS -c $stem.c
%.o:   %.s
       $AS -o $stem.o $stem.s
%.o:   %.f
       $FC $FFLAGS -c $stem.f
%.o:   %.y
       $YACC $YFLAGS $stem.y &&
       $CC $CFLAGS -c y.tab.c && mv y.tab.o $stem.o; rm y.tab.c
%.o:   %.l
       $LEX $LFLAGS -t $stem.l > $stem.c &&
       $CC $CFLAGS -c $stem.c && rm $stem.c'
ENVIRON=
```

The builtin rules are obtained from the variable `BUILTINS` after all input has been processed. The `ENVIRON` variable is split into parts at control-A characters, the control-A characters are deleted, and the parts are placed in the environment. The variable `MKFLAGS` contains all the option arguments (arguments starting with − or containing =) and `MKARGS` contains all the targets in the call to *mk*.

Syntax

Leading white space (blank or tab) is ignored. Input after an unquoted `#` (a comment) is ignored as are blank lines. Lines can be spread over several physical lines by placing a `\` before newlines to be elided. Non-recipe lines are processed by substituting for `` `cmd` `` and then substituting for variable references. Finally, the filename metacharacters `[]*?` are expanded. Quoting by `''`, `""`, and `\` is supported. The semantics for substitution and quoting are given in *sh*(1).

The contents of files may be included by lines beginning with < followed by a filename.

Assignments and rule header lines are distinguished by the first unquoted occurrence of `:` (rule header) or = (assignment).

A rule definition consists of a header line followed by a recipe. The recipe consists of all lines following the header line that start with white space. The recipe may be empty. The first character on every line of the recipe is elided. The header line consists of at least one target followed by the rule separator and a possibly empty list of prerequisites. The rule separator is either a single `:` or is a `:` immediately followed by attributes and another `:`. If any prerequisite is more recent than any of the targets, the recipe is executed. This meaning is modified by the following attributes

- `<` The standard output of the recipe is read by *mk* as an additional mkfile. Assignments take effect immediately. Rule definitions are used when a new dependency dag is constructed.
- `D` If the recipe exits with an error status, the target is deleted.
- `N` If there is no recipe, the target has its time updated.
- `P` The characters after the `P` until the terminating `:` are taken as a program name. It will be invoked as `sh -c prog 'arg1' 'arg2'` and should return 0 exit status if and only if arg1 is not out of date with respect to arg2. Date stamps are still propagated in the normal way.

`Q` The recipe is not printed prior to execution.
`R` The rule is a metarule using regular expressions.
`U` The targets are considered to have been updated even if the recipe did not do so.
`V` The targets of this rule are marked as virtual. They are distinct from files of the same name.

Similarly, assignments may have attributes terminated by =. The only assignment attribute is

`U` Do not export this variable to recipe executions.

EXAMPLES

A simple mkfile to compile a program.

```
prog:  a.o b.o c.o
       $CC $CFLAGS -o $target $prereq
```

Override flag settings in the mkfile.

```
$ mk target CFLAGS='-O -s'
```

To get the prerequisites for an aggregate.

```
$ membername 'libc.a(read.o)' 'libc.a(write.o)'
read.o write.o
```

Maintain a library.

```
libc.a(%.o):N:%.o
libc.a:libc.a(abs.o) libc.a(access.o) libc.a(alarm.o) ...
       names=`membername $newprereq`
       ar r libc.a $names && rm $names
```

Backquotes used to derive a list from a master list.

```
NAMES=alloc arc bquote builtins expand main match mk var word
OBJ=`echo $NAMES|sed -e 's/[^ ][^ ]*/&.o/g'`
```

Regular expression metarules. The single quotes are needed to protect the \s.

```
'([^/]*)/(.*)\.o':R:  '\1/\2.c'
       cd $stem1; $CC $CFLAGS -c $stem2.c
```

A correct way to deal with *yacc*(1) grammars. The file `lex.c` includes the file `x.tab.h` rather than `y.tab.h` in order to reflect changes in content, not just modification time.

```
YFLAGS=-d
lex.o: x.tab.h
x.tab.h:y.tab.h
       cmp -s x.tab.h y.tab.h || cp y.tab.h x.tab.h
y.tab.c y.tab.h:gram.y
       $YACC $YFLAGS gram.y
```

The above example could also use the `P` attribute for the `x.tab.h` rule:

```
x.tab.h:Pcmp -s:y.tab.h
       cp y.tab.h x.tab.h
```

SEE ALSO

make(1), *chdate*(1), *sh*(1), *regexp*(3)
A. Hume, '*Mk*: a Successor to *Make*', this manual, Volume 2

BUGS

Identical recipes for regular expression metarules only have one target.
Seemingly appropriate input like `CFLAGS=-DHZ=60` is parsed as an erroneous attribute; correct it by inserting a space after the first =.

NAME

mkdir – make a directory

SYNOPSIS

`mkdir` *dirname* ...

DESCRIPTION

Mkdir creates specified directories in mode 777, subject to masking by *umask*(2). Standard entries, '`.`', for the directory itself, and '`..`' for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

rm(1) *cd* in *sh*(1)

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made. Otherwise it prints a diagnostic and returns nonzero.

NAME

monk, monksample, monkspell, monkmerge − typeset documents and letters

SYNOPSIS

`monk` [*options* | *files*]

`monksample` [*sample*]

`monkspell` [*options*] [*files*]

`monkmerge` [*files*]

DESCRIPTION

Monk formats the text in the named *files* for phototypesetting, using other *troff*(1) preprocessors as necessary. *Options* are given below. Any other arguments or flags (e.g., `-o1-2`) are passed to *troff*. Options can occur in any order and can be intermixed with files.

`-A`*commands*
: Invoke the *commands* after all preprocessors and before *troff*.

`-B`*commands*
: Invoke the *commands* after *monk* and before any other preprocessor.

`-E` Invoke the `-e` option of *troff*.

`-N` Use the uncompressed monk databases. This facilitates debugging monk database entries.

`-R`*file* Use *file* as the index file for *prefer*(1).

`-S`*dest* Send output to device *dest*. Supported forms are:

`-Sapsend`	Linotronic L200P; see *apsend*(1).
`-Sd202`	Mergenthaler Linotron 202; see *d202*(A).
`-Slp`	Postscript line printer; see *lp*(1) (default).
`-Sproof`	Teletype 5620 or 630 terminal; see *proof*(9.1).
`-Sthink`	HP ThinkJet; see *thinkblt*(9.1).
`-S-`	Standard output.

`-T`*device*
: Prepare output for device specified as in `-T` option of *troff*(1).

`-x` Shows the preprocessors that are being invoked.

The following options are not normally needed because monk automatically determines which preprocessors are required. However, if the commands in `-A` or `-B` options require a preprocessor, their use can be forced by the following options.

`-c`	*col* postprocessor; see *column*(1). (Automatically invoked for many printing terminals.)
`-cn`	*cite*(A)
`-e`	*eqn*(1)
`-g`	*grap*(1)
`-i`	*ideal*(1)
`-ipa`	*ipa*(A)
`-p`	*pic*(1)
`-r`	*prefer*(1)
`-t`	*tbl*(1)
`-tp`	*tped*; see *ped*(9.1)

Monksample produces on the standard output a skeleton document that you can redirect into a file and edit. If no argument is given, *monksample* prints a list of the available *samples.* They are:

`acm`	Association for Computing Machinery galley sheets.
`centerpb`	Center Phone Book.
`cspress`	Computer Science Press galley sheets.

`form1`	AT&T Bell Laboratories merit review form 1.
`im`	AT&T Bell Laboratories internal memorandum.
`kluwer`	Kluwer Academic Publishers book format.
`letter`	Letters with optional AT&T letterhead.
`memo`	Internal AT&T correspondence.
`model`	IEEE/ACM model sheets.
`research`	AT&T Bell Laboratories bi-annual research report.
`rp`	AT&T Bell Laboratories release paper.
`song`	Song sheets for singing at nursing homes.
`tc`	AT&T Bell Laboratories technical correspondence.
`tm`	AT&T Bell Laboratories technical memorandum.

Monkspell looks up words from the named *files* (standard input default) in a public spelling list and in a private list. Possible misspellings—words that occur in neither and are not plausibly derivable from the former—are placed on the standard output. It ignores constructs of *monk*(1), *troff*(1) and its standard preprocessors. It runs *demonk*(1) with all specified options and passes its output to *spell*(1). The following options, in addition to all options for *deroff*(1), are available:

`-d`*dir* Use non-standard *monk* database directory *dir*.

`-i` Ignore *monk* `|insert` and `|source` commands as well as *troff* `.so` and `.nx` requests.

Monkmerge reads each file and copies it to standard output, replacing *monk* `insert` commands with the file contents. It ignores *monk* `source` commands, which include a file without processing the *monk* commands within. If no input file is given, *monkmerge* reads from standard input.

EXAMPLES

```
monk paper
```
Format the file using the default typesetter fonts and output device.

```
monk -Sproof paper
```
Format a file and proof it on a 5620 terminal.

FILES

`$MHOME/cite`	forward and backward reference preprocessor
`$MHOME/db`	monk databases
`$MHOME/monk`	monk compiler
`$MHOME/tmac.p`	macros for *pic*(1) preprocessor
`$MHOME/sample`	directory for existing samples

SEE ALSO

prefer(1), *troff*(1), *deroff*(1), *tex*(1), *lp*(1), *apsend*(1), *d202*(A), *thinkblt*(9.1), *proof*(9.1)
Murrel, S. L., Kowalski, T. J., ‘Typing Documents on the UNIX System: Using Monk 0.6’, this manual, Volume 2

NAME

newgrp – change to a new group

SYNOPSIS

`newgrp` *group* [*command* [*arg* ...]]

DESCRIPTION

Newgrp runs a *command* with the (real and effective) groupid temporarily set to *group*. If no command is given, the user's shell (see *passwd*(5)) is executed.

A password is demanded if the group has a password and the user does not.

EXAMPLES

```
exec newgrp bin
```

Restart the shell with a different groupid.

FILES

```
/etc/group
/etc/passwd
```

SEE ALSO

login(8), *getuid*(2)

BUGS

On other systems, *newgrp* is built into the shell. Here it will spawn a new shell unless invoked with `exec`.

NAME

nice, renice, nohup – run commands at low priority or immune to hangup

SYNOPSIS

`nice` [−*number*] *command* [*argument* ...]

`/etc/renice` [−*number*] *pid* ...

`nohup` *command* [*argument* ...]

DESCRIPTION

Nice executes *command* with low scheduling priority. If the *number* argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 19. The default *number* is 10.

The super-user may run commands with priority higher than normal by using a negative *number*, e.g. `−−10`.

Renice increments the scheduling priority of the processes with the named *process-ids* by *number*. The default *number* is 19, making the process least likely to run.

Only the owner of the process or the super-user may change the priority. Only the super-user may use negative increments.

Nohup executes *command* immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5.

Any output not explicitly redirected is appended to the file `nohup.out` in the current directory.

FILES

`nohup.out` default destination for standard output and standard error

SEE ALSO

nice(2)

FILES

`/proc` (*renice*)

DIAGNOSTICS

Nice returns the exit status of the subject command.

BUGS

Quoted *arguments* don't work right in all cases. The difficulty may be avoided by quoting the *command*, with arguments in inner quotes.

NAME

nm – name list (symbol table)

SYNOPSIS

`nm` [`-agnopru`] [*file* ...]

DESCRIPTION

Nm prints the name list of each object *file* in the argument list. If an argument is a library archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in `a.out` are listed.

Each symbol name is preceded by its hexadecimal value (blanks if undefined) and one of the letters

`U`	undefined
`A`	absolute
`T`	text segment symbol
`D`	data segment symbol
`B`	bss segment symbol
`C`	common symbol
`f`	source file name
`-`	extra symbols for debuggers; see `-a` below

If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

`-a`	Print all symbols; normally extra debugger symbols are excluded.
`-g`	Print only global (external) symbols.
`-n`	Sort numerically rather than alphabetically.
`-o`	Prepend file or archive element name to each output line rather than printing it once separately.
`-p`	Don’t sort; print in symbol-table order.
`-r`	Sort in reverse order.
`-u`	Print only undefined symbols.

SEE ALSO

ar(1), *ar*(5), *a.out*(5), *stab*(5), *adb*(1), *pi*(9.1)

NAME

ocr − optical character recognition

SYNOPSIS

`ocr` [*option* ...] [*file*]

DESCRIPTION

Ocr reads a black-and-white image of a page from *file*, and writes ASCII to the standard output. If no *file* is specified, it reads from the standard input.

The input is a *picfile*(5) image of one column of machine-printed text, normally scanned in by *cscan*(1). Fonts, sizes, and line-spacings may vary within the column, but each line should have a constant text size and baseline. Lines should be parallel and roughly horizontal.

In the output, white space approximates the original page layout. Words that *spell*(1) are preferred, and hyphenations across lines are recombined.

The options are:

`-a`*s* The alphabet is the union of symbol sets selected by characters in string *s*, from among:

`A`	ABCDEFGHIJKLMNOPQRSTUVWXYZ	
`a`	abcdefghijklmnopqrstuvwxyz	
`0`	0123456789	
`.`	.,-:;*’"?!/&$()[]#@%	(basic punctuation)
`^`	^~‘|{}_	(extended punct’n)
`+`	+−*/<>=.Ee[]	(numerical punct’n)
`s`	§†‡¢•©®°’− −—	(selected non-ASCII)
`l`	fi fl ff ffi ffl r s i j	(ligatures and digraphs)
`g`	αβγδεζηθικλμνξοπρστυφχψω	(Greek lower case)
`G`	ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ	(Greek upper case)

The default is `-aAa0.+^`, the full printable-ASCII set, which may be abbreviated as `-ap`. Thus, `-apslgG` selects all of the above.

`-c` Find columns in complex nested layouts using greedy white covers algorithm.

`-m`*l[,r]* Trim the left and right margins of the image by *l* and *r* inches, respectively, before looking for columns. If *r* is omitted, it is assumed to equal *l*.

`-n`*n* Find the *n* largest columns by analysis of a single vertical projection. Each column should be compactly-printed and separated from the others by at least 2 ems of horizontal white space.

`-p`*n,m* Point sizes lie in the range [*n*, *m*]; other sizes are discarded. The default is `-p6,24`.

`-s` Defeat spelling check (but continue to favor numeric strings and good punctuation).

`-t` Write *troff*(1) format. Each column is shown on a separate page, lines at their original height, words at their original horizontal location, and characters roughly original size in Times roman. Hyphenated words are not recombined.

`-u` Unspellable words are prefixed with ‘?’ or, if `-t` is specified, printed boldface.

`-w`*w* Find the largest column of width *w* inches, within a single vertical projection.

Fonts

Trained on over 100 Latin-alphabet book fonts in various italic, bold, etc styles. Only one font of Greek, without diacriticals. Also Swedish and Tibetan, on request.

SEE ALSO

bcp(1), *cscan*(1), *font*(6), *picfile*(5), *spell*(1), *troff*(1)

BUGS

For best results, use images of high-contrast, cleanly-printed original documents digitized at a resolution of 400 pixels/inch or higher. It may help to restrict the alphabet and sizes to what’s there.

NAME

p, pg, more – paginate

SYNOPSIS

`p` [–*number*] [*file* ...]

DESCRIPTION

P copies its standard input, or the named files if given, to its standard output, stopping at the end of every 22nd line, and between files, to wait for a newline from the user. The page size may be set by saying (for example)

```
p -60
```

While waiting for a newline, *p* interprets some commands:

– Reprint last page. –– reprints the second last page, etc.

`!` Pass the rest of the line to the shell as a command.

`q` Quit.

Pg and *more* are synonyms for *p.*

BUGS

Because of limited storage, *p* can't back up too far.
Pg and *more* exist only to placate old programs that call paginators.

NAME

pack, unpack, pcat, compress, uncompress, zcat – compress and expand files

SYNOPSIS

`pack` [`-`] *file* ...

`unpack` *file* ...

`pcat` [*file* ...]

`compress` [*option* ...] [*file* ...]

`uncompress` [*option* ...] [*file* ...]

`zcat` [`-V`] [*file* ...]

DESCRIPTION

Pack attempts to compress the *files* and places the results in corresponding files named *file*`.z` with the same access modes, dates, and owner as the originals. Successfully packed files are removed.

Unpack reverses the process.

Pcat unpacks files to the standard output.

The `.z` suffix may be omitted from the name of the input file for *unpack* or *pcat.*

Pack encodes individual characters in a Huffman code. Option - causes statistics of the encoding to be printed. The option toggles on and off at each appearance among the list of *files.*

Compress, uncompress, and *zcat* work like *pack, unpack,* and *pcat,* putting each compressed *file* into *file*`.Z`. The options are

`-f` (force) Compress even when it doesn't save space.

`-c` Write to the standard output; change no files. *Zcat* is identical to *uncompress* `-c`.

`-b`*bits* *Compress* uses a modified Lempel-Ziv encoding. Common substrings in the file are replaced by variable-length codes up to size *bits* (default 16). Smaller limits devour less address space.

`-v` Print percent reduction for each file.

`-V` Print program version number.

Compress-uncompress pack better and are faster overall; *pack-unpack* work on smaller machines and are much more widely available.

SEE ALSO

T. A. Welch, 'A Technique for High Performance Data Compression,' *IEEE Computer,* 17 (1984) 8-19.

DIAGNOSTICS

The exit code of *pack, unpack,* or *pcat* is the number of files it failed to process.

The exit code of *compress, uncompress,* or *zcat* is 0 normally, 1 for error, 2 for ineffective compression (i.e. expansion).

NAME

passwd – change login password

SYNOPSIS

`passwd` [`-an`] [*name*]

DESCRIPTION

This command changes a password associated with the user *name* (your own name by default).

The program prompts for the old password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password.

If the `-a` option is given, *passwd* prompts for new values of certain fields of the password file entry.

The super-user may use the `-n` option to install new users. The prompts are self-explanatory, and most of the defaults obvious. A null response to the `UID:` prompt assigns a numeric userid one greater than the largest one previously in `/etc/passwd`. A null response to `Directory:` assigns a home directory in `/usr`. If the first character of the response to this prompt is an asterisk, the remaining characters are taken as the name of the new user's home directory, and a symbolic link to this directory is placed in `/usr`.

If `/etc/stdprofile` exists, each new user's home directory starts with a file name `.profile`, which is a copy of `/etc/stdprofile` with `\N` replaced by the user's name, and `\D` replaced by the name of the home directory.

FILES

`/etc/passwd`
`/etc/stdprofile`

SEE ALSO

crypt(3) *passwd*(5)
Robert Morris and Ken Thompson, 'UNIX password security,' *AT&T Bell Laboratories Technical Journal* 63 (1984) 1649-1672

BUGS

The password file information should be kept in a different data structure allowing indexed access.

NAME

pic, tpic – troff and tex preprocessors for drawing pictures

SYNOPSIS

`pic` [*files*]

`tpic` [*files*]

DESCRIPTION

Pic is a *troff*(1) preprocessor for drawing figures on a typesetter. *Pic* code is contained between `.PS` and `.PE` lines:

> `.PS` *optional-width optional-height*
> *element-list*
> `.PE`
> If *optional-width* is present, the picture is made that many inches wide, regardless of any dimensions used internally. The height is scaled in the same proportion unless *optional-height* is present. If `.PF` is used instead of `.PE`, the typesetting position after printing is restored to what it was upon entry.

A line of the form

> `.PS<file` causes *pic* to treat the the named file as if it stood in place of the `.PS` line.

An *element-list* is a list of elements:

> *primitive attribute-list*
> *placename* **:** *element*
> *placename* **:** *position*
> *var* **=** *expr*
> *direction*
> `{` *element-list* `}`
> `[` *element-list* `]`
> `for` *var* **=** *expr* `to` *expr* `by` *expr* `do {` *anything* `}`
> `if` *expr* `then {` *anything* `} else {` *anything* `}`
> `copy` *file*, `copy thru` *macro*, `copy` *file* `thru` *macro*
> `sh {` *commandline* `}`
> `print` *expr*
> `reset` *optional var-list*
> *troff-command*

Elements are separated by newlines or semicolons; a long element may be continued by ending the line with a backslash. Comments are introduced by a `#` and terminated by a newline. Variable names begin with a lower case letter; place names begin with upper case. Place and variable names retain their values from one picture to the next.

After each primitive the current position moves in the current direction (`up`,`down`, `left`,`right` (default)) by the size of the primitive. The current position and direction are saved upon entry to a `{...}` block and restored upon exit. Elements within a block enclosed in `[...]` are treated as a unit; the dimensions are determined by the extreme points of the contained objects. Names, variables, and direction of motion within a block are local to that block.

troff-command is any line that begins with a period. Such a line is assumed to make sense in the context where it appears; generally, this means only size and font changes. Changes to vertical spacing will produce broken pictures.

The *primitive* objects are:

> `box circle ellipse arc line arrow spline move` *text-list*

`arrow` is a synonym for `line ->`.

An *attribute-list* is a sequence of zero or more attributes; each attribute consists of a keyword, perhaps followed by a value.

`h(eigh)t` *expr*	`wid(th)` *expr*
`rad(ius)` *expr*	`diam(eter)` *expr*
`up` *opt-expr*	`down` *opt-expr*
`right` *opt-expr*	`left` *opt-expr*
`from` *position*	`to` *position*
`at` *position*	`with` *corner*
`by` *expr, expr*	`then`
`dotted` *opt-expr*	`dashed` *opt-expr*
`chop` *opt-expr*	`->` `<-` `<->`
`invis`	`same`
text-list	*expr*

Missing attributes and values are filled in from defaults. Not all attributes make sense for all primitives; irrelevant ones are silently ignored. The attribute `at` causes the geometrical center to be put at the specified place; `with` causes the position on the object to be put at the specified place. For lines, splines and arcs, `height` and `width` refer to arrowhead size. A bare *expr* implies motion in the current direction.

Text is normally an attribute of some primitive; by default it is placed at the geometrical center of the object. Stand-alone text is also permitted. A text list is a list of text items:

text-item:

`"..."` *positioning ...*

`sprintf("`*format*`",` *expr*`, ...)` *positioning ...*

positioning:

`center ljust rjust above below`

If there are multiple text items for some primitive, they are arranged vertically and centered except as qualified. Positioning requests apply to each item independently. Text items may contain in-line *troff* commands for size and font changes, local motions, etc., but make sure that these are balanced so that the entering state is restored before exiting.

A position is ultimately an *x,y* coordinate pair, but it may be specified in other ways.

position:

expr, expr

place ± *expr, expr*

place ± `(` *expr, expr* `)`

`(` *position*`,` *position* `)` — *x* from one, *y* the other

expr [*of the way*] between *position* and *position*

expr < *position* , *position* >

(*position*)

place:

placename optional-corner

corner `of` *placename*

nth primitive optional-corner

corner `of` *nth primitive*

`Here`

An *optional-corner* is one of the eight compass points or the center or the start or end of a primitive.

optional-corner:

`.n .e .w .s .ne .se .nw .sw .c .start .end`

corner:

`top bot left right start end`

Each object in a picture has an ordinal number; *nth* refers to this.

nth:

n`th`, *n*`th last`

The built-in variables and their default values are:

`boxwid = 0.75`	`boxht = 0.5`

```
circlerad = 0.25        arcrad = 0.25
ellipsewid = 0.75       ellipseht = 0.5
linewid = 0.5           lineht = 0.5
movewid = 0.5           moveht = 0.5
textwid = 0             textht = 0
arrowwid = 0.05         arrowht = 0.1
dashwid = 0.1           arrowhead = 2
scale = 1
```

These may be changed at any time, and the new values remain in force from picture to picture until changed again or reset by a `reset` statement. Variables changed within `[` and `]` revert to their previous value upon exit from the block. Dimensions are divided by `scale` during output.

Expressions in *pic* are evaluated in floating point. All numbers representing dimensions are taken to be in inches.

expr:

expr op expr
– *expr*
! *expr*
(*expr*)
variable
number
place `.x` *place* `.y` *place* `.ht` *place* `.wid` *place* `.rad`
`sin(`*expr*`)` `cos(`*expr*`)` `atan2(`*expr*`,`*expr*`)` `log(`*expr*`)` `exp(`*expr*`)`
`sqrt(`*expr*`)` `max(`*expr*`,`*expr*`)` `min(`*expr*`,`*expr*`)` `int(`*expr*`)` `rand()`

op:

`+` `-` `*` `/` `%` `<` `<=` `>` `>=` `==` `!=` `&&` `||`

The `define` and `undef` statements are not part of the grammar.

`define` *name* `{` *replacement text* `}`
`undef` *name*

Occurrences of `$1`, `$2`, etc., in the replacement text will be replaced by the corresponding arguments if *name* is invoked as

name`(`*arg1*`,` *arg2*`, ...)`

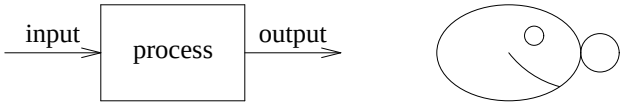
Non-existent arguments are replaced by null strings. Replacement text may contain newlines. The `undef` statement removes the definition of a macro.

Tpic is a *tex*(1) preprocessor that accepts *pic* language. It produces Tex commands that define a box called `\graph`, which contains the picture. The box may be output this way:

```
\centerline{\box\graph}
```

EXAMPLES

```
arrow "input" above; box "process"; arrow "output" above
move
A: ellipse
   circle rad .1 with .w at A.e
   circle rad .05 at 0.5 <A.c, A.ne>
   arc from A.c to A.se rad 0.5
```



SEE ALSO

cip(9.1), *ideal*(1), *ped*(9.1), *grap*(1), *dag*(1), *doctype*(1), *troff*(1)
B. W. Kernighan, ‘PIC—a Graphics Language for Typesetting’, this manual, Volume 2

NAME

pico – graphics editor

SYNOPSIS

`pico` [`-mfto`] [`-wN -hN`] [*files*]

DESCRIPTION

Pico is an interactive editor for grey-scale and color images. Editing operations are expressed in a C-like style. The options are

`-m`*n* Display on a Metheus frame buffer, `/dev/om`*n*. A missing *n* is taken to be 0.

`-f` Display on an Itoh frame buffer, `/dev/iti0`.

`-t` Show parse trees for expressions; toggled by the interactive command `tree`.

`-o` Turn off the optimizer; toggled by `optim`.

Files are referred to in expressions as `$`*n*, where *n* is the basename or an integer, see `f` below. Otherwise file names are given as strings in double quotes, which may be elided from names that do not contain `/`.

In general, the result of the previous edit operation is available under the name `old`. The destination of the current operation is called `new`.

Pico handles images with coordinates (0,0) in the upper left hand corner and (`X`,`Y`) in the lower right. Brightnesses range from 0 (black) to `Z` (white, `Z`=255). The quantities `X`,`Y`,`Z` may be used in expressions and set by options:

`-w` *n* Set the width `X` of the work area to *n* pixels, default 511.

`-h` *n* Set the height `Y`, default 511.

Pico reads commands from the standard input:

`help` Give a synopsis of commands and functions.

`a` *file*
`a` *x y w d file*
Attach a new file. Optional parameters *x* and *y* give the origin of a subrectangle in the work buffer; *w* and *d* define width and depth of the image as stored in the file.

`d` *file*
`d $`*n* Delete (close) the file.

`h` *file* Read header information from the file.

`r` *file* Read commands from *file* as if they were typed on the terminal. Can not be done recursively.

`w` *file*
`w -` *file*
Write the file, restricted to the current window (see below). Use *pico* format by default. With a minus flag, write a headerless image (red channel only, if picture is colored); see also *picfile*(5).

`nocolor`
`color` Set the number of channels updated in the work buffer to 1 (black and white) or 3 (red, green, blue).

`window` *x y w d*
Restrict the work area to a portion of the work buffer with the upper left corner at (*x,y*), and the lower right at (*x*+*w*,y+d*)*.

`get` *file*
`get $`*n*
The picture file is (re)opened and read into the work area.

`f` Show names, sizes, and file numbers of open files.

`faster`
`slower`
In slow display the screen is updated once per pixel computed; in fast display (default), once per line of pixels.

`show` *name*
Show symbol table information, such as the current value of variables. If *name* is omitted, the whole symbol table is shown.

`functions`
Print information on all user defined and builtin functions.

`def` *name* **(** *args* **)** { *program* }
Define a function, with optional arguments. Variables are declared in these styles:
```
int var;
global int var;
array var[N];
global array var[N];
```

`x` *expr* Execute the expression in a default loop over all pixels in the current window.

`x {`*program* `}`
Execute the *program.* The program must define its own control flow.

`q` Quit.

EXAMPLES

```
pico -w1280 -h1024 -m5
```
Get a work buffer that exactly fills a Metheus screen.

```
a "/tmp/images/rob"
```
Make a file accessible. It will be known henceforth as `$rob`.

```
a 1280 0 3072 512 junk
get $junk
```
Direct attention to a 512×512 subrectangle in the middle of a 3072×512 image stored in a file named junk, and read it into the workspace.

```
x new = Z - old
x new[x,y] = Z - old[x,y]
x {for(x=0; x<=X; x++) for(y=0; y<=Y; y++) new[x,y] = Z-old[x,y];}
```
Three ways to make a negative image. Note the defaults on control flow and array indexing.

```
window 0 0 256 256
x new = $1[xclamp(x*2), yclamp(y*2)]
```
Scale a 512×512 image to one quarter of the screen. The built-in functions `xclamp` and `yclamp` guard against indexing out of range.

```
x { printf("current value of %s[%d]:\t%d\n", "histo", 128, hist[128]); }
```
Turn off the default control flow (curly braces) and use the builtin function printf to check the value of an array element.

SEE ALSO

bcp(1), *imscan*(1), *flicks*(9.1), *rebecca*(9.1), *picfile*(5), *flickfile*(9.5)
G. J. Holzmann, ‘PICO Tutorial’, this manual, Volume 2
G. J. Holzmann, *Beyond Photography—the Digital Darkroom*, Prentice-Hall, 1988

NAME

plot – graphics filters

SYNOPSIS

`plot` [`-T`*terminal*]

DESCRIPTION

These filters read plotting instructions (see *plot*(5)) from the standard input, and in general produce plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* type is specified, the environment parameter `TERM` (see *environ*(5)) is used. Known *terminals* are:

`2621`
`hp` Hewlett-Packard screen terminal.

`4014`
`tek` Tektronix 4014 storage scope.

`troff` Convert to *troff*(1) input.

`5620`
`jerq` Teletype DMD display under *mux*(9.1). The filter persists after plotting, to make a layer that skips downloading on subsequent plots.

`750`
`pen` Hewlett-Packard pen plotter.

FILES

```
/usr/lib/plot/hpplot
/usr/lib/plot/trplot
/usr/lib/plot/penplot
/usr/lib/plot/tek
/usr/jerq/bin/jplot
```

SEE ALSO

plot(3), *plot*(5)

BUGS

A persistent *plot* layer on a 5620 terminal imitates `term 33`, it loses most abilities of a *mux*(9.1) layer; see *term*(9.1).
Which plotters are known depends on which computer you are on.

NAME

pr − print file

SYNOPSIS

`pr` [*option* ...] [*file* ...]

DESCRIPTION

Pr produces a printed listing of one or more *files.* The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. For no file arguments, or for a file argument −, *pr* prints its standard input.

Options apply to all following files but may be reset between files:

`−`*n* Produce *n*-column output.

`+`*n* Begin printing with page *n.*

`−b` Balance columns on last page, in case of multi-column output.

`−d` Double space.

`−h` Take the next argument as a page header (*file* by default).

`−f` Use formfeeds to separate pages. Pause, ring bell, and wait for newline before beginning.

`−l`*n* Take the length of the page to be *n* lines instead of the default 66.

`−m` Print all *files* simultaneously, each in one column.

`−n` Number the lines of each *file.*

`−o`*n* Offset the left margin *n* character positions.

`−p` Between pages pause, ring bell, and wait for newline.

`−s`*c* Separate columns by the single character *c* instead of aligning them with white space. A missing *c* is taken to be a tab.

`−t` Do not print the 5-line header or the 5-line trailer normally supplied for each page.

`−w`*n* For purposes of multi-column output, take the width of the page to be *n* characters instead of the default 72.

Inter-terminal messages via *write*(1) are forbidden during a *pr*.

FILES

`/dev/tty` to suspend messages

SEE ALSO

cat(1), *lp*(1), *thinkblt*(9.1)

DIAGNOSTICS

There are no diagnostics when *pr* is printing on a terminal.

NAME

prefer, pinvert, penter, plook, pconvert − maintain and use bibliographic references

SYNOPSIS

`prefer` [*option ...*]

`penter` [*outfile*]

`pinvert` [*option ...*] [*file ...*]

`plook` [`-p` *dbfile*] [*keyword ...*]

`pconvert` [`-d`] *file*

DESCRIPTION

Prefer is a *troff*(1) preprocessor for bibliographic references. It copies a document from the standard input to the standard output, using a bibliographic database to change symbolic references into full references ready for typesetting by *troff*(1). Although symbolic references are in the style of *monk*(1), *prefer* does not depend on *monk*. The options are:

`-n` Format for *nroff*.

`-o`*sortkey* Under the `|reference_list` command, sort according to *sortkey*, any combinations of the letters `a` (author) `d` (date), and `t` (title), rather than in database sequence. If *sortkey is* `sort`, sort according to the current style.

`-p`*dbfile* Use *dbfile* as the bibliographic database (default `prefer.out`).

`-r` Format as a released paper (technical memorandum default).

`-s`*style* Set the formatting *style* , one of `att` (default), `acm`, `apa`, `ieee`, `lsa`, `pami`, `spectrum`.

Prefer recognizes the following commands, which may appear anywhere in a document. Parentheses `()` in the commands may be replaced by any of `{} [] <>` **.**

`|reference_style(` *style arg ...*`)`

Switch to a new formatting style. All previous references are forgotten and a new list of references is begun. If *style* is `same` the current style remains (but all previous references are forgotten). Optional *args* are:

`tm` Format as a technical memorandum.

`rp` Format as a released paper.

`nroff` Format for *nroff.*

`troff` Format for *troff.*

`sort` Print a `|reference_list` in an order appropriate for the current style.

`sequence` Print a `|reference_list` in database sequence.

sortkey Print a `|reference_list` according to the *sortkey*, any combination of the letters `a`, `d`, `t` as above.

`|reference(`*keywords*
%ref_fields %flags`)`

Insert a citation mark in the current style (e.g. [7], [3], (Knuth, 1975)). One or more *keywords* cause selection from the bibliographic database. Each `%` argument must begin a new line. *%ref_field* lines override information from the database; with no *keywords* a complete reference may be given. For the form of reference fields, see the output of *penter* or the paper in Volume 2. The following *%flags* may modify the citation.

`%no_author` Exclude author information.

`%no_date` Exclude date from the citation mark.

`%no_cite` Omit the entire citation, but include the entry in the final reference list.

`%pre_text` *string* Insert *string* before the citation mark.

`%post_text` *string* Insert *string* after the citation mark

`|reference_include(` *dbfile ...*`)`

Include the contents of the database(s) *dbfile(s)* in the list of references, treating them as

`%no_cite` entries.

`|reference_placement`
Produce a list of all references specified in `|reference` or `|reference_include` commands since the beginning of the document or the last `|reference_style` or `|reference_placement`.

`|reference_list(` *dbfile ...*`)`
Format the contents of the database(s) *dbfile*.

`|reference_database(` *dbfile* `)`
Switch to database *dbfile*

Penter helps build *prefer* bibliographic databses. It prompts for a reference type, and then for admissible attributes, such as author, date, etc. A default value proposed in brackets `[]` may be accepted by typing a newline, skipped by typing spaces before the newline, or overridden by typing a new value. The character `&` appended to an attribute causes *penter* to prompt for the attribute again (to enter multiple authors, for example).

The answer `?` to the initial prompt gets a list of all reference types. The answer `help` gets a subprompt for a reference type whose pertinent attributes will then be listed. The answer `?` to the subprompt gets attributes for every type.

The attribute `also` permits one entry to refer to another by naming keywords for the other reference. An entire 'also' citation may be included within a `|reference` thus:

`%also_begin` *text*
%ref_fields
`%also_end`

The attribute `keywords` prompts for distinguishing keys for the current entry, in addition to those already occurring within author, title, etc.

The 'reference type' `quit` causes *penter* to exit, first appending the collected database information to *outfile* (`prefer.out` by default).

The 'attribute' `~e` permits editing of the current reference with the editor specified by environment variable `EDITOR`, *ed*(1) by default; `~v` gets the editor `VISUAL`, *vi*(1) by default.

Pinvert creates an inverted index to one or more bibliographic database *files.* The index is placed in *file*`.i`, where *file* is the first input file. An associated *file*`.h` contains the names of the input files. The options are:

`-c`*common* Do not index words listed in file *common* (default `/usr/lib/eign`).
`-i`*ignore* Do not index information about attributes listed in file *ignore.* (The default `/usr/lib/prefer/ignore` lists `%volume`, `%number`, `%part`, `%pages`, `%X` (location status), `%Y` (read status), `%Z` (comment).)
`-k`*i* Maximum number of keys kept per record (default 100).
`-l`*i* Maximum length of keywords (default 6, none is less than 3).
`-p`*file* The basename of the index is *file.* Prefer will write the index to *file*`.i`.
`-v` Verbose. Print statistics.

Plook uses the inverted index to retrieve bibliographic records by *keywords* from the command line or the standard input. Records that contain all the keywords in the request are sent to the standard output. Option `-p` is the same as for *pinvert.*

Pconvert converts a *refer*(1) database to *prefer* style. Under option `-d` it converts *refer*-style commands in a document to *prefer* style.

FILES

`prefer.out`	default database
`prefer.out.i`	default index file

`prefer.out.h`	default header file containing names of databases
`/usr/lib/eign`	default list of common words
`/usr/lib/prefer/ignore`	default list of *%ref_fields* to ignore for indexing
`/usr/lib/prefer/styles/*`	*awk* scripts of formatting instructions for each style
`/tmp/prefer*`	scratch file
`/usr/lib/prefer/ptemplate`	reference type definitions, self-describing
`/usr/lib/prefer/mypubenter`	program executed by penter

SEE ALSO

M. A. Derr, 'Formatting References with Prefer', this manual, Volume 2
refer(1), *monk*(1), *troff*(1)

BUGS

Prefer commands don't work immediately after certain formatting macros, e.g. .SM, .I, .B.
Plook complains if the first key matches more references than it can store. Try rearranging your request so a less common word comes first.
Pinvert does not record options `-c` and `-l`. If you use them with *pinvert*, you will have to supply them for *prefer* and *plook* as well.

NAME

printenv − print environment

SYNOPSIS

`printenv` [*name*]

DESCRIPTION

With no arguments, *printenv* places the strings of the environment, described in *environ*(5), on the standard output one per line.

If a *name* is specified, its value is retrieved from the environment and printed.

SEE ALSO

sh(1), *rc*(1), *environ*(5), *getenv*(3)

DIAGNOSTICS

Exit status 1 is returned when a specified *name* is not present in the environment.

BUGS

The *name* feature cannot handle functions.

NAME

prof – display profile data

SYNOPSIS

`prof` [*option* ...] [*a.out* [*mon.out* ...]]

DESCRIPTION

Prof interprets files produced by *monitor*(3) or the `-p` option of *cc* or *f77*. The symbol table in the named object file (`a.out` by default) is read and correlated with the profile file (`mon.out` by default). For each symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the time spent there and the number of times that routine was called. If more than one profile file is specified, the output represents the sum of the profiles.

Zero call counts are tallied for subroutines not compiled with option `-p`. The flag `-p` must be passed to the loader to get the profiling output written.

Options are:

`-l` Sort the output by symbol value.

`-n` Sort the output by number of calls.

`-s` Produce a summary profile file in `mon.sum`.

`-v` *-low -high*
Produce a graphic profile on the standard output for display by the *plot*(1) filters. Optional numbers *low* and *high*, by default 0 and 100, select a percentage of the profile to be plotted.

`-z` Include routines with zero usage in the output.

FILES

`mon.out`	for profile
`a.out`	for namelist
`mon.sum`	for summary profile

SEE ALSO

time(1), *lcomp*(1), *monitor*(3), *getopt*(3), *profil*(2), *plot*(1), *cc*(1), *f77*(1)

BUGS

Beware of quantization errors.
Prof is confused by *f77*, which puts the entry points at the bottom of subroutines and functions.
Option `-v` has been disabled.

NAME

ps – process status

SYNOPSIS

`ps` [`afhnlrux`[`Ft`*name*][`num`]]

DESCRIPTION

Ps prints information about processes. Normally, only the user's non-shell processes are reported.

All output formats include, for each process, the process id, its control terminal, status, cpu time, and command. The status is given by a sequence of letters:

`R`	Running.	`T`	Stopped.
`P`	Waiting for a page.	`S`	Sleeping (< 20 seconds).
`I`	Idle (> 20 seconds).	`N`	Process is 'nice'; see *nice*(1).
`W`	Swapped out.		

`R`, `T`, `P`, `S`, and `I` are mutually exclusive.

Options are given in blank-separated clusters:

`a`	Give information about all users (ordinarily only one's own processes are displayed).
`F`*file*	Display all processes having the *file* open; this option must be the last in a cluster.
`f`	Display file descriptors (with prepended process id).
`ff`	Display file descriptors (omitting process id).
`h`	Print column headers.
`l`	(long) Print the process size in kilobytes, resident set size, parent process id, and wait channel.
`n`	Print unsorted (output is normally sorted by process id).
`r`	Print processes which match the current real or effective user id (normally only processes matching the current effective user id are printed).
`t`*s*	Print processes whose control stream is *s*, where *s* is either `.` (for user's tty), or one of the forms printed by *ps*, e.g. `03` for `/dev/tty03`, `conso` for console, `dk12` for `/dev/dk/dk12`, `t?` for processes with no tty, etc. This option must be the last in a cluster.
`u`	Include a name of the effective userid and cpu share in percent; sort by usage.
`x`	Include shell and *init*(8) processes. (A stopped shell is reported without `x`.)
num	Print the designated process; must be the last option in a cluster.

Multiple `F`, `t`, and *n* options are allowed; the union of all selections is printed.

FILES

`/proc`	process images
`/dev/drum`	swap device
`/dev/kmem`	kernel memory
`/dev`	searched to find tty names
`/dev/dk`	searched to find dk names
`/etc/fstab`	searched to find local file system names

SEE ALSO

kill(1), *nice*(2), *proc*(4), *load*(1)

BUGS

Things can change while *ps* is running.

NAME

ptx − permuted index

SYNOPSIS

`ptx` [*option* ...] [*input* [*output*]]

DESCRIPTION

Ptx generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *Ptx* produces output exemplified by:

```
.xx "tail" "before" "keyword and after" "head"
```

where `.xx` may be defined as a *troff*(1) macro for user-defined formatting. The *before* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head*, at least one of which is an empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, `/` marks the spot.

The following options can be applied:

`−f` Fold upper and lower case letters for sorting.

`−t` Prepare the output for the phototypesetter; the default line length is 100 characters.

`−w` *n* Use the next argument, *n*, as the width of the output line. The default line length is 72 characters.

`−g` *n* Use the next argument, *n*, as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

`−o` *only*
Use as keywords only the words given in the *only* file.

`−i` *ignore*
Do not use as keywords any words given in the *ignore* file. If the `−i` and `−o` options are missing, use `/usr/lib/eign` as the *ignore* file.

`−b` *break*
Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.

`−r` Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx*.

FILES

`/usr/lib/eign`

BUGS

Line length counts do not account for overstriking or proportional spacing.

NAME

push, pull, npush, npull − datakit remote file copy

SYNOPSIS

`push` [`−v`] *machine file ... remotedir*

`pull` [`−v`] *machine file ... localdir*

`npush` [`−v`] *machine file ... remotedir*

`npull` [`−v`] *machine file ... localdir*

DESCRIPTION

Push and *pull* copy files between machines over Datakit. *Push* copies *files* from the local machine to the directory *remotedir* on the named *machine*. *Pull* copies *files* from the named *machine* to the directory *localdir* on the local machine. The last component of the name of a copy is the same as that of the original. If one of the *files* is a directory, a corresponding directory is created and the directory's files are copied, recursively.

Option `−v` announces each file as it is copied.

Pushing and pulling involve two programs running in different contexts on different machines. In particular, pulling to directory `.` puts files in the local current directory, but pushing to `.` puts files in the remote home directory. Shell metacharacters which are to be interpreted on the remote machine must be quoted.

Npush and *npull* behave exactly like *push* and *pull*, but use a different protocol, necessary for communicating to some other Datakit clusters.

FILES

`/usr/lib/Rpull` link to `/usr/bin/pull` for remote end of transaction
`/usr/lib/Rpush` link to `/usr/bin/push`

SEE ALSO

con(1), *cp*(1), *rcp*(1), *cu*(1), *uucp*(1)

DIAGNOSTICS

Messages marked `(remote)` are from the sister process running on the remote machine.

NAME

pwd, where – machine name and working directory

SYNOPSIS

`pwd`

`where`

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

Where prints the name of the machine and the pathname of the current directory in the form

machine ! *fullpathname*

SEE ALSO

cd in *sh*(1)

NAME

qsnap – high resolution digital film printer

SYNOPSIS

`qsnap` [`-bfmrxRXY`] [*N* ...] *file*

DESCRIPTION

Qsnap produces images on a QCR digital film printer. The input *file* should be in the form of *picfile*(5), 3-channel for color or 1-channel for black and white.

Option letters appear in one string. Certain options require a numeric argument, *N*, which follows the string as a separate argument.

`b N` Set the brightness of the image to *N*, $0 \le N \le 8$; 0 is brightest. The default brightness value is 2 in high resolution mode (4K×4K pixels), and 0 in low resolution mode (2K×2K pixels) as set by option `R`.

`f N` Correct the exposure for a given type of film. Valid film types are: 7, 8, and 0 to 5 inclusive.
0 = (default) Polaroid Type 52 (4×5 inch).
1 = linear correction table.
2 = Polaroid Type 559 color pack film (4×5 inch), 2K mode.
3 = Ektachrome 100 color film (35mm), 2K mode.
4 = Ektachrome 100 color film (35mm), 4K mode.
5 = Polaroid Type 559 color pack film (4×5 inch), 4K mode.
7 = Polaroid Type 809 color film (8×10 inch), 4K mode.
8 = Tmax-100 black&white film (35mm), 2K or 4K mode.

`m N` Set the enlargement factor. The image is enlarged with a simple box filter. Default values are *N*=3 for 2K resolution and *N*=6 for 4K.

`r` Expose the red channel of a color image only (for multiple red overlays).

`x N` Expose the image N times. (Useful if the maximum brightness value is not bright enough.)

`R N` Set the resolution, where *N* is either 2 or 4, to select low (2K×2K) or high (4K×4K) resolution, respectively. The default it to leave the resolution unchanged.

`X N` Offset the image along the x-coordinate by N pixels. The 35mm camera in 4K mode may require an X- and/or a Y-offset. The offset in each direction is multiplied by the enlargement factor.

`Y N` Offset the image along the y-coordinate by N pixels.

The imaging resolution for 35mm film is 114 pixels/mm (2895 dots/inch) in high resolution mode, and 57 pixels/mm in low resolution mode. The maximum size image that fits a 35mm negative is 3840x3072 pixels. It takes about 2 minutes to render such an image in black and white, or 6 minutes in color. Since the color film is usually less sensitive to red, it is good practice to expose the red channel of a color image twice, using option `r`. Kodak Ektachrome color film, 100 ASA, or Kodak Tmax-100 black and white film are recommended. For 100 ASA film, imaging at brightness level 2 produces the best results.

With the 4×5 inch module, the imaging resolution is 34 pixels/mm (864 dots/inch) in high resolution mode, 17 pixels/mm in low resolution. Polaroid Type 559 color film or Polaroid Type 52 black and white film is recommended.

With the 8×10 inch module, the imaging resolution is 17 pixels/mm. This module can only be used in high resolution mode. Polaroid Type 809 color film is recommended.

SEE ALSO

pico(1), *bcp*(1), *cscan*(1), *imscan*(1), *picfile*(5)

NAME

random, fortune − sample lines from a file, return cookies

SYNOPSIS

`random` [`-e`] [*n*]

`/usr/games/fortune` [*file*]

DESCRIPTION

Random reads the standard input and copies each line to the standard output with probability 1/*n*. The default value of *n* is 2.

Option `-e` writes no output and returns a random exit code in the range [0,*n*−1].

Fortune prints a one-line aphorism chosen at random. If a *file* is specified, the saying is taken from that file; otherwise it is selected from `/usr/games/lib/fortunes`.

FILES

`/usr/games/lib/fortunes`
`/usr/games/lib/fortunes.index` fast lookup table, maintained automatically

BUGS

Successive results of option `-e` are highly correlated if *random* is called more than once per second.

NAME

rc, cd, wait, whatis − command language

SYNOPSIS

`rc [ -dilxepv ] [ -c command ] [ file [ arg ... ] ]`

DESCRIPTION

Rc is the Plan 9 shell. It executes command lines read from a terminal or a file or, with the `-c` flag, from *rc's* argument list.

Command Lines

A command line is a sequence of commands, separated by ampersands or semicolons (`&` or `;`) and terminated by a newline. The commands are executed in sequence from left to right. *Rc* does not wait for a command followed by `&` to finish executing before starting the following command. Whenever a command followed by `&` is executed, its process id is assigned to the *rc* variable `$apid`. Whenever a command *not* followed by `&` exits normally, the low 8 bits of the argument to its *exit*(2) call are assigned to the *rc* variable `$status`. If it terminates abnormally, `$status` has value 1000 plus the termination status provided to wait(2).

A long command line may be continued on subsequent lines by typing a backslash (`\`) followed by a newline. This sequence is treated as though it were a blank. Backslash is not otherwise a special character.

A number-sign (`#`) and any following characters up to (but not including) the next newline are ignored, except in quotation marks.

Simple Commands

A simple command is a sequence of arguments interspersed with I/O redirections. If the first argument is the name of an *rc* function or of one of *rc's* built-in commands, it is executed by *rc*. Otherwise if the name contains a slash (`/`), it must be the pathname of the program to be executed. Names containing no slash are searched for in a list of directory names stored in `$path`. The first executable file of the given name found in a directory in `$path` is the program to be executed.

The first word of a simple command cannot be a keyword unless it is quoted or otherwise disguised. The keywords are

```
for in while if not switch fn ~ ! @
```

Arguments and Variables

A number of constructions may be used where *rc's* syntax requires an argument to appear. In many cases a construction's value will be a list of arguments rather than a single string.

The simplest kind of argument is the unquoted word: a sequence of one or more characters none of which is a blank, tab, newline or any of the following:

```
# ; & | ^ $ = ` ' { } ( ) < >
```

An unquoted word that contains any of the characters `*` `?` or `[`, is a pattern for matching against file names. The character `*` matches any sequence of characters, `?` matches any single character and `[`*class*`]` matches any character in the *class*. If the first character of *class* is `~`, the class is complemented. The *class* may also contain pairs of characters separated by `-`, standing for all characters lexically between the two. The character `/` must appear explicitly in a pattern, as must the first character of the pathname components `.` and `..`. A pattern is replaced by a list of arguments, one for each pathname matched, except that a pattern matching no names is not replaced by the empty list, but rather stands for itself. Pattern matching is done after all other operations. Thus,

```
x=/tmp echo $x^/*.c
```

matches `/tmp/*.c`, rather than matching `/*.c` and then prepending `/tmp`.

A quoted word is a sequence of characters surrounded by single quotes (`'`). A single quote is represented in a quoted word by a pair of quotes (`''`).

Each of the following is an argument.

`(`*arguments*`)`

The value of a sequence of arguments enclosed in parentheses is a list comprising the members of each element of the sequence. Argument lists have no recursive structure, although their syntax may suggest it. The following are entirely equivalent:

```
echo hi there everybody
((echo) (hi there) everybody)
```

`$`*argument*

`$`*argument* `(`*subscript*`)`

The *argument* after the `$` is the name of a variable whose value is substituted. Multiple levels of indirection are possible, but of questionable utility. Variable values are lists of strings. If *argument* is a number *n*, the value is the *n*th element of `$*`, unless `$*` doesn't have *n* elements, in which case the value is empty. If *argument* is followed by a parenthesized list of subscripts, the value substituted is a list composed of the requested elements (origin 1). The parenthesis must follow the variable name with no spaces. Assignments to variables are described below.

`$#`*argument*

The value is the number of elements in the named variable. A variable never assigned a value has zero elements.

`` `{`*command*`} ``

rc executes the *command* and reads its standard output, splitting it into a list of arguments, using characters in `$ifs` as separators. If `$ifs` is not otherwise set, its value is `' \t\n'`.

`<{`*command*`}`

`>{`*command*`}`

The *command* is executed asynchronously with its standard output or standard input connected to a pipe. The value of the argument is the name of a file referring to the other end of the pipe. This allows the construction of non-linear pipelines. For example, the following runs two commands `old` and `new` and uses `cmp` to compare their outputs

```
cmp <{old} <{new}
```

This feature does not work on systems that do not support `/dev/fd` or its equivalent, e.g. on Sunos and System V.

argument`^`*argument*

The `^` operator concatenates its two operands. If the two operands have the same number of components, they are concatenated pairwise. If not, then one operand must have one component, and the other must be non-empty, and concatenation is distributive.

Free Carets

In a most circumstances, *rc* will insert the `^` operator automatically between words that are not separated by white space. Whenever one of `$ ' `` follows a quoted or unquoted word or an unquoted word follows a quoted word with no intervening blanks or tabs, a `^` is inserted between the two. If an unquoted word immediately follows a `$` and contains a character other than an alphanumeric, underscore, or `*`, a `^` is inserted before the first such character. Thus

```
cc -$flags $stem.c
```

is equivalent to

```
cc -^$flags $stem^.c
```

I/O Redirections

The sequence `>`*file* redirects the standard output file (file descriptor 1, normally the terminal) to the named *file;* `>>`*file* appends standard output to the file. The standard input file (file descriptor 0, also normally the terminal) may be redirected from a file by the sequence `<`*file*`,` or from an inline 'here document' by the sequence `<<`*eof-marker*`.` The contents of a here document are lines of text taken from the command input stream up to a line containing nothing but the *eof-marker*, which may be either a quoted or unquoted word. If *eof-marker* is unquoted, variable names of the form `$`*word* have their values substituted from *rc's* environment. If `$`*word* is followed by a caret (`^`), the caret is deleted. If

eof-marker is quoted, no substitution occurs.

Redirections may be applied to a file-descriptor other than standard input or output by qualifying the redirection operator with a number in square brackets. For example, the diagnostic output (file descriptor 2) may be redirected by writing `cc junk.c >[2]junk`.

A file descriptor may be redirected to an already open descriptor by writing `>[`*fd0*`=`*fd1*`]` or `<[`*fd0*`=`*fd1*`]`. *Fd1* is a previously opened file descriptor, and *fd0* becomes a new copy (in the sense of *dup*(2)) of it. A file descriptor may be closed by writing `>[`*fd0*`=]` or `<[`*fd0*`=]`.

Redirections are executed from left to right. Therefore, `cc junk.c >/dev/null >[2=1]` and `cc junk.c >[2=1] >/dev/null` have different effects – the first puts standard output in `/dev/null`, and then puts diagnostic output in the same place, where the second directs diagnostic output to the terminal and sends standard output to `/dev/null`.

Compound Commands

A pair of commands separated by a pipe operator (`|`) is a command. The standard output of the left command is sent through a pipe to the standard input of the right command. The pipe operator may be decorated to use different file descriptors. `|[`*fd*`]` connects the output end of the pipe to file descriptor *fd* rather than 1. `|[`*fd0*`=`*fd1*`]` connects output to *fd0* of the left command and input to *fd1* of the right command.

A pair of commands separated by `&&` or `||` is a command. In either case, the left command is executed and its exit status examined. If the operator is `&&` the right command is executed if the left command's status is zero. `||` causes the right command to be executed if the left command's status is non-zero.

The exit status of a command may be inverted (non-zero is changed to zero, zero is changed to one) by preceding it with a `!`.

The `|` operator has highest precedence, and is left-associative (i.e. binds tighter to the left than the right.) `!` has intermediate precedence, and `&&` and `||` have the lowest precedence.

The unary `@` operator, with precedence equal to `!`, causes its operand to be executed in a subshell.

Each of the following is a command.

`if (` *list* `)` *command*
: A *list* is a sequence of commands, separated by `&`, `;` or newline. It is executed and if its exit status is zero, the *command* is executed.

`if not` *command*
: The immediately preceding command must have been `if (` *list* `)` *command*. If its condition was non-zero, the *command* is executed.

`for (` *name* `in` *arguments* `)` *command*

`for (` *name* `)` *command*
: The *command* is executed once for each *argument* with that argument assigned to *name*. If the argument list is omitted, `$*` is used.

`while (` *list* `)` *command*
: The *list* is executed repeatedly until its exit status is non-zero. Each time it returns zero status, the *command* is executed. The empty *list* always yields zero status.

`switch(`*argument*`){`*list*`}`
: The *list* is searched for simple commands beginning with the word `case`. (The search is only at the 'top level' of the *list*. That is, `case`s in nested constructs are not found.) *Argument* is matched against each word following `case` using the pattern-matching algorithm described above, except that `/` and the first characters of `.` and `..` need not be matched explicitly. When a match is found, commands in the list are executed up to the next following `case` command (at the top level) or the closing parenthesis.

`{`*list*`}`
: Braces serve to alter the grouping of commands implied by operator priorities. The *body* is a sequence of commands separated by `&`, `;` or newline.

`fn` *name* { *list* }
`fn` *name*

The first form defines a function with the given *name*. Subsequently, whenever a command whose first argument is *name* is encountered, the current value of the remainder of the command's argument list will be assigned to `$*`, after saving its current value, and *rc* will execute the *list*. The second form removes *name*'s function definition.

`fn` *signal* { *list* }
`fn` *signal*

A function with the name of a signal, in lower case, is defined in the usual way, but called when *rc* receives that signal; see *signal*(2). By default *rc* exits on receiving any signal, except when run interactively, in which case interrupts and quits normally cause *rc* to stop whatever it's doing and start reading a new command. The second form causes *rc* to handle a signal in the default manner. *Rc* recognizes an artificial signal, `sigexit`, which occurs when *rc* is about to finish executing.

name=*argument command*

Any command may be preceded by a sequence of assignments interspersed with redirections. The assignments remain in effect until the end of the command, unless the command is empty (i.e. the assignments stand alone), in which case they are effective until rescinded by later assignments.

Built-in Commands

These commands are executed internally by *rc*, usually because their execution changes or depends on *rc*'s internal state.

`.` *file ...*

Execute commands from *file*. `$*` is set for the duration to the remainder of the argument list following *file*. *File* is searched for using `$path`.

`builtin` *command ...*

Execute *command* as usual except that any function named *command* is ignored.

`cd` [*dir*]

Change the current directory to *dir*. The default argument is `$home`. *dir* is searched for in each of the directories mentioned in `$cdpath`.

`eval` [*arg ...*]

The arguments are concatenated separated by spaces into a single string, read as input to *rc*, and executed.

`exec` [*command ...*]

Rc replaces itself with the given (non-built-in) *command*.

`exit` [status]

Exit with the given exit status. If none is given, the current value of `$status` is used.

`shift` [*n*]

Delete the first *n* (default 1) elements of `$*`.

`umask` [*octal*]

Set *rc*'s file-creation mask (see *umask*(2)) to the given octal value. If no value is given, the current mask value is printed.

`wait` [*pid*]

Wait for the process with the given *pid* to exit. If no *pid* is given, all outstanding processes are waited for.

`whatis` *name ...*

Print the value of each *name* in a form suitable for input to *rc*. The output is an assignment to any variable, the definition of any function, a call to `builtin` for any built-in command, or the completed path name of any executable file.

`~` *subject pattern ...*

The *subject* is matched against each *pattern* in sequence. If it matches any pattern, `$status` is set to zero. Otherwise, `$status` is set to one. Patterns are the same as for file name matching, except that `/` and the first character of `.` and `..` need not be matched explicitly.

The *patterns* are not subjected to file name matching before the ~ command is executed, so they need not be enclosed in quotation marks.

Environment

The *environment* is a list of strings made available to executing binaries. *Rc* creates an environment entry for each variable whose value is non-empty, and for each function. The string for a variable entry has the variable's name followed by = and its value. If the value has more than one component, these are separated by ctrl-a (`'\001'`) characters. The string for a function is just the *rc* input that defines the function.

When *rc* starts executing it reads variable and function definitions from its environment.

Special Variables

The following variables are set or used by *rc*.

`$*` Set to *rc*'s argument list during initialization. Whenever a `.` command or a function is executed, the current value is saved and `$*` receives the new argument list. The saved value is restored on completion of the `.` or function.

`$apid` Whenever a process is started asynchronously with `&`, `$apid` is set to its process id.

`$home` The default directory for `cd`. Initially, if `$home` is not set and `$HOME` is, then `$home` is set to the value of `$HOME`.

`$ifs` The input field separators used in backquote substitutions. If `$ifs` is not set in *rc*'s environment, it is initialized to blank, tab and newline.

`$path` The search path used to find commands and input files for the `.` command. If not set in the environment, it is initialized by `path=(. /bin /usr/bin)`.

`$pid` Set during initialization to *rc*'s process id.

`$prompt`
When *rc* is run interactively, the first component of `$prompt` is printed before reading each command. The second component is printed whenever a newline is typed and more lines are required to complete the command. If not set in the environment, it is initialized by `prompt=('% ' ' ')`.

`$status`
Set to the low 8 bits of the *exit*(2) argument of a normally terminating binary (unless started with `&)`, or to 1000 plus the termination status on abnormal termination. `!` and ~ also change `$status`. Its value is used to control execution in `&&`, `||`, `if` and `while` commands. When *rc* exits at end-of-file of its input or on executing an `exit` command with no argument, `$status` is its exit status.

Invocation

If *rc* is started with no arguments it reads commands from standard input. Otherwise its first non-flag argument is the name of a file from which to read commands (but see `-c` below). Subsequent arguments become the initial value of `$*`. *Rc* accepts the following command-line flags.

`-c` *string*
Commands are read from *string*.

`-d` Debugging flag, causes *rc* only to catch `SIGINT`, so that `SIGQUIT` will cause it to dump core.

`-e` Exit if `$status` is non-zero after executing a simple command.

`-i` If `-i` is present, or *rc* is given no arguments and its standard input is a terminal, it runs interactively. Commands are prompted for using `$prompt` and `SIGINT` and `SIGQUIT` are caught and sloughed off.

`-l` If `-l` is given or the first character of argument zero is `-`, *rc* reads commands from `$home/.rcrc`, if it exists, before reading its normal input.

`-p` A no-op.

`-v` Echo input on file descriptor 2 as it is read.

`-x` Print each simple command before executing it.

BUGS

It's too slow and too big.

There should be away to match patterns against whole lists rather than just single strings.
Using ~ to check the value of `$status` changes `$status`.
Functions that use here documents don't work.
Environment entries for variables are kludgy for UNIX compatibility. Woe betide the imported variable whose value contains a ctrl-a.

NAME

rcp − remote file copy

SYNOPSIS

`rcp` *filename1 filename2*

`rcp` [`-r`] *filename ... directory*

DESCRIPTION

Rcp copies files across TCP/IP connections. Each *filename* or *directory* argument is either a remote file name of the form:

hostname **:** *path*

or a local file name (containing no **:** unless preceded by `/`).

If a *filename* is not a full path name, it is interpreted relative to your home directory on machine *hostname*. A *path* on a remote host may be quoted to cause metacharacters to be interpreted remotely.

Your current local user name must exist on *hostname* and allow remote command execution by *rsh*; see *con*(1).

Rcp handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form

username `@` *hostname* **:** *filename*

to use *username* rather than your current local user name as the user name on the remote host. In this usage, *hostname* may be a full internet domain name.

The option is

`-r` Copy each subtree rooted at *filename*; in this case the destination must be a directory.

FILES

```
.cshrc
.login
.profile
```

SEE ALSO

con(1), *cu*(1), *push*(1), *uucp*(1)

BUGS

There is no check against copying a file onto itself.
Certain cases where a file name is given when a directory is required are not diagnosed.

NAME

refer, lookbib, pubindex − maintain and use bibliographic references

SYNOPSIS

`refer` [*option* ...] [*file* ...]

`lookbib` [*file* ...]

`pubindex` *file* ...

DESCRIPTION

Refer is a preprocessor for *nroff* or *troff*(1) that finds and formats references. The input files (standard input default) are copied to the standard output, except for lines between `.[` and `.]` which are assumed to contain keywords and are replaced by information from the bibliographic data base. The user may avoid the search, override fields from it, or add new fields. The reference data, from whatever source, are assigned to a set of *troff* strings. Macro packages such as *ms*(6) print the finished reference text from these strings. A flag is placed in the text at the point of reference; by default the references are indicated by numbers.

The following options are available:

`-a`*r* Reverse the first *r* author names (Jones, J. A. instead of J. A. Jones). If *r* is omitted all author names are reversed.

`-b` Bare mode: do not put any flags in text (neither numbers nor labels).

`-c`*string*
Capitalize (with CAPS SMALL CAPS) the fields whose key-letters are in *string*.

`-e` Instead of leaving the references where encountered, accumulate them until a sequence of the form

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to the same source.

`-k`*x* Instead of numbering references, use labels as specified in a reference data line beginning `%`*x;* by default *x* is `L`.

`-l`*m*`,`*n* Instead of numbering references, use labels made from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either *m* or *n* is omitted the entire name or date respectively is used.

`-p` Take the next argument as a file of references to be searched. The default file is searched last.

`-n` Do not search the default file.

`-s`*keys* Sort references by fields whose key-letters are in the *keys* string; permute reference numbers in text accordingly. Implies `-e`. The key-letters in *keys* may be followed by a number to indicate how many such fields are used, with + taken as a very large number. The default is `AD` which sorts on the senior author and then date; to sort, for example, on all authors and then title use `-sA+T`.

A bibliographic reference in a `-p` file is a set of lines that contain bibliographic information fields. Empty lines separate references. Each field starts on a line beginning with `%`, followed by a key-letter, followed by a blank, and followed by the contents of the field, which continues until the next line starting with `%`. The most common key-letters and the corresponding fields are:

`A` Author name
`B` Title of book containing article referenced

`C`	City
`D`	Date
`d`	Alternate date
`E`	Editor of book containing article referenced
`G`	Government (CFSTI) order number
`I`	Issuer (publisher)
`J`	Journal
`K`	Other keywords to use in locating reference
`M`	Technical memorandum number
`N`	Issue number within volume
`O`	Other commentary to be printed at end of reference
`P`	Page numbers
`R`	Report number
`r`	Alternate report number
`T`	Title of article, book, etc.
`V`	Volume number
`X`	Commentary unused by *pubindex*

Except for `A`, each field should only be given once. Only relevant fields should be supplied. When *refer* is used with *eqn*, *neqn* or *tbl*(1), *refer* should be first, to minimize the volume of data passed through pipes.

Lookbib accepts keywords from the standard input and searches a bibliographic data base for references that contain those keywords anywhere in the title, author, journal name, etc. Matching references are printed on the standard output. Blank lines are taken as delimiters between queries.

Pubindex makes a hashed inverted index to the named bibliographic *files* for use by *refer*.

EXAMPLES

```
%T 5-by-5 Palindromic Word Squares
%A M. D. McIlroy
%J Word Ways
%V 9
%P 199-202
%D 1976
```

FILES

`/usr/dict/papers`	directory of default publication lists and indexes
`/usr/lib/refer`	directory of programs
x`.ia`, *x*`.ib`, *x*`.ic`	where *x* is the first argument to *pubindex*

SEE ALSO

M. E. Lesk, 'Some Applications of Inverted Indexes on UNIX' in AT&T Bell Laboratories, *UNIX Programmer's Manual, Volume 2*, Holt-Rinehart (1984)
troff(1), *doctype*(1), *prefer*(1)

BUGS

Refer is unmaintained; better use *prefer*(1).

NAME

rev, revpag − reverse lines or pages

SYNOPSIS

`rev` [*file* ...]

`revpag` [*option* ...] [*file* ...]

DESCRIPTION

Rev copies the standard input or the named files to the standard output, reversing the order of characters in every line.

Revpag copies the standard input or the named files to the standard output, reversing the order of the pages. (The name − means the standard input.) Options define what constitutes a ‘page’:

`−d` The input is *troff*(1) output; page breaks are encoded in it.

`−f` Append a new-page character (014) to the last input page (which is the first page on the output), if this page is not of the declared length.

`−l` *n* Set the number of lines per page in ordinary ASCII input (66 by default). A new-page character (014) is also recognized as a page break.

`−o` *list* Output only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A ‘page number’ means the ordinal position of a page in the input. A range *n*−*m* means pages *n* through *m*. In a range, a missing *m* means the beginning; a missing *n* means the end.

EXAMPLES

```
rev <webster | sort | rev >walker
```

From a standard Webster’s dictionary, produce Walker’s rhyming dictionary, which is alphabetized from right to left.

```
tail −r <forward >backward
```

Reverse the order of lines in a file; see *tail*(1).

NAME

rm − remove (unlink) files

SYNOPSIS

`rm` [`-fri`] *file* ...

DESCRIPTION

Rm removes directory entries. If an entry was the last link to a file, the file is destroyed. If an entry is a directory it is removed only if empty. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file lacks write permission and the the standard input is a terminal, a query is written to the standard output and a line is read from the standard input. If that line begins with `y` the file is deleted, otherwise the file remains. The options are

`-f` (force) Ask no questions about unwritable files and report no errors.

`-r` Recursively delete the entire contents of a directory and the directory itself.

`-i` (interactive) Ask whether to delete each file, and, under `-r`, whether to examine each directory. If the first character of the response is `y`, the answer is yes; otherwise the answer is no.

SEE ALSO

unlink(2)

DIAGNOSTICS

It is forbidden to remove the file `..` merely to avoid the antisocial consequences of inadvertently doing something like `rm -r .*`.

NAME

sed − stream editor

SYNOPSIS

`sed` [`−n`] *script* [*file ...*]

`sed` [`−e` *script*] [`−f` *sfile*] [*file ...*]

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a command script. Script options accumulate.

`−e` *script*
Script is given literally in command line.

`−f` *sfile* Script is given in file *sfile.*

`−n` Suppress the default output.

A script consists of editing commands, usually one per line. If a command ends with **;**, `{`, or `}`, the next command begins immediately thereafter. Empty commands are ignored. Commands have the form

[*address* [**,** *address*]] *function* [*argument ...*] [**;**]

In normal operation *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a `D` command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under `−n`) and deletes the pattern space.

An *address* is either a decimal number that counts input lines cumulatively across files, a `$` that addresses the last line of input, or a context address, */regular-expression/*, in the style of *ed*(1), with the added convention that `\n` matches a newline embedded in the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address. (Address `0` is never matched.)

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied to non-selected pattern spaces by use of the negation function `!` (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

A *text* argument consists of one or more lines, all but the last of which end with `\` to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an `s` command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line.

An *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 120 distinct *wfile* arguments.

(1) `a\`
text Append. Place *text* on the output before reading the next input line.

(2) `b` *label*
Branch to the **:** command bearing the *label.* If *label* is empty, branch to the end of the script.

(2) `c\`
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) `d` Delete the pattern space. Start the next cycle.

(2) `D` Delete the initial segment of the pattern space through the first newline. Start the next cycle.

(2) `g` Replace the contents of the pattern space by the contents of the hold space.

(2) `G` Append the contents of the hold space to the pattern space.

(2) `h` Replace the contents of the hold space by the contents of the pattern space.

(2) `H` Append the contents of the pattern space to the hold space.

(1) `i\`
text Insert. Place *text* on the standard output.

(2) `l` Literal. Place an unambiguous image of the pattern space on the standard output, using C escape sequences. Break long lines, indicating the breakpoint by a single backslash. Append `\n` if pattern space ends with space or newline.

(2) `n` Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) `N` Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2) `p` Print. Copy the pattern space to the standard output.

(2) `P` Copy the initial segment of the pattern space through the first newline to the standard output.

(1) `q` Quit. Branch to the end of the script. Do not start a new cycle.

(2) `r` *rfile*
Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) `s`/*regular-expression*/*replacement*/*flags*
Substitute the *replacement* string for instances of the *regular-expression* in the pattern space. Any character may be used instead of `/`. For a fuller description see *ed*(1); although unlike *ed* , the trailing `/` *must* be supplied. *Flags* is zero or more of

`g` Global. Substitute for all non-overlapping instances of the *regular expression* rather than just the first one.

`p` Print the pattern space if a replacement was made.

`w` *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) `t` *label*
Test. Branch to the `:` command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a `t`. If *label* is empty, branch to the end of the script.

(2) `w` *wfile*
Write. Append the pattern space to *wfile.*

(2) `x` Exchange the contents of the pattern and hold spaces.

(2) `y`/*string1*/*string2*/
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2) `!` *function*
Don't. Apply the *function* (or group, if *function* is `{`) only to lines *not* selected by the address(es).

(0) `#` Comment. Ignore the rest of the line.

(0) `:` *label*
This command does nothing; it bears a *label* for `b` and `t` commands to branch to.

(1)= Place the current line number on the standard output as a line.

(2){ Execute the following commands through a matching } only when the pattern space is selected.

(0) An empty command is ignored.

EXAMPLES

`sed 10q file`
Print the first 10 lines of the file.

`sed '/^$/d'`
Delete empty lines from standard input.

`sed 's/UNIX/& system/g'`
Replace every instance of `UNIX` by `UNIX system`.

```
sed 's/ *$//        drop trailing blanks
/^$/d               drop empty lines
s/  */\             replace blanks by newlines
/g
/^$/d' chapter*
```
Print the files `chapter1`, `chapter2`, etc. one word to a line.

```
nroff -ms manuscript | sed '
${
       /^$/p        if last line of file is empty, print it
}
//N                 if current line is empty, append next line
/^\n$/D'            if two lines are empty, delete the first
```
Delete all but one of each group of empty lines from a formatted manuscript.

```
ls /usr/* | sed '
/^$/d               delete empty lines
/^[/].*:$/{         look for lines like /usr/lem:
       s/:$/\//     replace : by /
       h            hold directory name
       d            don't print; get next line
}
G                   append held directory name
s/\(.*\)\n\(.*\)/\2\1/'  exchange file and directory
```
List all files in user directories, as `ls -d /usr/*/*` would do if it didn't cause argument list overflow.

SEE ALSO

ed(1), *gre*(1), *awk*(1), *lex*(1), *cut*(1), *split*(1), *sam*(9.1)
L. E. McMahon, 'SED — A Non-interactive Text Editor', this manual, Volume 2.

BUGS

If input is from a pipe, buffering may consume characters beyond a line on which a `q` command is executed.

NAME

seq − print sequences of numbers

SYNOPSIS

`seq` [`−w`] [`−f`*format*] [*first* [*incr*]] *last*

DESCRIPTION

Seq prints a sequence of numbers, one per line, from *first* (default 1) to as near *last* as possible, in increments of *incr* (default 1). The numbers are interpreted as floating point.

Normally integer values are printed as decimal integers. The options are

`−f`*format*
: Use the *printf*(3)-style *format* for printing each (floating point) number. The default is `%g`.

`−w`
: Equalize the widths of all numbers by padding with leading zeros as necessary. Not effective with option `−f`, nor with numbers in exponential notation.

EXAMPLES

`seq 0 .05 .1`
: Print `0 0.05 0.1` (on separate lines).

`seq −w 0 .05 .1`
: Print `0.00 0.05 0.10`.

BUGS

Option `−w` always surveys every value in advance, although that's not necessary for integers. Thus `seq −w 1000000000` is a hopeless way to get an 'infinite' sequence.

NAME

sh, cd, wait, whatis − shell, the standard command programming language

SYNOPSIS

`sh` [`-acefiknpstuvx`] [*args*]

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. See 'Invocation' below for the meaning of arguments to the shell.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters `*`, `@`, `#`, `?`, `-`, `$`, and `!`. A *word* is a sequence of characters and quoted strings set off by operators, blanks, or newlines; see 'Quoting'.

Commands

A *simple-command* is a sequence of *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0; see *exec*(2). The *value* of a simple-command is its exit status if it terminates normally, or `0200`+status if it terminates abnormally; see *signal*(2) for a list of status values.

A *pipeline* is a sequence of one or more *commands* separated by `|`. If there is more than one command, each is run in a subshell; `|` denotes a *pipe*(2) connecting the standard output of one command to the standard input of the next. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by `;`, `&`, `&&`, or `||`, and terminated by `;` or `&`. Of these four symbols, `;` and `&` have equal precedence, which is lower than that of `&&` and `||`. The symbols `&&` and `||` also have equal precedence. A semicolon (`;`) causes sequential execution of the preceding pipeline; an ampersand (`&`) causes asynchronous execution of the pipeline; the shell does not wait and proceeds as if the pipeline had returned zero exit status. The symbol `&&` (`||`) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. One or more newlines may follow any sequencing operator (`;` `&` `&&` `||`).

One or more newlines may always be used in place of a single semicolon, and newlines may be freely inserted after any of `|` `;` `&` `&&` `||` `;;` `if` `do` `then` `elif` `else` `fi` `done` `while` `until`.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

`for` *name* [`in` *word* ... `;`] `do` *list* `;` `done`
: A `for` command executes a *list* of commands once for each *word*, with *name* set to each *word* in turn. If `in` *word* ... `;` is omitted or replaced by newlines, then the *list* is executed once for each positional parameter that is set; see 'Parameter Substitution'.

`case` *word* `in` [*pattern* [`|` *pattern*] ... `)` *list* `;;`] ... `esac`
: A `case` command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see 'File Name Generation') except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly. Newlines may precede each *pattern* and replace the last `;;` before `esac`.

`if` *list* `then` *list* [`elif` *list* `then` *list*] ... [`else` *list*] `fi`
: The *list* following `if` is executed and, if it returns a zero exit status, the *list* following the first `then` is executed. Otherwise, the *lists* in `elif` clauses are executed in turn until one returns zero status; then the *list* following the next `then` is executed. Otherwise, the `else` *list* is executed. If no `else` *list* or `then` *list* is executed, then the `if` command returns a zero exit status.

`while` *list* `do` *list* `done`
: A `while` command repeatedly executes the `while` *list* and, if the exit status of the last

command in the list is zero, executes the `do` *list*; otherwise the loop terminates. If no commands in the `do` *list* are executed, then the `while` command returns a zero exit status; `until` may be used in place of `while` to negate the loop termination test.

(*list*)
: Execute *list* in a sub-shell.

{*list*}
: *list* is simply executed.

name `()` *command*
: Define a function which is referenced by *name.* The body of the function is the *command*. The most useful form of *command* is a sequence of commands enclosed in braces `{ }`. Execution of functions is described under *Execution* below.

These words are only recognized as the first word of a command and when not quoted: `if then else elif fi case esac for while until do done`.

Comments

A word beginning with `#` causes that word and all the following characters up to a newline to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents `` ` ` `` may be used as part or all of a word; trailing newlines are removed.

Parameter Substitution

The character `$` is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by `set`. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter*}
: The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is `*` or `@`, all the positional parameters, starting with `$1`, are substituted (separated by spaces). Parameter `$0` is set from argument zero when the shell is invoked.

${*parameter*`:-`*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter*`:=`*word*}
: If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter*`:?`*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message ''parameter null or not set'' is printed.

${*parameter*`:+`*word*}
: If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, `pwd` is executed only if `d` is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (`:`) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

`#` The number of positional parameters in decimal.
`-` Flags supplied to the shell on invocation or by the `set` command.
`?` The decimal value returned by the last synchronously executed command; see *exit*(2).
`$` The process number of this shell.
`!` The process number of the last background command invoked.

The following parameters are used by the shell:

`HOME` The default argument (home directory) for the *cd* command.
`PATH` The search path for commands; see 'Execution'.
`CDPATH` The search path for the *cd* command.
`MAIL` If this parameter is set to the name of a mail file the shell informs the user of the arrival of mail in the specified file. The file is inspected every three minutes.
`HISTORY`
If this parameter is set to the name of a writable file, the shell appends interactive input to the file, for use by the command =(1).
`PS1` Primary prompt string, by default `$`.
`PS2` Secondary prompt string, by default >.
`IFS` Internal field separators, normally space, tab, and newline.

The shell gives default values to `PATH`, `PS1`, `PS2` and `IFS`. `HOME` is set by *login*(8).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in `IFS`) and split into distinct arguments where such characters are found. Explicit null arguments (`""` or `''`) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation

Following substitution, each command *word* is scanned for the characters `*`, `?`, and `[`. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The directories `.` and `..` (initially or after a `/`) are only matched by patterns beginning with an explicit period. The character `/` itself must be matched explicitly.

`*` Matches any string, including the null string.
`?` Matches any single character.
`[...]` Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening `[` is a `^` any character not enclosed is matched.

Quoting

These characters have a special meaning to the shell and terminate a word unless quoted:

`;` `&` `(` `)` `|` `<` `>` `{` `}` newline space tab

(The characters `{` and `}` need not be quoted inside a `${}` construction.) A character may be *quoted* (i.e., made to stand for itself) by preceding it with a `\`. The pair \newline is ignored. All characters enclosed between a pair of single quote marks `''` (except a single quote) are quoted. Inside double quote marks `""` parameter and command substitution occurs and `\` quotes the characters `\`, `` ` ``, `"`, and `$`. `"$*"` is equivalent to `"$1 $2 ..."`, whereas `"$@"` is equivalent to `"$1" "$2" ...`.

Prompting

When used interactively, the shell prompts with the value of `PS1` before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of `PS2`) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or

follow a *command* and are not passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<*word*	Use file *word* as standard input (file descriptor 0).
>*word*	Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.
>>*word*	Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
<<*word*	The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) \newline is ignored, and \ must be used to quote the characters \, $, `, and the first character of *word*.
<&*digit*	Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using >&*digit*.
<&−	The standard input is closed. Similarly for the standard output using >&−.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

```
... 1>xxx 2>&1
```

first associates file descriptor 1 with file `xxx`, then associates file descriptor 2 with the same file as descriptor 1, namely `xxx`, while

```
... 2>&1 1>xxx
```

associates file descriptor 2 with the current value of file descriptor 1 (typically the terminal) and file descriptor 1 with `xxx`.

If a command is followed by `&`, the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The *environment* is a list of strings, conventionally function definitions and name-value pairs, that is passed to an executed program in the same way as a normal argument list; see *environ*(5). The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter or function for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the `export` command is used to bind the shell's parameter to the environment; see also `set −a`. A parameter may be removed from the environment with the `unset` command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by `unset`, plus any modifications or additions, all of which must be noted in `export` commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters (but not functions). Thus `tabs` gets the same environment in both lines below, but the shell has one less variable in the second.

```
(export TERM; TERM=450; tabs)
TERM=450 tabs
```

If the `−k` flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name.

Signals

SIGINT and SIGQUIT (see *signal*(2)) for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent (but see also the trap command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches the name of a defined function, the function is executed in the shell process. (Note how this differs from calling a shell script.) The positional parameters $1, $2, ... are set to the arguments of the function. If the command name does not match a function, but matches one of the builtin commands listed below, it is executed in the shell process. If the command name matches neither a builtin command nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter PATH defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is :/bin:/usr/bin (specifying the current directory, /bin, and /usr/bin, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a / the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not executable by *exec*(2), it is assumed to be a 'shell script', a file of shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

Builtin Commands

Input/output redirection is permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by PATH is used to find the directory containing *file*.

builtin [*command*]
Execute the builtin *command* (such as break) regardless of functions defined with the same name.

break [*n*]
Exit from the enclosing for or while loop, if any. If *n* is specified break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing for or while loop. If *n* is specified resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter HOME is the default *arg*. The shell parameter CDPATH defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The current directory (default) is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*.

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The non-builtin command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]
The given *names* are marked for automatic export to the *environment* of subsequently-executed

commands. If no arguments are given, a list of all names that are exported in this shell is printed.

`read` [*name* ...]
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

`return` [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

`set` [`--aehknptuvx` [*arg* ...]]
`-a` Mark variables which are modified or created for export.
`-e` Exit immediately if a command exits with a non-zero exit status.
`-f` Disable file name generation
`-k` All keyword arguments are placed in the environment for a command, not just those that precede the command name.
`-n` Read commands but do not execute them.
`-p` Remove the definitions for all functions imported from the environment, and set `IFS` to blank, tab and newline.
`-t` Exit after reading and executing one command.
`-u` Treat unset variables as an error when substituting.
`-v` Print shell input lines as they are read.
`-x` Print commands and their arguments as they are executed.
`--` Do not change any of the flags; useful in setting `$1` to `-`.
Using + rather than − causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in `$-`. The remaining arguments are positional parameters and are assigned, in order, to `$1`, `$2`, ... If no arguments are given the values of all names are printed.

`shift` [*n*]
The positional parameters from `$`*n*+1 ... are renamed `$1` ... If *n* is not given, it is assumed to be 1.

`times`
Print the accumulated user and system times for processes run from the shell.

`trap` [*arg*] [*n*] ...
The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. If *arg* is absent all traps *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The `trap` command with no arguments prints a list of commands associated with each signal number.

`umask` [*nnn*]
The user file-creation mask is set to *nnn*; see *umask*(2). If *nnn* is omitted, the current value of the mask is printed.

`unset` [*name* ...]
For each *name*, remove the corresponding variable or function. The variables `PATH`, `PS1`, `PS2` and `IFS` cannot be unset.

`wait` [*n*]
Wait for the specified process and report its termination status. If *n* is not given all currently active child processes are waited for and the return code is zero.

`whatis` [*name* ...]
For each *name*, print the associated value as a parameter, function, builtin or executable file as appropriate. In each case, the value is printed in a form that would yield the same value if typed as input to the shell itself: parameters are printed as assignments, functions as their

definitions, builtins as calls to `builtin`, and executable files as completed pathnames.

Invocation

Normally the shell reads commands from the file named in its first argument (standard input default). The remaining arguments are interpreted as position parameters; see 'Parameter substitution' above. If the shell is invoked through *exec*(2) and the first character of argument zero is −, commands are read first from `$HOME/.profile`, if it exists. Certain options modify this behavior:

`-c` *string* Read commands from *string*; ignore remaining arguments.
`-s` Write shell output (except for builtin commands) on file descriptor 2.
`-i` Interactive. Ignore signal `SIGTERM` (interactive shell is immune to `kill 0`). Catch and ignore `SIGINT` (`wait` is interruptible). The shell always ignores `SIGQUIT`.

Other options are described under the `set` command above.

FILES

```
$HOME/.profile
/tmp/sh*
/dev/null
```

SEE ALSO

=(1), *echo*(1), *newgrp*(1), *test*(1), *dup*(2), *exec*(2), *fork*(2), *pipe*(2), *signal*(2), *umask*(2), *exit*(2), *environ*(5)

B. W. Kernighan and R. Pike, *The Unix Programming Environment*, Prentice-Hall, 1984

DIAGNOSTICS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed; see also the `exit` command.

BUGS

Errors arising from builtins terminate shell scripts.

NAME

size − size of an object file

SYNOPSIS

`size` [*object* ...]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in hex and decimal, of each *object* file argument. If no file is specified, `a.out` is used.

SEE ALSO

a.out(5), *nm*(1)

NAME

sleep − suspend execution for an interval

SYNOPSIS

`sleep` *time*

DESCRIPTION

Sleep suspends execution for *time* seconds.

EXAMPLES

```
(sleep 100; command)&
```

Execute a command 100 seconds hence.

```
while :
do
        command
        sleep 30
done
```

Repeat a command every 30 seconds.

SEE ALSO

alarm(2), *sleep*(3)

NAME

sml – Standard ML compiler

SYNOPSIS

`sml` [*arg* ...]

DESCRIPTION

Sml is the Standard ML of New Jersey compiler. It reads declarations and expressions incrementally from standard input, compiles and evaluates them, and places results on the standard output. Some useful system-related facilities are

`System.argv : unit -> string list`
Return the argument list with which *sml* was invoked.

`System.environ : unit -> string list`
Return the environment list with which *sml* was invoked.

`use : string -> unit`
Temporarily take *sml* source from the file named in the argument.

`exportML : string->bool`
Save the current memory image as the named file, which may later be executed as an argumentless UNIX command. Return `true` in the original and `false` upon resumption of the saved image.

Save a function an executable file and quit ML. The function
takes a UNIX argument list and environment as input; see *exec*(2).

`system : string -> unit`
Invoke a shell command.

`cd : string -> unit`
Change working directory.

`System.Control.primaryPrompt : string ref`
`System.Control.secondaryPrompt : string ref`
Primary and secondary prompts analogous to `PS1` and `PS2` of *sh*(1).

`System.Control.Print.printDepth : int ref`
Limit on depth of printing complex objects; default 5.

`System.Control.Print.stringDepth : int ref`
Limit on length to which strings will be printed; default 70.

`System.Control.Print.signatures : bool ref`
Print signatures only if true.

EXAMPLES

```
fun timeit (f: unit->'a) =   (* use the system timer *)
     let open System.Timer
         val start = start_timer()
         val result = f()
      in print(makestring(check_timer(start)));
         print "\n";
         result
     end;
```

SEE ALSO

Robert Harper, 'Introduction to Standard ML', Edinburgh University report ECS-LFSC-86-14 (1986)
Robert Harper, Robin Milner, and Mads Tofte, *The Definition of Standard ML*, MIT Press (1990)

NAME

snocone − snobol with syntactic sugar

SYNOPSIS

`snocone` *file* ...

DESCRIPTION

Snocone is a programming language, syntactically similar to C, that compiles into SNOBOL4. The Snocone compiler translates the concatenation of all the input files into a SNOBOL4 program, which it writes in `a.out`. When `a.out` is executed, the SNOBOL4 interpreter will automatically be invoked. A synopsis of Snocone syntax follows.

Lexical conventions

Everything after the first unquoted `#` on an input line is ignored.

Statements normally end at the end of the line. If the last character on a line is an operator, open parenthesis or bracket, or comma, the statement is continued on the next line.

Binary operators, grouped by decreasing precedence

`[]` Array and table indexing (denoted in SNOBOL4 by <>).

`$ .` conditional and immediate pattern value assignment, as in SNOBOL4

`^` power; right-associative as in SNOBOL4

`* / %` multiplication, division, remainder; unlike SNOBOL4, all have the same precedence.

`+ −` addition, subtraction

`< > <= >= == != :<: :>: :<=: :>=: :==: :!=:`
comparison operators; the ones surrounded by colons compare strings, the others compare numbers. These operators behave as SNOBOL4 predicates: they return the null string if the condition is true, and fail if it is false.

`&&` concatenation; evaluates its right operand only after its left operand has been successfully evaluated. It therefore acts as logical *and* when applied to predicates. The null string may be concatenated to any value.

`||` the value of the left operand if possible, otherwise the value of the right operand.

`|` pattern value alternation.

`?` pattern match. Returns the part of the left operand matched by the right operand, which must be a pattern. May be used on the left of an assignment if the left operand is appropriate. Right-associative.

`=` assignment

Unary operators

`+` The numeric equivalent of its argument.

`−` The numeric equivalent of its argument, with the sign reversed.

`*` Unevaluated expression, as in SNOBOL4.

`$` If *v* is a value of type `name`, then `$`*v* is the variable of that name.

`@` Pattern matching cursor assignment.

`~` Logical negation: returns the null string if its argument fails, and fails otherwise.

`?` Returns the null string if its argument succeeds, and fails otherwise.

`.` Returns a value of type `name` that refers to its (lvalue) argument.

Statements

Statements may be prefixed by one or more labels. A label is an identifier followed by a colon, as in C. All labels are global: it is a good idea to prefix labels in procedures with the name of the procedure.

expression
The given *expression* is evaluated for its side effects.

`{` *statement* ... `}`
The *statements* are executed sequentially.

`if (`*expression*`)` *statement* [`else` *statement*]
If evaluation of the *expression* succeeds, the first *statement* is executed. Otherwise, the second

statement, if any, is executed. An *else* belongs to the closest unmatched `if`.

`while (`*expression*`)` *statement*
The *statement* is executed repeatedly, as long as the *expression* can be successfully evaluated.

`do` *statement* `while (`*expression*`)`
Like the `while` statement, except that the *statement* is executed once before the first time the *expression* is evaluated.

`for (`*e1*`,` *e2*`,` *e3*`)` *statement*
As in C, except that commas are used instead of semicolons.

`return` [*expression*]
returns the value of the *expression* from the current function. If *expression* fails or is missing, the value returned is that of the variable with the same name as the function. If that variable was never set, the function returns the null string.

`nreturn` [*expression*]
The *expression* must be the name of a variable. That variable is returned from the current function as an lvalue. If the *expression* fails or is missing, the variable with the same name as the function must have been set to the name of a variable.

`freturn`
The current function returns failure.

`goto` *label*
Transfer control to the given *label.*

Procedures may not be textually nested, but may be recursive and may call each other in forward references. The general form of a procedure declaration is:

`procedure` *name* `(`*args*`)` *locals* `{` *statement ...* `}`

The *args* and *locals* are lists of variable names, separated by commas. Since Snocone is a dynamically typed language, further declarations are not necessary. Although procedures are not textually nested, names are dynamically scoped: a procedure can reference the local variables and parameters of its caller as if they were global variables.

Assigning a (string) value to the variable `output` causes that value to be written as a single line on the standard output. Accessing the variable `input` causes a line to be read from the standard input. The access fails at end of file. Accessing or assigning to the variable `terminal` causes a line to be read from or written to the standard error file. Other input-output is as implemented by the Macrospitbol interpreter; see *langs*(1).

SEE ALSO

A. R. Koenig, ‘The Snocone Programming Language’, this manual, Volume 2
langs(1)

BUGS

Run-time diagnostics refer to SNOBOL4 source statement numbers, not to Snocone line numbers.
Extremely long statements can overflow the SNOBOL4 compiler’s limits on input line length.

NAME

sort − sort and/or merge files

SYNOPSIS

`sort` [−cmusMbdfinrt*x*] [−o *output*] [*option* ...] [*file* ...]

DESCRIPTION

Sort sorts lines of all the *files* together and writes the result on the standard output. The name − means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

−b Ignore leading white space (spaces and tabs) in field comparisons.

−d ‘Phone directory’ order: only letters, digits and white space are significant in string comparisons.

−f Fold lower case letters onto upper case.

−i Ignore characters outside the ASCII range 040-0176 in string comparisons.

−n An initial numeric string, consisting of optional white space, optional sign, and a nonempty string of digits with optional decimal point, is sorted by value.

−g Numeric, like −n, with e-style exponents allowed.

−M Compare as month names. The first three characters after optional white space are folded to lower case and compared. Invalid fields compare low to jan.

−r Reverse the sense of comparisons.

−t*x* ‘Tab character’ separating fields is *x*.

−k *pos1*, *pos2*
Restrict the sort key to a string beginning at *pos1* and ending at *pos2*. *Pos1* and *pos2* each have the form *m*.*n*, optionally followed by one or more of the flags Mbdfginr; *m* counts fields from the beginning of the line and *n* counts characters from the beginning of the field. If any flags are present they override all the global ordering options for this key. If .*n* is missing from *pos1*, it is taken to be 1; if missing from *pos2*, it is taken to be the end of the field. If *pos2* is missing, it is taken to be end of line.

−a *pos1*, *pos2*
Pos1, *pos2* designates an accumulating −n-style field, disjoint from the sort key. When the keys of two records compare equal, discard the second and replace the accumulating field in the first with the sum of that field in both. The receiving field must be large enough for the sum to fit without destroying any field separator. The sum has a ones digit always, a decimal point if either addend has one, a fraction part as long as that of either addend, leading zeros padded to full field width if either addend has a leading zero before the ones digit, and a + sign if either addend has a + sign and the sum is not negative.

Under option −t*x* fields are strings separated by *x*; otherwise fields are non-empty strings separated by white space. White space before a field is part of the field, except under option −b. A b flag may be attached independently to *pos1* and *pos2*.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Except under option −s, lines with all keys equal are ordered with all bytes significant.

Single-letter options may be combined into a single string, such as −cnrt:. The option combination −di and the combination of −n with any of −diM are improper.

These option arguments are also understood:

−c Check that the single input file is sorted according to the ordering rules; give no output unless the file is out of sort.

`-m` Merge; the input files are already sorted.

`-u` Unique. Keep only the first of two lines that compare equal on all keys. Implies `-s`.

`-s` Stable sort. When all keys compare equal, preserve input order. Unaffected by `-r`.

`-o` *output*
Place output in a designated file instead of on the standard output. This file may be the same as one of the inputs. The option may appear among the *file* arguments, except after `--`.

`+`*pos1* `-`*pos2*
Classical alternative to `-k`, with counting from 0 instead of 1, and *pos2* designating next-after-last instead of last character of the key. A missing character count in *pos2* means 0, which in turn excludes any `-t` tab character from the end of the key. Thus `+1 -1.3` means the same as `-k 2,2.3` and `+1r -3` means the same as `-k 2r,3`.

Options `-a`, `-g`, `-M`, and `-s` are not in the Posix standard, nor are the following tuning options.

`-T` *tempdir*
Put temporary files in *tempdir* rather than in `/usr/tmp`.

`-y` *n* Use up to *n* bytes of internal store, or a huge number if *n*=0.

`-w n` Merge up to *n* files at a time.

EXAMPLES

`sort -u -k1f -k1 list`
Print in alphabetical order all the unique spellings in a list of words where capitalized words differ from uncapitalized.

`sort -t: -k3n /etc/passwd`
Print the password file (*passwd*(5)) sorted by userid (the third colon-separated field).

`sort -umM dates`
Print the first instance of each month in an already sorted file.

`sort -k1,1 -a2,2 items_and_costs`
Reduce a file of items (field 1) and costs (field 2) to a summary file of total cost per type of item.

FILES

`/usr/tmp/stm???`

SEE ALSO

comm(1), *join*(1), *uniq*(1), *look*(1)

DIAGNOSTICS

Sort comments and exits with non-zero status for various trouble conditions and for disorder discovered under option `-c`. Overflow in a `-a` field warns and leaves the record uncombined.

BUGS

When `-o` overwrites an input file, premature termination by interrupt, crash or file-system overflow can destroy data.
Overflow in any but the first of multiple `-a` fields is fatal.
Perhaps fields delimited by `-t` should grow to avoid `-a` overflows.

NAME

spell − find spelling errors

SYNOPSIS

`spell` [*option*] [*file*]

DESCRIPTION

Spell looks up words from the named *files* (standard input default) in a public spelling list and in a private list. Possible misspellings (words that occur in neither and are not plausibly derivable from the former) are placed on the standard output.

Spell ignores constructs of *troff*(1) and its standard preprocessors, or constructs of *tex*(1). The options are

`−b` Check British spelling.

`−v` Print all words not literally in the spelling list, with derivations.

`−x` Print on standard error, marked with =, every stem as it is looked up in the spelling list, along with its affix classes. Used for maintenance.

`−c`
`−C` Input must come from standard input, one word per line. Output is a single byte per word, delivered immediately: − if the word is rejected, + if the word is accepted under `−c`, and a digit if the word is accepted under `−C`. Digit zero indicates a word known directly; larger numbers indicate words derived by more elaborate paths. Used by other programs piping queries to spell.

`−n` Print line number (counted from 1) and character number (counted from 0) with each spelling error. Used by WYSIWYG tools.

The private list, by default `$HOME/lib/spelldict`, is arranged one word per line.

Pertinent files may be specified by environment variables, listed below with their default settings. To help in gathering local vocabularies, copies of all output are accumulated in the history file, if it exists and is writable.

Spell rarely admits multiple spellings of the same word. Variants that follow general rules are preferred over over those that don't, even when the unruly spelling is more common. Thus, in American usage, 'modeled', 'sizable', and 'judgement' are preferred to 'modelled', 'sizeable', and 'judgment'. Agglutinated variants are shunned: 'crew member' and 'back yard' (noun) or 'back-yard' (adjective) are preferred to 'crewmember' and 'backyard'.

FILES

`/usr/lib/spell/amspell`	spelling list, compressed (`D_SPELL`)
`/usr/lib/spell/brspell`	British spelling list
`/usr/lib/spell/spellhist`	history file (`H_SPELL`)
`$HOME/lib/spelldict`	private list (`A_SPELL`)
`/usr/lib/spell/sprog`	the main routine (`P_SPELL`)
`deroff` (or `delatex`)	(or `delatex`) for removing punctuation and *troff*(1) constructs (`DEROFF`)

SEE ALSO

dict(7), *deroff*(1), *wwb*(1)

BUGS

Words in a private list are recognized only by exact match, including capitalization and affixing.
The heuristics of *deroff*(1) and *delatex*, used to excise formatting information, are imperfect.
The spelling list's coverage is uneven; in particular biology, medicine, and chemistry, and perforce proper names, are covered very lightly.
British spelling was done by an American.

NAME

spin – protocol analysis software

SYNOPSIS

`spin` [`-n`*N*] [`-pglprsm`] [`-at`] [*file*]

DESCRIPTION

Spin is a tool for analyzing the logical consistency of concurrent systems, specifically communication protocols. The system is specified in a guarded command language called Promela. The language, described in the reference, allows for the dynamic creation of processes, nondeterministic case selection, loops, gotos, variables and assertions. The tool has fast and frugal algorithms for analyzing liveness and safeness conditions.

Given a model system specified in Promela, *spin* can either perform random simulations of the system's execution or it can generate a C program that performs a fast exhaustive validation of the system state space. The validator can check, for instance, if user specified system invariants may be violated during a protocol's execution, or if any non-progress execution cycles exist.

Without any options the program performs a random simulation. With option

`-n`*N* the seed for the simulation is set explicitly to the integer value `N`.

The second group of options `-pglrs` is used to set the desired level of information that the user wants about the simulation run. Every line of output normally contains a reference to the source line in the specification that caused it.

`p` Show at each time step which process changed state.

`l` In combination with option `p`, show the current value of local variables of the process.

`g` Show at each time step the current value of global variables.

`r` Show all message-receive events, giving the name and number of the receiving process and the corresponding the source line number. For each message parameter, show the message type and the message channel number and name.

`s` Show all message-send events.

`m` Changes the semantics of send events. Ordinarily, a send action will be delayed if the target message buffer if full. With this option a message sent to a full buffer is lost. The option can be combined with `-a` (see below).

`a` Generate a protocol-specific analyzer. The output is written into a set of C files, named `pan.[cbhmt]`, that can be compiled (`cc pan.c`) to produce an executable analyzer. Large systems, that require more memory than available on the target machine, can still be analyzed by compiling the analyzer with a bit state space:

```
cc -DBITSTATE pan.c
```

This collapses the state space to 1 bit per system state, with minimal side-effects.

A compiled analyzer has its own set of options, which can be seen by typing `a.out -?`.

`t` If the analyzer finds a violation of an assertion, a deadlock, a non-progress loop, or an unspecified reception, it writes an error trail into a file named `pan.trail`. The trail can be inspected in detail by invoking *spin* with the `t` option. In combination with the options `pglrs` different views of the error sequence are then easily obtained.

SEE ALSO

cospan in *langs*(1)

G.J. Holzmann, 'Spin — A Protocol Analyzer', this manual, Volume 2.

NAME

split, fsplit − split a file into pieces

SYNOPSIS

`split` [*option* ...] [*file*]

`fsplit` [*option* ...] [*file* ...]

DESCRIPTION

Split reads *file* (standard input by default) and writes it in pieces of 1000 lines per output file. The names of the output files are `xaa`, `xab`, and so on to `xzz`. The options are

−*n* Split into *n*-line pieces.

−`e` *expression*
File divisions occur at each line that matches a *grep*-style regular *expression*; see *gre*(1). Multiple −`e` options may appear. If a subexpression of *expression* is contained in escaped parentheses `\(...\)`, the output file name is the portion of the line which matches the subexpression.

−`f` *stem*
Use *stem* instead of `x` in output file names.

−`s` *suffix*
Append *suffix* to names identified under −`e`.

−`x` Exclude the matched input line from the output file.

−`i` Ignore case in option −`e`; force output file names (excluding the suffix) to lower case.

Fsplit splits a collection of Fortran subprograms in one *file* into separate files. The options are

−`f`
−`e`
−`r` Set the file suffix: procedure `proc` will go into file `proc.f` (default), `proc.e`, or `proc.r` accordingly. Block data subprograms will go into files named `BLOCKDATA1.f`, etc.

−`i` Force output file names to lower case.

−`s` Strip off data beyond column 72 together with any resulting trailing blanks.

SEE ALSO

sed(1), *awk*(1) *grep* in *gre*(1)

NAME

strings − find printable strings in a file

SYNOPSIS

`strings` [*option* ...] [*file* ...

DESCRIPTION

Strings looks for and prints ASCII strings in *files.* A string is a sequence of printing characters, tabs, or backspaces terminated by a newline or a null character. In object files, strings are normally looked for only in the text and data segments. The options are:

`−t` Look for strings in the text segment of an object file.

`−d` Look for strings in the data segment of an object file.

`−s` Look for symbol strings in the symbol table of an object file.

`−a` Look for strings throughout the file.

`−o` Precede each string by its octal offset in the file.

−*number*
Ignore strings less than *number* characters long (excluding newlines). Default length is 4.

Strings is useful for identifying random object files and many other things.

SEE ALSO

gre(1), *xd*(1)

BUGS

Newlines are quietly inserted in very long strings.

NAME

strip − remove symbols and relocation bits

SYNOPSIS

`strip` [`-s`] [`-g`] [`-v`] *file* ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This saves space and hampers debuggers. Option `-s` of *ld*(1) does the same thing. The options are

`-s` Squeeze the symbol table by removing duplicate information.

`-g` Delete line-number information, thus negating the effect of `cc -g`. Implies `-s`.

`-v` Print size information.

FILES

`shrink?` temporary file

SEE ALSO

cc(1), *ld*(1)

NAME

stty − set terminal options

SYNOPSIS

`stty` [*option* ...]

DESCRIPTION

Stty sets certain I/O options for the terminal open on `/dev/tty` (file descriptor 3). With no argument, it reports the current settings of the options. The options are:

`even` Allow even parity.
`-even` Disallow even parity.
`odd` Allow odd parity.
`-odd` Disallow odd parity.
`raw` Raw mode input: no erase, kill, interrupt, quit, EOT; parity bit passed to processes.
`-raw` Turn off raw mode.
`8bit` Eight-bit mode: don't strip parity in the device driver.
`-8bit` Turn off eight-bit mode.
`cooked` Same as `-raw`.
`cbreak` Make each character available to *read*(2) as received; no erase and kill.
`-cbreak` Make characters available to *read* only when newline is received.
`-nl` Allow carriage return for new-line, and output CR-LF for carriage return or new-line.
`nl` Accept only new-line to end lines.
`echo` Echo back every character typed.
`-echo` Turn off echo.
`lcase` Map upper case to lower case.
`-lcase` Preserve case.
`-tabs` Replace tabs by spaces when printing.
`tabs` Preserve tabs.
`ek` Reset erase and kill characters to traditional backspace `@`.
`erase` *c* Set erase character to *c* (initially backspace). In this and other character-setting options a `^` may precede *c* to signify control-*c*.
`kill` *c* Set kill character to *c* (initially `@`).
`intr` *c* Set interrupt character to *c* (initially DEL).
`quit` *c* Set quit character to *c* (initially control-`\`).
`stop` *c* Set stop character to *c* (initially control-`S`).
`start` *c* Set start character to *c* (initially control-`Q`).
`eof` *c* Set 'end of file' character to *c* (initially control-`D`).
`brk` *c* Set 'line-break' character to *c* (initially undefined).
`cr0 cr1 cr2 cr3`
Select style of delay for carriage return; see *ioctl*(2).
`nl0 nl1 nl2 nl3`
Select style of delay for linefeed.
`tab0 tab1 tab2 tab3`
Select style of delay for tab.
`ff0 ff1` Select style of delay for form feed.
`bs0 bs1` Select style of delay for backspace.
`hup` Hang up on last close.
`0` Hang up immediately.
`50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb`
Set terminal baud rate to the number given, if possible.
`old` Arrange to use normal teletype line discipline, *ttyld*(4).
`old!` Force normal teletype line discipline, even if no teletype driver was present.
`notty` Remove the top-level teletype driver.

SEE ALSO

ttyld(4), *ioctl*(2), *tabs*(1)

NAME

sum, treesum − sum and count blocks in a file

SYNOPSIS

`sum` [`−5ri`] [*file* ...]

`treesum` [*file* ...]

DESCRIPTION

By default, *sum* calculates and prints a 32-bit checksum, a byte count and the name of each *file*. The checksum is also a function of the input length. If no files are given, the standard input is summed. Other summing algorithms are available. The options are

`−i` Read file names from standard input.

`−r` Sum with the algorithm of System V's `sum −r` and print the length (in 1K blocks) of the input.

`−5` Sum with System V's default algorithm and print the length (in 512-byte blocks) of the input.

Sum is typically used to look for bad spots, to validate a file communicated over some transmission line or as a quick way to determine if two files might be the same.

Treesum is similar to `sum −r`, except that if *file* is a directory, then *treesum* recursively descends it, summing all non-directories encountered. If no files are given, *treesum* recursively sums the current directory.

SEE ALSO

wc(1)

NAME

tabs − set terminal tabs

SYNOPSIS

`tabs` [+*num*] [`−T`*term*]

DESCRIPTION

Tabs sets the tabs on a variety of terminals. *Term* is a name given in *term*(6). The name may also be supplied by the environment variable `TERM`.

The +*num* option offsets the left margin *num* positions.

SEE ALSO

stty(1), *term*(6), *tset*(A)

NAME

tail, readslow, head − print the last part of a file

SYNOPSIS

`tail` [*otion ... number*`[`lbc`][`rf`]`] [*file*]

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place, normally 10 lines from the end. If no file is named, the standard input is used. The options are

number`[`lbc`][`rf`]`

Copying begins at position +*number* measured from the beginning, or −*number* from the end of the input. *Number* is counted in lines, 1K blocks or characters, according to the appended flag `l` (default), `b`, or `c`. Further flags `r` and `f` have the effect of options `−r` and `−l` .

`−r` Print lines from the end of the file in reverse order. Default line count is unbounded.

`−f` Follow. After printing to the end, keep watch and print further data as it appears.

`−c` *number*
`−n` *number*

Number may be signed, with sign − assumed by default. The effect is the same as `number`*c* or `number`*l* [sic] respectively.

EXAMPLES

`tail file`

Print the last 10 lines of a file.

`tail +0f file`

Print a file, and continue to watch data accumulate as it grows. A similar function is sometimes called *readslow*.

`sed 10q file`

Print the first 10 lines of a file. A similar function is sometimes called *head.*

SEE ALSO

dd(1)

BUGS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length, even under option `−r`.

According to custom, option +*number* counts lines from 1, and counts blocks and characters from 0.

NAME

tape, mt − identify and manipulate magnetic tape

SYNOPSIS

`tape`

`mt` [`-t` *tapename*] *command* [*count*]

DESCRIPTION

Tape experiments with the magnetic tape drive and reports under which device name the tape mounted there can be read, how many files and records there are, and how big the records are. *Mt* applies a *command* to the named tape drive (default `/dev/nrmt1`) *count* times (default 1). The *commands* are

`eof`	write end-of -file mark
`fsf`	forward space file
`fsr`	forward space record
`bsf`	backspace file
`bsr`	backspace record
`rewind`	rewind
`offline`	rewind and take off line

FILES

`/dev/nrmt1`

SEE ALSO

mt(4), *dd*(1)

NAME

tar − tape archiver

SYNOPSIS

`tar` *key* [*file* ...]

DESCRIPTION

Tar saves and restores files, normally on magnetic on tape. The *key* is a string that contains at most one function letter plus optional modifiers. Other arguments to the command are names of files or directories to be dumped or restored. A directory name implies all the contained files and subdirectories (recursively).

The function is one of the following letters:

`r` The named files are written on the end of the tape.

`x` Extract the named files from the tape. If a file is a directory, the directory is extracted recursively. Owners and modes are restored if possible. If no file argument is given, extract the entire tape. If the tape contains multiple entries for a file, the latest one wins.

`t` List all occurrences of each *file* on tape, or of all files if there are no *file* arguments.

`u` Add the named files if they are not on the tape or are newer than the tape version.

`c` Create a new tape; writing begins at the beginning of the tape instead of after the last file.

`o` Omit owner and modes of directories, for compatibility with old versions of *tar*.

`p` Restore files to their original modes, ignoring the present *umask*(2). Setuid and sticky information will be restored when *tar* is executed by the super-user.

The modifiers are:

`0`,...,`7` Select a tape drive. The default is `1`. Incompatible with modifier `f`.

`v` (verbose) Print the name of each file treated preceded by the function letter. With `t`, give more details about the tape entries.

`w` Print the action to be taken followed by file name, then wait for user confirmation. If the answer begins with `y`, the action is performed. Any other input means don't do it.

`f` Use the next argument as the name of the archive instead of the default `/dev/rmt1` . If the name of the file is −, tar writes to standard output or reads from standard input, whichever is appropriate. *Tar* can be used to move hierarchies thus:

```
(cd fromdir; tar cf - .) | (cd todir; tar xf -)
```

`b` Write output in *n*×512-byte blocks, where *n* is the next argument, default 20, maximum 40. Useful for raw magnetic tape archives (see `f` above); destructive for disk archives.

`l` Complain if links cannot be resolved. If `l` is not specified, no error messages are printed.

`L` Write information needed to re-create symbolic links on the tape instead of following the links. Tapes thus written cannot be read on older versions of *tar* .

FILES

`/dev/rmt?`
`/tmp/tar*`

SEE ALSO

cpio(1), *bundle*(1), *mt*(4)

BUGS

There is no way to ask for any but the last occurrence of a file.
Tape errors are handled ungracefully.
The `u` option can be slow, and works only with archives on disk files.
File names are limited to 100 characters.

NAME

tbl − format tables for nroff or troff

SYNOPSIS

`tbl` [*file* ...]

DESCRIPTION

Tbl is a preprocessor for formatting tables for *nroff* or *troff*(1). The input files are copied to the standard output, except for segments of the form

```
.TS
options ;
format .
data
.T&
format .
data
...
.TE
```

which describe tables and are replaced by *troff* requests to lay out the tables. If no arguments are given, *tbl* reads the standard input.

The (optional) *options* line is terminated by a semicolon and contains one or more of

`center`	center the table; default is left-adjust
`expand`	make table as wide as current line length
`box`	
`doublebox`	enclose the table in a box or double box
`allbox`	enclose every item in a box
`tab(`*x*`)`	use *x* to separate input items; default is tab
`linesize(`*n*`)`	set rules in *n*-point type
`delim(`*xy*`)`	recognize *x* and *y* as *eqn*(1) delimiters

Each line, except the last, of the obligatory *format* describes one row of the table. The last line describes all rows until the next `.T&`, where the format changes, or the end of the table at `.TE`. A format is specified by key letters, one per column, upper or lower case

`L`	Left justify: the default for columns without format keys.
`R`	Right justify.
`C`	Center.
`N`	Numeric: align at decimal point (inferred for integers) or at `\&`.
`S`	Span: extend previous column across this one.
`A`	Alphabetic: left-aligned within column, widest item centered, indented relative to `L` rows.
`^`	Vertical span: continue item from previous row into this row.
−	Draw a horizontal rule in this column.
=	Draw a double horizontal rule in this column.

Key letters may be followed by modifiers, also either case:

|	Draw vertical rule between columns.
||	Draw a double vertical rule between columns.
n	Gap between column is *n* ens wide. Default is 3.
`F`*font*	Use specified *font*. `B` and `I` mean `FB` and `FI`.
`T`	Begin vertically-spanned item at top row of range; default is vertical centering (with `^`).
`P`*n*	Use point size *n*.

`V`*n* Use *n*-point vertical spacing in text block; signed *n* means relative change.

`W(`*n*`)` Column width as a *troff* width specification. Parens are optional if *n* is a simple integer.

`E` Equalize the widths of all columns marked `E`.

Each line of *data* becomes one row of the table; tabs separate items. Lines beginning with `.` are *troff* requests. Certain special data items are recognized:

`_` Draw a horizontal rule in this column.

`=` Draw a double horizontal rule in this column. A data line consisting of a single _ or = draws the rule across the whole table.

`\_` Draw a rule only as wide as the contents of the column.

`\R`*x* Repeat character *x* across the column.

`\^` Span the previous item in this column down into this row.

`T{` The item is a text block to be separately formatted by *troff* and placed in the table. The block continues to the next line beginning with `T}`. The remainder of the data line follows at that point.

When it is used in a pipeline with *eqn*, the *tbl* command should be first, to minimize the volume of data passed through pipes.

EXAMPLES

Let <tab> represent a tab (which should be typed as a genuine tab).

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town<tab>Households
<tab>Number<tab>Size
Bedminster<tab>789<tab>3.26
Bernards Twp.<tab>3087<tab>3.74
Bernardsville<tab>2018<tab>3.30
.TE
```

```
c s s
c c s
c c c
l n n.
Household Population
Town  Households
      Number      Size
Bedminster  789   3.26
Bernards Twp.     3087  3.74
Bernardsville     2018  3.30
```

SEE ALSO

troff(1), *eqn*(1), *doctype*(1)

M. E. Lesk and L. L. Cherry, ‘TBL—a Program to Format Tables’, this manual, Volume 2

NAME

tee − pipe fitting

SYNOPSIS

`tee` [`−i`] [`−a`] [*file* ...]

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*. The options are

`−i` Ignore interrupts.

`−a` Append the output to the *files* rather than rewriting them.

NAME

test, [, newer − condition commands

SYNOPSIS

`test` *expr*

[*expr* **]**

`newer` *file1 file2*

DESCRIPTION

Test evaluates the expression *expr*. If the value is true the exit status is 0; otherwise the exit status is nonzero. If there are no arguments the exit status is nonzero.

The following primitives are used to construct *expr*.

`-r` *file* True if the file exists (is accessible) and is readable.
`-w` *file* True if the file exists and is writable.
`-x` *file* True if the file exists and has execute permission.
`-e` *file* True if the file exists.
`-f` *file* True if the file exists and is a plain file.
`-d` *file* True if the file exists and is a directory.
`-c` *file* True if the file exists and is a character special file.
`-b` *file* True if the file exists and is a block special file.
`-L` *file* True if the file is a symbolic link.
`-u` *file* True if the file exists and has set userid permission.
`-g` *file* True if the file exists and has set groupid permission.
`-s` *file* True if the file exists and has a size greater than zero.
`-t` *fildes*
: True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.

`-S` True if the effective userid is zero.
s1 `=` *s2*
: True if the strings *s1* and *s2* are identical.

s1 `!=` *s2*
: True if the strings *s1* and *s2* are not identical.

s1 True if *s1* is not the null string. (Deprecated.)
`-z` *s1* True if the length of string *s1* is zero.
n1 `-eq` *n2*
: True if the integers *n1* and *n2* are arithmetically equal. Any of the comparisons `-ne`, `-gt`, `-ge`, `-lt`, or `-le` may be used in place of `-eq`. The (nonstandard) construct `-l` *string*, meaning the length of *string*, may be used in place of an integer.

These primaries may be combined with the following operators:

`!` unary negation operator
`-o` binary *or* operator
`-a` binary *and* operator; higher precedence than `-o`
`(` *expr* `)`
: parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the Shell and must be escaped.

`[` is a synonym for *test*, except that `[` requires a closing `]`.

Newer returns a zero exit code if *file1* exists and *file2* does not, or if *file1* and *file2* both exist and *file1* was modified at least as recently as *file2*. It returns a non-zero return code otherwise.

EXAMPLES

Test is a dubious way to check for specific character strings: it uses a process to do what a shell case

statement can do. The first example is not only inefficient but wrong, because *test* understands the purported string `"-c"` as an option. Furthermore `$1` might be empty.

```
if test $1 = "-c" # wrong!
then echo OK
fi
```

A correct way is

```
case "$1" in
-c)   echo OK
esac
```

Test whether `abc` is in the current directory.

```
test -e abc -o -L abc
```

SEE ALSO

sh(1), *find*(1)

NAME

tex, dvips, dvit, dvicat − text formatting and typesetting

SYNOPSIS

`tex` [*first-line*]

`dvips` [*option ...*] *dvifile*

`dvit` [*option ...*] *dvifile*

`dvicat` *dvifile ...*

DESCRIPTION

Tex formats interspersed text and commands and outputs a `.dvi` ('device independent') file.

An argument given on the command line behaves as the first input line. That line should begin with a (possibly truncated) file name or a `\controlsequence`. Thus `tex paper` processes the file `paper.tex`. The basename of `paper` becomes the *jobname*, and is used in forming output file names. If no file is named, the jobname is `texput`. The default `.tex` extension can be overridden by specifying an extension explicitly.

The output is written on *jobname*`.dvi`, which can be printed using *lp*(1). A log of error messages goes into *jobname*`.log`.

As well as the standard TeX fonts, many *Postscript* fonts can be used (see the contents of `/usr/lib/tex/fonts/psvf`). The file `testfont.tex` (in the standard macro directory) will print a table of any font.

These environment variables adjust the behavior of *tex*:

`TEXINPUTS`	Search path for `\input` and `\openin` files. It should be colon-separated, and start with dot. Default: `.:/usr/lib/tex/macros`
`TEXFONTS`	Search path for font metric files. Default: `/usr/lib/tex/fonts/tfm`
`TEXFORMATS`	Search path for format files. Default: `/usr/lib/tex/macros`
`TEXPOOL`	Search path for strings. Default: `/usr/lib/tex`
`TEXEDIT`	Template for the switch-to-editor-on-error option, with `%s` for the filename and `%d` for the line number. Default: `/bin/ed %s`

Dvips and *dvit* convert `.dvi` files to Postscript and *troff* output format, respectively, writing the result on standard output. They are invoked by *lp*(1) and accept a subset of *lp* options that make sense for `.dvi` files. In the `-o`*pagelist* option for only printing selected pages, the numbers refer to TeX page numbers. In addition, there is a `-T`*dev* option, where *dev* is one of `laserwriter` (default for *dvips*), `jerq` (default for *dvit*), `gnot`, `fax`, or `lino` (the computer center's high resolution Postscript service). The `-Tjerq` or `-Tgnot` options should be used for preparing output for *proof*(9.1) or *psi*(9.1).

When converting a number of short `.dvi` files to Postscript or using *lp* to print them, it is much more efficient first to combine them via *dvicat*. Simply concatinating `.dvi` files would not work, but *dvicat* achieves this effect and sends the result to standard output. Since the output is in binary, it must be directed to a file or piped into *lp*.

The following environment variables affect *dvips*:

`TEXPKS`	Search path for font bitmaps (PK files).
`TEXVFONTS`	Search path for 'virtual font' descriptions.

Dvips and *dvit* understand some extended graphics commands that can be output using *tpic specials* in the TeX source. Many of them work by building up a path of *x,y* pairs, and then doing something with the path. The tpic coordinate system has its origin at the current dvi position when a drawing special is emitted; all length arguments are in units of milli-inches, and the y-axis goes positive downward.

`\special{pa` *x y*`}`
: Add *x,y* to the current path.

`\special{fp}`
Flush the current path: draw it as a polygonal line and reset the path to be empty.

`\special{da` *dlen*`}`
Like `fp` but draw dashed line, with dashes *dlen* milli-inches long.

`\special{dt` *slen*`}`
Like `fp` but draw a dotted line, with dots *slen* apart.

`\special{sp}`
Like `fp` but draw a quadratic spline. The spline goes through the midpoints of the segments of the path, and straight pieces extend it to the endpoints.

`\special{ar` *x y xr yr s e*`}`
Draw a circular or elliptical arc with center at *x*,*y* and radii *xr* and *yr*. The arc goes clockwise from angle *s* to angle *e* (angles measured clockwise from the positive x-axis).

`\special{pn` *n*`}`
Set line width (pen diameter) to *n* milli-inches.

`\special{bk}`
Set shading to black (will fill the next object drawn with black).

`\special{sh}`
Set shading to grey.

`\special{wh}`
Set shading to white.

`\special{psfile=`*file options*`}`
(Only *dvips*). Include *file*, which should be a Postscript illustration, making its origin be the current dvi position. The default Postscript transformation matrix will be in effect, but it can be modified by the *options*, a list of *key=value* assignments. Allowed keys are: hoffset, voffset, hscale, vscale, and rotate. If supplied, these values are supplied to Postscript *translate*,*scale*, and *rotate* commands, in that order. Also, keys hsize and vsize may be supplied, to cause clipping to those sizes.

`\special{include` *horg yorg file*`}`
(Only *dvit*). Include the *troff*(1) output *file* at the current place on TeX's page. The included file should have only one page. The *horg* and *yorg* distances give the origin of the included file; that point will be superimposed on the current position.

All of the specials leave TeX at the same position on the page that it started in.

FILES

`/usr/lib/tex/macros/*`	macros and preloaded format files
`/usr/lib/tex/macros/doc/*`	more TeX-related documentation
`/usr/lib/tex/fonts/tfm`	font metrics
`/usr/lib/tex/fonts/psvf`	PostScript virtual font metrics
`/usr/lib/tex/fonts/canonpk`	bitmaps for canon engines (300 dpi)
`/usr/lib/tex/fonts/linopk`	bitmaps for Linotron (1270 dpi)
`/usr/lib/tex/*`	miscellaneous configuration files and Postscript headers

SEE ALSO

latex(6), *pic*(1), *lp*(1), *proof*(9.1), *psi*(9.1), *troff*(1), *monk*(1)
Donald E. Knuth, *The TEXbook*, Addison Wesley, 1986

BUGS

It should be possible to make TeX output go on standard output.

NAME

time − time a command

SYNOPSIS

`time` [`-usrtdmfio`] [`-v`] *command*

DESCRIPTION

The given command is executed; after it is complete, *time* reports statistics about the program on its standard error output. The statistics printed are controlled by the option string; the default is `-usr`; `-v` reports everything. The statistics are:

`u` User cpu time consumed by the command

`s` System cpu time attributed to the command

`r` Real (clock) time taken by the command

`t` Average resident text size, in Kb.

`d` Average resident data size, in Kb.

`m` Maximum total resident size, in Kb.

`f` Number of page faults resulting in disk I/O.

`i` Number of disk blocks read.

`o` Number of disk blocks written.

After the statistics, *time* prints a tab and the command, up to the fifth argument. Times are reported in seconds. The numbers are the sum of those for *command* and any children it spawns and waits for.

SEE ALSO

lcomp(1), *prof*(1)

BUGS

Elapsed time is accurate to the second, while the CPU times are measured to 1/60 second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

tr − translate characters

SYNOPSIS

`tr` [`-cds`] [*string1* [*string2*]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. When *string2* is short it is padded to the length of *string1* by duplicating its last character. Any combination of the options `-cds` may be used:

`-c` Complement *string1*: replace it with a lexicographically ordered list of all other 8-bit unsigned characters.

`-d` Delete from input all characters in *string1*.

`-s` Squeeze repeated output characters that occur in *string2* to single characters.

In either string a noninitial sequence *−x*, where *x* is any character (possibly quoted), stands for a range of characters: a possibly empty sequence of codes running from the successor of the previous code up through the code for *x*. The character \ followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A \ followed by any other character stands for that character.

EXAMPLES

```
tr A-Z a-z <mixed >lower
```

Replace all upper-case letters by lower-case.

```
tr -cs A-Za-z '
' <file1 >file2
```

Create a list of all the words in `file1` one per line in `file2`, where a word is taken to be a maximal string of alphabetics. *String2* is given as a quoted newline.

SEE ALSO

ed(1), *ascii*(6)

NAME

troff, nroff − text formatting and typesetting

SYNOPSIS

`troff` [*option* ...] [*file* ...]

`nroff` [*option* ...] [*file* ...]

DESCRIPTION

Troff formats text in the named *files* for printing on a phototypesetter; *nroff* for typewriter-like devices. Their capabilities are described in the references.

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (−) is taken to be a file name corresponding to the standard input. The options, which may appear in any order so long as they appear before the files, are:

`−o`*list* Print pages in the comma-separated *list* of numbers and ranges. A range *N*−*M* means *N* through *M*; initial −*M* means up to *M;* final *N*− means from *N* to the end.

`−n`*N* Number first generated page *N*.

`−m`*name* Prepend the macro file `/usr/lib/tmac/tmac.`*name* to the input *files*.

`−r`*aN* Set register *a* (one character name) to *N*.

`−i` Read standard input after the input files are exhausted.

`−q` Invoke the simultaneous input-output mode of the `rd` request.

`−z` produce no output: diagnostics and `.tm` messages only

Troff only

`−a` Send a printable ASCII approximation of the results to the standard output.

`−T`*dest* Prepare output for typesetter *dest*:
`−Tpost` Apple LaserWriter and other PostScript printers (default)
`−T202` Mergenthaler Linotron 202
`−Taps` Autologic APS-5

`−F`*dir* Take font information from directory *dir*.

Nroff only

`−s`*N* Halt prior to every *N* pages (default *N*=1) to allow paper loading or changing.

`−T`*name* Prepare output for specified terminal. Known *names* include `37` for the (default) Teletype model 37, `lp` (‘line-printer’) for any terminal without half-line capability, `450` for the DASI-450 (Diablo Hyterm), and `think` (HP ThinkJet, see *thinkblt*(9.1)).

`−e` Produce equally-spaced words in adjusted lines, using full terminal resolution.

`−h` Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

FILES

`/tmp/trtmp*`	temporary file
`/usr/lib/tmac/tmac.*`	standard macro files
`/usr/lib/term/*`	terminal driving tables for *nroff*
`/usr/lib/font/*`	font width tables for *troff*

SEE ALSO

lp(1), *d202*(A), *proof*(9.1), *apsend*(1), *reader*(9.7)
eqn(1), *tbl*(1), *prefer*(1), *pic*(1), *ideal*(1), *grap*(1), *dag*(1), *cip*(9.1), *ped*(9.1)
doctype(1), *ms*(6), *mpm*(6), *mbits*(6), *mpictures*(6), *mcs*(6), *font*(5), *monk*(1), *tex*(1)
J. F. Ossanna and B. W. Kernighan, ‘Nroff/Troff User’s Manual’, this manual, Volume 2
B. W. Kernighan, ‘A TROFF Tutorial’, *ibid.*

NAME

true, false − provide truth values

SYNOPSIS

```
true

false
```

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully.

SEE ALSO

sh(1)

DIAGNOSTICS

True has exit status zero, *false* nonzero.

BUGS

For most purposes, *true* is a slow equivalent for the shell builtin command **:**.

NAME

tsort − topological sort

SYNOPSIS

`tsort` [*file*]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

NAME

tty, logtty − get terminal name

SYNOPSIS

`tty` [`-s`]

`logtty`

DESCRIPTION

Tty prints the pathname associated with the standard input file if it can be found in the `/dev` directory, `nameless pipe/` and a unique string if the file is a pipe, `not a tty` otherwise. Option `-s` suppresses output, returning exit status only.

Logtty prints the pathname associated with the terminal on which the current session was logged in. If the login terminal can't be found, `no login tty` is printed.

In a *mux*(9.1) window, *tty* reports the name of the window, while *logtty* reports the name of the terminal. But see BUGS.

SEE ALSO

who(1)

DIAGNOSTICS

Exit status is 0 if a real pathname was printed, 1 for `not a tty` or `no login tty`, 2 for a pipe.

BUGS

Mounting something atop the login terminal hides it from *logtty*. *Vismon*(9.1) does this.

Try `tty </`.

NAME

ul − print underlines on screen terminals

SYNOPSIS

`ul` [`−i`] [`−t` *terminal*] [*file* ...]

DESCRIPTION

Ul replaces backspaced, overstruck underscores by control sequences suitable for the terminal given by the environment variable `TERM` or by option `−t`. It reads from the standard input or the named files and writes on the standard output. Option `−i` represents underlining by a separate line of − characters.

SEE ALSO

column(1)

NAME

uniq − report repeated lines in a file

SYNOPSIS

`uniq` [`−udc` [`+−`*num*]] [*file*]

DESCRIPTION

Uniq copies the input *file,* or the standard input, to the standard output comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed. Repeated lines must be adjacent in order to be found.

`−u` Print unique lines.

`−d` Print (one copy of) duplicated lines.

`−c` Prefix a repetition count and a tab to each output line. Implies `−u` and `−d`.

`−`*num* The first *num* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

`+`*num* The first *num* characters are ignored. Fields are skipped before characters.

EXAMPLES

```
cut -d: -f3 /etc/passwd | sort | uniq -d
```

Print duplicated userids from the password file, *passwd*(5). *Cut* picks out the userid (the third colon-delimited field) and *sort* brings repetitions together.

SEE ALSO

sort(1), *comm*(1)

BUGS

Field-selection and comparison should be compatible with *sort*(1).

NAME

uucp, uulog, uuname − unix-to-unix remote file copy

SYNOPSIS

`uucp` [*options ...*] *source ... destination*

`uulog` [*option ...*] [*system*]

`uuname` [`-l`]

DESCRIPTION

Uucp copies *source* files to the *destination* file or directory. A file name may be an ordinary path name, or may have the form:

system-name ! *filename*

where *system-name* is a computer that *uucp* knows about.

Quoted shell metacharacters `?`, `*` and `[ ]` appearing in a remote *filename* will be expanded on the remote system.

Path names may be:

(1) a full path name;

(2) a path name preceded by *~user/* where *user* is a login name on the specified system and is replaced by that user's login directory;

(3) a path name preceded by *~/destination* where ~ stands for `/usr/spool/uucppublic`. The destination will be treated as a file name unless more than one file is being transferred by this request, the destination is already a directory, or the destination ends with `/`. For example, `~/dan/` as the destination will make the directory `/usr/spool/uucppublic/dan` if it does not exist, and put the requested file(s) in that directory.

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination* is a directory, the basename of the *filename* is used. *Uucp* preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2)).

For obvious security reasons, the domain of remotely accessible files may be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. Similarly you will probably not be able to send files to arbitrary path names. By default, the only remotely accessible files are those whose names begin `/usr/spool/uucppublic/` (equivalent to `~/`).

The options are

`-C` Copy files in their present state to a spool directory for later transfer. `-c` Do not copy to the spool directory; send files in their state at the time of transmission (default).

`-d` Make all necessary directories for the file copy (default).

`-f` Do not make intermediate directories for the file copy.

`-g`*grade* Grade is a single letter/number; earlier ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation. The default is `N`.

`-j` Output the job identification ASCII string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.

`-m` Send mail to the requester when the copy is completed.

`-n`*user* Notify *user* on the remote system that a file was sent.

`-r` Don't start the file transfer, just queue the job.

`-x`*debug-level*
Produce debugging output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

Uulog queries a log file of *uucp* or *uuxqt* transactions, optionally limited to a given *system*. Its options are

`-f`	Print recent transactions and follow further transactions as they occur.
`-x`	Look in the uuxqt log file for the given system.
`-`*number*	Print the last *number* transactions.

Uuname lists the uucp names of known systems. The `-l` option returns the local system name.

FILES

`/usr/spool/uucp`	spool directories
`/usr/spool/uucppublic`	public directory for receiving and sending
`/usr/lib/uucp/*`	other data and program files
`/usr/spool/uucp/.Log/uuxqt/`*system*	log of uuxqt transactions with *system*
`/usr/spool/uucp/.Log/uucico/`*system*	log of uucp transactions with *system*

SEE ALSO

uuto(1), *mail*(1), *push*(1), *rcp*(1), *uux*(1), *uustat*(1), *uucico*(8)
P. Honeyman, 'UUCP—the Program that Wouldn't Go Away', this manual, Volume 2

BUGS

For various reasons remote systems may decline to forward files transmitted through them.
All files received by *uucp* will be owned by user 'uucp'.
Option `-m` works only with a single file.
Uucp may run under a daemon userid, in which case files to be sent need general read permission.

NAME

uustat − uucp status inquiry and job control

SYNOPSIS

`uustat` [*option*]

`uustat` [`-s`*system*] [`-u`*user*]

DESCRIPTION

Uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. The options are

`-a` List the status of all pending *uucp* requests for all machines.

`-k`*jobid*
Kill the *uucp* request whose job identification is *jobid*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

`-m` Report the status of accessibility of all machines.

`-p` Report on the status of all processes that are in the lock files.

`-q` List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. A parenthesized number next to the number of `C` or `X` files gives the age in days of the oldest file for that system. The retry field represents the number of hours until the next possible call. The count field is the number of failure attempts.

`-r`*jobid*
Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup demon from deleting the job until its modification time reaches the limit imposed by the demon.

`-s`*sys* Report the status of all *uucp* requests for remote system *sys*.

`-u`*user* Report the status of all *uucp* requests issued by *user*.

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user.

Requests are listed in the form

```
jobid    date    type    machine stuff
         date    type    machine stuff        ...
```

Jobid identifies the request; it is useful for `-k` and `-r`. The remainder of the line describes a transfer queued at *date* for *machine*. *Type* is `S` if a file is to be sent to *machine*, `R` if it is to be received. Ordinary files are followed by the requestor's userid, the length of the file in bytes, and the name of the spooled file; requests for remote execution are followed by the userid and the command. If the request involves more than one file, the remaining files are listed without a *jobid*.

The most common case is a mail request, which has two lines, one for the mail message itself and one for the request to execute *rmail* on the remote system.

FILES

`/usr/spool/uucp/*` spool directories

SEE ALSO

uucp(1), *uux*(1)

NAME

uuto, uupick − simplified unix-to-unix remote file copy

SYNOPSIS

`uuto` [*option ...*] *file ... recipient*

`uupick` [`-s` *system*]

DESCRIPTION

Uuto sends the named *files* to a *recipient* on another system by means of *uucp*(1). The recipient is named thus:

system ! *user*

The options are

`-p` Make a copy of the *files*. This option allows you to delete or modify the files without worrying about whether they have yet been sent.

`-m` Notify the sender by mail when the copy is complete. (The recipient is always notified.)

Uupick accepts or rejects files received from *uuto*. For each file currently available, it announces

`from system` *sys-name* `:  file` *file-name*

followed by `?`. Give one of these answers on the standard input:

<new-line>
Go on to next entry.

`d` Delete the entry.

`m` *dir* Move the entry to directory *dir*. If *dir* is missing, use the current directory.

`a` *dir* Like `m`, but move all files from the given system.

`p` Print the file.

<EOT> (control-d)
`q` Stop.

`!` *command*
Escape to *sh*(1) command.

anything else
Print a usage summary.

Option `-s` picks files only from the named *system*.

FILES

`/usr/spool/uucppublic/receive/`*recipient*`/`*sendingsystem*`/*` where the files go

SEE ALSO

mail(1), *push*(1), *rcp*(1), *uucp*(1)

NAME

uux − unix to unix command execution

SYNOPSIS

`uux` [*options*] *command-arg* ...

DESCRIPTION

Uux will execute a command on a specified system with files and standard output on specified files and systems. For security reasons, most machines allow only selected commands to be run, perhaps only receipt of incoming mail.

The *command-args* make up a *sh*(1) command with with command and file arguments possibly written *system*!*file*. A missing *system* is interpreted as the local system. *Files* may be prefixed by *~xxx/* to represent the home directory for login name *xxx* on the specified system.

Uux copies all files to the execution system. Files to be returned as outputs must be enclosed in (escaped) parentheses. Files must have general read permission.

The options are

`-a`*name*	Use *name* as the user identification replacing the initiator user-id. (Notification will be returned to the user.)
`-b`	Return standard input to the command if the exit status is non-zero.
`-c`	Don't copy local file to the spool directory for transfer to the remote machine (default).
`-C`	Force the copy of local files to the spool directory for transfer.
`-g`*grade*	*Grade* is a single letter/number; earlier ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation. The default is `N`.
`-j`	Place the jobid, an ASCII string, on the standard output. The jobid can be used by *uustat*(1) to obtain the status or terminate a job.
`-n`	Suppress mail notification about failures.
`-p`	
`-`	Take the standard input to *uux* as the standard input to the executed command.
`-r`	Don't start the file transfer, just queue the job.
`-s`*file*	Report status of the transfer in *file*.
`-x`*debug*	Produce debugging output on stdout. *Debug* is a number between 0 and 9; higher numbers give more detailed information.
`-z`	Notify the user if the command succeeds.

FILES

`/usr/spool/uucp`	spool directory
`/usr/lib/uucp/*`	other data and programs

SEE ALSO

uucp(1), *uucico*(8), *uustat*(1)

BUGS

All the commands in a shell pipeline are executed on the machine of the first command.

Because files are gathered into a common directory, two files for one command cannot have the same basename. This won't work: `uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"`.

NAME

ex, edit, vi − text editor

SYNOPSIS

`ex` [*option* ...] *file* ...

`edit` [*option* ...] *file* ...

`vi` [*option* ...] *file* ...

DESCRIPTION

Ex is elaborated from *ed*(1). *Vi* is the display editing aspect of *ex*; *edit* is a simplified subset of *ex*. The editors work on several *files* simultaneously. *Vi* gets the terminal type from environment variable `TERM`.

The options are:

`−r` *file* Recover the *file* after a crash in the editor. If no file is specified, list all saved files.

`−l` Lisp indent mode; give special meaning to editor commands `()[]{}`.

`−w`*n* Default window size is *n* lines.

`−x` Edit encrypted files; a key will be prompted for.

`−R` Read only; no files will be changed.

`+`*cmd* Execute the specified *ex* command upon entering the editor.

The editors begin in 'command mode'. An ESC character typed in command mode cancels the command. The commands `a A c C i I o O R s S` enter input mode, where arbitrary text may be entered. ESC or interrupt returns to command mode. The commands `: / ? !` take input from the last line of the screen, which is ended by a newline, or canceled by an interrupt.

A number preceding commands `z G |` specifies a line. A number before almost any other command causes it to be repeated.

In the following summary of *vi* commands ^ means the 'control' key. Commands so marked work in input mode; others work in command mode.

File manipulation

`:w`	write back changes	`:e +`	file edit, starting at end
`:w` *file*	write *file*	`:e +` *starting*	at line *n*
`:w!` *file*	overwrite *file*	`:sh`	run shell, then return
`:q`	quit	`:!` *cmd*	run shell command
`:q!`	quit, discarding changes	`:n`	edit next *file* argument
`:e` *file*	edit *file*	`:` *ex-cmd*	do editor command
`:e!`	reedit, discarding changes		

Positioning in file

`^F`	forward screen	`n`	repeat last `\ or ?`
`^B`	backward screen	/*pat*/± *n*	*n*th line after or before *pat*
`^D`	down half screen	`(`	beginning of sentence
`^U`	up half screen	`)`	end of sentence
`G`	to specified line (end default)	`{`	beginning of paragraph
`/` *pat*	next line matching *pat*	`}`	end of paragraph
`?` *pat*	previous line matching	`%`	find matching `(){}`

Adjusting the screen

`^L`	clear and redraw	`z.`	... at center
`^R`	retype, eliminate `@` lines	/*pat*/`z−`	*pat* line at bottom
`z`	redraw, current line at window	`z` *n*	use *n*-line window
`z−`	... at bottom	`^E`	scroll down one line

^Y	scroll up one line

Marking and returning

``	move cursor to previous context	` *x*	move cursor to mark
''	... at first nonwhite in	' *x*	... at first nonwhite
m *x*	mark position with letter *x*		

Line positioning

H	top line of screen	–	previous line, at first nonwhite
L	bottom line	<newline>	same as +
M	middle line	j	next line, same column
+	next line, at first nonwhite	k	previous line, same column

Character positioning

^	first nonwhite (not CTRL)	F *x*	find backward
0	beginning of line	t *x*	upto *x* forward
$	end of line	T *x*	back upto *x*
l	forward	;	repeat last f F t
h	backward	,	opposite of ;
^L	^H same as l h	|	to specified column
<space>	same as l	%	find matching ()\{\}
f *x*	find *x* forward		

Words, sentences, paragraphs

w	word forward	}	next paragraph
b	word backward	{	previous paragraph
e	end of word	W	blank-delimited word
)	next sentence	B	backward to ...
(	previous sentence	E	to end of ...

Corrections

^H	erase last character	ESC	return to command mode
^W	erase last word	^?	interrupt, return to command mode
erase	your erase, same as ^H	^D	backtab over autoindent
kill	your kill, erase this line	^V	quote nonprinting character
\	quotes your erase or kill		

Insert and replace

a	append after cursor	o	open below line
A	append at end of line	O	open above
i	insert before cursor	r *x*	replace one character
I	insert before first non-blank	R *text*	replace characters

Operators

Operators are followed by cursor motion and affect all text that would have been moved over: dw deletes a word. Double the operator, e.g. dd, to affect whole lines.

d	delete	>	right shift
c	change	!	filter through command
y	yank lines to buffer	=	Lisp indent
<	left shift		

Miscellaneous operators

C	change rest of line (c$)	J	join lines
D	delete rest of line (d$)	x	delete characters (dl)
s	substitute characters (cd)	X	... before cursor (dh)
S	substitute lines (cc)	Y	yank lines (yy)

Yank and put

`p`	put buffer after cursor	`x` *y*	yank to buffer *x*
`P`	put buffer before cursor	`x` *d*	delete to buffer *x*
`"` *x* `p`	put from buffer *x*		

Undo, redo, retrieve

`u`	undo last change	`.`	repeat last change
`U`	restore current line	`"` *n* `p`	retrieve *n*-th last delete

Special ex commands

Ex understands most *ed* commands, plus the commands indicated with `:` in the *vi* summary, plus the following, shown with their shortest possible spellings:

`ab`*brev*	`so`*urce*
`ar`*gs*	`st`*op*
`co`*py*	`una`*bbrev*
`map`	`u`*ndo*
`ma`*rk*	`unm`*ap*
`n`*ext*	`ve`*rsion*
`nu`*mber*	`vi`*sual*
`pre`*serve*	`x`*it*
`pu`*t*	`ya`*nk*
`re`*ad*	`z` *window*
`rec`*over*	`<` *lshift*
`rew`*ind*	`>` *rshift*
`se`*t*	`^D` *scroll*
`sh`*ell*	

The `set` command controls various options; `set all` shows the list.

EXAMPLES

←↓↑→	arrow keys move the cursor	`3dw`	delete 3 words
`hjkl`	same as arrow keys	`ZZ`	exit, saving changes
`i`*text*<ESC>	insert *text*	`/`*text*<newline>	search for *text*
`cw`*new*<ESC>	change word to *new*	`^U ^D`	scroll up or down

FILES

`/usr/lib/ex?.?recover`	recover command
`/usr/lib/ex?.?preserve`	preserve command
`/etc/termcap`	describes capabilities of terminals
`$HOME/.exrc`	editor startup file
`/tmp/Ex*`	editor temporary
`/tmp/Rx*`	named buffer temporary
`/usr/preserve`	preservation directory

SEE ALSO

ed(1), *sam*(9.1), *sed*(1)

W. Joy and M. Horton, ‘An Introduction to Display Editing with Vi’, in *Unix Programmer’s Manual, Seventh Edition, Virtual VAX-11 Version*, 1980, Vol 2C (Berkeley)

NAME

view2d, regrid, vdata − movie of a function f(x, y, t)

SYNOPSIS

`view2d` [*option ...*] *file*

`regrid` [*option ...*] *file*

`vdata` [*option ...*] *file*

DESCRIPTION

View2d displays a sequence of functional surfaces on a Teletype 5620 or frame buffer. On the 5620, the surface is ruled with a square mesh and projected isometrically; on a frame buffer, the surface is viewed from above, with height indicated by color. The options are

`−T5` Output on the 5620;

`−Tc` Output on frame buffer (default); `−Tc` is assumed when `TERM` is not 5620.

`−Tp` Produce input for *plot*(1) to produce contour plots.

`−c`*n* Use *n* colors (*n*=32 by default) or contours (6 by default).

`−p`*s* Run movie for *s* seconds (*s*=5 by default).

There are two supported ways to generate input for *view2d*. *Vdata* takes ASCII input and allows scattered data; *view2d*(3) can be called from a program that already has data on a grid.

Regrid changes the grid sizes in *view2d* binary files, with options:

`−b` Add a bar at the bottom of the image to indicate time.

`−n`*NX*`,`*NY*
Output grid is *NX*×*NY*; `,`*NY* may be omitted.

`−f`*NF* Output will have *NF* frames.

`−r` The plane of closest fit to the first frame is subtracted from all the frames.

`−m`*fmin*`,`*fmax*
Clip the data to the range *fmin*,*fmax.*

One option applies to both *view2d* and *regrid*:

`−t`*TS*`,`*TE*
Display frames from *TS* (default first frame time) to *TE* (may be omitted, default last frame time).

Vdata takes input in the following format and converts to input for *view2d*. The first line of each frame has the number of data points in the frame (an integer) and the time (a floating point value). The rest of the lines in the frame consist of blank-separated *x y z* triples. The order of points is irrelevant. The options are:

`−n`*NX*`,`*NY*
Output grid is *NX*×*NY*; `,`*NY* may be omitted.

`−c` Use piecewise constant rather than linear interpolation.

`−s` Scatterplot the data on a black background.

`−i` The data is already on a square grid, in the order *x* and *y* ascending with *x* varying fastest.

FILES

`/usr/lib/view2d/*`

SEE ALSO

view2d(3), *view2d*(5), *plot*(1)
`/n/bowell/usr/lib/view2d/howto` for more options and how to run equipment.

NAME

vis − show invisible characters

SYNOPSIS

`vis` [`−t`] [`−s`] [*file* ...]

DESCRIPTION

Vis reads each *file* in sequence and writes it on the standard output. Non-printing characters are translated on output as in the `l` command of *ed*(1). If no *file* is given *vis* reads from the standard input. The options are

`−t` Do not translate tabs.

`−s` Strip invisible characters; same effect as `tr −cd '`<tab><newline><space>`−~'`.

SEE ALSO

cat(1), *ed*(1), *xd*(1)

NAME

wc − word count

SYNOPSIS

`wc` [`-lwc`] [*file* ...]

DESCRIPTION

Wc counts lines, words and characters in the named *files*, or in the standard input if no file is named. A word is a maximal string of characters delimited by spaces, tabs or newlines.

If the optional argument is present, just the specified counts (lines, words or characters) are selected by the letters `l`, `w`, or `c`.

NAME

who, whois, last − who is or was on the system

SYNOPSIS

`who` [`−i`] [*who-file*]

`who am i`

`whois` *username*

`last` [`−f` *who-file*] [*userid* ...] [*terminal*]

DESCRIPTION

Who, without an argument, lists the login name, terminal name, and login time for each current user. With the `−i` option, the report includes the number of minutes that the user's terminal has been idle.

With two arguments, as in `who am i`, *who* tells who you are logged in as.

Without an argument, *who* examines the `/etc/utmp` file to obtain its information. If a file is given, that file is examined. Typically the given file will be `/usr/adm/wtmp`, which contains a record of all the logins since it was created. Then *who* lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with `/dev/` suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with `x` in the place of the device name, and a fossil time indicative of when the system went down.

Whois consults administrative files to identify the *username.* (Actually, *whois* uses *grep* and can locate information by any useful key, such as real name or telephone number.)

Last reports logins and logouts in reverse chronological order. Optional arguments restrict attention to selected userids or terminals. Terminals `tty0`, `tty1`, ... may be abbreviated `0`, `1`, ...

By default, *last* examines the list of logins and logouts in `/usr/adm/wtmp`; option `−f` specifies a different file.

Last reports userid, terminal, time on, and time off for all users, or for selected *userids.* A pseudo-user, `reboot`, is logged in at reboots of the system.

Upon interrupt, *last* tells how far back it has looked; upon quit (control-`\`) it tells how far and keeps on looking.

EXAMPLES

`last reboot`

Report recent system outages.

FILES

`/etc/utmp`
`/usr/adm/wtmp`
`/usr/adm/usrlist`
`/etc/passwd`

SEE ALSO

getuid(1), *getuid*(2), *utmp*(5), *ac*(8), *tty*(1), *vwhois* in *vismon*(9.1)

NAME

write, mesg − write to other users, allow or forbid messages

SYNOPSIS

`write` *person ...*

`mesg [ n ] [ y ]`

DESCRIPTION

Write copies lines from your terminal to terminals of other *persons* designated either by login name or (to circumvent occasional ambiguities) by terminal name as given by *who*(1). It announces to each *person* your login name, your terminal, and the other *persons.* To respond, each recipient should execute a corresponding *write* to the *persons* he wants to talk to.

When you are writing to more than one person, your messages are identified to the recipients. Writing ceases upon end of file or interrupt, and the message `EOF` is sent to the others.

Write recognizes certain commands during a conversation:

`!`*cmd* Execute a shell on the string *cmd* and then return to *write*.

`:a` *person*
Add *person* to the list of people to whom you are talking, and send an appropriate announcement to all parties. They must do `:a` for themselves if they want to include the new person.

`:d` *person*
Drop *person* from your list and make appropriate announcements.

`:l` Print a list of people to whom you are talking.

The following protocol is suggested for using *write*. When you invoke *write,* wait for the other user to write back before starting to send. Each party should end each message with a distinctive signal that the other may reply − the customary convention is `(o)` for ‘over’; `(oo)` for ‘over and out’ is suggested when conversation is about to be terminated.

Mesg with argument `n` forbids messages via *write*(1) by revoking non-user write permission on the user’s terminal. *Mesg* with argument `y` reinstates permission. All by itself, *mesg* reports the current state without changing it.

Certain commands, in particular *nroff* and *pr*(1) disallow messages in order to prevent messy output.

FILES

`/etc/utmp` to find user
`/bin/sh` to execute `!`

SEE ALSO

who(1), *mail*(1), *vismon*(9.1)

DIAGNOSTICS

Mesg yields exit status 0 if messages are receivable, 1 if not, 2 on error.

BUGS

Mux(9.1) generally prevents the receipt of *write* messages, but *vismon*(9.1) permits them.
Messages ought to be identified when the recipient is receiving from more than one writer, rather than when a writer is sending to more than one recipient, but that requires cooperating processes and isn’t worth the effort.

NAME

wwb, style, diction, punct − writer's workbench

SYNOPSIS

`wwb style` [*option ...*] *file ...*

`wwb diction` [*option ...*] *file ...*

`wwb suggest`

`wwb punct` *file ...*

`wwb wwb` [`−ms`] [`−mm`] *file ...*

DESCRIPTION

Wwb controls many subprograms documented in the references. For the full treatment use `wwb wwb`, but for quick use the following subprograms are recommended.

Style reports on readability, sentence length and structure, word length and usage, verb type, and sentence openers in the named documents. Options peculiar to *style* are

`−a` Print all sentences with their length and readability index.

`−e` Print all sentences that begin with an expletive.

`−p` Print all sentences that contain a passive verb.

`−l` *num* Print all sentences longer than *num*.

`−r` *num* Print all sentences whose readability index is greater than *num*.

`−P` Print parts of speech of the words in the document.

Diction prints all sentences that contain phrases from a list of bad or verbose word patterns. Options peculiar to *diction* are

`−p` *pfile* Use an additional private pattern file.

`−n` Skip the standard pattern file.

Suggest reads, from the standard input, phrases deprecated by *diction* and proposes alternatives.

Punct reports sentences that appear to violate standard punctuation rules or that contain doubled words.

Style, diction, and *wwb* expect documents prepared for *troff*(1) with the *ms*(6) macro package and the preprocessors *pic, ideal, grap, eqn, refer,* and *tbl*(1). Two options override:

`−mm` The document uses the `−mm` macro package.

`−ml` The document uses the `−mm` macro package. All *mm*-style lists should be skipped (to avoid attempting to parse nonsentences).

FILES

`/usr/bin/WWB/*`
`/usr/lib/style/*`

SEE ALSO

Directories `/n/bowell/usr/src/man/manw` and `/usr/spool/man/manw` contain manual pages for further *wwb* subcommands; to print them, execute `man proofr`.
Unix System V Writer's Workbench Software Release 2.0 Documentation Set, AT&T Select Code 302-949
deroff(1), *spell*(1)

BUGS

Use of non-standard formatting macros causes incorrect sentence breaks.
Imperative sentences cannot be recognized.

NAME

wwv − print or set the date from accurate clock

SYNOPSIS

`wwv` [option ...]

DESCRIPTION

Wwv connects to a clock synchronized with a time standard. With no argument it prints the time obtained. It accepts these options:

`−b` Print both the WWV time and the system's current idea of the time.

`−s` Set the system's time to agree with WWV. Only the super-user can do this. The system time cannot be changed by more than 20 minutes.

`−f` With `−s`, force the time to be set by WWV even if the local time disagrees by more than 20 minutes.

`−u` Report time in GMT rather than local time.

FILES

`/usr/adm/wtmp` to record time-setting; see *utmp*(5)

SEE ALSO

date(1), *time*(2), *stime*(2)

NAME

xd, od – hex, octal, decimal, or ASCII dump

SYNOPSIS

`xd` [*option ...*] [`-`*format ...*] [*file ...*]

`od` [`-bcdox`] [*file*] [`+`*offset*]

DESCRIPTION

Xd concatenates and dumps the *files* (standard input by default) in one or more formats. Groups of 16 bytes are printed in each of the named formats, one format per line. Each line of output is prefixed by its address (byte offset) in the input file. The first line of output for each group is zero-padded; subsequent are blank-padded.

Formats other than `-c` are specified by pairs of characters telling size and style, `4x` by default. The sizes are

`1` or `b` 1-byte units.
`2` or `w` 2-byte units.
`4` or `l` 4-byte units.

The styles are

`o` Octal.
`x` Hexadecimal.
`d` Decimal.

Other options are

`-c` Format as `1x` but print ASCII representations or C escape sequences where possible.

`-a`*style* Print file addresses in the given style (and size 4).

`-s` Reverse (swab) the order of bytes in each group of 4 before printing.

`-r` Print repeating groups of identical 16-byte sequences as the first group followed by an asterisk.

Od dumps a *file* or the standard input in one or more formats as selected by the first argument, octal by default.

The format characters mean

`b` Bytes in octal.
`c` Bytes in ASCII with C escapes and 3-digit octal for other characters.
`d` 16-bit words in decimal.
`o` 16-bit words in octal.
`x` 16-bit words in hex.

The *offset* argument tells where in the file to begin dumping. The offset, normally interpreted in octal, is interpreted in hexadecimal if it begins with `x` or `0x`, and in decimal if it ends with `.` or `.b`. If it ends in `b`, it is multiplied by 512. The preceding + may be omitted if *file* is present.

SEE ALSO

adb(1), *strings*(1), *vis*(1)

BUGS

The various output formats don't line up properly in the output of *xd.*

A spurious zero byte reported by *od* at the end of odd-length files is betrayed by the correctly printed final address.

The *offset* is ineffectual if *lseek*(2) won't work on the *file.*

On some raw devices offsets must be a multiple of the natural block size.

NAME

yacc, eyacc − yet another compiler-compiler

SYNOPSIS

`yacc` [*option* ...] *grammar*

`eyacc` [`-v`] [*grammar*]

DESCRIPTION

Yacc converts a context-free grammar and translation code into a set of tables for an LR(1) parser and translator. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, `y.tab.c`, must be compiled by the C compiler to produce a program `yyparse`. This program must be loaded with a lexical analyzer function, *yylex*() (often generated by *lex*(1)), with a *main*() program, and with an error handling routine, *yyerror*(). Simple default versions of the last two are loaded by option `-ly` of *ld*(1).

The options are

`-o` *output*
: Direct output to the specified file instead of `y.tab.c`.

`-D`
: Create file `y.debug`, containing diagnostic messages. To incorporate them in the parser, compile it with preprocessor symbol `YYDEBUG` defined. The amount of diagnostic output from the parser is regulated by values of an external variable `yydebug`:

  0 Report errors.

  1 Also report reductions.

  2 Also report the name of each token returned by `yylex`.

`-v`
: Create file `y.output`, containing a description of the parsing tables and of conflicts arising from ambiguities in the grammar.

`-d`
: Create file `y.tab.h`, containing `#define` statements that associate *yacc*-assigned ‘token codes’ with user-declared ‘token names’. Include it in source files other than `y.tab.c` to give access to the token codes.

`-s` *stem*
: Change the prefix `y` of the file names `y.tab.c`, `y.tab.h`, `y.debug`, and `y.output` to *stem*.

Eyacc is a special version of *yacc*, with systematic error recovery. It is used to compile *pascal*(A).

FILES

`y.output`	
`y.tab.c`	
`y.tab.h`	
`y.debug`	
`y.tmp.*`	temporary file
`y.acts.*`	temporary file
`/usr/lib/yaccpar`	parser prototype for C programs
`/usr/lib/liby.a`	library `-ly`

SEE ALSO

lex(1)
S. C. Johnson and R. Sethi, this manual, Volume 2

BUGS

The parser may not have full information when it writes to `y.debug` so that the names of the tokens returned by `yylex` may be missing.

2 SYSTEM CALLS

2 SYSTEM CALLS

NAME

intro, errno – introduction to system calls and error numbers

SYNOPSIS

```
#include <errno.h>
```

DESCRIPTION

Section 2 describes the entries into the operating system.

File I/O

Files are opened for input or output by *open*(2) or *creat*. These calls return a integer called a *file descriptor* which identifies the file to subsequent I/O calls, notably *read*(2) and *write*. File descriptors range from 0 to 127 in the current system. The system gets to pick the numbers, but they may be reassigned by *dup*(2) and *dup2*.

By agreement among user programs, file descriptor 0 is the standard input, 1 is the standard output, 2 is for error messages, and 3 is the controlling terminal if any. The operating system is unaware of these conventions; it is permissible to close file 0, or even to replace it by a file open only for writing, but many programs will be confused by such chicanery.

Files are normally read or written in sequential order. *Lseek*(2) addresses arbitrary locations.

Files have associated status, consisting of ownerships, permission modes, access dates, and so on. The status is retrieved by *stat*(2); the calls in *chmod*(2) alter parts of it.

New files are made with *creat* (in *open*(2)). An existing file may be given an additional name by *link*(2) or *symlink*; names are removed by *unlink*(2). Directories are created and removed by *mkdir*(2) and *rmdir*.

Device files and communication channels (streams) admit a plethora of special operations, most specific to the device in question; see *ioctl*(2) and the device writeups in section 4, especially *ttyld*(4) for terminals and *stream*(4) for communications. *Pipe*(2) creates nameless streams, useful for local communication. Several streams may be monitored in parallel by *select*(2).

Process execution and control

A new process is created when an existing one calls *fork*(2). The new (child) process starts out with copies of the address space and most other attributes of the old (parent) process. In particular, the child starts out running the same program as the parent; *exec*(2) will bring in a different one.

Each process has an integer process id, unique among all currently active processes; a process group id, used to distribute signals among processes in the same session or window; a userid and groupid, which determine access permissions; and a character-string login name for the current user (not the same as permissions). The calls in *getuid*(2) retrieve and change these values.

Various events cause software traps (signals): program errors like addressing violations, software events like the interrupt key on the terminal, the alarm clock set by *alarm*(2), calls to *kill* (in *signal*(2)). Most signals terminate the process by default; *signal*(2) will arrange to trap or ignore them instead.

A process terminates on receiving a signal or by calling *exit*(2). A parent process may call *wait* (in *exit*(2)) to wait for some child to terminate. A single byte of status information may be passed from *exit* to *wait*.

Timekeeping

Time(2) and *ftime* return the time of day and related information. *Times*(2) returns runtime accounting for this process and its children. *Profil* arranges to increment various locations in memory whenever the clock ticks; it is useful for execution profiling.

Times, *profil*, and a few other calls measure time in clock ticks. The clock frequency is given by the constant `HZ` in `<sys/param.h>`; 60 ticks per second in this system.

SEE ALSO

intro(3), *perror*(3)

DIAGNOSTICS

A 'Diagnostics' paragraph appears in the page for each system call that has an error return. Unless otherwise stated, the error value is the integer −1, and the success value is 0. When an error occurs, an error number is assigned to the external variable *errno*. *Errno* is not cleared on successful calls, so it should be tested only after an error has occurred.

There is a table of messages that describe the errors and a routine for printing them; see *perror*(3). The list below gives the number, the name (as defined in `<errno.h>`), and the *perror* message for each error type. The reasons for error returns are explained in general terms; further explanations for less obvious error returns appear in the writeups of individual system calls.

0 Error 0
No error has occurred.

1 `EPERM` Not owner
An attempt was made to modify a file in some way forbidden except to its owner or the super-user, or an ordinary user attempted to do something allowed only to the super-user.

2 `ENOENT` No such file or directory
A file name was specified and the file should exist but didn't, or one of the directories in a path name did not exist.

3 `ESRCH` No such process
The process whose number was given to *kill* did not exist, or was already dead.

4 `EINTR` Interrupted system call
A signal which the user has elected to catch occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

5 `EIO` I/O error
A physical I/O error or timeout occurred, usually in *read*, *write*, or *ioctl*. This error may in some cases be returned on a call following the one to which it actually applies.

6 `ENXIO` No such device or address
I/O on a special file referred to a device which does not exist or is off line, or beyond the limits of the device.

7 `E2BIG` Arg list too long
An argument list longer than 16384 bytes was presented to *exec*.

8 `ENOEXEC` Exec format error
A request was made to execute a file which, although it had the appropriate permissions, did not start with a valid magic number, see *a.out*(5).

9 `EBADF` Bad file number
A file descriptor referred to no open file, or a *read* (resp. /IR write) a file which was open only for writing (resp. reading).

10 `ECHILD` No children
In *wait*, the process had no living or unwaited-for children.

11 `EAGAIN` No more processes
In *fork*, the system's process table was full or the user was not allowed to create any more processes.

12 `ENOMEM` Not enough memory
During *exec* or *brk*, a program asked for more memory or swap space than the system was able to supply.

13 `EACCES` Permission denied
An attempt was made to access a file in a way forbidden by the protection system.

14 `EFAULT` Bad address
The system encountered a hardware fault in attempting to access the arguments of a system call.

15 `EHASF` Directory not empty
An attempt was made to remove a nonempty directory.

16 `EBUSY` In use
An attempt was made to mount a device that was already mounted (or crashed or was copied in mounted state), to dismount a device on which there was an active file (open file, current directory, mounted-on file, active text segment), or to remove the current directory of some process.

17 `EEXIST` File exists
An existing file was mentioned in an inappropriate context, e.g. *link*.

18 `EXDEV` Cross-device link
A link to a file on another device was attempted.

19 `ENODEV` No such device
An attempt was made to apply an inappropriate system call to a device; e.g. read a write-only device.

20 `ENOTDIR` Not a directory
A non-directory was specified where a directory is required, for example in a path name or as an argument to *chdir*.

21 `EISDIR` Is a directory
An attempt to write on a directory.

22 `EINVAL` Invalid argument
Some invalid argument: dismounting a non-mounted device, mentioning an unknown signal in *signal*, reading or writing a file for which *lseek* has generated a negative pointer. Also set by math functions, see *intro*(3).

23 `ENFILE` File table overflow
The system's table of open files was full, and temporarily no more *opens* could be accepted.

24 `EMFILE` Too many open files
The limit is 128 per process.

25 `ENOTTY` Illegal ioctl
The function code mentioned in *ioctl* does not apply to the file or device.

26 `ETXTBSY` Text file busy
An attempt to execute a pure-procedure program which was open for writing, or to open for writing a pure-procedure program that was being executed.

27 `EFBIG` File too large
The size of a file exceeded the maximum (about 10^9 bytes).

28 `ENOSPC` No space left on device
During a *write* to an ordinary file, there was no free space left on the device.

29 `ESPIPE` Illegal seek
Lseek was issued to a stream device.

30 `EROFS` Read-only file system
An attempt to modify a file or directory was made on a device mounted read-only.

31 `EMLINK` Too many links
An attempt to make more than 32767 links to a file.

32 `EPIPE` Broken pipe
Write to a stream that has been hung up, usually a pipe with no process to read the data. This

condition normally generates a signal; the error is returned if the signal is ignored.

33 `EDOM` Math argument
The argument of a function in the math package (3M) was out of the domain of the function.

34 `ERANGE` Result too large
The value of a function in the math package (3M) was unrepresentable within machine precision.

35 `ELOOP` Link loop
An endless cycle of symbolic links was encountered.

36 `ECONC` Concurrency violation
An *open* or *creat* was in violation of the concurrent access specified for the file.

37 `EGREG` It's all Greg's fault
Something went wrong.

BUGS

Device and file system drivers may use error codes in unexpected or unconventional ways; it is infeasible to list them all. The writeups in section 4 list some such special cases.

NAME

access – determine accessibility of file

SYNOPSIS

```
int access(name, mode)
char *name;
```

DESCRIPTION

Access checks the given file *name* for accessibility. If *mode*`&4` is nonzero, read access is checked. If *mode*`&2` is nonzero, write access is checked. If *mode*`&1` is nonzero, execute access is checked. If *mode*==0, the file merely need exist. In any case all directories leading to the file must permit searches. 0 is returned if the access is permitted,, −1 if not.

Permission is checked against the real userid and groupid of the process; this call is most useful in set-userid and set-groupid programs.

Only access bits are checked. A directory may be announced as writable by *access*, but an attempt to open it for writing will fail (although files may be created there); a file may look executable, but *exec*(2) will fail unless it is in proper format.

If the userid of the process is the owner of the file access is determined by the three owner bits (0700). Otherwise, if the groupid of the process is the group of the file access is determined by the three group bits (0070). Otherwise access is determined by the three other bits (0007).

SEE ALSO

stat(2)

DIAGNOSTICS

`EACCES`, `EFAULT`, `EIO`, `ELOOP`, `ENOENT`, `ENOTDIR`, `EROFS`, `ETXTBSY`

BUGS

On symbolic links permissions are irrelevant and *access* returns nonsense.

NAME

acct – turn accounting on or off

SYNOPSIS

```
int acct(file)
char *file;
```

DESCRIPTION

Acct, with a null-terminated string naming an existing file as argument, turns on accounting; records for each terminating process are appended to *file*. An argument of `(char *)0` causes accounting to be turned off.

Acct may only be invoked by the super-user.

The accounting file format is given in *acct*(5).

SEE ALSO

acct(5), *sa*(8)

DIAGNOSTICS

`EACCES` (file not a regular file), `EBUSY` (accounting already turned on), `EFAULT`, `EIO`, `ELOOP`, `ENOENT`, `ENOTDIR`, `EPERM`, `EROFS`, `ETXTBSY`

BUGS

No accounting is produced for programs running when a crash occurs. In particular nonterminating programs are never accounted for.

NAME

alarm, nap, pause – schedule timing delays

SYNOPSIS

```
unsigned alarm(seconds)
unsigned seconds;

void nap(ticks)

void pause()
```

DESCRIPTION

Alarm causes signal `SIGALRM`, see *signal*(2), to be sent to the invoking process in the number of seconds given by the argument. Unless caught or ignored, the signal terminates the process.

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is 0, any alarm request is canceled. Because the clock has a one second resolution, the signal may occur up to one second early; because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount.

The return value is the amount of time previously remaining in the alarm clock.

Nap suspends execution of the current process for the specified number of clock ticks. If *ticks* is negative, it is taken to be zero; if it is greater than two seconds, it is taken to be two seconds.

Pause only returns upon termination of a signal handler started during the *pause*. It is used to give up control while waiting for a signal, usually from *kill* (see *signal*(2)), *alarm*(2), or the terminal driver *ttyld*(4).

SEE ALSO

kill(1), *signal*(2), *setjmp*(3), *sleep*(3)

BUGS

If the argument to *alarm* is greater than 65535, it is treated as 65535.
If the alarm clock expires during a call to *alarm*, the return value will be 0, and the signal will be delivered immediately after the system call returns. If the routine calling *alarm* saves the return value and later restores it, it will disable any alarm set by the signal handler.

NAME

brk, sbrk – change core allocation

SYNOPSIS

```
int brk(addr)
char *addr;

char *sbrk(incr)
```

DESCRIPTION

Brk sets the system's idea of the lowest location not used by the program (called the break) to *addr* rounded up to the next multiple of 1024 bytes. Locations not less than *addr* and below the stack pointer may cause a memory violation if accessed.

In the alternate function *sbrk*, *incr* more bytes are added to the program's data space and a pointer to the start of the new area is returned. Rounding occurs as with *brk*, but a nominal break is remembered, so rounding does not accumulate.

When a program begins execution via *exec* the break is set at the highest location defined by the program and data storage areas. Ordinarily, therefore, only programs with growing data areas need to use *brk*.

The error return from *sbrk* is `(char *)-1`.

SEE ALSO

exec(2), *end*(3), *malloc*(3)

DIAGNOSTICS

`ENOMEM`

NAME

chdir, chroot – change working or root directory

SYNOPSIS

```
int chdir(dirname)
char *dirname;

int chroot(dirname)
char *dirname;
```

DESCRIPTION

Chdir changes the working directory of the invoking process to *dirname*; *chroot* changes its root directory.

The root directory is the starting point when searching for pathnames beginning with `/`. The working directory is the starting point for pathnames that don't. The root directory normally points to the system root. *Login*(8) initially sets the working directory as specified in the password file.

After *chroot*, it is impossible to name a file outside the subtree rooted at the current root, provided that the current directory is located within the subtree and there are no links pointing outside the subtree (except for the entry `..` in the root directory).

Chroot may only be used by the super-user.

SEE ALSO

sh(1), *passwd*(5)

DIAGNOSTICS

`EACCES`, `EFAULT`, `EIO`, `ELOOP`, `ENOENT`, `ENOTDIR`, `EPERM` (*chroot* only)

BUGS

Using *chroot*, it is quite easy to fool set-userid programs about the contents of the password file (for example).

NAME

chmod, fchmod, chown, fchown, utime – change file mode, owner, group, or times

SYNOPSIS

```
int chmod(file, mode)
char *name;

int fchmod(fildes, mode)

int chown(file, uid, gid)
char *name;

int fchown(fildes, uid, gid)

#include <sys/types.h>

int utime(file, timep)
char *file;
time_t timep[2];
```

DESCRIPTION

These functions change inode information for the file named by a null-terminated string or associated with file descriptor *fildes*.

Chmod and *fchmod* change file permissions and other mode bits to *mode*. Mode values are defined in *stat*(2). Only the `07777` bits of *mode* are significant. Only the owner of a file (or super-user) may change the mode. Only a process in the file's group (or super-user) may set the set-group-id bit, `S_ISGID`.

Chown and *fchown* change the owner, *uid,* and the group, *gid,* of a file. Only the super-user may change a file's owner. The owner of a file may change its group to match the current effective groupid. Other changes are restricted to the super-user.

Utime sets the `st_atime` (access time) and `st_mtime` (modify time) fields for *file* to `timep[0]` and `timep[1]` respectively. The `st_ctime` (inode change time) field for *file* is set to the current time.

The caller must be the owner of the file or the super-user.

SEE ALSO

stat(2), *time*(2)

DIAGNOSTICS

all: `EIO`, `EPERM`
chmod, chown, utime: `ELOOP`, `ENOENT`, `ENOTDIR`, `EACCES`, `EFAULT`
fchmod, fchown: `EBADF`

BUGS

An attempt to change the inode data for a file on a read-only file system is quietly ignored.

NAME

reboot, vadvise, vlimit, vswapon, getgroups, setgroups – system calls to be avoided

SYNOPSIS

```
int reboot(how)

int vadvise(how)

int vlimit(what, limit)

int vswapon(special)
char *special;

#include <sys/param.h>

int getgroups(ngroups, gidset)
short *gidset;

setgroups(ngroups, gidset)
short *gidset;
```

DESCRIPTION

These calls are hangovers from prior versions of the system. Some exist only for system maintenance purposes; some depend on the virtual memory implementation. None should be used except as a last resort. Most are not included in `/lib/libc.a`.

Reboot finishes any pending I/O and reboots the system (if *how* is 0) or puts the system into a tight loop with interrupts disabled (if *how* is 8). It is restricted to the super-user.

Vadvise gives the virtual memory system hints about the paging behavior of the current process.

Vlimit sets various resource limits, such as the amount of memory allowed for text and data, and the maximum size of core images.

Vswapon adds the block device *special* to the pool of swap space. The device must be listed in a table compiled into the operating system; *vswapon* merely enables it.

Getgroups stores at most *ngroups* elements of the group access list of the current process in the array *gidset*.

Setgroups sets the group access list of the current user process from *gidset*. *Ngroups* gives the number of entries; it must not exceed `NGROUPS`, defined in `<param.h>`. Only the super-user may add groups to the list.

SEE ALSO

Unix Programmer's Manual, Seventh Edition, Virtual VAX-11 Version, Volume 1, 1980 (Berkeley)

NAME

dirread – read from directory, hiding format

SYNOPSIS

```
int dirread(fildes, buffer, nbytes)
char *buffer;
```

DESCRIPTION

Dirread reads at most *nbytes* bytes of directory information from the file pointer location in the directory associated with *fildes* into memory at *buffer*. The information is converted into a canonical form, independent of the format used by the file system in which the directory resides.

Each canonicalized entry consists of a decimal ASCII inode number, a tab character, a file name, and an ASCII NUL. *Buffer* is filled with as many entries as will fit; the number of bytes used is returned. 0 is returned when there are no more entries.

The file pointer is advanced by the number of bytes passed over in the directory, not the number of bytes placed in *buffer*. Ask *lseek* for the new pointer; don't try to compute it.

SEE ALSO

open(2), *read*(2), *directory*(3), *dir*(5)

DIAGNOSTICS

`EBADF`, `ENOSPC` (buffer too small), `ENOTDIR`, `EFAULT`, `EINVAL`

BUGS

Not all file system types support *dirread*. The routines in *directory*(3) know how to cope.
Seeking in directories is dangerous, because the contents often change under foot.

NAME

dup, dup2 – duplicate an open file descriptor

SYNOPSIS

```
int dup(fildes)
int fildes;

int dup2(fildes, fildes2)
int fildes, fildes2;
```

DESCRIPTION

Given a file descriptor *dup* allocates another file descriptor synonymous with the original. The new file descriptor is returned.

In *dup2*, *fildes* is a file descriptor referring to an open file, and *fildes2* is an integer in the range of legal file descriptors. *Dup2* causes *fildes2* to refer to the same file and returns *fildes2*. If *fildes2* already referred to another open file, it is closed first.

SEE ALSO

open(2), *pipe*(2), *fd*(4)

DIAGNOSTICS

`EBADF`, `EMFILE`

BUGS

Dup of a file descriptor greater than 63 turns into a *dup2* with a random second argument.

NAME

execl, execv, execle, execve, execlp, execvp, exect – execute a file

SYNOPSIS

```
int execl(name, arg0, arg1, ..., argn, (char *)0)
char *name, *arg0, *arg1, ..., *argn;

int execv(name, argv)
char *name, *argv[];

int execle(name, arg0, arg1, ..., argn, (char *)0, envp)
char *name, *arg0, *arg1, ..., *argn, *envp[];

int execve(name, argv, envp)
char *name, *argv[], *envp[];

int execlp(name, arg0, arg1, ..., argn, (char *)0)
char *name, *arg0, *arg1, ..., *argn;

int execvp(name, argv)
char *name, *argv[];

int exect(name, argv, envp)
char *name, *argv[], *envp[];
```

DESCRIPTION

Exec in all its forms overlays the calling process with the named file, then transfers to the entry point of the image of the file. There can be no return from a successful *exec*; the calling image is lost.

Files remain open across *exec* unless explicit arrangement has been made; see *ioctl*(2). Signals that are caught (see *signal*(2)) are reset to their default values. Other signal settings are unchanged.

Each user has a *real* userid and groupid and an *effective* userid and groupid. The real userid (groupid) identifies the person using the system; the effective userid (groupid) determines access privileges. *Exec* changes the effective userid and groupid to the owner of the executed file if the file has the set-userid or set-groupid modes. The real userid is not affected.

Name points to the name of the file to be executed. It must be a regular file (type `S_IFREG`, see *stat*(2)) and its permissions must allow execution. *Arg0, arg1, ...* or the pointers in *argv* address null-terminated argument strings to be made available when the new image starts. *Argv* must end with a 0 pointer. Conventionally argument 0 is the name of the program.

In *execle*, *execve*, and *exect*, the *envp* array contains pointers to a set of null-terminated strings composing the environment of the process. *Envp* must end with a 0 pointer. The other calls copy the present environment from the global cell *environ*; see *environ*(5).

The *execl* and *execv* forms differ only in argument syntax; one is more convenient when the number of arguments is known in advance, the other when arguments are assembled on the fly.

If the first two bytes of the file are the characters `#!`, subsequent text up to a newline is examined. The first word, up to a blank or tab, names an interpreter program; anything left over is a single supplemental argument. The original *name*, preceded by the supplemental argument if any, is inserted in the argument list between *arg0* and *arg1* (or between the first pair of *argv* pointers). The interpreter is executed with the modified argument list.

If the file doesn't start with `#!`, a standard header for a binary image is expected; see *a.out*(5). If the file doesn't begin with a valid header either, `ENOEXEC` is returned. The shell *sh*(1) takes this to mean that the file contains shell commands.

When a C program is executed, it is called as follows:

```
main(argc, argv, envp)
int argc;
char **argv, **envp;
```

Argv is the array of argument pointers passed to *exec*; *argc* is the number of arguments. *Argv* is directly usable in a subsequent *execv* because `argv[argc]==0`. *Envp* is the environment array; the same value has already been stored in *environ*.

Execlp and *execvp* take the same arguments as *execl* and *execv*, but search the directories listed in the `PATH` environment variable for an executable file called *name*, mimicking the shell's path search.

Exect is the same as *execve*, except that it arranges for the process to stop just before the first instruction of the new image; see *proc*(4).

FILES

`/bin/sh` shell, invoked if command file found by *execlp* or *execvp*

EXAMPLES

This file, if created with execute permissions and run by *exec*, calls *awk*(1) to count the lines in all the files named in its arguments:

```
#!/usr/bin/awk -f
END   { print NR }
```

SEE ALSO

sh(1), *fork*(2), *ioctl*(2), *signal*(2), *proc*(4), *environ*(5)

DIAGNOSTICS

`E2BIG`, `EACCES`, `EFAULT`, `EIO`, `ELOOP`, `ENOENT`, `ENOEXEC`, `ENOMEM`, `ENOTDIR`, `ENXIO`, `ETXTBSY`

BUGS

If *execvp* is called to execute a file that turns out to be a shell command file, and the shell cannot be executed, some of the values in *argv* may be modified before return.

Neither the shell's path search nor that of *execlp* and *execvp* extends to the interpreter named after `#!`. The interpreter file may not itself begin with `#!`. The text after `#!` may be no more than 30 characters long, including the newline.

NAME

_exit, wait wait3 – terminate process, wait for child to terminate

SYNOPSIS

```
void _exit(status)
int status;

int wait(status)
int *status;

int wait((int *)0)

#include <sys/vtimes.h>

wait3(status, options, ch_vt)
int *status;
struct vtimes *ch_vt;
```

DESCRIPTION

_exit closes all the process's files and notifies the parent process when the parent executes *wait*. The low-order 8 bits of *status* are available to the parent process. The call never returns.

The function *exit*(3), which is the normal means of terminating a process, may cause cleanup actions before finally calling *_exit*. Therefore, *_exit* should be called to terminate a child process after a *fork*(2) to avoid flushing buffered output twice.

Wait delays until a signal is received or until a child processes terminates or receives signal SIGSTOP. There is no delay if any child has died since the last *wait*, or if there are no extant children. The normal return yields the process id and status of one terminated child. The status of other children may be learned from further *wait* calls.

If *status* is nonzero, *wait* sets **status* = (*s*<<8)+*t* where *s* is the low 8 bits of *status* from the child's *exit*, if any, and *t* is the termination status of the child. See *signal*(2) for a list of termination statuses (signals); status 0 indicates normal termination, 0177 a (restartable) process stopped on SIGSTOP. If the 0200 bit of the termination status is set, a core image of the process was produced by the system.

Wait3 is similar to *wait.* An *option* value of 1 prevents waiting for extant, non-stopped children and causes 0 to be returned if children exist but none have reportable status. If *ch_vt* is nonzero, resource usage data for the child are reported as by *vtimes*(2).

If the parent process terminates without waiting on its children, they are inherited by process 1 (the initialization process, *init*(8)).

SEE ALSO

fork(2), *exit*(3), *signal*(2), *sh*(1)

DIAGNOSTICS

wait, *wait3*: ECHILD

BUGS

If the argument to *wait* is bogus, the user program gets a memory fault rather than an EFAULT.
The 0 third argument to *wait3* is a required historical dreg.

NAME

fmount, funmount − mount or remove file system

SYNOPSIS

```
int fmount(type, fildes, name, flag)
char *name;

int funmount(name)
char *name;
```

DESCRIPTION

Fmount mounts a file system of the named *type* described by the file descriptor *fildes* on pathname *name*. Henceforth, references to *name* (the mount point) will refer to the root file on the newly mounted file system.

Name must already exist. Its old contents are inaccessible while the file system is mounted.

The meaning of *flag* varies with the file system type.

Allowed types are

`0` Regular (block device) file system. *Fildes* must be a block special file. If *flag* is nonzero, the file system may not be written on; this must be used with physically write-protected media or errors will occur when access times are updated, even if no explicit write is attempted.

`2` Process file system, *proc*(4). *Fildes* is ignored.

`3` Mounted stream. *Fildes* must refer to a stream; future calls to *open*(2) on *name* will reopen that stream. The mount is undone if the other end of the stream is closed or hung up.

`4` Stream (network) file system. *Fildes* is a stream connected to a file system server, *netfs*(8).

Types 5 and 6 are used internally to close off errors and for pipes; these types may not be mounted.

Funmount removes knowledge of the file system mounted at *name*. The mount point reverts to its previous interpretation.

The userid owning *name* may mount or unmount file systems of type 3 or 4. For other types, these calls are restricted to the super-user.

SEE ALSO

mount(8), *netfs*(8), *proc*(4), *stream*(4)

DIAGNOSTICS

`EBADF`, `EBUSY`, `EINVAL`, `EIO`, `ENODEV`

BUGS

Although *fildes* for type 2 file systems is ignored, it must be a valid file descriptor.

NAME

fork – spawn new process

SYNOPSIS

```
int fork()
```

DESCRIPTION

Fork is the only way new processes are created. The new process's image is a copy of that of the caller of *fork*. The only distinction is that the value returned in the old (parent) process is the process id of the new (child) process, while the value returned in the child is 0. Process ids range from 1 to 30,000. The process id is used by *wait* (see *exit*(2)) and *kill* (see *signal*(2)).

Files open before the *fork* are shared, and have a common read-write pointer. This is the way that *sh*(1) passes standard input and output files and sets up pipes.

SEE ALSO

exit(2), *signal*(2), *sh*(1)

DIAGNOSTICS

`EAGAIN`, `ENOMEM`

NAME

getuid, getgid, geteuid, getegid, getlogname, getpid, getppid, getpgrp, setuid, setgid, setruid, setlogname, setpgrp – get or set user, group, or process identity

SYNOPSIS

```
int getuid()

int geteuid()

int getgid()

int getegid()

int getlogname(buf)
char *buf;

int getpid()

int getppid()

int getpgrp(pid)
int pid;

int setuid(uid)

int setgid(gid)

int setruid(uid)

int setlogname(buf)
char buf[8];

int setpgrp(pid, pgrp)
int pid, pgrp;
```

DESCRIPTION

Getuid returns the real userid of the current process, *geteuid* the effective userid. The real userid identifies the person who is logged in, rather than the effective userid, which determines access permission at the moment. It is thus useful to set-userid programs to find out who invoked them.

Getgid returns the real groupid, *getegid* the effective groupid.

Getlogname copies the login name of the current process into the buffer pointed to by *buf*, which must be at least eight characters long.

Getpid returns the process id of the current process, *getppid* that of its parent process.

Getpgrp returns the process group id of process *pid*; 0 means the current process.

Setuid (*setgid*) sets the effective and real userid (groupid) of the current process to *uid* (*gid*). Both the effective and the real userid (groupid) are set. These calls are permitted only if the process is super-user or if the argument is the real or effective userid (groupid).

Setruid sets the real userid only. It may only be used by the super-user.

Setlogname sets the login name returned by *getlogname*. It may only be used by the super-user.

Setpgrp sets the process group id of process *pid* to *pgrp*. *Pid* 0 is the current process. Only the super-user may set the process group of processes with other userids or set a process group to 0.

SEE ALSO

getuid(1), *getlogin*(3)

DIAGNOSTICS

getlogname: `EFAULT`
setlogname: `EFAULT`, `EPERM`
setuid, *setgid*, *setruid*, *setpgrp*: `EPERM`

BUGS

Non-super-user processes may set the process group of descendant processes; only certain unsupported shells use this, and the facility may vanish presently.

NAME

ioctl – miscellaneous control operations

SYNOPSIS

```
#include <sys/filio.h>

int ioctl(fildes, request, param)
void *param;
```

DESCRIPTION

Ioctl performs a variety of *requests* on open files, most applying only to particular device files. The writeups of various devices in section 4 discuss how *ioctl* applies to them; see *stream*(4) in particular for operations applying to communication devices.

Param points to a parameter buffer. Some *requests* read data from the buffer; others write data to the buffer; some do both. The buffer's size varies according to the request. A few provide meaningful data in the return value from *ioctl* as well; others return a conventional value of 0 for success, −1 for failure.

Requests for different drivers are defined in different header files. Two standard calls, defined in `<sys/filio.h>`, apply to any open file:

```
ioctl(fildes, FIOCLEX, (void *)0);
ioctl(fildes, FIONCLEX, (void *)0);
```

The first causes the file to be closed automatically upon a successful *exec*(2); the second causes the file to be left open.

SEE ALSO

exec(2), *stream*(4)

DIAGNOSTICS

`EBADF`, `EFAULT`, `EIO`, `ENODEV`, `ENOTTY`

BUGS

Ioctl requests vary among UNIX systems; undisciplined use is likely to compromise portability.
The size of the parameter buffer should be an explicit argument.

NAME

link, symlink, readlink – link to a file

SYNOPSIS

```
int link(name1, name2)
char *name1, *name2;

int symlink(name1, name2)
char *name1, *name2;

int readlink(name, buf, size)
char *name, *buf;
```

DESCRIPTION

Link and *symlink* create a link to file *name1* with new name *name2*. Either name may be an arbitrary path name.

After *link*, *name2* is entirely equivalent to *name1*; it is a directory entry referring to the same file as *name1*. Only the super-user can make the link if *name1* is a directory.

After *symlink*, *name2* is a new symbolic link; when it is encountered in any path name, *name1* is substituted for *name2*, and path name parsing continues. If *name1* begins with the / character, it is interpreted with respect to the root directory; if not, it is interpreted with respect to the directory in which *name2* resides.

Symbolic links are slightly slower than normal links but may span file systems; normal links are confined to a single file system.

Readlink copies the pathname inside symbolic link *name* into memory at *buf*. No more than *size* bytes are copied; the actual number of bytes read is returned. The contents of *buf* will not be null-terminated. An error is returned if *name* is not a symbolic link.

SEE ALSO

cp(1), *unlink*(2), *stat*(2)

DIAGNOSTICS

all: `EIO`, `ELOOP`, `ENOENT`, `ENOTDIR`
link: `EEXIST`, `EROFS`, `EXDEV`
symlink: `EEXIST`, `EROFS`
readlink: `ENXIO`

NAME

lseek, llseek – seek, move read/write pointer

SYNOPSIS

```
long lseek(fildes, offset, whence)
long offset;

Long llseek(fildes, offset, whence)
Long offset;
```

DESCRIPTION

Lseek and *llseek* set the file pointer for the file associated with *fildes* as follows:

If *whence* is 0, the pointer is set to *offset* bytes.

If *whence* is 1, the pointer is set to its current location plus *offset*.

If *whence* is 2, the pointer is set to the size of the file plus *offset*.

The new file pointer value is returned.

Type *Long* is a 64-bit quantity.

Seeking far beyond the end of a file, then writing, creates a gap or ‘hole,’ which occupies no physical space and reads as zeros.

SEE ALSO

open(2), *fseek*(3)

DIAGNOSTICS

`EBADF`, `ESPIPE`

BUGS

Lseek doesn’t affect some special files.

NAME

mkdir, rmdir – make or remove a directory

SYNOPSIS

```
int mkdir(name, mode)
char *name;

int rmdir(name)
char *name;
```

DESCRIPTION

Mkdir creates a new directory whose name is the null-terminated string pointed to by *name*. The mode of the directory is set to *mode*, as modified by the process's mode mask; see *stat*(2) and *umask*(2). The directory initially contains two entries: `.` (a link to the directory itself) and `..` (a link to the parent directory).

Rmdir removes the directory *name*, which must have only `.` and `..` entries.

SEE ALSO

mkdir(1), *rm*(1), *mknod*(2), *stat*(2), *umask*(2)

DIAGNOSTICS

mkdir: `EEXIST`, `EFAULT`, `EIO`, `ELOOP`, `ENOENT`, `ENOTDIR`, `EROFS`
rmdir: `EFAULT`, `EHASF`, `EINVAL`, `EIO`, `ELOOP`, `ENOENT`, `ENOTDIR`, `EROFS`

NAME

mknod – make a directory or a special file

SYNOPSIS

```
int mknod(name, mode, addr)
char *name;
```

DESCRIPTION

Mknod creates a new file whose name is the null-terminated string pointed to by *name*. The mode of the new file (including directory and special file bits) is initialized from *mode*. (The protection part of the mode is modified by the process's mode mask; see *stat*(2) and *umask*(2)). The first block pointer of the inode is initialized from *addr*. For ordinary files and directories *addr* is normally zero. For a special file, *addr* is the device number; see *mknod*(8) and the writeups in section 4.

Mknod may be invoked only by the super-user.

SEE ALSO

open(2) for *creat*, *mkdir*(2), *stat*(2), *umask*(2), *filsys*(5), *mknod*(8)

DIAGNOSTICS

EEXIST, EFAULT, EIO, ELOOP, ENOENT, ENOTDIR, EPERM, EROFS

NAME

nice – set program priority

SYNOPSIS

```
void nice(incr)
```

DESCRIPTION

The scheduling priority of the process is augmented by *incr*. Positive priorities get less service than normal. Priority 10 is recommended to users who wish to execute long-running programs without flak from the administration. Priority 19 is recommended for programs that should only execute in ''idle'' time.

Only the super-user can effect negative increments. Priorities less than −20 (most urgent) are treated as −20. Priorities greater than 19 are treated as 19.

The priority of a process is passed to a child process by *fork*(2). For a privileged process to return to normal priority from an unknown state, *nice* should be called successively with arguments −40 (goes to priority −20 because of truncation), then 20 (to get to 0).

SEE ALSO

nice(1), *fork*(2)

NAME

open, creat, close – open a file for reading or writing, create file

SYNOPSIS

```
int open(file, rwmode)
char *file;

int creat(file, mode)
char *file;

int close(fildes)
```

DESCRIPTION

Open opens the *file*, for reading if *rwmode* is 0, for writing if *rwmode* is 1, or for both if *rwmode* is 2, and returns an associated file descriptor. *File* is a null-terminated string representing a path name. The file pointer is set to 0.

Creat creates a new *file* or prepares to rewrite an existing *file*, opens it for writing, and returns an associated file descriptor. If the file is new, the owner is set to the effective userid of the creating process; the group to that of the containing directory; the mode to *mode* as modified by the mode mask of the creating process; see *umask*(2). *Mode* need not allow writing. If the file already exists, it is truncated to 0 length; the mode, owner, and group remain unchanged, and must permit writing.

A program may reserve a filename for exclusive use by calling *creat* with a mode that forbids writing. If the file does not exist, *creat* will succeed; further attempts to *creat* the same file will be denied. More sophisticated (but less portable) concurrent access control may be obtained by setting the `S_ICCTYP` field in the mode; see *stat*(2).

Close closes the file associated with a file descriptor. Files are closed upon termination of a process, but since there is a limit on the number of open files per process, *close* is necessary for programs which deal with many files. It is possible to arrange for files to be closed by *exec*(2); see `FIOCLEX` in *ioctl*(2).

SEE ALSO

dup(2), *pipe*(2), *read*(2), *exec*(2), *ioctl*(2), *stat*(2), *unlink*(2)

DIAGNOSTICS

open, creat: `EACCES`, `EBUSY`, `ECONC`, `EFAULT`, `EINTR`, `EIO`, `EISDIR`, `ELOOP`, `EMFILE`, `ENFILE`, `ENOENT`, `ENOTDIR`, `EROFS`, `ETXTBSY`
creat: `ENOSPC`
close: `EBADF`

BUGS

It should be possible to call *open* without waiting for carrier on communication lines.
The group of a newly-created file should (once again) be the effective groupid of the creating process.
The trick of creating a file with an unwritable mode fails for the super-user.

NAME

pipe – create an interprocess channel

SYNOPSIS

```
int pipe(fildes)
int fildes[2];
```

DESCRIPTION

Pipe creates a buffered channel for interprocess I/O communication. Two file descriptors returned in *fildes* are the ends of pair of cross-connected streams; see *stream*(4). Data written via `fildes[1]` is available for reading via `fildes[0]` and vice versa.

After the pipe has been set up, cooperating processes created by subsequent *fork*(2) calls may pass data through the pipe with *read* and *write* calls. The bytes placed on a pipe by one *write* are contiguous even if many process are writing. *Writes* induce a record structure: a *read* will not return bytes from more than one *write*; see *read*(2).

Write calls on a one-ended pipe raise signal `SIGPIPE`. Read calls on a one-ended pipe with no data in it return an end-of-file for the first several attempts, then raise `SIGPIPE`, and eventually `SIGKILL`.

SEE ALSO

sh(1), *fork*(2), *read*(2), *select*(2), *stream*(4)

DIAGNOSTICS

`EIO`, `EMFILE`, `ENFILE`, `ENXIO`

BUGS

Buffering in pipes connecting multiple processes may cause deadlocks.

Some line discipline modules discard the record delimiters inserted by *write*.

On many other versions of the system, only `fildes[0]` may be read and only `fildes[1]` may be written.

NAME

profil – execution time profile

SYNOPSIS

```
void profil(buff, bufsiz, offset, scale)
unsigned short *buff;
int bufsiz, offset;
unsigned scale;
```

DESCRIPTION

Buff points to an area of core whose length in bytes is given by *bufsiz*. After this call, the user's program counter is examined each clock tick; *offset* is subtracted from it, and the result multiplied by *scale*, divided by 65536 and then rounded up to a multiple of two. If the resulting number (*n*) is less than *bufsiz*, then `buff[n/2]` is incremented.

Profiling is turned off by giving a *scale* of 0. It is rendered ineffective by giving a *bufsiz* of 0. Profiling is turned off when an *exec* is executed, but remains on in child and parent both after a *fork*. Profiling is turned off if an update in *buff* would cause a memory fault.

SEE ALSO

prof(1), *monitor*(3)

BUGS

Because of the rounding up, single byte instructions cannot be exactly profiled.

NAME

read, write – read or write file

SYNOPSIS

```
int read(fildes, buffer, nbytes)
char *buffer;

int write(fildes, buffer, nbytes)
char *buffer;
```

DESCRIPTION

Read reads *nbytes* bytes of data from the file pointer location in the file associated with *fildes* into memory at *buffer*. The file pointer is advanced by the number of bytes read. It is not guaranteed that all *nbytes* bytes will be read; for example if the file refers to a terminal at most one line will be returned. In any event the number of characters read is returned. A return value of 0 is conventionally interpreted as end of file.

Write writes *nbytes* bytes of data starting at *buffer* to the file associated with *fildes* at the file pointer location. The file pointer is advanced by the number of bytes written. The number of characters actually written is returned. It should be regarded as an error if this is not the same as requested.

Reads and writes which are aligned with file system blocks are more efficient than others; see *filsys*(5).

SEE ALSO

open(2), *dup*(2), *pipe*(2), *select*(2), *dirread*(2)

DIAGNOSTICS

read: `EBADF`, `EFAULT`, `EINTR`, `EINVAL`, `ENXIO`
write: `EBADF`, `EFAULT`, `EINTR`, `EINVAL`, `EIO`, `ENXIO`, `EPIPE`, `EROFS`

BUGS

A *read* or a *write* call may fail because of a prior call to *lseek*(2).

NAME

select – synchronous input/output multiplexing

SYNOPSIS

```
#include <sys/types.h>

int select(nfds, readfds, writefds, milli);
fd_set *readfds, *writefds;
```

DESCRIPTION

Select examines a set of file descriptors to see if they will block if read or written. *Readfds* points to an object of type `fd_set`, which contains a set of descriptors to be checked for reading; *writefds* similarly for writing. Only descriptors 0 through *nfds*−1 are considered. The number of ready descriptors is returned, and the `fd_set` pointed to by *readfds* (*writefds*) is overwritten with a set of descriptors ready to be read (written). The call waits until at least one descriptor is ready, or until *milli* milliseconds have passed.

Either *readfds* or *writefds* may be 0 if no descriptors are interesting.

The `fd_set` type looks like

```
typedef struct {
        unsigned int fds_bits[FDWORDS];
} fd_set;
```

`FDWORDS` is sufficient to contain as many file descriptors as the system will allow (currently 128). If there are *B* bits in an `unsigned int`, file descriptor *n* is represented by `1<<((`*n*`%`*B*`)-1)` in word `fds_bits[`*n*`/`*B*`]`.

These macros should be used to manipulate the contents of an `fd_set`:

`FD_ZERO(s)`	clear all bits in set *s*
`FD_SET(n, s)`	set bit for file descriptor *n* in set *s*
`FD_CLR(n, s)`	clear bit for file descriptor *n* in s
`FD_ISSET(n, s)`	return a value of 1 if bit for file descriptor *n* is set in *s*, 0 otherwise

EXAMPLES

```
int p[2];
fd_set wfs;
pipe(p);
do {
        FD_SET(p[1], wfs);
        write(p[1], ".", 1);
        i++;
} while(select(p[1]+1, (fd_set*)0, wfs, 0) == 1);
printf("Pipe capacity = %d\n", i);
```

SEE ALSO

read(2)

DIAGNOSTICS

`EBADF`, `EFAULT`, `EINTR`

BUGS

Milli is rounded up to the nearest second.

Select is intended for use with streams; file descriptors referring to ordinary files or to non-stream special files always appear ready. This is a lie for some special files.

NAME

signal, kill – receive and send signals

SYNOPSIS

```
#include <signal.h>

SIG_TYP signal(sig, func)
SIG_TYP func;

int kill(pid, sig)
```

DESCRIPTION

A signal is generated by some abnormal event initiated by a user at a terminal (quit, interrupt), by a program error (bus error, etc.), or by *kill* in another process. Normally, most signals cause termination of the receiving process, but *signal* allows them either to be ignored or to be caught by interrupting to a specified function. The following signal names are defined in `<signal.h>`:

`SIGHUP`	1	hangup
`SIGINT`	2	interrupt
`SIGQUIT`	3*	quit
`SIGILL`	4*	illegal instruction (not reset when caught)
`SIGTRAP`	5*	trace trap (not reset when caught)
`SIGIOT`	6*	IOT instruction
`SIGEMT`	7*	EMT instruction
`SIGFPE`	8*	floating point exception
`SIGKILL`	9	kill (cannot be caught or ignored)
`SIGBUS`	10*	bus error
`SIGSEGV`	11*	segmentation violation
`SIGSYS`	12*	bad argument to system call
`SIGPIPE`	13	write on a pipe with no one to read it
`SIGALRM`	14	alarm clock
`SIGTERM`	15	software termination signal
	16	unassigned
`SIGSTOP`	17+	stop (cannot be caught or ignored)
`SIGCONT`	19#	continue a stopped process
`SIGCHLD`	20#	child has stopped or exited

* places core image in file `core` if not caught or ignored
\+ suspends process until `SIGCONT` or `PIOCRUN`; see *proc*(4)
\# ignored if not caught

Signals 1 through `NSIG`-1, defined in the include file, exist. Those not listed above have no conventional meaning in this system. (Berkeley systems use 1-15 and 17-25.)

Signal specifies how signal *sig* will be handled. If *func* is `SIG_DFL`, the default action listed above is reinstated. If *func* is `SIG_IGN`, the signal will be ignored. Otherwise, when the signal occurs, it will be caught and a function, pointed to by *func*, will be called. The type of pointer *func* is `SIG_TYP`:

```
typedef int (*SIG_TYP)();
```

It must point to a function such as,

```
int catcher(sig) { ... }
```

which will be called with a signal number as argument. A return from the catcher function will continue the process at the point it was interrupted.

Except as indicated, a signal is reset to `SIG_DFL` after being caught. Thus if it is desired to catch every such signal, the catching routine must issue another *signal* call.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during *read*(2) or *write* on a slow device (like a typewriter, but not a disk), and during

pause and *wait*; see *alarm*(2) and *exit*(2). The interrupted system call will return error `EINTR`. The user's program may then, if it wishes, re-execute the call.

Signal returns the previous (or initial) value of *func* for the particular signal.

After a *fork*(2) the child inherits all signal settings. *Exec*(2) resets all caught signals to default action.

Kill sends signal *sig* to the process specified by process id *pid.* Signal 0 has no effect on the target process and may be used to test the existence of a process. The success of sending a signal is independent of how the receiving process treats the signal.

The effective userid of the sending process must be either 0 or the effective userid of the receiving process.

If *pid* is 0, the signal is sent to all other processes in the sender's process group; see *stream*(4).

If *pid* is −1, and the user is the super-user, the signal is broadcast universally except to processes 0 (scheduler), 1 (initialization) and 2 (pageout); see *init*(8). If *pid* is less than −1, it is negated and taken as a process group whose members should receive the signal.

Processes may send signals to themselves.

FILES

`core`

SEE ALSO

kill(1), *setjmp*(3), *stream*(4)

DIAGNOSTICS

signal: `EINVAL`
kill: `EINVAL`, `EPERM`, `ESRCH`

BUGS

The reason for a trap should be distinguishable by extra arguments to the signal handler.
If a repeated signal arrives before the last one can be reset, there is no chance to catch it.
For historical reasons, the return value of a catcher function is `int`; it is `void` in ANSI standard C.

NAME

stat, lstat, fstat – get file status

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(name, buf)
char *name;
struct stat *buf;

int lstat(name, buf)
char *name;
struct stat *buf;

int fstat(fildes, buf)
struct stat *buf;
```

DESCRIPTION

Stat puts detailed information about the file *name* in a structure whose address is *buf*. *Lstat* does the same except that when *name* is a symbolic link (see *link*(2)), it supplies information about the link itself. *Fstat* does what *stat* does for the file open on descriptor *fildes*.

It is unnecessary to have any permissions at all with respect to *name*, but all directories leading to the file must be searchable.

```
struct stat
{
        dev_t st_dev;             device number for this file system
        ino_t st_ino;             inode number
        unsigned short st_mode;   file mode encoded as below
        short st_nlink;           number of links (not symbolic links)
        short st_uid;             uid of owner
        short st_gid;             gid of owner
        dev_t st_rdev;            if device file, the device number
        off_t st_size;            size in bytes
        time_t st_atime;          time file was last read or created
        time_t st_mtime;          time file was last written or created
        time_t st_ctime;          time file or inode was last written or created
};
```

The bits in `st_mode` are defined by

```
S_IFMT   0170000  file type
S_IFDIR  0040000  directory
S_IFCHR  0020000  character device
S_IFBLK  0060000  block device
S_IFREG  0100000  regular file
S_IFLNK  0120000  symbolic link
S_ISUID  0004000  set userid on execution
S_ISGID  0002000  set groupid on execution
S_ICCTYP 0007000  type of concurrency control
S_ISYNC  0001000  1 writer and n readers (synchronized access)
S_IEXCL  0003000  1 writer or n readers (exclusive access)
         0000400  read permission by owner
         0000200  write permission by owner
         0000100  execute permission (search on directory) by owner
         0000070  read, write, execute (search) by group
         0000007  read, write, execute (search) by others
```

`S_IFMT` and `S_ICCTYP` are field masks; the other constants encode modes. Codes contained in the `S_IFMT` field are mutually exclusive. If bit `01000` is set, the concurrency modes contained in `S_ICCTYP` are in effect; otherwise the set-id flags `S_ISUID` and `S_ISGID` apply.

The concurrency modes affect *open* and *creat* calls. Synchronized access, `S_ISYNC`, guards against inconsistent updates by forbidding concurrent opens for writing. Exclusive access, `S_IEXCL`, guards against inconsistent views by forbidding concurrent opens if one is for writing.

SEE ALSO

chmod(1), *ls*(1), *chmod*(2), *filsys*(5)

DIAGNOSTICS

stat, *lstat*: `EACCES`, `EFAULT`, `EIO`, `ELOOP`, `ENOENT`, `ENOTDIR`
fstat: `EBADF`, `EFAULT`, `EIO`

BUGS

For efficiency, `st_atime` is not set when a directory is searched, although this might be more logical.

NAME

stime, biasclock – set time

SYNOPSIS

```
int stime(tp)
long *tp;

biasclock(milli)
long milli;
```

DESCRIPTION

Stime sets the system's idea of the time and date. Time, pointed to by *tp*, is measured in seconds from 00:00:00 GMT Jan 1, 1970.

Biasclock informs the system that its idea of the time should be incremented by *milli* milliseconds. The system will make the adjustment gradually and without causing time to run backwards.

Only the super-user may use these calls.

SEE ALSO

date(1), *wwv*(1), *time*(2), *ctime*(3)

DIAGNOSTICS

EPERM

NAME

sync − force writing of system buffers

SYNOPSIS

```
void sync()
```

DESCRIPTION

Sync schedules all information in memory that should be on disk or other block media to be written out. This includes modified super-blocks, modified inodes, and delayed block I/O.

It should be used by programs which examine a file system, for example *icheck*(8). It is highly recommended before a shutdown or reboot.

SEE ALSO

sync(8)

BUGS

The writing, although scheduled, is not necessarily complete upon return from *sync*.

NAME

syscall – indirect system call

SYNOPSIS

```
int syscall(number, arg, ... )
```

DESCRIPTION

Syscall performs the system call with the specified *number* and arguments and returns its result. The numbers may be found in the system source.

BUGS

The simulation fails for system calls such as *pipe*(2), which return multiple values.

NAME

time, ftime – get date and time

SYNOPSIS

```
long time((long *)0)

long time(tloc)
long *tloc;

#include <sys/types.h>
#include <sys/timeb.h>
ftime(tp)
struct timeb *tp;
```

DESCRIPTION

Time returns the time since the epoch 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

If *tloc* is nonnull, the return value is also stored in the place to which *tloc* points.

Ftime stores a more accurate time and other horological data in the structure pointed to by *tb*:

```
struct timeb
{
        time_t time;            time since the epoch in seconds
        unsigned short millitm; up to 1000 milliseconds of more-precise interval
        short timezone;         local time zone measured in minutes of time
                                westward from Greenwich
        short dstflag;          if nonzero, daylight saving time applies locally
                                during the appropriate part of the year
};
```

SEE ALSO

date(1), *stime*(2), *ctime*(3)

DIAGNOSTICS

ftime: `EFAULT`

BUGS

If the argument to *time* is bogus, the user program gets a memory fault rather than an `EFAULT`.

NAME

times – get process times

SYNOPSIS

```
#include <sys/types.h>
#include <sys/times.h>

int times(buffer)
struct tms *buffer;
```

DESCRIPTION

Times delivers time-accounting information for the current process and for the terminated child processes of the current process. All times are in clock ticks.

```
struct tms
{
        time_t tms_utime;     user time for this process
        time_t tms_stime;     system time for this process
        time_t tms_cutime;    user time for all child processes
        time_t tms_cstime;    system time for all child processes
};
```

The children times are the sum of the children's process times and their children's times.

SEE ALSO

time(1), *time*(2)

DIAGNOSTICS

EFAULT

NAME

umask – set file creation mode mask

SYNOPSIS

```
int umask(complmode)
```

DESCRIPTION

Umask sets the process mode mask. The mask modifies the *mode* argument of *creat* (see *open*(2)), *mkdir*(2), and *mknod*(2) thus:

```
mode &= (07777 & ~(complmode & 0777))
```

In other words, the mask specifies permission bits to be turned off when files are created.

The previous value of the mask is returned by the call. The initial value is set by *login*(8), and may be modified by the *umask* command of *sh*(1). The mask is inherited by child processes.

SEE ALSO

open(2), *mkdir*(2), *mknod*(2), *stat*(2)

NAME

unlink – remove directory entry

SYNOPSIS

```
int unlink(name)
char *name;
```

DESCRIPTION

Unlink removes the entry for the file pointed to by *name* from its directory. If this entry was the last link to the file, the contents of the file are freed and the file is destroyed. If, however, the file was open in any process, the actual destruction is delayed until it is closed, even though the directory entry has disappeared.

Only the super-user can *unlink* a directory, but see *rmdir* in *mkdir*(2).

SEE ALSO

rm(1), *link*(2), *mkdir*(2)

DIAGNOSTICS

EFAULT, EIO, ELOOP, ENOENT, ENOTDIR, EROFS

NAME

vtimes − get usage of time, space, and paging resources

SYNOPSIS

```
#include <sys/vtimes.h>

vtimes(par_vm, ch_vm)
struct vtimes *par_vm, *ch_vm;
```

DESCRIPTION

Vtimes places accounting information for the current process in the area pointed to by *par_vm* and for its terminated children in the area pointed to by *ch_vm*. If either pointer is 0, the corresponding information is omitted.

After the call, each area contains information in the form

```
struct vtimes {
        int      vm_utime;  /* user time */
        int      vm_stime;  /* system time */
                            /* rss = resident storage size in 512-byte pages */
        unsigned vm_idsrss; /* rss time integral, data+stack */
        unsigned vm_ixrss;  /* rss time integral, text */
        int      vm_maxrss; /* maximum rss */
        int      vm_majflt; /* major page faults */
        int      vm_minflt; /* minor page faults */
        int      vm_nswap;  /* number of swaps */
        int      vm_inblk;  /* block reads */
        int      vm_oublk;  /* block writes */
};
```

Times are expressed in clock ticks of 1/60 (or 1/50) second. The time integrals are computed by cumulating the number of 512-byte pages in use at each clock tick.

A major page fault involves a disk transfer; a minor fault gathers page-reference information. Block reads and writes are file system disk transfers; blocks found in the buffer pool are not counted.

SEE ALSO

time(2), *exit*(2)

3 SUBROUTINES

3 SUBROUTINES

NAME

intro – introduction to library functions

SYNOPSIS

```
#include <libc.h>

#include <stdio.h>

#include <math.h>
```

DESCRIPTION

This section describes functions that may be found in various libraries, other than the system calls described in section 2. Functions are divided into various libraries distinguished by the section number at the top of the page:

(3) These functions, together with those of section 2 and those marked (3S) and (3M), constitute library *libc*, which is automatically loaded by the C compiler *cc*(1) and the Fortran compiler *f77*(1). The same functions appear also in *libC*, which is automatically loaded by the C++ compiler; see *c++*(1). The link editor *ld*(1) searches this library under option `-lc` (`-lC` for *libC*). Declarations for some of these functions may be obtained from include files indicated on the appropriate pages. Other declarations can be found in `<libc.h>`.

(3F) These functions are in the Fortran library, *libF77*, automatically loaded by the Fortran compiler, and searched under option `-lF77` of the link editor.

(3M) These functions constitute the math library, part of *libc*. (On some other systems they must be loaded by `-lm`). Declarations for these functions may be obtained from the include file `<math.h>`.

(3S) These functions constitute the 'standard IO package' (see *stdio*(3)) part of *libc* already mentioned. Declarations for these functions may be obtained from the include file `<stdio.h>`.

(3X) Various specialized libraries have not been given distinctive captions. Files in which such libraries are found are named on appropriate pages.

(3+) C++ functions in *libC* that are not in *libc*.

FILES

`/lib/libc.a`

SEE ALSO

stdio(3), *nm*(1), *ld*(1), *cc*(1), *c++*(1), *f77*(1), *intro*(2)

DIAGNOSTICS

Functions in the math library (3M) may return conventional values when the function is undefined for the given arguments or when the value is not representable. In these cases the external variable *errno* (see *intro*(2)) is set to the value `EDOM` or `ERANGE`, defined in the include file `<math.h>`.

NAME

abort − generate a fault

SYNOPSIS

```
int abort()
```

DESCRIPTION

Abort sends the current process a `SIGIOT` signal, which normally terminates the process with a core dump.

SEE ALSO

adb(1), *signal*(2), *exit*(2)

DIAGNOSTICS

Usually ‘abort − core dumped’ from the shell.

NAME

abs, sgn, gcd, lcm, min, max, labs – integer arithmetic functions: absolute value, sign, greatest common divisor, least common multiple, minimum, maximum

SYNOPSIS

```
int abs(a)

int sgn(a)

int gcd(a, b)

long lcm(a, b)

int min(a, b)

int max(a, b)

long labs(a)
long a;
```

DESCRIPTION

Abs returns the absolute value of *a.*

Sgn returns −1, 0, 1, if *a*<0, *a*=0, *a*>0, respectively.

Gcd returns the greatest common divisor of *a* and *b.* More precisely, *gcd* returns the largest machine-representable generator of the ideal generated by *a* and *b.* This means that `gcd(0,0)` = 0, and `gcd(N,0)` = `gcd(N,N)` = `N`, where `N` is the most negative integer.

Lcm returns the least common multiple of *a* and *b.* When the result is representable, it satisfies `abs(a*b)== lcm(a,b)*gcd(a,b).`

Min (*max*) returns the minimum (maximum) of *a* and *b.*

SEE ALSO

floor(3) for *fabs*

DIAGNOSTICS

Abs returns the most negative integer when the true result is unrepresentable.

There are no guarantees about the value of *lcm* when the true value is unrepresentable.

BUGS

The result of *lcm* is undefined when it doesn't fit in a long.
Labs, provided for ANSI compatibility, is lonely; there is no *lsign*, *lmax,* etc.

NAME

assert – assertion checking

SYNOPSIS

```
#include <assert.h>

void assert(expression);
```

DESCRIPTION

Assert is a macro that indicates *expression* is expected to be nonzero at this point in the program. It causes an *abort*(3) with a diagnostic comment on the standard output when *expression* is zero. Compiling with the *cc*(1) option `-DNDEBUG` effectively makes the expression always nonzero.

DIAGNOSTICS

'Assertion failed: file *f* line *n*', where *f* is the source file and *n* the source line number of the *assert* statement.

NAME

atof, atoi, atol, strtod, strtol, strtoul – convert ASCII to numbers

SYNOPSIS

```
double atof(nptr)
char *nptr;

int atoi(nptr)
char *nptr;

long atol(nptr)
char *nptr;

double strtod(nptr, rptr)
char *nptr, **rptr;

long strtol(nptr, rptr, base)
char *nptr, **rptr;

unsigned long strtoul(nptr, rptr, base)
char *nptr, **rptr;
```

DESCRIPTION

Atof, *atoi*, and *atol* convert a string pointed to by *nptr* to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

Atof recognizes an optional string of tabs and spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional `e` or `E` followed by an optionally signed integer.

Atoi and *atol* recognize an optional string of tabs and spaces, then an optional sign, then a string of decimal digits.

Strtod, *strtol*, and *strtoul*, behave similarly to *atof*, and *atol* and, if *rptr* is not zero, set **rptr* to point to the input character immediately after the string converted.

Strtol and *strtoul* interpret the digit string in the specified *base*, from 2 to 36, each digit being less than the base. Digits with value over 9 are represented by letters, a-z or A-Z. If *base* is 0, the input is interpreted as an integral constant in the style of C (with no suffixed type indicators): numbers are octal if they begin with `0`, hexadecimal if they begin with `0x` or `0X`, otherwise decimal. *Strtoul* does not recognize signs.

SEE ALSO

scanf(3)

DIAGNOSTICS

Zero is returned if the begining of the input string is not interpretable as a number.

If overflow is detected by *atof*, *strtod*, *strtol*, or *strtoul*, a maximum value of the correct sign is returned and *errno* is set to `ERANGE`.

BUGS

Atoi and *atol* have no provisions for overflow.

NAME

besj0, besj1, besjn, besy0, besy1, besyn – bessel functions

SYNOPSIS

```
#include <math.h>

double besj0(x)
double x;

double besj1(x)
double x;

double besjn(n, x)
double x;

double besy0(x)
double x;

double besy1(x)
double x;

double besyn(n, x)
double x;
```

DESCRIPTION

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS

Negative arguments cause *besy0*, *besy1*, and *besyn* to return a huge negative value and set *errno* to EDOM.

NAME

bits − variable length bit strings

SYNOPSIS

```
#include <Bits.h>

typedef unsigned long Bits_chunk;

struct Bits {
      Bits() { }
      Bits(Bits_chunk, unsigned = 1);
      Bits(const Bits&)
      ~Bits();
      Bits& operator= (const Bits&);       // also = &= |= ^=
      Bits& operator<<= (int);             // also >>=
      int operator[] (unsigned);
      operator Bits_chunk();
      unsigned size();
      unsigned size(unsigned);
      Bits& compl();
      Bits& concat(const Bits&);
      Bits& set(unsigned, unsigned long = 1);
      Bits& reset(unsigned);
      Bits& compl(unsigned);
      unsigned count();
      unsigned signif();
      unsigned trim();
};

Bits operator~ (const Bits&);
Bits operator& (const Bits&, const Bits&); // also | ^
Bits operator<< (const Bits&, int);         // also >>
int operator< (const Bits&, const Bits&);  // also > <= >= == !=
```

DESCRIPTION

A `Bits` object contains a variable-length bit string. The bits of a `Bits` object *b* are numbered from 0 through *b*`.size()`−1, with the rightmost bit numbered 0.

`Bits_chunk` is the largest unsigned integral type acceptable for conversion to and from `Bits`, `unsigned long` in this implementation.

Constructors

`Bits()` An empty bit string.

`Bits(`*n*`)` The bits are copied from the binary representation of *n* with the ones-digit of *n* becoming bit 0. Leading zero-bits are removed, except that `Bits(0)` is a one-bit string.

`Bits(`*n*`,`*m*`)` The same, but padded with leading zeros to size *m* if necessary.

Operators

Bit strings have value semantics; assigning a `Bits` object or passing it to or returning it from a function creates a copy of its value. The meanings of operators are mostly predictable from C.

Under binary and comparison operators, the shorter operand is considered to be padded on the left with zeros to the length of the longer. If, after padding, two strings of different length compare equal, the shorter is deemed the smaller.

Negative shift amounts reverse the sense of shift operators.

Under conversion (or assignment) to a `Bits_chunk`, a `Bits` is interpreted as an unsigned integer. If it has a value small enough to fit, that value is assigned. Otherwise, a non-zero value is assigned. Thus

a `Bits` is considered ‘true’ in an `if` test if it contains any one-bit, ‘false’ otherwise.

a`[`*s*`]` Bit number *s* of *a*; 0 if *s* is out of bounds.

Other functions

a`.size(`*s*`)` Set the size of *a* to *s* by truncating or padding with zeros on the left as necessary. Return the new size.

a`.compl()` Complement the bits of *a*. Return *a*.

a`.set(`*s*`)`
a`.reset(`*s*`)`
a`.compl(`*s*`)` Set, reset, or complement bit *s* of *a*. No effect if *a*`.size()<=`*s*. Return *a*.

a`.set(`*s*`,`*n*`)` If *n* is 0, reset bit *s* of *a*, otherwise set bit *s*. Equivalent to `n? a.set(s): a.reset(s)`.

a`.count()` Return the number of one-bits in *a*.

a`.signif()` Return the number of significant bits in `a`: one more than the number of the leftmost one-bit, or zero if there is no one-bit.

a`.trim()` Discard high-order zero-bits. Equivalent to `a.size(a.signif())`.

a`.concat(`*b*`)` Concatenate the value of *b* to the right (low-order) end of *a*. Return *a*.

`concat(`*a*`,`*b*`)`
Return a newly created concatenated object.

DIAGNOSTICS

An operation that runs out of memory sets the length of the affected `Bits` to zero.

BUGS

Too bad C++ can’t support `a[s] = n`.
Things would be more consistent if `Bits(0).size()` were zero.

NAME

block − adjustable arrays

SYNOPSIS

```
#include <Block.h>

Blockdeclare(T)
Blockimplement(T)

struct Block(T) {
      Block(T)(unsigned = 0);
      Block(T)(const Block(T&);
      ~Block(T);
      Block(T)& operator= (const Block(T)&);
      T& operator[] (int);
      operator T* ();
      unsigned size();
      unsigned size(unsigned);
      T* end();
      void swap(const Block(T)&);
}
```

DESCRIPTION

A `Block(T)` is an array of zero or more *elements* of type *T*, where *T* is a type name. *T* must have assignment (`operator=`) and initialization (`T(T&)`) operations.

The macro call `Blockdeclare(T)` declares the class `Block(T)`. It must appear once in every source file that uses type `Block(T)`. The macro call `Blockimplement(T)` defines the functions that implement the block class. It must appear exactly once in the entire program.

New elements are initialized to the value of an otherwise uninitialized static object of type *T*.

Constructors

`Block(T)` An empty block.

`Block(T)(n)`
A block of *n* elements.

`Block(T)(b)`
A new block whose elements are copies of the elements of `b`.

Operations

Assignment copies elements and size.

b`[`*k*`]` A reference to element `k` of block *b*; undefined if *k* is out of bounds.

`(T*)`*b* A pointer to the first element of block *b*.

Other functions

b`.size()` Return the number of elements in *b*.

b`.size(`*n*`)` Set the size of *b* to `n`. If the new size is greater than the old. Otherwise, *n* old elements are kept. Return the new size.

b`.reserve(`*n*`)`
The size of *b* is increased, if necessary, to some value greater than *n*. If *b* already has room, *b* is not changed. Return zero if memory could not be allocated and non-zero otherwise.

b`.end()` Returns a pointer to just past the last element in `b`. Equivalent to `(T*)b+b.size()`.

a`.swap(`*b*`)` The memory associated with blocks *a* and *b* is exchanged.

Performance

Most operations are implemented by the obvious uses of the `new` and `delete` operators. *Reserve* checks the size inline. If it isn't big enough, the size is increased by multiplying by 3/2 (and adding one) enough times to increase it beyond *n* .

EXAMPLES

```
Blockdeclare(long)
unsigned n = 0;
Block(long) b;
long x;
while (cin >> x) {
      b.reserve(n);
      b[n++] = x;
}
```

SEE ALSO

malloc(3), *map*(3)

DIAGNOSTICS

The only error detected is running out of memory; this is indicated in all cases by setting the size of the block for which allocation failed to zero.

BUGS

Elements are copied during reallocation by using `T::operator=` instead of `T(T&)`.

Because the 'type parameter' *T* is implemented by the C preprocessor, white space is forbidden inside the parentheses of `Block(`*T*`)`.

NAME

bopen, bclose, bseek, bfirst, bkey, breclen, bread, bdelete, bwrite – compressed B-tree subroutines

SYNOPSIS

```
#include <cbt.h>

bfile *bopen(name, typ) char *name;

void bclose(b) bfile *b;

bseek(b, key) bfile *b; mbuf key;

bfirst(b) bfile *b;

mbuf bkey(b) bfile *b;

breclen(b) bfile *b;

bread(b, key, val) bfile *b; mbuf *key, *val;

bdelete(b, key) bfile *b; mbuf key;

bwrite(b, key, val) bfile *b; mbuf key, val;
```

DESCRIPTION

These functions manipulate files of key/value records. Such files are created by `cbt creat`; see *cbt*(1). To load the functions use the *ld*(1) option `-lcbt`.

The records occur sorted (increasing lexicographical order) by their keys, which must be distinct. Both keys and values are arrays of characters accessed through the structure

```
typedef struct {
        char *mdata; address of data bytes
        short mlen;  number of data bytes
} mbuf;
```

Bopen attempts to open a named B-tree, and if successful establishes a read pointer pointing to the beginning of the file and returns a pointer to be used in calling the other routines. *Typ* is 0 for read-only or 2 for read-write. *Bopen* returns a descriptor that identifies the file to the other functions.

Bclose closes a B-tree.

Bseek positions the read pointer of the file to the record whose key is the first not less than *key*. The routine returns 1 if *key* is in the file, `EOF` if *key* is greater than any key in the file, and 0 otherwise.

Bfirst sets the read pointer to the beginning of the file. It has the same error return as `bseek`.

Bkey returns the current key. The element *mdata* of the returned structure is 0 on errors, otherwise it points to a static buffer.

Breclen returns the length of the value part of the current record.

Bread reads the value at the read pointer into the space pointed to by `val->mdata`, places its length in `val->mlen`, and advances the read pointer to the record with the next greater key. If *key* is not 0 the key of the record is read into the space pointed to by `key->mdata` and its length is placed in `key->mlen`. *Bread* returns 0 if successful.

Bdelete removes the record with the given key, returning 1 if the record was found, −1 if there was an error, and 0 otherwise, The read pointer is left undefined.

Bwrite writes the given value with the given key. An existing record with that key will be replaced. The read pointer is left undefined.

FILES

name`.T`
name`.F`

SEE ALSO

cbt(1), *dbm*(3)

DIAGNOSTICS

Routines which return pointers return 0 on errors, routines which return integers return −1.

BUGS

The length of any key is limited to 255.

The `mbuf` arguments are passed inconsistently to the routines, sometimes by value and sometimes by reference.

Cbt files are not directly portable between big-endian and little-endian machines.

NAME

chrtab – simple character bitmaps

SYNOPSIS

```
extern char chrtab[95][16];
```

DESCRIPTION

Chrtab contains 8-by-16 bitmaps for ASCII printing characters. The 16 bytes pointed to by `chrtab[c-' ']` are the 16 rows of character *c* from top to bottom. The most significant bit is the leftmost bit. The bottom row is always empty (has all bits 0).

SEE ALSO

font(9.5)

BUGS

The bitmaps of *chrtab* are intended for use with line printers, not bitmap devices.

NAME

crypt, setkey, encrypt – DES encryption

SYNOPSIS

```
char *crypt(key, salt)
char *key, *salt;

setkey(key)
char *key;

encrypt(block, edflag)
char *block;
```

DESCRIPTION

Crypt is the password encryption routine. It is based on the NBS Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to *crypt* is a user's typed password. The second is a 2-character string chosen from the set `[a–zA–Z0–9./]`. The *salt* string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The other functions provide (rather primitive) access to the actual DES algorithm. The argument of *setkey* is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine.

The argument to *encrypt* is also a character array of length 64 containing 0's and 1's. The 64 argument 'bits' are encrypted in place by the DES algorithm using the key previously set by *setkey*. If *edflag* is 0, the argument is encrypted; if non-zero, it is decrypted.

SEE ALSO

crypt(1), *passwd*(1), *passwd*(5), *getpass*(3)

BUGS

The return value points to static data whose content is overwritten by each call.
Encrypt is not available outside the United States and Canada.

NAME

ctime, localtime, gmtime, asctime, timezone − convert date and time to ASCII

SYNOPSIS

```
#include <time.h>

char *ctime(clock)
long *clock;

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *timezone(zone, dst)
```

DESCRIPTION

Ctime converts a time pointed to by *clock* such as returned by *time*(2) into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
Sun Sep 16 01:03:52 1973\n\0
```

Localtime and *gmtime* return pointers to structures containing the broken-down time. *Localtime* corrects for the time zone and possible daylight savings time; *gmtime* converts directly to GMT, which is the time UNIX uses. *Asctime* converts a broken-down time to ASCII and returns a pointer to a 26-character string.

```
struct tm {
        int tm_sec;    seconds (range 0..59)
        int tm_min;    minutes (0..59)
        int tm_hour;   hours (0..23)
        int tm_mday;   day of the month (1..31)
        int tm_mon;    month of the year (0..11)
        int tm_year;   year A.D. − 1900
        int tm_wday;   day of week (0..6, Sunday = 0)
        int tm_yday;   day of year (0..365)
        int tm_isdst; zero means normal time, nonzero means daylight saving time
};
```

When local time is called for, the program consults the system to determine the time zone and whether the standard U.S.A. daylight saving time adjustment is appropriate. The peculiarities of this conversion are read from the file `/lib/dst`, which contains lines of the form

y0 y1 bmon bday boff emon eday eoff

meaning that for years between *y0* and *y1* inclusive, daylight saving time begins (ends) *boff* (*eoff*) days after the first Sunday after the day *bmon*/*bday* (*emon*/*eday*).

Timezone returns the name of the time zone associated with its first argument, which is measured in minutes westward from Greenwich. If the second argument is 0, the standard name is used, otherwise the Daylight Saving version. If the required name does not appear in a table built into the routine, the difference from GMT is produced. Thus, as Afghanistan is 4:30 ahead of GMT, `timezone(-(60*4+30), 0)` returns `"GMT+4:30"`.

SEE ALSO

time(2), *timec*(3)

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii – character classification

SYNOPSIS

```
#include <ctype.h>
isalpha(c)
isupper(c)
islower(c)
isdigit(c)
isxdigit(c)
isalnum(c)
isspace(c)
ispunct(c)
isprint(c)
isgraph(c)
iscntrl(c)
isascii(c)
```

DESCRIPTION

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *Isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value `EOF`; *stdio*(3).

isalpha *c* is a letter, a-z or A-Z

isupper *c* is an upper case letter, A-Z

islower *c* is a lower case letter, a-z

isdigit *c* is a digit, 0-9

isxdigit *c* is a hexadecimal digit, 0-9 or a-f or A-F

isalnum *c* is an alphanumeric character, a-z or A-Z or 0-9

isspace *c* is a space, horizontal tab, vertical tab, carriage return, newline, or formfeed (040, 011, 012, 013, 014, 015)

ispunct *c* is a punctuation character (one of `!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~`)

isprint *c* is a printing character, 040 (space) through 0176 (tilde)

isgraph *c* is a visible printing character, 041 (exclamation) through 0176 (tilde)

iscntrl *c* is a delete character, 0177, or ordinary control character, 000 through 037

isascii *c* is an ASCII character, 000 through 0177

SEE ALSO

tolower(3), *ascii*(6)

NAME

curses – screen functions with 'optimal' cursor motion

DESCRIPTION

These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen, and the user sets up an image of a new one. Then *refresh()* tells the routines to make the current screen look like the new one. The initialization routine *initscr()* must be called before any other routines that deal with windows and screens. The routine *endwin()* should be called before exiting.

To load the functions use the *ld*(1) options `-lcurses -ltermcap`.

SEE ALSO

ioctl(2), *termcap*(5)
Ken Arnold, 'Screen Updating and Cursor Movement Optimization: A Library Package', *UNIX Programmer's Manual*, Seventh Edition, Virtual VAX-11 Version, 1980 (Berkeley)

FUNCTIONS

`addch(ch)`	add a character to *stdscr*
`addstr(str)`	add a string to *stdscr*
`box(win,vert,hor)`	draw a box around a window
`crmode()`	set cbreak mode
`clear()`	clear *stdscr*
`clearok(scr,boolf)`	set clear flag for *scr*
`clrtobot()`	clear to bottom on *stdscr*
`clrtoeol()`	clear to end of line on *stdscr*
`delwin(win)`	delete *win*
`echo()`	set echo mode
`endwin()`	end window modes
`erase()`	erase *stdscr*
`getch()`	get a char through *stdscr*
`getstr(str)`	get a string through *stdscr*
`gettmode()`	get tty modes
`getyx(win,y,x)`	get (y,x) co-ordinates
`inch()`	get char at current (y,x) co-ordinates
`initscr()`	initialize screens
`leaveok(win,boolf)`	set leave flag for *win*
`longname(termbuf,name)`	get long name from *termbuf*
`move(y,x)`	move to (y,x) on *stdscr*
`mvcur(lasty,lastx,newy,newx)`	actually move cursor
`newwin(lines,cols,begin_y,begin_x)`	create a new window
`nl()`	set newline mapping
`nocrmode()`	unset cbreak mode
`noecho()`	unset echo mode
`nonl()`	unset newline mapping
`noraw()`	unset raw mode
`overlay(win1,win2)`	overlay win1 on win2
`overwrite(win1,win2)`	overwrite win1 on top of win2
`printw(fmt,arg1,arg2,...)`	printf on *stdscr*
`raw()`	set raw mode
`refresh()`	make current screen look like *stdscr*
`resetty()`	reset tty flags to stored value
`savetty()`	stored current tty flags
`scanw(fmt,arg1,arg2,...)`	scanf through *stdscr*
`scroll(win)`	scroll *win* one line
`scrollok(win,boolf)`	set scroll flag

`setterm(name)`	set term variables for name
`standend()`	end standout mode
`standout()`	start standout mode
`subwin(win,lines,cols,begin_y,begin_x)`	create a subwindow
`touchwin(win)`	'change' all of *win*
`unctrl(ch)`	printable version of *ch*
`waddch(win,ch)`	add char to *win*
`waddstr(win,str)`	add string to *win*
`wclear(win)`	clear *win*
`wclrtobot(win)`	clear to bottom of *win*
`wclrtoeol(win)`	clear to end of line on *win*
`werase(win)`	erase *win*
`wgetch(win)`	get a char through *win*
`wgetstr(win,str)`	get a string through *win*
`winch(win)`	get char at current (y,x) in *win*
`wmove(win,y,x)`	set current (y,x) co-ordinates on *win*
`wprintw(win,fmt,arg1,arg2,...)`	printf on *win*
`wrefresh(win)`	make screen look like *win*
`wscanw(win,fmt,arg1,arg2,...)`	scanf through *win*
`wstandend(win)`	end standout mode on *win*
`wstandout(win)`	start standout mode on *win*

NAME

dbminit, fetch, store, delete, firstkey, nextkey – database subroutines

SYNOPSIS

```
dbminit(file)
char *file;

datum fetch(key)
datum key;

store(key, value)
datum key, value;

delete(key)
datum key;

datum firstkey()

datum nextkey(key)
datum key;
```

DESCRIPTION

These functions maintain key/value pairs (each pair is a *datum*) in a data base. The functions will handle very large databases in one or two file system accesses per key. The functions are loaded with *ld*(1) option `-ldbm`. A datum is defined as

```
typedef struct {
        char   *dptr;
        int    dsize;
} datum;
```

A `datum` object specifies a string of `dsize` bytes pointed to by `dptr`. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has `.dir` as its suffix. The second file contains all data and has `.pag` as its suffix.

Before a database can be accessed, it must be opened by *dbminit.* At the time of this call, the files *file*`.dir` and *file*`.pag` must exist. (An empty database has empty `.dir` and `.pag` files.)

The value associated with a key is retrieved by *fetch* and assigned by *store*. A key and its associated value are deleted by *delete*. A linear pass through all keys in a database may be made, in random order, by use of *firstkey* and *nextkey*. *Firstkey* will return the first key in the database. With any key *nextkey* will return the next key in the database. This code will traverse the data base:

```
for(key = firstkey(); key.dptr != NULL; key = nextkey(key))
```

SEE ALSO

cbt(3)

DIAGNOSTICS

All functions that return integers indicate errors with negative values. A zero return indicates success. Routines that return a `datum` indicate errors with zero `dptr`.

BUGS

The `.pag` file contains holes; its apparent size is about four times its actual content. These files cannot be copied by normal means (*cat*(1), *tar*(1), *cpio*(1), *ar*(1)) without filling in the holes.

Pointers returned by these subroutines refer to static data that is changed by subsequent calls.

The sum of the sizes of a key/value pair must not exceed a fixed internal block size. Moreover all key/value pairs that hash together must fit on a single block. *Store* will return an error in the event that a disk block fills with inseparable data.

Delete does not physically reclaim file space, although it does make it available for reuse.

NAME

dialout – place call on ACU

SYNOPSIS

```
int dialout(telno, class)
char *telno, *class;
```

DESCRIPTION

Dialout places a data call via an automatic calling unit directly attached to the calling computer. To use an ACU on Datakit see the example in *ipc*(3).

Dialout searches for an ACU of the appropriate service class and places a data call on the associated line, using the given telephone number. If successful, it returns an open file descriptor for the line. The file is in raw mode, and has exclusive-use and hangup-on-close modes set. It returns −1 if all ACUs of the given class are busy, −3 if carrier could not be set, and −9 if the service class is unidentifiable.

The routine consults a data file `/etc/aculist` that consists of lines containing six blank- or tab-separated fields.

service class Each line with a service class matching the one specified is tried in turn until an unoccupied one is found. Service classes specify a switching office and a baud rate.

Defined service classes at the 'research' site are `300` and `1200`, for 300- and 1200-baud calls on 665- phone lines, with synonyms `D300` and `D1200`. Internal calls on these lines require 5 digits. Service classes `C300` and `C1200` use 582- phone lines. Internal calls on these lines require 4 digits and reach only other 582- lines.

file The file name of the associated special file for the telephone line.

acu The file name of the associated ACU. If specified as `none`, no ACU is used and the telephone number is ignored. This is for hardwired connections.

speed The bit rate of the interface, chosen from the numbers given in *ttyld*(4).

prefix A string to be prefixed to the number. This is handy for shared ACUs in which the first digit specifies a line. The prefix − is taken to be an empty prefix.

postfix A string to be postfixed to the number to be dialed. Some ACUs require an 'end of number' code; it should be specified here.

FILES

`/etc/aculist`

SEE ALSO

ttyld(4), *cu*(1), *ipc*(3)

NAME

opendir, readdir, telldir, seekdir, closedir – directory operations

SYNOPSIS

```
#include <sys/types.h>
#include <ndir.h>

DIR *opendir(filename)
char *filename;

struct direct *readdir(dirp)
DIR *dirp;

long telldir(dirp)
DIR *dirp;

seekdir(dirp, loc)
DIR *dirp;
long loc;

closedir(dirp)
DIR *dirp;
```

DESCRIPTION

Opendir opens the directory named by *filename* and associates a ‘directory stream’ with it. *Opendir* returns a pointer to be used to identify the directory stream in subsequent operations. The pointer value 0 is returned if *filename* cannot be accessed or is not a directory.

Readdir returns a pointer to the next directory entry. It returns 0 upon reaching the end of the directory or detecting an invalid *seekdir* operation.

Telldir returns the current location associated with the named directory stream.

Seekdir sets the position of the next *readdir* operation on the directory stream. The new position reverts to the one associated with the directory stream when the *telldir* operation was performed. Values returned by *telldir* are good only for the lifetime of the `DIR` pointer from which they are derived.

Closedir causes the named directory stream to be closed, and the structure associated with the `DIR` pointer to be freed.

```
struct direct {
        u_long d_ino;                inode for the entry
        short  d_reclen;             don't use
        short  d_namlen;             equivalent to strlen(d_name)
        char   d_name[MAXNAMLEN+1]; null-terminated entry name
};
```

The preferred way to search the current directory is:

```
        DIR *dirp;
        dirp = opendir(".");
        for(dp = readdir(dirp); dp != 0; dp = readdir(dir))
                if(strcmp(dp->d_name, name) == 0)
                        break;
        closedir(dirp);
        /* found name if dp != 0 */
```

SEE ALSO

dir(5), *open*(2), *dirread*(2), *read*(2), *lseek*(2), *ftw*(3)

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

ecvt, fcvt, gcvt − convert numbers to ascii

SYNOPSIS

```
char *ecvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt(value, ndigit, buf)
double value;
char *buf;
```

DESCRIPTION

Ecvt converts the *value* to a null-terminated string of *ndigit* ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt* (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero. The low-order digit is rounded.

Fcvt is similar to *ecvt* and produces output for the Fortran format `F*.`*ndigit*`.` If *decpt*≤−*ndigit*, then the returned string is null. Otherwise, *decpt*+*ndigit*+1 characters (including terminating null) are returned.

Gcvt converts the *value* to a null-terminated ASCII string in *buf* and returns a pointer to *buf*. It attempts to produce *ndigit* significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

SEE ALSO

printf(3)

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

end, etext, edata – last locations in program

SYNOPSIS

```
extern end;
extern etext;
extern edata;
```

DESCRIPTION

These names refer neither to routines nor to locations with interesting contents. The address of *etext* is the first address above the program text, *edata* above the initialized data region, and *end* above the uninitialized data region.

When execution begins, the program break coincides with *end*, but it is reset by the routines *brk*(2), *malloc*(3), standard input/output (*stdio*(3)), the profile (`-p`) option of *cc*(1), etc. The current value of the program break is reliably returned by `sbrk(0)`; see *brk*(2).

SEE ALSO

brk(2), *malloc*(3), *stdio*(3), *cc*(1)

NAME

erf, erfc – error function

SYNOPSIS

```
#include <math.h>

double erf(x)
double x;

double erfc(x)
double x;
```

DESCRIPTION

These functions calculate the error function $\mathrm{erf}(x) = 2\pi^{-1/2} \int_0^x \mathrm{e}^{-t^2}\, dt$ and the complementary error function $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$. The error criterion for both *erf* and *erfc* is relative.

DIAGNOSTICS

There are no error returns.

NAME

exit, atexit, onexit – terminate process

SYNOPSIS

```
void exit(status)
int status;

int atexit(fn)
int (*fn)();
```

DESCRIPTION

Exit is the conventional way to terminate a process. Before calling *_exit* (see *exit*(2)) with *status* as an argument, it calls in reverse order all the functions recorded by *atexit*.

Exit can never return.

Atexit records *fn* as a function to be called by *exit*. It returns zero if it failed, nonzero otherwise. Typical uses include cleanup routines for *stdio*(3) and profiling; see *monitor*(3).

Calling *atexit* twice (or more) with the same function argument causes *exit* to invoke the function twice (or more).

The function *fn* should be declared as

```
int fn()
```

The function *onexit* is an obsolescent synonym for *atexit.* The constant `NONEXIT` defined in `<libc.h>` determines how many functions can be recorded.

SEE ALSO

exit(2)

NAME

exp, log, log10, pow, sqrt – exponential, logarithm, power, square root

SYNOPSIS

```
#include <math.h>

double exp(x)
double x;

double log(x)
double x;

double log10(x)
double x;

double pow(x, y)
double x, y;

double sqrt(x)
double x;
```

DESCRIPTION

Exp returns the exponential function of *x.*

Log returns the natural logarithm of *x*; *log10* returns the base 10 logarithm.

Pow returns x^y.

Sqrt returns the square root of *x.*

SEE ALSO

hypot(3), *sinh*(3), *intro*(2)

DIAGNOSTICS

Exp and *pow* return a huge value when the correct value would overflow; *errno* is set to ERANGE. *Pow* returns 0 and sets *errno* to EDOM when the second argument is negative and non-integral and when both arguments are 0.

Log returns a huge negative value when *x* is zero or negative; *errno* is set to EDOM.

Sqrt returns 0 when *x* is negative; *errno* is set to EDOM.

NAME

feof, ferror, clearerr, fileno – stream status inquiries

SYNOPSIS

```
#include <stdio.h>

int feof(stream)
FILE *stream;

int ferror(stream)
FILE *stream

int clearerr(stream)
FILE *stream

int fileno(stream)
FILE *stream;
```

DESCRIPTION

Feof returns non-zero if end of file has been encountered on the named input *stream*, otherwise zero. After returning `EOF`, *stdio*(3) functions will not necessarily return `EOF` again; *feof* provides a lasting indication.

Ferror returns non-zero when an error has occurred reading or writing the named *stream*, otherwise zero. Unless cleared by *clearerr*, the error indication lasts until the stream is closed.

Clearerr resets the error indication on the named *stream.*

Fileno returns the integer file descriptor associated with the *stream*, see *open*(2).

These functions are implemented as macros; they cannot be redeclared.

SEE ALSO

stdio(3)

NAME

fgets, puts, fputs, gets – string input/out on streams

SYNOPSIS

```
#include <stdio.h>

char *fgets(s, n, stream)
char *s;
FILE *stream;

int puts(s)
char *s;

int fputs(s, stream)
char *s;
FILE *stream;
```

DESCRIPTION

Fgets reads $n-1$ characters, or up to a newline character, whichever comes first, from the *stream* into the string *s*. The last character read into *s* is followed by a null character. *Fgets* returns its first argument.

Puts copies the null-terminated string *s* to the standard output stream *stdout* and appends a newline character.

Fputs copies the null-terminated string *s* to the named output *stream*.

Neither routine copies the terminal null character. Both return the result of calling *putc* with the last character written; see *getc*(3).

SEE ALSO

getc(3) *stdio*(3)

DIAGNOSTICS

Fgets returns a null pointer upon end of file or error. *Puts* and *fputs* return the constant on write error.

BUGS

For safety reasons the ANSI standard function `char *gets(s)`, which reads from standard input up to a newline and discards the newline, is not supported.
Puts appends a newline, *fputs* does not, all in the name of backward compatibility.

NAME

Finit, Frdline, Fgetc, Fread, Fseek, Fundo, Fputc, Fprint, Fwrite, Fflush, Ftie, Fclose, Fexit – fast buffered input/output

SYNOPSIS

```
#include <fio.h>

void Finit(fd, buf)
char *buf;

int Fclose(fd);

int Fprint(fildes, format [, arg ...])
int fildes;
char *format;

char *Frdline(fd)

int FIOLINELEN(fd)

long FIOSEEK(fd)

int Fgetc(fd)

void Fundo(fd)

long Fseek(fd, offset, ptr)
long offset;

int Fputc(fd, c)

long Fread(fd, addr, nbytes)
char *addr;
long nbytes;

long Fwrite(fd, addr, nbytes)
char *addr;
long nbytes;

int Fflush(fd)

void Ftie(ifd, ofd)

Fexit(type)
```

DESCRIPTION

These routines provide buffered I/O, faster than, and incompatible with *stdio*(3). The routines can be called in any order. I/O on different file descriptors is independent.

Finit initializes a buffer (whose type is *Fbuffer*) associated with the file descriptor *fd*. Any buffered input associated with *fd* will be lost. The buffer can be supplied by the user (it should be at least `sizeof(Fbuffer)` bytes) or if *buf* is `(char *)0`, *Finit* will use *malloc*(3). *Finit* must be called after a stretch of *non*-fio activity, such as *close* or *lseek*(2), between *fio* calls on the same file descriptor number; it is unnecessary, but harmless, before the first *fio* activity on a given file descriptor number.

Fclose flushes the buffer for *fd*, *free*s the buffer if it was allocated by *Finit*, and then closes *fd*.

Frdline reads a line from the file associated with the file descriptor *fd*. The newline at the end of the line is replaced by a 0 byte. *Frdline* returns a pointer to the start of the line or `(char *)0` on end of file or read error. The macro *FIOLINELEN* returns the length (not including the 0 byte) of the most recent line returned by *Frdline*. The value is undefined after a call to any other *fio* routine.

Fgetc returns the next character from the file descriptor *fd*, or a negative value at end of file.

Fread reads *nbytes* of data from the file descriptor *fd* into memory starting at *addr*. The number of bytes read is returned on success and a negative value is returned if a read error occurred.

Fseek applies *lseek*(2) to *fd* taking buffering into account. It returns the new file offset. The macro *FIOSEEK* returns the file offset of the next character to be processed.

Fundo makes the characters returned by the last call to *Frdline* or *Fgetc* available for reading again. There is only one level of undo.

Fputc outputs the low order 8 bits of *c* on the file associated with file descriptor *fd*. If this causes a *write* (see *read*(2)) to occur and there is an error, a negative value is returned. Otherwise, zero is returned.

Fprint is a buffered interface to *print*(3). If this causes a *write* to occur and there is an error, a negative value is returned. Otherwise, the number of chars output is returned.

Fwrite outputs *nbytes* bytes of data starting at *addr* to the file associated with file descriptor *fd*. If this causes a *write* to occur and there is an error, a negative value is returned. Otherwise, the number of bytes written is returned.

Fflush causes any buffered output associated with *fd* to be written; it must precede a call of *close* on *fd.* The return is as for *Fputc*.

Ftie links together two file descriptors such that any *fio*-initiated *read*(2) on *ifd* causes a *Fflush* of *ofd* (if it has been initialized). It is appropriate for most programs used as filters to do `Ftie(0,1)`. The tie may be broken by `Ftie(ifd, -1)`.

Fexit is used to clean up all *fio* buffers. If *type* is zero, the buffers are *Fflush*ed, otherwise they are *Fclose*d. `Fexit(0)` is automatically called at *exit*(3).

SEE ALSO

open(2), *print*(3), *stdio*(3)

DIAGNOSTICS

Fio routines that return integers yield `-1` if *fd* is not the descriptor of an open file or if the operation is inapplicable to *fd.*

BUGS

Frdline deletes characters from lines longer than 4095 characters, ignores characters after the last newline in a file, and will read past and end-of-file indication on a stream.
The data returned by *Frdline* may be overwritten by calls to any other *fio* routine.
Fgetc is much slower than access through a pointer returned by *Frdline.*
There is no *scanf*(3) analogue.

NAME

fabs, fmod, floor, ceil – absolute value, remainder, floor, ceiling functions

SYNOPSIS

```
#include <math.h>

double floor(x)
double x;

double ceil(x)
double x;

double fabs(x)
double x;

double fmod(x,y)
double x, y;
```

DESCRIPTION

Fabs returns the absolute value $|x|$.

Floor returns the largest integer not greater than x.

Ceil returns the smallest integer not less than x.

Fmod returns x if y is zero, otherwise the number f with the same sign as x, such that $x = iy + f$ for some integer i, and $|f| < |y|$.

SEE ALSO

arith(3), *frexp*(3)

NAME

fopen, freopen, fdopen, fclose, fflush – open, close, or flush a stream

SYNOPSIS

```
#include <stdio.h>

FILE *fopen(filename, type)
char *filename, *type;

FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen(fildes, type)
char *type;

int fclose(stream)
FILE *stream;

int fflush(stream)
FILE *stream;
```

DESCRIPTION

Fopen opens the file named by *filename* and associates a stream with it. *Fopen* returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

`"r"` open for reading
`"w"` create for writing
`"r+w"`
`"w+r"` open for reading and writing
`"a"` append: open for writing at end of file, or create for writing

Freopen substitutes the named file in place of the open *stream*. It returns the original value of *stream*. The original stream is closed. *Freopen* is typically used to attach the preopened constant names `stdin`, `stdout` and `stderr` to specified files.

Fdopen associates a stream with a file descriptor. The *type* of the stream must agree with the mode of the open file.

Fclose causes any buffers for the named *stream* to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling *exit*(3).

Fflush causes any buffered data for the named output *stream* to be written to that file. The stream remains open.

SEE ALSO

open(2), *popen*(3), *stdio*(3), *ferror*(3)

DIAGNOSTICS

Fopen and *freopen* return `NULL` if *filename* cannot be accessed.

Fclose and *fflush* return `EOF` if *stream* is not associated with a file, or if buffered data cannot be transferred to that file.

NAME

fread, fwrite – buffered binary input/output

SYNOPSIS

```
#include <stdio.h>

int fread(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

int fwrite(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;
```

DESCRIPTION

Fread reads from the named input *stream* at most *nitems* of data of the type of **ptr* into a block beginning at *ptr*. It returns the number of items actually read.

Fwrite appends to the named output *stream* at most *nitems* of data of the type of **ptr* from a block beginning at *ptr*. It returns the number of items actually written.

SEE ALSO

read(2), *stdio*(3)

DIAGNOSTICS

Fread and *fwrite* return a short count upon end of file or error.

BUGS

Write errors from *fwrite* may be delayed until a subsequent *stdio*(3) writing, seeking, or file-closing call. These routines are much slower than you might imagine.

NAME

frexp, ldexp, modf – split into mantissa and exponent

SYNOPSIS

```
#include <math.h>

double frexp(value, eptr)
double value;
int *eptr;

double ldexp(value, exp)
double value;

double modf(value, iptr)
double value, *iptr;
```

DESCRIPTION

Frexp returns the mantissa of *value* and stores the exponent indirectly through *eptr*. If $1/2 \leq |X| < 1$, then $x = frexp(x \times 2^{*eptr}, eptr)$.

Ldexp returns the quantity $value \times 2^{exp}$.

Modf returns the positive fractional part of *value* and stores the integer part indirectly through *iptr*.

DIAGNOSTICS

On underflow *ldexp* returns 0; on overflow it returns a properly signed largest value. In both cases it sets *errno* to `ERANGE`.

NAME

fseek, ftell, rewind – reposition a stream

SYNOPSIS

```
#include <stdio.h>

int fseek(stream, offset, ptrname)
FILE *stream;
long offset;

long ftell(stream)
FILE *stream;

int rewind(stream)
```

DESCRIPTION

Fseek sets the position of the next input or output operation on the *stream*. The new position is at the signed distance *offset* bytes from the beginning, the current position, or the end of the file, as *ptrname* has the value 0, 1 or 2 respectively.

Ftell returns the current value of the file pointer for the file associated with the named *stream*.

Rewind(*stream*) is equivalent to `fseek(stream, 0L, 0)`.

SEE ALSO

lseek(2), *stdio*(3)

DIAGNOSTICS

Fseek returns −1 for improper seeks. *Ftell* returns −1 if seeking is impossible.

BUGS

The interaction of *fseek* and *ungetc*(3) is undefined.

NAME

ftw − file tree walk

SYNOPSIS

```
#include <ftw.h>

int ftw(path, fn, depth)
char *path;
int (*fn)();
int depth;

#include <sys/types.h>
#include <sys/stat.h>

fn(name, statb, code, S)
char *name;
struct stat *statb;
struct FTW *S;
```

DESCRIPTION

Ftw recursively descends the directory hierarchy rooted in *path*. For each entry in the hierarchy, *ftw* calls *fn*, passing it information about the entry: a pointer to a null-terminated pathname string, a pointer to a `stat` structure (see *stat*(2)), and a pointer to the following structure.

```
struct FTW {
        int quit;   see below
        int base;   &name[base]  points to basename
        int level;  recursion level (initially 0)
};
```

Possible values of *code*, defined in `<ftw.h>`, are

`FTW_D`	Entry is a directory (before visiting descendants).
`FTW_DP`	Entry is a directory (after visiting descendants).
`FTW_SL`	Entry is a symbolic link.
`FTW_F`	Entry is some other kind of file.
`FTW_DNR`	Entry is a directory that cannot be read; no descendants will be visited.
`FTW_NS`	*Lstat* (see *stat*(2)) failed on *name*; contents of *statb* are undefined
`FTW_NSL`	*Lstat* succeeded, but *stat* failed; contents of *statb* are undefined.

The tree traversal continues until the tree is exhausted or *fn* returns a nonzero value. When the tree is exhausted, *ftw* returns zero. When *fn* returns a nonzero value, *ftw* stops and returns that value.

Normally symbolic links are not followed. But if on a symbolic link (`FTW_SL`) *fn* sets `S->quit` to `FTW_FOLLOW`, *ftw* will next attempt to follow the link.

Ftw normally visits a readable directory twice, before and after visiting its descendants. But if on a previsit (`FTW_D`) *fn* sets `S->quit` to `FTW_SKD`, *ftw* will skip the descendants and the postvisit (`FTW_DP`).

Ftw uses one file descriptor for each level in the tree up to a maximum of *depth* (or 1, if *depth*<1) descriptors. *Depth* must not exceed the number of available file descriptors; small values of *depth* may cause *ftw* to run slowly, but will not change its effect.

SEE ALSO

stat(2), *directory*(3)

DIAGNOSTICS

Ftw returns −1 with *errno* set to `ENOMEM` when *malloc*(3) fails.

Errno is set appropriately when *ftw* calls *fn* with code `FTW_DNR`, `FTW_NS`, or `FTW_NSL`.

NAME

galloc, gfree, garbage – storage allocation with garbage collection

SYNOPSIS

```
char *galloc(n)
unsigned n;

void gfree(p)
char *p;

void garbage()
```

DESCRIPTION

These functions perform heap storage allocation with garbage collection.

Galloc allocates a block of at least *n* bytes and returns a pointer to it. *Gfree* frees a block previously allocated by *galloc.*

When space gets tight, garbage blocks are freed automatically. A block allocated by *galloc* is deemed to be garbage unless it is reachable. A reachable block is one whose first byte is pointed to by a declared C variable or by a pointer in a reachable block.

The frequency of garbage collection is controlled by external variables declared

```
long gcmax = 5000, gcmin = 50;
```

No more than *gcmax* allocations may intervene between automatic collections; this feature will help contain the growth of virtual address space. At least *gcmin* allocations must intervene, otherwise garbage collection will be abandoned as fruitless. *Garbage* may be called to do garbage collection at an arbitrary time.

Malloc(3) and *galloc* allocate from the same arena, but garbage collection affects only *galloc* blocks. *Free* (see *malloc*(3)) must not be used on blocks allocated with *galloc*.

SEE ALSO

malloc(3)

DIAGNOSTICS

Galloc returns 0 when space cannot be found.

BUGS

Garbage collection is conservative; blocks that appear to be pointed to from within declared storage will not be freed, regardless of whether the apparent ‘pointers’ were declared as such.

NAME

gamma – log gamma function

SYNOPSIS

```
#include <math.h>

double gamma(x)
double x;

extern int signgam;
```

DESCRIPTION

Gamma returns ln $|\Gamma(x)|$. The sign of $\Gamma(x)$ is returned in the external integer *signgam*.

EXAMPLES

Computation of the gamma function:

```
errno = 0;
y = gamma(x);
if(errno || (y > 88.0))
        error();
y = signgam*exp(y);
```

DIAGNOSTICS

A large value (HUGE) is returned for negative integer arguments and *errno* is set to EDOM.

BUGS

There should be a positive indication of error.

The name should indicate the answer is a logarithm, perhaps *lgamma*.

NAME

getarg, iargc – command arguments to Fortran

SYNOPSIS

```
subroutine getarg(argno, string)
integer argno
character *(*) string

iargc()
```

DESCRIPTION

These procedures permit Fortran programs to access the command arguments. The integer function *iargc* returns the number of command arguments. The subroutine *getarg* stores the *argno*th command argument in its second argument. The string is truncated or padded with blanks, in accord with the rules of Fortran character assignment.

EXAMPLES

`a.out arg1 arg2`

In a program invoked this way *iargc* will return 2.

```
character*4 s
call getarg(2, s)
```

Place `arg2` in *s*.

SEE ALSO

exec(2)

NAME

getc, getchar, fgetc, getw, putc, putchar, fputc, putw – character- or word-at-a-time stream input/output

SYNOPSIS

```
#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar()

int getw(stream)
FILE *stream;

int fgetc(stream)
FILE *stream;

int putc(c, stream)
char c;
FILE *stream;

putchar(c)

putw(w, stream)
FILE *stream;

fputc(c, stream)
FILE *stream;
```

DESCRIPTION

Getc returns the next character from the named input *stream*.

Getchar() is identical to *getc*(*stdin*).

Getw returns the next word (32-bit integer on a VAX) from the named input *stream*. *Getw* assumes no special alignment in the file.

Putc appends the character *c* to the named output *stream*. It returns the character written.

Putchar(*c*) is identical to *putc*(*c, stdout*).

Putw appends word (i.e. `int`) *w* to the output *stream*. It returns the word written. *Putw* neither assumes nor causes special alignment in the file.

Fgetc and *fputc* behave like *getc* and *putc*, but are genuine functions, not macros; they may be used to save object text.

The standard output stream `stdout` is normally buffered, but is flushed whenever *getc* causes a buffer to be refilled from `stdin`. The standard error stream `stderr` is normally unbuffered. These defaults may be changed by *setbuf*(3). When an output stream is unbuffered, information appears on the destination file or terminal as soon as written. When an output stream is buffered, many characters are saved up and written as a block. *Fflush* may be used to force the block out early; see *fopen*(3).

SEE ALSO

fopen(3), *ungetc*(3), *stdio*(3)

DIAGNOSTICS

These functions return the integer constant `EOF` at end of file or error. For *getw* or *putw* this indication is ambiguous; *ferror*(3) may be used to distinguish.

BUGS

Because they are implemented as macros, *getc* and *putc* treat *stream* arguments with side effects incorrectly. For example, `getc(*f++)` is wrong.

The routines in *printf*(3) provide temporary buffering even when buffering has been turned off.

Write errors may be delayed until a subsequent *stdio (3)* writing, seeking, or file-closing call.

NAME

getenv – value for environment name

SYNOPSIS

```
char *getenv(name)
char *name;
```

DESCRIPTION

Getenv searches the environment list (see *environ*(5)) for a string starting with *name*=. If no such a string is found, 0 is returned. Otherwise, the address of the character following the = is returned.

SEE ALSO

printenv(1), *environ*(5), *exec*(2)

BUGS

Getenv ignores shell functions; see *sh*(1).

NAME

getfields, getmfields, setfields – break a string into fields

SYNOPSIS

```
int getfields(str, ptrs, nptrs)
char *str, **ptrs;

int getmfields(str, ptrs, nptrs)
char *str, **ptrs;

char *setfields(fielddelim)
char *fielddelim;
```

DESCRIPTION

Getfields breaks the null-terminated string *str* into at most *nptrs* null-terminated fields and places pointers to the start of these fields in the array *ptrs*. It returns the number of fields and terminates the list of pointers with a zero pointer. It overwrites some of the bytes in *str*. If there are *nptr* or more fields, the list will not end with zero and the last 'field' will extend to the end of the input string and may contain delimiters.

A field is defined as a maximal sequence of characters not in a set of field delimiters. Adjacent fields are separated by exactly one delimiter. No field follows a delimiter at the end of string. Thus a string of just two delimiter characters contains two empty fields, and a nonempty string with no delimiters contains one field.

Getmfields is the same as *getfields* except that fields are separated by maximal strings of field delimiters rather than just one.

Setfields makes the field delimiters (space and tab by default) be the characters of the string *fielddelim* and returns a pointer to a string of the previous delimiters.

EXAMPLES

Print the words in a string, where words are non-whitespace strings. There is no bound on the number of words.

```
printwords(string)
char *string;
{
        char *ptrs[2];
        int n;
        setfields(" \t\n");
        for(n=2; n>=2; string=ptrs[1]) {
                n = getmfields(string, ptrs, 2);
                if(n == 0)
                        break;
                if(ptrs[0][0] != 0)     /* skip initial blanks */
                        printf("%s\n", ptrs[0]);
        }
}
```

SEE ALSO

string(3)

NAME

getflags – process flag arguments in argv

SYNOPSIS

```
#include </usr/include/getflags.h>

int getflags(argc, argv, flags)
char **argv, *flags;

usage(tail)
char *tail;

extern char **flag[], cmdline[], *cmdname, *flagset[];
```

DESCRIPTION

Getflags digests an argument vector *argv*, finding flag arguments listed in *flags*. *Flags* is a string of flag letters. A letter followed by a colon and a number is expected to have the given number of parameters. A flag argument starts with '−' and is followed by any number of flag letters. A flag with one or more parameters must be the last flag in an argument. If any characters follow it, they are the flag's first parameter. Otherwise the following argument is the first parameter. Subsequent parameters are taken from subsequent arguments.

The global array *flag* is set to point to an array of parameters for each flag found. Thus, if flag `-x` was seen, `flag['x']` is non-zero, and `flag['x'][i]` is the flag's *i*th parameter. If flag `-x` has no parameters `flag['x']==flagset`. Flags not found are marked with a zero. Flags and their parameters are deleted from *argv*. *Getflags* returns the adjusted argument count.

Getflags stops scanning for flags upon encountering a non-flag argument, or the argument `--`, which is deleted.

Getflags places a pointer to *argv[0]* in the external variable *cmdname*. It also concatenates the original members of *argv*, separated by spaces, and places the result in the external array *cmdline*.

Usage constructs a usage message, prints it on the standard error file, and exits with status 1. The command name printed is *argv[0]*. Appropriate flag usage syntax is generated from *flags*. As an aid, explanatory information about flag parameters may be included in *flags* in square brackets as in the example. *Tail* is printed at the end of the message. If *getflags* encountered an error, *usage* tries to indicate the cause.

EXAMPLES

```
main(argc, argv)
char *argv[];
{
      if((argc=getflags(argc, argv, "vinclbhse:1[expr]", 1))==-1)
            usage("[file ...]");
}
```

might print:

```
Illegal flag -u
Usage: grep [-vinclbhs] [-e expr] [file ...]
```

SEE ALSO

getopt(3)

DIAGNOSTICS

Getflags returns −1 on error: a syntax error in *flags*, setting a flag more than once, setting a flag not mentioned in *flags*, or running out of argv while collecting a flag's parameters.

NAME

getfsent, getfsspec, getfsfile, setfsent, endfsent – get file system description file entry

SYNOPSIS

```
#include <fstab.h>

struct fstab *getfsent()

struct fstab *getfsspec(name)
char *name;

struct fstab *getfsfile(name)
char *name;

int setfsent()

int endfsent()
```

DESCRIPTION

Getfsent, *getfsspec* and *getfsfile* each return a pointer to a structure containing the broken-out fields of a line in *fstab*(5), which describes mountable file systems.

```
struct fstab {
        char fs_spec[FSNMLG];   block device name
        char fs_file[FSNMLG];   file system mount point
        int fs_ftype;           file system type
        int fs_flags;           file system flags
        int fs_passno;          pass number for parallel fsck(8)
};
```

Type numbers and flags are listed in *fmount*(2). Entries that aren't file systems (should not be mounted) have negative values for `fs_ftype`:

`FSNONE` (−1) Ignore this entry.
`FSSWAP` (−2) `fs_spec` is a device available for swapping.

Getfsent reads the next line of the file, opening the file if necessary.

Setfsent opens and rewinds the file.

Endfsent closes the file.

Getfsspec and *getfsfile* sequentially search from the beginning of the file until a matching special file name or file system file name is found, or until EOF is encountered.

FILES

`/etc/fstab`

SEE ALSO

fmount(2), *fstab*(5)

DIAGNOSTICS

Zero is returned on EOF or error.

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent – get group file entry

SYNOPSIS

```
#include <grp.h>

struct group *getgrent();

struct group *getgrgid(gid)

struct group *getgrnam(name)
char *name;

int setgrent();

int endgrent();
```

DESCRIPTION

Getgrent, *getgrgid* and *getgrnam* each return pointers to a structure containing the broken-out fields of a line in `/etc/group`.

```
struct group {
        char  *gr_name;     the group name
        char  *gr_passwd;   the encrypted group passwd
        int   gr_gid;       the numeric groupid
        char  **gr_mem;     null-terminated vector of pointers to the individual member names
};
```

Getgrent simply reads the next line while *getgrgid* and *getgrnam* search until a matching *gid* or *name* is found (or until EOF is encountered). Each routine picks up where the others leave off so successive calls may be used to search the entire file.

A call to *setgrent* has the effect of rewinding the group file to allow repeated searches. *Endgrent* may be called to close the group file when processing is complete.

FILES

`/etc/group`

SEE ALSO

getlogin(3), *getpwent*(3), *passwd*(5)

DIAGNOSTICS

Zero is returned on EOF or error.

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

getlogin – get login name

SYNOPSIS

```
char *getlogin()
```

DESCRIPTION

Getlogin returns a pointer to the login name as set by *setlogname*; see *getuid*(2).

It is preferable, but less portable, to call *getlogname*; see *getuid*(2).

FILES

```
/etc/utmp
```

SEE ALSO

getpwent(3), *getgrent*(3), *utmp*(5), *getuid*(2)

DIAGNOSTICS

Zero is returned if the name could not be found.

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

getopt – get option letter from argv

SYNOPSIS

```
int getopt (argc, argv, optstring)
int argc;
char **argv;
char *optstring;

extern char *optarg;
extern int optind;
extern int opterr;
```

DESCRIPTION

Getopt returns the next option letter in *argv* that matches a letter in *optstring*. *Optstring* is a string of recognized option letters; if a letter is followed by a colon, the option is expected to have an argument, which may or may not be separated from it by white space. *Optarg* is set to point to the start of the option argument, if any.

Optind, initially 1, holds the index in *argv* of the next argument to be processed. When *opterr* is nonzero (the default state), errors cause diagnostic messages.

Option letters appear in nonempty clusters preceded by −. The special option −− may be used to mark the end of the options.

EXAMPLES

This fragment processes arguments for a command that can take option `a` and option `f`, which requires an argument.

```
main (argc, argv) char **argv;
{
        int c, errflg = 0;
        extern int optind;
        extern char *optarg, *ifile;
        while((c = getopt(argc, argv, "af:")) != -1)
                switch (c){
                case 'a':  aflg=1; break;
                case 'f':  ifile = optarg; break;
                case '?':  errflg=1; break;
                }
        if(errflg){
                fprintf(stderr, "usage: . . . ");
                exit(2);
        }
        for( ; optind < argc; optind++){
                if(access(argv[optind], 4)){
                ...
                }
        }
        ...
}
```

SEE ALSO

getflags(3)

DIAGNOSTICS

When all options have been processed, −1 is returned; *optind* refers to the first non-option argument.
When *getopt* encounters an option letter not included in *optstring* or finds an option argument missing, it prints a diagnostic on `stderr` under control of *opterr* and returns a question mark `?`.

NAME

getpass – read a password

SYNOPSIS

```
char *getpass(prompt)
char *prompt;
```

DESCRIPTION

Getpass reads a password from the file `/dev/tty`, or if that cannot be opened, from the standard input, after prompting with the null-terminated string *prompt* and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters.

FILES

`/dev/tty`

SEE ALSO

crypt(3)

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent, pwdecode – get password file entry

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent()

struct passwd *getpwuid(uid)
int uid;

struct passwd *getpwnam(name)
char *name;

int setpwent()

int endpwent()

struct passwd *pwdecode(p)
char *p;
```

DESCRIPTION

Getpwent, *getpwuid* and *getpwnam* each return a pointer to a structure containing the broken-out fields of a line in `/etc/passwd`.

```
struct passwd {
        char    *pw_name;       login name
        char    *pw_passwd;     encrypted password
        int     pw_uid;         numeric userid
        int     pw_gid;         numeric groupid
        int     pw_quota;       unused
        char    *pw_comment;    unused
        char    *pw_gecos;      field for local use
        char    *pw_dir;        login directory
        char    *pw_shell;      program to use as Shell
};
```

Getpwent reads the next line (opening the file if necessary); *setpwent* rewinds the file; *endpwent* closes it.

Getpwuid and *getpwnam* search from the beginning until a matching *uid* or *name* is found (or until end-of-file is encountered).

Pwdecode breaks out a null-terminated character string *p* containing a password file entry. The input string is modified by the call and the output structure contains pointers into it.

FILES

`/etc/passwd`

SEE ALSO

getlogin(3), *getgrent*(3), *passwd*(5)

DIAGNOSTICS

These routines return 0 for end of file or error.

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

getwd, getcwd – get current directory

SYNOPSIS

```
char *getwd(buf)
char *buf;

char *getcwd(buf, size)
char *buf;
```

DESCRIPTION

Getwd and *getcwd* fill *buf* with a null-terminated string representing the current directory and return *buf*.

Getwd is in the style of BSD systems and *getcwd* in that of System V. If *buf* is 0, *getcwd* will call *malloc*(3) to allocate *size* bytes for the buffer.

SEE ALSO

pwd(1)

DIAGNOSTICS

On error, zero is returned and *buf* is filled with a diagnostic message.

NAME

huff – huffman codebook/tree generator

SYNOPSIS

```
#include <huff.h>

NODE *huff(inrout)
int (*inrout)();
```

DESCRIPTION

Huff generates a binary Huffman codebook. It obtains a list of messages one at a time from an input routine, *inrout*, declared as

```
int inrout(str, p)
char ** str;
float *p;
```

Inrout makes **str* point to a null-terminated string identifying a message, and places in **p* the (arbitrarily normalized) frequency of the message. *Inrout* returns non-zero when data is returned and zero when there is no more data.

Huff returns a pointer to a root of type `NODE`:

```
typedef struct node {
      char *datump;
      struct node *to;
      struct node *from;
      struct node *ldad;
      struct node *rdad;
      struct node *kid;
      float prob;
} NODE;
```

The root heads a linked list and the Huffman tree. The doubly linked list, connected via `from` and `to`, is ordered as the codebook was generated. The tree is connected via `kid`, `ldad`, and `rdad`, with null pointers at the various ends. The `kid` field points towards the root, `ldad` and `rdad` point away: `node->ldad->kid==node` and `node->rdad->kid==node`. the `datump` field is null or points to a message identifier.

The codeword for a message may be read off from the path from the root to the node containing the message identifier, counting *ldad* branches as 0 and *rdad* branches as 1.

BUGS

A code with only one message dumps core.

NAME

hypot, cabs – euclidean distance

SYNOPSIS

```
#include <math.h>

double hypot(x, y)
double x, y;

double cabs(z)
struct { double x, y; } z;
```

DESCRIPTION

Hypot and *cabs* return *sqrt(x*x + y*y)*, taking precautions against unwarranted overflows.

SEE ALSO

exp(3)

NAME

in_host, in_ntoa, in_address, in_service − internet networking functions

SYNOPSIS

```
#include <sys/inet/in.h>

char *in_host(hostaddr)
in_addr hostaddr;

char *in_ntoa(hostaddr)
in_addr hostaddr;

in_addr in_address(hostname)
char *hostname;

struct in_service *in_service(name, proto, port)
char *name, *proto;
unsigned long port;
```

DESCRIPTION

These routines are loaded by the `-lin` option of *ld*(1).

Internet addresses, type *in_addr*, are 32-bit quantities global to the network. The ASCII representation of an *in_addr* can be either a host name or of the form *b1.b2.b3.b4*, where each *'bx'* is the value of the *x*'th byte of the address in decimal. Since host names are considered local 'aliases' for internet addresses, the host-to-address mapping is subjective.

In_address maps an internet host name to an address and returns 0 if the name is not found in the host table.

In_host maps an internet address into a host name. If the host is not found in the host table, the ASCII representation of the address is returned.

In_ntoa maps an internet address to its ASCII numeric format.

In_service returns the closest match to *name* in the services file. If either *name* or *port* are 0, they will match any name or port. If *proto* is `(char *)0`, the `tcp` protocol is assumed.

FILES

`/usr/inet/lib/hosts`	mapping between host names and addresses
`/usr/inet/lib/networks`	mapping between network names and addresses
`/usr/inet/lib/services`	database of services
`/usr/inet/lib/hosts.equiv`	machines with common administration

SEE ALSO

ipc(3), *tcp*(3), *udp*(3)

BUGS

The mappings between internet addresses and names is arbitrary at best. The hosts file may contain many addresses for each name and/or many names for each address. *In_address* and *in_host* each start at the beginning of the file and search sequentially for a match. Therefore, `in_addr(in_host(addr)) == addr` is not necessarily true.

NAME

ipccreat, ipcopen, ipclisten, ipcaccept, ipcreject, ipcexec, ipcpath , ipclogin, ipcrogin – set up connections between processes or machines

SYNOPSIS

```
#include <ipc.h>

char *ipcpath(name, network, service)
char *name;
char *network;
char *service;

int ipcopen(name, param)
char *name;
char *param;

int ipccreat(name, param)
char *name;
char *param;

ipcinfo *ipclisten(fd)
int fd;

int ipcaccept(ip)
ipcinfo *ip;

int ipcreject(ip, no, str)
ipcinfo *ip;
int no;
char *str;

int ipcexec(name, param, cmd)
char *name;
char *param;
char *cmd;

int ipclogin(fd)
int fd;

int ipcrogin(fd, opt)
int fd;
char *opt;

extern char *errstr;
```

DESCRIPTION

These routines establish communication between unrelated processes, often for networking purposes. They are loaded by the `-lipc` option of *ld*(1).

End points in the network are identified by names of the form: `element[!element]...` . The name is translated element by element relative to the name space selected by the previous element. The first element is always a name in the local file system. By convention, all network interfaces and services mount themselves in `/cs`. For example:

`/cs/exec`
refers to a local process which has mounted itself (via *ipccreat)* on `/cs/exec`.

`/cs/dk!nj/astro/voice`
refers to a voice synthesizer attached to Datakit; process `/cs/dk` is the Datakit interface.

`/cs/dk!dutoit!exec`
is the process that has mounted itself on `/cs/exec` in machine ‘dutoit’.

Ipcpath, forms a network name from its arguments and returns a pointer to it. It takes three arguments: the destination *name,* the default *network,* and the default *service.* It assumes that *name* is a three part name of the form: network!host!service. If either network or service is missing from *name, ipcpath* supplies them from the default arguments. It then tacks a `/cs` on the front and returns a pointer to that. Thus,

```
ipcpath("dutoit", "dk", "dcon")
```

returns a pointer to the string `/cs/dk!dutoit!dcon`.

Ipcopen places a call to a named network end point and returns a file descriptor for the resulting connection. *Param,* a whitespace-delimited string of values, specifies properties which the connection must have. At present four parameter values are defined:

`heavy`
`light` Heavy (usually computer-to-computer) or light (computer-to-terminal) traffic is expected.
`delim` The connection must support delimiters; see *stream*(4).
`hup` `SIGHUP` must be generated at end of file; see *signal*(2).

Ipccreat attaches a process to a name space. It returns a file descriptor representing the attachment. *Name* and *param* mean the same as for *ipcopen.*

Ipclisten waits for calls (from *ipcopen* in other processes) appearing on file descriptor *fd* (obtained from *ipccreat*). When a call arrives, it returns an `ipcinfo` structure, defined in `<ipc.h>`:

```
typedef struct {
        int  reserved[5];
        char *name;       that being dialed
        char *param;      parameters used to set up call
        char *machine;    machine id of caller
        char *user;       user name of caller
        int  uid, gid;    uid, gid of caller
} ipcinfo;
```

The call may be accepted by *ipcaccept* or rejected by *ipcreject. Ipcaccept* returns a file descriptor for the connection. *Ipcreject* takes an integer error number and an error message string, which will be passed back to the caller as *errno* and *errstr.*

A higher-level routine, *ipcexec,* executes the command, *cmd,* on a named machine. The file descriptor returned by *ipcexec* is the standard input, standard output, and standard error of the command. As in *ipcopen, param* lists properties required of the channel.

Once a connection is established using *ipcopen* it is often necessary to authenticate yourself to the destination. This is done using *ipclogin* and *ipcrogin. Ipclogin* runs the client side of the authentication protocol described in *svcmgr*(8) for the *v9auth* action. The supplied *fd* is the descriptor returned by *ipcopen.* Until the authentication is accepted, *ipclogin* will prompt the user (using `/dev/tty`) for a login id and password to be sent over *fd.*

Ipcrogin runs the client side of the authentication protocol used by BSD's *rlogin* and *rsh* services. Unlike *ipclogin,* it will not prompt the user if the authentication fails. *Ipcrogin* takes a second argument that is written to *fd* after the authentication is accepted.

EXAMPLES

To connect to the voice synthesizer attached to the Datakit:

```
#include <ipc.h>
main() {
        int fd;
        fd = ipcopen(ipcpath("voice", "dk", 0), "light");
        if(fd<0){
                printf("can't connect: %s\n", errstr);
                exit(1);
```

```
        }
        ...
        close(fd);
}
```

To place a Dataphone call via Datakit; the service name is derived in an obvious way from the ACU service class; see *dialout*(3).

```
        fd = ipcopen(ipcpath("9-1-201-582-0000", "dk", "dial1200"), "light");
```

To announce as a local service and wait for incoming calls:

```
#include <ipc.h>
main() {
        int fd;
        ipcinfo *ip;
        fd = ipccreat("/tmp/service", 0);
        if(fd<0){
                printf("can't announce: %s\n", errstr);
                exit(1);
        }
        while(ip = ipclisten(fd)){
                int nfd;
                if(i_hate_this_user(ip->machine, ip->user)) {
                        ipcreject(ip, EACCES, "i hate you");
                        continue;
                }
                nfd = ipcaccept(ip);
                ...
                close(nfd);
        }
        printf("lost the announced connection somehow\n");
        exit(1);
}
```

FILES

`/cs/dk`	the announce point for the Datakit dialer
`/cs/tcp`	the announce point for the internet dialer

SEE ALSO

dialout(3), *connld*(4), *dkmgr*(8), *svcmgr*(8), *tcpmgr*(8)

D. L. Presotto, 'Interprocess Communication in the Eighth Edition UNIX System', this manual, Volume 2

DIAGNOSTICS

Integer return values of −1 and pointer return values of 0 denote error. *Errno* contains an error code (see *intro*(2)) and *errstr* points to an explanatory string.

BUGS

Files created by *ipccreat* in the local name space are not removed when the file descriptor returned by *ipccreat* is closed.

Information in `ipcinfo` is no more trustworthy than its origin. Information, such as user name, sent by foreign machines may be suspect. On Ethernet or dialup connections (but not on Datakit) machine names can be forged. Let's not even think about wire-swappers and wiretappers.

NAME

l3tol, ltol3 – convert between 3-byte integers and long integers

SYNOPSIS

```
l3tol(lp, cp, n)
long *lp;
char *cp;

ltol3(cp, lp, n)
char *cp;
long *lp;
```

DESCRIPTION

L3tol converts a list of *n* three-byte integers packed into a character string pointed to by *cp* into a list of long integers pointed to by *lp*.

Ltol3 performs the reverse conversion from long integers (*lp*) to three-byte integers (*cp*).

These functions are useful for file-system maintenance where the block numbers are three bytes long.

SEE ALSO

filsys(5)

NAME

malloc, free, realloc, calloc, cfree − memory allocator

SYNOPSIS

```
char *malloc(size)
unsigned size;

free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

cfree(ptr)
char *ptr;
```

DESCRIPTION

Malloc and *free* provide a simple memory allocation package. *Malloc* returns a pointer to a new block of at least *size* bytes. The block is suitably aligned for storage of any type of object. No two active pointers from *malloc* will have the same value.

The argument to *free* is a pointer to a block previously allocated by *malloc*; this space is made available for further allocation.

Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. The call `realloc((char*)0, size)` means the same as `malloc(size)`.

Calloc allocates space for an array of *nelem* elements of size *elsize.* The space is initialized to zeros. *Cfree* frees such a block.

SEE ALSO

galloc(3), *brk*(2), *pool*(3), *block*(3)

DIAGNOSTICS

Malloc, realloc and *calloc* return 0 if there is no available memory or if the arena has been detectably corrupted.

BUGS

When *realloc* returns 0, the block pointed to by *ptr* may have been destroyed.

User errors can corrupt the storage arena. The most common gaffes are (1) freeing an already freed block, (2) storing beyond the bounds of an allocated block, and (3) freeing data that was not obtained from the allocator. To help find such errors, a diagnosing allocator may be loaded; use flag `-ldmalloc` of *cc*(1). An even more stringently checking version may be created by recompilation; see the source.

NAME

map − associative array classes

SYNOPSIS

```
#include <Map.h>

Mapdeclare(S,T)
Mapimplement(S,T)

struct Map(S,T) {
        Map(S,T)();
        Map(S,T)(const T&);
        Map(S,T)(const Map(S,T)&);
        ~Map(S,T);
        Map(S,T)& operator= (const Map(S,T)&);
        T& operator[] (int);
        int size();
        Mapiter(S,T) element(const S&);
        Mapiter(S,T) first();
        Mapiter(S,T) last();
};

struct Mapiter(S,T) {
        Mapiter(S,T) (const Map(S,T)&);
        ~Mapiter(S,T);
        operator int();
        S key();
        T value();
        Mapiter(S,T)& operator++ (Mapiter(S,T)&);
        Mapiter(S,T)& operator-- (Mapiter(S,T)&);
};
```

DESCRIPTION

A *map* is a collection of *elements*, each of which contains a *key* part of type *S* and a *value* part of type *T*, where *S* and *T* are type names. Both *S* and *T* must have value semantics: assignment or initialization have the effect of copying. (It is unlikely for *S* and *T* to be pointers.)

Map elements are ordered by key: type *S* must have a transitive boolean `operator<`.

The macro call `Mapdeclare(S,T)` declares the classes `Map(S,T)` and `Mapiter(S,T)`. It must appear once in every source file that uses either. The macro call `Mapimplement(S,T)` defines the functions that implement the map classes. It must appear exactly once in the entire program.

Map constructors

`Map(S,T)()` An empty map. The value part of future elements is the value of an otherwise uninitialized static object of type *T* .

`Map(S,T)(x)` An empty map whose future elements have default value *x* .

`Map(S,T)(m)` A copy of map *m* obtained by copying the elements and default value of *m.*

Map operators

n = *m* All the elements of map *n* are deleted and and copies of the elements of *m* are added. The default value of *n* does not change. Running time is $O(\log(|m|) + \log(|n|))$, where $|m|$ means *m*`.size()`.

m`[`*k*`]` A reference to the value part of the element of map *m* with key `k`. If the element does not exist, it is created. Running time is $O(\log(|m|))$.

Other Map functions

m`.size()` The number of elements in `m`. Running time is $O(1)$.

m`.element(`*k*`)`
A map iterator referring to the element of *m* with key *k* if such an element exists. Otherwise the result is vacuous. Running time is $O(\log(|m|))$.

m`.first()`
m`.last()` A map iterator referring to the element of *m* with the smallest (or largest) key. If `m` has no elements, the result is vacuous. Running time is $O(\log(|m|))$.

Map iterators

For every class `Map(`*S*`,`*T*`)` there is a class `Mapiter(`*S*`,`*T*`)`. A map iterator identifies a map object and possibly an element in that map. An iterator that does not identify an element is *vacuous*.

Mapiter constructors

`Mapiter(`*S*`,`*T*`)(`*m*`)`
A vacuous iterator referring to map *m*. Running time is $O(1)$.

Mapiter operators

i `=` *j* Make iterator *i* refer (for now) to the same map as does *j*.

`(int)`*i* Zero if iterator *i* is vacuous, otherwise nonzero.

`++`*i*
`--`*i* If iterator *i* is vacuous, make it refer to the map element with the smallest (or largest) key Otherwise, make it refer to the map element with the next key greater (or less) than the key of the current element. If no such element exists, *i* becomes vacuous. The running time of a single increment operation for map *m* is $O(\log(|m|))$. However an iterator takes only time $O(|m|)$ to sequence through the whole map.

Other mapiter functions

i`.key()`
i`.value()` The key (or value) part of the element referred to by *i* . If *i* is vacuous, return the value of an otherwise uninitialized static object of appropriate type. Running time is $O(1)$.

EXAMPLES

```
struct city { char name[100]; };
typedef int population;
int operator< (const city&, const city&);

Mapdeclare(name,population)

Map(name,population) gazetteer;

// Print big cities; set populations of others to zero.

      for(Mapiter(name,population) i = gazetteer.first(); i; i++)
            if(i.value() > 1000000)
                  printf("%s\n", i.key().name);
            else
                  gazetteer[i.key()] = 0;
```

BUGS

A ‘type name’ `Map(`*S*`,`*T*`)` or `Mapiter(`*S*`,`*T*`)` that contains spaces will be mangled by *cpp*(8).
There is no way to delete a single element.
Ambiguities can occur if the type name *S* contains an underscore.
No precautions are taken against running out of memory.

NAME

memccpy, memchr, memcmp, memcpy, memmove, memset – memory operations

SYNOPSIS

```
char *memccpy(s1, s2, c, n)
char *s1, *s2;
int c, n;

char *memchr(s, c, n)
char *s;
int c, n;

int memcmp(s1, s2, n)
char *s1, *s2;
int n;

char *memcpy(s1, s2, n)
char *s1, *s2;
int n;

char *memmove(s1, s2, n)
char *s1, *s2;
int n;

char *memset(s, c, n)
char *s;
int c, n;
```

DESCRIPTION

These functions operate efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

Memccpy copies characters from memory area *s2* into *s1*, stopping after the first occurrence of character *c* has been copied, or after *n* characters have been copied, whichever comes first. It returns a pointer to the character after the copy of *c* in *s1*, or zero if *c* was not found in the first *n* characters of *s2*.

Memchr returns a pointer to the first occurrence of character *c* in the first *n* characters of memory area *s*, or zero if *c* does not occur.

Memcmp compares its arguments, looking at the first *n* characters only, and returns an integer less than, equal to, or greater than 0, according as *s1* is lexicographically less than, equal to, or greater than *s2*.

Memcpy copies *n* characters from memory area *s2* to *s1*. It returns *s1*.

Memmove is the same as *memcpy*, except it is guaranteed to handle overlapping strings as if the move had been made to a temporary and then to the destination.

Memset sets the first *n* characters in memory area *s* to the value of character *c*. It returns *s*.

SEE ALSO

string(3)

BUGS

Memcmp uses native character comparison, which is signed on some machines, unsigned on others; thus the sign of the value returned when a character has its high-order bit set is implementation-dependent.
Thanks to ANSI X3J11 for the *memcpy/memmove* distinction.

NAME

mktemp, tmpnam – make a unique file name

SYNOPSIS

```
char *mktemp(template)
char *template;

#include <tmpnam.h>

char *tmpnam(s)
char s[L_tmpnam];
```

DESCRIPTION

Mktemp replaces *template* by a unique file name, and returns the address of the template. The template should look like a file name with six trailing Xs, which will be replaced with the current process id and a unique letter.

Tmpnam places in the string pointed to by *s* a unique file name referring to the standard `/tmp` directory for temporary files and returns *s*. If *s* is 0, *tmpnam* returns the address of a fixed internal buffer that contains the name. (Note: it is bad form to leave files in the temporary directory.)

SEE ALSO

getpid in *getuid*(2)

BUGS

After many calls to *tmpnam*, the resulting filenames may have strange characters.

NAME

monitor – prepare execution profile

SYNOPSIS

```
monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)(), (*highpc)();
short buffer[];
```

DESCRIPTION

An executable program created by `cc -p` automatically includes calls for *monitor* with default parameters; *monitor* needn't be called explicitly except to gain fine control over profiling.

Monitor is an interface to *profil*(2). *Lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* bytes. *Monitor* arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of *lowpc* and the highest is just below *highpc*. At most *nfunc* call counts can be kept; only calls of functions compiled with the profiling option `-p` of *cc*(1) are recorded. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled. The default values for *bufsize* and *nfunc* are (*highpc*−*lowpc*)/8 and 300 respectively.

To profile the entire program, use

```
extern etext();
. . .
monitor((int (*)())2, etext, buf, bufsize, nfunc);
```

Etext lies just above all the program text, see *end*(3). For the highest resolution profiling on the VAX, use `bufsize = ((int)highpc)-((int)lowpc)+12+8*nfunc`.

To stop execution monitoring and write the results on the file *mon.out*, use

```
monitor((int (*)())0);
```

then *prof*(1) can be used to examine the results.

FILES

`mon.out`

SEE ALSO

prof(1), *profil*(2), *cc*(1)

NAME

itom, mfree, madd, msub, mult, mdiv, sdiv, msqrt, mgcd, min, mout, fmin, fmout, move, mcmp, rpow, mpow – multiple precision integer arithmetic

SYNOPSIS

```
#include <mp.h>
#include <stdio.h>

mint *itom(n)
short n;

mfree(a)
mint *a;

madd(a, b, c)
mint *a, *b, *c;

msub(a, b, c)
mint *a, *b, *c;

mult(a, b, c)
mint *a, *b, *c;

mgcd(a, b, c)
mint *a, *b, *c;

mdiv(a, b, q, r)
mint *a, *b, *q, *r;

sdiv(a, n, q, r)
mint *a, *q;
short n, *r;

msqrt(a, b, r)
mint *a, *b, *r;

rpow(a, n, c)
mint *a, *c;

mpow(a, b, m, c)
mint *a, *b, *m, *c;

move(a, b)
mint *a, *b;

mcmp(a, b)
mint *a, *b;

int min(a)
mint *a;

mout(a)
mint *a;

int fmin(a, f)
mint *a;
FILE *f;

fmout(a, f)
mint *a;
FILE *f;
```

DESCRIPTION

These routines perform arithmetic on arbitrary-length integers of defined type *mint*. The functions are obtained with the *ld*(1) option `-lmp`.

Pointers to *mint* must be initialized using the function *itom*, which sets the initial value to *n*. Thereafter space is managed automatically. The space may be freed by *mfree*, making the variable uninitialized.

Madd, *msub*, *mult*, and *mgcd* assign to their third arguments the sum, difference, product, and greatest common divisor, respectively, of their first two arguments.

Mdiv assigns the quotient and remainder, respectively, to its third and fourth arguments. The remainder is nonnegative and less than the divisor in magnitude. *Sdiv* is like *mdiv* except that the divisor is an ordinary integer.

Msqrt assigns the square root and remainder to its second and third arguments, respectively.

Rpow calculates *a* raised to the power *n*; *mpow* calculates this reduced modulo *m*.

Move assigns (by copying) the value of its first argument to its second argument.

Mcmp returns a negative, zero, or positive integer if the value of its first argument is less than, equal to, or greater than, respectively, the value of its second argument.

Min and *mout* do decimal conversion from `stdin` and to `stdout`, *fmin* and *fmout* use file *f*; see *stdio*(3). *Min* and *fmin* return `EOF` on end of file.

DIAGNOSTICS

Illegal operations and running out of memory produce messages and core images.

BUGS

Itom and *sdiv* fail if *n* is the most negative short integer.

NAME

nlist – get entries from name list

SYNOPSIS

```
#include <nlist.h>
nlist(filename, nl)
char *filename;
struct nlist nl[];
```

DESCRIPTION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The list is terminated with a null name.

```
struct nlist {
        char *n_name;           symbol name
        unsigned char n_type;   type flag
        char n_other;           unused
        short n_desc;           unused
        unsigned long n_value;  value (or offset) of this symbol
};
```

Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in `n_type` and `n_value` respectively. If the name is not found, both entries are set to 0.

This subroutine is useful for examining the system name list kept in `/unix`. In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out(5)

DIAGNOSTICS

All type entries are set to 0 if the file cannot be found or if it is not a valid namelist.

BUGS

On some UNIX systems you must include `<a.out.h>` rather than `<nlist.h>`. This is unfortunate, but `<a.out.h>` can't be used on the VAX because it contains a union, which can't be initialized.

NAME

perror, sys_errlist, sys_nerr – system error messages

SYNOPSIS

```
perror(s)
char *s;

int sys_nerr;
char *sys_errlist[];
```

DESCRIPTION

Perror produces a short error message on the standard error file describing the last error encountered during a call to the system from a C program. First the argument string *s* is printed, then a colon, then the message and a new-line. The error number is taken from the external variable *errno* (see *intro*(2)), which is changed only when errors occur.

Sys_errlist is a vector of message strings. *Errno* can be used as an index in this table to get the message string without the newline. *Sys_nerr* is the number of messages in the table.

SEE ALSO

intro(2)

NAME

picopen_r, picopen_w, picread, picwrite, picclose, picputprop, picgetprop, picunpack, picpack, picerror − picture file I/O

SYNOPSIS

```
#include <picfile.h>

PICFILE *picopen_r(name)
char *name;

PICFILE *picopen_w(name, type, x0, y0, w, h, chan, argv, cmap)
char *name, *type, *chan, *argv[], *cmap;

int picread(pf, buf)
PICFILE *pf;
char *buf;

int picwrite(pf, buf)
PICFILE *pf;
char *buf;

void picclose(pf)
PICFILE *pf;

PICFILE *picputprop(pf, name, value)
PICFILE *pf;
char *name, *value;

char *picgetprop(pf, name)
PICFILE *pf;
char *name;

void picunpack(pf, pix, fmt, arg ...)
PICFILE *pf;
char *pix, *fmt;

void picpack(pf, pix, fmt, arg ...)
PICFILE *pf;
char *pix, *fmt;

void picerror(string)
char *string;
```

DESCRIPTION

These functions read and write raster images in *picfile*(5) format. They are loaded by option `-lpicfile` of *ld*(1). Open picture files are referred to by pointers of type `PICFILE*`.

Picopen_r opens the named picfile for reading and returns a pointer to the open file. If *name* is `"IN"`, standard input is used.

Picopen_w similarly creates the named image file for writing. The name `"OUT"` refers to standard output. *Type* is a `TYPE` attribute, as described in *picfile*(5); *x0* and *y0* are the upper left coordinates of the `WINDOW` attribute; *w* and *h* are the image width and heigth in pixels. *Chan* is a string specifying the order of channels for the `CHAN` attribute; the length of this string becomes the value of `NCHAN`. *Argv*, if nonzero, is conventionally the second argument of the main program; see *exec*(2). It becomes a `COMMAND` attribute recording the provenance of the file.

The special call `picopen_w(name, PIC_SAMEARGS(pf))` creates a file with the same attributes as an already open picfile. `PIC_SAMEARGS` mentions *argv* by name, hence the name must be visible at the point of call.

Picread and *picwrite* read or write a single row of pixels using the character array *buf*. The length of the row is determined from the file's `WINDOW` and `NCHAN` attributes. One-bit-per-pixel images (of type

`bitmap` or `ccitt-g4`, for example) are decoded to one byte per pixel, 0 for black, 255 for white, and are encoded as 1 for pixel values less than 128 and 0 otherwise. Files of type `ccir601` are decoded into conventional `rgb` channels.

Picclose closes a picfile and frees associated storage.

Picputprop called after *picopen_w* but before *picwrite* adds header attributes, returning a (probably changed) value of the `PICFILE` pointer.

Picgetprop returns a pointer to the value of the named attribute, or 0 if the picfile does not have the attribute. In both *Picputprop* and *picgetprop,* with multiple appearances (e.g. `COMMAND`) are expressed as a sequence of values separated by newlines.

The header file defines macros to extract commonly-used attributes:

```
PIC_NCHAN(pf), PIC_WIDTH(pf), PIC_HEIGHT(pf),
PIC_SAMEARGS(pf)  (see picopen_w)
```

Picunpack extracts the channels of pixel array *pix* into separate array *args* of types described by the *fmt* character string. Format characters are `c`, `s`, `l`, `f`, `d`, for arrays of types unsigned char, short, long, float, and double. Format character _ designates a picfile channel to be skipped. *Picpack* reverses the process. These routines effect a standard machine-independent byte ordering.

Picerror prints messages for errors resulting from calls to *picfile* routines. (*Perror*(3) cannot describe some error conditions, like malformed header lines.)

EXAMPLES

Unpack the green and z channels from a file with channels `rgbz...`

```
PICFILE *pf = picopen_r("file");
extern char pixels[], green[][1000];
extern float zdepth[][1000];
for(i=0; picread(pf, pixels); i)
      picunpack(pf, pixels, "_c_f", green[i], zdepth[i]);
```

Reflect a picture about its vertical midline.

```
PICFILE *in = picopen_r("picture");
PICFILE *out = picopen_w("OUT", PIC_SAMEARGS(in));
int w = PIC_WIDTH(in);
int n = PIC_NCHAN(in);
char *buffer = malloc(w*n), *temp = malloc(n);
while (picread(in, buffer)) {
      char *left = buffer;
      char *right = buffer + n*(w - 1);
      for( ; left<right; left+=n, right-=n) {
            strncpy(temp, left, n);
            strncpy(left, right, n);
            strncpy(right, temp, n);
      }
      picwrite(out, buffer);
}
```

SEE ALSO

picfile(5), *pico*(1), *bcp*(1)

DIAGNOSTICS

Picread returns 1 on success, 0 on end of file or error.
Picopen_r and *picopen_w* return 0 for unopenable files.

BUGS

Picpack and *picunpack* store and retrieve floating point channels (types `f` and `d`) using native floating-point, rather than something machine independent like IEEE format.

NAME

vec, move, etc. – plot graphics interface

SYNOPSIS

```
#include <plot.h>

openpl(s)
char *s;

closepl()

erase()

move(x, y)
double x, y;

rmove(dx, dy)
double dx, dy;

point(x, y)
double dx, dy;

vec(x, y)
double x, y;

rvec(dx, dy)
double dx, dy;

line(x1, y1, x2, y2)
double x1, y1, x2, y2;

arc(x1, y1, x2, y2, x, y, r)
double x1, y1, x2, y2, x, y, r;

circle(xc, yc, r)
double xc, yc, r;

box(x1, y1, x2, y2)
double x1, y1, x2, y2;

sbox(x1, y1, x2, y2)
double x1, y1, x2, y2;

parabola(x1, y1, x2, y2, x3, y3)
double x1, y1, x2, y2, x3, y3;

fill(n, arr) int n[];
double *arr[];

poly(n, arr) int n[];
double *arr[];

spline(n, arr)
int n[];
double *arr[];

cspline(n, arr)
int n[];
double *arr[];

fspline(n, arr)
int n[];
double *arr[];

lspline(n, arr)
int n[];
double *arr[];

dspline(n, arr)
int n[];
double *arr[];

text(s)
char *s;

color(s)
char *s;

cfill(s)
char *s;

pen(s)
char *s;

range(x1, y1, x2, y2)
double x1, y1, x2, y2;

frame(x1, y1, x2, y2)
double x1, y1, x2, y2;

grade(x)
double x;

save()

restore()

ppause()
```

DESCRIPTION

These functions generate either a device-independent graphic stream (see *plot*(5)) or device-dependent graphics commands. The include file `<plot.h>` is used only for device-independent output. An alternative include file, `<iplot.h>`, supports device-independent output using identically named functions of integer, instead of double, arguments.

Libraries for different devices are loaded with the following *ld*(1) flags:

`-lplot`	general stream output
`-l2621`	HP2621 terminal
`-l4014`	Tektronix 4014 terminal
`-ltr`	Troff input, tuned for the Mergenthaler Linotron 202 phototypesetter

`-lpen` HP7580 pen plotter
`-l5620` 5620 terminal running *mux*

String arguments are null-terminated and may not contain embedded newlines. For details on string arguments, see *plot*(5). *Poly*, *fill*, and the various spline functions take an integer array and an array of pointers to double floating point arrays. The integers specify the number of vertices (*x*-*y* pairs) in the floating point array. The last integer entry should be 0.

SEE ALSO

plot(1), *plot*(5)

BUGS

The `-ltr` library should be tuned for PostScript.

NAME

poly_lk, poly_read – polyhedron database routines

SYNOPSIS

```
#include <poly.h>

int poly_lk(name)
char *name;

int poly_read(p, dir, n)
Polyhedron *p;
char *dir;
```

DESCRIPTION

These routines access the *poly*(7) database of polyhedra.

Poly_lk tries to interpret *name* as a polyhedron reference. If it is a number, it returns that number. Otherwise, it returns the number of the first polyhedron for which *name* is a prefix of the polyhedron's name.

Poly_read forms an in-core description of the polyhedron number *n* in the directory *dir*. If *dir* is 0, the normal directory (`/usr/lib/polyhedra`) is used.

SEE ALSO

poly(5), *poly*(7)

DIAGNOSTICS

Poly returns −1 on unknown names.

Poly_read returns zero on success, nonzero on error.

NAME

pool − fast memory allocation

SYNOPSIS

```
#include <Pool.h>

struct Pool {
      Pool(unsigned);
      ~Pool();
      void* alloc();
      void free(void*)
};
```

DESCRIPTION

Every `Pool` is a collection of *elements*, each of which is an array of bytes. All elements of a pool are the same size. Pool functions are

`Pool(`*n*`)` Construct a pool whose elements are of size *n*.

p`.alloc()` Allocate a new element in pool `p`. Return a pointer to the element.

p`.free(`*ep*`)` Free the element of *p* pointed to by *ep*. The element must have been allocated from *p*.

Destroying a pool frees all the memory occupied by its elements.

The memory in a pool element is aligned on the same boundary as memory returned by *malloc*(3) so that it may be used to contain an object of any type. In typical use, there would be one pool per class, with the pool known only to the `new` and `delete` operators of that class.

Performance

Pool memory is allocated in chunks that are typically about 1,000 bytes each. Once a chunk is allocated to a particular pool, that chunk is only released when the pool itself is destroyed.

Elements are allocated inline except when a new chunk must be added to the pool. Elements are always freed inline.

EXAMPLES

```
#include <Pool.h>

struct Mytype {
      static Pool mypool;
      // constructors and members
      void* operator new(unsigned) { return mypool.alloc(); }
      void operator delete(void* p) { mypool.free(p); }
};

Pool Mytype::mypool(sizeof(Mytype));
```

SEE ALSO

malloc(3)

NAME

popen, ppopen, vepopen, pclose – open a pipe to/from a process

SYNOPSIS

```
#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

FILE *ppopen(command, type)
char *command, *type;

FILE *vepopen(command, type, args, env)
char *command, *type, **args, **env;

int pclose(stream)
FILE *stream;
```

DESCRIPTION

The first argument to *popen* is a pointer to a null-terminated string containing a command line for *sh*(1). *Type* is as in *fopen*(3). *Popen* creates a pipe between the calling process and the command and returns a stream pointer that can be used to write to the standard input of the command or to read from the standard output.

Ppopen uses the `-p` shell flag to restrict the environment of the shell. Both *popen* and *ppopen* set the effective userid to the real userid before calling the shell.

Vepopen has arguments akin to those of *execve* (see *exec*(2)): a file to be executed, a mode as above, a null-terminated vector of argument strings, and a null-terminated vector of environment strings. The shell is not called, and the effective userid is preserved.

A stream opened by these routines should be closed by *pclose*, which waits for the associated process to terminate and returns the exit status of the command.

Because the *command* inherits open files, in particular standard input and output, a type `"r"` call may be used to insert a filter in the input, and type `"w"` in the output.

SEE ALSO

exec(2), *pipe*(2), *fopen*(3), *stdio*(3), *system*(3)

DIAGNOSTICS

Popen returns a null pointer if files or processes cannot be created, or the Shell cannot be accessed.

Pclose returns −1 if there is no process to wait for.

BUGS

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be forestalled by calling *fflush*; see *fopen*(3).

The resetting of the userid is probably gratuitous; it is there as a defense against incautious use of the routine by set-uid programs.

I/O type `"r+w"` exists but is not useful.

NAME

port – mathematical library for Fortran

DESCRIPTION

The Port library of hundreds of scientific subroutines covers approximation, ordinary differential equations, partial differential equations, linear algebra, optimization and mathematical programming, quadrature, differentiation, roots, special functions, and transforms. It is built upon a framework of service routines for error handling, stack management, and machine constant parameterization.

The routines are loaded by the *ld*(1) option `-lport`.

The manual describes the software and gives examples.

SEE ALSO

P. A. Fox, *The PORT Mathematical Subroutine Library,* AT&T Bell Laboratories, May 8, 1984.
P. A. Fox, *The PORT Mathematical Subroutine Library Installation Manual,* AT&T Bell Laboratories, September, 1984.

NAME

print, fprint, sprint, fmtinstall – print formatted output

SYNOPSIS

```
int print(format [ , arg ] ... )
char *format;

int fprint(fildes, format [ , arg ] ... )
int fildes;
char *format;

int sprint(s, format [ , arg ] ... )
char *s, *format;

fmtinstall(c, fn)
char c;
int (*fn)();

strconv(s, f1, f2)
char *s;

extern int printcol;
```

DESCRIPTION

Print places output on the standard output. *Fprint* places output on the named output file descriptor; a buffered form is described in *fio*(3). *Sprint* places output followed by the null character (`\0`) in consecutive bytes starting at *s*; it is the user's responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the `\0` in the case of *sprint*), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its trailing arguments under control of a *format* string. The *format* contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more arguments. The results are undefined if there are arguments of the wrong type or too few arguments for the format. If the format is exhausted while arguments remain, the excess are ignored.

Each conversion specification has the following format

```
% [flags] [[−] digits [. digits]] verb
```

The *flags* modify the meaning of the conversion verb. The first (possibly negative) number is called *f1*, the second number is *f2*. The flags and numbers are arguments to the *verb* described below.

The numeric verbs `d`, `o`, and `x` format their arguments in decimal, octal and hex respectively. Each interprets the flags `h`, `l`, `u`, to mean short, long, and unsigned. If neither short nor long is specified, then the arg is an `int`. If unsigned is specified, then the arg is interpreted as a positive number and no sign is output. *F1* is the minimum field width and, if negative, means left justified rather than right justified; in both cases, padding is done with blanks. The converted number is padded with `0` on the left to at least *f2* characters.

The floating point verbs `f`, `e`, and `g` take a double argument. No flags apply to floating point conversions. *F1* is the minimum field width and, if negative, means left justified. *F2* is the number of digits that are converted after the decimal place. The first unconverted digit has suffered decimal rounding. The `f` verb produces output of the form `[−]digits[.digits]`. `e` conversion appends an exponent `e[−]digits`. The `g` verb will output the arg in either `e` or `f` with the goal of producing the smallest output.

The `s` verb copies a string (pointer to character) to the output. The number of characters copied (*n*) is the minimum of the size of the string and *f2*. These *n* characters are justified within a field of *f1* characters as described above.

The `c` verb copies a single character (int) justified within a field of *f1* characters as described above.

Fmtinstall is used to install your own conversions and flags. *Fn* should be declared as

```
int fn(o, f1, f2, f3)
void *o;
int f1, f2, f3;
```

Fn is passed a pointer *o* to whatever argument appears in the list to *print*. *Fn* should return the size of the argument in bytes so *print* can skip over it. *F1* and *f2* are the decoded numbers in the conversion. A missing *f1* is denoted by the value zero. A missing *f2* is denoted by a negative number. *F3* is the logical or of all the flags seen in the conversion. If *c* is a flag, *fn* should return a negative number that is negated and then logically *or*ed with any other flags and ultimately passed to a conversion routine. All interpretation of *f1*, *f2*, and *f3* is left up to the conversion routine. The standard flags are `h`(2), `l`(1), and `u`(4).

Sprint is designed to be recursive in order to help prepare output in custom conversion routines.

The output of any conversion routine must be passed through *strconv*. *S* is the character string, *f1* and *f2* have the same meaning as above.

Printcol indicates the position of the next output character. Tabs, backspaces and carriage returns are interpreted appropriately.

EXAMPLES

This adds a verb to print complex numbers.

```
typedef struct {
      double r, i;
} complex;
complex x = { 1.5, -2.3 };
int Xconv();

main()
{

      fmtinstall('X', Xconv);
      print("x = %X\n", x);
}

Xconv(o, f1, f2, f3)
      complex *o;
{
      char str[50];

      sprint(str, "(%g,%g)", o->r, o->i);
      strconv(str, f1, f2);
      return(sizeof(complex));
}
```

SEE ALSO

fio(3), *printf*(3)

BUGS

There are internal buffers which may overflow silently.

NAME

printf, fprintf, sprintf, snprintf – print formatted output

SYNOPSIS

```
#include <stdio.h>

int printf(char *format, ... );

int fprintf (FILE *stream, char *format, ... );

int sprintf (char *s, char *format, ... );

int snprintf (char *s, int len, char *format, ... );
```

DESCRIPTION

Printf places output on the standard output stream *stdout*. *Fprintf* places output on the named output *stream*. *Sprintf* places output followed by the null character (`\0`), in consecutive bytes starting at *s*; it is the user's responsibility to ensure that enough storage is available. *Snprintf* corresponds to *sprintf* except that no more than *len* bytes are placed into *s*. Each function returns the number of characters transmitted (not including the `\0` in the case of *sprintf*), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its trailing arguments under control of a *format* string. The *format* contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more arguments. The results are undefined if there are arguments of the wrong type or too few arguments for the format. If the format is exhausted while arguments remain, the excess are ignored.

Each conversion specification is introduced by the character `%`. After the `%`, the following appear in sequence:

Zero or more *flags*, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it is padded to the field width. When the width specification begins with zero, padding is with leading zeros. Otherwise padding is with leading spaces (trailing spaces, with the left-adjustment flag –, described below) to the field width.

A *precision* that gives the minimum number of digits to appear for the `d`, `o`, `u`, `x`, or `X` conversions, the number of digits to appear after the decimal point for the `e` and `f` conversions, the maximum number of significant digits for the `g` conversion, or the maximum number of characters to be printed from a string in `s` conversion. The precision takes the form of a period (`.`) followed by a decimal digit string; a null digit string is treated as zero.

An optional `l` (ell) specifying that a following `d`, `o`, `u`, `x`, or `X` conversion character applies to a long integer *arg*. An `l` before any other conversion character is ignored.

A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (`*`) or an exclamation point (`!`) instead of a digit string. In this case, an integer *arg* supplies the field width or precision.

The flag characters and their meanings are:

– The result of the conversion is left-justified within the field.

\+ The result of a signed conversion always begins with a sign (+ or –).

blank If the first character of a signed conversion is not a sign, a blank is prefixed to the result. This implies that if the blank and + flags both appear, the blank flag is ignored.

\# This flag specifies that the value is to be converted to an ''alternate form.'' For `c`, `d`, `s`, and `u` conversions, the flag has no effect. For `o` conversion, it increases the precision to force the first digit of the result to be a zero. For `x or X` conversion, a non-zero result has `0x or 0X` prefixed to it. For `e`, `E`, `f`, `g`, and `G` conversions, the result always contains a decimal point, even if no digits follow the point (normally, a decimal point appears in the

result of these conversions only if a digit follows it). For `g` and `G` conversions, trailing zeros are *not* be removed from the result as they normally are.

The conversion characters and their meanings are:

`d,o,u,x,X` The integer *arg* is converted to signed decimal, unsigned octal, decimal, or hexadecimal notation (`x` and `X`), respectively; the letters `abcdef` are used for `x` conversion and the letters `ABCDEF` for `X` conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading zeros. (For compatibility with other versions of *printf*, a field width with a leading zero results in padding with leading zeros. This does not imply an octal value for the field width.) The default precision is 1. The result of converting a zero value with a precision of zero is a null string.

`f` The float or double *arg* is converted to decimal notation in the style [-]*d.ddd*, where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, six digits are output; if the precision is explicitly `0`, no decimal point appears.

`e, E` The float or double *arg* is converted in the style [-]*d.ddd***e**±*dd*, where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, six digits are produced; if the precision is zero, no decimal point appears. The `E` format code produces a number with `E` instead of `e` introducing the exponent. The exponent always contains at least two digits.

`g, G` The float or double *arg* is printed in style `f` or `e` (or in style `E` in the case of a `G` format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style `e` is used only if the exponent resulting from the conversion is less than -4 or greater than the precision. Trailing zeros are removed from the result; a decimal point appears only if it is followed by a digit. Precision 0 yields a result with just enough significance to round to exactly the original value when converted back to binary as by *scanf*(3).

`c` The character argument is printed.

`s` The argument is taken to be a string (character pointer) and characters from the string are printed until a null character (`\0`) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A zero value for the argument yields undefined results. (For compatibility with other versions of *printf*, a field width with a leading zero results in zero-padding the string instead of blank-padding it. This does not imply an octal value for the field width.)

`%` Print a `%`; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by *printf* and *fprintf* are printed as if *putc* had been called; see *getc*(3).

EXAMPLES

```
printf("%s, %s %d, %d:%.2d", weekday, month, day, hour, min);
```

Print a date and time in the form ‘Sunday, July 3, 10:02’, where *weekday* and *month* are pointers to null-terminated strings.

```
printf("pi = %.5f", 4*atan(1.0));
```

Print π to 5 decimal places.

SEE ALSO

ecvt(3), *scanf*(3), *stdio*(3), *print*(3)

BUGS

The `!` indicator for field width is nonstandard.

NAME

orient, normalize − map projections

SYNOPSIS

```
orient(lat, lon, rot)
double lat, lon, rot;

normalize(p)
struct place *p;
```

DESCRIPTION

Users of *map*(7) may skip to the description of 'Projection generators' below.

The functions *orient* and *normalize* plus a collection of map projection generators are loaded by option `-lmap` of *ld*(1). Most of them calculate maps for a spherical earth. Each map projection is available in one standard form, into which data must be normalized for transverse or nonpolar projections.

Each standard projection is displayed with the Prime Meridian (longitude 0) being a straight vertical line, along which North is up. The orientation of nonstandard projections is specified by *orient.* Imagine a transparent gridded sphere around the globe. First turn the overlay about the North Pole so that the Prime Meridian (longitude 0) of the overlay coincides with meridian *lon* on the globe. Then tilt the North Pole of the overlay along its Prime Meridian to latitude *lat* on the globe. Finally again turn the overlay about its 'North Pole' so that its Prime Meridian coincides with the previous position of (the overlay's) meridian *rot.* Project the desired map in the standard form appropriate to the overlay, but presenting information from the underlying globe. It is not useful to use *orient* without using *normalize*.

Normalize converts latitude-longitude coordinates on the globe to coordinates on the overlaid grid. The coordinates and their sines and cosines are input to *normalize* in a `place` structure. Transformed coordinates and their sines and cosines are returned in the same structure.

```
struct place {
        double radianlat, sinlat, coslat;
        double radianlon, sinlon, coslon;
};
```

The projection generators return a pointer to a function that converts normalized coordinates to *x-y* coordinates for the desired map, or 0 if the required projection is not available. The returned function is exemplified by *proj* in this example. Warning: coordinates are in degrees for *orient,* in radians for `place`.

```
struct place pt;
int (*proj)() = mercator();
double x, y;

orient(45.0, 30.0, 180.0);      /* set coordinate rotation */

. . .                           /* fill in the pt structure */
normalize(&pt);                 /* rotate coordinates */
if((*proj)(&pt, &x, &y) > 0)    /* project onto x,y plane */
        plot(x, y);
```

The projection function `(*proj)()` returns 1 for a good point, 0 for a point on a wrong sheet (e.g. the back of the world in a perspective projection), and −1 for a point that is deemed unplottable (e.g. points near the poles on a Mercator projection).

Scaling may be determined from the *x-y* coordinates of selected points. Latitudes and longitudes are measured in degrees for ease of specification for *orient* and the projection generators but in radians for ease of calculation for *normalize* and *proj.* In either case latitude is measured positive north of the equator, and longitude positive west of Greenwich. Radian longitude should be limited to the range $-\pi \le lon < \pi$.

Projection generators

Equatorial projections centered on the Prime Meridian (longitude 0). Parallels are straight horizontal lines.

`mercator()` equally spaced straight meridians, conformal, straight compass courses
`sinusoidal()` equally spaced parallels, equal-area, same as *bonne(0)*
`cylequalarea(lat0)` equally spaced straight meridians, equal-area, true scale on *lat0*
`cylindrical()` central projection on tangent cylinder
`rectangular(lat0)` equally spaced parallels, equally spaced straight meridians, true scale on *lat0*
`gall(lat0)` parallels spaced stereographically on prime meridian, equally spaced straight meridians, true scale on *lat0*
`mollweide()` (homalographic) equal-area, hemisphere is a circle

Azimuthal projections centered on the North Pole. Parallels are concentric circles. Meridians are equally spaced radial lines.

`azequidistant()` equally spaced parallels, true distances from pole
`azequalarea()` equal-area
`gnomonic()` central projection on tangent plane, straight great circles
`perspective(dist)` viewed along earth's axis *dist* earth radii from center of earth
`orthographic()` viewed from infinity
`stereographic()` conformal, projected from opposite pole
`laue()` *radius* = tan(2×*colatitude*), used in xray crystallography
`fisheye(n)` stereographic seen from just inside medium with refractive index n
`newyorker(r)` *radius* = log(*colatitude*/*r*): extreme 'fisheye' view from pedestal of radius *r* degrees

Polar conic projections symmetric about the Prime Meridian. Parallels are segments of concentric circles. Except in the Bonne projection, meridians are equally spaced radial lines orthogonal to the parallels.

`conic(lat0)` central projection on cone tangent at *lat0*
`simpleconic(lat0,lat1)` equally spaced parallels, true scale on *lat0* and *lat1*
`lambert(lat0,lat1)` conformal, true scale on *lat0* and *lat1*
`albers(lat0,lat1)` equal-area, true scale on *lat0* and *lat1*
`bonne(lat0)` equally spaced parallels, equal-area, parallel *lat0* developed from tangent cone

Projections with bilateral symmetry about the Prime Meridian and the equator.

`polyconic()` parallels developed from tangent cones, equally spaced along Prime Meridian
`aitoff()` equal-area projection of globe onto 2-to-1 ellipse, based on *azequalarea*
`lagrange()` conformal, maps whole sphere into a circle
`bicentric(lon0)` points plotted at true azimuth from two centers on the equator at longitudes ±*lon0*, great circles are straight lines (a stretched gnomonic projection)
`elliptic(lon0)` points are plotted at true distance from two centers on the equator at longitudes ±*lon0*
`globular()` hemisphere is circle, circular arc meridians equally spaced on equator, circular arc parallels equally spaced on 0- and 90-degree meridians
`vandergrinten()` sphere is circle, meridians as in *globular*, circular arc parallels resemble *mercator*

Doubly periodic conformal projections.

`guyou()` W and E hemispheres are square
`square()` world is square with Poles at diagonally opposite corners
`tetra()` map on tetrahedron with edge tangent to Prime Meridian at S Pole, unfolded into equilateral triangle
`hex()` world is hexagon centered on N Pole, N and S hemispheres are equilateral triangles

Miscellaneous projections.

`harrison(dist,angle)` oblique perspective from above the North Pole, *dist* earth radii from center of earth, looking along the Date Line *angle* degrees off vertical

`trapezoidal(lat0,lat1)` equally spaced parallels, straight meridians equally spaced along parallels, true scale at *lat0* and *lat1* on Prime Meridian

Retroazimuthal projections. At every point the angle between vertical and a straight line to 'Mecca', latitude *lat0* on the prime meridian, is the true bearing of Mecca.

`mecca(lat0)` equally spaced vertical meridians

`homing(lat0)` distances to 'Mecca' are true

Maps based on the spheroid. Of geodetic quality, these projections do not make sense for tilted orientations. For descriptions, see corresponding maps above.

`sp_mercator()`

`sp_albers(lat0,lat1)`

SEE ALSO

map(7), *map*(5), *plot*(3)

BUGS

Only one projection and one orientation can be active at a time.

The west-longitude-positive convention betrays Yankee chauvinism.

NAME

qsort – quicker sort

SYNOPSIS

```
qsort(base, nel, width, compar)
char *base;
int (*compar)();
```

DESCRIPTION

Qsort (quicker sort) sorts an array into nondecreasing order. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of a comparison routine to be called with pointers to elements being compared. It should be declared as

```
compar(a, b)
char *a, *b;
```

The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

SEE ALSO

sort(1)

NAME

rand, lrand, frand, nrand, srand – random number generator

SYNOPSIS

```
int rand()

long lrand()

double frand()

int nrand(val)
int val;

srand(seed)
int seed;
```

DESCRIPTION

Rand uses a linear feedback random number generator to return uniform pseudo-random numbers x, $0 \le x < 2^{15}$, with period about 2^{32}.

Lrand returns a uniform long x, $0 \le x < 2^{31}$, with period about 2^{48}.

Frand returns a uniform double x, $0.0 \le x < 1.0$, always a multiple of 2^{-31}.

Nrand returns a uniform integer x, $0 \le x < val$.

The generators are initialized by calling *srand* with whatever you like as argument. To get a different starting value each time,

```
srand((int)time((long *)0));
```

will work as long as it is not called more often than once per second.

BUGS

Rand and *lrand* are quite machine-dependent. Although *frand* and *nrand* are more portable, they appear in few versions of Unix.

NAME

re_bm, re_cw, re_re – string and pattern matching

SYNOPSIS

```
#include <re.h>

re_bm *re_bmcomp(b, e, map)
char *b, *e;
unsigned char map[256];

int re_bmexec(pat, rdfn, matchfn)
re_bm *pat;
int (*rdfn)(), (*matchfn)();

void re_bmfree(pat);
re_bm *pat;

re_cw *re_cwinit(map)
unsigned char map[256];

void re_cwadd(pat, b, e)
re_cw *pat;
char *b, *e;

void re_cwcomp(pat)
re_cw *pat;

int re_cwexec(pat, rdfn, matchfn)
re_cw *pat;
int (*rdfn)(), (*matchfn)();

void re_cwfree(pat);
re_cw *pat;

re_re *re_recomp(b, e, map)
char *b, *e;
unsigned char map[256];

re_reexec(pat, b, e, match)
re_re *pat;
char *b, *e, *match[10][2];

void re_refree(pat);
re_re *pat;

void re_rerror(str);
char *str;
```

DESCRIPTION

These routines search for patterns in strings. The *re_re* routines search for general regular expressions (defined below) using a lazily evaluated deterministic finite automaton. The more specialized and faster *re_cw* routines search for multiple literal strings using the Commentz-Walter algorithm. The still more specialized and efficient *re_bm* routines search for a single string using the Boyer-Moore algorithm. The routines handle strings designated by pointers to the first character of the string and to the character following the string.

To use the *re_bm* routines, first build a recognizer by calling *re_bmcomp*, which takes the search string and a character map; all characters are compared after mapping. The recognizer can be run (multiple times) by calling *re_bmexec*, which stops and returns the first non-positive return from either *rdfn* or *matchfn*. The recognizer calls the supplied function *rdfn* to obtain input and *matchfn* to report text matching the search string.

Rdfn should be declared as

```
int rdfn(pb, pe)
char **pb, **pe;
```

where `*pb` and `*pe` delimit an as yet unprocessed text fragment (none if `*pb==*pe`) to be saved across the call to *rdfn*. On return, `*pb` and `*pe` point to the new text, including the saved fragment. *Rdfn* returns 0 for EOF, negative for error, and positive otherwise. The first call to *rdfn* from each invocation of *re_bmexec* has `*pb==0`.

Matchfn should be declared as

```
int matchfn(pb, pe)
char **pb, **pe;
```

where `*pb` and `*pe` delimit the matched text. *Matchfn* sets `*pb`, `*pe`, and returns a value in the same way as *rdfn.*

To use the *re_cw* routines, first build the recognizer by calling *re_cwinit*, then *re_cwadd* for each string, and finally *re_cwcomp*. The recognizer is run by *re_cwexec* analogously to *re_bmexec*.

A full regular expression recognizer is compiled by *re_recomp* and executed by *re_reexec*, which returns 1 if there was a match and 0 if there wasn't. The strings that match subexpressions are returned in

array *match*. `match[0]` refers to the whole matched expression. If *match* is zero, then no match delimiters are set.

The routine *re_error* prints its argument on standard error and exits. You may supply your own version for specialized error handling.

The recognizers that these routines construct occupy storage obtained from *malloc*(3). The storage can be deallocated by *free*.

Regular Expressions

The syntax for a regular expression `e0` is

```
e3:  literal | charclass | '.' | '^' | '$' | '\'n | '(' e0 ')'

e2:  e3
  |  e2 REP
REP: '*' | '+' | '?'

e1:  e2
  |  e1 e2

e0:  e1
  |  e0 ALT e1
ALT: '|' | newline
```

A literal is any non-metacharacter or a metacharacter (one of `.*+?[]()|\^$`) preceded by `\`.

A charclass is a nonempty string *s* bracketed `[` *s* `]` (or `[^` *s* `]`); it matches any character in (or not in) *s*. In *s*, the metacharacters other than `]` have no special meaning, and `]` may only appear as the first letter. A substring *a*`-`*b*, with *a* and *b* in ascending ASCII order, stands for the inclusive range of ASCII characters between *a* and *b*.

A `\` followed by a digit *n* matches a copy of the string that the parenthesized subexpression beginning with the *n*th `(`, counting from 1, matched.

A `.` matches any character.

A `^` matches the beginning of the input string; `$` matches the end.

The `REP` operators match zero or more (`*`), one or more (`+`), and zero or one (`?`) instances respectively of the preceding regular expression `e2`.

A concatenated regular expression, `e1 e2`, matches a match to `e1` followed by a match to `e2`.

An alternative regular expression, `e0 ALT e1`, matches either a match to `e0` or a match to `e1`.

A match to any part of a regular expression extends as far as possible without preventing a match to the remainder of the regular expression.

SEE ALSO

regexp(3), *gre*(1)

DIAGNOSTICS

Routines that return pointers return 0 on error.

BUGS

Between *re*(3) and *regexp*(3) there are too many routines.

NAME

regcomp, regexec, regsub, regerror – regular expression

SYNOPSIS

```
#include <regexp.h>

regexp *regcomp(exp)
char *exp;

int regexec(prog, string, match, msize)
regexp *prog;
char *string;
regsubexp *match;
int msize;

void regsub(source, dest, match, msize)
char *source, *dest;
regsubexp *match;
int msize;

void regerror(msg)
char *msg;
```

DESCRIPTION

Regcomp compiles a regular expression and returns a pointer to a compiled regular expression. The space is allocated by *malloc*(3) and may be released by *free*. Regular expressions are as in *re*(3) except that newlines are not operators and back-references (with *n*) are not supported.

Regexec matches a null-terminated *string* against the compiled regular expression in *prog*. If it matches, *regexec* returns a non-zero value and fills in the array *match* with character pointers to the substrings of *string* that correspond to the parenthesized subexpressions of *exp*: `match[`*i*`].sp` points to the beginning and `match[`*i*`].ep` points just beyond the end of the *i*th substring. (Subexpression *i* begins at the *i*th left parenthesis, counting from 1.) Pointers in `match[0]` pick out the substring that corresponds to the whole regular expression. Unused elements of *match* are filled with zeros. Matches involving `*`, `+`, and `?` are extended as far as possible. The number of array elements in *match* is given by *msize*. The structure of elements of *match* is:

```
typedef struct {
      char *sp;
      char *ep;
} regsubexp;
```

Regsub places in *dest* a substitution instance of *source* in the context of the last *regexec* performed using *match*. Each instance of `\`*n*, where *n* is a digit, is replaced by the string delimited by `match[`*n*`].sp` and `match[`*n*`].ep`. Each instance of `&` is replaced by the string delimited by `match[0].sp` and `match[0].ep`.

Regerror, called whenever an error is detected in *regcomp*, *regexec*, or *regsub*, writes the string *msg* on the standard error file and exits. *Regerror* can be replaced to perform special error processing.

SEE ALSO

gre(1), *re*(3), *expr*(1)

DIAGNOSTICS

Regcomp returns `(regexp *)0` for an illegal expression or other failure. *Regexec* returns 0 if *string* is not accepted.

NAME

scanf, fscanf, sscanf – formatted input

SYNOPSIS

```
#include <stdio.h>

scanf(format [ , pointer ] ... )
char *format;

fscanf(stream, format [ , pointer ] ... )
FILE *stream;
char *format;

sscanf(s, format [ , pointer ] ... )
char *s, *format;
```

DESCRIPTION

Scanf reads from the standard input stream *stdin*. *Fscanf* reads from the named input *stream*. *Sscanf* reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string *format*, described below, and a set of arguments, normally pointers, indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines, which match optional white space in the input.
2. An ordinary character (not `%`) which must match the next character of the input stream.
3. Conversion specifications, consisting of the character `%`, an optional assignment suppressing character `*`, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by `*`. Conversions other than `c` and `[` skip white space and consume non-white-space characters up to the next inappropriate character or until the field width, if specified, is exhausted. The field width is either an integer constant or `!`. In the latter case, the width is taken from an integer argument that precedes the next pointer argument.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

`%` A single `%` is expected in the input at this point; no assignment is done.

`d` A decimal integer is expected; the corresponding argument should be an integer pointer.

`o` an octal integer is expected; the corresponding argument should be an integer pointer.

`x` A hexadecimal integer is expected; the corresponding argument should be an integer pointer.

`s` A character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating `\0`, which will be added. The input field is terminated by a space character or a newline.

`c` A character is expected; the corresponding argument should be a character pointer. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

`e`
`f` A floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits possibly containing a decimal point, followed by an

optional exponent field consisting of an `E` or `e` followed by an optionally signed integer.

`[`

`[^` A character string is expected. The left bracket (or bracket and circumflex) is followed by a set of characters and a right bracket. When the set is introduced by `[` (or `[^`), the string consists only of characters in (or not in) the set. The corresponding argument must point to a character array.

The conversion characters `d`, `o` and `x` may be preceded by `l` to indicate that a pointer to `long` rather than to `int` is in the argument list. Similarly, the conversion characters `e` or `f` may be preceded by `l` to indicate a pointer to `double` rather than to `float`. The conversion characters `d`, `o` and `x` may be preceded by `h` to indicate a pointer to `short`.

The *scanf* functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant `EOF` is returned upon end of input; note that this is different from `0`, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```
int i; float x; char name[50];
scanf("%d%f%s", &i, &x, name);
```

with the input line

```
25   54.32E-1  thompson
```

will assign to *i* the value `25`, *x* the value `5.432`, and *name* will contain `thompson\0`. Or,

```
int i; float x; char name[50];
scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input

```
56789 0123 56a72
```

will assign `56` to *i*, `789.0` to *x*, skip `0123`, and place the string `56\0` in *name*. The next call to *getchar* will return `a`.

SEE ALSO

atof(3), *stdio*(3), *ungetc*(3)

DIAGNOSTICS

The *scanf* functions return `EOF` on end of input, and a short count for missing or illegal data items.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.
The input scan stops short of the end of excessively long numbers.
There is no `%#`.
When no maximum field width is given in a `%s` or `%[]` conversion specification, improper input can overrun the output string and corrupt the program in arbitrarily malicious ways. The best alternative, `%!s`, is nonstandard.
A deprecated usage allows upper-case conversion characters as equivalents for lower-case characters preceded by `l`.

NAME

setbuf – assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char buf[BUFSIZ];
```

DESCRIPTION

Setbuf is used after a stream has been opened but before it is read or written. It causes the character array *buf* to be used instead of an automatically allocated buffer. If *buf* is the constant pointer `NULL`, input/output will be completely unbuffered.

A buffer is normally obtained from *malloc*(3) upon the first *getc*(3) or *putc* on the file. Initially, the standard stream *stderr* is unbuffered, and the standard stream *stdout* is flushed automatically whenever new data is read by *getc*. The latter magic may be dissolved by a call to *setbuf*.

SEE ALSO

stdio(3), *malloc*(3)

NAME

setjmp, longjmp – non-local goto

SYNOPSIS

```
#include <setjmp.h>

setjmp(env)
jmp_buf env;

longjmp(env, val)
jmp_buf env;
```

DESCRIPTION

These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

Setjmp saves its stack environment in *env* for later use by *longjmp*. It returns value 0.

Longjmp restores the environment saved by the last call of *setjmp*. It then causes execution to continue as if the call of *setjmp* had just returned with value *val*. The invoker of *setjmp* must not itself have returned in the interim. All accessible data have values as of the time *longjmp* was called.

SEE ALSO

signal(2)

NAME

sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

SYNOPSIS

```
#include <math.h>

double sin(x)
double x;

double cos(x)
double x;

double tan(x)
double x;

double asin(x)
double x;

double acos(x)
double x;

double atan(x)
double x;

double atan2(y, x)
double x, y;
```

DESCRIPTION

Sin, *cos* and *tan* return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

Asin returns the arc sin in the range $-\pi/2$ to $\pi/2$.

Acos returns the arc cosine in the range 0 to π.

Atan returns the arc tangent of x in the range $-\pi/2$ to $\pi/2$.

Atan2 returns the arc tangent of y/x in the range $-\pi$ to π.

DIAGNOSTICS

Arguments of magnitude greater than 1 cause *asin* and *acos* to return value 0; *errno* is set to `EDOM`. The value of *tan* at its singular points is a huge number, and *errno* is set to `ERANGE`.

BUGS

The value of *tan* for arguments greater than about 2^{31} is garbage.

NAME

sinh, cosh, tanh – hyperbolic functions

SYNOPSIS

```
#include <math.h>

double sinh(x)
double x;

double cosh(x)
double x;

double tanh(x)
double x;
```

DESCRIPTION

These functions compute the designated hyperbolic functions for real arguments.

DIAGNOSTICS

Sinh and *cosh* return a huge value of appropriate sign when the correct value would overflow and set *errno* to EDOM.

NAME

sleep – suspend execution for an interval

SYNOPSIS

```
sleep(seconds)
unsigned seconds;
```

DESCRIPTION

The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be up to one second less than that requested, because scheduled wakeups occur at fixed second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an alarm clock signal and pausing until it occurs. The previous state of this signal is saved and restored. If the sleep time exceeds the time to the alarm signal, the process sleeps only until the signal would have occurred, and the signal is sent one second later.

SEE ALSO

alarm(2)

NAME

stdio – standard buffered input/output package

SYNOPSIS

```
#include <stdio.h>

FILE *stdin;
FILE *stdout;
FILE *stderr;
```

DESCRIPTION

The functions described in Sections 3S constitute an efficient user-level buffering scheme. The in-line macros *getc*(3) and *putc* handle characters quickly. The higher level routines *fgets, scanf, fscanf, fread, puts, fputs, printf, fprintf, fwrite* all use *getc* and *putc;* they can be freely intermixed.

A file with associated buffering is called a *stream,* and is declared to be a pointer to a defined type `FILE`. *Fopen*(3) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

`stdin`	standard input file
`stdout`	standard output file
`stderr`	standard error file

A constant pointer `NULL` designates no stream at all.

An integer constant `EOF` is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file `<stdio.h>` of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include file and need no further declaration. The constants, and the following 'functions' are implemented as macros: *getc, getchar, putc, putchar, feof, ferror, fileno*.

SEE ALSO

printf(3), *scanf*(3), *fopen*(3), *getc*(3), *fgets*(3), *fread*(3), *fseek*(3), *ungetc*(3), *popen*(3), *setbuf*(3), *ferror*(3)
open(2), *read*(2), *fio*(3)

DIAGNOSTICS

The value `EOF` is returned uniformly to indicate that a `FILE` pointer has not been initialized with *fopen,* input (output) has been attempted on an output (input) stream, or a `FILE` pointer designates corrupt or otherwise unintelligible `FILE` data.

BUGS

Buffering of output can prevent output data from being seen until long after it is computed − perhaps never, as when an abort occurs between buffer filling and flushing.
Buffering of input can cause a process to consume more input than it actually uses. This can cause trouble across *exec*(2) or *system*(3) calls.
BUffering may delay the receipt of a write error until a subsequent *stdio* writing, seeking, or file-closing call.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok, strdup – string operations

SYNOPSIS

```
#include <libc.h>

char *strcat(s1, s2)
char *s1, *s2;

char *strncat(s1, s2, n)
char *s1, *s2;
int n;

int strcmp(s1, s2)
char *s1, *s2;

int strncmp(s1, s2, n)
char *s1, *s2;
int n;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;
int n;

int strlen(s)
char *s;

char *strchr(s, c)
char *s;
int c;

char *strrchr(s, c)
char *s;
int c;

char *strpbrk(s1, s2)
char *s1, *s2;

int strspn(s1, s2)
char *s1, *s2;

int strcspn(s1, s2)
char *s1, *s2;

char *strtok(s1, s2)
char *s1, *s2;

char *strdup(s)
char *s;
```

DESCRIPTION

The arguments *s1*, *s2* and *s* point to null-terminated strings. The functions *strcat*, *strncat*, *strcpy*, and *strncpy* all alter *s1*. These functions do not check for overflow of the array pointed to by *s1*.

Strcat appends a copy of string *s2* to the end of string *s1*. *Strncat* appends at most *n* characters. Each returns a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer less than, equal to, or greater than 0, according as *s1* is lexicographically less than, equal to, or greater than *s2*. *Strncmp* makes the same comparison but looks at at most *n* characters.

Strcpy copies string *s2* to *s1*, stopping after the null character has been copied. *Strncpy* copies exactly *n* characters, truncating *s2* or adding null characters to *s1* if necessary. The result will not be null-terminated if the length of *s2* is *n* or more. Each function returns *s1*.

Strlen returns the number of characters in *s*, not including the terminating null character.

Strchr (*strrchr*) returns a pointer to the first (last) occurrence of character *c* in string *s*, or `(char *)0` if *c* does not occur in the string. The null character terminating a string is considered to be part of the string.

Strpbrk returns a pointer to the first occurrence in string *s1* of any character from string *s2*, `(char *)0` if no character from *s2* exists in *s1*.

Strspn (*strcspn*) returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

Strtok considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call, with pointer *s1* specified, returns a pointer to the first character of the first token, having replaced the character after the token by 0. Subsequent calls, signified by *s1* being `(char *)0`, will scan from where the preceding call left off. The separator string *s2* may be different from call to call. When no token remains in *s1*, `(char *)0` is returned.

Strdup returns a pointer to a distinct copy of the null-terminated string *s* in space obtained from *malloc*(3) or `(char *)0` if no space can be obtained.

SEE ALSO

memory(3)

BUGS

Strcmp and *strncmp* use native character comparison, which may be signed or unsigned.
The outcome of overlapping moves varies among implementations.

NAME

swab – swap bytes

SYNOPSIS

```
swab(from, to, nbytes)
char *from, *to;
```

DESCRIPTION

Swab copies *nbytes* bytes pointed to by *from* to the position pointed to by *to*, exchanging adjacent even and odd bytes. It is useful for carrying binary data between machines with different byte orders. *Nbytes* should be even.

NAME

system – issue a shell command

SYNOPSIS

```
system(string)
char *string;
```

DESCRIPTION

System causes the command `sh -c` *string* to be executed. The current process waits until the shell has completed, then returns the exit status of the shell.

SEE ALSO

popen(3), *exec*(2), *fork*(2)

DIAGNOSTICS

Exit status 127 indicates the shell couldn't be executed.

NAME

tcp_sock, tcp_connect, tcp_listen, tcp_accept, tcp_rcmd – tcp networking functions

SYNOPSIS

```
#include <sys/inet/tcp_user.h>

int tcp_sock();

int tcp_connect(fd, tp)
int fd;
struct tcpuser *tp;

int tcp_listen(fd, tp)
int fd;
struct tcpuser *tp;

int tcp_accept(fd, tp)
int fd;
struct tcpuser *tp;

int tcp_rcmd(host, port, locuser, remuser, cmd, fd2p)
char *host, *port, *locuser, *remuser, *cmd;
int *fd2p;
```

DESCRIPTION

These routines are loaded by the `-lin` option of *ld*(1).

TCP is a protocol layered upon IP (internet protocol). It provides full-duplex byte stream connections between end points called sockets. The address of a socket is composed of the internet address of its host and the port number to which the socket is bound.

Tcp_sock returns the file descriptor of an unbound socket. Once opened, a socket may be bound to a port number within the host and set up as the active or passive end of a connection.

Addresses and parameters are passed in `tcpuser` structures:

```
struct tcpuser {
       int code;
       tcp_port lport, fport;
       in_addr laddr, faddr;
       int param;
};
```

Lport and *laddr* refer to the port and address numbers of the local end of a connection. *Fport* and *faddr* refer to the port and address numbers of the foreign end of a connection.

Tcp_connect binds socket *fd* to port *tp*`->lport` and attempts to set up a connection to the socket bound to port *tp*`->fport` on host *tp*`->faddr`. If *tp*`->lport` is 0, a local port number is automatically chosen. *Tcp_connect* returns 0 if the connection is established, −1 otherwise. *Tp*`->lport` and *tp*`->laddr` are filled in to reflect the local port and address numbers for the connection. Communication proceeds by performing *read*(2) and *write* on *fd*. If *tp*`->param` is non-zero, it specifies options to set for the connection. The only option supported is `SO_KEEPALIVE` which causes empty messages to be sent periodically to detect dead connections.

Tcp_listen binds socket *fd* to port *tp*`->lport` and configures the socket to listen for connection requests to that port. If *tp*`->faddr` and *tp*`->fport` are non-zero, only connections coming from sockets on machine *faddr* and bound to port *fport* are listened for. *Tcp_listen* returns 0 on success, −1 otherwise. *tp*`->laddr` is filled in to reflect the local address number for the connection. *Select*(2) can be used with a listening socket to provide asynchronous polling of connection requests by selecting for pending input on the socket.

Tcp_accept waits for and accepts a connection request sent to the listening socket *fd*. When a connection arrives, *tcp_accept* returns a new file descriptor over which communications can proceed. *Tp*`->faddr`, *tp*`->fport`, *tp*`->laddr`, and *tp*`->lport` are filled in to identify the two ends of the connection. *Tp*`->param` is filled in with the minor device number of the tcp device used for the new connection. *Fd* is left open and continues listening for connections.

Tcp_rcmd remotely executes a *cmd* on *host* as user *remuser*. Standard input is attached to *cmd's* standard input and *cmd's* standard output is attached to standard output. If *fd2p* is non-zero, it is filled with the file descriptor of a new TCP connection attached to *cmd's* standard error. Otherwise, *cmd's* standard error is attached to its standard output.

FILES

`/dev/tcp*` the socket devices

SEE ALSO

ipc(3), *internet*(3), *udp*(3)

DIAGNOSTICS

Tcp_sock returns −1 if no sockets are available.

NAME

tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs – device-independent terminal screen control

SYNOPSIS

```
char PC;
char *BC;
char *UP;
short ospeed;

tgetent(bp, name)
char bp[1024], *name;

tgetnum(id)
char *id;

tgetflag(id)
char *id;

char *
tgetstr(id, area)
char *id, **area;

char *
tgoto(cm, destcol, destline)
char *cm;

tputs(cp, affcnt, outc)
char *cp;
int (*outc)();
```

DESCRIPTION

These functions are loaded by option `-ltermcap` of *ld*(1). They extract and use capabilities from the terminal capability data base *termcap*(5). These are low level routines; see *curses*(3) for a higher level package.

Tgetent extracts the entry for terminal *name* into the buffer at *bp*. *Bp* should be a character buffer of size 1024 and must be retained through all subsequent calls to *tgetnum*, *tgetflag*, and *tgetstr*. *Tgetent* returns −1 if it cannot open the *termcap* file, 0 if the terminal name given does not have an entry, and 1 if all goes well. It will look in the environment for a `TERMCAP` variable. If found, and the value does not begin with a slash, and the terminal type *name* is the same as the value of the environment variable `TERM`, the `TERMCAP` string is used instead of reading the termcap file. If it does begin with a slash, the string is used as a path name rather than `/etc/termcap`.

Tgetnum gets the numeric value of capability *id*, returning −1 if is not given for the terminal. *Tgetflag* returns 1 if the specified capability is present in the terminal's entry, 0 if it is not. *Tgetstr* gets the string value of capability *id*, placing it in the buffer at ***area*, advancing the *area* pointer. It decodes the abbreviations for this field described in *termcap*(5), except for cursor addressing and padding information.

Tgoto returns a cursor addressing string decoded from `cm` to go to column *destcol* in line *destline*. It uses the external variables `UP` (from the `up` capability) and `BC` (if `bc` is given rather than `bs`) if necessary to avoid placing `en`, `^D`, or `^@` in the returned string. (Programs which call *tgoto* should be sure to turn off the `XTABS` bit(s), since *tgoto* may now output a tab. Note that programs using termcap should in general turn off `XTABS` anyway since some terminals use `^I` for other functions, such as nondestructive space.) If a `%` sequence is given which is not understood, then *tgoto* returns `'OOPS'`.

Tputs decodes the leading padding information of the string *cp; affcnt* gives the number of lines affected by the operation, or 1 if this is not applicable; *outc* is a routine which is called with each character in turn. The external variable *ospeed* should contain the output speed of the terminal as in *tty*(4). The external variable `PC` should contain a pad character to be used (from the `pc` capability) if a null (`^@`) is inappropriate.

FILES

`/etc/termcap`

SEE ALSO

vi(1), *curses*(3), *termcap*(5)

NAME

timec, timegm, timelocal − convert ASCII to time

SYNOPSIS

```
#include <time.h>

long timec(string)
char *string;

long timegm(timep)
struct tm *timep;

long timelocal(timep, zone)
struct tm *timep;
char *zone;
```

DESCRIPTION

These routines are inverse to *ctime*(3) and its relatives. See *ctime*(3) for data layouts.

Timec converts to system format a date *string* as produced by *ctime*, *date*(1), or *ls*(1). An optional day of the week is ignored. A month name and day are required. A missing hour:min[:sec] field is taken to be `00:00:00`. An optional time zone (local time by default) may appear before or after the year. A missing year is assumed to be the past 12-month interval.

Timegm returns the system-format time corresponding to the broken-down GMT time pointed to by *timep*. In a copy of the broken-down time `tm_mon` is reduced mod 12 by carrying (positively or negatively) to `tm_year`. Next `tm_mon` and `tm_mday` are added to `tm_yday` appropriately for `tm_year`. Then `tm_sec`, `tm_min`, `tm_hour`, `tm_yday`, and `tm_year` are adjusted by carrying. Finally the system-format date is calculated from these 5 fields.

Timelocal is like *timegm*, except that the broken-down time belongs to the specified time *zone*, or is local time if *zone* is zero.

Time zones and months are recognized by the first three characters, regardless of case. *Strings* for *ctime* may contain names longer than three characters and may contain extra white space and commas.

EXAMPLES

Set a date ahead one month:

```
struct tm brk_out = *localtime(&date);
brk_out.tm_yday = 0;
brk_out.tm_mon++;
date = timelocal(&brk_out,0);
```

Convert a *date*(1) string to system format:

```
date = timec("Sat Sep 27 20:59:11 EDT 1986");
```

SEE ALSO

ctime(3), *time*(2)

BUGS

Unknown time zone names are taken to be GMT.
Times before the epoch yield nonsense.

NAME

tolower, toupper – force upper or lower case

SYNOPSIS

```
tolower(character)

toupper(character)
```

DESCRIPTION

If *character* is an upper case letter, *tolower* returns the same lower case letter, otherwise it returns the original character.

Toupper does the reverse.

SEE ALSO

ctype(3)

NAME

ttyname, isatty, nametty – find or set name of a terminal

SYNOPSIS

```
char *ttyname(fildes)

isatty(fildes)

nametty(fildes, file)
char *file;
```

DESCRIPTION

Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor *fildes*.

Isatty returns 1 if *fildes* is associated with a terminal device, 0 otherwise.

Nametty arranges that future opens of *file* will refer to the stream opened on *fildes*. *File* must exist before *nametty* is called. The arrangement is terminated when the other end of the stream is closed or hung up.

FILES

`/lib/ttydevs` list of tty directories for *ttyname*

SEE ALSO

fmount(2), *ioctl*(2)

DIAGNOSTICS

Ttyname returns `NULL` if *fildes* does not describe an entry in any of the directories listed in `/lib/ttydevs`.

Nametty returns 1 for success, 0 for failure (*file* does not exist, *fildes* is not a stream).

BUGS

The return value of *ttyname* points to static data whose content is overwritten by each call.

If *fildes* is a network connection, *isatty* may produce answers having more to do with the network than to the file to which network data is copied. In particular, it always returns no for connections set up by *rx*, and always returns yes for connections arranged by *dcon*; see *con*(1).

NAME

udp_connect, udp_listen, udp_datagram – udp networking functions

SYNOPSIS

```
#include <sys/inet/udp_user.h>

int udp_connect(sport, dhost, dport)
in_addr dhost;
udp_port sport, dport;

int udp_listen(sport, reply)
udp_port sport;
struct udpreply *reply;

int udp_datagram(sport)
udp_port sport;
```

DESCRIPTION

These routines are loaded by the `-lin` option of *ld*(1).

UDP (universal datagram protocol) is a protocol layered upon IP (internet protocol). It provides datagram service between end points called sockets. A socket address is composed of the internet address of its host and the port number to which the socket is bound.

Udp_connect returns the file descriptor of a UDP socket bound to port *sport.* Each *read*(2) from this file descriptor will only accept datagrams from the UDP socket at host *dhost*, port *dport*; a *write* on this file descriptor will be sent to that socket.

Udp_listen returns the file descriptor of a UDP socket bound to port *sport* and waits for a datagram to be sent to that port. Once a message has been received from another socket, all writes will go to that socket and reads will only accept data from that socket.

Udp_datagram returns the file descriptor of a UDP socket bound to port *sport.* Messages written to the file descriptor must start with a `struct udpaddr` which contains the destination of the message.

```
struct udpaddr {
        in_addr host;
        int     port;
};
```

Messages read from the file descriptor also start with a `struct udpaddr` and contain the address of the source socket.

FILES

`/dev/udp*` the socket devices

SEE ALSO

internet(3), *tcp*(3)

DIAGNOSTICS

All these routines returns −1 on failure.

NAME

uname − identify machine and kernel

SYNOPSIS

```
#include <utsname.h>

int uname(name)
struct utsname *name;
```

DESCRIPTION

Uname stores information identifying the current UNIX system in the structure pointed to by *name.*

```
struct utsname {
        char sysname[32];
        char nodename[32];
        char release[32];
        char version[32];
};
```

Uname returns a null-terminated character string naming the current UNIX system in the character array *sysname.* Similarly, *nodename* contains the name that the system is known by on some communications network. *Release* and *version* further identify the operating system.

FILES

`/etc/whoami`

BUGS

Since a machine can have different names on different networks, *nodename* is pretty useless.

This call is a partial simulation of one appearing in other systems. *Sysname* and *nodename* are copied from `/etc/whoami`; the other fields are meaningless.

NAME

ungetc – push character back into input stream

SYNOPSIS

```
#include <stdio.h>

ungetc(c, stream)
FILE *stream;
```

DESCRIPTION

Ungetc pushes the character *c* back on an input stream. That character will be returned by the next *getc* call on that stream. *Ungetc* returns *c*.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push `EOF` are rejected.

SEE ALSO

getc(3), *stdio*(3), *fseek*(3)

DIAGNOSTICS

Ungetc returns `EOF` if it can't push a character back.

NAME

varargs – variable argument list

SYNOPSIS

```
#include <varargs.h>
function(va_alist)
va_dcl

va_list pvar;

va_start(pvar);

va_arg(pvar, type);

va_end(pvar);
```

DESCRIPTION

This set of macros allows portable procedures that accept variable argument lists to be written. Routines which have variable argument lists (such as *printf*(3)) that do not use varargs are inherently nonportable, since different machines use different argument passing conventions.

The literal identifier *va_alist* is used in a function header to declare a variable argument list. It is declared by *va_dcl*. Note that there is no semicolon after *va_dcl*.

`Va_list` is the type of the variable *pvar*, which is used to traverse the list. One variable of this type must always be declared.

Va_start initializes *pvar* to the beginning of the list.

Va_arg returns the next argument in the list pointed to by *pvar*. *Type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type is expected, since it cannot be determined at runtime.

Va_end is used to finish up.

Multiple traversals, each bracketed by *va_start* and *va_end*, are possible.

EXAMPLES

How to define *execl* in terms of *execv*; see *exec*(2):

```
#include <varargs.h>
execl(va_alist)
va_dcl
{
      va_list ap;
      char *file;
      char *args[100];
      int argno = 0;
      va_start(ap);
      file = va_arg(ap, char*);
      while(args[argno++] = va_arg(ap, char*));
      va_end(ap);
      execv(file, args);
}
```

NAME

view2d, moviefil – movie of a function f(x, y, t)

SYNOPSIS

```
view2d(fd, nx, ny, time, u, v, fixuv, pmin, pmax, p)
short p[];
double time;

moviefil(fd, nx, ny, time, outside, f)
float time, outside, f[];
```

DESCRIPTION

View2d writes a frame in the format *view2d*(5) onto the file specified by file descriptor *fd*. To load these routines, use the *ld*(1) option `-lview2d`. *Nx*, *ny* give the grid size. *Time* is a (nondecreasing) frame index, typically set to simulation time or iteration count. *U* and *v* describe the relation between pixel values and user function values:

$$p = u + f \times 2^{-v}.$$

U and *v* may vary from one frame to the next. When the global scaling is known beforehand, put *fixuv* = 1 and set *pmin* and *pmax* to the limits of the data. (Otherwise put *fixuv*=0; *pmin* and *pmax* will be ignored.)

P is the *nx* by *ny* array of pixel values, with the *x* index running fastest. There is a threshold for describing nonrectangular regions: any pixel value less than or equal to `-32766` is treated as an out-of-bounds marker and will appear as black. Other pixel values should lie in the range `-32765..32765` inclusive.

Moviefil is an alternate version that is somewhat less flexible but easier to use. It takes floats and scales automatically to shorts. An element of *f* less than *outside* is treated as undefined and will appear as black.

SEE ALSO

view2d(1), *view2d*(5)

BUGS

The array arguments are 2-D Fortran arrays.

4 SPECIAL FILES

4 SPECIAL FILES

NAME

intro − introduction to devices, line disciplines, and file systems

DESCRIPTION

This section describes drivers for devices, stream line disciplines, and file systems.

Devices are accessed through special files of type `S_IFBLK` (block devices) or `S_IFCHR` (character devices); see *stat*(2). Block devices use a block buffering scheme within the system, so that sectored devices like disks may be accessed a byte at a time. Character devices don't use the block buffers. Only block devices may be mounted as disk file systems. Most block devices have associated 'raw' character devices that bypass all buffering for fast direct I/O.

The device associated with a special file is identified by a pair of numbers: a major device number naming the driver, and a minor device number picking some particular device or subunit. Major numbers are listed in *mknod*(8). Minor numbers are specific to each driver; see the writeups in this section. Minor numbers are stored in a single unsigned byte; they are chosen from the range 0-255.

Some character devices are also stream devices. These use a different internal buffering mechanism to allow data to flow asynchronously. Various special operations are possible on streams; see *stream*(4).

Line disciplines are processing modules that may be inserted into streams. They are identified by integers passed to the calls that insert and remove them. The C library contains global variables initialized to the numbers for various line disciplines; *stream*(4) has a list.

There are several different types of file system: conventional disk volumes, remote file systems accessed by the system sending messages though a stream to a server as described in *netfs*(8), a file system containing a file representing each process in the system, and so on. All of these appear the same to ordinary processes, except that not all file systems implement all operations; for example, a process file has a name, and may be opened, read, and written like an ordinary file, but may not be renamed because *proc*(4) doesn't allow that.

File system types are identified by integers, used by and listed in *fmount*(2). They are just magic numbers at present.

SEE ALSO

fmount(2), *stream*(4), *mknod*(8)

NAME

buf_ld − buffering line discipline

DESCRIPTION

Buf_ld treasures up data for a while, then emits it in a burst. It is otherwise transparent. It is meant to reduce overhead of programs such as *cu*(1) and *uucp*(1) that read input from moderate-speed lines in raw or cbreak mode.

It saves characters until 16 have arrived, or until 1/20 sec has passed and no more characters have come.

SEE ALSO

stream(4), *ttyld*(4)

NAME

conn_ld − line discipline for unique stream connection

SYNOPSIS

```
#include <sys/filio.h>
```

DESCRIPTION

This line discipline provides unique connections to a server. The server process should push the line discipline on a pipe (see `FIOPUSHLD` in *stream*(4)) and *fmount*(2) the pipe end on a file. A subsequent attempt to *open*(2) or *creat* that file causes a new pipe to be created. A file descriptor for one end of the new pipe is passed on the mounted pipe to to the server process as if by `FIOSNDFD`; see *stream*(4). The opening process is blocked until the server responds. The server should receive the passed file descriptor with FIORCVFD and respond in one of the following ways:

— Accept the new file descriptor by performing

```
ioctl(fd, FIOACCEPT, (void *)0);
```

The originating *open* completes and returns a file descriptor for the other end of the new pipe.

— Write some data on the new file descriptor. This performs an implicit `FIOACCEPT`.

— Pass a different file descriptor:

```
ioctl(fd, FIOSNDFD, &newfd);
```

The originator's end of the new pipe is closed, and a file descriptor for the open file designated by *newfd* is returned to the originating *open*.

— Reject the connection, by closing the new file descriptor or by performing

```
ioctl(fd, FIOREJECT, (void *)0);
```

The originating *open* fails with `ENXIO` and the new pipe is discarded.

SEE ALSO

fmount(2), *stream*(4)

NAME

console − VAX console interface

DESCRIPTION

The console terminal is either in program mode (connected to the program running in the VAX) or in console mode (connected to the console interpreter, which prompts with >>>). On most VAXes, control-`p` switches to console mode. If the VAX CPU is still running, `SET TERM PROG` returns to program mode. If the CPU is halted, `C` restarts it and connects to the VAX. Hitting the break key in either mode may halt the console processor and produce an `@` prompt; hit `P` to escape.

On an 11/750, switching to console mode always halts the VAX; only `C` will escape. Control-`d` while in console mode may induce micro-debugging mode, where the prompt is `RDM>`. Type `RET` to return to console mode.

On a MicroVAX, the break key (only) halts the VAX and switches to console mode.

All these modes are implemented by the VAX hardware. To the operating system, the console looks like an ordinary terminal as described in *tty*(4), except that the speed is fixed in the hardware.

FILES

`/dev/console`

SEE ALSO

tty(4), *ttyld*(4), *reboot*(8)

NAME

dk, dkp_ld, unixp_ld, cmc_ld − Datakit interface and protocols

SYNOPSIS

```
#include <sys/dkio.h>
```

DESCRIPTION

These device drivers and line disciplines are used to connect to a Datakit network. Normally the programs in *dkmgr*(8) do all the work.

Several combinations of hardware and software may be used to connect a system to Datakit:

The *dk* driver works with a DR11-C or DRV11-J connected through an adapter box to a Datakit CPM-422. The host computer does all the protocol work.

The *kdi* driver works with a KMC11-B and one of several line units (KDI, DUBHI, KMS11-P) connected to one of several Datakit or ISN interface boards. The KMC11 runs microcode that handles the URP protocol.

The experimental *kmcdk* driver works with a KMC11-B and a line unit, as above, but the KMC11 runs different microcode implementing a simple DMA engine, and the host does all the protocol work. This is slower, but rather more robust, than the *kdi* setup.

The experimental *cure* driver works with a custom-built microprocessor board connected to a Datakit, ISN, or Hyperkit fiber interface. The host does all the protocol work.

Each minor device number represents a Datakit channel; the device number is the channel number. The *kdi* driver allows only 96 channels per KMC11-line unit pair; devices 96-191 are channels 0-95 on a second pair, if present, and devices 192-255 are channels 0-63 on a third. For the other drivers, there may be only one hardware interface, which may have up to 256 channels.

Usually there is one interface, with files in directory `/dev/dk`. See *dkmgr*(8) for more about naming conventions.

Dkp_ld is a stream line discipline implementing the URP protocol. The *kdi* driver makes its own URP arrangements; other interfaces need the line discipline. A separate copy of *dkp_ld* must be pushed on each active channel.

Cmc_ld and *unixp_ld* are line disciplines set up calls handle and controller handshake messages. *Cmc_ld* runs a Research-only call setup protocol; *unixp_ld* runs the standard one. One copy of the appropriate line discipline must be pushed on the common signaling channel to deal with occasional controller keep-alive and maintenance messages. Other copies of the line discipline come and go as calls are placed.

These *ioctl* calls are provided by the device drivers:

`DIOCNXCL` Allow this channel to be opened many times. By default, if a channel is open, it may not be opened again. The default is restored whenever the channel is completely closed.

`KIOCSHUT` Reset the *kdi* driver, hanging up all channels.

These *ioctl* calls are provided by the URP processors, *dkp_ld* and *kdi*:

`DIOCSTREAM` Don't generate a stream delimiter when this channel receives a BOT trailer.

`DIOCRECORD` Insert a stream delimiter after receiving BOT; the default.

`DIOCSCTL` The third argument points to a byte; send that as a Datakit control envelope.

`DIOCRCTL` The third argument points to a byte; copy the most recently received non-URP control envelope there. Zero means no control has been received since the last call.

`DIOCXWIN` Set transmit window size. The third argument points to an array of two long integers. The first number is the maximum size of each URP block; the second is the number of blocks that may be outstanding. Blocks may be no more than 4096 bytes, and the protocol allows no more than eight blocks in a window. A `KIOCINIT` call should follow immediately, or things may go awry.

`KIOCISURP` Return success if some URP processor is active on this channel.
`KIOCINIT` Initialize URP.

These *ioctl* calls are provided by the call setup line disciplines:

`DIOCLHN` This is the common signalling channel.
`DIOCHUP` Tell the controller to initialize, hanging up all channels.
`DIOCSTOP` Temporarily hold back received data, so it won't be lost in call setup messages.
`DIOCSTART` Release data held by `DIOCSTOP`.
`DIOCCHAN` The third argument points to an integer; fill it in with the number of a free channel for calling out. This is a hint, not a promise; the channel may already be taken by the time it is opened. The caller should be prepared to try again.

FILES

`/dev/dk`

SEE ALSO

dkmgr(8), *kmc*(8)
A. G. Fraser and W. T. Marshall, 'Data Transport in a Byte Stream Network', *IEEE J-SAC*, (September, 1989)
Datakit VSC Internal Interface Specification, select code 700-283, AT&T Customer Information Center, Indianapolis

BUGS

Dkp_ld and *kdi* insist on using exactly three blocks in a window, no matter what they are told in `DIOCXWIN`.
The *kdi* driver has only two block sizes, 28 and 252 bytes.

NAME

drum − paging device

DESCRIPTION

This file is the paging and swapping device. It usually refers to an indirect driver that allows swapping to be spread over several disk drives.

FILES

`/dev/drum`

BUGS

The indirect driver divides the swap area into interleaved sections of half a megabyte or so; reads may not span the boundary between sections. Since the system doesn't allocate blocks across sections, this probably doesn't matter.

NAME

ethernet – Ethernet interface

SYNOPSIS

```
#include <sys/enio.h>
#include <sys/ethernet.h>
```

DESCRIPTION

There are drivers for several hardware interfaces to Ethernet. All have the same programming interface.

There are eight software channels for each hardware device. A channel sends and receives packets for a single Ethernet interface; hence eight protocols may be used independently on the same device. If a channel is open, it may not be opened again.

Read and *write* deal in Ethernet packets, consisting of a header followed by no less than 46 but no more than 1500 bytes of data. The header, defined in `<sys/ethernet.h>`, is as follows:

```
#define     ETHERALEN   6     /* bytes in an ethernet address */

struct etherpup {
      unsigned char dhost[ETHERALEN];     /* destination address */
      unsigned char shost[ETHERALEN];     /* source address */
      unsigned short type;    /* protocol type */
};
```

The protocol type is in the network's byte order, most significant byte first.

Read on a channel returns at most one complete packet. If only part of a packet fits in the *read* buffer, successive reads return the remainder. *Write* should be given a single complete packet; `dhost` and `type` must be filled in. The system supplies `shost`.

There are a few *ioctl* calls, defined in `<sys/enio.h>`:

`ENIOTYPE` The third argument points to a short integer; use that as the protocol type for this channel.

`ENIOADDR` The third argument points to a six-character buffer; copy the hardware address of this interface there.

Minor device numbers 0-7 are the eight channels of the first hardware device of a given type; 8-15 are the second device, and so on. File names usually end in two digits, like `/dev/il13` for the fourth channel of the second Interlan device.

FILES

`/dev/il??`	Interlan NI1010A devices
`/dev/qe??`	DEQNA devices
`/dev/bna??`	DEBNA devices

SEE ALSO

internet(3), *ipconfig*(8)

BUGS

The DEQNA driver fills in the protocol type field in transmitted packets; other drivers don't.

NAME

fd, stdin, stdout, stderr, tty − file descriptor files

DESCRIPTION

These files, conventionally called `/dev/fd/0`, `/dev/fd/1`, ... `/dev/fd/127`, refer to files accessible through file descriptors. If file descriptor *n* is open, these two system calls have the same effect:

```
fd = open("/dev/fd/n", mode);
fd = dup(n);
```

On these devices *creat* (see *open*(2)) is equivalent to *open,* and *mode* is ignored. As with *dup*, subsequent reads or writes on *fd* fail unless the original file descriptor allows the operations.

FILES

`/dev/fd/*`	
`/dev/stdin`	linked to `/dev/fd/0`
`/dev/stdout`	linked to `/dev/fd/1`
`/dev/stderr`	linked to `/dev/fd/2`
`/dev/tty`	linked to `/dev/fd/3`

SEE ALSO

open(2), *dup*(2)

DIAGNOSTICS

Open returns −1 and `EBADF` if the related file descriptor is not open and in the appropriate mode (reading or writing).

NAME

ip, ip_ld − DARPA internet protocol

SYNOPSIS

```
#include <sys/inio.h>
#include <sys/inet/in.h>
#include <sys/inet/ip_var.h>
```

DESCRIPTION

The *ip_ld* line discipline and the `/dev/ip*` files together implement the DARPA IP datagram protocol. They are used by the programs described in *ipconfig*(8) and *route*(8).

Each Ethernet device, Datakit channel, or other stream that is to send and receive IP packets must be registered as an 'IP interface' by pushing *ip_ld* and setting local and remote addresses with *ioctl* calls. Thereafter, data received from the network are assumed to be IP packets, and are intercepted by the line discipline. Packets destined for the local address of an active IP interface are routed for reading on one of the *ip* device files. Other packets are routed to the IP interface with a matching remote address and retransmitted.

Data written on *ip* devices are taken to be IP packets, are handed to the IP interface with a matching address, and are sent. Packets destined for unreachable places are quietly dropped.

A packet consists of a single stream record, followed by a delimiter: at most one packet is returned by a *read* call; an entire packet must be presented in a single *write*. A packet includes the IP header. Numbers in the header are in host byte order.

Different *ip* devices handle different protocols atop IP. The minor device is the protocol number in the IP header; e.g. 6 for TCP or 17 for UDP. While an *ip* device is open, it may not be opened again. IP packets are often processed by pushing a line discipline such as *tcp_ld* on an *ip* device, rather than by explicit *read* and *write* calls; see *tcp*(4).

The following *ioctl*(2) calls, defined in `<sys/inio.h>`, apply to an IP interface. `IPIOLOCAL` and either `IPIOHOST` or `IPIONET` must be called on each interface before packets will be routed correctly. Type `in_addr`, defined in `<sys/inet/in.h>`, is a 32-bit integer representing an IP address in host byte order.

`IPIOLOCAL` The third argument points to an `in_addr`: the local IP address for this interface.

`IPIOHOST` The third argument points to an `in_addr`: the remote IP address of the single host reachable through this interface.

`IPIONET` The third argument points to an `in_addr`: the remote IP address of the network of many hosts reachable through this interface. IP addresses are matched to the network address by applying an internal bit-mask: any IP address for which (*address*&*mask*)==*net-address* is part of the network. The default mask depends on the IP address class; see the IP protocol standard for details.

`IPIOMASK` The third argument points to an `in_addr` containing a new network mask for this interface.

`IPIOMTU` The third argument points to an integer number of bytes. IP packets larger than this size (1500 by default) will be split into smaller ones before being sent through this interface.

`IPIOARP` The network device for this interface is an Ethernet. Discard the Ethernet header from each incoming packet. When sending a packet, prefix an Ethernet header containing protocol type `0x8` and a destination address obtained by looking up the IP destination address in a table. If the IP address is not in the table, discard the packet, and make an `in_addr` containing the offending address available for reading on this file descriptor (the one on which *ip_ld* was pushed).

`IPIORESOLVE` The third argument points to a structure:

```
struct {
        in_addr inaddr;
        unsigned char enaddr[6];
};
```

Add an entry to the table consulted after `IPIOARP`, mapping IP address `inaddr` to Ethernet address `enaddr`.

The following *ioctl* calls, define in `<sys/inio.h>`, apply to the entire IP subsystem; they may be used on any file with *ip_ld* pushed.

`IPIOROUTE` The third argument points to a structure:

```
struct route {
        in_addr dst;
        in_addr gate;
};
```

Arrange that henceforth, any IP packet destined for address `dst` will be routed as if destined for `gate`.

`IPIOGETIFS` The third argument points to a union as follows. The structure is defined in `<sys/inet/ip_var.h>`.

```
union {
        int index;
        struct ipif {
                struct queue *queue;
                int flags;
                int mtu;
                in_addr thishost;
                in_addr that;
                in_addr mask;
                in_addr broadcast;
                int ipackets, ierrors;
                int opackets, oerrors;
                int arp;
                int dev;
        } ipif;
};
```

Before the call, `index` should contain an integer naming an entry in the system's table of active interfaces. Interfaces are numbered in a continuous sequence starting at 0. Out-of-range numbers return an error. After the call, `ipif` is filled in with various numbers about that interface.

FILES

`/dev/ip*`

SEE ALSO

ioctl(2), *internet*(3), *ipconfig*(8), *route*(8)
DARPA standards RFC 791, RFC 1122

BUGS

The ARP mechanism should be generalized to deal with networks other than Ethernet. There is only one ARP table for the entire system; there should be one for each interface.
The structures used by `IPIOROUTE` and `IPIORESOLVE` should appear in a header file somewhere.

NAME

mem, kmem, kUmem, mtpr, fineclock − memory and VAX processor registers

DESCRIPTION

Mem is a file that is an image of the main memory of the computer. It may be used to examine (and even to patch) the system. Byte addresses in *mem* are interpreted as physical memory addresses. References to non-existent locations return errors.

Kmem and *kUmem* are like *mem*, but access kernel-mode virtual memory. *KUmem* promises that reads and writes will be done in two-byte quantities; this is convenient for UNIBUS accesses.

Mtpr accesses VAX internal processor registers. Each register is 4 bytes long; register *n* may be read or written at address *n*∗4.

Fineclock reads a high-resolution clock. Reading four bytes returns a 32-bit unsigned integer representing the number of microseconds since the epoch 00:00:00 GMT, Jan. 1, 1970, with high-order bits discarded.

Mem, *kmem*, *kUmem*, and *mtpr* have minor device numbers 0, 1, 3, and 5. *Fineclock* is a separate driver; the minor device number is ignored.

FILES

`/dev/mem`
`/dev/kmem`
`/dev/kUmem`
`/dev/mtpr`
`/dev/fineclock`

SEE ALSO

time(2)
VAX Hardware Handbook

BUGS

Examining and patching device and processor registers may give unexpected results when read-only or write-only bits are present.
An attempt to read a nonexistent processor register returns 0 instead of an error.
The *fineclock* counter overflows every hour or so. It is only as precise as the hardware; hence it is inaccurate on a MicroVAX.

NAME

mesg_ld, rmesg_ld − message line discipline modules

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stream.h>
```

DESCRIPTION

Mesg_ld and *rmesg_ld* turn internal stream messages into ordinary data and vice versa. They supply a way to splice a stream connection together through a process or across a network connection. The messages represent ordinary data and various control operations.

After *mesg_ld* has been pushed on a stream, *read*(2) on the stream returns encoded messages; *write* is expected to supply messages in the same coding. An encoded message consists of a header followed by zero or more bytes of associated data. The header, as defined in `<sys/stream.h>`, is of the form

```
struct mesg {
      char  type;
      unsigned char     magic;
      unsigned char     losize, hisize;
};
#define     MSGMAGIC    0345
#define     MSGHLEN     4     /* true length of struct mesg in bytes */
```

The header is `MSGHLEN` bytes long; beware that this is not always the same as `sizeof(struct mesg)`. The `magic` field contains the constant `MSGMAGIC`, to help prevent interpreting bad data as a message header. There are `losize+(hisize<<8)` bytes of associated data.

Messages may be written in pieces, or several messages may be written at once. At most one message will be read at a time. If an impossible message is written, the stream may be shut down.

Rmesg_ld is exactly the opposite of *mesg_ld*. It is intended for use with devices that generate data containing encoded messages. Here is a list of message types, defined in `<sys/stream.h>`:

`M_DATA` (0) Ordinary data.

`M_BREAK` (01) A line break on an RS232-style asynchronous connection. No associated data.

`M_HANGUP` (02) When received, indicates that the other side has gone away. Thereafter the stream is useless. No associated data.

`M_DELIM` (03) A delimiter that introduces a record boundary in the data. No associated data.

`M_IOCTL` (06) An *ioctl*(2) request. The associated data is a four-byte integer containing the function code, least significant byte first, followed by some amount of associated data. An `M_IOCACK` or `M_IOCNAK` reply is expected.

`M_DELAY` (07) A real-time delay. One byte of data, giving the number of clock ticks of delay time.

`M_CTL` (010) Device-specific control message.

`M_SIGNAL` (0101) Generate signal number given in the one-byte message.

`M_FLUSH` (0102) Flush input and output queue if possible.

`M_STOP` (0103) Stop transmission immediately.

`M_START` (0104) Restart transmission after `M_STOP`.

`M_IOCACK` (0105) Successful reply to `M_IOCTL`. Associated data is to be written back to the caller.

`M_IOCNAK` (0106) Failed reply to `M_IOCTL`. A single-byte message, if present, will be returned in by the failing *ioctl*.

`M_PRICTL` (0107) High-priority device-specific control message.

SEE ALSO

stream(4)

BUGS

The format of arguments to `M_IOCTL` is machine dependent.
The amount of associated data is limited, but large (>4K).

NAME

mt − magtape interface

SYNOPSIS

```
#include <sys/types.h>
#include <sys/mtio.h>
```

DESCRIPTION

The *rmt* files refer to magnetic tape drives. Filenames beginning with `rmt` are rewound when closed; those beginning with `nrmt` are not. When a file open for writing is closed, two file marks are written. If the tape is not to be rewound, it is positioned with the head between the two file marks.

Conventionally `rmt0` is 800, `rmt1` is 1600, and `rmt2` is 6250 bpi.

Each *read*(2) or *write* call reads or writes the next record on the tape. *Read* returns at most a single record; the return value is the record size. If the next record is larger than the read buffer, an error is returned. A file mark causes *read* to return 0; the next *read* will return the next record. Seeks are ignored.

An *ioctl*(2) call performs special operations:

`MTIOCTOP` perform a suboperation encoded as below in the `mt_op` field of a structure whose address is passed as the third argument of *ioctl*.

```
struct mtop {
        short   mt_op;      /* operation */
        daddr_t mt_count;   /* repeat count */
};
```

`MTWEOF`	write an end-of-file record
`MTFSF`	forward space file
`MTBSF`	backward space file
`MTFSR`	forward space record
`MTBSR`	backward space record
`MTREW`	rewind
`MTOFFL`	rewind and put the drive offline

The files described above provide a ‘raw’ interface. There is also a ‘block’ interface which attempts to treat the tape like an ordinary file as much as possible. Block tapes are accessed through files with names beginning with `mt` or `nmt`. Such a tape contains a single file, consisting of a series of 1024-byte records followed by a file mark. Seeks have their usual meaning, and it is possible to read and write a byte at a time, though writing in very small units may create enormous record gaps. The file always ends at the most recently written byte.

Conventions for minor device numbers vary among different hardware drivers:

For the TU78, the drive unit number is encoded in the two low-order bits. Adding 4 prevents the tape from rewinding at close time. Adding 8 selects 6250 bpi for writing; the default is 1600. The tape drive senses density automatically when reading.

The TE16 is like the TU78, except that the default density is 800 bpi, and adding 8 to the device number selects 1600.

For TMSCP tape drives such as the TU81 and the TK50, the unit number is encoded in the three low-order bits. Adding 128 prevents the tape from rewinding on close. Density is selected by octal bits 070; the eight possible values represent eight different device-dependent tape formats. For 9-track tape drives, add 0 for 800 bpi, 8 for 1600, 16 for 6250. For TK50 cartridge drives, add 24 (old-style block format). For TK70 drives, add 8. The tape drives sense density automatically on reading, but if a drive doesn’t support a particular density, the hardware may complain when the device is opened.

FILES

`/dev/mt?`

```
/dev/rmt?
/dev/nmt?
/dev/nrmt?
```

SEE ALSO

tape(1)

BUGS

If any non-data error is encountered, the tape drivers generally refuse to do anything more until closed.
The naming convention behaves poorly with multiple tape drives.
Block tape has probably outlived its usefulness.

NAME

null − data sink

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

`/dev/null`

NAME

proc − process file system

SYNOPSIS

```
#include <sys/types.h> #include <sys/proc.h>
#include <sys/pioctl.h>
```

DESCRIPTION

Proc is a file system that contains memory images of each running process in the system. The name of each entry in the `/proc` directory is the process id of the subject process, expressed in decimal with optional leading zeros. Each process file is owned by the userid of the subject process. The file mode includes read and write permission for the owner if that userid has read access to the associated text file; all other permission bits are zero. The file size is the sum of the sizes of virtual memory segments in the subject process.

The subject process is unaffected, except that setuid bits will be ignored if it does an *exec*(2). (Setuid bits are also ignored if the *exec*ing process has traced signals, or stops on *exec*; see the description of `PIOCSMASK` and `PIOCSEXEC` below.)

Data may be transferred from or to any locations in the subject's address space through *lseek*(2), *read*(2), and *write*. The *text segment* begins at address 0; the *data segment* starts above the text. The *user area* extends downward below address 0x80000000, and is `UPAGES*NBPG` bytes long (see the header files listed below); the *stack segment* grows downward below the user area. The text, data, and stack sizes may be determined from the process's `proc` structure (see `PIOCGETPR` below). It is an error to access addresses between data and stack. No read or write may span a segment boundary; in the user area only the locations of saved user registers are writable.

Ioctl(2) calls control the subject process. The third argument usually points to an integer. The *ioctl* codes are:

`PIOCSTOP` Send signal `SIGSTOP` to the process, and wait for it to enter the stopped state.

`PIOCWSTOP` Wait for the process to stop.

`PIOCRUN` Make the process runnable again after a stop.

`PIOCSMASK` Define a set of signals to be traced. The process will stop when it receives any signal whose number, as given in *signal*(2), corresponds to a 1-bit in the given integer, with the least significant bit counted as 1. The traced state and mask bits are inherited by the child of a *fork*(2). When the process file is closed, the mask becomes zero, but the traced state persists.

`PIOCSEXEC` Cause the process to stop after *exec*(2). This condition is inherited across *fork*(2) and persists when the process file is closed.

`PIOCREXEC` Reverse the effect of `PIOCSEXEC`.

`PIOCCSIG` Clear the subject's currently pending signal (if any).

`PIOCKILL` Set the subject's currently pending signal to a given number.

`PIOCOPENT` Return a read-only file descriptor for the subject process's text file. (Thus a debugger can find the symbol table without knowing the name of the text file.)

`PIOCNICE` Increment the priority of the subject process by a given amount as if by *nice*(2).

`PIOCGETPR` Copy the subject's `proc` structure (see `<sys/proc.h>`) from the kernel process table into an area pointed to the third argument. (This information, which resides in system space, is not accessible via a normal read.)

Any system call is guaranteed to be atomic with respect to the subject process, but nothing prevents more than one process from opening and controlling the same subject.

The following header files are useful in analyzing *proc* files:

`<signal.h>` list of signal numbers

`<sys/param.h>` size parameters

`<sys/types.h>` special system types

`<sys/user.h>`	user structure
`<sys/proc.h>`	proc structure
`<sys/reg.h>`	locations of saved user registers
`<sys/pioctl.h>`	ioctl codes for *proc* files

FILES

`/proc/*`

SEE ALSO

adb(1), *ps*(1), *hang*(1), *fmount*(2), *signal*(2), *mount*(8), *pi*(9.1)

DIAGNOSTICS

These errors can occur in addition to the errors normally associated with the file system; see *intro*(2):

`ENOENT`	The subject process has exited.
`EIO`	The subject process has attempted I/O at an illegal address.
`EBUSY`	The subject is in the midst of changing virtual memory attributes, or has pages locked for physical I/O.
`ENOSPC`	A write has been attempted on a shared text segment and there is no room on the swap space to make a copy.
`EPERM`	A non-super-user has attempted to better the subject's priority with `PIOCNICE`.

BUGS

A process must be swapped in for reading and writing (but not *ioctl*); this may cause a noticeable delay.
The spectrum of states which result in `EBUSY` is too conservative.
A process loaded from a text file with magic number 0407 does not have as a read-only text segment; in this (presumably rare) case `PIOCOPENT` does not work, and the process is accessible even if the text file is read-only.
The interface involves too many VAX-specific magic numbers.

NAME

ra − DEC MSCP disks (RA60, RA80, RA81, RA90)

DESCRIPTION

Ra devices occupy disk drives conforming to DEC's Mass Storage Control Protocol standard: drives such as the RA81 connected via controllers such as the UDA50. Files with minor device numbers 0 through 7 refer to different sections of drive 0, minor devices 8 through 16 refer to drive 1, and so on up to 63 (8 drives).

Normally the disk is accessed in 1024-byte blocks (1K). If 64 is added to the minor device number, 4096-byte blocks (4K) are used instead. A 4K device mounted as a file system is bitmapped; see *filsys*(5).

Conventionally the files are given names like `ra37` for section 7 of drive 3. There are no name rules distinguishing 1024-byte files from 4096-byte files; in practice the files are almost always the 4096-byte kind.

The start and size of the sections of each drive are as follows. Sizes are measured in 512-byte hardware sectors.

disk	start	length
0	0	10240
1	10240	20480
2	30720	249848
3	280568	249848
4	530416	249848
5	780264	arbitrarily large
6	30720	749544
7	0	arbitrarily large

The 'arbitrarily large' sections reach to the end of the disk. *Rarct* will display disk sizes; see *rarepl*(8). For example, an RA81 has 891072 sectors, so section 7 is that size, and section 5 is 891072−780264=110808 sectors. An RA90 has 2376153 sectors; section 7 is that size, section 5 is 2376153−780264=1595889 sectors. For other disks, run *rarct* and do the arithmetic.

The *ra* files discussed above access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with `rra` and end with a number which selects the same disk as the corresponding `ra` file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek*(2) calls should specify a multiple of 512 bytes.

Several *ioctl*(2) calls apply to the raw devices.

`UIOCHAR` The third argument to *ioctl* points to an object to be filled with drive parameters:

```
struct ud_unit {
        daddr_t radsize;  /* disk size, sectors */
        daddr_t rctsize;  /* RCT size, including pad */
        long    medium;   /* medium id */
        short   tracksz;  /* sectors per track */
        short   groupsz;  /* tracks per group */
        short   cylsz;    /* groups per cylinder */
        char    rbns;     /* RBNs per track */
        char    copies;   /* number of RCT copies */
};
```

`UIORRCT` The third argument points to an object of type

```
struct ud_rctbuf {
        caddr_t buf;
        int     lbn;
};
```

`buf` points to a 512-byte buffer, into which block `lbn` of the replacement and caching table (RCT) is read. As many copies of the RCT as necessary are examined to find a readable copy of the block.

`UIOWRCT` The third argument is like that of `UIORRCT`. Block `lbn` of the RCT is written in all copies.

`UIOREPL` The third argument points to an object of type:

```
struct ud_repl {
        daddr_t replbn;    /* good block */
        daddr_t lbn;       /* bad block */
        short   prim;      /* nonzero if primary replacement */
};
```

A 'replace' command is sent to the controller, requesting that attempts to access logical block *lbn* henceforth be revectored to replacement block *replbn*. *Prim* should be set nonzero if and only if *replbn* is the primary replacement block for *lbn*.

`UIOSPDW` Arrange that the disk drive will spin down when the last file using it is closed.

`UIORST` Reset the controller to which this disk is connected. Any pending operations are abandoned and return an error.

FILES

```
/dev/ra*
/dev/rra*
```

SEE ALSO

rarepl(8)
MSCP Basic Disk Functions Manual
DEC Standard Disk Format Specification

BUGS

In raw I/O *read*(2) and *write* truncate file offsets to 512-byte block boundaries, and *write* scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read*, *write* and *lseek*(2) should always deal in 512-byte multiples.

`UIORRCT` and `UIOWRCT` will misbehave if invoked on a section that doesn't start at the beginning of the disk. Section 7 (the whole disk) is the best choice.

The 1K/4K flag bit in the device number is unfortunate.

NAME

scsi − SCSI interface

SYNOPSIS

```
#include <scsi.h>
```

DESCRIPTION

The special file `/dev/scsi` provides a low level interface to a SCSI bus. Commands are transmitted to the bus by *write*; the response to each command is received with *read*(2). The format of a command is

```
unsigned long tran_id;      /* transaction id */
unsigned char target;       /* SCSI id of target device */
unsigned char flags;        /* flags for this transaction */
unsigned long nreturn;      /* number of bytes to be read back */
char cmd[10];               /* SCSI command */
char data[];                /* optional data */
```

Thus, to send *n* bytes of data, the byte count for *write* should be *n*+20. Possible flags are `SCSI_WR` (data goes from host to SCSI), `SCSI_RD` (data goes from SCSI to host), `SCSI_BRESET` (reset the SCSI bus), `SCSI_RESET` (reset the controller), and `SCSI_SENSE` (return extended sense data on error). For most controllers, `SCSI_BRESET` implies `SCSI_RESET`. Flags are OR'ed together, and there must be exactly one of `SCSI_WR` and `SCSI_RD`. The SCSI command should terminate within a small time (currently 10 seconds); a longer limit (300 seconds) can be specified by using `SCSI_LTMOUT`.

The data read is structured as

```
unsigned long tran_id;      /* transaction id */
unsigned char status;       /* scsi status byte */
unsigned char message;      /* scsi message byte */
unsigned char flags;        /* flags for this transaction */
unsigned char c_type;       /* 1=td 2=us */
unsigned short c_reg1;      /* td=sa, us=csr */
unsigned short c_reg2;      /* td=mscp, us=per */
unsigned char sense[22];    /* extended sense data */
char pad[2];
char data[];                /* any data */
```

Thus, to read *n* bytes of data. the byte count to *read* should be *n*+36. If *flags* has the `SCSI_CERR` bit set, there was a controller error, which is described by the `c_` fields. The values of `csr` (or `sa`) and `per` (or `mscp`) are documented in the interface manual for the U.S. Design 1158 Unibus controller (or T.D. Systems Viking controller). If the `SCSI_SENSE` bit was set in the *write*, and the status byte shows a check condition, an attempt is made to get extended sense information. If the attempt succeeds the `SCSI_SENSE` is set in *flags*. Otherwise, the status and message bytes for the failed attempt are placed in `sense[0]` and `sense[1]` respectively.

The transaction id identifies which *write* caused the results for this *read*. This will become more important when multiple simultaneous transactions are allowed.

FILES

`/dev/scsi`

SEE ALSO

scsish(8)

NAME

stream − communication channels

SYNOPSIS

```
#include <sys/filio.h>
#include <sys/ttyio.h>
```

DESCRIPTION

A stream is a connection between two processes, or between a process and a device. It is referred to by a file descriptor, and ordinary read and write calls apply. When a write call is given on a stream whose other end has disappeared, for example because the process at other end of a pipe has terminated, or a device has hung up, signal `SIGPIPE` is generated; if the signal is ignored, the write call returns error `EPIPE`. The first 64 attempts to read such a disconnected stream return 0; subsequent attempts generate `SIGPIPE` signals.

Pipes are streams; so are most communication devices like terminals and networks.

Line disciplines may be inserted into a stream to do various sorts of processing. Line disciplines within a stream are identified by their position; level 0 is nearest the process. Line disciplines on one end of a pipe cannot be seen from the other. Line discipline types are integers; a list is given below.

These *ioctl* calls, defined in `<sys/filio.h>`, manipulate line disciplines:

`FIOPUSHLD` The third argument points to an integer naming a line discipline; insert that line discipline at level 0.

`FIOINSLD` The third argument points to a structure:

```
struct insld {
        short ld;
        short level;
};
```

Insert the line discipline named by `ld` in the stream at depth `level`. If there are fewer than `level` line disciplines in the stream, an error is returned.

`FIOPOPLD` The third argument points to an integer; remove the line discipline at that level in the stream. A null pointer (the usual case) means level 0.

`FIOLOOKLD` The third argument points to an integer; return the type of the line discipline at that level, both in the same integer and as the return value from *ioctl*. A null pointer means level 0.

These *ioctl* calls, also in `<sys/filio.h>`, perform other operations on streams:

`FIOSNDFD` The third argument points to an integer naming an open file descriptor. Send a reference to that file to the other end of the stream. This works only on pipes. The reference is unaffected by intervening line disciplines; in particular it cannot be intercepted or forged by *mesgld*(4). `FIOSNDFD` returns immediately; it does not wait for the reference to be received.

`FIORCVFD` The third argument points to a structure:

```
struct passfd {
        int   fd;
        short uid;
        short gid;
        short nice;
        char  logname[8];
};
```

Receive a file reference sent by `FIOSNDFD`; fill in the structure with a file descriptor `fd` for the passed file, and the userid `uid`, groupid `gid`, login name `logname`, and

scheduling priority `nice` of the sending process. The file reference must be the next message in the stream; if data precedes it, `EIO` is returned. If the stream is empty, `FIORCVFD` blocks until data or a file reference arrives.

`FIONREAD` The third argument points to an integer; fill it in with the number of characters that may be read from this stream without blocking.

These *ioctl* calls also work on streams, but are defined in `<sys/ttyio.h>` for historic reasons:

`TIOCSPGRP` The third argument points to a short integer. If the pointer is null (the usual case), make a new process group for the current process, and associate the group with this stream. If the pointer is not null, it points to a process group id; associate that group with this stream. When the lowest level of a stream receives a signal message (like `SIGINT` or `SIGQUIT` from *ttyld*(4)), the signal is sent to processes in the associated process group. If the stream is shut down prematurely, the process group is sent `SIGHUP`.

`TIOCGPGRP` The third argument points to a short integer; fill it in with the process group id associated with this stream. 0 means no group.

`TIOCEXCL` Mark this stream so that future opens are forbidden except to the super-user or to processes in the associated process group.

`TIOCNXCL` Remove the mark left by `TIOCEXCL`.

`TIOCSBRK` Send a break on a serial line.

`TIOCFLUSH` Throw away queued data anywhere in the stream.

Here is a list of line discipline types. The names refer to global integers defined in the C library.

`tty_ld` Regular terminal processing, *ttyld*(4).
`ntty_ld` Restricted Berkeley ‘new tty’ terminal processing; see the Berkeley Users Manual.
`cdkp_ld` Character-mode Datakit universal receiver protocol, *dk*(4).
`dkp_ld` Block-mode Datakit universal receiver protocol, *dk*(4).
`rdk_ld`
`uxp_ld` Datakit protocols used in call setup, *dk*(4).
`buf_ld` Buffering mechanism, *bufld*(4).
`mesg_ld`
`rmesg_ld` Turn stream controls into ordinary data and vice versa, *mesgld*(4).
`conn_ld` Make unique connections to a server, *connld*(4).
`ip_ld`
`tcp_ld`
`udp_ld` Internet protocols, *internet*(3).

SEE ALSO

ioctl(2), *signal*(2), *mesgld*(4), *ipc*(3)
D. M. Ritchie, ‘A Stream I/O System’, this manual, Volume 2

NAME

tcp, tcp_ld − DARPA transmission control protocol

SYNOPSIS

```
#include <sys/inio.h>
#include <sys/inet/tcp_user.h>
```

DESCRIPTION

The *tcp_ld* line discipline and the `/dev/tcp*` devices together implement the DARPA TCP circuit protocol. They are normally used through *tcpmgr*(8) and the routines in *ipc*(3).

One instance of *tcp_ld* should be pushed on an IP device stream, usually `/dev/ip6`; see *ip*(4). Thereafter, data written on the *tcp* devices is turned into IP packets written to the IP device, and vice versa.

Different *tcp* devices represent different software channels. Files with odd minor device numbers are for placing calls; while such a file is open, it may not be opened again. Files with even device numbers receive calls.

To place a call, open an unused odd-numbered *tcp* file; write a `struct tcpuser` describing the address to be called; and read a `struct tcpuser` for status. The structure is defined in `<sys/inet/tcp_user.h>`:

```
struct tcpuser {
      int code;
      tcp_port lport, fport;
      in_addr laddr, faddr;
      int param;
};

#define TCPC_LISTEN     1
#define TCPC_CONNECT    2

#define TCPC_OK         3
#define TCPC_SORRY      4     /* unknown error */
#define TCPC_BADDEV     5     /* tcp device is bad */
#define TCPC_NOROUTE    6     /* no routing to dest */
#define TCPC_BADLOCAL   7     /* bad local address */
#define TCPC_BOUND      8     /* address already bound */

#define SO_KEEPALIVE    0x2   /* generate keepalives */
```

In the structure describing the call, `code` should be `TCPC_CONNECT`; `faddr` and `fport` are the destination IP address and TCP port number; `laddr` is the IP address associated with a local IP interface, or `INADDR_ANY` to let the system pick; `lport` is the local TCP port number, or 0 to let the system pick; `param` is 0 or `SO_KEEPALIVE`.

In the structure returned for status, `code` is `TCPC_OK` if the call completed correctly; henceforth data written to and read from the file is transported on the circuit. Other codes mean the circuit was not set up.

To listen for incoming calls, open an odd-numbered device and write a `struct tcpuser` with `code` set to `TCPC_LISTEN`; `laddr` set to the local IP address for which calls should be taken, or `INADDR_ANY` to catch any calls not explicitly taken by another listener; `lport` set to the port on which to listen, or 0 for any port; and `param` set to 0. Thereafter, reads return successive `tcpuser` structures, each describing a new call; `faddr` and `fport` identify the caller, `laddr` and `lport` the assigned local address. The local *tcp* device number, *n*, assigned to the call is returned in `param`. The corresponding device, `/dev/tcp`*n*, should be opened; data read and written there is transported by the circuit.

Several *ioctl*(2) calls, defined in `<sys/inio.h>`, apply to *tcp* devices:

`TCPIOHUP` When the remote end of the circuit is disconnected, send signal `SIGHUP` to the local process group associated with the stream.

`TCPMAXSEG` The third argument points to an integer giving the maximum segment size for this connection: the greatest number of bytes to be packed into one IP packet.

`TCPGETADDR` The third argument points to a `struct tcpuser`; fill in `laddr`, `lport`, `faddr`, and `fport` with the local and foreign addresses associated with the circuit.

FILES

`/dev/tcp??`
`/dev/ip6`

SEE ALSO

ip(4), *internet*(3), *tcpmgr*(8)
DARPA standards RFC 793, 1122

NAME

tty − serial line interface drivers

SYNOPSIS

```
#include <sys/ttyio.h>
```

DESCRIPTION

The files `/dev/tty*` refer to serial line devices such as the DZ11. They are normally used in conjunction with the terminal line discipline, *ttyld*(4).

Certain device-related parameters, such as parity and line speed, may be set by *ioctl*(2) calls:

`TIOCGDEV` The argument points to a `ttydevb` structure to be filled in with current settings.
`TIOCSDEV` The argument points to a `ttydevb` structure from which the parameters are set.

The `ttydevb` structure, as defined in `<sys/ttyio.h>`, is

```
struct ttydevb {
        char    ispeed; /* input speed */
        char    ospeed; /* output speed */
        short   flags;  /* mode flags */
};
```

The speeds are encoded as follows. Impossible speeds are ignored.

`B0`	0	(hang up device)
`B50`	1	50 baud
`B75`	2	75 baud
`B110`	3	110 baud
`B134`	4	134.5 baud
`B150`	5	150 baud
`B200`	6	200 baud
`B300`	7	300 baud
`B600`	8	600 baud
`B1200`	9	1200 baud
`B1800`	10	1800 baud
`B2400`	11	2400 baud
`B4800`	12	4800 baud
`B9600`	13	9600 baud
`EXTA`	14	External A
`EXTB`	15	External B

The flags are:

`F8BIT`	040	eight-bit input and output
`ODDP`	0100	odd parity
`EVENP`	0200	even parity

If `F8BIT` is set, all eight bits of each output character are transmitted without imposing parity, and all eight bits of each input character are passed back without parity checking or stripping. Otherwise, `EVENP` requests that even parity be accepted and generated, `ODDP` odd parity. If both `EVENP` and `ODDP` are set, or if both are clear, even parity is generated and any parity is accepted.

For DZ11 lines, 1200 baud and 8-bit mode are the defaults. The transmit and receive speeds are the same; `ospeed` is ignored.

SEE ALSO

ioctl(2), *ttyld*(4)

BUGS

Every hardware interface doesn't support every operation.

NAME

tty_ld − terminal processing line discipline

SYNOPSIS

```
#include <sys/ttyio.h>
```

DESCRIPTION

Tty_ld is usually inserted into a stream connected to a terminal device. It gathers input into lines, handles special characters like erase, kill, and interrupt, inserts output delays, and the like. It does not deal with hardware parameters such as speed and parity; see *tty*(4) for such matters.

Certain special characters have particular meaning on input. These characters are not passed to a program except in raw mode, where they lose their special character. It is possible to change these characters from the default.

The *erase* character (backspace by default) erases the last-typed character. It will not erase beyond the beginning of a line or an end-of-file character.

The *kill* character (default `@`) erases the entire preceding part of the line, but not beyond an end-of-file character.

The *end-of-file* character (default control-`d`) causes any characters waiting to be read to be passed immediately to the program, without waiting for newline. The end-of-file character itself is discarded. Thus if the end-of-file character occurs at the beginning of a line, there are no characters waiting, and zero characters will be passed back; this is the standard end-of-file indication.

The *escape* character (`\`) escapes a following erase, kill, or end-of-file character and allows it to be treated as ordinary data.

The *interrupt* character (default DEL) is not passed to a program but sends signal `SIGINT` to any processes in the process group of the stream; see *signal*(2) and *stream*(4).

The *quit* character (default FS, control-`\`) sends signal `SIGQUIT`.

The *stop* character (default DC3, control-`s`) delays printing on the terminal until something is typed in.

The *start* character (default DC1, control-`q`) restarts printing after a stop character without generating any input to a program.

Two *ioctl*(2) calls affect these characters:

`TIOCGETC` The argument points to a `tchars` structure to be filled in with current settings.
`TIOCSETC` The argument points to a `tchars` structure from which the characters are set.

The `tchars` structure, as defined in `<sys/ttyio.h>`, is

```
struct tchars {
        char    t_intrc;  /* interrupt */
        char    t_quitc;  /* quit */
        char    t_startc; /* start output */
        char    t_stopc;  /* stop output */
        char    t_eofc;   /* end-of-file */
        char    t_brkc;   /* input delimiter (like nl) */
};
```

A character value of 0377 eliminates the effect of that character. The `t_brkc` character, by default 0377, acts like a new-line in that it terminates a line, is echoed, and is passed to the program. The stop and start characters may be the same, to produce a toggle effect. It is counterproductive to make other special characters (including erase and kill) identical.

Two *ioctl* calls affect other terminal processing parameters:

`TIOCGETP` The argument points to a `sgttyb` structure to be filled in with the current settings.

TIOCSETP The argument points to a sgttyb structure from which the parameters are set.

The sgttyb structure, as defined in <sys/ttyio.h>, is

```
struct sgttyb {
        char    sg_ispeed;/* unused */
        char    sg_ospeed;/* unused */
        char    sg_erase; /* erase character */
        char    sg_kill;  /* kill character */
        short   sg_flags; /* mode flags */
};
```

The flag bits are

```
ALLDELAY 0177400 Delay algorithm selection
VTDELAY  0040000 Form-feed and vertical-tab delays:
FF0      0
FF1      0040000
CRDELAY  0030000 Carriage-return delays:
CR0      0
CR1      0010000
CR2      0020000
CR3      0030000
TBDELAY  0006000 Tab delays:
TAB0     0
TAB1     0002000
TAB2     0004000
XTABS    0006000
NLDELAY  0001400 New-line delays:
NL0      0
NL1      0000400
NL2      0001000
NL3      0001400
RAW      0000040 Raw mode: wake up on all characters
CRMOD    0000020 Map CR into LF; echo LF or CR as CR-LF
ECHO     0000010 Echo (full duplex)
LCASE    0000004 Map upper case to lower on input
CBREAK   0000002 Return each character as soon as typed
TANDEM   0000001 Automatic flow control
```

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay.

If a form-feed/vertical tab delay is specified, it lasts for about 2 seconds.

Carriage-return delay type 1 lasts about .08 seconds; type 2 about .16 seconds; type 3 about .32 seconds.

New-line delay type 1 is supposed to be for the Teletype model 37; type 2 is about .10 seconds.

Tab delay type 1 is supposed to be for the Teletype model 37. Type 3, called XTABS, is not a delay at all but causes tabs to be replaced by the appropriate number of spaces on output.

In RAW mode, every character is passed immediately to the program without waiting until a full line has been typed. No erase or kill processing is done; the end-of-file, interrupt, and quit characters are not treated specially. There are no delays and no echoing, and no replacement of one character for another.

CRMOD causes input carriage returns to be turned into new-lines; input of either CR or LF causes CR-LF both to be echoed (for terminals without a new-line function).

CBREAK is a sort of half-cooked mode. Programs read each character as soon as typed, instead of waiting for a full line, but quit and interrupt work, and output delays CRMOD, XTABS, and ECHO work

normally. On the other hand there is no erase or kill, and no special treatment of \ or end-of-file.

`TANDEM` mode causes the system to transmit the stop character whenever the input queue is in danger of overflowing, and the start character when the input queue has drained sufficiently. It is useful for flow control when the 'terminal' is actually another machine that obeys the conventions.

SEE ALSO

getty(8), *stty*(1), *signal*(2), *ioctl*(2), *stream*(4), *tty*(4)

BUGS

The escape character cannot be changed.

5 FILE CONVENTIONS

5 FILE CONVENTIONS

NAME

a.out − object file format

SYNOPSIS

```
#include <a.out.h>
```

DESCRIPTION

A.out is the default name of the output file of the assembler *as*(1) or the link editor *ld*(1). Both programs make *a.out* executable if there were no errors and no unresolved external references. An object file has five sections: a header, the program text and data, relocation information, a symbol table and a string table (in that order). The last three may be absent; see *strip*(1) and option `-s` of *ld*(1). The header format, given in `<a.out.h>`, is

```
struct exec {
        long      a_magic;   /* magic number */
        unsigned  a_text;    /* size of text segment */
        unsigned  a_data;    /* size of initialized data */
        unsigned  a_bss;     /* size of uninitialized data */
        unsigned  a_syms;    /* size of symbol table */
        unsigned  a_entry;   /* entry point */
        unsigned  a_trsize;  /* size of text relocation */
        unsigned  a_drsize;  /* size of data relocation */
};
#define  OMAGIC    0407       /* old impure format */
#define  NMAGIC    0410       /* read-only text */
#define  ZMAGIC    0413       /* demand load format */
```

Macros which take `exec` structures as arguments and tell whether the file has a reasonable magic number or return offsets:

```
#define  N_BADMAG(x) (((x).a_magic)!=OMAGIC && \
         ((x).a_magic)!=NMAGIC && ((x).a_magic)!=ZMAGIC)
#define  N_TXTOFF(x) \
         ((x).a_magic==ZMAGIC ? 1024 : sizeof (struct exec))
#define N_SYMOFF(x) (N_TXTOFF(x) + (x).a_text+(x).a_data + \
         (x).a_trsize+(x).a_drsize)
#define  N_STROFF(x) (N_SYMOFF(x) + (x).a_syms)
```

Sizes are expressed in bytes. The size of the header is not included in any of the other sizes.

When an `a.out` file is executed, a memory image of three segments is set up: the text segment, the data segment, and a stack. The text segment begins at virtual address 0. Following the text segment is the data segment, in which explicitly initialized data come first, then other data, initialized to 0. The stack occupies the highest possible locations in the core image, automatically growing downwards from `0x7ffff400` as needed. The data segment may be extended by *brk*(2).

If the magic number in the header is `OMAGIC`, the text segment is neither write-protected nor shared and the data segment is immediately contiguous with the text segment. This kind of executable program is rarely used. If the magic number is `NMAGIC` or `ZMAGIC`, the data segment is loaded at the first 0 mod 1024 address following the text segment, and the text segment is write-protected and shared among all processes that are executing the same file. `ZMAGIC` format, which *ld*(1) produces by default, supports demand paging: the text and data segments are multiples of 1024 bytes long and begin at byte offsets of 0 mod 1024 in the `a.out` file.

Macros are provided to compute the absolute offset of various parts of the file:

`N_TXTOFF` Text segment.
`N_SYMOFF` Symbol table.
`N_STROFF` String table.

The offsets of the data segment, text relocation information, and data relocation information are obtained by successively adding to `N_TXTOFF` the size fields `a_text`, `a_data`, and `a_trsize`. The first 4 bytes of the string table contain it size, including the 4 bytes.

The layout of a symbol table entry, as given in `<a.out.h>`, is

```
struct nlist {
        union {
            char      *n_name;/* for use when in-core */
            long      n_strx; /* index into file string table */
        } n_un;
        unsigned char n_type; /* type flag; see below */
        char          n_other;
        short         n_desc; /* see stab(5) */
        unsigned      n_value;/* value of this symbol (or struct offset) */
};
```

Basic values for `n_type`:

```
#define  N_UNDF        0x0     /* undefined */
#define  N_ABS         0x2     /* absolute */
#define  N_TEXT        0x4     /* text */
#define  N_DATA        0x6     /* data */
#define  N_BSS         0x8     /* bss */
#define  N_COMM        0x12    /* common (internal to ld) */
#define  N_FN          0x1f    /* file name symbol */
#define  N_EXT         01      /* external bit, or'ed in */
#define  N_TYPE        0x1e    /* mask for all the type bits */
```

Other permanent symbol table entries have some `N_STAB` bits set. These are given in `<stab.h>`:

```
#define  N_STAB        0xe0    /* if any of these bits set, keep */
```

The field `n_un.n_strx` gives an index into the string table; 0 indicates that no name is associated with the entry. The field `n_un.n_name` can be used to refer to the symbol name only if the program sets this up using `n_strx` and appropriate data from the string table.

A symbol of type undefined external with nonzero value field names a common region; the value specifies its size.

Relocation information occupies eight bytes per relocatable datum:

```
struct relocation_info {
        int       r_address;       /* address of datum to be relocated */
        unsigned  r_symbolnum:24,  /* local symbol ordinal */
                  r_pcrel:1,       /* is referenced relative to pc */
                  r_length:2,      /* 0=byte, 1=word, 2=long */
                  r_extern:1,      /* symbol value unknown */
                  :4;              /* nothing, yet */
};
```

If `r_extern` is 1, the datum designated by `r_address` and `r_length` will be relocated by adding to it the value of the associated external symbol. If `r_extern` is 0, `r_symbolnum` is encoded in the style of `n_type` and the value will be relocated by adding the relocated base of the designated area (text, initialized data, or common data).

SEE ALSO

adb(1), *as*(1), *ld*(1), *nm*(1), *stab*(5), *strip*(1)

NAME

acct − execution accounting file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/acct.h>
```

DESCRIPTION

Acct(2) causes an entry to be appended to an accounting file for each process that terminates. The layout of an accounting file entry, as given in `<sys/acct.h>`, is

```
typedef unsigned short comp_t;

struct  acct
{
        char   ac_comm[10];  /* command name */
        comp_t ac_utime;     /* user time */
        comp_t ac_stime;     /* system time */
        comp_t ac_etime;     /* elapsed time */
        time_t ac_btime;     /* beginning time */
        short  ac_uid;       /* user ID */
        short  ac_gid;       /* group ID */
        short  ac_mem;       /* average memory usage */
        comp_t ac_io;        /* number of disk IO blocks */
        dev_t  ac_tty;       /* control typewriter */
        char   ac_flag;      /* flag */
};
```

Values in `ac_flag`:

```
#define AFORK  01            /* has executed fork, but no exec */
#define ASU    02            /* used super-user privileges */
```

If the process does an *exec*(2), the first 10 characters of the filename appear in `ac_comm`.

The type `comp_t` counts 60- or 50-cycle clock ticks in a private floating-point format: a three-bit base-8 exponent and a 13-bit unsigned mantissa. Thus the number of clock ticks that a process ran is expressed by *(ac_etime&017777)<<((ac_etime>>13)&03)*. The beginning time, `ac_btime`, is recorded in the format of *time*(2).

SEE ALSO

acct(2), *sa*(8)

NAME

ar − archive (library) file format

SYNOPSIS

```
#include <ar.h>
```

DESCRIPTION

The archive command *ar*(1) is used to combine several files into one. Archives are used mainly as libraries to be searched by the link-editor *ld.*

A file produced by *ar* has a magic string at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
#define ARMAG   "!<arch>\n"
#define SARMAG  8

#define ARFMAG  "`\n"

struct ar_hdr {
        char    ar_name[16];
        char    ar_date[12];
        char    ar_uid[6];
        char    ar_gid[6];
        char    ar_mode[8];
        char    ar_size[10];
        char    ar_fmag[2];
};
#define SAR_HDR 60
```

The name is a blank-padded string. The `ar_fmag` field contains `ARFMAG` to help verify the presence of a header. The other fields are left-adjusted, blank-padded numbers. They are decimal except for `ar_mode`, which is octal. The date is the modification date of the file at the time of its insertion into the archive. The length of the header is `SAR_HDR`. Because `struct ar_hdr` may be padded on some machines, `SAR_HDR` should be used in preference to `sizeof(struct ar_hdr)` when reading and writing file headers.

Each file begins on an even (0 mod 2) boundary; a newline is inserted between files if necessary. Nevertheless `ar_size` reflects the actual size of the file exclusive of padding.

There is no provision for empty areas in an archive file.

If an archive contains only printable files, the archive itself is printable.

SEE ALSO

ar(1), *ld*(1), *nm*(1)

BUGS

File names lose trailing blanks.
Most software that deals with archives takes an embedded blank as a name terminator.

NAME

backup − incremental backup files

DESCRIPTION

The backup system consists of a number of client machines, and a ‘backup machine’, which has a database and a collection of backup copies of files. On clients files concerned with backup live in a directory, normally `/usr/lib/backup`, defined in the shell script `/usr/bin/backup`. The file `/usr/lib/backup/conf`, needed on both client and backup machine, has up to three lines, namely the backup machine name, the default backup device, and the directory, hereafter called `$FM`, where the rest of the backup software lives on the backup machine. Client systems normally have just the first line. The rest of this description applies to the backup machine.

The databases live in `$FM/db` and are maintained in *cbt*(3) form. The main database, called `filemap`, stores two mappings. The first maps filename-time pairs to backup copy names, thus:

```
/n/bowell/usr/jim/goo//519487622    v/v22/17
```

The number after is the inode change date, expressed in seconds since the epoch; see *stat*(2). If the backup copy is still on magnetic disk, it will be called `$FM/v/v22/17`; otherwise it will be `v22/17` on some optical disk. (The mapping of backup copy name to optical disk name is kept in `$FM/adm/volidmap`.) The second mapping maps filenames to the time of their most recently backed-up version:

```
/n/bowell/usr/jim/goo    520514116
```

The second database, `dir`, maps directoryname-time pairs to the contents of that directory. This allows quick recovery of file trees.

The third database, `fs`, maps filename-time pairs to (essentially) inodes. This allows efficient implementation of `backup mount`; see *backup*(1).

The program `$FM/bin/dbupdate` manages these databases. The *dir* and *fs* databases are optional; they will be updated only if they already exist. The program `$FM/bin/sweep` also assigns the backup copy names into a flat directory structure. A new directory is used when the total size of the files in the current directory would exceed 20000K bytes, rounding each file size up to a multiple of 4K.

The backup copy of a file consists of a header that gives the original inode, pathname and owner (as a string), followed by the contents of the file. Directories are stored as a sequence of entry names.

To prevent multiple writers into a database, a lockfile is used. The content of this file is the process id of the process accessing the database. Locks are removed by `$FM/bin/rmlocks` executed by *rc*(8) when the system boots.

The backup system supports multiple *filemap* databases (this allows the current database to be kept small). The list of database names is kept in `$FM/db/filemaplist`, one per line in order of increasing priority. The last name is assumed to be the active database; all the others are read-only.

Programs such as *sweep* and *dbupdate* leave droppings in the log file `$FM/log`.

Statistics of the numbers of files and bytes saved for users of a given system are kept in `$FM/stat.dir/`*system*. Each file consists of a sequence of records with a machine-independent structure; generally, one record per user per day. The records are maintained by `$FM/bin/procstats`, which processes the file `$FM/stat.log` that is maintained by *dbupdate*.

To allow quick searching for filenames with full regular expressions, a simple sorted list of all saved filenames is often kept, normally in `$FM/filenames`.

The device (and system) used for recovering files can be specified in many ways. In order of decreasing priority: a `-f` option in *backup recover* or *backup fetch* (see *backup*(1)), a default device on the client system (in line 2 of `/usr/lib/backup/conf`), the default device on the backup system.

FILES

`/usr/lib/backup/*`

```
/usr/lib/backup/conf
/usr/backup/db
/usr/backup/locks
/usr/backup/log
/usr/backup/filenames
```

SEE ALSO

backup(1), *worm*(8), *backup*(8), *cbt*(1), *stat*(2)
A. Hume, ‘The File Motel: an Owner’s Manual’, this manual, Volume 2

NAME

core − format of memory image file

DESCRIPTION

A memory image of a terminated process is written into file `core` when any of various errors occur; see *signal*(2). The memory image is written in the process's working directory, if it can be; normal access controls apply.

A core file consists of a copy of the user block for the terminated process, followed by an image of its data and stack segments. The user block, which occupies 5K bytes, contains descriptive information about the process and a stack used by the operating system when serving the process; see the include file `<sys/user.h>`.

SEE ALSO

adb(1), *pi*(9.1), *bigcore*(1), *signal*(2)

NAME

cpio − format of cpio archive

DESCRIPTION

The archived files are recorded consecutively, each preceded by a `header`. The header structure, when the `-c` option of *cpio*(1) is not used, is:

```
typdef unsigned short ushort;
struct {
      short  h_magic,
             h_dev;
      ushort h_ino,
             h_mode,
             h_uid,
             h_gid;
      short  h_nlink,
             h_rdev,
             h_mtime[2],
             h_namesize,
             h_filesize[2];
      char   h_name[h_namesize rounded to word];
} Hdr;
```

When the `-c` option is used, the `header` information is printable, as described by the *printf*(3) call

```
printf(Chdr, "%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
      Hdr.h_magic, Hdr.h_dev, Hdr.h_ino, Hdr.h_mode,
      Hdr.h_uid, Hdr.h_gid, Hdr.h_nlink, Hdr.h_rdev,
      Longtime, Hdr.h_namesize, Longfile, Hdr.h_name
```

`Longtime` and `Longfile` are equivalent to `Hdr.h_mtime` and `Hdr.h_filesize`, respectively. Every instance of `h_magic` contains the octal constant `070707`. The items `h_dev` through `h_mtime` have meanings explained in *stat*(2). The length of the null-terminated path name `h_name`, including the null byte, is given by `h_namesize`.

The last element of the archive is a dummy entry for the name `TRAILER!!!`, with padding to a multiple of 512 bytes. Special files, directories, and the trailer are recorded with `h_filesize` equal to zero.

SEE ALSO

cpio(1), *find*(1), *stat*(2).

NAME

dir − format of directories

SYNOPSIS

```
#include <sys/types.h>
#include <sys/dir.h>
```

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its inode entry; see *filsys*(5). The structure of a directory entry as given in the include file is:

```
#define DIRSIZ  14
struct  direct
{
        ino_t   d_ino;
        char    d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are `.` for the directory itself and `..` for the parent directory. In the ultimate root directory `..` is the same as `.`.

It is inadvisable to read directories using this structure. The routines in *directory*(3) and *dirread*(2) are more efficient and portable.

SEE ALSO

filsys(5), *directory*(3), *dirread*(2)

NAME

environ − user environment

SYNOPSIS

```
extern char **environ;
```

DESCRIPTION

An array of strings called the 'environment' is made available by *exec*(2) when a process begins. By convention these strings have either the form *name*=*value*, defining a variable, or *name*`()`{*value*}, defining a function; see *sh*(1). The following variables are used by various commands:

`PATH` The sequence of directory prefixes that *sh*, *time*(1), *nice*(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by `:`. *Login*(8) sets `PATH=:/bin:/usr/bin`.

`HOME` A user's login directory, set by *login*(8) from the password file *passwd*(5).

`TERM` The kind of terminal for which output is to be prepared. This information is used by commands, such as *nroff* or *plot*(1), which may exploit special terminal capabilities. Some customary values of `TERM` are `2621` (HP), `4014` (Tektronix), `5620` (Teletype), and `630` (Teletype). See `/etc/termcap`, described in *termcap*(5), for a longer list.

`SHELL` The name of the login shell.

The environment may be queried by *getenv*(3) or by the `set` or `whatis` commands of *sh*(1). Names may be placed in the environment by the `export` command and by *name*=*value* arguments of *sh*(1). Names may also be placed in the environment at the point of an *exec*(2). It is unwise to conflict with certain *sh*(1) variables that are frequently exported by `.profile` files: `MAIL`, `PS1`, `PS2`, `IFS`.

SEE ALSO

sh(1), *printenv*(1), *exec*(2), *getenv*(3), *term*(6)

BUGS

Function definitions in the environment break some old programs, including old shells.

NAME

filsys, flblk, ino − format of a disk file system

SYNOPSIS

```
#include <sys/types.h>
#include <sys/fblk.h>
#include <sys/filsys.h>
#include <sys/ino.h>
```

DESCRIPTION

Every file system is divided into a certain number of blocks of 1K or 4K bytes, as determined by the predicate `BITFS()` applied to the minor device number where the file system is mounted. Block 0 is unused and is available to contain a bootstrap program, pack label, or other information.

Block 1 is the 'super-block'. Its layout is defined in `<sys/filsys.h>`:

```
struct filsys {
        unsigned short s_isize;
        daddr_t   s_fsize;
        short     s_ninode;
        ino_t     s_inode[NICINOD];
        char      s_flock;
        char      s_ilock;
        char      s_fmod;
        char      s_ronly;
        time_t    s_time;
        daddr_t   s_tfree;
        ino_t     s_tinode;
        short     s_dinfo[2];
#define s_m       s_dinfo[0]
#define s_n       s_dinfo[1]
#define s_cylsize s_dinfo[0]
#define s_aspace  s_dinfo[1]
        char      s_fsmnt[14];
        ino_t     s_lasti;
        ino_t     s_nbehind;
        union {
                  struct {
                      short   S_nfree;
                      daddr_t S_free[NICFREE];
                  } R;
                  struct {
                      char    S_valid;
#define BITMAP    961
                      long    S_bfree[BITMAP];
                  } B;
                  struct {
                      char    S_valid;
                      char    S_flag;  /* 1 means bitmap not in S_bfree */
                      long    S_bsize; /* size of bitmap blocks */
                      struct buf *S_blk[BITMAP-1];
                  } N;
        } U;
};
#define s_nfree   U.R.S_nfree
#define s_free    U.R.S_free
```

```
#define s_valid   U.B.S_valid
#define s_bfree   U.B.S_bfree
```

`s_isize` The address of the first block after the i-list, which starts in block 2. Thus the i-list is `s_isize-2` blocks long.

`s_fsize` The address of the first block not in the file system.

`s_inode` Array of free inode numbers.

`s_ninode` The number of free i-numbers in the `s_inode` array. Inodes are placed in the list in LIFO order. If the list underflows, it is replenished by searching the i-list to obtain the numbers of free inodes. When the list is full, freed inodes are not recorded in `s_inode`.

`s_lasti` Where the last search for free inodes ended.

`s_nbehind`
Number of free inodes before `s_lasti` that are not listed in `s_inode`. The system will search forward for free inodes from `s_lasti` for more inodes unless `s_nbehind` is sufficiently large, in which case it will search the i-list from the beginning.

`s_flock`
`s_ilock` Flags maintained in the core copy of the super-block while the file system while it is mounted. The values on disk are immaterial.

`s_fmod` Flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information. The value on disk is immaterial.

`s_ronly` Flag for read-only file system. The value on disk is immaterial.

`s_time` Time of the last change to the super block.

`s_dinfo` Disk interleave information: `s_cylsize=` blocks per cylinder, `s_aspace=` blocks to skip; see *fsck*(8).

`s_fsmnt` Unused.

`s_tfree`
`s_tinode` Numbers of free blocks and free inodes. Maintained for the benefit of *df* (see *du*(1)), these values are otherwise irrelevant.

Different data are used to manage free space in 1K and 4K file systems. These fields are for 1K file systems:

`s_free` An array of free block numbers. `s_free[0]` is the block address of the next in a chain of blocks constituting the free list. The layout of these blocks is defined in `<sys/fblk.h>`:

```
struct fblk {
        int       df_nfree;
        daddr_t   df_free[NICFREE];
}
```

where `df_nfree` and `df_free` are exactly like `s_nfree` and `s_free`.

`s_nfree`
Blocks given in `s_free[1]` through `s_free[s_nfree-1]` are available for allocation. Blocks are allocated in LIFO fashion from this list. If freeing would cause the array to overflow, it is cleared by copying into the newly freed block, which is pushed onto the free chain. If allocation would cause underflow, the array is replenished from the next block on the chain.

These are for 4K file systems:

`s_bfree` a bit array specifying the free blocks of a 4K file system. The bit `(s_bfree[i/w]>>(i%w))&1`, where *w* is the bit size of a long, is nonzero if the *i*th

data block is free. If the file system is too large for the bitmap to fit here, then it is stored at the end of the file system, and locked into memory when the file system is mounted. The `N` variant of the union is used by the kernel in this case.

`s_valid` The bitmap of a mounted file system is maintained only in main memory; the bitmap on the medium is marked invalid by setting `s_valid` to zero. Unmounting updates the medium copy and sets `s_valid` to 1. A file system with invalid bitmap may be mounted read-only; its bitmap can be corrected by *chuck*(8).

I-numbers begin at 1, and the storage for inodes begins in block 2. Inodes are 64 bytes long. Inode 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each inode represents one file.

The layout of an inode is defined in `<sys/ino.h>`:

```
struct dinode {
        unsigned short di_mode;
        short    di_nlink;
        short    di_uid;
        short    di_gid;
        off_t    di_size;
        char     di_addr[40];
        time_t   di_atime;
        time_t   di_mtime;
        time_t   di_ctime;
};
```

`di_mode` The kind of file; it is encoded as `st_mode field of` *stat*(2), and is 0 for a free inode.

`di_nlink` The number of directory entries (links) that refer to this inode

`di_uid` Owner's userid.

`di_gid` Owner's groupid.

`di_size` Number of bytes in the file.

`di_atime` Time of last access; see *times*(2).

`di_mtime` Time of last modification.

`di_ctime` Time of last change to inode or contents.

`di_addr` For special files the first two bytes of `di_addr` contain the device code; see *intro*(4) and *types*(5).

For plain files and directories `di_addr` contains block numbers packed into 3 bytes each. The first 10 numbers specify device blocks directly. The last 3 are singly, doubly, and triply indirect and point to blocks of block pointers of type `daddr_t` (see *types*(5)). A zero pointer indicates a 'hole' where no data has been written. Holes read as if they contained all zeros.

A symbolic link is, aside from mode, a plain file whose sole content is the name of the file linked to.

SEE ALSO

chuck(8), *fsck*(8), *icheck*(8), *dir*(5), *mount*(8), *stat*(2), *types*(5), *l3tol*(3)

NAME

font − description files for troff

DESCRIPTION

Directories `/usr/lib/font/dev`*dest* describe typesetters, where *dest* is as in the `-T` option of *troff*(1). Such directories contain files named as in FILES below.

Lines of a typesetter description in file `DESC` have the following forms.

`res` *n*	Resolution of device is *n* basic units per inch.
`hor` *n*	Horizontal motion occurs in increments of *n* units.
`vert` *n*	Vertical motion occurs in increments of *n* units.
`unitwidth` *n*	Widths are given for pointsize *n.*
`sizescale` *n*	Scaling for fractional pointsizes, not used.
`paperwidth` *n*	Width of paper is *n* units.
`paperlength` *n*	Length of paper is *n* units.
`biggestfont` *n*	Maximum number of characters in a font is *n.*
`sizes` *n n n* ... `0`	Pointsizes *n* ... are available.
`fonts` *n name* ...	Number of initial fonts followed by their names, for example `fonts 4 R I B S`
`charset`	This line comes last, followed by a list of special character names for the device, separated by spaces or newlines, as `bu` for `\(bu`

Lines of a font description file have the following forms.

`name` *name*	name of the font, such as `R` or `CW`
`internalname` *name*	The typesetter's name for the font, independent of the *troff* name or font position.
`special`	A *troff* special font, logically part of all non-special fonts.
`ligatures` *name* ... `0`	The named ligatures are available. Legal names are `ff fi fl ffi ffl`.
`spacewidth` *n*	Space is *n* units wide (default 1/3 of an em).
`charset`	Must come last. Each line following `charset` describes one character thus:

name width height code

Name is either a single ASCII character or a special character listed in `DESC`. *Width* is in basic units. *Height* is 1 if the character descends below the baseline, 2 if it rises above the letter 'a', 3 if it both rises and descends, 0 for neither. *Code* is the number sent to the typesetter to produce the character. If a character name is a synonym for the preceding one, its width, height, and code may be replaced by one double quote `"`.

Lines beginning with `#` are comments in both `DESC` and font description files.

Troff and its postprocessors use the binary versions as compiled by a program *makedev*.

FILES

`/usr/lib/font/dev*`	typesetter description directory
`DESC`	typesetter description (ASCII)
`DESC.out`	typesetter description (binary); created by *makedev*
font	description of the named *font* (ASCII)
font`.out`	description of the named *font* (binary); created by *makedev*

`/n/bowell/usr/src/cmd/troff/makedev`

SEE ALSO

troff(1)

B. W. Kernighan, 'A Typesetter-Independent Troff', this manual, Volume 2

NAME

fstab, mtab − information about file systems

SYNOPSIS

```
#include <fstab.h>
```

DESCRIPTION

The file `/etc/fstab` describes the normal configuration of file systems. It guides the default operation of *mount*, *umount*, *swapon*, and *fsck*(8). The order of records in `/etc/fstab` is important.

Each line of the file describes one file system. Fields separated by colons specify

pathname of block device or other mounted object
pathname of mount point
file system type number
integer mount flags
pass number for checking; see *fsck*(8)

File system type numbers and flags are listed in *fmount*(2).

Two special non-numeric file system types signify things that aren't file systems: `xx` causes the line to be ignored, `sw` signifies a swap device.

Use *getfsent*(3) to read data from `/etc/fstab`.

The file `/etc/mtab` lists file systems currently mounted. Each entry is a structure of the form

```
#define FSNMLG 32

struct mtab {
        char file[FSNMLG];      mount point
        char spec[FSNMLG-1];    mounted object
        char type;              file system type
};
```

EXAMPLES

A simple *fstab.*

```
/dev/ra00:/:0:0:1
/dev/ra02:/usr:0:0:2
/dev/ra05:/tmp:0:0:3
/dev/ra10:/ra10:0:1:1
/dev/ra11::sw:0:0
/dev/ra15:/ra15:0:1:3
/dev/null:/proc:2:0:0
```

FILES

```
/etc/fstab
/etc/mtab
```

SEE ALSO

fmount(2), *getfsent*(3), *mount*(8)

BUGS

Swap areas are not file systems, and should not be described in *fstab*.

For compatibility with old programs and habits, two deprecated magic file system types survive: `rw` means 'type 0, flag 0' (a disk file system, mounted for reading and writing); `ro` means 'type 0, flag 1' (a disk file system, mounted read-only).

Only file systems mounted with *mount*(8) are listed in *mtab*.

NAME

map − digitized map formats

DESCRIPTION

Files used by *map*(7) are a sequence of structures of the form:

```
struct {
        signed char patchlatitude;
        signed char patchlongitude;
        short n;
        union {
                struct {
                        short latitude;
                        short longitude;
                } point[n];
                struct {
                        short latitude;
                        short longitude;
                        struct {
                                signed char latdiff;
                                signed char londiff;
                        } point[-n];
                } highres;
        } segment;
};
```

Fields `patchlatitude` and `patchlongitude` tell to what 10-degree by 10-degree patch of the earth's surface a segment belongs. Their values range from −9 to 8 and from −18 to 17, respectively, and indicate the coordinates of the southeast corner of the patch in units of 10 degrees.

Each segment of `|n|` points is connected; consecutive segments are not necessarily related. Latitude and longitude are measured in units of 0.0001 radian. If `n` is negative, then differences to the first and succeeding points are measured in units of 0.00001 radian. Latitude is counted positive to the north and longitude positive to the west.

The patches are ordered lexicographically by `patchlatitude` then `patchlongitude`. A printable index to the first segment of each patch in a file named *data* is kept in an associated file named *data*`.x`. Each line of an index file contains `patchlatitude, patchlongitude` and the byte position of the patch in the map file. Both the map file and the index file are ordered by patch latitude and longitude.

Shorts are stored in little-endian order, low byte first, regardless of computer architecture. To assure portability, *map* accesses them bytewise.

SEE ALSO

map(7), *proj*(3)

NAME

netnews − usenet news articles, utility files

DESCRIPTION

There are two formats of news articles: A and B. Format A is the only format that the older *netnews*(A) understands. *Readnews* and *postnews*(7) deal with both formats, but produce B by default.

Format A looks like this:

A*article-ID*
newsgroups
path
date
title
body of article

Format B contains two extra pieces of information: receipt date and expiration date. A file in B format consists of a series of headers and then the body. A header is a line with a capital letter in the 1st column and a colon somewhere on the line. Unrecognized header fields are ignored. News is stored in whichever format it was created. The following fields are among those recognized:

`From:`
`Newsgroups:`
`Subject:`
`Date:`
`Date-Received:`
`Expires:`
`Reply-To:`
`References:` ID of article this is a follow-up to
`Control:` Text of a control message

Each line of the control file `/usr/lib/news/sys` file line has four fields, separated by colons:

system-name : *subscriptions* : *flags* : *transmission command*

Only the *system-name* and *subscriptions* need to be present.

The *system name* is the name of the system being sent to. The *subscriptions* are the newsgroups it gets. The `flags` are a set of letters describing how the article should be transmitted. The default is `B`. Valid flags include `A`, `B`, `N` (use ihave/sendme protocol), `U` (use `uux -c` and the name of the stored article in a `%s` string).

The *transmission command* is executed by the shell with the article to be transmitted as the standard input. The default is `uux - -z -r` *sysname*`!rnews`.

Somewhere in the control file, there must be a line for the host system. This line has no *flags* or *transmission commands.* A `#` as the first character in a line denotes a comment.

FILES

`/usr/lib/news/*`
`/usr/spool/news/*`

SEE ALSO

postnews(7), *readnews*(7)
M. Horton, *Standard for the Interchange of USENET Messages*, RFC850, DARPA Information Processing Techniques Office, Arlington VA, 1983

NAME

passwd, group − password and group files

DESCRIPTION

The file `/etc/passwd` has one line for each user with the following information:

name (login name, contains no upper case)
encrypted password
numerical user ID
numerical group ID
comp center account number, box number, optional user-id
initial working directory
program to use as shell

Fields are separated by colons. The comp center field is used only when communicating with certain systems, and in other installations can contain any desired information. If the password field is null, no password is demanded; if the shell field is null, `/bin/sh` is used.

The file `/etc/group` defines the membership of users in permission groups. It contains one line for each group with the following colon-separated fields:

group name
encrypted password
numerical group ID
comma-separated list of all users allowed in the group

Each group is separated from the next by a new-line. If the password field is null, *newgrp*(1) will not demand a password.

Because the passwords are encrypted, these files can and do have general read permission and can be used, for example, to map numerical ids to names.

FILES

`/etc/passwd` `/etc/group`

SEE ALSO

newgrp(1), *getpwent*(3), *login*(8), *crypt*(3), *passwd*(1)

BUGS

Passwd(1) won't change passwords in the group file.

NAME

picfile − raster graphic image format

DESCRIPTION

Files in this format store images represented as two-dimensional arrays of multiple-channel pixels. A *picfile* consists of an ASCII header followed by binary data encoding the pixels in row-major order. The header is a list of attribute/value pairs separated by newlines, terminated by an empty line. Each header line has the form *name*=*value*. The name may not contain an ASCII NUL, newline or =; the value may not contain null or newline. The last line of a header is empty.

The standard attributes are described below; all but `TYPE` and `WINDOW` are optional. `TYPE` must come first; otherwise order is irrelevant. As any unrecognised attribute is passed over uninterpreted by all standard software, applications are welcome to include arbitrary annotations, like `SHOESIZE=10`, if they wish.

`TYPE=`*type*

How the pixels are encoded. Standard types are

`runcode` A run-length encoding. The data are a sequence of (*nchan*+1)-byte records each containing a count *k* and *nchan* bytes giving a pixel value to be repeated *k*+1 times. A run may not span scanlines.

`dump` A two-dimensional array of *nchan*-byte records in row major order.

`bitmap` One-bit pixels, packed into bytes high bit leftmost. Zero bits are white, one bits are black. Rows are padded with zeros to a multiple of 16 bits.

`ccitt-g4` A black-and-white image under CCITT FAX Group 4 compression. This format is highly compressive on images of text and line art. Similarly, `ccitt-g31` and `ccitt-g32` for Group 3, 1-D and 2-D.

`pico` A sequence of *nchan* two-dimensional arrays of single bytes.

`ccir601` Pixels are in dump order, 2 bytes per pixel encoded according to the IEEE digital component video standard.

`WINDOW=`*x0 y0 x1 y1*

The *x*,*y* coordinates of the upper left corner and the point just diagonally outside the lower right corner, *x* increasing to the right, *y* down.

`NCHAN=`*nchan*

The number of channels, default 1.

`CHAN=`*value*

The order of channels.

`RES=`*x y*

The digitizing resolution horizontally and vertically, in pixels/inch.

`CMAP=` (The value is empty.) A color map, a 256×3-byte translation table for color values, follows the header. In a full-color picture, each color-map row maps pixel values of the corresponding channel. In a monochrome picture, pixel values index the color map to yield red, green and blue, like this:

```
unsigned char cmap[256][3];
red=cmap[pixel][0];
green=cmap[pixel][1];
blue=cmap[pixel][2];
```

EXAMPLES

`sed '/^$/q' image`

Print a header. A sample header follows.

```
TYPE=dump
WINDOW=0 0 512 512
NCHAN=1
CHAN=m
RES=300 300
```

```
CMAP=
COMMAND= antiquantize 'halftone CLASSIC' 512.halftone LIBERTY.anticlassic
COMMAND=  halftone CLASSIC 512.liberty 512.halftone 1.75 512.halftone
COMMAND=   transpose IN OUT
COMMAND=    resample 512 IN OUT
COMMAND=     transpose IN OUT
COMMAND=      resample 512 IN OUT
COMMAND=       clip 400 400 LIBERTY OUT
```

SEE ALSO

bcp(1), *cscan*(1), *imscan*(1), *pico*(1), *flicks*(9.1), *mugs* in *face*(9.7), *rebecca*(9.1), *flickfile*(9.5)
T. Duff, 'The 10th Edition Raster Graphics System', this manual, Volume 2

NAME

plot − graphics interface

DESCRIPTION

Files of this format are produced by routines described in *plot*(3), and are interpreted for various devices by commands described in *plot*(1). A graphics file is an ASCII stream of instruction lines. Arguments are delimited by spaces, tabs, or commas. Numbers may be floating point. Punctuation marks (except `:`) , spaces, and tabs at the beginning of lines are ignored. Comments run from `:` to newline. Extra letters appended to a valid instruction are ignored. Thus `...line`, `line`, `li` all mean the same thing. Arguments are interpreted as follows:

1. If an instruction requires no arguments, the rest of the line is ignored.
2. If it requires a string argument, then all the line after the first field separator is passed as argument. Quote marks may be used to preserve leading blanks. Strings may include newlines represented as `\n`.
3. Between numeric arguments alphabetic characters and punctuation marks are ignored. Thus `line from 5 6 to 7 8` draws a line from (5, 6) to (7, 8).
4. Instructions with numeric arguments remain in effect until a new instruction is read. Such commands may spill over many lines. Thus the following sequence will draw a polygon with vertices (4.5, 6.77), (5.8, 5.6), (7.8, 4.55), and (10.0, 3.6).

```
move 4.5 6.77
vec 5.8, 5.6 7.8
4.55 10.0, 3.6 4.5, 6.77
```

The instructions are executed in order. The last designated point in a `line`, `move`, `rmove`, `vec`, `rvec`, `arc`, or `point` command becomes the 'current point' (*X*,*Y*) for the next command. Each of the following descriptions corresponds to a routine in *plot*(3).

Open & Close

`o` *string* Open plotting device. For *troff*, *string* specifies the size of the plot (default is `6i`.)

`cl` Close plotting device.

Basic Plotting Commands

`e` Start another frame of output or erase the screen on CRT terminals without scroll.

`m` *x y* (move) Current point becomes *x y*.

`rm` *dx dy* Current point becomes *X*+*dx Y*+*dy*.

`poi` *x y* Plot the point *x y* and make it the current point.

`v` *x y* Draw a vector from the current point to *x y*.

`rv` *dx dy* Draw vector from current point to X + dx Y + dy

`li` *x1 y1 x2 y2*
Draw a line from *x1 y1* to *x2 y2*. Make the current point *x2 y2*.

`t` *string* Place the *string* so that its first character is centered on the current point (default). If *string* begins with `\C` (`\R`), it is centered (right-adjusted) on the current point. A backslash at the beginning of the string may be escaped with another backslash.

`a` *x1 y1 x2 y2 xc yc r*
Draw a circular arc from *x1 y1* to *x2 y2* with center *xc yc* and radius *r*. If the radius is positive, the arc is drawn counterclockwise; negative, clockwise. The starting point is exact but the ending point is approximate.

`ci` *xc yc r* Draw a circle centered at *xc yc* with radius *r*. If the range and frame parameters do not specify a square, the 'circle' will be elliptical.

`di` *xc yc r* Draw a disc centered at *xc yc* with radius *r* using the filling color (see `cfill` below). Only works on the 5620; on other devices is the same as `circle`.

`bo` *x1 y1 x2 y2*
Draw a box with lower left corner at *x1 y1* and upper right corner at *x2 y2*.

`sb` *x1 y1 x2 y2*
Draw a solid box with lower left corner at *x1 y1* and upper right corner at *x2 y2* using the filling color (see `cfill` below).

`par` *x1 y1 x2 y2 xg yg*
Draw a parabola from *x1 y1* to *x2 y2* 'guided' by *xg yg*. The parabola passes through the midpoint of the line joining *xg yg* with the midpoint of the line joining *x1 y1* and *x2 y2* and is tangent to the lines from *xg yg* to the endpoints.

`pol { {`*x1 y1 ... xn yn*`} ... {`*X1 Y1 ... Xm Ym*`} }`
Draw polygons with vertices *x1 y1 ... xn yn* and *X1 Y1 ... Xm Ym*. If only one polygon is specified, the inner brackets are not needed.

`fi { {`*x1 y1 ... xn yn*`} ... {`*X1 Y1 ... Xm Ym*`} }`
Fill a polygon. The arguments are the same as those for `pol` except that the first vertex is automatically repeated to close each polygon. The polygons do not have to be connected. Enclosed polygons appear as holes.

`sp { {`*x1 y1 ... xn yn*`} ... {`*X1 Y1 ... Xm Ym*`} }`
Draw a parabolic spline guided by *x1 y1 ... xn yn* with simple endpoints.

`fsp { {`*x1 y1 ... xn yn*`} ... {`*X1 Y1 ... Xm Ym*`} }`
Draw a parabolic spline guided by *x1 y1 ... xn yn* with double first endpoint.

`lsp { {`*x1 y1 ... xn yn*`} ... {`*X1 Y1 ... Xm Ym*`} }`
Draw a parabolic spline guided by *x1 y1 ... xn yn* with double last endpoint.

`dsp { {`*x1 y1 ... xn yn*`} ... {`*X1 Y1 ... Xm Ym*`} }`
Draw a parabolic spline guided by *x1 y1 ... xn yn* with double endpoints.

`csp { {`*x1 y1 ... xn yn*`} ... {`*X1 Y1 ... Xm Ym*`} }`

`in` *filename*
(include) Take commands from *filename*.

`de` *string* `{` *commands* `}`
Define *string* as *commands*.

`ca` *string scale*
Invoke commands defined as *string* applying *scale* to all coordinates.

Commands Controlling the Environment

`co` *string* Draw lines with color *string*. Available colors depend on the device. *String* may contain definitions for several devices separated by `/`. Colors possible for the various devices are:

pen	`black`, `red`, `green`, `blue`, `Tblack`, `Tred`, `Tgreen`, `Tblue` (assumes default carousel, `T`=thick)
	`1-8` (pen number)
	`S`*number* character size as % of plotting area
troff	`F`*font*
	`P`*point size*
2621	`H`*character* for plotting

`pe` *string* Use *string* as the style for drawing lines. Not all pen styles are implemented for all devices. *String* may contain definitions for several devices separated by `/`. The available pen styles are:

pen	`solid`, `dott`[ed], `short`, `long`, `dotd`[ashed], `cdash`, `ddash`
4014	solid , `dott`[ed], `short`, `long`, `dotd`[ashed], `ddash`
troff	`solid`, `dash` only straight lines will be dashed
5620	`B`*number* line thickness
2621	`H`*character* for plotting

`cf` *string* Color for filling; may contain the definitions for several devices. separated by `/`. The following colors are available on the specified devices:

pen	`black`, `red`, `green`, `blue`, `Tblack`, `Tred`, `Tgreen`, `Tblue`
	`1-8` pen number

5620 `B`*texture* string with octal numbers for texture; see *types*(9.5). The 16 words of texture should be followed by one word for the mode used by *texture()*; see *bitblt*(9.3).
2621 `H`*character* for filling
All `/A`*degrees* slant of shading lines
`/G`*number* gap between shading lines (in user units)

`ra` *x1 y1 x2 y2*

The data will fall between *x1 y1* and *x2 y2*. The plot will be magnified or reduced to fit the device as closely as possible.

Range settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *plot*(1). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices with nonsquare faces.

4014 `range 0. 0. 3120. 3120.`
troff `range 0. 0. 6144. 6144.`
2621 `range 0. 0. 22. 22.`
5620 range dependent on layer size
pen range dependent on paper size

`fr` *px1 py1 px2 py2*

Plot the data in the fraction of the display specified by *px1 py1* for lower left corner and *px2 py2* for upper right corner. Thus `frame .5 0 1. .5` plots in the lower right quadrant of the display; `frame 0. 1. 1. 0.` uses the whole display but inverts the *y* coordinates.

`sa` Save the current environment, and move to a new one. The new environment inherits the old one. There are 7 levels.

`re` Restore previous environment.

SEE ALSO

plot(1), *plot*(3), *graph*(1)

NAME

poly − polyhedron database

DESCRIPTION

The directory `/usr/lib/polyhedra` contains an index file and many polyhedron description files, each describing a solid polyhedron and its (not necessarily unique) planar net. Each line of the index file consists of a polyhedron's number followed by a horizontal tab and the polyhedron's name. The polyhedron's number is also the name of its description file. The routines of *poly*(3) read such description files.

Each description file consists of a number of fields. Each field begins with a line consisting of `:` and the field name. The field continues until the next header line or end of file. Some fields contain *values*, which consist of a floating point number optionally followed by an algebraic expression enclosed in `[]`, or *angles*, which are a *value* in radians and optionally two more values (sin and cos) separated by `@`. The fields include, but are not limited to,

`number` The polyhedron's number.

`name` The polyhedron's name, less than 128 characters long and not capitalized.

`symbol` The Schlaefli symbol, a tab, and the Johnson symbol for the polyhedron, given in *eqn*(1) style with delimiters `@@`.

`dual` The name of the dual polyhedron optionally followed by a horizontal tab and the number of the dual.

`vertices` The first line is the number of vertices, which follow, one per line. Each vertex has a coordinate (three *values* separated by spaces), a number n, and n face,edge pairs that surround the vertex.

`faces` The first line contains the number of faces and the maximum number of vertices in any face. The remaining lines are the faces, each with a vertex count n, followed by $2n$ vertex numbers (planar, solid), $2n$ edge numbers (planar, solid), and n angles. The vertices of each face are listed in the same order for both the planar and solid forms: counterclockwise as viewed from above the planar net (i.e. from $z>0$) which generally corresponds to outside the solid polyhedron.

`edges` The first line contains the number of edges. Each edge is one line: *face1 vertex1 face2 vertex2 length angle*. The length is a *value*.

`summary` The three lines summarise the different kinds of faces, vertices and edges respectively. Each line consists of a total and a list of *count example symbolic* triples; where *example* is an index into the appropriate list and *symbolic* is given in *eqn*(1) style with delimiters `@@`.

`EOF` The end of the polyhedron's description. (Another polyhedron description may follow in this file.)

An expression in a *value* gives the exact value in the syntax of *bc*(1) using these functions: $a(x) = \tan^{-1}(x)$, $b(x) = (x)^{1/3}$, $c(x) = \cos(x)$, $d(x) = \tan(x)$, $p = \pi$, $q(x) = x^2$, $r(x) = \cos^{-1}(x)$, $s(x) = \sin(x)$, $t = \phi = (1+\sqrt{5})/2$. The code may include assignments but does not include white space.

FILES

`/usr/lib/polyhedra/index` index file
`/usr/lib/polyhedra/[0-9]*` description files

SEE ALSO

poly(3), *poly*(7)

NAME

stab − symbol table types

SYNOPSIS

```
#include <stab.h>
```

DESCRIPTION

The include file `<stab.h>` defines some values of the `n_type` field of the symbol table of object files; see *a.out*(5). These are the types for permanent symbols used by the compilers *cc*(1) and *f77*(1) and the debugger *pi*(9.1). Symbol table entries are produced by assembler directives:

`.stabs` specifies a name in quotes `" "`, a symbol type (`n_type`), one char (`n_other`), one short (`n_desc`), and an unsigned long (`n_value`, usually an address).
`.stabd` the same, referring to the current location without an explicit name.
`.stabn` generates entries with no name.

The loader *ld*(1) preserves the order of symbol table entries produced by these directives.

The low bits of the `n_type` field place a symbol into at most one segment, according to the following masks, defined in `<a.out.h>`.

```
#define N_UNDF  0x0   /* undefined */
#define N_ABS   0x2   /* absolute */
#define N_TEXT  0x4   /* text */
#define N_DATA  0x6   /* data */
#define N_BSS   0x8   /* bss */
#define N_EXT   0x1   /* external bit, or'ed in */
```

The `n_value` field of a symbol is relocated by *ld* as an address within the appropriate segment, or is unchanged for a symbol not in any segment. In addition, the loader will discard certain symbols, according to rules of its own, unless the `n_type` field has one of the following bits set:

```
#define N_STAB  0xe0
```

This allows up to 112 symbol types, split among the various segments. Some of these have already been claimed. Option `-g` of *cc* uses the following values, all 4 mod 16, for text symbols. Comments show the pertinent fields of the `.stabs` directive.

```
#define N_BFUN  0x24  /* procedure: name,,0,lineno,address */
#define N_FUN   0x24
#define N_NARGS 0x34  /* function call: ,,0,nbytes,address */
#define N_SLINE 0x44  /* src line: ,,0,lineno,address */
#define N_SO    0x64  /* source file: name,,0,lineno,address */
#define N_SOL   0x84  /* #include file: name,,0,lineno,address */
#define N_ESO   0x94  /* end source file: name,,0,lineno,address */
#define N_ENTRY 0xa4  /* alternate entry: name,,0,lineno,address */
#define N_RFUN  0xb4  /* return from function: ,,0,lineno,address */
#define N_LBRAC 0xc4  /* left  bracket: ,,0,level,address */
#define N_RBRAC 0xd4  /* right bracket: ,,0,level,address */
#define N_EFUN  0xf4  /* end of function: name,,0,lineno,address */
```

These values, all 8 mod 16, are used for data symbols:

```
#define N_LCSYM 0x28  /* .lcomm symbol: name,,0,type,address */
#define N_ECOML 0xe8  /* end common (local name): ,,address */
```

And these for non-relocated symbols:

```
#define N_GSYM  0x20  /* global symbol: name,,0,type,0 */
#define N_FNAME 0x22  /* procedure name (f77 kludge): name,,0 */
#define N_STFUN 0x32  /* static function: name,,0,type,0 */
#define N_RSYM  0x40  /* register sym: name,,0,type,register */
```

```
#define N_BSTR  0x5c  /* begin structure: name,,0,type,length */
#define N_ESTR  0x5e  /* end structure: name,,0,type,length */
#define N_SSYM  0x60  /* structure elt: name,,0,type,offset */
#define N_SFLD  0x70  /* structure field: name,,0,type,offset */
#define N_LSYM  0x80  /* local sym: name,,0,type,offset */
#define N_PSYM  0xa0  /* parameter: name,,0,type,offset */
#define N_BCOMM 0xe2  /* begin common: name,, */
#define N_ECOMM 0xe4  /* end common: name,, */
#define N_VER   0xf0  /* symbol table version number */
#define N_TYID  0xfa  /* struct, union, or enum name */
#define N_DIM   0xfc  /* dimension for arrays */
```

Field `n_desc` holds a type specifier in the form used by *cc*(1), by up to 6 qualifiers, with `q1` most significant:

```
struct  desc   {
        short  q6:2, q5:2, q4:2, q3:2, q2:2, q1:2;
        short  basic:5;
};
```

The qualifiers are coded thus:

0	none	2	function
1	pointer	3	array

The basic types are coded thus:

0	undefined	9	union
1	function argument	10	enumeration
2	character	11	member of enumeration
3	short	12	unsigned character
4	int	13	unsigned short
5	long	14	unsigned int
6	float	15	unsigned long
7	double	16	void
8	structure		

The Pascal compiler, *pc*(A), uses the following `n_type` value:

```
#define N_PC   0x30/* global pascal symbol: name,,0,subtype,line */
```

and uses the following subtypes to do type checking across separately compiled files:

1	source file name	6	global variable
2	included file name	7	global function
3	global label	8	global procedure
4	global constant	9	external function
5	global type	10	external procedure

SEE ALSO

a.out(5), *pi*(9.1), *as*(1), *ld*(1)

BUGS

The loader's relocation conventions limit the number of useful `n_type` values.

NAME

termcap − terminal capability file

DESCRIPTION

Termcap describes terminals as used, for example, by *vi*(1) and *curses*(3) or in the `TERMCAP` environment variable. A *termcap* entry is a line containing fields separated by `:`. Lines may be broken; `\` at the end of a line signifies continuation. Empty fields are ignored.

The first field for each entry gives names for a terminal separated by `|`. The first name is conventionally two characters long for the benefit of older systems; the second name is the customary abbreviation; and the last name fully identifies the terminal.

There are three types of capability: Boolean for the presence of a feature, numeric for sizes and time delays, and strings for performing operations. Some string fields may be preceded by a number, which specifies padding, a time delay required with the operation. These capabilities are marked 'P' or 'P*' below. Padding is measured in milliseconds; P* signifies that the padding is proportional to the number of lines (or characters) affected, for example, `3.5*` specifies 3.5 milliseconds per unit.

Name	Type	Pad	Description
`ae`	str	P	End alternate character set
`al`	str	P*	Add new blank line
`am`	bool		Automatic margin
`as`	str	P	Start alternate character set
`bc`	str		Backspace char if not `^H`
`bs`	bool		Terminal can backspace
`bt`	str	P	Back tab
`bw`	bool		Backspace wraps from column 0 to last column
`CC`	str		Command character in prototype if terminal settable
`cd`	str	P*	Clear to end of display
`ce`	str	P	Clear to end of line
`ch`	str	P	Like `cm` but horizontal motion only
`cl`	str	P*	Clear screen
`cm`	str	P	Cursor motion
`co`	num		Number of columns
`cr`	str	P*	Carriage return, default `^M`
`cs`	str	P	Change scrolling region (vt100), like cm
`cv`	str	P	Like ch, but vertical only
`da`	bool		Display may be retained above
`dB`	num		Backspace delay
`db`	bool		Display may be retained below
`dC`	num		Carriage return delay
`dc`	str	P*	Delete character
`dF`	num		Form feed delay
`dl`	str	P*	Delete line
`dm`	str		Enter delete mode
`dN`	num		Newline delay
`do`	str		Down one line
`dT`	num		tab delay
`ed`	str		End delete mode
`ei`	str		End insert mode; give `:ei=:` if `ic`
`eo`	str		Can erase overstrikes with blank
`ff`	str	P*	Hard copy page eject, default `^L`
`hc`	bool		Hardcopy terminal
`hd`	str		Half line down
`ho`	str		Home cursor if no `cm`
`hu`	str		Half line up

hz	str		Hazeltine, can't print ~
ic	str	P	Insert character
if	str		Name of file containing initialization is
im	bool		Enter insert mode; give :im=: if ic
in	bool		Insert mode distinguishes nulls on display
ip	str	P*	Insert pad after character insert
is	str		Terminal initialization string
k0-k9	str		Other function key codes
kb	str		Backspace key code
kd	str		Down arrow key code
ke	str		Leave keypad transmit mode
kh	str		Home key code
kl	str		Left arrow key code
kn	num		Number of function keys
ko	str		Termcap entries for other non-function keys
kr	str		Right arrow key code
ks	str		Enter keypad transmit mode
ku	str		Up arrow key code
l0-9	str		Labels on other function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column, if no cm
ma	str		Arrow key map
mi	bool		Safe to move in insert mode
ml	str		Memory lock above cursor
ms	bool		Safe to move in standout or underline mode
mu	str		Turn off memory lock
nc	bool		No correctly working CR (DM2500, H2000)
nd	str		Nondestructive space (cursor right)
nl	str	P*	Newline character, default \n
ns	bool		Nonscrolling CRT
os	bool		Terminal overstrikes
pc	str		Pad character, default NUL
pt	bool		Has hardware tabs (possibly set by is)
se	str		Leave standout mode
sf	str	P	Scroll forward
sg	num		Number of blanks left by so, se
so	str		Enter standout mode
sr	str	P	Scroll reverse (backward)
ta	str	P	Tab, if not ^I or if padded
tc	str		Entry of similar terminal, must be last
te	str		String to end programs that use cm
ti	str		String to begin programs that use cm
uc	str		Underscore one char and move past it
ue	str		Leave underscore mode
ug	num		Number of blanks left by us, ue
ul	bool		Terminal underlines but doesn't overstrike
up	str		Cursor up one line
us	str		Enter underscore mode
vb	str		Visible bell, (may not move cursor)
ve	str		Leave open/visual mode
vs	str		Enter open/visual mode
xb	bool		Beehive (f1=escape, f2=^C)
xn	bool		Newline ignored after wrap (Concept)

`xr`	bool	Return acts like `ce \r \n` (Delta Data)
`xs`	bool	Standout not erased by writing over (HP264?)
`xt`	bool	Tabs are destructive, magic `so` (Teleray 1061)

The following example is one of the more elaborate *termcap* entries. (Do not believe it; see the file for current facts.)

```
co|c100|concept 100:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E:\
      :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:cm=\Ea%+ %+ :co#80:\
      :dc=16\E^A:dl=3*\E^B:ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:\
      :nd=\E=:se=\Ed\E:so=\ED\EE:ta=8\t:ul:up=\E;vb=\Ek\Ek:xn:
```

Among the Boolean capabilities shown for the Concept are automatic margins `am`: automatic return and linefeed at the end of a line. Numeric capabilities are indicated by `#`; `co#80` means the Concept has 80 columns. String capabilities are indicated by `=`; to clear to end of line (`ce`) on the Concept, issue <escape> <control-C> and pad with 16 milliseconds delay.

In strings the ASCII ESC character is represented by `\E` and control characters are represented by `^`*c*, where character *c* has ASCII code 0100 greater than the desired control character. Newline, return, tab, backspace, form feed, `\` and `^` are represented by `\n \r \t \b \f \\ \^`. Backslash `\` followed by 3 digits specifies a byte in octal. A null character is encoded `\200`: the routines that use *termcap* information mask out the high bit of all bytes.

Local cursor motions are undefined if they run off the left or top of the screen; the *curses* routines refrain from issuing such motions. It is assumed that the screen will scroll up upon running off the bottom; this assumption is negated by `ns`. Capability `am` (automatic margin) describes the handling of the right margin.

Cursor addressing is described by capability `cm`, which contains *printf*(3)-like format codes for line and column positions. The leftmost column is column 0.

`%d`	as in *printf*
`%2`	like `%2d`
`%3`	like `%3d`
`%.`	like `%c`
`%+`*x*	adds *x* before converting
`%>`*xy*	if value exceeds *x*, add *y*, no output
`%r`	reverse order of line and column, no output
`%i`	increment line/column (1-origin)
`%%`	single `%`
`%n`	exclusive or row and column with 0140 (DM2500)
`%B`	BCD: 16∗(*x/10)* + *(x*%10), no output
`%D`	Reverse coding: *x*−2∗(*x*%16), no output

For example, to go to line 3 column 12, a HP2645 terminal must get `\E&a12c03Y` padded for 6 milliseconds: `:cm=6\E&%r%2c%2Y: .`

Capability `al` adds an empty line before the line where the cursor is and leaves the cursor on the new line. This will always be done with the cursor at column 0. Capability `dl` deletes the line where the cursor is and is also done with the cursor at column 0. Capabilities `da` and `db` warn that off-screen lines may appear at the top or bottom of the screen upon scrolling or deleting lines. The *curses* routines do not use this feature, but do guard against its effects.

Insert-character operations usually affect only the current line and shift characters off the end of the line rigidly. Some terminals, such as the Concept 100, distinguish typed from untyped blanks on the screen, shifting upon insertion only up to an untyped blank, i.e. a space caused by cursor motion; these terminals have capability `in` (insert null).

Some terminals have an insertion mode; others require a special sequence to open up a blank position on the current line. Insertion mode is entered and left by `im` and `ie`, which should be null strings if there

is no insertion mode. String `ic` is sent just before each character to be inserted, and padding `ip` is sent after. Capability `mi` says it is possible to move around without leaving insertion mode. Delete mode works similarly: enter with `dm`, leave with `de`, and issue `dc` before each character.

Highlighting, or 'standout' mode is entered by `so` and left by `se`. Underline mode is entered by `us` and left by `ue`. Terminals that underline characters individually have capability `uc`. The visual bell capability `vb` flashes the screen without moving the cursor.

A terminal with a keypad that transmits cursor motions may be described by capabilities `kl kr ku kd kh` that give the codes for left, right, up, down, and home. Up to ten function keys may be described by `k0` through `k9`. Special labels for the function keys may be given as `l0` through `l9`.

The initialization string `is` is expected to set tabs if that is necessary. That string may come from a file (`if`); if both are present `is` is done first.

The entry for a terminal may be continued by jumping to another entry given by `tc`. Duplicate capabilities are resolved in favor of the first.

FILES

`/etc/termcap`

SEE ALSO

curses(3), *termcap*(3), *vi*(1), *ul*(1)

BUGS

Termcap entries, including `tc` continuations, are limited to 1024 characters.

NAME

ttys − terminal initialization data

DESCRIPTION

The file `/etc/ttys` directs *init*(8) in associating login processes with terminal ports. It contains one line per port.

If the first character of a line is `0` the line will be ignored; if it is `1` the line will be effective. The second character is used as an argument to *getty*(8), which performs such tasks as baud-rate recognition, reading the login name, and calling *login*(8). For normal lines, the character is `0`. Other characters can be used, for example, with hard-wired terminals where speed recognition is unnecessary or which have special characteristics; see *getty*(8) for a list. The remainder of the line is the terminal's entry in the device directory, `/dev`.

FILES

`/etc/ttys`

SEE ALSO

init(8), *getty*(8), *login*(8)

NAME

types − primitive system data types

SYNOPSIS

```
#include <sys/types.h>
```

DESCRIPTION

The data types defined in the include file are used in the operating system. Some data of these types are useful in user code.

`typedef long`	`daddr_t;`	disk block number, see *filsys*(5)
`typedef char *`	`caddr_t;`	general memory pointer
`typedef unsigned short`	`ino_t;`	inode number, *filsys*(5)
`typedef long`	`size_t;`	file size, *stat*(2)
`typedef long`	`time_t;`	time, *time*(2)
`typedef unsigned short`	`dev_t;`	device code, *stat*(2)
`typedef long`	`off_t;`	file offset, *lseek*(2)

The following macros analyze and synthesize device numbers; see *intro*(4).

```
#define major(x)       ((int)(((unsigned)(x)>>8)&0377))
#define minor(x)       ((int)((x)&0377))
#define makedev(x,y)   ((dev_t)(((x)<<8) | (y)))
```

The file contains other definitions as well, internal to the system or specific to particular system calls. Pages in section 2 tell which calls need `<sys/types.h>`.

SEE ALSO

filsys(5), *time*(2), *intro*(4)

NAME

utmp, wtmp − login records

SYNOPSIS

```
#include <utmp.h>
```

DESCRIPTION

The *utmp* file allows one to discover information about who is currently logged in. The file is a sequence of entries with the following structure declared in the include file:

```
struct utmp {
      char  ut_line[8];  /* tty name */
      char  ut_name[8];  /* user id */
      long  ut_time;     /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of *time*(2).

The *wtmp* file records logins and logouts. Its format is exactly like *utmp* except that a null user name indicates a logout on the associated terminal. Furthermore, the terminal name ~ indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names | and } indicate the system-maintained time just before and just after a *date*(1) command changed the system's idea of the time.

Wtmp is maintained by *login*(8) and *init*(8). Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by *ac*(8).

FILES

```
/etc/utmp
/usr/adm/wtmp
```

SEE ALSO

login(8), *init*(8), *who*(1), *ac*(8)

NAME

view2d − movie of a function f(x, y, t)

DESCRIPTION

Files of this format are produced by functions in *view2d*(3), and displayed by commands in *view2d*(1). A movie file consists of one or more frames, each consisting of a header and a sequence of 16-bit signed integer values for each pixel, scanned left to right and bottom to top. (Left-to-right is the inner loop.)

The header consists of the 32-bit magic number 0135246, then eight 16-bit integers:

VER The version number.

NX, NY The number of pixels in the frame. These may not vary from frame to frame.

u, v relate pixel values *p* in the file to user function values *f* by

$p = u + f \times 2^{-v}$.

FIXUV normally 0; 1 if *u, v, PMIN, PMAX* of first frame give a bound on the data in the entire file.

PMIN, PMAX
limits of the data; only used when *FIXUV*=1.

and finally a 16-byte ASCII representation of a floating point value:

TIME is a frame index, typically set to simulated time or to an iteration counter. This need not be uniformly spaced from frame to frame, but should be nondecreasing.

The range of displayable pixel values is [−32765,32765]. Values below this range are deemed out of bounds and not plotted; values above are reserved.

SEE ALSO

view2d(1), *view2d*(3)

NAME

whoami − computer name

DESCRIPTION

The file `/etc/whoami` contains one line of information − the name of the computer, as used in *mail*(1) and *uucp*(1).

FILES

`/etc/whoami`

NAME

worm − format of worm disks

SYNOPSIS

```
#include <worm.h>
```

DESCRIPTION

A WORM disk is a linked list of ‘superblocks’, roughly one for every *worm write* on the WORM. The ‘governing’ superblock is at block zero if it exists and has a valid magic number; otherwise the governing superblock is the last superblock in the linked list starting at block 1. (The link to the next superblock is preallocated and thus the last superblock in the list will be unwritten.) Each superblock has some status information and a pointer to a set of ‘inodes’ describing a set of files. The status information for the WORM is that of the governing superblock; the set of files on the WORM is the accumulation of all the superblocks taken in order. The structure of a superblock as given in the include file is:

```
#define SMAGIC   0x21746967
#define VLINK    1           /* linked list superblock */
#define VBTREE   2           /* cbt superblock */
typedef struct superblock
{
        long magic;          /* magic number for superblock */
        unsigned short blocksize; /* physical size of blocks */
        short version;       /* type of superblock */
        long nblocks;        /* number of blocks on device */
        long zero;           /* first logical data block */
        long nfree;          /* number of free blocks */
        long ninodes;        /* number of inodes */
        long ninochars;      /* number of bytes of inode names */
        long binodes;        /* start of inodes */
        long nextffree;      /* next free file block */
        long nextsb;         /* next superblock */
        short fd;            /* fildes for device (in core) */
        char vol_id[128];    /* name the disk can be mounted as */
        char comment[128];   /* comments */
        long myblock;        /* where this superblock is */
        long nF;             /* bytes for .F (VBTREE) */
        long nT;             /* bytes for .T (VBTREE) */
        long ctime;          /* create time for this superblock */
} superblock;
```

superblocks are padded with zeros to `blocksize`.

Following each `VLINK` superblock is a set of inodes, a string table, and then the data blocks for the files described by the inodes. Following a (there is at most one) `VBTREE` superblock there is a set of inodes, a string table and the `.F` and `.T` files for a *cbt*(1) database where the keys are filenames and the data is an inode number.

```
#define DMAGIC   0x3A746967
typedef struct Inode
{
        long magic;          /* magic number for Dirent */
        long block;          /* starting block of file */
        long nbytes;         /* bytes in file */
        long ctime;          /* creation time */
        union {
                 char *n;    /* core - name */
                 long o;     /* disk - offset into chars block */
```

```
        } name;             /* filename */
        long pad1           /* to 32 bytes */
        short mode          /* as in stat(2) */
        short uid           /* owner */
        short gid           /* owner */
        short pad2          /* to 32 bytes */
} Inode
```

If the `block` field of an inode is negative, the file has been deleted.

SEE ALSO

worm(8)

6 LANGUAGE CONVENTIONS

6 LANGUAGE CONVENTIONS

NAME

ascii, latin1 − character set maps

DESCRIPTION

The file `/usr/pub/ascii` is a map of the ASCII character set, to be printed as needed. It contains:

000	nul	001	soh	002	stx	003	etx	004	eot	005	enq	006	ack	007	bel
010	bs	011	ht	012	nl	013	vt	014	np	015	cr	016	so	017	si
020	dle	021	dc1	022	dc2	023	dc3	024	dc4	025	nak	026	syn	027	etb
030	can	031	em	032	sub	033	esc	034	fs	035	gs	036	rs	037	us
040	sp	041	!	042	"	043	#	044	$	045	%	046	&	047	’
050	(	051	)	052	*	053	+	054	,	055	−	056	.	057	/
060	0	061	1	062	2	063	3	064	4	065	5	066	6	067	7
070	8	071	9	072	:	073	;	074	<	075	=	076	>	077	?
100	@	101	A	102	B	103	C	104	D	105	E	106	F	107	G
110	H	111	I	112	J	113	K	114	L	115	M	116	N	117	O
120	P	121	Q	122	R	123	S	124	T	125	U	126	V	127	W
130	X	131	Y	132	Z	133	[	134	\	135	]	136	^	137	_
140	`	141	a	142	b	143	c	144	d	145	e	146	f	147	g
150	h	151	i	152	j	153	k	154	l	155	m	156	n	157	o
160	p	161	q	162	r	163	s	164	t	165	u	166	v	167	w
170	x	171	y	172	z	173	{	174	|	175	}	176	~	177	del

The file `/usr/pub/latin1` is a map of characters 0200-0377 in the ISO Latin-1 extension to the ASCII code. Escape sequences are given for typing the characters in *mux*(9.1).

FILES

`/usr/pub/ascii`
`/usr/pub/latin1`

NAME

eqnchar − special character definitions for eqn

SYNOPSIS

`eqn /usr/pub/eqnchar` [*file* ...] | `troff` [*option* ...]

`neqn /usr/pub/eqnchar` [*file* ...] | `nroff` [*option* ...]

DESCRIPTION

Eqnchar contains *nroff* and *troff*(1) character definitions for constructing characters that are not available on standard fonts. These definitions are primarily intended for use with *neqn* and *eqn*(1). It contains definitions for the following characters.

NAME

font − typesetter fonts

DESCRIPTION

Listed below are some of the more common (or more cryptically named) typesetter fonts.

In the directories `/usr/lib/font` and `/usr/jerq/font` are kept tables for fonts available on some or all of the Autologic APS-5 phototypesetter, the Mergenthaler Linotron 202 phototypesetter, the Canon laser printer, and the Teletype 5620 terminal running under *mux*(9.1). Those directories may be consulted for the current list of offerings.

Most fonts come in families, e.g. `Souvenir`, `SouvenirI` (italic), `SouvenirB` (bold), `SouvenirBI` (bold italic), `SouvenirBK` (demi-bold, 'black') `SouvenirKI` (demi-bold italic). For the 5620 and the Canon there is a separate font table for each point size, e.g. `CW.11`.

In *troff* fonts must be 'mounted' in a 'font position'. The default set is `R I B BI H HB HK CW CH S` for the 202 and the Canon and `R I B H CW S S1 GR` for the APS-5. Other fonts may be mounted with the `.fp` request. Fonts with long names must be given aliases; see the *troff* manual.

`B`	Times bold
`BI`	Times bold italic
`CH`	Chess
`CS`	Constant Width Slanted
`CT`	Courier Typewriter (APS-5)
`CW`	Constant width (ASCII typewriter)
`CX`	News Gothic condensed italic (APS-5)
`GB`	Greek bold (APS-5)
`GR`	Greek
`GS`	German Script (APS-5)
`H`	Helvetica
`HB`	Helvetica bold
`HI`	Helvetica italic
`HK`	Helvetica black
`HX`	Helvetica bold italic
`I`	Times italic
`M1`	Universal [sic] Math 1
`M2`	Universal Math 2
`M3`	Universal Math 3
`OE`	Old English
`OK`	Helvetica outline black
`PA`	Palatino
`PB`	Palatino bold
`PI`	Palatino italic
`PO`	Printout (ASCII constant width)
`PX`	Palatino bold italic
`R`	Times Roman
`S`	Special (math symbols)
`S1`	Special (APS-5)
`SC`	Script (APS-5)
`SM`	Stymie medium (APS-5)
`TB`	Techno bold (APS-5)
`TX`	Techno bold italic (APS-5)
`US`	USA state maps (202)
`X1`	Universal Newspaper Pi (202)
`defont`	Default *mux* font

FILES

`/usr/lib/font`	
`/usr/lib/font/dev202/DESC.out`	description of 202 typesetter
`/usr/lib/font/dev202/R.out`	tables for font `R`
	`/usr/jerq/font`

SEE ALSO

troff(1), *jf*(9.1), *font*(9.5)

NAME

latex, slitex, bibtex − tex macro package and bibliographies

SYNOPSIS

`latex` *file*[.tex]

`slitex` *file*[.tex]

`bibtex` *auxname*

DESCRIPTION

Latex is a standard set of macros for *tex*(1) inspired by, but not identical to, Scribe. The command *latex file* processes *file* `.tex` and produces *file* `.dvi,` which should be printed with *lp*(1). It will probably be necessary to run *latex* twice, to get all of the cross-referencing done properly. *Latex* writes cross-referencing information in *file* `.aux.` *Slitex* is version of *latex* for making slides.

Bibtex reads the top-level auxiliary (`.aux`) file output by *latex* and creates a bibliography (`.bbl`) file to be included in the source file. The *auxname* on the command line should be given without an extension. Each `\cite` in the source file is looked up in bibliography files to gather together those used in the document. Then a bibliography style file is executed to write a `\thebibliography` environment.

The source file should have defined the bibliography (`.bib`) files to search with the `\bibliography` command, and the bibliography style (`.bst`) file to execute with the `\bibliographystyle` command. *Bibtex* searches the `TEXINPUTS` path (see *tex*(1)) for `.bst` files, and the `BIBINPUTS` path for `.bst` files. The manual describes how to make bibliography files.

See files in `/usr/lib/tex/macros/doc` for more documentation. In particular, `local.tex` is the *Local Guide* referred to in the manual.

SEE ALSO

Leslie Lamport, *LATEX: A Document Preparation System*, Addison Wesley, 1986
Howard Trickey, *Latex User Guide*, this manual, Volume 2,
tex(1), *lp*(1)

NAME

mail − mail addresses

DESCRIPTION

Mail(1) uses the programs of *upas*(8) to interpret mail addresses.

Network addresses

A general network mail address has the form *machine*!...!*name*, with one or more machines mentioned. A machine in the middle of the list gets the mail marked 'from' the preceding part of list and forwards it to the next to handle the rest of the list.

Rules for converting addresses among the conventions of different networks are given by rewrite rules; see *upas*(8). A rough description of the rewrite rules for the local research network follows.

A simple name, containing no punctuation, is translated according to 'Local addresses' below to produce more addresses, which get rewritten in turn.

The conventional network address `local!`*name* is delivered to the mailbox `/usr/spool/mail/name` if it exists or if *name* is registered as a login name in the password file *passwd*(5). Otherwise the mail is undeliverable.

Mail to another machine is forwarded.

Addresses in other forms are rewritten recursively.

Local addresses

'Alias files' specify local name translation. Each line of an alias file begins with # (comment) or with a name. The rest of a name line gives the translation. The translation may contain multiple addresses and may be continued to another line by appending a backslash. Items are separated by white space.

In translating a name, the sender's personal alias file `$HOME/lib/names` is checked first. Then the system alias files, listed one per line in `/usr/lib/upas/namefiles`, are checked in order. If the name is not found, the translation is taken to be `local!`*name*.

On research network machines, the first system alias file is `/usr/lib/upas/names.local`; it is never touched from afar. Alias files for various organizations, e.g `/usr/lib/upas/names.1127`, are maintained, often by users themselves, at selected sites and sent around the network when changed by *ship*(8). The master alias file for center 1122 is kept on 'alice', those for other centers on 'bowell'.

Addresses to/from major networks

A 'from' address is automatically supplied as a return postmark on outgoing mail addressed 'to' the several networks. Respondents should be able to send to these addresses. For non-research AT&T machines that use the research gateway, 'from' addresses with `@` should be replaced by `person%machine@research.att.com`.

UUCP:	(to) `machine!person`
	(from) `research!person`
CSNET:	(to) `csnet!machine-domain-name!person`
	(from) `person@research.att.com`
ARPANET:	(to) `arpa!machine-domain-name!person`
	(from) `person@research.att.com`
ACSNET:	(to) `acsnet!machine-domain-name!person`
	(from) `person@research.usa`
BITNET:	(to) `bitnet!machine!person`
	(from) `person@research.att.com`

FILES

`/usr/lib/upas/namefiles`
`/usr/lib/upas/names.*`

SEE ALSO

uucp(1), *mail*(1), *upas*(8)

NAME

man – macros to typeset manual

SYNOPSIS

`nroff -man` *file* ...

`troff -man` *file* ...

DESCRIPTION

These macros are used to lay out pages of this manual.

Except in `.LR` and `.RL` requests, any text argument denoted *t* in the request summary may be zero to six words. Quotes `"` ... `"` may be used to include blanks in a 'word'. If *t* is empty, the special treatment is applied to the next text input line (the next line that doesn't begin with dot). In this way, for example, `.I` may be used to italicize a line of more than 6 words, or `.SM` followed by `.B` to make small letters in 'bold' font.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

The fonts are

`R` roman, the main font, preferred for diagnostics

`I` italic, preferred for parameters, short names of commands (use `F` for full path names), names of manual pages, and naked function names

`B` 'bold', actually the constant width font CW, preferred for examples, declarations, keywords, names of `struct` members, and literals (numbers are rarely literals)

`F` also font CW; used for filenames to help cross-indexing

`L` also font CW. In *troff* `L=B`; in *nroff* arguments of the macros `.L`, `.LR`, and `.RL` are printed in quotes; preferred only where quotes really help (e.g. lower-case literals and punctuation).

Type font and size are reset to default values before each paragraph, and after processing font- or size-setting macros.

The `-man` macros admit equations and tables in the style of *eqn*(1) and *tbl*(1), but do not support arguments on `.EQ` and `.TS` macros.

These strings are predefined by `-man`:

`\*R` '®', '(Reg)' in *nroff.*

`\*S` Change to default type size.

FILES

`/usr/lib/tmac/tmac.an`
`/usr/man/man0/xx`

SEE ALSO

troff(1), *man*(1)

REQUESTS

Request	Cause Break	If no Argument	Explanation
`.B` *t*	no	*t*=n.t.l.*	Text *t* is 'bold'.
`.BI` *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and italic.
`.BR` *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and Roman.
`.CT` *c x*...	no		Chapter *c*, topics *x* in topic index; see `/usr/man` for topic codes.
`.DT`	no		Restore default tabs.
`.EE`	yes		End displayed example
`.EX`	yes		Begin displayed example
`.F` *t*	no	*t*=n.t.l.	Text *t* is filename.
`.FR` *t*	no	*t*=n.t.l.	Join words of *t* alternating filename and Roman.
`.HP` *i*	yes	*i*=p.i.*	Set prevailing indent to *i*. Begin paragraph with hanging indent.

`.I` *t*	no	*t*=n.t.l.	Text *t* is italic.
`.IB` *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and bold.
`.IP` *x i*	yes	*x*=""	Same as `.TP` with tag *x*.
`.IR` *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and Roman.
`.L` *t*	no	*t*=n.t.l.	Text *t* is literal.
`.LP`	yes		Same as `.PP`.
`.LR` *t*	no		Join 2 words of *t* alternating literal and Roman.
`.PD` *d*	no	*d*=`.4v`	Interparagraph distance is *d*.
`.PP`	yes		Begin paragraph. Set prevailing indent to default.
`.RE`	yes		End of relative indent. Set prevailing indent to amount of starting `.RS`.
`.RF` *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and filename.
`.RI` *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and italic.
`.RL` *t*	no		Join 2 or 3 words of *t* alternating Roman and literal.
`.RS` *i*	yes	*i*=p.i.	Start relative indent, move left margin in distance *i*. Set prevailing indent to default for nested indents.
`.SH` *t*	yes	*t*=""	Subhead; reset paragraph distance.
`.SM` *t*	no	*t*=n.t.l.	Text *t* is small.
`.SS` *t*	no	*t*=""	Secondary subhead.
`.TF` *s*	yes		Prevailing indent is wide as string *s* in font `L`; paragraph distance is 0.
`.TH` *n c x*	yes		Begin page named *n* of chapter *c; x* is extra commentary, e.g. 'local', for page head. Set prevailing indent and tabs to default.
`.TP` *i*	yes	*i*=p.i.	Set prevailing indent to *i*. Restore default indent if *i*=0. Begin indented paragraph with hanging tag given by next text line. If tag doesn't fit, place it on separate line.
`.1C`	yes		Equalize columns and return to 1-column output
`.2C`	yes		Start 2-column nofill output

* n.t.l. = next text line; p.i. = prevailing indent

BUGS

There's no way to fool *troff* into handling literal double quote marks `"` in font-alternation macros, such as `.BI`.

There is no direct way to suppress column widows in 2-column output; the column lengths may be adjusted by inserting `.sp` requests before the closing `.1C`.

NAME

mbits − macros to typeset bitmaps

SYNOPSIS

`troff` [*option* ...] `-mbits` [*option* ...] *file* ...

DESCRIPTION

These macros are used to typeset bitmaps. They are compatible with other *troff* macro packages, so that bitmap figures may be included in documents.

`.BM` *f* — Set the format for subsequent .BM requests to *f*. The default is `b` for *blitblt*(9.1) output. The other possibility is `i` for faces and large icons in ASCII format.

`.BM` *f s* — Insert the bitmap from file *f*; each pixel will be *s* basic units square. The bitmap origin is placed at the current point, which is left unchanged.

`.BM` *f s dX dY* — Set the number registers *dX* and *dY* to the width and height (in basic units) of the bitmap in file *f*, assuming pixel size *s*. This form may be used to calculate positioning.

EXAMPLES

This sequence centers the bitmap and spaces past it to continue with the text:

```
.BM i
.BM /n/face/48x48x1/pjw 6 dX dY
.sp
.in (\n(.lu-\n(dXu)/2u
.BM /n/face/48x48x1/pjw 6
.in
.sp \n(dYu
```



FILES

`/usr/lib/btroff`

SEE ALSO

blitblt(9.1), *troff*(1), *bitfile*(9.5), *ms*(6)

BUGS

At time of writing, *mbits* does not work with PostScript output devices. See *mpictures*(6) for an alternative.

`.BM` does not work inside a diversion.

The concept of ‘pixel size’ varies among typesetting devices.

NAME

mcs − macros for formatting cover sheets

SYNOPSIS

`troff −mcs` *file ...*

DESCRIPTION

The *mcs* package of *troff*(1) macros generates cover sheets for Bell Labs documents. The macros were not meant for human production. Use *docgen*(1) to write them:

```
docgen −mcs
```

The *mcs* macros automatically load the *ms*(6) macros.

In the following description macros marked * are mandatory; all others are optional. Optional arguments are enclosed in square brackets. The macros must be used in the order presented:

Request	Explanation
* `.TI` [*draft*]	Title text follows. Unless there is an argument, the cover sheet will be shipped automatically to the Bell Labs library ITDS. Duplicate shipments are harmless.
* `.AH` *author loc dept ext rm e-mailaddre co*	
`.AP` *name*	Responsible AT&T person
* `.SA`	Begin Abstract
* `.SE`	End Abstract
`.KW` [*k1 ... k9*]	Keywords
* `.TY` *type software*	Memo *type*: `TM` technical memorandum, `IM` internal memorandum, `TC` technical correspondence; *software*: `y` if memo is software related
* `.NU` *org-date-seq filing_case work_project*	Document Number
`.ED` *doc_number*	Earlier document number.
* `.MY` [*a1 ... a8*]	Mercury Code, positional arguments, `y`-`n`, at most 3 `y`: 1 Chemistry and Materials, 2 Communications, 3 Computing, 4 Electronics, 5 Life Science, 6 Mathematics and Statistics, 7 Physics, 8 Manufacturing
`.RL` *code*	`y` release to any AT&T employee; `n` release only on approval of each request.
`.PR` [`BR`]	Proprietary Marking, default ATT-BL Proprietary, `0` unmarked, `BR` restricted
`.GS`	Government Security
`.CO`	Complete Copy Distribution List follows
`.CE`	End distribution lists
`.CV`	Cover Sheet Only Distribution List follows
* `.SC` *pages*	Total *pages ;* do not include coversheet pages (if old format of *pages* and *otherpages* , the arguments are added).

To turn the paper into released paper format put a `.RP` before the title macro. To make it into a CSTR add a `.TR` before the title and `.AI` `.MH` after the author macro.

FILES

`/usr/lib/tmac/tmac.cs`
`/usr/lib/tmac/tmac.rscover`

SEE ALSO

docgen(1), *ms*(6), *sendcover*(8)

NAME

mpictures − picture inclusion macros

SYNOPSIS

`troff -mpictures` [*options*] *file* ...

DESCRIPTION

Mpictures macros insert PostScript pictures into *troff*(1) documents. The macros are:

`.BP` *source height width position offset flags label*

Define a frame and place a picture in it. Null arguments, represented by `""`, are interpreted as defaults. The arguments are:

source Name of a PostScript picture file, optionally suffixed with (*n*) to select page number *n* from the file (first page by default).
height Vertical size of the frame, default `3.0i`.
width Horizontal size of the frame, current line length by default.
position `l` (default), `c`, or `r` to left-justify, center, or right-justify the frame.
offset Move the frame horizontally from the original *position* by this amount, default `0i`.
flags One or more of:
`a`*d* Rotate the picture clockwise *d* degrees, default *d*=90.
`o` Outline the picture with a box.
`s` Freely scale both picture dimensions.
`w` White out the area to be occupied by the picture.
`l`,`r`,`t`,`b` Attach the picture to the left right, top, or bottom of the frame.
label Place *label* at distance `1.5v` below the frame.

If there's room, `.BP` fills text around the frame. Everything destined for either side of the frame goes into a diversion to be retrieved when the accumulated text sweeps past the trap set by `.BP` or when the diversion is explicitly closed by `.EP`.

`.PI` *source height*`,` *width*`,` *yoffset*`,`*xoffset flags*.

This low-level macro, used by `.BP`, can help do more complex things. The two arguments not already described are:

xoffset Offset the frame from the left margin by this amount, default `0i`.
yoffset Offset the frame from the current baseline, measuring positive downward, default `0i`.

`.EP` End a picture started by `.BP`; `.EP` is usually called implicitly by a trap at frame bottom.

If a PostScript file lacks page-delimiting comments, the entire file is included. If no `%%BoundingBox` comment is present, the picture is assumed to fill an 8.5×11-inch page. Nothing prevents the picture from being placed off the page.

FILES

`/usr/lib/tmac/tmac.pictures`

SEE ALSO

troff(1)

DIAGNOSTICS

A picture file that can't be read by the PostScript postprocessor is replaced by white space.

BUGS

A picture and associated text silently disappear if a diversion trap set by `.BP` isn't reached. Call `.EP` at the end of the document to retrieve it.

Macros in other packages may break the adjustments made to the line length and indent when text is being placed around a picture.

A missing or improper `%%BoundingBox` comment may cause the frame to be filled incorrectly.

NAME

mpm − macros for page makeup

SYNOPSIS

`troff -mpm` *file* ...

DESCRIPTION

These *troff*(1) macros, largely compatible with *ms*(6), make better pages. They silently invoke and provide information to a postprocessor that moves floating figures, avoids widows, and justifies pages vertically by stretching vertical spaces that result from `.PP`, `.LP`, `.IP`, `.QP`, `.SH`, `.NH`, `.DS`/`.DE`, `.EQ`/`.EN`, `.TS`/`.TE`, `.PS`/`.PE`, `.P1`/`.P2`, and `.QS`/`.QE`.

The following macros are different from or not part of `-ms`. Values denoted *n* have default value `1v`.

`.BP` Begin a new page.
`.FL` Flush: force out previous keeps.
`.FC` Finish a two-column region and start a new one.
`.KF` *m* Floating keep, with preferred center at vertical position *m*. Special values `top` (default) and `bottom` are permitted.
`.NE` *n* Start new page if remaining vertical space on this page is less than *n* .
`.P1` Begin a program display (Courier font).
`.P2` End a program display.
`.P3` Insert optional break point in program display.
`.SP` *n* `exactly`
`.SP` *n* Insert fixed or stretchable vertical space of height *n.*
`.Tm` *text*
Place page number and *text* on the standard error output.
`.X` *text* Present *text* to the hidden page-makeup program as part of a device-dependent output sequence `x X` *text*. Equivalent to `\X'`*text*`'`.

Useful number registers:
`HM` Header margin; default 1 inch.
`FM` Footer margin; default 1 inch.
`FO` Footer position; default 10 inches.
`%#` Page number of current page.
`dP`,`dV` Shrinkage of point size and vertical spacing for `.P1`, in points.

Useful strings:
`%e`,`%o` Even and odd page title commands, as `.tl ''''`.

FILES

`/usr/lib/tmac/tmac.pm`
`/usr/lib/tmac/pm`

SEE ALSO

ms(6), *troff*(1)
B. W. Kernighan and C. J. Van Wyk, 'The −mpm Macro Package', this manual, Volume 2

BUGS

These features of `-ms` are missing:
Document styles other than the default `.RP`.
Space between front matter and first paragraph. Recover it with `.SP 2`.
Separating rule above footnotes.
Keeps assigned to a separate page.
Pages with more than two columns.
Troff option `-o` doesn't work with `-mpm` because only the postprocessor knows the page numbers.

NAME

ms − macros for formatting manuscripts

SYNOPSIS

`nroff −ms` [*options*] *file* ...
`troff −ms` [*options*] *file* ...

DESCRIPTION

This package of *nroff* and *troff*(1) macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col*; see *column*(1).

The macro requests are defined below. Many *nroff* and *troff* requests are unsafe in conjunction with this package, but the following requests may be used with impunity after the first `.PP`: `.bp`, `.br`, `.sp`, `.ls`, `.na`.

Output of the *eqn*(1), *neqn*, *tbl*(1), *pic*(1), *refer*(1), and *prefer*(1) preprocessors for equations, tables, pictures, and references is acceptable as input.

Diacritical marks may be applied to letters, as in these examples:

`\*`e`	`*`a`	`\*'e`	`\*^e`	`\*^o`	`\*:u`	`\*~n`	`\*,c`	`\*vc`
è	à	é	ê	ô	ü	ñ	ç	č

FILES

`/usr/lib/tmac/tmac.s`

SEE ALSO

M. E. Lesk, 'Typing Documents on the UNIX System: Using the −ms Macros with Troff and Nroff', this manual, Volume 2
eqn(1), *troff*(1), *refer*(1), *prefer*(1), *tbl*(1), *pic*(1), *mcs*(6)

REQUESTS

Request	Initial Value	Cause Break	Explanation
`.1C`	yes	yes	One column format on a new page.
`.2C`	no	yes	Two column format.
`.AB`	no	yes	Begin abstract.
`.AE`	-	yes	End abstract.
`.AI`	no	yes	Author's institution follows. Suppressed in `.TM`.
`.AT`	no	yes	Print 'Attached' and turn off line filling.
`.AU` *x y*	no	yes	Author's name follows. *x* is location and *y* is extension, ignored except in `TM`.
`.B` *x y*	no	no	Print *x* in boldface, append *y*; if no argument switch to boldface.
`.B1`	no	yes	Begin text to be enclosed in a box.
`.B2`	no	yes	End boxed text.
`.BI` *x y*	no	no	Print *x* in bold italic and append *y*; if no argument switch to bold italic.
`.BT`	date	no	Bottom title, automatically invoked at foot of page. May be redefined.
`.BX` *x*	no	no	Print *x* in a box.
`.CW` *x y*	no	no	Constant width font for *x*, append *y*; if no argument switch to CW.
`.CT`	no	yes	Print 'Copies to' and turn off line filling.
`.DA` *x*	nroff	no	'Date line' at bottom of page is *x*. Default is today.
`.DE`	-	yes	End displayed text. Implies `.KE`.
`.DS` *x*	no	yes	Start of displayed text, to appear verbatim line-by-line: `I` indented (default), `L` left-justified, `C` centered, `B` (block) centered with straight left margin. Implies `.KS`.
`.EG`	no	-	Print document in BTL format for 'Engineer's Notes.' Must be first.
`.EN`	-	yes	Space after equation produced by *neqn* or *eqn*(1).
`.EQ` *x y*	-	yes	Display equation. Equation number is *y*. Optional *x* is `I`, `L`, `C` as in `.DS`.
`.FE`	-	yes	End footnote.
`.FP` *x*	-	no	Set font positions for a family, e.g., `.FP palatino`

.FS	no	no	Start footnote. The note will be moved to the bottom of the page.
.HO	-	no	'AT&T Bell Laboratories, Holmdel, New Jersey 07733'.
.I *x y*	no	no	Italicize *x*, append *y*; if no argument switch to italic.
.IH	no	no	'AT&T Bell Laboratories, Naperville, Illinois 60540'
.IM	no	no	Print document in BTL format for an internal memorandum. Must be first.
.IP *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.KE	-	yes	End keep. Put kept text on next page if not enough room.
.KF	no	yes	Start floating keep. If the kept text must be moved to the next page, float later text back to this page.
.KS	no	yes	Start keeping following text.
.LG	no	no	Make letters larger.
.LP	yes	yes	Start left-blocked paragraph.
.LT *x r p*	no	yes	Start a letter with today's date. If *x* is missing, use letterhead paper; otherwise print letterhead with room *r* and phone *p*; address follows.
.MF	-	-	Print document in BTL format for 'Memorandum for File.' Must be first.
.MH	-	no	'AT&T Bell Laboratories, Murray Hill, New Jersey 07974'.
.MR	-	-	Print document in BTL format for 'Memorandum for Record.' Must be first.
.ND *date*	troff	no	Use date supplied (if any) only in special BTL format positions; omit from page footer.
.NH *n*	-	yes	Same as .SH, with automatic section numbers like '1.2.3'; *n* is subsection level (default 1).
.NL	yes	no	Make letters normal size.
.PE	-	yes	End picture; see *pic*(1).
.PF	-	yes	End picture; restore vertical position.
.PP	no	yes	Begin paragraph. First line indented.
.PS *h w*	-	yes	Start picture; height and width in inches.
.PY	-	no	'AT&T Bell Laboratories, Piscataway, New Jersey 08854'
.QE	-	yes	End quoted material.
.QP	-	yes	Begin quoted paragraph (indent both margins).
.QS	-	yes	Begin quoted material (indent both margins).
.R	yes	no	Roman text follows.
.RE	-	yes	End relative indent level.
.RP	no	-	Cover sheet and first page for released paper. Must precede other requests.
.RS	-	yes	Start level of relative indentation from which subsequent indentation is measured.
.SG *x*	no	yes	Insert signature(s). In .TM *x* is initials of author and typist; in .LT *x* is author's name.
.SH	-	yes	Section head follows, font automatically bold.
.SM	no	no	Make letters smaller.
.TA *x*...	5...	no	Set tabs in ens. Default is 5 10 15 ...
.TE	-	yes	End table; see *tbl*(1).
.TH	-	yes	End heading section of table.
.TL	no	yes	Title follows.
.TM *x*...	no	-	Print document in BTL technical memorandum format. Arguments are TM number, (quoted list of) case number(s), and file number. Must precede other requests.
.TR *x*	-	-	Print in BTL technical report format; report number is *x*. Must be first.
.TS *x*	-	yes	Begin table; if *x* is H table heading is repeated on new pages.
.UL *x*	-	no	Underline argument (even in troff).
.UX *y z*	-	no	'*z*UNIX*y*'; first use gives registered trademark notice.
.WH	-	no	'AT&T Bell Laboratories, Whippany, New Jersey 07981'.
.[	-	no	Begin reference; see *refer*(1).
.]	-	no	End reference.

NAME

terminals − conventional names

DESCRIPTION

These names are used by certain commands and are maintained as part of the shell environment; see *sh*(1), *environ*(5).

`2621`	Hewlett-Packard HP262? series terminals
`1620`	DIABLO 1620 (and others using HyType II)
`33`	Teletype Model 33
`37`	Teletype Model 37
`43`	Teletype Model 43
`5620`	Teletype Model 5620 dotmap display
`dumb`	terminals with no special features
`4014`	Tektronix 4014
`vt52`	Digital Equipment Corp. VT52

The list goes on and on. Consult `/etc/termcap` (see *termcap*(5)) for the whole truth.

Commands whose behavior may depend on the terminal typically consult `TERM` in the environment or accept arguments of the form `-T`*term*, where *term* is one of the names given above.

SEE ALSO

stty(1), *tabs*(1), *plot*(1), *sh*(1), *environ*(5), *ul*(1), *column*(1), *termcap*(5), *nroff* in *troff*(1)

BUGS

The programs that ought to adhere to this nomenclature do so only fitfully.

7 INFORMATION SOURCES

7 INFORMATION SOURCES

NAME

apnews, ap.keys − present AP wire stories

SYNOPSIS

`apnews` [`−f` *dir*] [`−r`]

DESCRIPTION

Apnews presents news from the AP wire on a cursor-addressed screen. The top half of the screen contains 20 story slugs (two-word labels). *Apnews* responds to these commands:

n Print story for slug *n*; page through it by typing newlines.

`m` Present more slugs.

`.` Return to current slug list.

`t` Top. Return to first list of slugs

`s` *keywords*
Present slugs for stories containing these keywords.

`y` Present slugs for stories containing words from the last story read.

`c` *file* Copy. Add story being read to named file or directory.

`?` Print some help.

To suggest interest, slugs may be followed by a bracketed number that shows the average number of pages (up to 5) that readers have perused. Option `−r` turns this feature off.

Option `−f` directs the attention of *apnews* to a specified directory of AP stories, as may be collected by `ap.keys`.

To monitor news automatically, put a file `ap.keys` in your home directory. This file contains instructions marked by `*`, each followed by one or more search lines. Instructions specify what to capture:

`S` whole story
`P` first paragraph
`H` heading

then what to do with it:

`|` *command*
specifying a command (often mail) to be executed with the story as standard input
`>` *file* specifying a file or directory to add the story to; pathnames are relative to your home directory

If no instruction is present, the default is

```
*S | mail yourself
```

Search lines may contain:

(1) a sequence of blank-separated words; these words must occur in this order
(2) a sequence of words separated by commas; these words must appear in the same sentence
(3) a sequence of words separated by periods; these words may occur anywhere in the story, but all must appear

Combinations are allowed, e.g. `x . y, z` specifies `y` and `z` in the same sentence and `x` somewhere in the same story. The character `!` means not, so that `!chocolate chip` means `chip` not preceded by `chocolate`. Some suffixes are removed; and capitals are ignored except when entire words are capitalized. Thus `ERA` and `era` are distinguished, but `Waters` and `waters` are not. Special 'words' specify story types:

`#f` flash
`#b` bulletin
`#u` urgent news

```
#r      regular news
#d      deferred news
```

EXAMPLES

```
*S > stuff
bell laboratories
FCC . telephone, regulation
*P | mail joe
#b
AM-NewsDigest
```

FILES

```
ap.keys
/usr/spool/ap/*
```

BUGS

Apnews can fail to work well in a *mux*(9.1) window, for two reasons.
(1) The window needs a terminal emulator. Before invoking *apnews*, do `exec term 5620` (or `2621`); see *term*(9.1).
(2) Remote execution needs a transparent connection. If logged in elsewhere make the connection to the serving machine by doing, for example, `ndcon alice` or `nrx alice apnews` (after downloading an emulator, if necessary); see *dcon*(1).

NAME

cal − print calendar

SYNOPSIS

`cal` [*month*] *year*

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. *Year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

BUGS

The year is always considered to start in January even though this is historically naive.
Beware that `cal 90` refers to the early Christian era, not the 20th century.

NAME

dict − look up words in English dictionaries

SYNOPSIS

`dict` [`-p`] [*dictionary* [*word*]]

DESCRIPTION

Dict looks up words in the specified *dictionary*, `webster` by default. Words are read, one per line from the standard input, and entries are written on the standard output. Characters other than letters, digits, and space are ignored. A single word may be specified in the command. The options are

`-p` Find all entries of which the specified *word* is a prefix.

`-r` Print raw form, including diacriticals, font marks, etc. Different for each dictionary to which it applies.

Dictionary is one of

`webster` Merriam-Webster Collegiate Dictionary, Seventh Edition, full text. No option `-p`.

`web7` Same, words only.

`etym`[`ology`] Inverted index to `webster` by root words.

`web2` Merriam-Webster New International Dictionary, Second Edition, unabridged, words only.

`oed` The Oxford New English Dictionary, or OED, full text. No option `-p`.

`oxford` The Oxford Advanced Learner's Dictionary of Contemporary English, full text. No `-p`.

`slang` New Dictionary of American Slang (Harper). Field identifiers: `me` main entry (perhaps flagged `@`), `nu` sense number (also given as `*`*n*`*`), `pr` pronunciation, `ps` part of speech, `vr` variations `la` provenance, `df` definition, `dx` definition by example, `ex` example, `ed` editorial note, `et` etymology, `xr` cross reference, `xx` indirect address, `sq` sequence number in original text.

`names` The Oxford Dictionary of British surnames.

`thesaurus` Collins Thesaurus.

`thesaurusa` Same, augmented with complete backreferences among words.

`places` USGS Gazetteer of populated places in US and possessions, with standard county code, latitude, longitude, year of listing, altitude (feet), 1980 population, topo sheet code. For full name search, use thus: `dict places 'new york, ny'`; without state, use prefix search: `dict -p places 'new york'`.

towns A shorter gazetteer (PICADAD) of US populated places with latitude, longitude, zip code, and population class (0:0-1000, 1:1000-2500, 2:2500-5000, 3:5000-10,000, 4:10,000-25,000, 5:25,000-50,000, 6:50,000-100,000, 7:100,000-250,000, 8:250,000-500,000, 9:500,000+), and something else.

`spell` Word list of *spell*(1).

`acro`[`nym`] 17000 AT&T acronyms

anything else Print list of available dictionaries.

The dictionaries are copyrighted and must not be copied without permission, except for `web2`, `spell`, and `acro`.

FILES

`/usr/dict/*`	(`/n/bowell`)
`/usr/dict/words`	`spell` or `web7`, depending on machine
`/usr/dict/oed`	(`/n/kwee`)

```
/usr/lib/dict/*
/usr1/maps/usplaces        (/n/bowell)
/usr/spool/town/ustowns*   (/n/alice)
```

SEE ALSO

look(1), *town*(7)

BUGS

In `webster`, `web7`, and `pron`, diacriticals are done right − by overstrikes − which means they disappear on most screen terminals.

In `towns`, all data are lower case; missing zip codes look like normal codes ending in ‘000’. Latitude and longitude denote the center of population of the containing political entity; unincorporated places are spotted at the county center.

In `slang`, some entries contain extra trash; option `-p` helps overcome the trouble.

NAME

dkname – map system name to Datakit address

SYNOPSIS

`dkname` *sysname* ...

DESCRIPTION

Dkname looks up machine names and prints the corresponding full Datakit address on the standard output. The addresses are not necessarily printed in the same order as the arguments. The addresses are obtained by the first possible match in the file `/usr/lib/uucp/Systems.dk`.

FILES

`/usr/lib/uucp/Systems.dk`

NAME

library, bellcat − bell labs library services

SYNOPSIS

`library` [*option* ...] [*item* ...]

`bellcat` [`−q`] [*database*]

DESCRIPTION

Library mails orders to the Bell Labs library network for books, technical reports, etc. Its use is self-explanatory.

A long response may be inserted with `~e` or `~r` as in *Mail*(A). Interaction may be forestalled by answering questions on the command line (see example) and in a personal identity file, named in environment variable `LIBFILE` (`.lib` by default), which contains one or more lines like these:

`ID:` PAN or SSN
`LIBNAME:` last name
`LIBLOG:` log file, readable with `mail −f`
`LIBCNTL:` concatenated search control codes: `a` acknowledge, `m`*number* max on retrieved items
`LIBLOCAL:` interaction control code: `x` brief prompts

Most `LIBFILE` items may be entered as environment variables by the same names.

Bellcat places a call to on-line library databases. Once entered, *bellcat* is self-explanatory. to exit. Option `−q` gets a ‘quick search’, which uses no special terminal features and does not offer help.

Some of the databases are

`books` (default)
`journals`
`released`
: released papers by Bell Labs authors

`tech_reports`
: non-AT&T technical reports

xxx
: unknown name causes a list of databases to be printed

The databases of *Bellcat* and more are available through the LINUS service of the library network. You may use the *library* command to sign up for LINUS.

EXAMPLES

`library −1 123456−851234−56tm`
: Order a technical memorandum, giving answer 1 for kind of query and specifying a document number.

`library −4 −p waldstein, r k`
: Consult the people file.

FILES

`$HOME/.lib`
`/usr/lib/bellcat`

BUGS

Except under option `−q`, *bellcat* requires a native-mode Teletype 5620 or a (possibly simulated) HP2621 terminal. Under *mux*(9.1) *bellcat* invokes a simulator if necessary. This introduces an extra level of shell, which can can be avoided thus: `exec bellcat`.

NAME

map − draw maps on various projections

SYNOPSIS

`map` *projection* [*param* ...] [*option* ...]

DESCRIPTION

Map prepares on the standard output a map suitable for display by any plotting filter described in *plot*(1). A menu of projections is produced in response to an unknown *projection*. For the meanings of *params* pertinent to particular projections see *map*(3).

The default data for *map* are world shorelines. Option `−f` accesses more detailed data classified by feature.

`−f` [*feature* ...]
Known features are ranked 1 to 4 from major to minor, with a higher-numbered rank including all lower-numbered ones. A missing rank is taken to be 1. Features are

`shore`[1-4]	seacoasts, lakes, and islands; option `−f` always shows `shore1`
`ilake`[1-2]	intermittent lakes
`river`[1-4]	rivers
`iriver`[1-3]	intermittent rivers
`canal`[1-3]	3=irrigation canals
`glacier`	
`iceshelf`[12]	
`reef`	
`saltpan`[12]	
`country`[1-3]	2=disputed boundaries, 3=indefinite boundaries
`state`	states and provinces (US and Canada only)

In the following list of other options all coordinates are in degrees, with north latitude and west longitude counted as positive.

`−l` *S N E W*
the southern and northern latitude and the eastern and western longitude limits of the desired map. Missing arguments are filled out from the list −90, 90, −180, 180, or lesser limits suitable to the projection at hand.

`−k` *S N E W*
Set the scale as if for a map with limits `−l` *S N E W*. Do not consider any `−l` or `−w` option in setting scale.

`−o` *lat lon rot*
Orient the map in a nonstandard position. Imagine a transparent gridded sphere around the globe. First turn the overlay about the North Pole so that the Prime Meridian (longitude 0) of the overlay coincides with meridian *lon* on the globe. Then tilt the North Pole of the overlay along its Prime Meridian to latitude *lat* on the globe. Finally again turn the overlay about its 'North Pole' so that its Prime Meridian coincides with the previous position of meridian *rot*. Project the desired map in the standard form appropriate to the overlay, but presenting information from the underlying globe. Missing arguments are filled out from the list 90, 0, 0. Default orientation is 90, 0, *m*, where *m* is the middle of the longitude range.

`−w` *S N E W*
Window the map by the specified latitudes and longitudes in the tilted, rotated coordinate system. Missing arguments are filled out from the list −90, 90, −180, 180. (It is wise to give an encompassing `−l` option with `−w`. Otherwise for small windows computing time varies inversely with area!)

`−d` *n* Alter the density of points by plotting only every *n*th one.

`-r` Reverse left and right side of map, for star charts and inside-out views.

`-s` Save the screen, don't erase before drawing. Output made under `-s` must be appended to output of another *map* command.

`-g` *dlat dlon res*
Grid spacings are *dlat, dlon* (10 degrees by default). Zero spacing means no grid. Missing *dlat* is taken to be zero. Missing *dlon* is taken the same as *dlat*. Grid lines are drawn to a resolution of *res* (2 degrees or less by default).

`-p` *lat lon extent*
Position the point *lat, lon* at the center of the plotting area, and scale so that the height (and width) of the nominal plotting area is *extent* times the size of one degree of latitude at the center. For example, a standard Mercator wall map comes out as if positioned and scaled with `-p 0 0 360`. An *extent* overrides option `-k`.

`-c` *x y* After all other positioning and scaling operations have been performed, move the center of the map to position *x, y*, where the nominal plotting area is $-1 \le x \le 1$, $-1 \le y \le 1$.

`-m` [*file* ...]
Use map data from named files instead of the defaults. If no files are named, omit map data. Names that are not the pathname of real files are looked up in a standard directory, which contains, in addition to the data for `-f`,

`world`	World Data Bank I from CIA (the default map)
`states`	US map from Census Bureau
`counties`	US map from Census Bureau

The environment variables `MAP` and `MAPDIR` change the default map and default directory.

`-b` [*lat lon*...]
Suppress the drawing of the normal boundary (defined by option `-l`). Coordinates, if present, define the vertices of a drawn polygon to which the map is clipped. If only two vertices are given, they are taken to be the diagonal of a rectangle.

`-t` *file* ...
The following arguments name ASCII files that contain lists of points, given as latitude-longitude pairs in degrees. If the first file is named `-`, the standard input is taken instead. The points of each list are plotted as connected 'tracks'.

Points in a track file may be followed by label strings. A label breaks the track. A label may be prefixed by `"`, `:`, or `!` and is terminated by a newline. An unprefixed string or a string prefixed with `"` is displayed at the designated point. The first word of a `:` or `!` string names a special symbol (see option `-y`). An optional numerical second word is a scale factor for the size of the symbol, 1 by default. A `:` symbol is aligned with its top to the north; a `!` symbol is aligned vertically on the page.

`-u` *file* ...
Same as `-t`, except the tracks are unbroken lines. (`-t` tracks appear as dot-dashed lines if the plotting filter supports them.)

`-y` *file* The *file* contains *plot*(5)-style data for `:` or `!` labels in `-t` or `-u` files. Each symbol is defined by a comment `:`*name* then a sequence of `m` and `v` commands. Coordinates (0,0) fall on the plotting point. Default scaling is as if the nominal plotting range were `ra -1 -1 1 1`; `ra` commands in *file* change the scaling.

EXAMPLES

`map perspective 1.025 -o 40.75 74`
A view looking straight down on New York from 100 miles (0.025 earth radius) up. The job can be done faster by limiting the map so as not to 'plot' the invisible part of the world: `map perspective 1.025 -o 40.75 74 -l 20 60 30 100`

`map mercator -o 49.25 -106 180`
An 'equatorial' map of the earth centered on New York. The pole of the map is placed 90 degrees away (40.75+49.25=90) on the other side of the earth. A 180-degree twist around the pole of the map arranges that the 'Prime Meridian' of the map runs from the pole of the map over the North Pole to New York instead of down the back side of the earth. The same effect can be had from `map mercator -o 130.75 74`

`map albers 28 45 -l 20 50 60 130 -m states`
A customary curved-latitude map of the United States.

FILES

`/usr1/maps/[1-4]??`	World Data Bank II
`/usr1/maps/world`	World Data Bank I
`/usr1/maps/states`	DIMECO
`/usr1/maps/counties`	DIMECO
`/usr/lib/map`	the program

SEE ALSO

map(5), *proj*(3), *plot*(1)

DIAGNOSTICS

'Map seems to be empty'—the intersection of the limits and the window seems to be null; for very local maps, the grid resolution *res* may have to be refined.

BUGS

Border lines are drawn only along edges arising from `-b` or `-l` options (including default `-l`). No borders appear along edges arising from `-w` or from visibility limits.
Border lines arising from `-b` cannot be suppressed.
Segments that cross a border are dropped, not clipped.
Because 'close' messages stop some plotting filters, *map* puts no close message on the standard output. This prevents such filters from stopping early when an overlay follows with option `-s`. However such filters may not like to see an end of file; if funny things happen at the end of a map, tack on a close message (see *plot*(5)).

NAME

news – print news items

SYNOPSIS

`news` [`-a`] [`-n`] [`-s`] [*item ...*]

DESCRIPTION

When invoked without options, this simple local news service prints files that have appeared in `/usr/news` since last reading, most recent first, with each preceded by an appropriate header. The time of reading is recorded. The options are

`-a` Print all items, regardless of currency. The recorded time is not changed.

`-n` Report the names of the current items without printing their contents, and without changing the recorded time.

`-s` Report the number of current items.

Other arguments select particular news items.

If an interrupt is received during a news item, the next item is started immediately. Another interrupt within a second of the first causes the program to terminate.

To post a news item, create a file in `/usr/news`.

You may arrange to receive news automatically by registering your mail address in `/usr/lib/subscribers`. A daemon mails newly posted news items to all addresses on the list.

FILES

`/usr/news/*`
`$HOME/news_time` date of last read news
`/usr/lib/subscribers`

SEE ALSO

calendar(1), *readnews*(7)

NAME

findauthor, papers, makepaper − consult database of locally authored papers

SYNOPSIS

`findauthor` *person*

`papers` *person*

`makepaper` *papername*

DESCRIPTION

Findauthor produces a shell *cd* command to set the current directory to the place where papers by *person* (a login name or a last name) are stored.

Papers lists the names and titles of the papers stored under *person.*

Makepaper produces *troff*(1) output for *papername.*

The database is stored in file system `/n/bowell/pap`. File `/n/bowell/pap/Titles` lists titles, authors and installation date of papers in the database. Many papers have been preprocessed for quick presentation of figures and equations with *reader*(9.7). These papers are stored in directories named *papername*`.d`; other papers are stored as single files.

EXAMPLES

`` `findauthor aho` ``

Change to directory of Aho's works.

FILES

`/n/bowell/pap/Titles` titles, authors and installation dates
`/n/bowell/pap/*org` membership list
`/n/bowell/pap/`*center*`/`*department*`/`*author*`/`*papername*[`.d`]

SEE ALSO

reader(9.7), *docsubmit*(1), *troff*(1), *doctype*(1)

BUGS

Makepaper depends on *doctype*(1) to determine what preprocessors to run.
Makepaper does not know the author's original arguments to *refer*, so papers are produced with *refer*'s default arguments.
Makepaper does not work with non-*troff* formatters such as *tex*(1) or *monk*(1).

NAME

polypic, polypr − database of polyhedra

SYNOPSIS

`polypic` [`-phfis`] *solid* ...
`polypr` [`-cfhnpsi`*file*] *solid* ...

DESCRIPTION

Polypic outputs a picture of the planar nets for the specified solids in *plot*(5) format. The options are:

`-f` Print the face numbers.

`-h` Print the hinge numbers.

`-i`*file* Use *file* instead of the normal database.

`-s` Produce an orthogonal view of the 3D solid rather than the net.

`-p` Produce *pic*(1) output.

Polypr gives more general access to the data. By default, all the data is output. If no solid is specified, all the solids in the database are output. The options are:

`-n` Print only the database index number and the name.

`-h` Print only a header line with name and number of faces and hinges.

`-f` Print only the data for the flat net.

`-s` Print only the data for the 3D solid.

`-i`*file* Use *file* instead of the normal database.

`-c` Produce output for Tom Duff's polygon renderer (implies `-s`).

`-p` Produce useful parameters for a perspective view; implies `-c`.

For both *polypic* and *polypr*, the solid may be specified by either its index number or by any prefix of the name. Ambiguities are resolved by database order.

FILES

`/usr/include/poly.h`
`/usr/lib/polyhedra`

SEE ALSO

plot(1), *poly*(5)
A. G. Hume, 'Exact Descriptions of Regular and Semi-regular Polyhedra and Their Duals' *Computing Science Technical Report 130,* AT&T Bell Laboratories, November, 1986

BUGS

Not all the polyhedra have valid 3D data.
Option `-s` doesn't work yet.

NAME

postnews – submit netnews articles

SYNOPSIS

`postnews` [*file*]

DESCRIPTION

Postnews submits the *file* as a *readnews*(7) article. It prompts for title, primary newsgroup, and other newsgroup distribution. Good manners decree an informative title and accurate, minimal, distribution.

The names of newsgroups are relative pathnames of directories depending from `/usr/spool/news`, with slashes replaced by dots.

If no file is specified, *postnews* invokes an editor specified by the environment variable `EDITOR` (default `vi`). The editor's buffer is initialized with header information, which may be changed. The text of the article may be appended.

SEE ALSO

readnews(7)
BSD manual for more sophisticated uses, such as posting news from a program.

BUGS

The editor default is distinctly nonclassical.

NAME

qns − query name server

SYNOPSIS

`qns` [`-n` *server*] [*request*]

DESCRIPTION

Qns retrieves information from a database of naming information. It is used by *rsh* and *rlogin* (see *dcon*(1)) to translate names to internet addresses and by *mail*(1) to route electronic mail.

Entries in the database consist of one or more *value*`,`*type* pairs or simple *values.* A simple *value* declares the name of some entity. An entry may contain no name or several, and different entries containing the same name need not refer to the same entity.

These types are used:

`dk`	*Value* is a Datakit address.
`in`	*Value* is a numeric IP address.
`dom`	*Value* is an internet domain name.
`tel`	*Value* is a telephone number, possibly prefixed by a *uucp* `Dialcodes` name.
`org`	*Value* is an organization name.
`svc`	*Value* names a service.
`origin`	*Value* must be `local`, for sorting by ‘distance’; see below.

The following entries describe an entity `research` with a Datakit address, an IP address and domain name, belonging to organization `att`, and offering the `uucp` service:

```
192.11.4.55,in research research.astro.nj.att.com.,dom att,org
research nj/astro/research,dk uucp,svc att,org
```

Qns prints database entries that match *requests.* If a *request* is supplied on the command line, *qns* prints the answer and exits; otherwise it reads and answers requests from the standard input until end-of-file. The possible requests are:

`set` *key*`...`
: Print every entry matching all *keys*.

`value` *tlist key*`...`
: Examine entries matching the *keys* until a pair with type *tlist* is found; print the matching value and stop. *Tlist* may be a single type, or several separated by `|`.

`reset` Cause the name server to reinitialize its database.

`help` Print a list of requests.

`quit` Exit *qns*.

A *key* is a *value*`,`*type* pair; an entry matches if it contains that pair. If `,`*type* is omitted, any pair with the specified *value* will do. A `*` at the end of a *value* stands for an arbitrary suffix.

When a database search returns several entries with `dk` or `dom` types, and the database contains an entry with the conventional pair `local,origin`, the entries are sorted by increasing ‘distance’ from the `dk` or `dom` pairs in the `local,origin` entry. Datakit names in the same exchange are nearer than names in different exchanges in the same area, which are nearer than names in different areas. Domain names matching to four levels of domain hierarchy are nearer than names matching to only three levels, and so on.

Qns expects to reach the name server *ns*(8); option `-n` points it at service *server* instead.

SEE ALSO

ipc(3), *ns*(8)

NAME

checknews, readnews − read netnews articles

SYNOPSIS

`readnews` [`-a` *date*] [`-n` *newsgroup ...*] [`-t` *title ...*] [`-lprxhfuM`] [`-c` [*command*]]

`readnews -s`

`checknews` [`ynqve`] [*readnews-options*]

DESCRIPTION

Readnews prints unread articles that have arrived via the informal, worldwide 'netnews' network. Without arguments it prints unread articles from newsgroups to which you subscribe. The options are:

`-M` An interface to *Mail*(A). A *mail*(1)-like interface.

`-c` *command*
: Articles are written to a temporary 'mailbox' and the *command* (e.g. `mail -f %`) invoked, with the mailbox in place of `%`. A missing *command* gets something like *mail*(1).

`-p` Articles are sent to the standard output, no questions asked.

`-l` Titles only. The file `.newsrc` will not be updated.

`-r` Print articles in reverse order.

`-f` No followup articles.

`-h` Printed in a briefer format.

`-u` Update file `.newsrc` every 5 minutes.

`-n` *newsgroup ...*
: Select articles that belong to *newsgroups.*

`-t` *titles* Select articles whose titles contain one of the *title* strings.

`-a` *date* Select articles that were posted since *date*; a missing *date* means the beginning of time.

`-x` Ignore `.newsrc`; select previously read as well as unread articles.

`-s` Print subscription list.

The file `$HOME/.newsrc`, or a file specified by environment variable `NEWSRC`, tells what topics you are interested in and what you have read. If `.newsrc` contains a line starting with options (left justified, continued by trailing `\`), or if the environment variable `NEWSOPTS` is present, options are taken from there as well as the command line. In case of conflict, an option on the command line take precedence, followed by `.newsrc` and finally `NEWSOPTS`.

Readnews invokes some other programs to perform services. To reply to a news item it uses *mail*(1) or an alternate in environment parameter `MAILER`. It paginates with *p*(1), or an alternate in `PAGER`. `PAGER` is a command, perhaps containing `%` as in option `-c`, or empty for no pagination.

The default and *mail* interfaces support the following commands, and prompt with common alternatives: a newline accepts the first one. For example, `[ynq]` proposes yes, no, and quit; newline gets yes.

`y` Yes. Print current article and go on to next.

`n` No. Skip the current article. (In *mail* interface, it means `y`.)

`q` Quit; update `.newsrc`.

`c` Cancel the article. Only the author or the super-user can do this.

`r` Reply. Reply to article's author via mail. You are placed in *mail*, or an alternate in environment parameter `EDITOR`, with a header constructed from the article. You may change or add headers. Add your reply after the blank line. Upon exit the message is mailed.

`rd` Reply directly. You are placed in `MAILER` (*mail* by default). Type the text of the reply and then control-D.

`f` *title* Submit a followup article. If you omit the title, *readnews* generates an appropriate one. You will be placed in your `EDITOR` to compose the followup.

`fd` Follow up directly. This is like `f`, but does not construct headers.

`N` *newsgroup*
: Go to the named *newsgroup*, or the next newsgroup if none is named

`s` *file* Save. Append the article to *file*. The default is `Articles`. If *file* is not a full pathname, it is taken relative to `HOME`, overridden by environment parameter `NEWSBOX`. If the first character of *file* is `|`, the rest is taken as the name of a program, into which the article is piped.

`#` Report the name and size of the newsgroup.

`e` Erase. Forget that this article was read.

`h` Print a more verbose header.

`H` Print a very verbose, complete header.

`U` Unsubscribe from this newsgroup and go on to the next.

`d` Read a digest. Presents a digest as separate articles.

`D` *number*
Decrypt a Caesar cipher (usually used to obscure off-color material in `net.jokes`). The rotation is normally determined line-by-line from character frequencies. If this fails, an explicit *number* (usually 13) may be given.

`v` Print the current version of the news software.

`!` Shell escape.

number Go to article *number*.

±*n* Skip *n* articles, 1 by default. The articles skipped are recorded as 'unread'.

`-` Go back to last article. This is a toggle, typing it twice returns you to the original article.

`x` Exit. Like `q` except that `.newsrc` is not updated.

`X` *system*
Transmit article to the named system.

A `-` following `c`, `f`, `fd`, `r`, `rd`, `e`, `h`, `H`, or `s` refers to the previous article: `r -` is the normal way to reply to a just-read article when the next one is being offered.

Checknews reports whether there is news present, with options:

`y` Report only if news is present (default).

`n` Report only if news is absent.

`q` Turn off reports; nonzero exit status indicates news.

`v` Show the name of the first newsgroup containing unread news.

`vv` Explain any claim of new news, useful if checknews and readnews disagree.

`e` Execute *readnews* if there is news.

EXAMPLES

```
readnews
```
Read all unread articles.

```
readnews -n net.langs.c -a last thursday
```
Print every unread article about C since last Thursday.

```
readnews -p >/dev/null &
```
Discard all unread news: useful after returning from a long trip.

```
readnews -c "ed %" -l
```
Invoke *ed*(1) on a file containing the titles of all unread articles.

FILES

`/usr/spool/news/`*newsgroup*`/`*number*	News articles
`/usr/lib/news/active`	Active newsgroups and numbers of articles
`/usr/lib/news/help`	Help file for default interface
`$HOME/.newsrc`	

SEE ALSO

postnews(7), *Mail*(A)

BUGS

Readnews is baroque; many users prefer to browse among the files in `/usr/spool/news`.

NAME

scat − sky catalogue

SYNOPSIS

`scat`

DESCRIPTION

Scat looks up items in catalogues of objects outside the solar system. Items are read, one per line, from the standard input, and their descriptions or cross-index listings (suitable for input to *scat*) are printed on the standard output. An item is in one of the following formats:

`ngc1234`
: Number 1234 in the Revised New General Catalogue of Nonstellar Objects. The output identifies the type (`eg`=galaxy, `pn`=planetary nebula, `gc`=globular cluster, `oc`=open cluster, `dn`=diffuse nebula or `nc`=nebular cluster), possibly contained within the Large Magellanic Cloud (`in lmc`) or Small Magellanic Cloud (`in smc`), its position in 2000.0 coordinates and galactic coordinates, and a brief description.

`sao12345`
: Number 12345 in the Smithsonian Astrophysical Star Catalogue. Output identifies the visual and photographic magnitudes, 2000.0 coordinates, proper motion, spectral type, multiplicity and variability class, and HD number.

`m4` Catalog number 4 in Messier's catalog. The output is the NGC number.

`"alpha umi"`
: Star names are provided in double quotes. Known names are the Greek letter designations, proper names such as Betelgeuse, and bright variable stars. Constellation names must be the three-letter abbreviations. The output is the SAO number. For non-Greek names, SAO numbers and names are listed for all stars with names for which the given name is a prefix.

`12h34m −16`
: Coordinates in the sky are translated to the nearest 'patch', approximately one square degree of sky. The output is the coordinates identifying the patch, the constellations touching the patch, and the NGC and SAO objects in the patch.

`c umi` Gives voluminous output consisting of the patches covering the named constellation.

FILES

`/n/kwee/usr/rob/sky2/*.sky`

SEE ALSO

sky(7)
`/n/kwee/usr/rob/sky2/constelnames` for the three-letter abbreviations of the constellation names.

BUGS

The database is fine, but the program is feeble.
Coordinates printed by the program in the listings for SAO and NGC objects are not understood as input.

NAME

sky − astronomical ephemeris

SYNOPSIS

`sky [ -l ]`

DESCRIPTION

Sky predicts the apparent locations of the sun, moon, visible planets, and stars brighter than magnitude 2.5. It reads one line from the standard input to obtain the desired time expressed as five numbers: year, month, day, hour, and minute in GMT. An empty line means now. Each object is printed with astronomical coordinates, azimuth-elevation coordinates relative to Murray Hill, NJ, and magnitude. For variable stars the maximum magnitude is printed with `*`.

Option `-l` causes *sky* to prompt for another viewing location.

Standard astronomical effects are accounted for: nutation and precession of the equinox, annual aberration, diurnal parallax, and proper motion. Atmospheric effects (extinction and refraction) are not calculated, nor is perturbation of the earth by other bodies.

In ephemeris (slighly different from civil) time, the program yields positions of sun, moon, and stars good to a few tenths of an arc-second. Planets are good to a few seconds.

FILES

`/usr/lib/startab`

SEE ALSO

scat(7)
American Ephemeris and Nautical Almanac and *Explanatory Supplement to the American Ephemeris and Nautical Almanac*

NAME

tel, telno, dq − phone books

SYNOPSIS

`tel` *key*

`telno` *key* ...

`dq`

DESCRIPTION

Tel looks up *key* in a private telephone book, `$HOME/lib/tel`, if it exists, and in a local book `/usr/lib/tel`.

Tel is simply a script for *grep* (see *gre*(1)), so lookup may be by any part of a name or number. Customarily the local telephone book contains names, userids, home numbers, and office numbers of users. It also contains a directory of area codes and miscellaneous people of some general interest.

Telno and *dq* retrieve information from a central, current database of AT&T personnel. The former is a simple one-line interface to the latter.

For *telno,* a *key* is typically a name. Other fields in the database may be matched by using tabs within the key (the key must be quoted to protect spaces and tabs from the shell). The fields are, in order:

name (last; last,first; first last; or payroll account no.)
organization (e.g. att or bl)
area code and exchange
extension
location
room
department
electronic mail address

Only a prefix of a first name need by given. In some positions `*` (or `...`) and `?` may be used as a wild cards as in *sh*(1). Case is unimportant.

Dq provides interactive access to the central database. Under *mux*(9.1) it is best run in a terminal-simulator layer, either `term 5620` or `term 2621`; see *term*(9.1). It displays a form with the same fields as *telno.* Fill in one or more fields, using tabs to move (circularly) between them, backspace to erase, and space to delete a field. Upon carriage return a group of answers is displayed. Type `?` instead of a name to get more instructions.

EXAMPLES

In these examples, `\t` represents a literal tab.

`telno emlin`
Anyone named Emlin.

`telno '\t\t\t3744'`
Anyone whose phone extension is 3744.

`telno 'e*\t\t\t\t\t\t\tresearch!*'`
Anyone whose last name begins with E and who gets electronic mail on machine `research`.

FILES

`$HOME/lib/tel`
`/usr/lib/tel`

SEE ALSO

bellcat in *library*(7), *pq*(7)

BUGS

The algorithm used by *dq* (and hence *telno*) to match each field is distinct and unknown to us.
Not all entries have electronic mail addresses.

NAME

town − gazetteer of US places

SYNOPSIS

`town` [*place*]

DESCRIPTION

Town produces information about the *place*, which is the name of a US town, possibly followed by a comma and a two-letter state abbreviation. If no *place* is given, place names are read one per line from the standard input.

The information produced is latitude, longitude, approximate population, a forecast from *weather*(7), and a recent item from *apnews*(7).

FILES

`/usr/spool/town/ustown*` gazetteer in *cbt*(1) format

SEE ALSO

av(A), *dict*(7)

NAME

units − conversion program

SYNOPSIS

`units`

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
* 2.54
/ 0.393701
```

Quantities are specified using the following grammar:

Unit: *Empty* | *Unit Term* | *Unit* `/` *Term*

Term: *Number* | *Name* | `(` *Unit* `)` |
`square` *Term* | `sq` *Term* | `cubic` *Term* | `cu` *Term* |
Term `^` *Number*

Numbers are specified in the form expected by *atof*(3). Names are maximal strings of non-numeric, non-punctuation characters. Powers are indicated by the `^` operator or by the words `square` (`sq`) and `cubic` (`cu`). Parentheses alter grouping. The empty unit has value 1. Terms are multiplied together unless connected by `/` for inversion, e.g. `15 pounds force/sq in`.

Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

`pi`	ratio of circumference to diameter
`c`	speed of light
`e`	charge on an electron
`g`	acceleration of gravity
`force`	same as `g`
`mole`	Avogadro's number
`water`	pressure head per unit height of water
`au`	astronomical unit

The `pound` is a unit of mass. Compound names are run together, e.g. `lightyear`. British units that differ from their US counterparts are prefixed thus: `brgallon`. Currency is denoted `belgiumfranc`, `britainpound`, etc.

A response of `?` to 'You want:' displays all known units conformable with the 'You have:' quantity.

The complete list of units can be found in `/usr/lib/Units` and `/usr/lib/Monetary.Units`.

FILES

```
/usr/lib/Units
/usr/lib/Monetary.Units
/usr/lib/Units.bin
```

BUGS

Since *units* does only multiplicative scale changes, it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit.

Currency conversions are only as accurate as the most recent report of foreign exchange prices from the AP wire.

NAME

weather − conditions and forecast by town

SYNOPSIS

`weather [ -h ] [ -m ]`

DESCRIPTION

Weather reports current weather conditions and a forecast for towns in the contiguous United States. It reads town names from the standard input, one per line. Each input line should be in the style of

```
walla walla, wa
```

where the state is given as the 2-letter Post Office abbreviation.

The information provided is the temperature, humidity, whether or not it is raining or snowing, and an indication of cloudiness and visibility if it is not. If the wind is above 10 knots the wind speed and direction are given. Up to three forecasts are given (assuming they can be found within a 4×4 degree latitude-longitude square): a forecast of high and low temperatures plus probability of precipitation taken from some numerical model of the atmosphere, a general area forecast and a marine forecast (if near the coast and if the `-m` argument is given).

The command `weather -h` is equivalent to `weather` followed by a single input line of `murray hill, nj`.

SEE ALSO

avw(A)

8 MAINTENANCE

8 MAINTENANCE

NAME

11as, 11cc, 11ld, 11nm, 11ranlib, 11reloc, 11size, 11strip − pdp11 support

DESCRIPTION

These programs do cross-compiling and related support functions for the DEC PDP-11 family of computers. Their descriptions correspond closely with those of similarly named programs in Section 1 of this manual.

FILES

`11a.out` output file

NAME

ac − login accounting

SYNOPSIS

`/etc/ac` [*option* ...] [*person* ...]

DESCRIPTION

Ac prints the total connect time recorded in the accounting file, `/usr/adm/wtmp`. If *persons* are named, only those login names are considered. The options are

`−w` *file* Use *file* instead of `/usr/adm/wtmp`; − means the standard input.

`−p` Print total connect time for each user.

`−d` Print totals for each day.

The accounting file `/usr/adm/wtmp` is maintained by *init* and *login*(8), provided it exists. To start accounting, create it with length 0. When accounting is turned on, the file grows without limit. It is prudent periodically to process the data and truncate the file.

FILES

`/usr/adm/wtmp`

SEE ALSO

init(8), *sa*(8), *login*(8), *utmp*(5), *who*(1)

NAME

arff − read RT11 files

SYNOPSIS

`arff` [*key*] [*name* ...]

DESCRIPTION

Arff reads and writes files on an RT11 volume, such as the VAX-11/780 console floppy or 11/750 console tape. Its actions are controlled by the *key* argument, in the manner of *ar*(1). Other arguments are the names of files to be read or written.

The RT11 filesystem restricts names to the character set `a-z0-9.`, and there are no subdirectories. Filename arguments are trimmed to the last pathname element and mapped to lower case.

The key must contain one of the letters `rxdt`, which mean the same as in *ar*. Unlike *ar*, however, *arff* may reorder files to fit without moving existing files.

Other key letters are

v Verbose.

`f` The next argument is the name of the archive instead of the default `/dev/floppy`.

`m` Omit the DEC-standard sector interleaving algorithm; needed when the file isn't a floppy; e.g. the 11/750 console cassette.

FILES

`/dev/floppy`

BUGS

Floppy errors are handled ungracefully.
The program is too floppy-dependent. Mapping belongs in the device driver, or at least shouldn't be the default.

NAME

asd − automatic software distribution

SYNOPSIS

```
/usr/lib/asd/cdaemon
/usr/lib/asd/dkinstall
/usr/lib/asd/mkspool
/usr/lib/asd/rmlocks
/usr/lib/asd/udaemon
```

DESCRIPTION

These programs constitute the innards of the automatic software distribution system invoked by *ship*(8).

Mkspool creates an ASD spool directory for its invoker (if necessary), puts a file named `dummy` in the directory to prevent *cdaemon* from deleting it, and prints the directory's name.

Udaemon examines its invoker's ASD spool directory in lexical order for things to do. To forestall multiple daemons, it first makes an empty file named `L.`*pid* in the spool directory and tries to link to it a file `lock`. If the link fails, `L.`*pid* is removed and *udaemon* exits.

Shipping instructions appear in pairs of files with `.s` and `.d` suffixes. *Udaemon* examines each status (`.s`) file for destination names and places a network call to send the corresponding data (`.d`) file to *dkinstall* at each destination. *Dkinstall* verifies the data and gives it to *inspkg*, see *mkpkg*(8).

A status file contains zero or more entries, each of which has one or more lines. The first line of an entry is a network address, with default network `dk` and default service `asd`, possibly followed by a blank and a failure report. An entry with destination `#` is a comment and is ignored. Lines after the first begin with a tab, and contain output generated by the entry. If an entry has output, it is considered complete and will be processed no further.

Udaemon scans each status file once, and attempts to send the corresponding data file to the destination for each incomplete entry. If the attempt fails, a failure report is appended to the entry, and it remains incomplete. If the attempt succeeds, or the failure is severe, an error message or output from *dkinstall* is appended to the entry, which makes the entry complete. If an entry is completed successfully but there are no output lines, the entry is deleted.

Each instance of *udaemon* remembers which network addresses failed with non-severe errors; entries with the same address in later status files are given failure status 'deferred for sequence.'

If at least one additional entry was completed, *udaemon* sends the new status file to the owner by *mail*(1) after the whole file has been scanned. If no incomplete entries remain, both status and data files are removed.

Cdaemon examines every subdirectory of `/usr/spool/asd`, deletes empty directories and, impersonating the owner, invokes *udaemon* for nonempty directories. *Cdaemon* should be run regularly by *cron*(8) with super-user permissions.

Rmlocks removes all lock files in subdirectories of `/usr/spool/asd`. It should be called from *rc*(8).

Entries in an ASD spool directory may be made without regard to locks provided that everything is done in the right sequence: (1) call *mkspool*; (2) create the data file; (3) create the status file under a temporary name; (4) rename the status file to end with `.s`; (5) remove the `dummy` file, if present; and (6) call *udaemon*.

Because *mail* will not send an empty file, a status file must have a comment entry if acknowledgment is desired after a successful *udaemon* run.

FILES

`/usr/spool/asd/`*userid* user's ASD spool directory

SEE ALSO

mkpkg(8), *ship*(8)

NAME

backup − backup client administration

SYNOPSIS

`/usr/lib/backup/sel`

`/usr/lib/backup/fcheck` *maxsize maxdays files ...*

`/usr/lib/backup/act` [*stat*]

DESCRIPTION

These programs select and back up files to the incremental file backup system, *backup*(1).

Sel prints on the standard output a list of filenames that might need to be backed up. The initial version picks out files that have been changed in the past few days, skipping huge files and eliding boring names like `core`. *Sel* is a shell script; the local administrator is expected to customize it.

Fcheck is a fast, specialized file scanning program, used by *sel*. It examines each of the *files*, descending into directories, and prints the name of each file that has been changed in the last *maxdays* days and is smaller than *maxsize* kilobytes. Symbolic links are followed when presented as arguments, examined but not followed otherwise.

Act reads a list of filenames from the standard input. It searches the backup database *backup*(5) for the current version of each file, and backs up files that aren't registered.

By default, *act* sends errors by *mail*(1) to user `backup`. If the *stat* argument is non-empty, errors and additional comforting chatter are printed on the standard output instead.

One way to request automatic backups is to run

```
/usr/lib/backup/sel | /usr/lib/backup/act
```

regularly from *cron*(8). Particular files may be backed up by hand at any time by running *act* with a list of filenames. There are no special permissions involved; any user may run *act*.

SEE ALSO

backup(1), *backup*(5)

A. Hume, 'The File Motel: An Owner's Manual', this manual, Volume 2

NAME

chown − change owner or group

SYNOPSIS

`/etc/chown` *owner*`,`*group file ...*

DESCRIPTION

Chown changes the owner of the *files* to *owner* and the groupid to *group*. Either *owner* or *group* may be omitted to leave the owner or groupid unchanged.

Owner may be either a decimal userid or a login name found in `/etc/passwd`. *Group* may be either a decimal groupid or a group name found in `/etc/group`.

The owner of a file may change its group to that of the current process. Other changes are restricted to the super-user.

FILES

`/etc/passwd`
`/etc/group`

SEE ALSO

chown in *chmod*(2), *passwd*(5), *chmod*(1), *chdate*(1)

NAME

chuck − a file system checking program

SYNOPSIS

`/etc/chuck` [*option ...*] *device*

`/etc/chuck` `−M` *blocks device*

`/etc/upchuck` [`−w`] [`−p` *program*]

DESCRIPTION

Chuck checks and optionally repairs the file system on the named *device*. It is normally invoked by *upchuck* by *rc*(8) during reboots. The `−w` flag to *upchuck* is passed on to *program*. If *program* is not present, the default is `/etc/chuck`. (Try, as super-user, `/etc/upchuck −p /bin/echo` to see the normal arguments to *chuck*.) If *upchuck* can read the raw version of *device*, it will, except for the root file system.

The options are

`−w` Try to do standard repairs.

`−b` *blocksize*
Specify file system block size; default is 4096.

`−i` Interactive. Ask approval for each change.

`−I` *inode ...*
`−B` *block ...*
Report on inodes or blocks specified by number.

`−v` Verbose. Give more commentary.

Chuck can also make a new file system: `chuck −M` *size device* makes a bitmapped file system (only) of *size* 4096-byte blocks on *device*. It asks approval before writing.

FILES

`/etc/fstab`

SEE ALSO

fstab(5), *filsys*(5), *fsck*(8), *reboot*(8)

BUGS

Chuck does not replace real expertise. It will not automatically repair a file system with duplicate blocks. In complicated situations it may have to be run several times to get complete consistency.
It will not recover from I/O errors in reading the inodes, and does not yet extend `lost+found` when necessary.
It uses memory freely (about 12 bytes per file system block and 84 bytes per inode).
Chuck is still experimental, and acts the part. Error messages are usually inscrutable.
It believes even preposterous super-blocks and consequently can get core images.

NAME

clri − clear inode

SYNOPSIS

`/etc/clri` *special i-number ...*

DESCRIPTION

Clri writes zeros on the inodes with the decimal *i-numbers* on the file system in file *special*. After *clri*, any blocks in the affected file will show up as 'missing' in *icheck*(8).

The inode becomes allocatable.

The primary purpose of this program is to remove a file which for some reason appears in no directory. If it is used to clear an inode which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the inode is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated inode, so the cycle is likely to be repeated.

Clri is a last resort; normally *fsck*(8) can do the necessary repairs.

SEE ALSO

fsck(8), *icheck*(8)

BUGS

If the file is open, *clri* is likely to be ineffective.

NAME

cpp − C language preprocessor

SYNOPSIS

`/lib/cpp` [*option* ...] [*ifile* [*ofile*]]

DESCRIPTION

Cpp interprets preprocessor directives and does macro substitution for *cc*(1) and other compilers. The input *ifile* and output *ofile* default to standard input and standard output respectively.

The options are:

`−P` Do not place line number markings in output.

`−C` Do not remove comments.

`−U`*name*
`−D`*name*
`−D`*name*`=`*def*
`−I`*dir* Same as in *cc*(1). `−U` overrides `−D`.

`−H` Report all included files on standard error file,.

`−M` Attach modification date to file names in line number directives thus: *file*`@`*modtime*, where modtime is the integer number of seconds since the epoch.

`−T` Truncate preprocessor symbols to eight characters.

`−Y`*dir* Use *dir* instead of `/usr/include` as the last resort in searching for include files.

The output file contains processed text sprinkled with lines that show the original input line numbering:

`#` *linenumber* `"`*ifile*`"`

The input language is as described in the reference, with a few additions:

The `#` *linenumber* marks placed in the output are accepted as an alternative to the official `#line` directive.

These symbols are predefined in various implementations:

```
ibm gcos os tss unix
interdata pdp11 u370 u3b u3b5 vax
RES RT
lint
```

Preprocessor formal parameters are recognized within quoted strings in the replacement text.

When comments are removed they are replaced by null strings; this unofficial feature makes it possible to construct identifiers by concatenation.

FILES

`/usr/include` standard directory for include files

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1988

NAME

cron − clock daemon

SYNOPSIS

`/etc/cron`

DESCRIPTION

Cron executes commands at specified dates and times according to the instructions in the file `/etc/crontab`. It should be run once from *rc*(8).

Crontab entries are lines of seven fields separated by blanks or tabs:

login minute hour day month weekday command

Login is the login name under whose userid and groupid the command should be executed. The next five fields are integer patterns for

minute	0-59
hour	0-23
day of month	1-31
month of year	1-12
day of week	0-6; 0=Sunday

Each pattern may contain a number in the range above; two numbers separated by a hyphen meaning an inclusive range; a list of numbers separated by commas meaning any of the numbers; or an asterisk meaning all legal values.

The last field is a string to be executed by the shell, after replacing `%` with newline.

It is wise to spread the times of activities to avoid bogging the system down at favorite hours.

Cron examines *crontab* every minute.

EXAMPLES

```
daemon 17 3  *     *  *   /usr/bin/calendar − # reminders at 0317
adm    15 4  1,15  *  *   ac −p | mail adm # bimonthly accounts
root 0  12 22−28 11 4   /etc/wall Time for Thanksgiving dinner
```

FILES

`/etc/crontab`

SEE ALSO

at(1)

BUGS

The behavior of `%` in *crontab* entries is nonstandard. Strings following `%` should be delivered to the command as standard input.

NAME

dkhup, dkmgr, dkzap − manage Datakit interface

SYNOPSIS

`/usr/ipc/mgrs/dkhup` [`−N` *prefix*]
`/usr/ipc/mgrs/dkmgr` [`−N` *prefix*] [`−m` *outname*] [`−n` *service*]
`/usr/ipc/mgrs/dkzap` [`−N` *prefix*]

DESCRIPTION

Dkhup starts the common signaling channel protocol for a Datakit interface. Initially it sends several reset messages, and tells the controller to hang up all outstanding calls; thereafter it simply keeps the signaling protocol running.

Dkmgr receives and places Datakit calls. Outbound calls may be placed by calling *ipcopen* (*ipc*(3)) with the *outname* specified by option `−m`; the default is `dk`. *Dkmgr* announces itself to the Datakit network with the *service* name specified by option `−n`; the default is taken from `/etc/whoami`. Inbound calls to *service* are connected to the local `login` service; inbound calls to Datakit address *service* **.** *serv* are handed to local service *serv*.

Dkhup and *dkmgr* are normally run once from *rc*(8). Both programs must be running to make the network available.

Dkzap arranges for a KMC11-assisted Datakit interface to be reset, reloaded, and restarted. This should be done only if things are badly broken, as it hangs up all existing calls through that interface.

Datakit devices are expected to have names of the form `/dev/dk/dk`*nn* with *nn* a two-digit channel number. If there are more than 100 channels, the first digit overflows to lower-case letters: channel 100 is `a0`. The common signaling control channel is named `/dev/dk/dkctl`. All three programs accept an option `−N` *prefix* to change the naming convention; for example, `−N /dev/kb/kb` means that the files have names like `/dev/kb/kb32` and `/dev/kb/kbctl`.

Support also exists for a less general naming convention: there may be two sets of files, named `/dev/dk/dk0`*nn* and `/dev/dk/dk2`*nn*, with control channels `/dev/dk/dkctl0` and `/dev/dk/dkctl2`. *Dkhup* and *dkzap* take the extra argument `0` or `2` to point at one or the other of the control names. A separate copy of *dkhup* is needed for each name. *Dkmgr* takes an option `−u` *c*, where *c* is `0` or `2` to use one set of files, or `b` to use both simultaneously; in the latter case, *service* is announced to both networks. This scheme is obsolete and overdue for replacement; the missing piece is something to pick an interface for outcalls.

Dkmgr records its activity in file *service* in directory `/usr/ipc/log`, default `/usr/ipc/log/dk`.

FILES

`/dev/dk`

SEE ALSO

con(1), *kmc*(8), *svcmgr*(8), *ipc*(3)

BUGS

Dkhup should be folded into *dkmgr*; it is separate for historic reasons.

NAME

dmesg − system diagnostic messages

SYNOPSIS

`/etc/dmesg` [`−`] [`−i`] [*core namelist*]

DESCRIPTION

Dmesg looks in a system buffer for recent console messages from the operating system and reproduces them on the standard output. Under option −, *dmesg* produces only those messages printed by the system since the last time `dmesg −` was run. It is normally run periodically by *cron*(8) to produce the error log `/usr/adm/messages`.

Option `−i` prints messages produced since the last `dmesg −` without changing any records.

If *core* and *namelist* are specified, they are used in place of `/dev/kmem` and `/unix`.

FILES

`/usr/adm/messages`	error log
`/usr/adm/msgbuf`	record of option −

BUGS

Since the system error message buffer is small, not all error messages are guaranteed to be logged. Error messages generated immediately before a system crash may not be logged.

NAME

fsck − file system consistency check and interactive repair

SYNOPSIS

`/etc/fsck -p` [*special* ...]
`/etc/fsck` [`-y`] [`-n`] [`-s`*X*] [`-S`*X*] [`-t` *filename*] [*special* ...]

DESCRIPTION

Fsck inspects the disk filesystems in the named *special* files and repairs inconsistencies. If no files are named, every file system listed in *fstab*(5) with type 0 and a nonzero pass number is checked.

Under option `-p`, *fsck* runs without intervention, repairing minor inconsistencies and aborting on major ones. This form is usually called from *rc*(8). If no special files are named, file systems in *fstab* are checked in parallel passes: all file systems with pass number 1 are checked simultaneously, then all file systems with pass number 2, and so on until *fstab* is exhausted.

Here are the minor ailments repaired automatically under `-p`:

unreferenced inodes;
wrong link counts in inodes;
missing blocks in the free list;
blocks in the free list also in files; and
counts wrong in the super-block.

Other inconsistencies cause *fsck* to abandon the inconsistent file system, and exit with a nonzero status when the current pass finishes.

Without the `-p` option, *fsck* inspects one file system at a time, interactively. Each inconsistency causes *fsck* to print a message and ask permission to fix the problem. The operator may require arcane knowledge to guide *fsck* safely through repair of a badly damaged file system.

Here are the remaining options. They are allowed only if `-p` is absent.

`-y` Assume a yes response to all questions. This should be used with great caution.

`-n` Assume a no response to all questions; do not open the file system for writing. This option is assumed if the file system cannot be opened for writing.

`-s`*X* Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system. If the file system has a bitmap free list (see *filsys*(5)), the free list is always reconstructed unless the `-n` option is enabled.

Parameter *X* allows free-list parameters to be specified: `-s`*blocks-per-cylinder*`:`*blocks-to-skip*. If *X* is not given, the values used when the file system was created are used; see *mkfs*(8). If these values were not specified, *X* is assumed to be `400:9`.

`-S`*X* Conditionally reconstruct the free list. This option is like `-s`*X* except that the free list is rebuilt only if no discrepancies were found. `-S` implies `-n`.

`-t` If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the `-t` option is specified, the file named in the next argument is used as the scratch file, if needed. Without `-t`, *fsck* will prompt the operator for the name of the scratch file. The file chosen should not be on the file system being checked. If it did not already exist, it is removed when *fsck* completes.

Inconsistencies checked are:

Blocks claimed more than once.
Blocks designated outside the file system.
Incorrect link counts.
Directory size not 16-byte aligned.
Bad inode format.
Blocks not accounted for anywhere.
Directory entry pointing to unallocated inode.
Inode number out of range.
More than 65536 inodes.
More blocks for inodes than there are in the file system.
Bad free block list format.
Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are reconnected by placing them in the directory `lost+found` in the root of the file system being checked. The name assigned is the inode number, prefixed by `#`.

Checking the raw device is almost always faster, but *fsck* distinguishes bitmapped from non-bitmapped file systems by examining the minor device number, so the block device is safer.

FILES

`/etc/fstab`

SEE ALSO

fstab(5), *filsys*(5), *mkfs*(8), *reboot*(8)
T. J. Kowalski, 'Fsck—the UNIX File System Check Program', this manual, Volume 2

BUGS

Inode numbers for `.` and `..` in each directory should be checked for validity.

Some systems save core images after a crash in the swap area; on such machines, checking many large file systems in parallel may cause swapping, overwriting the crash dump. It is best just to write crash dumps in a a safer place. If disk space for dumps and swapping is scarce, avoid checking more than three 120-megabyte file systems in parallel on a machine with four megabytes of physical memory.

Examining the minor device number is a botch; there should be an explicit flag somewhere.

Fsck does not have supernatural powers.

NAME

getty − set terminal mode

SYNOPSIS

`/etc/getty` [*char*]

DESCRIPTION

Getty is invoked by *init*(8) after a terminal is opened. While reading the user's name *getty* attempts to adapt the system to the speed and type of terminal being used.

Init calls *getty* with an argument specified by the *ttys*(5) entry for the terminal line. The argument *char* determines the line speed and other characteristics; see below. *Getty* then types a banner identifying the system (from `/etc/whoami`) and the `login:` message. The user's name is then read, a character at a time. If a null character is received, it is assumed to be the result of the user pushing the 'break' ('interrupt') key. The speed is then changed to another baud rate and `login:` is typed again. Successive breaks cycle through a set of speeds.

The user's name is terminated by a newline or carriage return. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(2)).

Finally, *login*(8) is called with the user's name as argument.

Here are the possible values for *char*. If more than one speed is given, the first speed is used initially; others are selected by successive depressions of the BREAK key. Other things, like delays and tab expansion, are set in various ways as well.

char	speed
`0`	300-1200-150-110
`2`	9600
`3`	1200-300
`4`	300
`5`	300-1200
`6`	2400
`7`	4800
`8`	9600-1200-300
`9`	300-9600-1200
`a`	2400-1200
`b`	1200-2400
`j`	exta (usually 19200)

SEE ALSO

init(8), *login*(8), *ioctl*(2), *ttys*(5)

NAME

icheck, dcheck, ncheck − file system consistency check

SYNOPSIS

`/etc/icheck` [option ...] *filesystem* ...

`/etc/dcheck` [option ...] *filesystem* ...

`/etc/ncheck` [option ...] *filesystem* ...

DESCRIPTION

These programs perform consistency checks on file systems. For normal file system maintenance, see *fsck*(8). Common options are

`−B` The file system is bitmapped. If *filesystem* is a special file, this option is set automatically from the minor device number.

`−i` *number ...*
Report only on specified inode *numbers* (*dcheck* and *ncheck* only).

Icheck examines each *filesystem*, builds a list of used blocks, and compares this list against the free list maintained on the file system. The normal output of *icheck* includes a report of

The total number of files and the numbers of regular, directory, block special and character special files.

The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.

The number of free blocks.

The number of blocks missing; *i.e.* not in any file nor in the free list.

Other *icheck* options are

`−s` Ignore the free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent. The words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The normal output reports are suppressed.

`−b` *number ...*
Report each appearance of the selected block *numbers* in a file or on the free list.

`−d` Report each duplicate block.

`−m` Report each missing block.

`−e` Print at most one diagnostic per file; useful for badly curdled file systems.

Dcheck reads the directories in each *filesystem* and compares the link count in each inode with the number of directory entries by which it is referenced.

Ncheck generates a list of pathname vs i-number for each named *filesystem*. Other *ncheck* options are

`−a` Report `.` and `..`, which are normally ignored.

`−s` Report only special files, and files with set-userid or set-groupid mode; helpful in finding security breaches.

SEE ALSO

filsys(5), *chuck*(8), *fsck*(8), *clri*(8)

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) *icheck* announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the

bad block is printed and *icheck* considers it to contain 0. 'Bad freeblock' means that a block number outside the available space was encountered in the free list. 'Dups in free' means that blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

When a file turns up for which the link-count and the number of directory entries disagree, *dcheck* reports the relevant facts. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

When the filesystem structure is improper, *ncheck* prints `??` to denote the 'parent' of a parentless file. A pathname beginning with `...` denotes a loop.

BUGS

Extraneous diagnostics may be produced if these commands are applied to active file systems.
They believe even preposterous super-blocks and consequently can get core images.
Ncheck's report is in no useful order, and probably should be sorted. *Ncheck* fails to report the root inode.

NAME

init − process control initialization

SYNOPSIS

`/etc/init`

DESCRIPTION

Init is invoked by the operating system as the last step in the boot procedure. It is always process 1.

When started normally, *init* calls *rc*(8) with parameter `autoboot`. If this succeeds, *init* begins multi-user operation. If *rc* fails, *init* commences single user operation by giving the super-user a shell on the console. It is possible to pass parameters from the boot program to *init* so that single user operation is commenced immediately. When the single user shell terminates, *init* runs *rc* without the parameter, and begins multi-user operation.

Rc performs housekeeping such as checking and mounting file systems and starting daemons; see *rc*(8).

In multi-user operation, *init*'s role is to create a process for each directly connected terminal port on which a user may log in. To begin such operations, it reads the *ttys*(5) file and forks to create a process for each terminal specified in the file. Each of these processes opens the appropriate terminal for reading and writing on file descriptors 0, 1, 2, and 3 (the standard input and output, the diagnostic output and `/dev/tty`). Opening the terminal will usually involve a delay, since the *open* is not completed until someone dials and carrier is established on the channel. Then *getty*(8) is called with argument as specified by the second character of the *ttys* file line. *Getty* reads the user's name and invokes *login*(8) to log in the user and execute the shell.

Ultimately the shell will terminate because of an end-of-file or as a result of hanging up. The main path of *init*, which has been waiting for such an event, wakes up and removes the appropriate entry from the file *utmp*(5), which records current users, and makes an entry in *wtmp*, which maintains a history of logins and logouts. Then the appropriate terminal is reopened and *getty* is invoked again.

Init catches signal `SIGHUP` and interprets it to mean that the *ttys* file should be read again. The shell process on each line that has become inactive according to *ttys* is terminated; a new process is created for each line added; lines unchanged in the file are undisturbed. Thus it is possible to drop or add terminal lines without rebooting the system by changing the *ttys* file and sending a *hangup* signal to the *init* process: use `kill -1 1`.

Init will terminate multi-user operations, kill all outstanding processes, and resume single-user mode if sent signal `SIGTERM`: use `kill 1`. *Init* will wait at most 30 seconds for outstanding processes to die, to avoid waiting forever.

If, at bootstrap time, the *init* program cannot be executed, the system will loop in user mode at a low address.

FILES

`/dev/console`
`/dev/tty`
`/etc/utmp`
`/usr/adm/wtmp`
`/etc/ttys`
`/etc/rc`

SEE ALSO

login(8), *kill*(1), *sh*(1), *ttys*(5), *getty*(8), *rc*(8), *reboot*(8)

BUGS

Init's multi-user functions should be integrated with the world of *svcmgr*(8).

NAME

ipconfig, dkipconfig, udpconfig − set up DARPA Internet protocols

SYNOPSIS

`/usr/ipc/mgrs/ipconfig` [`−m` *mask*] [`−df`] *ip-device localhost network* `[`*arp-device*`] &`
`/usr/ipc/mgrs/dkipconfig` *gatemachine localhost remotehost* `&`
`/usr/ipc/mgrs/udpconfig` *udp-device* `&`

DESCRIPTION

Ipconfig activates the DARPA Internet protocol on a communications device, with Internet address *localhost* for the host and network address *network* for the device.

If *arp-device* is specified, the ARP address resolution protocol is started on that device. Option `−d` causes *ipconfig* to print ARP requests on the standard output as they are received.

Option `−m` declares a subnet mask for the network reached through *ip-device*. *Mask* may be a four-piece IP address like `255.255.255.0` or a 32-bit hexadecimal number like `ffffff00`.

Option `−f` is a special workaround for networks with obsolete hosts. It causes *ipconfig* to answer illegal ARP requests for the subnet's broadcast address with an illegal Ethernet address, to prevent broadcast storms.

Dkipconfig places a network call to *gatemachine* and activates the IP protocol on the connection, so that the remote machine becomes a gateway for the caller's IP traffic. *Localhost* becomes the calling machine's Internet address through this IP interface; *remotehost* is the Internet address to which local IP packets should be sent to reach the gateway.

Ipconfig and *dkipconfig* record unusual events and errors in log files `/usr/ipc/log/ipconfig` and `/usr/ipc/log/dkipconfig`.

Udpconfig activates the UDP datagram protocol on the named *udp-device*, usually `/dev/ipudp`. Only one *udpconfig* is needed for the entire collection of IP networks.

These programs are usually run once from *rc*(8).

EXAMPLES

The following calls start IP on system `fs` on the first Interlan Ethernet controller, with ARP active; arrange for machine `nj/astro/research` to pass IP packets to `fs`; and activate UDP.

```
/usr/ipc/mgrs/ipconfig /dev/il00 fs mh-astro-net /dev/il01 &
/usr/ipc/mgrs/dkipconfig nj/astro/research fs-dk research-dk127 &
/usr/ipc/mgrs/udpconfig /dev/ipudp &
```

FILES

```
/usr/ipc/log/ipconfig
/usr/ipc/log/dkipconfig
```

SEE ALSO

con(1), *qns*(7), *route*(8), *tcpmgr*(8)

NAME

kmc, kdiload, kmcdump − control KMC11 input/output processors

SYNOPSIS

`/etc/kdiload` [*dev* [*file*]]

`/etc/kmcdump` [*dev*]

DESCRIPTION

These commands control the KMC11-B microprocessors used for Datakit protocol processing.

Kdiload resets KMC device *dev*, copies the microcode in *file* into the KMC's memory, and starts the KMC. *Dev* may be a pathname or a single character key identifying the KMC; the default is `2`. *File* defaults to that specified in `/etc/kmctab` if a single character key is used, `/etc/dkk.dubhi` otherwise.

Kmcdump stops the KMC and copies its state into files in the working directory. *Dev* may be a single character key or a pathname; the default is `2`. The KMC's memory is copied to the file `core.`*k*`.`*nnn*, where *k* is the keyletter and *nnn* is some number; the state of the KMC's registers and some trace information from Unix is written to `regs.`*k*`.`*nnn*.

These commands search the file `/etc/kmctab` for KMC devices and microcode files. The file contains lines of three blank-separated fields:

single character identifying this KMC
full pathname of the KMC device file
full pathname of the microcode to be used in this KMC

The KMC with key *K* uses Datakit special files with names like `/dev/dk/dk`*K*`03`. If there is only one KMC for Datakit, its key is `2`. If the only KMC is the only Datakit interface in a machine, its key is `2`, and its special files look like `/dev/dk/dk03`.

FILES

`/etc/kmctab`
`/bin/kasb`

SEE ALSO

dkmgr(8)

BUGS

For the moment, the only permissible keys are `0` and `2`. The KMC and Datakit filename conventions are arcane, and based on obsolete notions; they should be replaced.

NAME

ldpcs − load comet microcode

SYNOPSIS

`/etc/ldpcs` [`−f`] [`−v`] *pcsfile*

DESCRIPTION

Ldpcs loads microcode from *pcsfile* into the VAX-11/750 patchable control store. Normally, the hardware ID register is checked to see that the system is an 11/750 and that its base microcode revision level is appropriately high; the `−f` option removes the checks. The `−v` option causes the microcode version number to be printed after loading.

Ldpcs is usually called from *rc*(8) to load the most recent DEC microcode patches from `/etc/pcs750.bin`.

The patch file consists of 1024 bytes of patch bits, followed by 10240 bytes of actual patches. Each patch bit represents a 20-bit microcode word; the patches themselves are 20-bit words packed together. The format is the same as that distributed by DEC.

FILES

`/dev/mem`
`/dev/mtpr`

BUGS

Calling *ldpcs* is a good idea, but it is not mandatory; the system will run without the patches.

NAME

login − sign on

SYNOPSIS

`/etc/login` *name*
`/etc/login` `−f` *name* [*cmd*]
`/etc/login` `−p` *passwd-line* [*cmd*]

DESCRIPTION

Login is executed by *getty*(8). See the Introduction to this volume for how to dial up initially.

Login asks for a password if appropriate. Echoing is turned off during the typing of the password. The `−f` option forces login of the named user, without a password. `−p` is similar to `−f`, but an entire line of password file information is supplied.

Login initializes the userid, the groupid, and the working directory according to specifications found in the password file; see *passwd*(5). It also initializes environment variables PATH and HOME . Finally it executes a command interpreter (usually *sh*(1)). Argument 0 of the command interpreter is its name with a dash (−) prepended. If a *cmd* argument was present, two additional arguments `−c` *cmd* are passed, and environment variable REXEC is set to `1`.

Upon a successful login, accounting files are updated and, if no options are present, the message of the day is printed and the user is informed of the existence of mail.

Successful logins are recorded in `/etc/utmp` and `/usr/adm/wtmp`. If *cmd* was present, `*` is appended to the login name in *wtmp*, and no record is made in *utmp*.

Only the super-user may execute *login*.

FILES

`/etc/utmp`	accounting
`/usr/adm/wtmp`	accounting
`/usr/spool/mail/*`	mail
`/etc/motd`	message-of-the-day
`/etc/passwd`	password file
`/etc/group`	groups file

SEE ALSO

newgrp(1), *passwd*(1), *environ*(5), *passwd*(5), *getty*(8), *init*(8), *su*(8), *svcmgr*(8).

DIAGNOSTICS

‘Login incorrect’: the name or the password is bad.
‘No Shell’ or ‘no directory’: the initial shell or home directory specified in the password file does not exist.
‘Cannot open password file’: things are badly curdled.

BUGS

Information passed to options `−p` and `−f` is not checked. Only trusted programs should run *login*. Only trusted programs may usefully do so anyway; *login* has no privileges.

NAME

makekey − generate encryption key

SYNOPSIS

`/usr/lib/makekey`

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e. to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, upper- and lower-case letters, `.` and `/`. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

Makekey is intended for programs, such as *crypt*(1), that perform encryption. Usually its input and output will be pipes.

SEE ALSO

crypt(1)

NAME

mkfs, mkbitfs, mklost+found − construct a disk file system

SYNOPSIS

`/etc/mkfs` *special size* [*spacing cylsize*]

`/etc/mkbitfs` *special size* [*spacing cylsize*]

`/etc/mklost+found`

DESCRIPTION

Mkfs (*mkbitfs*) constructs a regular (bitmapped) file system on the block device *special*. *Size* is the number of 1KB (4KB) blocks on the special file. Values of *size* for various special files, e.g. *ra*(4), are given in Section 4. The new file system has a single empty directory and a number of inodes that depends on the size.

Spacing and *cylsize* are used for block allocation. *Spacing* is the preferred distance between successive blocks of a file; *cylsize* is the number of blocks the system can use without causing a disk seek. Suboptimal default values are built into the programs; some reasonable values are suggested below.

Mklost+found creates a directory called `lost+found` in the working directory with several empty slots. This directory is used as a repository for disconnected files recovered by *fsck*(8) and *chuck*(8).

EXAMPLES

```
/etc/mkbitfs /dev/ra03 31231
/etc/mount /dev/ra03 /mnt
cd /mnt
/etc/mklost+found
```

Notice the change of units between *ra*(4) and here: 31231 = 249848∗512/4096.

SEE ALSO

filsys(5), *fsck*(8)

BUGS

Fsck should make *lost+found* automatically when needed.

MACHINE DEPENDENCIES

Here are empirically good values for *spacing* and *cylsize* for VAX hardware. Minor changes, such as a new version of a controller, can invalidate them. To be certain, run your own experiments. Cylinder size doesn't seem to matter much on RA81 disks. On RA90s, it doesn't matter as long as it is large.

Bitmapped (4K) file systems

space	cyl	disk, controller, CPU
2	500	RA90, KDB50, VAX 8550
4	500	RA90, UDA50, VAX 8550
2	40	RA81, KDB50, VAX 8550
3	40	RA81, old KDB50, VAX 8550
3	40	RA81, UDA50, VAX 8550
3	105	Wren VI, Viking UDD, VAX 8550
4	40	RA81, UDA50, VAX-11/750
3	40	RA60, UDA50, VAX-11/750
3	40	RA81, KDA50, MicroVAX II
1	40	RD54, RQDX3, MicroVAX II
3	40	RA81, KDA50, MicroVAX III

Old-style (1K) file systems

space	cyl	disk, controller, CPU
5	357	RA81, KDB50, VAX 8550
7	357	RA81, UDA50, VAX-11/780
7	357	RA81, KDA50, MicroVAX II
7	128	RD54, RQDX3, MicroVAX II
6	357	RA81, KDA50, MicroVAX III

NAME

mknod − construct special file

SYNOPSIS

`/etc/mknod` *name* [`c`] [`b`] *major minor*

DESCRIPTION

Mknod makes a special file. The first argument is the *name* of the entry. The second is `b` to make a file for a block device, `c` to make a character device. The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g. unit, drive, or line number).

Minor devices are specific to each device driver; see the writeups in Section 4. The assignment of major device numbers varies with the implementation. Here are some conventional ones in the system used on VAXes:

Block devices

0 MASSBUS SMD disk
1 block TE16 tape, *mt*(4)
2 UNIBUS SMD disk
3 VAX 8550 console disk
7 MSCP disk, *ra*(4)
8 block TU78 tape, *mt*(4)
10 block TMSCP tape, *mt*(4)

Character devices

0 console terminal, *console*(4)
1 DZ11 serial interface
2 VAX-11/750 console tape
3 memory, null device, *mem*, *null*(4)
4 raw MASSBUS SMD disk
5 TE16 tape, *mt*(4)
7 interleaved swap devices, *drum*(4)
8 VAX-11/780 console floppy
12 raw VAX 8550 console disk
13 raw UNIBUS SMD disk
14 DEBNA Ethernet controller, *ethernet*(4)
17 DR11-C Datakit interface, *dk*(4)
22 TU78 tape, *mt*(4)
26 KMC11-B control interface, *kmc*(4)
28 raw MSCP disk, *ra*(4)
31 KMC11-B protocol-support Datakit interface, *dk*(4)
40 file descriptor device, *fd*(4)
42 IP protocol endpoints
43 TCP protocol endpoints
44 Interlan NI1010A Ethernet controller, *ethernet*(4)
50 UDP protocol endpoints
57 DHV11 serial line controller
58 DEQNA Ethernet controller, *ethernet*(4)
59 TMSCP tape, *mt*(4)

SEE ALSO

mknod(2)

NAME

mkpkg, inspkg, seal, unseal − package files for automatic software distribution

SYNOPSIS

`mkpkg` [*option ...*] *file ...*

`inspkg` [*option ...*] [*file ...*]

`seal` [*option*] [*file ...*]

`unseal` [*option*] [*file ...*]

DESCRIPTION

These programs are used by *ship*(8) to keep files identical across machines.

Mkpkg packages *files* and writes the result on the standard output. *Inspkg* installs the named packages or the standard input.

Non-existent files given to *mkpkg* are deleted upon installation. Directories are copied with all their contents. Hard links are reproduced. Symbolic links and special files are reproduced with the same inode contents. File modification and access times and owner and group names are reproduced as far as possible. Old versions of files are removed before installation: *inspkg* needs write permission in containing directories.

Options for both *mkpkg* and *inspkg*:

`−v` Place running commentary on the standard error file.

`−D`*path1*`=`*path2*
Pretend that any pathname beginning with *path1* really begins with *path2*. Relative pathnames are extended to full pathnames before comparison.

Options for *mkpkg*; only one may occur:

`−x`*command*
Include in the package instructions to execute the shell *command* after all files have been installed. *Command* is unaffected by option `−D`.

`−X`*file* Include in the package instructions to run the shell script *file* after all files have been installed. The file name is subject to option `−D`.

Options for *inspkg*:

`−n` Skip the actual installation, but verify the input packages and produce a backup if requested.

`−b` Write on the standard output a backup package that contains whatever was destroyed.

A package is an *ar*(1) archive containing an extra ASCII file named `Instructions`.

Seal concatenates the named *files* or the standard input onto the standard output in an error-detecting form suitable for shipment by *mail*(1). *Unseal* reverses the process, concatenating copies of all the original inputs onto the standard output. When *asd*(8) uses *uucp*(1), it sends sealed packages.

A sealed file is printable, has fewer than 128 characters per line, and has no lines consisting of a single period. The first line is `!<seal>` and the last one begins with `!end`. Other lines, such as mail headers, can be added to either end of a sealed file without hindering *unseal*.

Options for *seal* and *unseal*:

`−k` A key will be demanded to encrypt the checksum calculation.

`−K` *keyfile* Same, but taking the first line of *keyfile* as the key.

SEE ALSO

ar(1), *cpio*(1), *tar*(1), *bundle*(1), *ship*(8), *ar*(5), *asd*(8)

BUGS

The pipeline `mkpkg ... | inspkg` fails if input and output files overlap.
Inspkg fills any holes in files.

NAME

mount, umount − mount and dismount file system

SYNOPSIS

`/etc/mount` [*special name* [*fstype* [*flags*]]]

`/etc/mount -a`

`/etc/mount` [*special name* [`-r`]]

`/etc/umount` *name*

`/etc/umount -a`

DESCRIPTION

Mount announces to the system that a removable file system of type *fstype* is present on the file *special*. The file *name* must exist already; it becomes the name of the newly mounted root. *Fstype* and *flags* are integers; if omitted, they default to 0. Type 0 is an ordinary disk file system. Other types and possible flag values are listed in *fmount*(2).

The shorthand `mount special file -r` is equivalent to `mount special file 0 1`: mount an ordinary file system read-only.

If option `-a` is present, *mount* attempts to mount, in order, every file system listed in *fstab*(5).

Umount announces to the system that the file system mounted on file *name* is to be removed.

These commands maintain a table of mounted file systems in `/etc/mtab`; see *fstab*(5). If invoked without an argument, *mount* prints the table. If option `-a` is present, *umount* attempts to remove, in reverse order, each file system listed in *mtab*.

Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, even if no explicit write is attempted.

EXAMPLES

`/etc/mount /dev/ra02 /usr`
: Mount the file system on disk `/dev/ra02` on directory `/usr`.

`/etc/mount /dev/null /proc 2`
: Mount the process file system.

FILES

`/etc/mtab` mount table
`/etc/fstab` file system table

SEE ALSO

fmount(2), *fstab*(5), *netfs*(8)

BUGS

Mounting file systems full of garbage may crash the system.
Mounting a root directory on a non-directory makes some apparently good pathnames invalid.

NAME

netfs − network file system

SYNOPSIS

`/usr/netb/setup.go`

DESCRIPTION

The network file system is conventionally a set of directories contained in `/n`, and a set of files and programs in `/usr/netb`. Connections in the network file system are asymmetric: files on a 'server' system are made accessible on a 'client' system, usually in directory `/n/`*server-name*.

Client

The client runs `/usr/netb/setup` to maintain connections; it is started by invoking `/usr/netb/setup.go` from *rc*(8). *Setup* uses `/usr/netb/friends` to control the connections to servers. Each line in *friends* contains six fields:

network address
network call argument
mount point
protocol id
unique identifier
debugging flag
network calling username

The network address and argument give the location of the server. They are interpreted differently according to the protocol id, which should be one of

`d` Call the server on the named network address, with default network `dk` and default service name `fsb`. The network call argument is ignored. The server machine should respond by calling *zarf*, described below; see *svcmgr*(8).

`t` Call the named network address, with default network `tcp`, and invoke the program named in the network call argument using the protocol of *rsh*; *con*(1).

Setup calls *setlogname* (see *getuid*(2)) to make the network call appear to have been placed by the calling username. The username may be omitted; `daemon` is the default.

The mount point is the directory on which the remote file system is to appear. The unique identifier is a integer in the range 0-255; it is used internally to distinguish connections, and must be unique among all active remote file systems (including those not maintained by *setup*, e.g. *faced*(9.5)). The debugging flag is usually 0; nonzero numbers increase the chatter in various logfiles.

Setup reads the *friends* file when it starts, and checks for changes once a minute. Each remote file system is probed once a minute; if there is no response to several consecutive probes, the connection is torn down and restarted. Failed connections are retried every minute.

Server

The server program is `/usr/netb/zarf`. A separate *zarf* process exists for each client.

When a connection is started, the client sends the server a list of valid user and group names and the corresponding numerical IDs on the client system. The userid and groupid of user and group names that exist on both machines are mapped so that client and server see IDs under the same names. Unmapped IDs on the server appear as −1 on the client. Client processes with unmapped IDs are denied access.

Zarf is subject to access control on the server. It will have access only to files that its own userid and groupid admit. Unless run as super-user, it will create files with its own, not mapped, userid.

Zarf reads configuration information from `/usr/netb/except.local` and `/usr/netb/except`. The files are read only once, when *zarf* starts, `except.local` first. Usually `except` is the same on all machines in some administrative cluster, `except.local` contains things specific to a particular server system.

The files contain sections beginning with the line `client` *origin*. *Origin* is the name of the calling client, as provided by the network; `*` matches any client. The first matching section is used.

Within each section, lines have of one of the following forms. Lines beginning with `#` are ignored.

`uid` *cname*`=`*sname*
: Regardless of the contents of password files, map client user name *cname* to server user name *sname*. If *cname* is not announced as valid by the client, the line is ignored. If *sname* is not a valid name on the server, any previous mapping for *cname* is discarded.

`gid` *cname*`=`*sname*
: Map client group name *cname* to server group name *sname*, as above.

`param otherok=`*val*
: If *val* is `1`, client processes with unmapped userids are granted world access to existing files on the server. Unmapped userids may never create files (who would own them?). If *val* is anything else, no access is permitted to unmapped client userids.

`param root=`*pathname*
: Use *pathname* rather than `/` as the root of the filename hierarchy made visible on the client.

EXAMPLES

A *friends* file for a connection to `alice` over Datakit, `shamash` over TCP/IP, and `bebop` over TCP/IP without administrative help:

```
alice           -                  /n/alice d   0   0
tcp!shamash!400 -                  /n/sun   d   1   0
bebop           /usr/pjw/netb/zarf /n/bebop t   2   0   pjw
```

Some *except* file rules:

```
client dk!nj/astro/research
param otherok=1
client *
uid root=
gid mail=other
param otherok=0
param root=/usr/spool
```

If the *research* machine calls as a client, the whole file system tree is visible, all userids including the super-user are permitted normal access, and user names unknown to the server are permitted world access. If any other machine calls, only the contents of `/usr/spool` are visible, *root* and unknown users are explicitly denied access, and processes in group *mail* on the client are treated as if in group *other* on the server.

FILES

`/n/*`	
`/usr/netb/friends`	client connection info
`/usr/netb/except.local`	
`/usr/netb/except`	
`/usr/netb/setupl`	log file for *setup*
`/usr/netb/zarf.log`	log file for zarf server control info

SEE ALSO

S. A. Rago, ‘A Look at the Version 9 Network File System’, this manual, Volume 2

BUGS

The scheme works only in a modest-sized, friendly community, as it requires a process per client, trust of clients’ security, and common login names.

File modification times are adjusted for clock-time differences between machines. Thus, when viewed across the network, identical files installed on different machines by *asd*(8) may appear to have different modification times, and symbol tables of random libraries (*ar*(1)) may appear to be out of date.

NAME

netstat, dkstat − show network status for internet and datakit networks

SYNOPSIS

`netstat` [`-acCirRst`] [*system*] [*core*]

`dkstat` [*interval* [*count*]]

DESCRIPTION

Netstat displays internet (TCP/IP and UDP/IP) traffic and configuration data. Without options, it lists all TCP and UDP connection assignments. A single option changes the listing:

`-a` known mappings between internet names and Ethernet addresses.

`-c` all TCP and UDP connection assignments (the default).

`-C` detailed state of active TCP connections.

`-i` active IP interfaces.

`-n` display numeric internet addresses rather than host and network names.

`-s` protocol statistics.

`-r` routing tables.

`-R` routing tables, including deleted entries (for debugging).

`-t`*buf* running trace of packets passing through *buf*: `il` for the Interlan Ethernet controller, `qe` for the DEQNA, `tcp` (the default) for all packets passing through TCP.

The arguments *system* and *core* are substitutes for the defaults `/unix` and `/dev/kmem`.

Dkstat reports the number of bytes received and sent over the Datakit network, together with error reports if any occurred. The first report is cumulative since a reboot. Further reports may be requested every *interval* seconds; these reports are incremental.

The optional *count* argument restricts the number of reports.

DIAGNOSTICS

`nlist /unix failed`: *netstat* could not find pertinent system information, perhaps because this system isn't set up for TCP/IP.

NAME

ns − name server database

SYNOPSIS

`/usr/ipc/mgrs/ns` [`−m` *server*] [`−d`]

DESCRIPTION

Ns maintains a database of naming information, accessed by *qns*(7) and other programs. It should be run once from *rc*(8).

The database is accessed through local service `ns`, or service *server* if option `−m` was specified.

The file `/usr/ipc/lib/ns.db` contains instructions for building the database. These instructions are lines of one of the following forms:

`#uusys` *file* Read the named *uucp* `Systems` *file*. For entries using caller `ACU`, add a database entry containing

system telephone-number`,tel uucp,svc`

For entries using caller `DK` or `DKH`, add an entry containing

system datakit-address`,dk uucp,svc`

For any other entry, add

system `uucp,svc`

`#inhost` *file* For each line in the named 4BSD-style internet hosts *file*, add a database entry of the form

ip-address`,in` *hostname host-domain-name*`,dom`

`#innet` *file* For each line in the named 4BSD-style internet networks *file*, add a database entry of the form

ip-net-address`,in` *netname*

`#include` *file* Interpret the contents of *file* in the same format as `ns.db`.

In all cases, *file* may be followed by a list of *value,attribute* pairs to be included with any database entries caused by that file. If the filename doesn't begin with `/`, it is prefixed with `/usr/ipc/lib`.

Blank lines and lines beginning with `#` followed by a space or tab are ignored.

Any other lines are taken as literal database entries: a collection of *value,attribute* pairs separated by spaces. Each line is a single entry.

The database is ephemeral; it is rebuilt whenever *ns* starts, when requested by `qns reset`, or when *ns* notices that `ns.db` or one of the files named therein has changed. Rebuilding can take several minutes, especially on a busy machine. During a rebuild, the server appears active but does not answer requests; calls will block until the rebuild finishes.

Ns leaves remarks in file `/usr/ipc/log/ns`. There are more remarks if the `−d` option was used.

FILES

`/usr/ipc/lib/ns.db`

SEE ALSO

ipc(3), *qns*(7)

NAME

postio − serial interface for postscript printers

SYNOPSIS

`/usr/bin/postscript/postio` [*option* ...] [*file* ...]

DESCRIPTION

Postio sends *files* to a PostScript printer. It is usually called by the innards of *lp*(1). If no files are named, the standard input is sent.

Mandatory argument `-l` names the printer. If the first character of *line* is `/`, it is assumed to be a local filename like `/dev/tty37`. Otherwise it is taken to be a network address, with default network `dk`, to which the printer is connected.

These options are probably the most useful:

`-b`*speed* Transmit data at baud rate *speed,* one of 1200, 2400, 4800, 9600 (default), and 19200.

`-q` Disable status queries while *files* are being sent to the printer. When status queries are disabled a dummy message is appended to the log file before each block is transmitted.

`-B`*num* Set the internal buffer size for reading and writing *files* to *num* bytes, 2048 by default.

`-D` Debug mode: copy everything read from the printer to the log file or standard error.

`-L`*file* Log data read from the printer in *file*. Standard error is the default. Normally only messages indicating a change in the printer's state are logged.

`-P`*string* Send *string* to the printer before any input *files*. The default is PostScript code that disables timeouts.

`-R`*num* If *num* is `1`, run as a single process; if `2`, use separate processes for reading and writing.

These options are not useful to spoolers like *lp*.

`-i` Interactive mode: send the *files* to the printer, then copy standard input to the printer and printer output to standard error. Overrides many other options. To have a friendly chat with the printer, begin by typing `executive` on a line by itself.

`-t` Copy printer output that doesn't look like status information to the standard output; intended for use with PostScript programs that write results.

This option should be used only as a last resort:

`-S` Take special measures to send data slowly. Limits the internal buffer to 1024 bytes, implies `-R1` and disables `-q` and `-i`. Expensive in CPU time.

When *postio* starts, it attempts to force the printer into IDLE state by sending a sequence of control-`t` (status query), control-`c` (interrupt), and control-`d` (end of job) characters. When the printer is idle, the files are transmitted with an occasional control-`t` interspersed (except under `-q`). After all data have been sent, *postio* waits until the printer appears to have finished before exiting. Fatal error messages from the printer cause *postio* to exit prematurely.

EXAMPLES

`postio -l/dev/tty01 file1 file2`
Runing as a single process at 9600 baud, send file1 and file2 to printer `/dev/tty01`.

`postio -R2 -B4096 -l/dev/tty01 -Llog file1 file2`
Similarly, but use two processes and a 4096-byte buffer, and copy printer messages to file `log`.

`postio -t -l/dev/tty22 -Llog program >results`
Send the PostScript `program` to printer `/dev/tty22`, place any data in `results`, put error messages in `log`.

`postio -i -l/cs/dk!my/printer`
Connect interactively to the printer at network address `/cs/dk!my/printer`.

SEE ALSO

lp(1), *postscript*(8)

DIAGNOSTICS

Exit status 1 means a system error (e.g. can't open the printer), 2 means a PostScript error, 3 means both. Status 2 is usually caused by a syntax error in an input file.

BUGS

Multiple files with PostScript end-of-job marks are not guaranteed to work.

If a network is involved, `-b` may be ineffective and attempts by *postio* to flow-control data in both directions may not work. Option `-q` can help if the printer is connected to Radian Datakit.

NAME

dpost, postdaisy, postdmd, postprint − filters to produce postscript

SYNOPSIS

`/usr/bin/postscript/dpost` [*option* ...] [*file* ...]

`/usr/bin/postscript/postdaisy` [*option* ...] [*file* ...]

`/usr/bin/postscript/postdmd` [*option* ...] [*file* ...]

`/usr/bin/postscript/postprint` [*option* ...] [*file* ...]

`/usr/bin/postscript/posttek` [*option* ...] [*file* ...]

DESCRIPTION

These programs convert files of various formats into PostScript. The input formats are

dpost *troff*(1) output

postdaisy Diablo 1640 daisy-wheel

postdmd *bitfile*(9.5) files, as produced by *blitblt*(9.1)

postprint ASCII text

posttek Tektronix 4014 graphics

Except as noted, the options are common to all the programs:

`−c`*num* Print *num* copies of each page. By default only one copy is printed.

`−m`*num* Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin, located near the upper left corner of the page. The default magnification is 1.0.

`−n`*num* Print *num* logical pages on each piece of paper. The default is 1.

`−o`*list* Print only pages specified in the comma-separated *list* of numbers and ranges. A range *N*−*M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end. Print only pages whose numbers are given in the comma-separated *list*. The list contains single numbers *N* and ranges *N1*−*N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest.

`−p`*mode* Print in *mode* `p` (portrait) or `l` (landscape). The default is `p`.

`−x`*num* Translate the origin *num* inches along the positive x axis. By default, the origin is fixed near the upper left corner of the page, with positive x to the right and positive y down the page. Positive *num* moves everything right. The default offset is 0 inches.

`−y`*num* Translate the origin *num* inches along the positive y axis. Positive *num* moves text down the page. The default offset is 0.

`−A`*file* Append a simple accounting record to *file* after all input *files* have been successfully translated. By default no accounting data is produced.

`−L`*file* Use *file* as the PostScript prologue.

`−f`*name* Print *files* using font *name*. Any PostScript font can be used, but constant width fonts yield the best results. The default font is Courier. (*postdaisy*, *postprint*, and *posttek* only)

`−f` Flip the sense of the bits in *files* before printing the bitmaps. (*postdmd* only)

In addition, three options allow the insertion of arbitrary PostScript at controlled points in the translation process:

`−C`*file* Copy *file* to the output file. *File* follows the prologue but precedes any job initialization commands. *File* becomes part of the job's global environment and must contain legitimate PostScript commands.

`-P`*string* Like `-C`, using a *string* instead of the contents of a file.

`-R`*action* Requests special *action* (e.g. manualfeed) on a per page or global basis. The *action* string has the general form *request*`:`*page*`:`*file*, from which `:`*page*`:`*file* or `: file` can be omitted. An omitted or 0 page number applies to all pages. If file is omitted the request lookup is done in `/usr/lib/postscript/ps.requests`. The collection of recognized requests can be modified or extended by changing this file. Multiple occurrences of the `-R` option behave as expected.

FILES

`/usr/lib/font/devpost/*.out`	
`/usr/lib/font/devpost/charlib/*`	
`/usr/lib/postscript/*.ps`	default prologues
`/usr/lib/tmac/tmac.pictures`	*troff* macros for PostScript

SEE ALSO

lp(1), *postio*(8)

DIAGNOSTICS

Exit status 2 usually means a syntax error in some input file.

BUGS

Output files will often violate Adobe's file structuring conventions. Pipe the output of *dpost* through *postreverse* to produce a minimally conforming PostScript file.

Although *dpost* can handle files formatted for any *troff* device, emulation is expensive and can easily double the print time and the size of the output file.

No attempt has been made to implement the character sets or fonts available on all devices supported by *troff*. Missing characters are replaced by white space; unrecognized fonts are replaced by one of the Times fonts.

Dpost requires an `x res` command before the first `x init` and any file data.

NAME

pstat − print system facts

SYNOPSIS

`/etc/pstat` [`-afipstuxT`] [*suboptions*] [*file*] [*namelist*]

DESCRIPTION

Pstat interprets the contents of certain system tables. If *file* is given, the tables are sought there, otherwise in `/dev/kmem`. The required namelist is taken from *namelist*, default `/unix`. Options are

`-a` Under `-p`, describe all process slots rather than just active ones.

`-i` Print the inode table with the these headings:

- `LOC` The core location of this table entry.
- `FLAGS` Miscellaneous state variables encoded thus:
  - `L` locked
  - `U` modified time (*filsys*(5)) must be corrected
  - `A` access time must be corrected
  - `O` file was opened
  - `W` wanted by another process (`L` flag is on)
  - `T` contains an active text
- `CNT` Number of active references to this inode.
- `FS` File system type, see *fmount*(2).
- `DEVICE` Device number of file system in which this inode resides.
- `INO` I-number within the file system.
- `MODE` Mode, see *stat*(2).
- `NLN` Number of links to this inode.
- `UID` Userid of owner.
- `SPTR` Core location of corresponding stream header, 0 if this is not a stream.
- `SIZ/DEV` Number of bytes in an ordinary file, or device number of a special file.
- `MROOT` Core location of root inode of file system mounted here, 0 if none.

`-x` Print the text table with these headings:

- `LOC` The core location of this table entry.
- `FLAGS` Miscellaneous state variables encoded thus:
  - `P` resulted from demand-page-from-inode exec format, see *exec*(2)
  - `T` traced through *proc*(4)
  - `W` text not yet written to swap device
  - `L` loading in progress
  - `K` locked
  - `w` wanted (`L` flag is on)
- `DADDR` Disk address in swap, in multiples of 512 bytes.
- `CADDR` Head of a linked list of loaded processes using this text segment.
- `RSS` Size of physical memory occupied by text segment, in multiples of 512 bytes.
- `SIZE` Size of text segment, in multiples of 512 bytes.
- `IPTR` Core location of corresponding inode.
- `CNT` Number of processes using this text segment.
- `CCNT` Number of processes in core using this text segment.

`-p` Print process table for active processes with these headings:

- `LOC` The core location of this table entry.
- `S` Run state encoded thus:
  - `0` no process
  - `1` waiting for some event
  - `3` runnable
  - `4` being created

5 being terminated
6 stopped under trace

F Miscellaneous state variables, or-ed together (hexadecimal):
0000001 loaded in memory
0000002 special system process (swapper or pager)
0000004 being swapped out
0000008 obscure swapout flag
0000010 traced
0000020 used in tracing
0000040 locked in core
0000080 waiting for pagein
0000100 prevented from swapping during *fork*(2)
0000200 gathering pages for raw i/o
0000400 exiting
0008000 associated text is demand paged from file
0030000 anomalous paging behaviour expected, see *vlimit* in *deprecated*(2)
0040000 in a sleep which will time out
0400000 in *select*(2)
0800000 traced via *proc*(4)
1000000 i/o via *proc* in progress
2000000 stop on exec
4000000 wanted by *proc* after pagein

ADDR The core location of the page table entry for the first page of the 'u-area.'
PRI Scheduling priority; smaller numbers run first.
SIG Signals received; signals 1-32 coded in bits 0-31.
UID Real userid.
SLP Time blocked in seconds; times over 127 coded as 127.
TIM Time resident in seconds; times over 127 coded as 127.
CPU Weighted integral of CPU time, for scheduler.
NI Nice level, see *nice*(2).
PGRP Process group number.
PID Process ID number.
PPID Process ID of parent process.
RSS Number of physical page frames allocated to this process.
SRSS RSS at last swap, 0 if never swapped.
SIZE Virtual size of process image (data+stack) in multiples of 512 bytes.
WCHAN Event address if waiting.
LINK Pointer to next entry in list of runnable processes.
TEXTP If text is pure, pointer to location of text table entry.
CLKT Countdown for *alarm*(2) measured in seconds.

−u Print information about a user process; the next argument is its address as given by ADDR under −p above. The process must be in main memory, or the file used can be a core image (*core*(5)) and the address 0.

−f Print the open file table with these headings:
LOC The core location of this table entry.
FLG Miscellaneous state variables encoded thus:
R open for reading
W open for writing
CNT Number of processes that know this open file.
INO The core location of the inode table entry for this file.
OFFS The file offset, see *lseek*(2).

`-s` Print information about swap space usage: the number of 1024 byte pages used and free, and the number of pages belonging to text images.

`-T` Print the number of used and free slots in several system tables; useful to see if they are nearly full.

FILES

`/unix`	namelist
`/dev/kmem`	default source of tables

SEE ALSO

ps(1), *stat*(2), *filsys*(5)
M. J. Bach, *The Design of the UNIX Operating System*, Prentice-Hall, 1986

BUGS

This program is never up to date.

NAME

quot, findo − file system usage and hogs

SYNOPSIS

`/etc/quot` [*option ...*] [*filesystem*]

`findo` [`−f`] [`−n`] [`−u` *userid*] *device mount-directory*

DESCRIPTION

Quot prints the number of blocks in the named *filesystem* device currently owned by each user. If no *filesystem* is named, `/dev/usr` is assumed. The options are:

`−n` Use as in the example below to list all files and owners.

`−c` Print three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in files of that size or smaller.

`−f` Print count of number of files as well as space owned by each user.

`−b` Print space-time product in block-years in addition to space owned by each user.

Findo discovers files you might want to delete on the given block *device*, which must be mounted on the given directory. It lists, on the standard output, the sizes, ages in days, and names of files with any of the following characteristics:

- *Troff*(1) output files older than 24 hours. The names are marked `troff:` in the output.
- Week-old files named `core`, `a.out`, `mon.out`, `.pilog`, `junk*`, `temp*`, `ed.hup`, `qed.hup:*`, `jim.recover`, `sam.save`, `sam.err`, `~~sam~~`, `[a−z]`, `dead.letter`, `foo[0−9]*`, `rst[0−9]+`, `.jx*`, `*.dvi`, and files whose names resemble *apnews*(7) spool entries.
- Files over a month old named `*.o`. The names are marked `old:`.
- Files owned by users selected with option `−u`; the names are marked `user:`.

The options are

`−f` List files owned by users not in the password file.

`−u` *userid*
List files over 2 days old owned by the user with the given numeric *userid.*

`−n` List files of any age owned by the specified users.

EXAMPLES

`ncheck filesystem | sort −n | quot −n filesystem`
List all files and their owners.

FILES

`/etc/passwd` to get user names

SEE ALSO

ls(1), *du*(1), *icheck*(8), *fstab*(5)

BUGS

Quot counts holes in files as if they actually occupied space.
Patterns specifying the names and ages are compiled into *findo.*
Findo's age distinction for files owned by a specific user is a historical dreg.

NAME

rarepl, rarct − replace bad blocks on MSCP disks

SYNOPSIS

`/etc/rarct` [`-c`] [`-h`] *special* ...

`/etc/rarepl` *special lbn* ...

DESCRIPTION

Rarct prints status information about MSCP disk drives like the RA60 and RA81. Normally the replacement table (RCT) is listed, as lines of the form

rbn **:** *flags* **:** *lbn*

where *rbn* is the replacement block number, *lbn* is the logical block number replaced by *rbn*, and *flags* are constructed from the following bits:

`01`	alternate (not primary) replacement block
`02`	normal, allocated replacement block
`04`	this replacement block is bad
`010`	this replacement block does not exist

Entries whose *flags* are zero, indicating a good, unused replacement block, are not listed.

The options suppress the RCT listing and perform other functions:

`-h` Print some header data from the first block of the RCT. The system does not use this information.

`-c` Print geometry information for the drive.

Rarepl causes logical block *lbn* on device *dev* to be marked as bad and replaced. The nearest available replacement block is used. The contents of *lbn* are copied into the replacement block if possible; if *lbn* is unreadable, the replacement block is initialized with zeros.

Both programs work only on the raw devices. *Rarepl* should be used only on a device which covers the entire drive (usually partition 7).

SEE ALSO

ra(4), *smash*(8)

BUGS

There are various controller- and drive-dependent anomalies. Some controllers, like the RQDX3, report an RCT but don't allow forwarding. On many controllers, the RCT exists only so programs in the host can look at it; the controller ignores its contents. There is no way to read the controller's 'real' forwarding data, only a way to set it for a particular block. Hence if the RCT is corrupted, the disk may still be used, but must be reformatted before additional bad blocks are remapped.

NAME

rc − boot script

SYNOPSIS

`/etc/rc`

DESCRIPTION

Rc is the command script invoked by *init*(8) to control reboots. During an automatic reboot, *rc* is invoked with the argument `autoboot`; typically this invokes `/etc/fsck −p` to repair minor filesystem inconsistencies. If *rc* exits with a successful status, *init* proceeds to multi-user mode.

When the system enters multi-user mode, either during an auto-reboot or after the single-user shell terminates, *rc* is invoked without arguments. This usually causes it to mount filesystems, start daemons, clear `/tmp`, and perform other housekeeping.

If any call to *rc* returns a nonzero status, *init* reverts to single-user mode.

EXAMPLES

A typical *rc* script:

```
date
case $1 in
autoboot)
      echo Autoboot:
      /etc/fsck −p || {echo "error in reboot"; exit 1}
esac
/etc/ldpcs /etc/pcs750.bin
>/etc/mtab
/etc/mount −a
/etc/savecore /tmp/dump /dev/ra11
/etc/swapon −a
trap "" 1 2 3
/etc/update
/etc/cron &
rm −f /tmp/*
/usr/lib/asd/rmlocks
date >> /usr/adm/lastboot
/etc/accton /tmp/acct > /tmp/acct
/usr/ipc/mgrs/svcmgr
/etc/kdiload
/usr/ipc/mgrs/dkhup; sleep 10
/usr/ipc/mgrs/dkmgr
/usr/netb/setup.go
/usr/net/face.go
wwv −s
```

SEE ALSO

init(8), *reboot*(8)

NAME

reboot − bootstrapping procedures

DESCRIPTION

Here are some recipes for booting and crashing the operating system on VAXes.

Rebooting a running system

The preferred way to reboot is to log in on the console as super-user, invoke `kill 1` to take the system to single user, unmount file systems with `/etc/umount -a` and halt and restart the system as described below under 'Console boots.'

Power fail and crash recovery

The system will reboot itself at power-up or after crashes if auto-boot is enabled on the machine front panel or in the console software. If auto-restart is enabled, the system will first attempt to save a copy of physical memory on a reserved piece of disk. An automatic consistency check of the file systems is performed. Unless this fails the system will resume multi-user operations.

Console boots

Sync the disks if necessary and possible. To recover hardware control of the console, type a control-P. This will yield a >>> prompt from the VAX console subsystem (sic). The command

```
>>> H
```

will halt the CPU (except on the 11/750, where control-P halts the CPU right away).

On MicroVAXes, control-P doesn't work; hit the BREAK key instead.

To boot multi-user with an automatic file system check, give the console command

```
>>> B
```

Commands to boot single-user vary. On the VAX-11/750 and on MicroVAXes, use

```
>>> B/3
```

On the VAX-11/780 and VAX 8550 and 8700, use

```
>>> B MAN
```

This will prompt with * for the name of the file to boot. The filename should be an executable image in the root directory of the filesystem at the beginning of the disk.

System core images

If the system crashes and auto-restart is enabled, a copy of physical memory is written to a reserved piece of disk. To save a core image of a hung system, type on the console (after control-P if necessary):

```
>>> S 80000010
```

The system will write the core image, then reboot automatically.

If the core image was written on `/dev/ra11`, the following incantation will print a stack traceback from the time of the crash:

```
adb /unix /dev/ra11
$<crash
$c
```

To save disk space, the core image is sometimes overlaid on part of the swap area, where normal system operation will soon overwrite it. *Savecore*(8) will copy the core image to an ordinary disk file.

FILES

`/unix` default system binary

SEE ALSO

fsck(8), *init*(8), *rc*(8), *savecore*(8)

BUGS

Older boot programs with different syntax are still around in a few places, especially on machines with Emulex UNIBUS disk controllers, for which silly boot ROMs are common.

There are commands `/etc/reboot` and `/etc/halt` which attempt to reboot and halt the system; their function is indeterminate and likely to change.

NAME

renice − change priority of running process

SYNOPSIS

`/etc/renice` [−*priority*] *pid* ...

DESCRIPTION

Renice increments the scheduling priority of the processes with the named *process-ids* by *pri*. The default *pri* is 19, making the process least likely to run.

Only the owner of the process or the super-user may change the priority. Only the super-user may use negative increments.

FILES

`/proc`

SEE ALSO

nice(1), *nice*(2), *proc*(4)

NAME

route, routed, remroutes − IP gateway routing

SYNOPSIS

`/usr/ipc/mgrs/routed` [`-v`] [`-t`] [`-q`] [*-hops*] [*addr ...*]
`route add` *dest gateway*
`route delete` *dest*
`/etc/remroutes`

DESCRIPTION

Routed runs the 4BSD RIP routing protocol on an IP network. It broadcasts routing information to the network at large, listens for routing messages from elsewhere, and informs the system of the routes it receives.

The options are

`-v` Log transmitted messages on the standard output.

`-t` Log received messages on the standard output.

`-q` Accept routing information but do not broadcast any.

-hops Add *hops* (a decimal number) to the hop count when broadcasting routes.

Information received for any named *addrs* is ignored.

Routed is usually run without options on gateway machines, and with the `-q` option on non-gateway machines.

Route sets up specific routes, to establish static routing or to adjust that set up by *routed*. The `add` command informs the system that internet address *dest* may be reached through internet address *gateway*; `delete` removes any routing for *dest*. The special destination `*` represents the default routing: `route add *` *gate* sets the default, `route delete *` removes any default.

Remroutes removes all known routes.

FILES

`/usr/ipc/log/routed`

SEE ALSO

ipconfig(8)

NAME

sa, accton − system accounting

SYNOPSIS

`/etc/sa` [`-abcdDfgijkKlnrstuv`] [`-e prefix`] [*file*]

`/etc/accton` [*file*]

DESCRIPTION

With an argument naming an existing *file*, *accton* causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

Sa reports on, cleans up, and generally maintains accounting files.

Sa is able to condense the information in `/usr/adm/acct` into a summary file `/usr/adm/savacct` which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system `/usr/adm/acct` can grow by 10000 blocks per day. The summary file is normally read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; `/usr/adm/acct` is the default.

Output fields are labeled: `cpu` for the sum of user and system times (in minutes), `re` for real time (also in minutes), `k` for cpu-time averaged core usage (in 1K units), `avio` for average number of IO operations per execution. With options fields labelled `tio` for total IO operations, `k*sec` for cpu storage integral (kilo-core seconds), `u` and `s` for user and system cpu time alone (both in minutes) will sometimes appear.

There are zillions of options:

`a` Place all command names containing unprintable characters and those used only once under the name `***other`.

`b` Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.

`c` Besides total user, system, and real time for each command print percentage of total time over all commands.

`d` Sort by average number of disk IO operations.

`D` Sort by total number of disk IO operations.

`e` Set the prefix for accounting file names to the next argument (`/usr/adm/` is the default).

`f` Assume answer `y` for option `-v`.

`g` Ignore `/usr/adm/acct`. Useful for processing only `savacct` and `usracct`.

`i` Don't read in summary file.

`j` Instead of total minutes time for each category, give seconds per call.

`k` Sort by cpu-time average memory usage.

`K` Print and sort by cpu-storage integral.

`l` Separate system and user time; normally they are combined.

`m` (money) Print number of processes and number of CPU minutes for each user.

`n` Sort by number of calls.

`r` Reverse order of sort.

`s` Merge accounting file into summary file `/usr/adm/savacct` when done.

`t` For each command report ratio of real time to the sum of user and system times.

`u` Superseding all other flags, print for each command in the accounting file the userid and command name.

`v` Followed by a number *n*, types the name of each command used *n* times or fewer. Await a reply from the terminal; if it begins with `y`, add the command to the category `**junk**`. This is used to strip out garbage.

FILES

`/usr/adm/acct`	raw accounting
`/usr/adm/savacct`	summary
`/usr/adm/usracct`	per-user summary

SEE ALSO

ac(8), *acct*(2)

BUGS

Sa needs more options.

NAME

savecore − save a core image of the operating system

SYNOPSIS

`/etc/savecore` *target dump*

DESCRIPTION

Savecore copies the core image saved after an operating system crash to an ordinary file. This is worth doing so that the crash image will not be overwritten immediately by another crash, or sometimes because the crash image was written in a place where normal system operation will overwrite it (e.g. in the swap area).

The crash image is taken from *dump* and written to *target*. If *target* exists and is a directory, the image is copied to a file in that directory with the first nonexistent name in the sequence `z.0 z.1 z.2 ...`; otherwise *target* is created or overwritten.

The crash image to be copied is checked for a magic number in a known location. If the magic number is correct, it is followed by the size of the image, and the time it was written; these numbers are printed before the dump is copied. If the magic number is wrong, the image is not copied. *Savecore* overwrites the magic number in *dump* after a successful copy.

The program runs faster if *dump* is the raw device.

For compatibility with an older program of the same name, the *dump* argument may be omitted; *savecore* will noisily examine each device specified for swapping in *fstab*(5) and each of several popular default swap devices for a valid magic number. The first device that looks right is taken to be the crash image.

Savecore is usually called when the system is booted, from *rc*(8).

EXAMPLE

```
/etc/savecore /tmp/dump /dev/rra11
```

SEE ALSO

reboot(8)

BUGS

The argument convention (the file to be written comes first) is unfortunate; it stems from compatibility.

NAME

scsish − SCSI shell

SYNOPSIS

`/usr/lib/worm/scsish`

DESCRIPTION

Scsish is a command interpreter for SCSI commands executed through `/dev/scsi` (see *scsi*(4)). Many commands are applicable to more or less all SCSI devices; some are specific to the SONY WDA-3000-10 optical disk jukebox. Any details not found here are in the manual for the jukebox.

Typically commands are sent to a particular drive (a number between 0 and 7 inclusive) on a particular device (normally a number between 0 and 5 inclusive). Most commands take a drive parameter (a number). The device number is set by the `id` command.

Occasionally, commands fail and will print the result of a `sense` command which is normally needed to clear the error status.

All input is in lower case and keywords and numbers are separated by white space. Commands are separated by a newline or semicolon.

General SCSI Commands

`capacity` *drive*
: Report the capacity of *drive* as `nblocks x blocksize`.

`disk eject` *drive*
: Eject the disk (or other removable medium) from *drive*.

`echo` *number*
: Print *number* on standard output.

`help` Print a summary of the available commands.

`id` *n* Set the destination SCSI bus device number. By default, it is 2 which is the normal device number for the SONY jukebox.

`inquiry` *drive*
: Print various bits of status about *drive*. For example,

```
drive 2,0: WORM device, '    SONY     WDA-3000-10 2.D'
      disk,write protect,,,ready (0x9)
```

If *drive* is omitted, an inquiry is performed for drives 0 through 7.

`read` *drive block*
: Print the contents of the 1024 byte block at *block* on *drive* in hexadecimal.

`read id` *drive*
: Print the string starting at byte 42 in block 1 on *drive*. This corresponds to the initial *vol_id* for *worm*(8) disks.

`reset` Attempt to reset the SCSI interface.

`sense` *drive*
: Print the sense data for *drive*. Some of the interpretations of the sense bytes are idiosyncratic to SONY.

`ext sense` *drive*
: Print the extended sense data for *drive*. Most of the interpretations of the extended sense bytes are idiosyncratic to SONY.

`sleep` *n*
: Sleep for *n* seconds.

`start` *drive*
: Start *drive* spinning.

`stop` *drive*
: Stop *drive*.

`test` *drive*
: Test unit ready for *drive*.

SONY Commands

`alternate` *drive*
: Print the replacement block tables from the disk.

`media` *drive blkno nblks*
: Print a summary of the media quality in *drive* for the *nblks* blocks starting at block number *blkno*. For example, a dirty disk can yield

```
drive 0: media check for 1000 blocks [0-999], upper drive
849 good, 1 unwritten, 147 <50% burst, 3 >96% burst,
```

Please report any instances of messages including `rare error` to the jukebox guru.

`ext media` *drive blkno nblks*
: A verbose form of the `media` command.

Jukebox Commands

`config`
: Print the configuration data for the jukebox.

`rel` *drive shelf side*
: Release the disk from *drive* to *shelf*. The value of *side* indicates whether it should be inverted on the way (`b`) or not (`a`). If *shelf* and *side* are absent, the disk is restored to its former shelf.

`set` *shelf side drive*
: Put the disk from *shelf* into *drive*. The value of *side* indicates whether it should be inverted on the way (`b`) or not (`a`).

`internal` *n*
: Execute various internal reports and diagnostics. `internal -1` with no argument will print a list of available diagnostics.

`status` *drive*
: Print the status for *drive*. An absent *drive* is taken as 0. As the status is for the jukebox as a whole, the value of *drive* doesn't matter. A sample status output shows the jukebox hides the mapping of logical drive number and actual drive:

```
drive 0: ready,disk in LUN,power on,disk in drive 0, return shelf 2
drive 1: not ready,no disk in LUN,power on,disk in shelf 0
drive 2: not ready,no disk in LUN,power on,disk in shelf 0
drive 3: ready,disk in LUN,power on,disk in drive 1, return shelf 0
drive 4: not ready,no disk in LUN,power on,disk in shelf 0
drive 5: not ready,no disk in LUN,power on,disk in shelf 0
drive 6: not ready,no disk in LUN,power on,disk in shelf 0
drive 7: not ready,no disk in LUN,power on,disk in shelf 0
0: no disk
1: no disk
2: disk,
I/O shelf: no disk
carrier: disk shelf=0
upper drive: disk, LUN=0
lower drive: disk, LUN=3
```

SEE ALSO

worm(8), *scsi*(4)

NAME

sendcover − send cover sheet to the library

SYNOPSIS

`sendcover` *file* ...

DESCRIPTION

Sendcover sends a document cover sheet to the Bell Laboratories library for their document database. It is invoked automatically as a byproduct of running *troff -mcs.*

The cover sheet is translated from the form of *mcs*(6) to a form used in the library and certain other protocol information is added. The destination is not the same as that of *docsubmit*(1).

FILES

`/usr/lib/tmac/tmac.cs`
`/usr/lib/tmac/cstrans`

SEE ALSO

docsubmit(1), *mcs*(6)

NAME

ship, shipstat − automatic software distribution

SYNOPSIS

`ship` [*option* ...] [*file* ...]

`shipstat`

DESCRIPTION

Ship distributes the named files to other computers, where the files are installed under the same names. Shipping privileges are determined by the network manager on the receiving machine; see *svcmgr*(8).

Destinations are taken from environment variable `dest`, or from `/usr/lib/asd/dest/default` if `dest` is empty. If a destination is the name of a file in `/usr/lib/asd/dest`, it is replaced by the contents of that file, each word of which is then examined in the same way. Otherwise the destination is a network address. The sending machine is omitted unless explicitly named in the environment variable, or unless option `−f` is present or environment variable `force` has a non-empty value.

Ship uses *inspkg* and *mkpkg*(8) to do its work. Links among the named files are imitated on the receiving computer, and named files that do not exist on the sending computer are deleted on the receiving computer. Other options are the same as those of *mkpkg*:

`−v` Emit running commentary on the standard error file.

`−D`*path1*`=`*path2*
: Pretend that any *file* name that begins with *path1* really begins with *path2*. Relative pathnames are extended to full pathnames before comparison.

`−x`*command*

`−X`*file* Include in the package instructions to execute the shell *command* or run the shell script *file* after all files have been installed. Only one of these options may occur. The *file* name in `−X` is affected by `−D`.

Shipments are generally acknowledged by mail after each destination has been tried at least once; see *asd*(8) for details.

Shipstat reports the status of all its caller's incomplete shipments, with the most recent first.

FILES

`/usr/lib/asd/dest/*`	distribution lists
`/usr/lib/asd/dest/default`	default distribution list
`/usr/spool/asd/`*logname*	outgoing spool directories

SEE ALSO

mkpkg(8), *asd*(8), *svcmgr*(8)

BUGS

The `−f` option, if given, must be the first option and must not be combined with any other.

NAME

showq − status of stream input/output system

SYNOPSIS

`/etc/showq` [`−v −V −s −m`] [*system*] [*mem*]

DESCRIPTION

Showq reports connectivity and contents of I/O streams. By default, it lists the maximum number of stream blocks (of various sizes) ever used, then each stream and the queue modules in each stream, and then blocks that are unaccounted for (not on any queue or the free list).

By default, the system namelist is `/unix` and the place the streams are kept is `/dev/mem`.

The options are:

`−v` Verbose. Show more, in particular the contents of data and control blocks on each queue.

`−V` Very verbose. Show all blocks on every queue instead of giving up after a while.

`−s` Silent. Examine queues for consistency, printing only a summary.

`−m` Missing. Show the contents of missing blocks. (Perhaps this will give a clue about who lost them.)

FILES

`/unix`
`/dev/mem`

SEE ALSO

stream(4), *netstat*(8)
mesgld(4) for a list of message types

NAME

smash − rewrite bad disk sectors

SYNOPSIS

`/etc/smash` *device sector*

DESCRIPTION

Smash attempts to read the named (decimal, 512-byte) *sector* from the named *device*, and prints the error status from the read and the data read, in octal, regardless of the error status. It then prompts `write?`, to which there are three answers:

`y` Write the data back to the sector.

`c` Write zeros to the sector.

anything else
Quit.

After the sector is written, it is read again and the cycle repeats.

Writing the sector, even if its contents could be correctly read, will recompute the error correcting code. This may make soft ECC errors vanish, and will recover what can be recovered (sometimes not much) from hard ECC errors.

SEE ALSO

rarepl(8)

NAME

smstat − list smtp queues

SYNOPSIS

`smstat`

DESCRIPTION

Smstat prints a summary of pending mail messages queued by the programs in *smtp*(8). Each line contains the name of a spooling directory; the number of outbound messages, followed by `C`; and the number of inbound messages, followed by `X`.

FILES

`/usr/spool/smtpq/*` spool directories

SEE ALSO

smtp(8)

NAME

smtp, smtpqer, smtpd, smtpsched − handle simple mail transfer protocol

SYNOPSIS

`/usr/lib/upas/smtp` [*option ...*] *replyaddr dest recipient ...*

`/usr/lib/upas/smtpqer` [*option ...*] *replyaddr dest recipient ...*

`/usr/lib/upas/smtpd` [`−n`] [`−H` *host*]

`/usr/lib/upas/smtpsched` [option ...] [*queue ...*]

DESCRIPTION

Smtp reads a mail message from the standard input, and sends it with the Internet SMTP protocol to the named *recipients* at network address *dest*. *Dest* has default network `tcp` and default service `tcp.25` (the conventional Internet SMTP port). Error reports are mailed to local address *replyaddr*.

Smtp operates in two modes, 'Internet' (default) and 'Unix'. In Internet mode *recipient* addresses should be in full domain form. `From:` and `Date:` headers will be inserted as necessary to conform to Internet standards. In Unix mode addresses and message contents are not touched. The options are

`−u` Run in Unix mode.

`−H` *host* Use *host* as the name of the sending system (taken from *whoami*(5) by default).

`−d` *domain*
Append the specified domain suffix to incomplete addresses.

Smtpqer reads a mail message from the standard input and stashes it away to be sent later by *smtpsched*. By default, *smtpsched* is started immediately; option `−n` prevents this. Other options and arguments are the same as for *smtp*.

Smtpd receives a message by speaking the server part of SMTP on the standard input and output. The message is stashed in a queue for later delivery as by *smtpqer*. Option `−n` prevents *smtpsched* from running immediately; option `−H` is as for *smtp.*

Smtpsched processes the queues assembled by *smtpqer* and *smtpd*, calling *mail*(1) for local messages and *smtp* for others. It should be run occasionally from *cron*(8).

The *queue* arguments name particular queue directories to be processed; if no queue is named, all queues are processed. The options are

`−w` *days*	Send a warning about each message more than *days* old to the reply address.
`−r` *days*	Mail an error reply about each message more than *days* old, and discard the message.
`−s` *nproc*	Do not run more than *nproc* simultaneous copies of *smtpsched* started with this option.
`−c`	Remove empty directories and inconsistent files.
`−t`	Log actions without performing them.
`−C`	Process 'C' command files (*smtp* calls) only.
`−X`	Process 'X' command files (*rmail* calls) only.
`−v`	Enable verbose logging.

The queues are kept in subdirectories of `/usr/spool/smtpq`, named by splitting the lower case remote system name into components separated by periods, concatenating the last two or fewer components, taking the last 14 characters, and stripping leading periods.

FILES

`/usr/spool/smtpq`	spooling directory
`/usr/spool/smtpq/smtpqsched.log`	logging
`/usr/spool/smtpq/.consumers`	list of process IDs running *smptqsched -s*

SEE ALSO

mail(1), *upas*(8), *smstat*(8)
DARPA standards RFC 822, RFC 976

NAME

su, setlog − substitute userid temporarily, become super-user

SYNOPSIS

`/etc/su` [*user*]

`/etc/setlog` *logname command ...*

DESCRIPTION

Su changes the userid to that of *user* (root by default) with groupid and login shell determined from the password file. If the current userid is not the super-user, the password for the new user is demanded. The userid stays in force until the new shell exits.

The current directory and environment are unchanged unless the new userid is super-user, in which case the environment variables `PS1` and `PATH`, if present, are set to standard values (white space and `/bin:/usr/bin:/etc`).

Setlog executes the specified *command* with login name *logname*. The environment is otherwise unchanged; in particular, the userid is not set. Only the super-user may use this command.

SEE ALSO

sh(1), *getuid*(2), *passwd*(5)

NAME

svcmgr − service remote computing requests

SYNOPSIS

`/usr/ipc/mgrs/svcmgr` [`-d`]

DESCRIPTION

Svcmgr performes services such as login and command execution, often in response to requests from network listeners like *dkmgr* and *tcpmgr*(8). It should be run once from *rc*(8).

Svcmgr is controlled by several files in directory `/usr/ipc/lib`: services are defined in files `serv` and `serv.local`, authorization in `auth` and `auth.local`. The `.local` files are searched first. The idea is that `serv` and `auth` will be the same throughout an administrative cluster of machines, and anything peculiar to specific systems will be kept in `serv.local` and `auth.local`.

Each service is announced as a name in directory `/cs` using the routines in *ipc*(3). When a connection is requested to one of these services, *svcmgr* receives a file descriptor connected to the requester. A new process is created to perform the actions listed for that service in the *serv* files, usually resulting in a *login*(8) with standard input, output, and error files attached to the connection. Often there are flags to *login* specifying a local user name or a command to be executed. Environment variable CSOURCE is set to a string of the form

`source=`*remote-machine* `user=`*ruser* `line=`*lineinfo*

Remote-machine and *ruser* are supplied in the connection message; *lineinfo* network-dependent stuff of varying interest and meaning. If a particular command was specified (the `cmd` or `exec` action), *login* sets environment variable REXEC to `1`.

The *auth* files are used to translate remote user names to local ones. They contain lines with four fields:

service name
calling system name
calling user name
local user name

The service, calling system, and calling user names are regular expressions in the style of *regexp*(3). The calling system and calling user fields may be omitted; `.*` is assumed. The local user name is a literal name, `&` to repeat the calling user name provided in the request, or `:` to explicitly reject a call. If the local user name is omitted, `&` is assumed.

Several service actions ‘look up the connection in the *auth* files.’ This means to find the first line in `auth.local` or `auth` for which the service, calling system, and calling user match the patterns, and return the local user name in that line (the same as the calling user if `&`). If no matching line is found, or if the first match has local user name `:`, the lookup fails.

The *serv* files contain lines with three fields:

service name
a list of actions, separated by +
the calling system name

The calling system name is a regular expression as in the *auth* file. The line matching an incoming call is the first whose service matches the requested service and whose regular expression matches the calling machine.

The possible actions are:

`user(`*x*`)` Use local username *x*.

`auth` Look up the connection in the *auth* files. If a match is found, use the resulting local user. Otherwise reject the call.

`v9auth` Look up the connection in the *auth* files; if a match is found, send `OK` to the caller, and use the result. If there is no match, send `NO`, and read a string of the form

'login,passwd\n'. If the login and password describe a valid local user, send `OK` and use that user; otherwise send `NO` and try again (until the caller gives up). This is the authentication protocol used by *ipclogin* (see *ipc*(3)), hence by *con*(1), *push*(1), and *pull.*

`inauth` Read two null-terminated strings from the caller. If they aren't the same, reject the call. Otherwise look up the service, calling system, and the null-terminated string (as a user name) in the *auth* files, use the resulting local user if there's a match, reject the call otherwise. This is the authentication protocol used by *ipcrogin,* hence by *rsh* and *rlogin*; see *ipc*(3) and *con*(1).

`ttyld` Push the terminal line discipline *ttyld*(4) onto the connection.

`mesgld` Push the reverse message line discipline (see *mesgld*(4)) onto the connection.

`term` Read a null-terminated string from the caller, and set environment variable TERM to the result.

`args` Read a null-terminated string from the caller, and save the result as arguments to a possible command.

`s5parms` Extract arguments from the destination address in a way compatible with the DKHOST network software used by System V Datakit implementations, and save for later use.

`cmd(`*x*`)` Execute shell command *x*, with any saved arguments, and with the connection as standard input, output, and error.

`login` Provide a login session with the connection as standard input, output, and error.

`password` Provide a login session, but ignore any local user name; always demand a login and password.

`exec` Use any saved arguments as a shell command to be executed.

`gateout(`*gateway*`)`
Call network address *gateway* and send the connection info there, If all is well, pass the new connection's file descriptor to the original caller: the result is a connection through the gateway. *Gateway* should be a *svcmgr* service, perhaps on some other machine, with action `gateway`.

`gateway(`*localout*`)`
The intended target for `gateout`: read new connection info from the connection, and place a call to the new destination; if it succeeds, loop passing data between the new connection and the original one.

If the file `/usr/ipc/log/svc` can be opened, *svcmgr* prints miscellaneous chatter there, including a record of each service request. The `-d` (debug) option increases the chatter.

FILES

```
/usr/ipc/lib/serv
/usr/ipc/lib/serv.local
/usr/ipc/lib/auth
/usr/ipc/lib/auth.local
/usr/ipc/log/svc
```

SEE ALSO

con(1), *ipc*(3), *dkmgr*(8), *tcpmgr*(8), *ipc*(3)

NAME

swapon − specify swapping device

SYNOPSIS

`/etc/swapon -a`
`/etc/swapon` *name* ...

DESCRIPTION

Swapon specifies additional devices on which paging and swapping are to take place. The system begins by using a single device; *swapon* must be used to enable others.

Usually there is a call to `swapon -a` in *rc*(8). Specific swap devices may be nominated with the second form.

SEE ALSO

rc(8), *vswapon* in *deprecated*(2)

BUGS

There is no way to stop paging and swapping on a device. It is therefore not possible to make use of devices which may be dismounted during system operation.
Possible swap devices must be listed in a table configured into the system; *swapon* can only enable devices in the table.

NAME

sync, update − update disk file systems

SYNOPSIS

```
sync
/etc/update
```

DESCRIPTION

Sync executes the *sync*(2) system primitive, to suggest that all disk writes be completed soon. It is wise to call *sync* before halting the system.

Update calls *sync*(2) every 30 seconds, so that file systems are fairly well up to date if the system crashes. It should be run once from *rc*(8).

SEE ALSO

sync(2), *reboot*(8)

NAME

tcpmgr − accept and place calls via the TCP protocol

SYNOPSIS

`/usr/ipc/mgrs/tcpmgr` [`-m` *outnet*]

DESCRIPTION

Tcpmgr receives and places TCP calls on an Internet TCP/IP network. Outbound TCP calls may be placed by calling *ipcopen* with network name `tcp` (see *ipc*(3)). Inbound calls to TCP port *n* are handed to the local service listed for that port in `/usr/ipc/lib/inservices`, or to service `tcp.`*n* if there is no listing.

Option `-m` tells *tcpmgr* to claim to place outbound calls for network *outnet* rather than `tcp`.

The TCP protocol runs atop one or more IP networks. *Tcpmgr* arranges to receive all inbound TCP calls on all active IP networks, but other arrangements must be used to activate the IP networks themselves; see *ipconfig*(8).

Tcpmgr records its activity in file *outnet* in directory `/usr/ipc/log`, default `/usr/ipc/log/tcp`.

This command is usually run once from *rc*(8).

FILES

`/usr/ipc/log/tcp`	
`/dev/tcp*`	TCP network devices
`/dev/iptcp`	IP channel for the TCP protocol
`/usr/ipc/lib/inservices`	mapping between service name and port number

SEE ALSO

con(1), *ipconfig*(8), *svcmgr*(8), *ipc*(3)

NAME

upas, rmail, translate − mail delivery system

SYNOPSIS

`rmail` *person ...*

`/usr/lib/upas/translate` *name*

`/usr/lib/upas/auth` *sender receiver*

DESCRIPTION

Users send mail by *mail*(1). Remote machines use *rmail.* Both call on *upas* programs.

Mail addresses are interpreted according to rewrite rules as described below. When the addresses for a *mail* or *rmail* command have been interpreted, they are bundled by network and passed to network-specific handlers, such as *route.inet*.

Translate looks up a mail *name* in an alias list (see *mail*(6)) and places the result on standard output.

Auth is called by *upas* to authorize mail delivery for each *sender, receiver* pair. The mail is accepted if the previous hop was a trusted gateway machine in `/usr/lib/upas/gateways`, or all the machines in the source or destination path are in `/usr/lib/upas/attlist`.

Rewrite rules

Each line of the file `/usr/lib/upas/rewrite` is a rule. Blank lines and lines beginning with `#` are ignored.

Each rewriting rule consists of (up to) 4 strings:

pattern A regular expression in the style of *regexp*(3). The *pattern* is applied to mail destination addresses. The pattern match is case-insensitive and must match the entire address.

type The type of rule; see below.

arg1 An *ed*(1) style replacement string, with `\`*n* standing for the text matched by the *n*th parenthesized subpattern.

arg2 Another *ed*(1) style replacement string.

In each of these fields the substring `\s` is replaced by the login id of the sender and the substring `\l` is replaced by the name of the local machine.

When delivering a message, *mail* starts with the first rule and continues down the list until a pattern matches the destination address. It then performs one of the following actions depending on rule type:

`>>` Append the mail to the file indicated by expanding *arg1*, provided that file appears to be a valid mailbox.

`|` Pipe the mail through the command formed from concatenating the expanded *arg1* and *arg2.*

`alias` Replace the address by the address(es) specified by expanding *arg1* and recur.

`translate` Replace the address by the address(es) output by the command formed by expanding *arg1* and recur.

`auth` Call the program in expanded *arg1* and authorize(reject) the mail if it returns a zero(non-zero) return code.

Mail expands the addresses recursively until each address has matched a `>>` or `|` rule or until the recursion depth indicates a rewriting loop (currently 32).

An `auth` operator is only applied once per address. If no `auth` rule is encountered, the mail is accepted.

If several addresses match `|` rules and result in the same expanded *arg1*, the message is delivered to all those addresses by a single command, composed by concatenating the common expanded *arg1* and each expanded *arg2*. This is meant to make less work of a message to several recipients on the same machine. For example, the rule

```
([^!]+)!(.+)    |   "uux - -a \s \1!rmail"   \2
```

causes `mail r70!pjw r70!ken` to generate the single delivery command `uux -a rob r70!rmail pjw ken`.

EXAMPLES

A sample rewrite file:

```
# local mail
[^!@%]+              translate  "exec translate '&'"
local!([^!@%]+)      >>     /usr/spool/mail/\1
\l!(.+)              alias  \1

# convert %@ format to ! format
(_822_)!((.+)!)?([^!]+)[%@]([^!%@]+)  \
                     alias  \1!\2\5!\4
([^!]+)[%@]([^!@%]+) alias  _822_!\2!\1
_822_!(.+)           alias  \1

# special domains
[^!]+wisc\.edu!.+    alias  xunet!&

# network gateways
(csnet|bitnet)!(.+)  alias  inet!&
acsnet!.+

# real networks
inet!([^!]+)!(.+)    |      "/usr/lib/upas/route.inet '\s' '\1'" "'\2'"
([^!]+)!(.+)         |      "/usr/lib/upas/route '\s' '\1'" "'\2'"
```

FILES

`/usr/lib/upas/namefiles`	list of files to search
`$HOME/lib/names`	private aliases
`/usr/lib/upas/rewrite`	rewriting rules
`/usr/lib/upas/attlist`	known AT&T machines
`/usr/lib/upas/gateways`	machines that check mail authorization reliably
`/usr/lib/upas/translate`	alias lookup
`/usr/lib/upas/route.*`	mail interfaces to specific networks
`/usr/lib/upas/route`	interface to local (i.e. AT&T) Datakit network
`/bin/mail`	shell file that calls the mailer
`/usr/lib/upas/send`	actually delivery program
`/bin/rmail`	linked to /usr/lib/upas/send
`/usr/spool/mail/*`	mailboxes
`/usr/spool/mail/mail.log*`	delivery logs
`/n/bowell/usr/lib/upas/mkfile`	updates various mail and uucp files
`/etc/passwd`	authentication
`/tmp/ma*`	temp file
`/tmp/ml*`	lock file
`$HOME/dead.letter`	unmailable text

SEE ALSO

uucp(1), *mail*(1), *mail*(6), *smtp*(8)

D. L. Presotto and W. R. Cheswick, 'Upas—a simpler approach to network mail', this manual, Volume 2

NAME

uucico, uusched, uuxqt, kick, debug − uucp file transport and execution

SYNOPSIS

`/usr/lib/uucp/uucico` [`-r1 -s` *system*] [`-x` *debug*] [`-d` *spool_directory*]

`/usr/lib/uucp/uusched` [`-x` *debug*] [`-u` *debug*]

`/usr/lib/uucp/kick` *system*

`/usr/lib/uucp/debug` *system*

`/usr/lib/uucp/uuxqt` [`-s` *system*] [`-x` *debug*]

DESCRIPTION

Uucico transfers files between systems for *uucp*(1). It is normally invoked by *login* for an incoming connection (a slave), or by *uusched* to call out to another system (a master). The options are

`-r1` This is a master; option `-s` is required. In the absence of `-r1` the process is a slave and expects to speak to a master on standard input and standard output.

`-s` *system* Call the named *system.*

`-x` *debug* Turn on debugging output; *debug* is a single digit, larger for more output.

Uusched invokes *uucico* for each system with work pending. It is called by *uucp* and *uux*(1) after work is queued, and should be invoked regularly by *cron*(8). The options are

`-x` *debug* As for *uucico.*

`-u` *debug* Invoke *uucico* with option `-x` *debug.*

Kick invokes *uucico* in the background to call out to the specified *system*. *Debug* causes a call with a moderate amount of debugging output (`-x4`). Both attempt to remove existing system status information for *system*, so that a call will be attempted regardless of recent failures.

Uuxqt executes commands requested remotely by *uux*(1). It searches the *uucp* spool directories looking for filenames starting with `X.`, checks the *Permissions* file to see that all required data files are available and accessible and that the requested command is permitted for the requesting system, and invokes the command if all is well.

Before a command is invoked, the file creation mask (*umask*(2)) is set to 0 and these environment variables are set:

`UU_MACHINE`	the name of the last sending machine
`UU_USER`	the user that sent the job
`PATH`	set to `/bin:/usr/bin`
`USER`	set to `uucp`

FILES

`/usr/lib/uucp/Systems*`	
`/usr/lib/uucp/Permissions`	
`/usr/lib/uucp/Devices`	
`/usr/lib/uucp/Maxuuscheds`	how many copies of *uusched* may be active at once
`/usr/lib/uucp/Maxuuxqts`	ditto for *uuxqt*
`/usr/spool/uucp/*`	
`/usr/spool/uucppublic/*`	
`/usr/spool/uucp/LCK*`	

SEE ALSO

uucp(1), *uux*(1), *uustat*(1)
D. A. Nowitz, 'UUCP Administration', this manual, Volume 2

BUGS

System and user names received by *uuxqt* should not be believed.

NAME

uucleanup − uucp spool directory clean-up

SYNOPSIS

`/usr/lib/uucp/uucleanup` [*options*]

DESCRIPTION

Uucleanup removes old files from the *uucp* spool directories. It is typically called by `/usr/lib/uucp/uudemon.cleanu`, which may be run regularly by *cron*(8). The options are:

`-C`*days* Remove `C.` (control) files that are at least *days* old, and send a message to the user who queued the job.

`-D`*days* Remove `D.` (data) files that are at least *days* old. If the data file appears to contain a mail message, an attempt will be made to deliver it; if it contains a netnews article from another system, it will be handed to *rnews*.

`-W`*days* Send a warning to the user who queued any jobs (`C.` files) at least *days* old. If one of the data files for the job appears to be a mail message, the message is included in the warning.

`-X`*days* Remove any `X.` (remote execution) files at least *days* old.

`-m`*string* Include *string* in warning messages. The default is 'See your local administrator to locate the problem.'

`-o`*days* Remove any other files that are at least *days* old.

`-s`*system* Examine only files associated with *system*.

By default, `C.` files generate a warning when one day old, and are removed after 7 days; `D.` files are removed after 7 days; and `X.` and other files are removed after 2 days.

FILES

`/usr/lib/uucp`	directory with commands used by *uucleanup*
`/usr/spool/uucp`	spool directory

SEE ALSO

uucp(1)

D. A. Nowitz, 'UUCP Administration', this manual, Volume 2

NAME

vmstat − report virtual memory statistics

SYNOPSIS

`vmstat` [`-st`] [*interval* [*count*]]

DESCRIPTION

Vmstat reports statistics about certain system activity. Option `-s` prints totals for miscellaneous events since the last boot. Option `-t` reports on paging events.

In the absence of other options, the optional *interval* argument causes *vmstat* to report once each *interval* seconds, repeated *count* times or forever.

The default format fields are:

`procs` information about numbers of processes in various states:
- `r` in run queue
- `b` blocked for resources (I/O, paging, etc.)
- `w` runnable or short sleeper (< 20 secs) but swapped

`memory` use of virtual and real memory:
- `avm` number of pages belonging to processes that have run in the last 20 seconds
- `fre` size of memory free list

`page` paging activity, averaged each five seconds, in units per second:
- `re` page reclaims (simulating reference bits)
- `at` text pages recovered from memory
- `pi` page-in events
- `po` page-out events
- `fr` pages freed per second
- `de` anticipated short term memory shortfall
- `sr` scan rate: pageout daemon rpm

`faults` trap rates, averaged each five seconds, in units per second:
- `sy` system calls
- `cs` process context switches

`cpu` percentage use of CPU time:
- `us` user time, both normal and low priority
- `sy` system time
- `id` cpu idle time
- `st` stream queue processing time

FILES

`/dev/kmem`
`/unix`

BUGS

This program is never up to date.

NAME

wall − write to all users

SYNOPSIS

`/etc/wall`

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by ‘Broadcast Message ...’, to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

FILES

`/etc/utmp`

SEE ALSO

write(1)

DIAGNOSTICS

‘Cannot send to ...’ when the open on a user’s tty file fails.

NAME

worm, jukebox − optical disk utilities

SYNOPSIS

`worm mkfs` [`−f`*device*] [`−c`*comments*] [`−b`*blksz*] [`−n`*nblks*] [`−v`*newvol_id*] *vol_id*

`worm stat` [`−f`*device*] [`−F`*n*] [`−v`] [*vol_id*]

`worm ls` [`−f`*device*] [`−l`] [*file ...*]

`worm rm` [`−f`*device*] *vol_id* [*file ...*]

`worm mv` [`−f`*device*] *vol_id src dest*

`worm write` [`−f`*device*] *vol_id* [*file ...*]

`worm read` [`−f`*device*] [`−dm`] *vol_id* [*file ...*]

`worm cat` [`−f`*device*] *vol_id file*

`worm copy` [`−v`] [`−m`*min_free*] [`−f`*src_dev*] *src_vol_id dest_dev dest_vol_id*

`worm offline` [`−f`*device*]

`worm btree` [`−f`*device*] *vol_id*

`worm dir` [`−f`*device*] *vol_id*

`worm tmpdir` [`−f`*device*] *vol_id*

`worm mount` [`−w`*secs*] [*vol_id*]

`jukebox` [`−aemprsuU`] [`−w`*secs*] [*vol_id*]

DESCRIPTION

The *worm* programs manipulate arbitrary files. They are intended for use with the raw device associated with a Write-Once Read-Many (WORM) optical disk. The default device is `/dev/worm0`. Other devices are specified by `−f`*device* and a device name of a single digit *n* is taken as an abbreviation for `/dev/worm`*n*. Most of the commands implement a simple file system. Programs just wanting a raw device should still use `worm mkfs` so that the disk is properly labeled. The *vol_id*, or label, should be unique and by convention, the vol_id's for the A and B sides of a disk should be the same string suffixed by a lowercase `a` and `b` respectively.

Worm mkfs labels an optical disk. The *comments* field is limited to 256 chars. It is purely descriptive and is printed by *worm stat* `−v`. The (default) blocksize is 1024 for our SONY disks. The number of blocks on a disk can be found by *ra*(4) or *scsish*(8); the default size (1,600,000 for single density, 3,250,000 for double density) sets aside 30MB or so as a hedge against oversights. If the disk has already been initialised, its vol_id must match *vol_id*. A new vol_id can be set with `−v`.

Worm stat prints out labeling information including the amount of free space left on the disk. Option *vol_id* turns off all output except exit status: zero if *vol_id* matches that of the disk, one otherwise. Option `−F` similarly exits with status zero if the disk has more than *n* free blocks, otherwise three. Option `−v` produces more output.

Worm ls simulates an emasculated *ls*(1).

Worm rm makes the specifed files unavailable to the rest of the *worm* commands.

Worm mv renames *src* to *dest*.

Worm write copies files onto the WORM. If no file arguments are given, filenames are read one per line from standard input. The total number of files and bytes is printed on standard output.

Worm read restores files from the WORM. If no file arguments are given, filenames are read one per line from standard input. Option `−d` causes directories to be created as needed. Option `−m` restores the original modification times.

Worm cat copies the named file from the WORM to the standard output.

Worm copy copies files directly from one disk to another. The names of the files to be copied are taken from standard input; groups (separated by blank lines) will be kept together. The names are typically generated by `worm ls`. The `-v` option prints out progress and summary information. The copy will terminate before copying a group that would leave the destination volume with less than *minfree* (deafult value is 40000) blocks free.

Worm offline makes the WORM go offline, ready for ejecting. This command is harmless; accessing an offline drive will cause it to spin up and go online without operator intervention. *Worm offline* only takes effect after the last close of the WORM and as a bonus, applies to any MSCP device such as an RA81.

Worm tmpdir saves a copy of the directory in `/usr/worm/tmp/`*vol_id* if the directory `/usr/worm/tmp` exists. This will speed up subsequent access substantially, although it will still be slower than *worm btree* below. On the other hand, *worm tmpdir* typically takes 5 minutes to run (on a VAX 11/750) whereas *worm btree* takes about 45 minutes.

Worm btree constructs a new directory for the whole disk (in the form of a *cbt*(1) database). The new superblock is at zero. All the worm commands go faster with such an index but it is intended to be done just once, after the disk is complete. The directory occupies of the order of 10MB but may be more. If you really have to add more files to the disk, you need to write zeros on the first 1K block of the WORM before using *worm write*.

Worm dir takes the btree directory from the disk and stores in `/usr/worm/dirs`. Future uses of the disk will be much faster.

Worm mount returns the device on which the disk labelled *vol_id* is mounted. If the drive(s) are busy and you have a jukebox, the `-w`*s* option tells how many seconds to wait before failing. The default is wait forever. If no *vol_id* is given, print the drive status.

Jukebox manages the disks in the SONY jukebox. There are several options (default is `-s`):

`-a` Allocate a blank disk and label it *vol_id*. Use *worm mkfs* to change any fields from their default value.

`-e` Eject the disk labeled *vol_id*. To physically retrieve the disk, press the `OUT` button (the `OUT READY` light should be on). Repeat until the `IN READY` light goes on.

`-m` Mount the disk labelled *vol_id* in some drive and print the drive number on standard output.

`-p` Print the list of disks in the jukebox.

`-r` Rebuild the list of disks by examining each disk in the jukebox. Do not do this unless you are sure you need to. If *vol_id* is given, it should be one of the following letters and governs how disks are assigned shelf numbers. The default is to leave the shelf number unchanged. Other options (mainly useful for demos) are `c` (compresses the disks in the jukebox towards the bottom or lower numbered shelves), `r` (distributes the disks randomly), and `s` (sorts the disks by vol_id).

`-s` Print the status of the jukebox.

`-u` Unload offline disks back onto their shelves.

`-U` Unload all disks (offline or not) back onto their shelves.

`-w`*secs* This option only affects the behavior of `-m`. If all drives are busy, try again for *secs* seconds before failing.

To load a disk into the jukebox, press the `IN` button on the jukebox when the `IN READY` light is on. After the shutter opens, push the disk in firmly. The disk (blank or initialised) is not examined immediately but on demand.

Etiquette

Vol_ids should be unique as discussed above. The file `/n/wild/usr/worm/vol_ids` contains known vol_ids. The commands for reading and writing require vol_id's to guard against accessing the wrong disk.

The recommended protocol for changing disks is if no one appears to be using the drive (by using *ps*(1)), execute *worm offline* and go to the drive. If, and only if, the drive has the DRIVE OFF (middle) light on, hit the EJECT button and change disks. If the light is not on, then someone is still using the disk and you should wait until they are done before hitting EJECT.

Programming considerations

Programs should not depend on writing any block more than once; however, our SONY optical disks implement a small number of multiple writes via bad block replacement. A *read*(2) of an unwritten block returns with an errno of `ENXIO`. On Vaxes, the WORM is an MSCP device; thus geometry information can be fetched as in *ra*(4).

For maximum speed, read and write in large blocks (preferably 63K) and avoid seeks. A seek across the whole disk takes about 1 second.

The device `/dev/worm?` is simply an appropriate raw *ra*(4) device, partition 7 (the whole disk).

EXAMPLES

```
worm mkfs -c"512x512x24 movies" tdmoviesa
worm write tdmoviesa < filenames
worm read -d tdmoviesa bumblebee/act2/frame1
```

FILES

```
/dev/worm?
/n/wild/usr/worm/vol_ids
/n/wild/usr/worm/jukedir
```

SEE ALSO

backup(8), *scsish*(8), *backup*(1)

BUGS

The output of *worm ls* is not necessarily sorted.

9 TELETYPE 5620

NAME

intro − introduction to 5620-related software

SYNOPSIS

`PATH=`*stuff*`:/usr/jerq/bin`

DESCRIPTION

Section 9 of this manual lists software for running or supporting Teletype DMD-5620 terminals. Subsections 9.1-9.7 mirror the purposes of the preceding sections 1-7, with 9.1 being commands, 9.6 being games, etc. Command synopses assume that the shell search path includes `/usr/jerq/bin`.

Few commands deal with a 5620 in native mode. *32ld*(9.1) loads programs into the terminal and *mux*(9.1) starts the characteristic 'layer' or window system. Almost all other commands in section 9 either run on Unix or within *mux* layers.

A layer is technically a virtual terminal, but is almost indistinguishable in software from a real terminal; in particular, the interface described in *ttyld*(4) applies to layers, except for the additional editing capabilities discussed in *mux*(9.1).

The commands in sections 9.1 and 9.6 run on Unix, but most implicitly call *32ld* to down-load a program that replaces the default terminal process running in the layer. To Unix the interface is still that of a terminal; in particular `/dev/tty` (see *fd*(4)) is always connected to the layer. The default *mux* terminal program implements the teletype driver function itself. When a program is down-loaded, there is no teletype driver; programs that desire one must push the teletype line discipline on the stream, and arrange to pop the line discipline on exit; see *stream*(4) and *ttyld*(4). Some commands may simply emulate other terminals by down-loading a terminal program (see *term*(9.1); others, such as the text editor *sam*(9.1), are really two programs — one on Unix and one in the layer — communicating using standard input/output on Unix and `sendchar()/rcvchar()` in the terminal; see *request*(9.2).

There is an identity between bitmaps and layers in the graphics software. Graphic objects are bitmaps. The *newlayer*(9.2) primitives that operate on layers are aliased to bitmap primitives of (9.3), and the data structures are isomorphic. When running under *mux*, a programmer need not consider layers as graphical objects at all; the operating system checks the arguments to the graphics primitives and dispatches the appropriate operator depending on the type of the argument. Except in stand-alone software, layers are an invisible implementation detail.

Teletype 630

Mux and various programs that run in *mux* layers have been ported to Teletype DMD-630 terminals. The ported software is not available on all machines and is not specifically documented. Look in `/usr/630/bin` to see what's there.

FILES

`/usr/jerq/bin`	jerq-related Unix object programs
`/usr/jerq/mbin`	terminal programs, usually down-loaded automatically by programs in `/usr/jerq/bin`
`/usr/630/bin`	630-related Unix object programs
`/usr/630/lib`	terminal programs

SEE ALSO

32ld(9.1), *mux*(9.1), *stream*(4)

BUGS

There are two mechanical-drawing programs, *cip* and *ped*, two 'artistic' drawing programs, *paint* and *twid*, one 'graphic' drawing program, *brush*, two pixel-level drawing programs, *icon* and *jf*, all for working on binary images. None dominates.

NAME

32ld − bootstrap loader for 5620

SYNOPSIS

`32ld` [*option* ...] *file* [*argument* ...]

DESCRIPTION

32ld loads the MAC-32 object *file* for execution in a 5620 connected to the standard output. When loading into a *mux*(9.1) layer, the *arguments* are passed to the program as in Unix. The options are:

`-d` Print on the standard error file the sizes of the text, data and bss segments of *file*. The standard error must be separated from the standard output to avoid corrupting the down-load.

`-p` Print down-loading protocol statistics on the diagnostic output (for stand-alone loading only).

`-z` Load the process but don't run it. It may be started using *3pi*; see *pi*(9.1). This option works only under *mux*.

The environment variable `JPATH` is the analog of the shell's `PATH` variable to define a set of directories in which to search for *file*.

SEE ALSO

jx(9.1), *mux*(9.1)

NAME

3cc, 3as, 3ar, 3ld, 3nm, 3size, 3strip, cprs − MAC-32 C compiler

SYNOPSIS

`3cc` [*option* ...] *file* ...

`cprs` *infile outfile*

DESCRIPTION

3cc is the C compiler for the MAC-32 microprocessor in the Teletype DMD-5620 terminal. Its default action is to compile programs to run under the *mux*(9.1) environment.

The behavior of *3cc* is similar to *cc*(1). Here are listed only options with special behavior for 5620s.

`−J` Compile the named programs, and link them for running stand-alone on a 5620 terminal.

`−O` Invoke an object-code improver (not recommended).

`−m` Compile the named programs for ordinary (non-jerq) environments.

`−D`*name*`=`*def*
`−D`*name*
Define the *name* to the preprocessor, as if by `#define`. If no definition is given, the name is defined as `1`. The symbol `MUX` is predefined unless `−J` or `−m` is set.

`−I`*dir* `#include` files whose names do not begin with `/` are always sought first in the directory of the *file* argument, then in directories named in `−I` options, then in directories on a standard list, which includes `/usr/jerq/include`.

Associated object-code manipulating programs exist. Their behavior is similar to the programs cited below. The loader, assembler and archive program are System V derivatives, and are slightly different in behavior; see the System V manuals. For typical uses, these differences are irrelevant. The support programs include:

3as assembler, see *as*(1)
3ar archive, see *ar*(1) (there is no *3ranlib*)
3ld link editor, see *ld*(1)
3nm name list, see *nm*(1), doesn't work on archives
3size object code size, see *size*(1)
3strip symbol table; see *strip*(1). (`−r` is mandatory for *mux*-runnable binaries.)

3strip has no `−g` flag; but *cprs* removes redundant symbol table entries while copying *infile* to *outfile*.

FILES

`a.out`	loaded output
`/tmp/ctm*`	temporary
`/lib/cpp`	preprocessor
`/usr/jerq/lib/m32/comp`	compiler
`/usr/jerq/lib/m32/optim`	optimizer
`/usr/jerq/lib/*.o`	runtime startoff, etc.
`/usr/jerq/lib/libc.a`	standard library
`/usr/jerq/lib/libj.a`	stand-alone graphics library `/usr/jerq/lib/libmj.a` mux-runnable graphics library (default)
`/usr/jerq/lib/muxmap`	loader I-file
`/usr/jerq/include`	standard directory for `#include` files

SEE ALSO

System V manuals for *3ar, 3ld, 3as* and *cprs* documentation.

NAME

blitblt, menudrop − save or print a screen image

SYNOPSIS

`blitblt` [`-p` *command*]

`menudrop`

DESCRIPTION

Blitblt copies a selected area of a *mux*(9.1) screen into a file or to a program. It is menu-driven off button 3 to select a rectangular area and to treat it by flipping the border from wide to narrow and back, inverting video, saving the selcted area in a file, or sending it to a program, usually for printing. Details of certain menu items:

gram in midmenu may get the same effect.

NAME

brush − painting program

SYNOPSIS

`brush` [`-f` *fontdir*] [`-p` *picdir*] [`-t` *texdir*]

DESCRIPTION

Brush paints images under mouse control. Options are

`-f` font directory (`/usr/jerq/font` by default)
`-p` the directory in which to keep pictures (current directory by default)
`-t` texture directory, where brushes and shades live (current directory by default)

In general, button 1 draws, button 2 erases; the cursor assumes the shape of the current brush. Button 3 is used to select options, sweep out areas, or cancel operations in progress.

The borders on either side of the drawing area contain menus of available shades and brushes. The current brush and shade are outlined by boxes. To choose another, click button 3 at it.

The top border contains a help area, drawing options, and certain commands. Selections are made by pointing with button 3. Some cycle through options; others bring up menus. The items are:

help Icons in three boxes indicate what buttons 1, 2, and 3 will do at any given time:

paintbrush	draw with this button
pencil eraser	erase with this button
menu with cursor	menu on this button
thumbs down	cancel or finish an operation
pointing hand	indicate a point
square with arrow	sweep a rectangle
circle with arrow	sweep a circle
skull	exit the program

smooth Smooth the contours of magnified images.

align Force circles, discs, text, and other images to align with texture cell boundaries.

`image` Manipulate the ‘current image’, (box, ellipse, etc.) selected from the drawing menu described below. Button 3 makes the image disappear, reserved for future use. The image menu contains:

`same` Bring back the current image.
`magnify` Sweep a rectangle indicating the size of the magnified image. The numbers that appear are horizontal and vertical magnification factors.
`shrink` Shrink to 1/4 size. Indicate whether image is shaded or black & white.
`flip` Reflect left-right or top-bottom.
`rotate` Rotate counterclockwise or clockwise 90 degrees.
`slant` Drag the current image rectangle into a parallelogram.
`outline` Replace the current image with its outline.
`shadow` Draw a ‘shadow’ behind the current image.
`shadow` Draw a ‘shadow’ behind the current image.
`new` Make a new image by copying a rectangular portion of the screen.

To move an image on the screen, select `new` from the `image` menu. Sweep the area to be moved, click button 2 to erase it, move it, and click button 1 to draw it.

drawing style
Select continuous curves, dotted lines, disconnected dots, or an ‘airbrush’ effect when painting.

constrain Select freehand (wavy-line icon) or horizontal-vertical drawing (angular icon).

reflect Draw symmetric figures. The icon shows the symmetries: `x=0` (left-right), `y=0` (top-bottom), or `both`, relative to the center of the screen.

draw mode
Set the drawing mode to one of `or`, `xor`, `store`, `and`, `copy` (preserves interior whitespace of images).

text style Set the text style to one of `normal`, `outline`, `bold`, `shadow`, `italic`.

font name Set the text font. Menu selection `new` prompts for a font name from the font directory.

`i/o` Interact with host machine. Menu items are:

- `save` Copy screen, brushes, or shades to a file. Prompts for a file name, starting with the default picture directory (if any). Backspace past this if you wish to save elsewhere; hit return to quit. Next sweep a rectangle to be saved. Bitmaps are saved in *bitfile*(9.5) format.
- `recall` Prompts for a file name. The recalled picture becomes the current image.
- `exit` Leave the program. Confirm by two clicks on button 3.

The menu on button 3 in the drawing area contains these selections:

`lines` Indicate first point, then position cursor with rubber band line for subsequent lines. Button 1 draws, button 2 erases.

`curves` Indicate first control point, then position cursor with rubber band line for subsequent control points. A curve (spline) will be drawn (erased) using these control points, depending on whether the last button hit is button 1 (draw), or button 2 (erase).

`box`
`ellipse`
`disc` (A disc is a filled ellipse). Sweep a rectangle; numbers show the dimensions. A single dot marks the center of an ellipse. The image becomes the current image; use buttons 1 and 2 to draw or erase with it.

`string` Type in text. The string becomes the current image.

`texture` Sweep a rectangle. The current image becomes a rectangle of this size textured with current shade.

`fill` Sweep a rectangle, then indicate interior seed points using button 1, or button 3 to quit. Enclosed regions will be filled with current shade. Any button cancels the fill.

`clear`
`invert` Sweep a rectangle to be cleared or color-inverted.

`fade` Sweep a rectangle. Holding button 2 will fade this area, as if erasing in spray paint mode with a random pattern instead of a shade.

`new brush`
`new shade`
Menu select whether to edit or snarf from screen (with button 3). If editing, the current brush (shade) will appear magnified in upper left corner. Edit with buttons 1 and 2, quit with 3. Several ‘spare’ brushes appear at the bottom of the brush menu.

`details` Select an area with box cursor to be magnified for detailed editing.

SEE ALSO

paint(9.1), *mbits*(6), *bitfile*(9.5)

BUGS

The smoothing operation fully smooths only for magnification factors that are powers of two.
Bitmaps moved off the top or bottom of the physical screen can pick up noise.
`Copy` mode generates a mask the first time a given image is moved. This can take a while for large images. Be patient.

NAME

cip − draw pictures for typesetting

SYNOPSIS

`cip`

DESCRIPTION

Cip prepares or modifies *pic*(1) descriptions, which may subsequently be typeset. It provides a palette of shapes: box, circle, ellipse, line, arc, spline, and text. Button 1 selects shapes from the palette or the screen. Button 2 places or redraws (`edit`, `move`) shapes. Button 3 controls menus.

File names and text strings are entered from the keyboard. Keyboard input always ends with a newline. A current file name is remembered and offered for file operations; backspace over it to substitute a new name, or type newline to accept it.

The `define macro` menu item allows a box to be swept, collecting all contained shapes into a group. Groups are selected as whole. When a group is selected, a special menu appears. Item `separate` dissolves the group; `reflect x` reflects about a horizontal midline; after `copy` button 2 places copies at the cursor. Item `edit` confines activity to the group. Changes are reflected in all copies of the group. To leave the group, click button 1 at `edit depth`.

SEE ALSO

pic(1), *ped*(9.1)
Sally A. Browning, 'Cip User's Manual: One Picture is Worth a Thousand Words', this manual, Volume 2

BUGS

Cip cannot handle arbitrary *pic* programs, just programs in the style that it produces.

NAME

flicks, fflicks, preflicks, 2mux − movie graphics for 5620

SYNOPSIS

`flicks` [`-fmte`] *file* ...

`preflicks` [`-fmtvloics`] *file* ...

`fflicks` [*flickfile*]

DESCRIPTION

Flicks interprets each of the *files* as a grey-scale frame in the form of *picfile*(5) (or a square raster of unsigned bytes), dithers them, and displays them on the terminal. Once the frames have been downloaded the frames can be played as a movie, controlled by a menu on button 3. Most menu selections are self-explanatory. *Step* shows individual frames, stepping forward with button 1, or backward with button 2. Button 3 brings back the main menu.

The size of a frame is an option:

`-f` full size: same size as the input (typically 512×512)

`-m` medium size: half the input size (typically 256×256) default

`-t` tiny size: quarter of the input size (typically 128×128)

If only one image is processed, full size is the default. For more than 11 pictures, tiny size is default. Anything in between is medium size by default.

The frames are rendered with dithering by default, and with error propagation if `-e` is specified.

Fflicks downloads frames that have been preprocessed by *preflicks* into a *flickfile*. *Fflicks* downloads much faster than *flicks*.

The options for *preflicks* include `-f`, `-m`, and `-t` as for *flicks*, plus

`-l` Use logarithmic dither.

`-v` Chatter on standard error.

`-o` Write the flickfile onto standard output; by default output goes into file `_movie`.

`-i` Print a summary of the contents of the flickfiles.

`-c` Catenate named flickfiles onto the standard output.

`-s`*X*`,`*Y* the (one only) input file is assumed to be a sequence of *X*×*Y*-byte frames. If *X* and *Y* may be omitted, 512×512 is assumed.

Fflicks display is controlled by a menu on button 3. The selection ‘movie rate’ tries to run the display at 24 frames/sec.

Frames prepared with *preflicks* are compacted. Thus *fflicks* can play a longer sequence than *flicks*: up to roughly 120 medium sized or 480 tiny frames (20 seconds of movie). Still longer sequences (about twice as long) can be downloaded if *fflicks* is run within *2mux* instead of *mux*. The price of compaction is speed. Menu selections are available for uncompacting some (even- or odd-numbered) frames.

FILES

`_movie`
`/usr/jerq/lib/2term`

SEE ALSO

pico(1), *picfile*(5), *rebecca*(9.1), *flickfile*(9.5), *movie*(9.1)

NAME

getfont − replace terminal's default font

SYNOPSIS

`getfont` [*option* ...] [*font*]

DESCRIPTION

Getfont reads font data from file *font.* The current layer and subsequently created layers use this font as *defont*; see *string*(9.3). If *font* does not directly name a file, it is looked for in directory `/usr/jerq/font`.

The options are:

`-m` The font change applies to the basic *mux*(9.1) menu as well as to layers.

`-l` The font change applies to the current layer only.

`-d` Print debugging information about fonts before and after.

Getfont discards inaccessible fonts. To reclaim store without loading a font, call it with no *font* argument.

EXAMPLES

`getfont pelm.10`
Larger type for demos and eyesight problems.

`getfont defont`
Restore the original font.

FILES

`/usr/jerq/font`

SEE ALSO

string(9.3), *font*(9.5), *font*(6)

NAME

graphdraw graphpic − edit (combinatoric) graphs, convert to pic files

SYNOPSIS

`graphdraw` [*file*]

`graphpic` [*option ...*] *file*

DESCRIPTION

Graphdraw interactively edits and displays undirected graphs, and can also be used to display real-time animation of algorithms. If a *file* is mentioned, the graph stored in that file is edited.

Click button 1 in command line (at bottom of window) to type in commands:

`r` *file*	Read file and display graph.
`w` *file*	Write current graph to file.
`cd` *directory*	Change directory.
`!` *program file*	Execute animation *program* with *file* as input.
`q`	Quit.

Button 3 gets a menu of actions, which are usually accomplished by pointing with button 1. The parenthesized equivalents in the following list are explained under ‘Algorithm animation’.

`create vertex`
: Vertex is placed where button 1 is clicked. (`vc` *x y*)

`delete`
: Delete selected vertex and associated edges. (`vd` *i*)

`move` Selected vertex moves with mouse until button 1 is released. (`vm` *i*)

`copy` Copy of selected vertex and associated edges moves with mouse. (`vc` *i x y*)

`create/delete edge`
: Point to first endpoint and click button 1. Point to second endpoint and click button 1. Continue selecting second endpoints with button 1. To unselect first endpoint, click button 2. (`ec` *i j* / `ed` *i j*)

`restart`
: Click button 1 to clear screen and discard current graph. (`pr`)

`standard window`
: Restart and reshape window to standard size, in which the drawing area is square and as large as possible.

`small/large/no grid`
: Impose/remove visible grid to which all new coordinates will be rounded.

`exit` Click button 1 to confirm.

`labels menu`

- `label vertex`
  Select vertex with button 1. Current label appears on command line. To accept it, click button 1. Otherwise, type in new label and hit return. (`vl` *i w*)
- `number vertices`
  Vertex labels are set to the consecutive integers 1,2,...; this is the default. (`vn`)
- `label edge`
  Default is `1`. (`el` *i j w*)
- `show/hide vertex labels` (`vs`, `vh`)
- `show/hide edge labels` (`es`, `eh`)
- `turn Euclidean edge labels on/off`
  Distances are measured in pixels. (`ee`)
- `show/hide sum of edges`

`graphics menu`

- `light/heavy/empty/full/invisible vertex`
  Select style from menu with button 3; select vertices to change with button 1. The default is `light`. (`vg` *i c*)

`light/heavy edge` (`eg` *i j c*)

`macros menu`

Arrange for sets of vertices to act together. Actions on any vertex in the set apply to the whole set. Copying duplicates edges internal to the set. Creating an edge between vertexes in two different sets creates edges from every vertex in one set to every vertex in the other (bipartite subgraph).

`select set` Sweep a rectangle around the set with button 1. Dissociate conflicting sets.

`unselect set` Dissociate set containing selected vertex.

`shrink/expand set`
Selected set is shrunk/expanded about its center.

`reshape set` Selected set is redrawn in swept rectangle.

`complete/disconnect subgraph`
Create/delete edges between every pair of vertices in a set.

Graphpic is a filter which, when applied to a file in graphdraw format, outputs *pic* code for the graph. The options are:

`-v` Print vertex labels.

`-e` Print edge labels.

`-i` Optimize for imagen printer (default is d202).

File format

Graphs are stored as adjacency lists.

First line: *n m t*, where *n* is the number of vertices, *m* is the number of edges, and *t* is an optional graph type. The only legal type is the default type `u` (undirected).

For each vertex, an initial line: *d w x y c*, where *d* is the degree of the vertex, *w* is its label, *x* and *y* are its coordinates in the window, and *c* is an optional graphics code, `L`=light (default), `H`=heavy (circled dot), `F`=full (large bullet), `E`=empty (empty circle), `I =invisible.` Window coordinates will be scaled to fit when graph is read in.

After the initial line follow *d* lines for the vertex's edges: *i w c*, where *i* is the index (1 to *n)* of the other endpoint, *w* is the edge label, and *c* is an optional graphics code, `L` or `H`.

Algorithm animation

The typed command `!`*program file* causes the standard output of *program* to be captured by the host and interpreted as commands to *graphdraw*. The resulting movie can be killed or temporarily halted from the the terminal by clicking button 2 and choosing the desired option from the resulting menu.

Animation codes (defined parenthetically with menu items above) appear one per line. Their arguments are: *i*, index of a vertex (normally the *i*th to be created); *x, y*, integer coordinates in the range 0 to `maxcoord`; *w* a label; or *c*, a graphic code.

Other animation codes are

`pw` *n* Change the value of `maxcoord to` *n*. Default is 10,000.

`vl` *i w* Give vertex *i* the label *w*.

`pd` *t* Delay program for *t* clicks of the 60Hz clock.

`ps` Halt program until user clicks button 2 to continue.

`pm` *message* Print *message* on command line.

SEE ALSO

dag(1), *pic*(1)

BUGS

It is impossible to move or reshape a *graphdraw* layer, except via `standard window`.

NAME

icon − icon editor

SYNOPSIS

`icon`

DESCRIPTION

Icon is a pixel-level editor for textures and small bitmaps. *Icon* presents a magnified pixel grid and a true-size image. Editing is done on the magnified grid. Pixels can be turned black by pressing the button 1, and white by pressing button 2.

Button 3 provides an iconic menu of editing commands. Some commands require a rectangle to be swept; this is done either by the middle button (which supplies a fixed 16×16 rectangle) or by the right button (for rectangles of any size).

arrow Move region (sweep rectangle and click at destination).

overlapping regions
: Copy region (sweep rectangle and click at destination).

cross Invert region (sweep rectangle).

eraser Erase region (sweep rectangle).

horizontal (vertical) folded arrow
: Reflect region horizontally (vertically) (sweep rectangle).

clockwise (counterclockwise) arrow
: Rotate region deasil (withershins) (sweep rectangle).

horizontal (vertical) sheared lines
: Shear a region horizontally (vertically) (sweep rectangle and point at destination of nearest corner of rectangle).

scaled square
: Scale a region (sweep rectangle and sweep destination rectangle).

tweed pattern
: Texture a region (sweep source rectangle and a (bigger) destination rectangle to be tiled with copies of the source).

glasses Read file (type file name and position the icon by clicking). The subdirectories of `/usr/jerq/icon/` are searched automatically after the current directory.

grid Switch on or off the background grids.

extend region
: Change the size of the drawing area.

pen Write file (sweep rectangle and type file name). See *bitfile*(9.5) for the format.

overlapping rectangles
: Bitblt region (driven by submenus on the right button).

mouse Pick up a 16×16 rectangle and make it the current cursor (click a button to pick up a 16×16 region, and click again to revert to normal).

help Display help information (click a button to revert to normal).

band-aid Undo last drawing operation.

FILES

`/usr/jerq/icon/*/*`

SEE ALSO

bitfile(9.5)

NAME

jf − font editor

SYNOPSIS

`jf` [*file* ...]

DESCRIPTION

Jf edits jerq font files. If *file* does not begin with a slash and is not a font file, it is looked up in a standard font directory.

Jf is mostly mouse- and menu-driven, except when prompting for file names. *Jf* divides its layer into two types of areas: Font displays show all characters in a given font in actual size. When characters are opened for editing, they appear magnified in edit displays.

Button 1 is the 'do it' button. Clicking button 1 inside a font display opens a character for editing; inside an edit display it sets a pixel. It may have other functions selected via menus, in which case the function is indicated by a special cursor.

Button 2 is the 'undo it' button. Clicking button 2 closes a character or clears a pixel, unless conditioned otherwise via menu selection.

Button 3 is the 'menu' button. Clicking button 3 selects a menu, pops control back to the top level, or (when the gunsight cursor shows) picks a font or character to be affected. Sometimes menu selection is the only (non-trivial) option available, as indicated by a 'menu' cursor.

A font is described by several parameters; these are either read from the font file, or set by default by the `make new font` function: `max width` (default 16 pixels), `height` (16)−measured from the top, `ascent` (16)−the distance of the printing baseline from the top, and `range` (1)−the highest-numbered character in the font. (The first character is numbered 0.) All may be changed under the `set sizes` menu. `Squeeze font,` in the `open/close font` menu, reduces max width as much as possible.

Each character has a width, which is shown by the length of the baseline in the edit display. The `char width` may be set under the `set sizes` menu; button 1 sets it to 0, button 2 sets it to a specified pixel within the max width. The quantity `char left` may be used for kerning. If positive, it shifts a character right and causes `max width` to increase if necessary; if negative, the character will be shifted left. Otherwise *char left* is irrelevant to font editing.

The `bit function` menu controls copying among characters in any of the *bitblt*(9.3) Codes: `F_STORE`, `F_CLR`, `F_OR`, `F_XOR`. Press button 3 on the source character; hold it down while moving and release it on the destination.

Several fonts may be open at once. When editing a font, it is often convenient to open a second copy for recovering botched characters.

FILES

`/usr/jerq/font/*`	jerq fonts
`/usr/jerq/include/font.h`	jerq font header file

SEE ALSO

font(9.5)

DIAGNOSTICS

When out of memory or screen area, *jf* ignores the offending operation.

NAME

jx − 5620 execution and stdio interpreter

SYNOPSIS

`jx` *file* [*argument* ...]

DESCRIPTION

Jx downloads the program in *file* to the terminal or layer on its controlling tty and runs it there, simulating standard I/O functions of *stdio*(3). *Jx* works either stand-alone or in a layer.

The `stdout` and `stderr` streams, if directed to the controlling terminal, will be squirreled away during execution and copied to the terminal after the down-loaded program exits.

Programs to be run by *jx* should include `<jerqio.h>` and call `exit()` upon termination in order to restart the default terminal program. Programs to be run stand-alone should be compiled with the `-J` option of *3cc*(9.1). No special options are required for running in a layer.

Stdio(3) functions available under *jx* are

```
getc     putc     fopen    popen    printf   fread
getchar  putchar  freopen  pclose   sprintf  fwrite
fgets    puts     fclose             fprintf
         fputs    access
         fflush
```

Unlike in *stdio*(3), *getc* and *putc* are functions, not macros. *Printf* has only `%d`, `%s`, `%c`, `%o`, and `%x`. `%u` prints an unsigned decimal number. `%D` prints an unsigned long decimal number.

Since *jx* uses *sendchar*, *sendnchars*, and *rcvchar*, *jx* programs should avoid these, and use only the standard I/O routines.

FILES

`/usr/jerq/include/jerqio.h`	
`/usr/jerq/lib/sysint`	standard I/O interpreter
`$HOME/.jxout`	saved standard output
`$HOME/.jxerr`	saved standard diagnostic output

SEE ALSO

request(9.2), *stdio*(3)

BUGS

Keyboard standard input doesn't work; use *kbdchar*; see *request*(9.2).
Stand-alone programs do not receive arguments.

NAME

lens − bitmap magnifier

SYNOPSIS

`lens`



DESCRIPTION

Lens is an interactive screen bitmap magnifier. When it starts, it displays an enlarged image of a magnifying glass in its layer, which becomes a setting sun when *lens* wants to confirm a command to exit.

The first item in the button 2 menu, which rotates among `go`, `peek`, and `stop`, determines the activity of the magnifier. Clicking button 1 serves as an abbreviation for selecting `go` or `peek`. When the magnifier is going, a crawling-bordered rectangle is drawn around the source, and the *lens* window contains the magnified image. The mouse controls the position of the source rectangle.

During peeking, the rectangle last selected while going is re-examined periodically, and the contents are magnified, whether or not the *lens* window is currently selected.

When stopped, the *lens* window is inactive.

The button 2 menu also allows changing the magnification factor. The magnification factors are chosen from the Fibonacci numbers, and menu items for the next size smaller and larger are presented as, e.g., `3x` or `8x`. The current magnification factor is not displayed in the menu, only the next factors larger and smaller. The initial magnification factor is two.

Button 2 may also be used to select the intervals at which peeking updates occur. These intervals are selected, in ticks, from among the powers of two, where a tick is one-sixtieth of a second. These choices are presented as, e.g., `32 ticks` or `128 ticks`. The initial interval between peeks is 64 ticks, approximately one second.

The image window may be controlled by the button 2 menu item which toggles between `inset` and `full size`. In inset mode, the image is displayed inside the image window of the magnified lens icon. In full size mode, the image is displayed in the entire *lens* window.

The final button 2 menu entry is *exit.* A setting sun is displayed, and button 3 must be clicked to confirm.

BUGS

While going, the display is only refreshed when the mouse is moved.

While peeking, it is assumed that the *lens* window contains an accurate magnification of what was on the screen at the time of the last magnification. If *lens* is used to examine its own image, strange things may occur.

Due to the bitmap reshaping techniques employed by the magnification algorithms, high magnification factors will not work with large image windows. Precisely, if the product of the vertical magnification factor and the width of the destination rectangle overflows a signed short integer, predictable but undesirable results will occur.

NAME

movie, stills − algorithm animation

SYNOPSIS

`movie` [`-t` *termprog*] [`-m` *memory*] [*file*]

`stills` [*files ...*]

DESCRIPTION

Movie converts a 'movie script' into an internal representation, then displays it in a window on a Teletype 5620, AT&T630, or X-11 system (depending on which version has been compiled). If the filename is of the form *file*`.s`, *movie* creates the intermediate form in *file*`.i`, which will be used in subsequent calls if it is more recent than *file*`.s`, The options are:

`-t` *termprog* Load *termprog* instead of the default terminal program.
`-m`*mem* Use *mem* bytes of terminal memory instead of the default.

In the terminal, button 1 stops and starts the movie; button 2 adjusts view sizes and selects clicks; button 3 sets various parameters.

Movie scripts

A movie consists of multiple independent views, each presented as a rectangular sub-window. If no `view` statements appear, there is a single implicit view `def.view`. Any text or geometrical object may be labeled with a name and colon. Labels and coordinates are local to views. A recurring label erases the previous object with that label.

Comments follow #; blank lines are ignored.

`text` *options x y string*
: Text is centered and medium size by default; options: one of `center ljust rjust above below`, and one of `small medium big bigbig`. A leading quote is stripped from *string*, as is a trailing quote if a leading one is present.

`line` *options x1 y1 x2 y2*
: Lines are solid by default; options: one of `fat fatfat dotted dashed` and one of `-> <- <->`.

`box` *options xmin ymin xmax ymax*
: A box may be `fill`ed.

`circle` *options x1 y1 radius*
: Radius is measured in the *x* dimension. A circle may be `fill`ed.

`erase` *label*
: Erase an object explicitly.

`clear` Erase all objects currently in the current view.

`click` *optional-name*
: Place a mark in the intermediate with this name; clicks are used to control stepping in a movie or to define frames for a set of stills.

`view` *name*
: Associate subsequent objects with this view, until changed again.

Stills converts selected frames of a movie into commands for *pic*(1). Commands for *stills* begin with `.begin stills` and end with `.end stills`.

FILES

All files are in `/usr/lib/movie`.

`develop`	Shell script to control conversion from script language to internal form.
`fdevelop`	C program that does the work.
`stills.awk`	Awk program to process stills language into *pic*(1).
`anim`	Host end of the animation system.
`animterm`	terminal end.

`newer` Test whether one file is newer than another.

SEE ALSO

flicks(9.1), *pic*(1)

J. L. Bentley and B. W. Kernighan, ‘A System for Algorithm Animation’, this manual, Volume 2

BUGS

The 630 can only handle 65000 bytes of memory.

NAME

mux, ismux, invert − layer multiplexer for 5620

SYNOPSIS

`mux` [`-l` *command ...*]

`mux exit`

`mux cd` *directory*

`ismux` [-]

`invert`

DESCRIPTION

Mux manages asynchronous windows, or layers. Upon invocation, it loads the terminal with a program (default `/usr/jerq/lib/muxterm`, settable by the environment variable `MUXTERM`) that is the primary user interface. Option `-l` also creates a layer and invokes the shell to run *commands* in it. (See *windows*(9.1)).

The command `mux exit` leaves *mux,* destroying all layers; `mux cd` changes the directory of *mux*, and hence of layers later created, but not of the current layers.

Each layer is essentially a separate terminal. Characters typed into the layer are sent to the standard input of a Unix process bound to the layer, and characters written on the standard output of that process appear in the layer. When a layer is created, a separate shell (the value of the `SHELL` environment variable, or *sh* by default) is established, and bound to the layer.

Layers are created, deleted, and rearranged using the mouse. Depressing mouse button 3 activates a menu of layer operations. Releasing button 3 then selects an operation. At this point, a gunsight or box cursor indicates that an operation is pending. Hitting button 3 again activates the operation on the layer pointed to by the cursor.

The `New` operation, to create a layer, requires a rectangle to be swept out, across any diagonal, while button 3 is depressed. A box outline cursor indicates that a rectangle is to be created. The `Reshape` operation, to change the size and location of a layer on the screen, requires first that a layer be indicated (gunsight cursor) and a new rectangle be swept out (box cursor). The other operations are self-explanatory.

In a non-current layer, button 1 is a shorthand for `Top` and `Current`, which pulls a layer to the front of the screen and makes it the active layer for keyboard and mouse input. Th current layer is indicated by a heavy border.

There is a point in each layer, called the ‘Unix point’, where the next character from the host Unix system will be inserted. The Unix point advances whenever characters are received from the host, but not when echoing typed characters. When a newline is typed after the Unix point, characters between the Unix point and the newline, inclusive, are sent to the host and the Unix point advanced to after the newline. This means that shell prompts and other output will be inserted before characters that have been typed ahead. No other characters are sent to the host (but see the discussion of raw mode below). Therefore partially typed lines or text anywhere before the Unix point may be edited.

The default terminal program allows any text on the screen to be edited, much as in *sam*(9.1). Text may be selected by sweeping it with button 1 depressed. Typed characters replace selected text.

All layers share a common ‘snarf buffer’ (distinct from *sam*’s). The `cut` operation on button 2 deletes selected text and puts it in the buffer; `snarf` copies selected text to the buffer; `paste` replaces selected text (which may be null) from the buffer; and `send` copies the snarf buffer to after the Unix point.

Normally the terminal doesn’t scroll as text is received, but a button 2 menu item selects scrolling.

A scroll bar indicates what portion of all the text stored for a layer is on the screen. (It measures characters, not lines.) Releasing button 1 in the scroll bar brings the line at the top of the screen to the cursor; releasing button 3 takes the line at the cursor to the top of the screen. Button 2, treating the scroll

bar as a ruler, brings the indicated point in the whole stored text to the top of the screen. Slide the cursor off either end of the scroll bar with button 2 depressed to get right to an end of the file.

The NUM LOCK key advances a half page.

Ismux reports on its standard error whether its standard output is a *mux* layer, and also generates the appropriate exit status. With option −, no message is produced.

Invert reverses the sense of video, from black on white to white on black, or *vice versa*.

Independent user-level programs can be loaded into layers, see *32ld*(9.1). SHIFT-SETUP freezes *mux* and complements the video of the layer of the running user-level terminal process. Hitting button 2 in this state will attempt to kill the process; 1 or 3 will leave it running.

In raw mode or no-echo mode (see *ttyld*(4)) the Unix point advances with each character typed after it. In 8bit mode, characters with octal codes 0200 and greater print according to the ISO Latin1 alphabet; see *ascii*(6).

FILES

`/tmp/.mux*` temporary file used by `-l` option

SEE ALSO

32ld(9.1), *sam*(9.1), *jx*(9.1), *term*(9.1), *windows*(9.1)
R. Pike, 'Blit Download Protocols', this manual, Vol. 2

DIAGNOSTICS

Mux refuses to create a layer when there is not enough memory. Space can be recovered by deleting a layer.
Error messages from *mux* are written directly to the layer which caused them. They are usually meaningful only to system administrators, and indicate system difficulties.

BUGS

Reshape only works properly for processes that arrange to see if they have been reshaped, although most programs make this arrangement.
The behavior of raw mode prohibits editing partially typed lines when running *cu*(1).

NAME

paint − draw pictures in a layer

SYNOPSIS

`paint`

DESCRIPTION

Paint is a program for artistic interactive drawing. Buttons 1 and 2 draw in different ways, e.g. depositing and erasing paint. Button 3 gets a menu. Certain menu items contain arrows, which if touched call submenus. Moving off the right of a submenu causes it to disappear. Some items toggle a state on and off; a `*` appears in the abnormal state. Pressing button 1 while holding button 3 gets a short help message for the menu item. The top-level menu contains:

`Style`	→ Different kinds of brush strokes
`Operation`	→ Ways of putting paint on canvas
`Texture`	→ Things to do to the texture pattern
`Brush`	→ Things to do to the paintbrush
`Canvas`	→ Things to do to the whole picture
`State`	→ Change things saved in *.paintstate*
`Fill`	Fill an area of the picture
`Green`	Erase the entire picture
`Mask`	Display mask instead of image
`Exit`	

The `Style` submenu:

`Paint`	Multiple brush spots while holding button 1 or 2
`Circles`	Circles; press at center and release at circumference
`Lines`	Rubber-band brush lines
`Curves`	Continuous strokes while holding button 1 or 2
`Line Style`	→ Solid, dotted, dashed, etc. lines

Entries in the `Line Style` sub-submenu are strings of `A`s, `B`s and dot that describe dotted and dashed lines. `A` stands for the brush on the button pushed, `B` stands for the brush on the other button; `.` for no brush at all. The string is cycled through at successive points when drawing Lines, Curves, or Circles. Thus `A` means a solid line, `A...` means a 1 in 4 dotted line, and `AAAA....` means 4-pixel dashes.

The `Operation` submenu assigns a pair of operations for buttons 1 and 2. A hidden ‘mask’ plane describes the shape that has been painted; black pixels in the mask are inside, green outside. Likewise, the brush consists of a pair of rectangular image and mask planes. There are 11 effective operations to combine the part of the brush inside its mask with the part of the picture it sits on (see the Porter/Duff paper for details); selected pairs can be assigned to the buttons:

`Above/Erase`	Button 1 paints on top, Button 2 erases
`Below/Erase`	Button 1 paints behind, Button 2 erases
`Above/Below`	Button 1 paints on top, Button 2 behind
`Inside/Erase`	Button 1 paints inside, Button 2 erases
`Brush/Clear`	Special effects
`AoutB/AinB`	Special effects
`BinA/BatopA`	Special effects
`Xor/Above`	Special effects

`Above` paints on top of the picture, as in ‘normal’ paint programs.

`Below` paints underneath—only in places that were not previously covered.

`Inside` paints on top, but only inside the already-painted part.

The other 7 operations are best described as ‘special effects’. Try them out to see what they do, or look at the Porter/Duff paper.

Texture facilities paint with a repeating 16×16 pattern instead of copies of a brush. The `Texture` submenu contains:

`Texture`	Turn texturing on or off
`Make`	Pick a texture from the picture
`Negate`	Reverse the texture's green and black
`Save`	Name a texture and copy it into a file
`Library`	→ List and and retrieve textures in library
`Get`	Type a name and get a texture from a file

`Make` gives a 16×16 square cursor with which to pick a texture.

The `Brush` submenu has the same items for brushes. `Make` allows you to sweep out a region to use as a brush.

The `Canvas` submenu contains `Negate`, `Save`, `Library`, and `Get`, in this case pertaining to entire pictures. A library picture is saved in a file containing the image plane then the mask plane in *bitfile*(9.5) format.

The file `.paintstate` in the current directory remembers the names of the current brush, texture, and libraries between sessions. The `State` submenu displays the library names at the bottom of the layer, where they can be edited:

`Brushes`	Name the brush directory
`Pictures`	Name the picture directory
`Textures`	Name the texture directory

The `Fill` menu item gives an arrowhead cursor. If you touch down with button 3 at a point not painted, the rookwise-connected region containing it will fill with black. On completion, the black will be replaced by the current texture. While the region is filling, any button click aborts the operation.

The current selections from the `Brush` `Library`, `Style`, `Operation`, `Texture`, and `Line Style` menus are marked with a `*`, and are displayed in the information box at the bottom of the layer.

FILES

`/usr/jerq/lib/paint/brush`	the default brush library
`/usr/jerq/lib/paint/tex`	the default texture library
`.paintstate`	state of terminated program

SEE ALSO

mbits(6), *bitfile*(9.5), *brush*(9.1), *cip*(9.1), *ped*(9.1)
Thomas Porter and Tom Duff, 'Compositing Digital Images,' Siggraph '84 Proceedings

NAME

ped, tped − picture editor

SYNOPSIS

`ped` [`-f`] [*file* ...]

`tped` [*option* ...] [*file* ...]

DESCRIPTION

Ped is an interactive drawing program for 5620 terminals. A *file* argument is equivalent to an `e` command as described below. Most features of *ped* are menu-controlled and self-explanatory; further details are in the reference.

Button 1 selects actions from a permanent menu and to draw or pick up an object. Button 3 terminates drawing actions or changes the permanent menu. Button 2 causes the permanent menu to revert to `basic`.

The operation of *ped* is split between host and terminal. When a file is first read, it is kept on the host; `bring in` gets it to the 5620.

Option `-f` causes *ped* to display all text in one size to save time and space.

Some of the actions on permanent menus (switched by button 3) are described below. The last action is usually remembered and may be executed repeatedly until another is selected. Thus, for example, one can fill many polygons with one button click per polygon. Actions marked `(t)` in the menu toggle on and off.

`basic` menu

- `blitblt` Copy part of the screen to file `BLITBLT`, see *blitblt*(9.1).
- `exit` Leave *ped*, requires a confirming push of button 3.
- `markers` Make visible the defining points of objects; these are the only points sensitive to selection by button 1.

`type comm`

Take input from the keyboard.

- `e` *file* Begin editing *file*, remember its name, as in *ed*(1). Commands `f` (file name), `r` (read), `w` (write), behave similarly.
- `qq` Same as `exit` in `basic` menu; altered files will be saved in `ped.save`.
- `cd` Change working directory.
- `pwd` Print working directory.
- newline Reactivate mouse.
- `u` *string* Remember *string* as a shell command for the selection `user oper` in menu `refine`.

`ch size`

- `rotate` displays a vector from the center (of the bounding box) of an object to the selected point. The object is rotated and scaled to bring that point to a second selected position.
- `h-elong` Change aspect ratio. The inverse is `v-elong`.

`move` Button 3 cancels a move or copy. To help untangle overlapping objects, the cancellation does not take place until returning to the basic menu.

- `attach` Move an open polygon (a broken line) and hook it to the end of another.
- `join` Connect the ends of two polygons with a new line.
- `link` Cause multiple polygons to move and be filled as one (useful for making holes). Linked polygons must all be open or all be closed.
- `match` Move objects to bring selected points together.
- `center` Move objects to bring their centers together.
- `family/pt` Select objects to be moved or deleted together.

`draw` Button 1 fixes a point; button 3 terminates an object.

`text` Type one or more lines terminated by an empty line.

`grid` Snap points to locations on a grid, which indexes through settings FMC (fine, medium, coarse, none).

`fix sz` Set option `-f`.

`family/bx`
Sweep a box around objects to be moved or deleted together.

`reshape`

`formal` Adjust nearly rectangular lines to be perfectly so.

`spline` A piecewise parabolic fit tangent to the midpoints of a broken line.

`corner` Make a guiding point of a spline to be multiple − a corner in an otherwise smooth curve.

`refine`

`adj t` Left-justify, right-justify, or center text.

`edit text`
Display text at the top, where button 1 selects a position for inserting by typing or deleting by backspacing. Button 3 concludes the editing.

`shade` Assign textures for filling polygons, circles, or spline-bounded regions. Curves are filled schematically on the 5620, but accurately on the host.

`color` Assign colors for display on other devices.

`remote`
Perform all editing on the host using the terminal as a display device only.

Tped converts files of graphic information produced by *ped* into typesetting requests for *troff*(1). The options are:

`-T`*dev* Prepare output for particular devices known to *troff*: `-Taps` or `-T202`.

`-b` Place a box around each picture.

The input may be straight *ped* output or may be arbitrary text files with *ped* output embedded between pairs of delimiting lines:

`.GS` [*size*] ped file . . . `.GE`

or in another file:

`.GS` [*size*] *pedfilename*

The optional size gives width or height: `w=`*inches* or `h=`*inches.*

FILES

`.pederr`
`ped.save`

SEE ALSO

cip(9.1), *paint*(9.1), *brush*(9.1), *graphdraw*(9.1), *pic*(1), *ideal*(1), *blitblt*(9.1)
T. Pavlidis, ‘PED Users Manual’, this manual, Volume 2

DIAGNOSTICS

Error messages from the host are placed in file `.pederr`.

BUGS

Pictures may spill into the menu or message areas.
Some experimentation with *tped* printout parameters may be needed to obtain satisfactory results.

NAME

pi, 3pi − process inspector

SYNOPSIS

`pi` [`-t` *corefile objectfile*]

`3pi` [`-p` *person*]

DESCRIPTION

Pi is a C debugger that is bound dynamically to multiple subject processes or core dumps. It works better for programs compiled *cc −g*. *Pi* uses the *Pads*(9.5) multi-window user interface. There are three types of windows: debugger control windows, which access the global state of the debugger; process control windows (exactly one per process), which start and stop processes and connect to process-specific functions; and process inspection windows, which include viewers for source text and memory, formatted various ways.

The most important debugger control window is the *pi* window itself. Each line within the *pi* window refers to a specific process. These lines may be introduced to the window by running *ps*(1) from the button 3 menu; by typing a file name, either a *proc*(4) name, or the name of a core image followed by the name of the binary that created the core; or by typing a command, prefixed by an exclamation `!`, to be executed as a child of *pi*. There are several ways to access a process (using the button 2 menu), each of which generates a process control window:

`open process`	Attach to a running process, often one started with *hang*(1).
`open core`	Attach to a core image.
`open child`	Attach to a process forked by a process being debugged by the current *pi.*
`take over`	Rebind an existing process window hierarchy (pointed to with the mouse) to the named process, which must be an instance of the identical program.
`hang & open proc`	Execute a command afresh, beginning it in the stopped state, and redirecting IO to `/dev/null`.
`hang & take over`	Same, also binding to an existing process window.

The process window indicates the process's state, shows the call stack traceback and connects to windows that access source text, local variables within a stack frame, raw memory, and so on. These windows are cross-connected, so, for example, an instruction in a process's assembly language window can be inspected in hexadecimal in the raw memory window. Closing the process control window closes all the windows under it.

The following menu functions are provided by the various window types in *pi*. Initially there are these windows available:

`Help` Reminder of user interface mechanics.

`Pi` Overall control of processes, core dumps and programs. A process is identified by its pathname or command line. Process symbols are found in the executable file from which the process was loaded, but may be overridden. Symbols for core dumps must be supplied explicitly, after the core filename. **Synopsis**: Identify and open process or core dump; run a program as *Pi's* child; take over a process with the debugging environment of a different one.

`Pwd/cd`
change the working directory of the debugger.

Process Window

Selecting and opening a process from the Pi window creates a new window with overall control of that process. It shows the process state, and a traceback if the process is halted or dead. States are:

`ACTIVE`	running normally
`HALTED`	halted asynchronously by a debugger
`BREAKPOINT`	halted on reaching breakpoint
`STMT STEPPED`	halted after executing C source statement(s)

`INSTR STEPPED`	halted after executing machine instruction(s)
`EVENT PENDING`	halted about to receive a signal being traced
`ERROR STATE`	the process has probably exited

The menu operations on the process are:

`go`	let the process run
`stop`	stop the process
`kill`	send `SIGKILL` to the process; see *signal*(2)
`src text`	open source text window(s)
`Signals`	open window for sending and trapping signals
`Globals`	open window for evaluating expression in global scope
`RawMemory`	open window for editing uninterpreted memory
`Assembler`	open window for disassembler

Each line of the call stack traceback describes one function. Each function in the traceback can open an expression evaluator window or display its current source line.

Globals and Stack Frame Windows

These windows evaluate expressions with respect to global scope, and scope in a function, respectively. A stack frame window is opened from a line in the call stack traceback or from a line of source text. A stack frame can find its active source line in a source window or the stack frame window of its caller.

C expressions can be entered by the keyboard or mouse. The unary operators *fabs* and *sizeof* are supported; the only assignment operator is =. Functions from the user program may be called. New expressions can be derived from old ones by the keyboard or mouse. In menus and the keyboard, `$` means the expression in the current line. The VAX registers are called `$r0` to `$r15`; the address of a register is the location at which it was saved. The format in which values are displayed can be changed. The raw memory editor may be entered using an expression's value as address.

An expression may be made a *spy*. The value of a spy expression is evaluated and displayed each time the debugger looks at the process. If the value of a spy changes the process is halted at the next instruction, statement or breakpoint.

The comma operator is useful in conditional breakpoints because the values of its subexpressions are displayed. E.g. x, y, x==y traces the values of x and y when the condition fails; x, y, 0 just traces.

To cross scope boundaries, the environment (a function identifier) in which an expression is to be evaluated may be specified as: { expr } function. From the source directory window, file static variables can be promoted to appear in the menu of global variables.

Source Text Windows

The source file directory window lists all the source files, if there are two or more. A textual prefix, entered from the keyboard, points to a source directory, without changing the working directory. Each source file is in a separate window, opened when needed. The source file's pathname as given to *cc* can be overridden from the keyboard. If things go wrong, use `reopen` to open the file afresh. **Synopsis**: set/clear (conditional) breakpoint; single-step source statements; step into (rather than over) a function; go the process; show the statement for the current PC; open a stack frame window for a source line's function; evaluate expression; disassemble first instruction of source statement; context search for string.

Breakpoints Window

Lists all the active source and assembler breakpoints and related errors. **Synopsis**: show source or assembler for a breakpoint; clear breakpoint; clear all breakpoints.

Signals Window

Lists all signal types, showing which ones are traced. **Synopsis**: Change which signals are traced; send a signal to the subject process; clear pending signal; stop process on *exec.*

Raw Memory Window

In this window memory is a viewed as a sequence of 1-, 2-, 4- or 8-byte cells. **Synopsis**: set cell address; change cell size; change display format; display cells above and below; indirect to cell; change

cell value; spy on memory cell; disassemble instruction at cell.

(Dis)assembler Window

In this window memory is viewed as a sequence of instructions. **Synopsis**: set instruction address; display more instructions; change display format; display instruction as cell in raw memory window; set/clear breakpoint on instruction; open stack frame window for instruction's function; display instruction at current PC; single step instruction(s); step over a function call instead of into the function.

Exec and Fork

If a process controlled by *pi* does an *exec*() and an exec break is set in the Signals window, the process is suspended as if started by *hang*(1). To debug the process after the *exec*, close the original process window and re-open it. When re-opened it will get the new symbol tables.

To debug a child process: (i) set a breakpoint in code that will be executed in the child after the fork; (ii) execute the fork *at full speed* (the child inherits the parent's breakpoints, which aren't there if the parent is stepped); (iii) *before altering any breakpoints*, get a fresh *ps* in the Pi window and apply `open child` to the child. The child should be stopped on the inherited breakpoint, but it and all other breakpoints should have been cleared.

Kernel

The state of kernel variables associated with a process may be examined by giving their name or virtual address. The `UNIX` environment variable specifies the file from which the system was loaded; the default is */unix*. Kernel dumps may be examined by opening the 'kernel pi' window.

Just A Traceback

With the `-t` option *pi* writes a traceback on its standard output and quits.

3pi

3pi is a variant of *pi* for debugging 5620 programs running under *mux*(9.1). It creates two terminal processes: one for its access to terminal memory and graphics and a second for its *Pads*(9.5) interface.

Remote Debugging

With the `-p` option *3pi* loads its first process, but not *Pads*. Instead, it mails a *3pi* command to *person*, to be executed on any host in the local network. That *3pi* command loads *Pads* on *person*'s terminal, and connects to the originator's terminal. If separate hosts are involved and the versions of critical files differ, be careful with pathnames.

3pi Graphics

Points, rectangles, textures and bitmaps can be displayed graphically.

3pi - pi

Most differences come from obvious differences in the hardware and software architectures. Also, in *3pi* function calls are executed by a debugger process on its own call stack.

SEE ALSO

T. A. Cargill, 'The Feel of Pi', this manual, Volume 2
hang(1), *proc*(4), *adb*(1), *cin*(1), *nm*(1), *pads*(9.5)

BUGS

In switch statements there is no boundary between the last case and the branch code; the program *appears* to jump to the last case (but is really in the branch) and then to the real case.

A changed spy only stops the process at a breakpoint or while stepping. An expression can be cast only by menu.

Functions may only be called when the process is stopped and not in a system call.

NAME

proof − troff output interpreter for 5620

SYNOPSIS

`proof` [`-f`*fonts*] [*file*]

DESCRIPTION

Proof reads *troff*(1) intermediate language from *file* or standard input and simulates the resulting pages on the screen. If no file name is given and standard input is a terminal, proof terminates immediately leaving a 'proof layer'. By invoking *proof* in a proof layer you can avoid download time.

Fonts are loaded as required. The usual *mux*(9.1) font, `defont`, is used for unknown fonts. Option `-f` preloads fonts. Names are given relative to `/usr/jerq/font` and are separated by commas. The most-used fonts are `-fR.10,I.10,B.10,S.10`.

After a layer's worth of text is displayed, *proof* pauses for a command from keyboard or mouse button 3. The typed versions of commands are:

newline Go on to next portion of text. (Button 3 equivalent: `more`.)

`q` Quit, leaving a proof layer.

`x` Exit and restart the regular terminal program. (Equivalent to `q` followed by `term mux`; see *term*(9.1)).

`p`*n* Print page *n*. An out-of-bounds page number means the end nearer to that number; a missing number means page 0; a signed number means an offset to the current page.

Button 1 gets a scroll box, which represents a full page of text. An interior rectangle shows what part of the page is now visible. The interior rectangle moves with the mouse, causing the layer to scroll both vertically and horizontally. Button 2 gets a speedometer. The bar of the speedometer moves with the mouse to control the rate at which new information is displayed.

EXAMPLES

`troff -ms memo | proof`
Format a memo and display it.

`(eqn memo | troff -ms) 2>diags | proof`
Display a memo with equations. Avoid sending diagnostics to the screen; see BUGS.

FILES

`/usr/jerq/font/*`	fonts
`/usr/jerq/font/.missing`	list of referenced but unconverted fonts

SEE ALSO

lp(1), *font*(6), *reader*(9.7), *psi*(9.1)
Brian W. Kernighan, *A Typesetter-independent Troff*

BUGS

Proof breaks if other messages are directed to its layer. In particular, unredirected *troff* diagnostics will break the pipeline `troff | proof`.
Windowing can get confused if the *troff* output is not approximately sorted in ascending *y*-order.
A proof layer imitates `term 33`, not *mux*. Among other difficulties, it will not be reusable if downloaded across the network.

NAME

psi − postscript interpreter

SYNOPSIS

`psi` [*option* ...] [*file*]

DESCRIPTION

Psi reads Postscript input from *file* or from standard input and simulates the resulting pages in a *mux*(9.1) layer. The program remains in the layer at exit; further invocations of *psi* in that layer avoid download time.

The options are

`−p`*n* Display page *n*, where *n* is determined from the `%%Page` comments in the file. If these are not present, page selection will not work.

`−R` Pages in the file are in reverse order. This flag must be used on such files for the *-p* option to work.

`−r` Display the image at full scale, with the bottom left corner positioned at the bottom left corner of the window. (By default, the image is scaled to fit the window, maintaining the aspect ratio of a printer.)

`−a` *x y*
Display the image at full scale with position *x*,*y* of the image placed at the bottom left corner of the window.

Psi works on either a Teletype 5620, 630 or 730 terminal as determined by the environment variable `TERM`.

Fonts are implemented with size-24 bitmap fonts. Those available are Symbol, Courier, Times-Roman, Times-Italic, Times-Bold, Times-BoldItalic, Helvetica, Helvetica-Oblique, Helvetica-Bold, Helvetica-BoldOblique. Fonts Courier-Bold, Courier-Oblique, and Courier-BoldOblique are mapped to Courier. Other postscript fonts, including type1, may be used if they are supplied before they're referenced.

When the 'cherries' icon is displayed, use mouse button 3 to move forward (`more`), to a particular page (`page`), or quit (`done`). Button 2 exits the program completely.

EXAMPLES

```
troff −ms memo | lp −dstdout −H | psi
troff −ms memo | dpost | psi
```
Two equivalent ways to format a memo, convert it to PostScript, and display it.

For best results with TeX documents, use `dvips` with the `−Tjerq`, `−Tgnot`, or `−D 100` option to get fonts of the proper resolution and run *psi* with the *-r* or *-a* flag to prevent *psi* from scaling.

FILES

`psi.err` error messages

SEE ALSO

lp(1), *dvips*(1), *postscript*(8), *proof*(9.1), *psifile*(1), *psix*(1)

DIAGNOSTICS

A 'dead mouse' icon signals an error; error comments are placed on file `psi.err`.

Symbols that lack bitmaps are replaced by '?' and an error is reported.

BUGS

A psi layer imitates `term 33`, not *mux*. Among other difficulties, it will not be reusable if downloaded across the network.
Unimplemented PostScript features are rotated images and half tone screens. Imagemasks may only be rotated by multiples of 90 degrees, not by arbitrary angles.
Skipping pages may cause operators to be undefined.

NAME

rebecca − graphics touch-up editor

SYNOPSIS

`rebecca` *file*

DESCRIPTION

Rebecca is an interactive retouching tool for digitized grey-scale images. The *file* must be a headerless 512×512 black-and-white digitized image. Example (read only) files are in directory `/n/kwee/t0/face/512x512x8`.

'Floating instruments' for editing can be dragged with button 2 to different locations.

Resolution. The tick mark on the long bar can be moved up or down with button 1. Printed to the right of the bar is the current resolution— a power of 2 representing the number of file pixels across the screen image.

Grid. Click button 1 at the circular button to toggle the grid. Turning on the grid is useful sometimes to see how fast a screen update is proceeding: it eats away the grid.

Write. Write the file on the host by clicking button 1 at the box labeled `write`. The write box has a `*` if a change was made to the file since it was last written.

Runlength encoding. Clicking button 1 at this box toggles the mode of data transmission between host and terminal.

Reopen. This instrument cancels any changes made to the file since the last time it was written.

Move/Pan. Click button 1 at one of the 5 areas of the diamond. The middle resets the display to a full size picture. Left, right, up, or down will move (pan) 1/4 screen in the corresponding direction (useful only on zoomed pictures).

Zoom/Unzoom. Click button 1 at `Z` (zoom) or `U` (unzoom). `Z` prompts with a square box to be positioned on the area of the picture to be inspected at full resolution. If you click button 1 before you confirm, the sides of the box are halved. Clicking button 2 doubles them. Any combination of two buttons cancels the zoom; button 3 confirms it.

Paint. Click button 1 at the box labeled + = −. Painting with + adds grey values to pixels; = assigns values; − subtracts values. Click button 1 at a pixel location to apply the paint. Click button 3 to sweep a rectangle to paint all pixels within it. Pick a paint value (default is white) by clicking button 2 at the grey scale at the bottom or at any pixel in the image. Click button 2 at the paint box to cancel the paint mode.

Smear. Pointing at a pixel with both buttons 1 and 2 down averages it with its 8 neighbors (most useful when zoomed in to pixel level). Typical usage: apply some white or black paint with the paint box, then smear it.

Probe. Click button 1 at the probe box `P:`. Point at a pixel in the image. The *x*-*y* coordinates and the greyscale value of the pixel will be printed.

Contrast. Move the ends of the line under the grey scale bar to expand or compress the grey scale.

Rubber Sheet. The box named `sheet` prompts for a rectangle. Sweep out the rectangle over an area you want to manipulate, then reposition the corners by dragging them to new locations with button 1. Confirm the selection with button 3. Other instruments are usable while the update proceeds.

SEE ALSO

pico(1), *flicks*(9.1), *picfile*(5), *flickfile*(9.5)

NAME

ruler − measure things on the screen

SYNOPSIS

`ruler`

DESCRIPTION

Ruler measures things on a *mux*(9.1) screen. Press button 1 to sweep out a rectangle anywhere on the screen. For each rectangle swept, *ruler* displays the coordinates of the rectangle's corners (labeled `down` and `up`), the size of the rectangle and length of its diagonal.

There is a menu on button 3. The `pixels` and `chars` items control whether the size and diagonal are measured in units of pixels or characters; `stop` deactivates *ruler* without exiting; `measure` reactivates *ruler*.

BUGS

Character units are arbitrarily defined as the width and height of a `0` in the *ruler* layer. This may have nothing to do with character sizes in other layers.

Ruler's menu must pop up in its own layer, perhaps far away from the cursor.

NAME

sam − screen editor with structural regular expressions

SYNOPSIS

`sam` [*option* ...] [*files*]

sam `-r` *machine*

`sam.save`

DESCRIPTION

Sam is a multi-file editor. It modifies a local copy of a Unix file. The copy is here called a *file*; a Unix file is distinguished by the trademarked adjective. The files are listed in a menu available through mouse button 3 or the `n` command. Each file has an associated name, usually the name of the Unix file from which it was read, and a 'modified' bit that indicates whether the editor's file agrees with the Unix file. The Unix file is not read into the editor's file until it first becomes the current file—that to which editing commands apply—whereupon its menu entry is printed. The options are

`-d` Do not download the terminal part of *sam*. Editing will be done with the command language only, as in *ed*(1).

`-r` *machine*
Run the host part remotely on the specified machine, the terminal part locally. This extends graphic editing to files on machines that don't ordinarily support it or across non-*nfs*(8) connections.

Regular expressions

Regular expressions are as in *egrep* (see *gre*(1)), with the addition of `@` and `\n`. A regular expression may never contain a literal newline character. The elements of regular expressions are:

`.` Match any character except newline.

`\n` Match newline.

`\x` For any character except `n` match the character (here `x`).

`@` Match any character.

`[abc]` Match any character in the square brackets. `\n` may be mentioned.

`[^abc]`
Match any character not in the square brackets, but never a newline. Both these forms accept a range of ASCII characters indicated by a dash, as in `a-z`.

`^` Match the null string immediately after a newline.

`$` Match the null string immediately before a newline.

Any other character except newline matches itself.

In the following, *r1* and *r2* are regular expressions.

`(`*r1*`)` Match what *r1* matches.

r1`|`*r2* Match what *r1* or what *r2* matches.

r1`*` Match zero or more adjacent matches of *r1*.

r1`+` Match one or more adjacent matches of *r1*.

r1`?` Match zero or one matches of *r1*.

The operators `*`, `+` and `?` are highest precedence, then catenation, then `|` is lowest. The empty regular expression stands for the last complete expression encountered. A regular expression in *sam* matches the longest leftmost substring formally matched by the expression. Searching in the reverse direction is equivalent to searching backwards with the catenation operations reversed in the expression.

Addresses

An address identifies a substring in a file. In the following, 'character *n*' means the null string after the *n*-th character in the file, with 1 the first character in the file. 'Line *n*' means the *n*-th match, starting at the beginning of the file, of the regular expression `.*\n?` . (The peculiar properties of a last line without a newline are temporarily undefined.) All files always have a current substring, called dot, that is the default address.

Simple Addresses

`#`*n* The empty string after character *n*; `#0` is the beginning of the file.

n Line *n*.

`/`*regexp*`/`
`?`*regexp*`?`
The substring that matches the regular expression, found by looking toward the end (`/`) or beginning (`?`) of the file, and if necessary continuing the search from the other end to the starting point of the search. The matched substring may straddle the starting point.

`0` The string before the first full line. This is not necessarily the null string; see + and − below.

`$` The null string at the end of the file.

`.` Dot.

`'` The mark in the file (see the `k` command below).

`"`*regexp*`"`
Preceding a simple address (default `.`), refers to the address evaluated in the unique file whose menu line matches the regular expression.

Compound Addresses

In the following, *a1* and *a2* are addresses.

a1+*a2* The address *a2* evaluated starting at the end of *a1*.

a1−*a2* The address *a2* evaluated looking in the reverse direction starting at the beginning of *a1*.

a1`,`*a2* The substring from the beginning of *a1* to the end of *a2*. If *a1* is missing, `0` is substituted If *a2* is missing, `$` is substituted.

a1`;`*a2* Like *a1*`,`*a2*, but with *a2* evaluated at the end of, and dot set to, *a1*.

The operators + and − are high precedence, while `,` and `;` are low precedence.

In both + and − forms, if *a2* is a line or character address with a missing number, the number defaults to 1. If *a1* is missing, `.` is substituted. If both *a1* and *a2* are present and distinguishable, + may be elided. *a2* may be a regular expression; if it is delimited by `?`'s, the effect of the + or − is reversed.

It is an error for a compound address to represent a malformed substring. Some useful idioms: *a1*+− (*a1*−+) selects the line containing the end (beginning) of a1. `0/`*regexp*`/` locates the first match of the expression in the file. (The form `0;//` sets dot unnecessarily.) `./`*regexp*`///` finds the second following occurrence of the expression, and `.,/`*regexp*`/` extends dot.

Commands

In the following, text demarcated by slashes represents text delimited by any printable ASCII character except alphanumerics. Any number of trailing delimiters may be elided, with multiple elisions then representing null strings, but the first delimiter must always be present. In any delimited text, newline may not appear literally; `\n` may be typed for newline; and `\/` quotes the delimiter, here `/`. Backslash is otherwise interpreted literally, except in `s` commands.

Most commands may be prefixed by an address to indicate their range of operation. Those that may not are marked with a * below. If a command takes an address and none is supplied, dot is used. The sole exception is the `w` command, which defaults to `0,$`. In the description, 'range' is used to represent whatever address is supplied. Many commands set the value of dot as a side effect. If so, it is always

set to the 'result' of the change: the empty string for a deletion, the new text for an insertion, etc. (but see the `s` and `e` commands).

Text commands

`a` / *text* /
or
`a`
lines of text
`.` Insert the text into the file after the range. Set dot.

`c`
`i` Same as `a`, but `c` replaces the text, while `i` inserts *before* the range.

`d` Delete the text in the range. Set dot.

`s` / *regexp* / *text* /
Substitute *text* for the first match to the regular expression in the range. Set dot to the modified range. In *text* the character `&` stands for the string that matched the expression. Backslash behaves as usual unless followed by a digit: *d* stands for the string that matched the subexpression begun by the *d*-th left parenthesis. If *s* is followed immediately by a number *n*, as in `s2/x/y/`, the *n*-th match in the range is substituted. If the command is followed by a `g`, as in `s/x/y/g`, all matches in the range are substituted.

`m` *a1*
`t` *a1* Move the range to after *a1* (`m`), or copy it (`t`). Set dot.

Display commands

`p` Print the text in the range. Set dot.

`=` Print the line address and character address of the range.

`=#` Print just the character address of the range.

File commands

In these commands a *file-list* may be expressed <*Unix-command* in which case the file names are taken as words (in the shell sense) generated by the Unix command.

* `b` *file-list*
Set the current file to the first file named in the list that *sam* also has in its menu.

* `B` *file-list*
Same as `b`, except that file names not in the menu are entered there, and all file names in the list are examined.

* `n` Print a menu of files. The format is:
`'` or blank
indicating the file is modified or clean,
`-` or `+` indicating the the file is unread or has been read (in the terminal, `*` means more than one window is open),
`.` or blank
indicating the current file,
a blank,
and the file name.

* `D` *file-list*
Delete the named files from the menu. If no files are named, the current file is deleted. It is an error to `D` a modified file, but a subsequent `D` will delete such a file.

I/O Commands

* `e` *filename*
Replace the file by the contents of the named Unix file. Set dot to the beginning of the file.

`r` *filename*
Replace the text in the range by the contents of the named Unix file. Set dot.

`w` *filename*
Write the range (default `0,$`) to the named Unix file.

* `f` *filename*
Set the file name and print the resulting menu entry.

If the file name is absent from any of these, the current file name is used. `e` always sets the file name, `r` and `w` do so if the file has no name.

`<` *Unix-command*
: Replace the range by the standard output of the Unix command.

`>` *Unix-command*
: Sends the range to the standard input of the Unix command.

`|` *Unix-command*
: Send the range to the standard input, and replace it by the standard output, of the Unix command.

* `!` *Unix-command*
: Run the Unix command.

* `cd` *directory*
: Change working directory. If no directory is specified, `$HOME` is used.

In any of `<`, `>`, `|` or `!`, if the *Unix command* is omitted the last *Unix command* (of any type) is substituted. If *sam* is downloaded, `!` sets standard input to `/dev/null`, and otherwise unassigned output (`stdout` for `!` and `>`, `stderr` for all) is placed in `$HOME/sam.err` and the first few lines are printed.

Loops and Conditionals

`x/`*regexp*`/` *command*
: For each match of the regular expression in the range, run the command with dot set to the match. Set dot to the last match. If the regular expression and its slashes are omitted, `/.*\n/` is assumed. Null string matches potentially occur before every character of the range and at the end of the range.

`y/`*regexp*`/` *command*
: Like `x`, but run the command for each substring that lies before, between, or after the matches that would be generated by `x`. There is no default behavior. Null substrings potentially occur before every character in the range.

* `X/`*regexp*`/` *command*
: For each file whose menu entry matches the regular expression, run the command. If the expression is omitted, the command is run in every file.

* `Y/`*regexp*`/` *command*
: Same as `X`, but for files that do not match the regular expression, and the expression is required.

`g/`*regexp*`/` *command*

`v/`*regexp*`/` *command*
: If the range contains (`g`) or does not contain (`v`) a match for the expression, set dot to the range and run the command.

These may be nested arbitrarily deeply, but only one instance of either `X` or `Y` may appear in a single command. An empty command in an `x` or `y` defaults to `p`; an empty command in `X` or `Y` defaults to `f`. `g` and `v` do not have defaults.

Miscellany

`k`
: Set the current file's mark to the range. Does not set dot.

* `q`
: Quit. It is an error to quit with modified files, but a second `q` will succeed.

* `u` *n*
: Undo the last *n* (default 1) top-level commands that changed the contents or name of the current file, and any other file whose most recent change was simultaneous with the current file's change. Successive `u`'s move further back in time. The only commands for which `u` is ineffective are `cd`, `u`, `q`, `w` and `D`.

(empty)
: If the range is explicit, set dot to the range. If *sam* is downloaded, the resulting dot is selected on the screen; otherwise it is printed. If no address is specified (the command is a newline) dot is extended in either direction to line boundaries and printed. If dot is thereby unchanged, it is set to `.+1` and printed.

Grouping and multiple changes

Commands may be grouped by enclosing them in braces `{}`. Commands within the braces must appear on separate lines (no backslashes are required between commands). Semantically, an opening brace is

like a command: it takes an (optional) address and sets dot for each sub-command. Commands within the braces are executed sequentially, but changes made by one command are not visible to other commands (see the next section of this manual). Braces may be nested arbitrarily.

When a command makes a number of changes to a file, as in `x/re/c/text/`, the addresses of all changes to the file are computed in the original file. If the changes are in sequence, they are applied to the file. Successive insertions at the same address are catenated into a single insertion composed of the several insertions in the order applied.

The terminal

What follows refers to behavior of *sam* when downloaded, that is, when operating as a display editor on a bitmap display. This is the default behavior; invoking *sam* with the `-d` (no download) option provides access to the command language only.

Each file may have zero or more windows open. Each window is equivalent and is updated simultaneously with changes in other windows on the same file. Each window has an independent value of dot, indicated by a highlighted substring on the display. Dot may be in a region not within the window. There is usually a 'current window', marked with a dark border, to which typed text and editing commands apply. Text may be typed and edited as in *mux*(9.1); also the escape key (ESC) selects (sets dot to) text typed since the last mouse button hit.

The button 3 menu controls window operations. The top of the menu provides the following operators, each of which prompts with one or more *mux*-like cursors to prompt for selection of a window or sweeping of a rectangle. 'Sweeping' a null rectangle gets a large window, disjoint from the command window or the whole screen, depending on where the null rectangle is.

`new` Create a new, empty file.
`xerox` Create a copy of an existing window.
`reshape` As in *mux.*
`close` Delete the window. In the last window of a file, `close` is equivalent to a `D` for the file.
`write` Equivalent to a `w` for the file.

Below these operators is a list of available files, starting with `~~sam~~`, the command window. Selecting a file from the list makes the most recently used window on that file current, unless it is already current, in which case selections cycle through the open windows. If no windows are open on the file, the user is prompted to open one. Files other than `~~sam~~` are marked with one of the characters `-+*` according as zero, one, or more windows are open on the file. A further mark `.` appears on the file in the current window and a single quote, `'`, on a file modified since last write.

Nothing can be done without a command window, for which *sam* prompts initially. The command window is an ordinary window except that text typed to it is interpreted as commands for the editor rather than passive text, and text printed by editor commands appears in it. The behavior is like *mux,* with a 'command point' that separates commands being typed from previous output. Commands typed in the command window apply to the current open file—the file in the most recently current window.

Manipulating text

Button 1 changes selection, much like *mux.* Pointing to a non-current window with button 1 makes it current; within the current window, button 1 selects text, thus setting dot. Double-clicking selects text to the boundaries of words, lines, quoted strings or bracketed strings, depending on the text at the click.

Button 2 provides a menu of editing commands:

`cut` Delete dot and save the deleted text in the snarf buffer.
`paste` Replace the text in dot by the contents of the snarf buffer.
`snarf` Save the text in dot in the snarf buffer.
`look` Search forward for the next occurrence of the literal text in dot. If dot is the null string, the text in the snarf buffer is used. The snarf buffer is unaffected.
`<mux>` Exchange snarf buffers with *mux.*
/ regexp
Search forward for the next match of the last regular expression typed in a command. (Not in

command window.)

`send` Send the text in dot, or the snarf buffer if dot is the null string, as if it were typed to the command window. Saves the sent text in the snarf buffer. (Command window only.)

`scroll`
`noscroll`
Select whether to reveal automatically text that appears off the end of the command window. (Command window only.)

Abnormal termination

If *sam* terminates other than by a `q` command (by hangup, deleting its layer, etc.), modified files are saved in an executable file, `$HOME/sam.save`. This program, when executed, asks whether to write each file back to a Unix file. The answer `y` causes writing; anything else skips the file.

FILES

`$HOME/sam.save`
`$HOME/sam.err`

SEE ALSO

ed(1), *sed*(1), *vi*(1), *gre*(1)

BUGS

The `u` command undoes characters—and backspaces—typed directly into a file window in unpredictable increments.

NAME

term − terminal emulators for mux

SYNOPSIS

`exec term` *termtype*

DESCRIPTION

Term replaces the program in the layer on its standard output with an emulator for the terminal type specified by *termtype*. In the resulting layer, environment variable TERM is set appropriately. Known types are

`2621` Hewlett-Packard 2621

`2621c` Same, with data compression between host and terminal; useful at line speeds of 2400 baud and lower.

`4014` Tektronix 4014

`5620` Teletype DMD 5620 stand-alone terminal.

`5620c` Same, with data compression.

`33` Teletype Model 33 (actually closer to 35).

Also, *termtype* `mux` restores and initializes a standard *mux*(9.1) terminal program.

BUGS

Nonstandard terminal emulators do not work across *dcon*, but usually do across *ndcon* connections; see *dcon*(1).

Unexported shell parameters and functions are lost.

NAME

thinkblt, think − print on thinkjet

SYNOPSIS

`thinkblt` [*stream*]

`think` [`-o` *stream*] [*file* ...]

`nroff -Tthink` ... | `think` [`-o` *stream*]

DESCRIPTION

Thinkblt downloads an interrupt driver for the HP ThinkJet printer, provides a menu of operations for printing various data residing in the terminal, and sets up a *stream* (`$HOME/.THINK` by default) on which *think* can print data from the host. It is intended to be down-loaded once per terminal session. Most of the menu items are identical to those of *blitblt*(9.1). The remaining ones are:

`print bitmap` Print whatever bitmap is currently selected, in analogy to *blitblt*(9.1). The widest printable bitmap is 640 pixels across.

`print mux buffer` Print the *mux* 'snarf' buffer.

`reset printer` Sends ESC-`E`.

While the printer is operating, a different menu allows one to abort or pause the print operation. The printer has a fairly large internal buffer, so response may be slow.

Files on the host may be printed by giving them as arguments or standard input to *think*. When used with *nroff*, names like `\('e` may be used to access the special characters provided by the hardware; the *nroff* terminal driving file has a complete list; see *troff*(1). Both *nroff* and *pr*(1) will paginate properly if top-of-form is set so that the paper tear is aligned flush with the top of the metal clip which holds the absorber.

FILES

`$HOME/.THINK`
`/usr/lib/term/tab.think` nroff descriptor file

SEE ALSO

troff(1), *pr*(1), *blitblt*(9.1)

BUGS

The 5620 ROM program is unable to cope with interrupts from the printer; it is therefore necessary to download *mux*(9.1) before turning on the printer.
Thinkblt substitutes its own interrupt routine for the (trivial) one provided by *mux*(9.1). The latter is restored upon exit, but havoc may result if the *thinkblt* layer is simply deleted.
The special *nroff* character names are not currently supported by any other device.

NAME

vismon, sysmon, vwhois − system statistics and mail notification

SYNOPSIS

`vismon` [−*n*] [`−m`] [*system ...*]

`sysmon` [−*n*] [`−m`] [*system ...*]

`vwhois` *person*

DESCRIPTION

Vismon monitors use of one or more Unix *systems*. It displays time of day, announcements, and CPU usage statistics.

CPU usage is reported as a numerical load average (average number of runnable processes) and its change in the last minute, and a bar graph showing, left-to-right, the proportion of CPU time spent in: default-priority user processes, low priority (nice) processes, system kernel, stream I/O, and idle time.

Arrival of mail or communications via *wall*(8) or *write*(1) is announced. Mail announcements include an icon of the sender. Communications appear in a shell (*sh*(1)) layer superimposed on *vismon's* layer. This layer may be used for reply.

The options are:

−*n* Update the bar graph every *n* seconds. (*n*=5 by default.)

`−m` Do not monitor CPU usage on other systems.

Button 2 selectively toggles the monitoring of other systems. The list of systems is obtained from one of the following: a file named in the VISMON enviroment variable, `$HOME/lib/vismon`, or `/usr/jerq/lib/vismon`.

Sysmon is the same as *vismon* without icons.

Vwhois causes a dummy mail announcement from *person* to appear in *vismon* layers.

FILES

`/usr/jerq/mbin/sysmon.m`	terminal program
`/usr/jerq/lib/sysdaemon`	remote monitoring program
`/usr/jerq/lib/sysdaemon`	responder for remote monitoring
`/usr/spool/mail`	
`/usr/spool/mail/mail.log`	
`/n/face/*`	vismon pictures
`/usr/jerq/lib/vismon`	
`$HOME/lib/vismon`	menu of machines

SEE ALSO

face(9.7), *faced*(9.5)

DIAGNOSTICS

'Can't open comm window' means a shell layer cannot be created. To receive any further communications, delete some layer.

BUGS

There's more to system performance than meets the eye.

NAME

windows, jps, reshape − create and initialize windows

SYNOPSIS

`windows` [*ox oy cx cy command ...*]

`jps`

`reshape` [`−r`] *x y*

DESCRIPTION

For each set of arguments, *windows* makes a *mux*(9.1) layer with rectangle Rect(*ox, oy, cx, cy*) (see *types*(9.5)), then executes the *command* therein. The *command* may be null (""). Any number of layers may be specified; each *command* and its arguments must be given as a single argument to *windows.*

In windows that are not expected to be reused and do not need a shell, it is good practice to invoke the *command* with `exec`; see *sh*(1).

Jps prints the rectangle coordinates of each window and the arguments (if any) with which it was down-loaded, to help set up the *windows* command.

Reshape adjusts its layer so that the display rectangle inside the border is *x* by *y* pixels. Under option `−r` it adjusts the width/height ratio to *x/y*, with the new shape as large as will fit inside the old.

SEE ALSO

mux(9.1), *ruler*(9.1)

DIAGNOSTICS

Windows may adjust rectangles to a minimum size or to stay within the usual layer bounds (8 pixels inside the screen edge).
Layer creation can fail if there are no process slots or memory left in the terminal.
Reshape clips a layer that is too big and does nothing if the layer is too small or if there is not enough memory.

BUGS

Jps reports what has been downloaded to the 5620; usually this is not the same as the command that must be used in *windows* to cause the download.
Reshape destroys the contents of the layer; it should work elsewhere.

NAME

button123, mouse, cursallow, cursinhibit, cursset, cursswitch, getrect123 − mouse control

SYNOPSIS

```
#include <jerq.h>

extern struct Mouse {
    Point xy;
    short buttons;
} mouse;

int button(n) int n;
int button1(), button2(), button3();
int button12(), button23(), button123();

void cursinhibit();
void cursallow();

void cursset(p); Point p;

Texture *cursswitch(t); Texture *t;

Rectangle getrect(n) int n;
Rectangle getrect1(), getrect2(), getrect3();
Rectangle getrect12(), getrect23(), getrect123();
```

DESCRIPTION

When the mouse is requested (see *request*(9.2)), the mouse state is updated asynchronously in the structure `mouse`. The coordinates of the mouse are held in `mouse.xy`, and the state of the buttons in `mouse.buttons`. Each process's `mouse` structure is independent of the others, so that (except for *cursset*) actions such as changing the tracking cursor do not affect the mouse in other processes.

The macro *button* and its counterparts return the state of the associated mouse button: non-zero if the button is depressed, 0 otherwise. The buttons are numbered 1 to 3 from left to right. *Button12* and the other multi-button functions return the OR of their states: true if either button 1 or button 2 is depressed.

Cursinhibit turns off interrupt-time cursor tracking (the drawing of the cursor on the screen), although the mouse coordinates are still kept current and available. *Cursallow* enables interrupt-time cursor tracking. *Cursallow* and *cursinhibit* stack: to enable cursor tracking after two calls to *cursinhibit*, two calls to *cursallow* are required.

Cursset moves the mouse cursor to the Point *p*.

Cursswitch changes the mouse cursor (a 16×16 pixel image) to that specified by the Texture `*`*t*. If the argument is `(Texture*)0`, the cursor is restored to the default arrow. *Cursswitch* returns the previous value of the cursor: the argument of the previous call to *cursswitch*.

Getrect prompts the user with a box cursor and waits for a rectangle to be swept out with the named button, identified as with the *button* primitives. It returns the screen coordinates of the box swept. The box may be partly or wholly outside the process's layer.

NAME

newlayer, dellayer, downback, lbitblt, lpoint, lrectf, lsegment, ltexture, upfront − layer control and graphics

SYNOPSIS

```
#include <jerq.h>

Layer *newlayer(r); Rectangle r;

void dellayer(l) Layer *l;

void lbitblt(sl, r, dl, p, f) Layer *sl, *dl; Rectangle r; Point p; Code
f;

void lpoint(l, p, f) Layer *l; Point p; Code f;

void lrectf(l, r, f) Layer *l; Rectangle r; Code f;

void segment(l, p, q, f) Layer *l; Point p, q; Code f;

void ltexture(l, r, t, f) Layer *l; Rectangle r; Texture *t; Code f;

void upfront(l) Layer *l;

void downback(l) Layer *l;
```

DESCRIPTION

Newlayer creates a layer in Rectangle *r* in the physical display bitmap, and returns its address, or 0 on failure. *Newproc*(9.2) explains how to attach a process to a layer.

Dellayer de-allocates a layer; the associated process must also be freed (see *newproc*(9.2)).

Upfront and *downback* are the subroutines corresponding to the *mux*(9.1) menu items `Top` and `Bottom`.

The routines *lbitblt*, *lpoint*, *lsegment* and *ltexture* are equivalent to their *bitblt*(9.3) counterparts except that they never inhibit the mouse cursor, so they are mainly useful only for implementation of efficient composite graphics operations such as circle-drawing. Because of the duality of Bitmaps and Layers, arguments of either type may be passed freely to any of the graphics primitives.

SEE ALSO

bitblt(9.3), *newproc*(9.2)
Rob Pike, *Graphics in Overlapping Bitmap Layers*, ACM Trans. on Graphics, April 1983.

NAME

P, newproc, muxnewwind, newwindow, tolayer, debug, getproc, getproctab, putname, getname − jerq process control

SYNOPSIS

```
#include <jerq.h>

extern struct Proc *P;

struct Proc *newproc(f) void (*f)();

struct Proc *newwindow(f); void (*f)();

void tolayer(l) Layer *l;

void debug();

struct Proc *getproc();

struct Proc *getproctab();

int putname(string, data) char *string; long data;

struct Nqueue *getname(string) char *string;

#include <msgs.h>
void muxnewwind(p, c) struct Proc *p; int c;
```

DESCRIPTION

Processes in the jerq consist of a coroutine-style process structure and an associated layer (see *newlayer*(9.2)), allocated independently. This section describes the process allocation and control primitives. They are direct links to the system's own control structures, so given *mux*'s open addressing, they should be used with care.

Each process has a global variable *P* that points to its process structure. The only regular use of *P* is to check that the process has been moved or reshaped:

```
if(P->state & RESHAPED){
	do_reshape();
	P->state &= ~RESHAPED;
}
```

The definition of `struct Proc` is in the include file `<jerqproc.h>`, which is included automatically by `<jerq.h>`.

Newproc allocates a new process, returning a pointer to it, or 0 if one cannot be allocated. Argument *f* points to the program text to be executed. The special case *f*=0 creates a process running the default terminal program, and is almost always how *newproc* should be called; use *32ld*(9.1) to run non-standard programs. A process is disabled by setting *p->state* to zero. After calling *newproc*, the process must be bound to a layer and Unix told of its presence, typically as:

```
struct Proc *p;
Rectangle r;
p = newproc((struct Proc *)0);
if(p == 0)
	error();
p->layer = newlayer(r);
if(p->layer == 0){
	p->state = 0;
	error();
}
p->rect = r;
muxnewwind(p, C_NEW);
```

The second argument to *muxnewwind* should be `C_RESHAPE` if an existing process is being given a new layer. If the process is *not* running the default terminal program, its variables `display` and `Drect` must be set:

```
struct udata *u=((struct udata *)p->data);
u->Drect=p->rect;
u->Jdisplayp=p->layer;
```

This procedure works regardless of whether the process being manipulated is itself.

Newwindow creates a process by the above procedure, going through the standard user interface to select the rectangle for the process's layer.

Tolayer takes an argument *layer* pointer and makes the process in that layer the receiver of mouse and keyboard events.

Getproc presents the user with a gunsight cursor and returns the address of the process whose layer is indicated with the mouse. *Getproctab* simply returns the address of the base of the process table array. This is an array of `NPROC` process structures. `NPROC` is stored in the word immediately lower in address than the process table.

Debug announces to the system that the calling process is prepared to handle exceptions by other processes.

Putname and *getname* manage a bulletin board for interprocess communication. Further communication may be arranged through shared memory. *Putname* associates *data* with *string*, returning nonzero normally, or 0 if the data could not be stored. *Getname* returns a pointer to a structure which contains

`struct Proc *proc`
: pointer to the process structure of the layer that most recently announced the string

`long data`
: the corresponding data

Getname returns 0 if no such string has been announced. A pointer returned by *getname* remains valid: a client may rendezvous with a server by calling *getname* once and repeatedly testing the associated `proc` pointer thereafter.

BUGS

These primitives are awkward at best, and are subject to change.
Creating a process without a layer or *vice versa* is dangerous.

NAME

request, own, wait, alarm, sleep, nap, kbdchar, rcvchar, realtime, sendchar, sendnchars, kill, exit − 5620 input/output requests

SYNOPSIS

```
#include <jerq.h>

void request(r) int r;

int own(r) int r;

int wait(r) int r;

void alarm(t) unsigned t;

void sleep(t) unsigned t;

void nap(t) unsigned t;

long realtime();

int kbdchar();

int rcvchar();

void sendchar(c) int c;

void sendnchars(n, cp) int n; char *cp;

void kill(s) int s;

void exit();
```

DESCRIPTION

Request announces a program's intent to use I/O devices and resources, and is usually called once early in a program. The bit vector *r* indicates which resources are to be used by OR'ing together one or more of the elements `KBD` (keyboard), `MOUSE`, `RCV` (characters received by terminal from Unix), `SEND` (characters sent from terminal to Unix) and `ALARM`. For example, `request(MOUSE|KBD)` indicates that the process wants to use the mouse and keyboard. If the keyboard is not requested, characters typed will be sent to the standard input of the Unix process. If the mouse is not requested, mouse events in the process's layer will be interpreted by the system rather than passed to the process. `SEND` and `CPU` (see `wait` below) are always implicitly requested. *Request* sleeps for one clock tick to synchronize mouse control with the kernel.

Own returns a bit vector of which I/O resources have data available. For example, `own()&KBD` indicates whether a character is available to be read by *kbdchar* (see below), `own()&MOUSE` tells if the process's `mouse` structure (see *button*(9.2)) is current, and `own()&ALARM` indicates whether the alarm timer has fired.

Wait's argument *r* is a bit vector composed as for *request*. *Wait* suspends the process, enabling others, until at least one of the requested resources is available. The return value is a bit vector indicating which of the requested resources are available — the same as `own()&r`.

Processes wishing to give up the processor to enable other processes to run may call `wait(CPU)`; it will return as soon as all other active processes have had a chance to run. `CPU` is a fake resource which is always requested. The `SEND` pseudo-resource is unused; `wait(SEND)` always succeeds.

Alarm starts a timer which will fire *t* ticks (60ths of a second) into the future. A pseudo-resource `ALARM` can be used to check the status of the timer with *own* or *wait*. Calling *alarm* implicitly requests the `ALARM` pseudo-resource.

Nap busy loops for *t* ticks of the 60Hz internal clock. To avoid beating with the display, programs drawing rapidly changing scenes should *nap* for two ticks between updates, to synchronize the display and memory. *Nap* busy loops until the time is up; *sleep* is identical except that it gives up the processor for the interval. Except when unwilling to give up the mouse, a program should call *sleep* in preference

to *nap*. *Sleep* does not interfere with *alarm*, and vice versa.

Realtime returns the number of 60Hz clock ticks since *mux* started.

Kbdchar returns the next keyboard character typed to the process. If no characters have been typed, or `KBD` has not been *request*ed, *kbdchar* returns −1.

Rcvchar returns the next character received from the host, typically written on the standard output of a Unix process. If there are no characters available, or `RCV` has not been *request*ed, *rcvchar* returns −1.

Sendchar sends a single byte to the host, which will normally be read on the standard input of the Unix process. *Sendnchars* sends to the host *n* characters pointed to by *p*.

Kill sends the associated Unix process the signal *s*; see *signal*(2).

Exit terminates the process. Unlike on Unix, *exit* does not return an exit status to a parent. Calling *exit* replaces the running process by the default terminal program. Any associated Unix process must arrange for its own demise; *exit* is a purely local function. When a process calls *exit*, all local resources: keyboard, mouse, storage, etc., are deallocated automatically.

Realtime returns the number of sixtieths of a second elapsed since *mux*(9.1) was started.

EXAMPLES

```
request(KBD|RCV);
for(;;){
	r=wait(KBD|RCV);
	if(r&KBD)
		keyboard(kbdchar());
	if(r&RCV)
		receive(rcvchar());
}
```

Take input from either the keyboard or the host.

SEE ALSO

button(9.2)

BUGS

`own()&MOUSE` does not guarantee that you own the mouse. The correct test is

```
(own()&MOUSE) && ptinrect(mouse.xy, Drect)
```

NAME

add, sub, mul, div, eqpt, eqrect, inset, muldiv, ptinrect, raddp, rsubp, rectXrect, rectclip − arithmetic on points and rectangles

SYNOPSIS

```
#include <jerq.h>

Point add(p, q) Point p, q;

Point sub(p, q) Point p, q;

Point mul(p, a) Point p; int a;

Point div(p, a) Point p; int a;

int eqpt(p, q) Point p, q;

int eqrect(r, s) Rectangle r, s;

Rectangle inset(r, n) Rectangle r; int n;

int muldiv(a, b, c) int a, b, c;

int ptinrect(p, r) Point p; Rectangle r;

Rectangle raddp(r, p) Rectangle r; Point p;

Rectangle rsubp(r, p) Rectangle r; Point p;

int rectXrect(r, s) Rectangle r, s;

int rectclip(rp, s) Rectangle *rp, s;
```

DESCRIPTION

Add returns the Point sum of its arguments: `Pt(`*p*`.x+`*q*`.x, `*p*`.y+`*q*`.y)`. *Sub* returns the Point difference of its arguments: `Pt(`*p*`.x−`*q*`.x, `*p*`.y−`*q*`.y)`. *Mul* returns the Point `Pt(`*p*`.x*`*a*`, `*p*`.y*`*a*`)`. *Div* returns the Point `Pt(`*p*`.x/`*a*`, `*p*`.y/`*a*`)`.

Eqpt and *eqrect* compare their arguments and return 0 if unequal, 1 if equal.

Inset returns the Rectangle `Rect(`*r*`.origin.x+`*n*`, `*r*`.origin.y+`*n*`, `*r*`.corner.x−`*n*`, `*r*`.corner.y−`*n*`)`. The following code creates a clear rectangle `r` with a 2-pixel wide border inside `r`:

```
rectf(&display, r, F_OR);
rectf(&display, inset(r, 2), F_CLR);
```

Muldiv is a macro that returns the 16-bit result `(`*a*`*`*b*`)/`*c*, with `(`*a*`*`*b*`)` calculated to 32 bits, so no precision is lost.

Ptinrect returns 1 if *p* is a point within *r*, and 0 otherwise.

Raddp returns the Rectangle `Rect(add(`*r*`.origin, `*p*`), add(`*r*`.corner, `*p*`))`; *rsubp* returns the Rectangle `Rect(sub(`*r*`.origin, `*p*`), sub(`*r*`.corner, `*p*`))`.

RectXrect returns 1 if *r* and *s* share any point; 0 otherwise.

Rectclip clips in place the Rectangle pointed to by *rp* so that it is completely contained within *s*. The return value is 1 if any part of **rp* is within *s*. Otherwise, the return value is 0 and **rp* is unchanged.

SEE ALSO

types(9.5)

NAME

alloc, free, balloc, bfree, gcalloc, gcfree − allocate memory

SYNOPSIS

```
#include <jerq.h>

char *alloc(nbytes) unsigned nbytes;

void free(s) char *s;

Bitmap *balloc(r) Rectangle r;

void bfree(b) Bitmap *b;

char *gcalloc(nbytes, where) unsigned long nbytes; char **where;

void gcfree(s) char *s;
```

DESCRIPTION

Alloc corresponds to the standard C function *calloc*; see *malloc*(3). It returns a pointer to a block of *nbytes* contiguous bytes of storage, or 0 if unavailable. The storage is aligned on 4-byte boundaries and is cleared to zeros. *Free* frees storage allocated by *alloc*.

Balloc returns a pointer to a Bitmap large enough to contain the Rectangle *r*, or 0 for failure. The coordinate system inside the Bitmap is set by *r*: the `origin` and `corner` of the Bitmap are those of *r*. *Bfree* frees the storage associated with a Bitmap allocated by *balloc*.

Gcalloc provides a simple garbage-compacting allocator. It returns a pointer to a block of *nbytes* contiguous bytes of storage, or 0 if unavailable. The storage is initialized to zeros. *Where* is a pointer to the user's data where the location of the block is to be saved. The return value of *gcalloc* is stored in **where* and will be updated after each compaction. Therefore, a program using *gcalloc* should never store the location of memory obtained from *gcalloc* anywhere other than *where*. Typically, this location is contained in a structure, such as a Bitmap (*balloc* uses *gcalloc*). *Gcfree* frees the storage block at *p*.

SEE ALSO

types(9.5), *malloc*(3)

NAME

Code, addr, bitblt, point, rectf, screenswap, segment, texture − graphics functions

SYNOPSIS

```
#include <jerq.h>

typedef int Code;
Code F_STORE, F_XOR, F_OR, F_CLR;

Word *addr(b, p) Bitmap *b; Point p;

void bitblt(sb, r, db, p, f) Bitmap *sb, *db; Rectangle r; Point p; Code
f;

void point(b, p, f) Bitmap *b; Point p; Code f;

void rectf(b, r, f) Bitmap *b; Rectangle r; Code f;

void screenswap(b, r, s) Bitmap *b; Rectangle r, s;

void segment(b, p, q, f) Bitmap *b; Point p, q; Code f;

void texture(b, r, t, f) Bitmap *b; Rectangle r; Texture *t; Code f;
```

DESCRIPTION

The type `Code` tells the graphics primitives what operation perform. The possible values are:

`F_STORE`	*target* = *source*
`F_OR`	*target* `|=` *source*
`F_XOR`	*target* `^=` *source*
`F_CLR`	*target* `&=~`*source*

In other words, if a Rectangle is copied to another place with Code `F_OR`, the result will be the bitwise OR of the contents of the source Rectangle and the target area. For operations with no explicit source, such as line drawing, the source is taken to be an infinite bitmap with zeros everywhere except on the object (e.g. line) generated by the operator, with coordinates aligned with the destination bitmap. `F_STORE` is the same as `F_OR` for non-rectangular operations.

Addr returns the address of the Word containing the bit at Point *p* in the Bitmap *b*.

Bitblt (bit-block transfer) copies the data in Rectangle *r* in Bitmap *sb* to the congruent Rectangle with *origin p* in Bitmap *db*. The nature of the copy is specified by the Code *f*.

Point draws the pixel at location *p* in the Bitmap *b* according to *Code f*.

Screenswap does an in-place exchange of the on-screen Rectangle *s* and the Rectangle *r* within the Bitmap *b*. Its action is undefined if *r* and *s* are not congruent. The Rectangle *s* is not clipped to the Bitmap *b*, only to the screen.

Segment draws a line segment in Bitmap *b* from Point *p* to *q*, with Code *f*. The segment is half-open: *p* is the first point of the segment and *q* is the first point beyond the segment, so adjacent segments sharing endpoints abut. Like all the other graphics operations, *segment* clips the line so that only the portion of the line intersecting the bitmap is displayed.

Texture draws, with function *f* in the Rectangle *r* in Bitmap *b*, the Texture specified by *t*. The texture is replicated to cover *r*. *Rectf* is equivalent to *texture* with **t* set to all one's.

In the above definitions, the type Bitmap may be replaced with Layer anywhere; see *newlayer*(9.2).

SEE ALSO

types(9.5)

NAME

circle, disc, arc, ellipse, eldisc, elarc − circle-drawing functions

SYNOPSIS

```
#include <jerq.h>

void circle(bp, p, r, f) Bitmap *bp; Point p; int r; Code f;

void disc(bp, p, r, f) Bitmap *bp; Point p; int r; Code f;

void arc(bp, p0, p1, p2, f) Bitmap *bp; Point p0, p1, p2; Code f;

void ellipse(bp, p, a, b, f) Bitmap *bp; Point p; int a, b; Code f;

void eldisc(bp, p, a, b, f) Bitmap *bp; Point p; int a, b; Code f;

void elarc(bp, p0, a, b, p1, p2, f) Bitmap *bp; Point p0, p1, p2; int a,
b; Code f;
```

DESCRIPTION

Circle draws the best approximate circle of radius *r* centered at Point *p* in the Bitmap *bp* with Code *f*. The circle is guaranteed to be symmetrical about the horizontal, vertical and diagonal axes. *Disc* draws the corresponding disc.

Arc draws a circular arc centered on *p0*, traveling counter-clockwise from *p1* to the point on the circle closest to *p2*.

Ellipse draws an ellipse centered at *p* with horizontal semi-axis *a* and vertical semi-axis *b* in Bitmap *bp* with Code *f*. *Eldisc* draws the corresponding elliptical disc. *Elarc* draws the corresponding elliptical arc, traveling counter-clockwise from the ellipse point closest to *p1* to the point closest to *p2*. (Beware the regrettable difference between the calling conventions for *arc* and *elarc*.)

BUGS

When an endpoint of an arc lies near a tail of an ellipse so thin that its ends degenerate into straight lines, *elarc* does not try to distinguish which side of the tail the point belongs on.

NAME

cos, sin, atan2, sqrt, norm − integer math functions

SYNOPSIS

```
int cos(d) int d;

int sin(d) int d;

int atan2(x, y) int x, y;

int norm(x, y, z) int x, y, z;

int sqrt(x) long x;
```

DESCRIPTION

Cos and *sin* return scaled integer approximations to the trigonometric functions. The argument values are in degrees. The return values are scaled so that `cos(0)==1024`. Thus, to calculate the mathematical expression $x = a\cos(d)$, the multiplication must be scaled:

```
x = muldiv(x0, cos(d), 1024)
```

Atan2 returns the approximate arc-tangent of y/x. The return value is in integral degrees.

Sqrt returns the 16-bit signed integer closest to the square root of its 32-bit signed argument.

Norm returns the Euclidean length of the three-vector (x, y, z).

DIAGNOSTICS

Sqrt returns 0 for negative arguments; and `atan2(0,0)==0`. *Norm* does not protect against overflow.

BUGS

Atan2 may be off by as much as two degrees.

NAME

libc − standard library functions

DESCRIPTION

Various standard functions from Section 3 are available in 5620 programs:

```
abs                    strchr
atoi                   strrchr
atol                   strcmp
chrtab                 strcpy
qsort                  strncat
rand                   strncmp
srand                  strncpy
strcat                 strlen
```

In addition, certain *stdio*(3) programs are available under the *jx*(9.1) emulator.

SEE ALSO

arith(3), *atof*(3), *chrtab*(3), *libc*(9.3), *qsort*(3), *rand*(3), *string*(3)

NAME

menuhit, hmenuhit − present user with menu and get selection

SYNOPSIS

```
#include <jerq.h>

int menuhit(m, b)
Menu *m;

#include <menu.h>

NMitem *hmenuhit(m, b)
NMenu *m;
```

DESCRIPTION

Menuhit presents the user with a menu specified by the Menu pointer *m* and returns an integer indicating the selection made, or −1 for no selection. The integer *b* specifies which button to use for the interaction: 1, 2 or 3. *Menuhit* assumes that the button is already depressed when it is called. The user makes a selection by lifting the button when the cursor points at the desired selection; lifting the button outside the menu indicates no selection.

Menus can be built in two ways, either as an array of strings or with a generator function:

```
typedef struct {
      char  **item;           /* string array, ending with 0 */
      char  *(*generator)();  /* used if item == 0 */
      short prevhit;          /* offset from top of last select */
      short prevtop;          /* topmost item displayed */
} Menu;

char *menutext[]={"Item 0", "Item 1", "Item 2", 0};
Menu stringsmenu={ menutext };
```

or

```
char *menugen();
Menu genmenu={ (char **)0, menugen };
```

The generator function is passed an integer parameter *n*, and must return the string for the *n*th menu entry, or 0 if *n* is beyond the number of entries in the menu. The *n*'s may come in any order but the result is only needed until the next call.

Regardless of the method of generation, characters with the `0200` bit set are regarded as fill characters. For example, the string `"\240X"` will appear in the menu as a right-justified `X` (`040` is the ASCII space character). Menu strings without fill characters are drawn centered in the menu.

The fields *prevhit* and *prevtop* are used to guide which items are displayed and which item the mouse points to initially. They should be nonnegative. Both *menuhit* and *hmenuhit* may choose to ignore these fields.

Hmenuhit supports hierarchical menus. Submenus are denoted graphically by a right-pointing arrow. Moving the cursor onto the arrow causes the submenu to appear. Hierarchical menus are built of `NMitem`s defined as

```
typedef struct NMenu {
      char  *text;
      char  *help;
      struct NMenu *next;
      void  (*dfn)(), (*bfn)(), (*hfn)();
      long  data;
} NMitem;
```

The `text` field is shown to the user; characters with the `0200` bit set behave as above. The contents of the `help` field are shown whenever the user holds down button 1 at the same time as the button specified by the parameter *b*. If *b* is 1, you get help all the time. The `next` field is the address of a submenu or `(NMenu *)0` if there is none. The two functions `(*dfn)()` and `(*bfn)()` support dynamic submenus. *Dfn* is called just before the submenu is invoked. Its argument is the current menu item. Similarly, *bfn* is called with the current menu item just after the submenu has finished. *Hfn* is called only when a menu item is selected; its argument is the current menu item. The menu has been undrawn before *hfn* is called. The return value from *hmenuhit* is the menu item selected or `(NMenu *)0` if none was selected. To permit communication between menu functions and the calling program, the *data* field is available for the user; it is ignored by *hmenuhit*.

An `NMenu`, like a `Menu`, may be built by list or by generator. An `NMenu` generator takes an integer parameter *n* and returns a pointer to an `NMitem`. In either case, the list of menu items is terminated by an item with a 0 `text` field.

EXAMPLES

Simple code to use `stringsmenu` declared above:

```
switch(menuhit(&stringsmenu, 3)){
case 0:  item_0();
         break;
case 1:  item_1();
         break;
case 2:  item_2();
         break;
case -1: noselection();
         break;
}
```

To provide a submenu for item 1:

```
NMitem *gen();
NMenu i1list = { 0, gen };
void item_2(), item_3();
NMitem imenu = {
      { "item 1", "item 1 help", &i1list },
      { "item 2", "item 2 help", 0, 0, 0, item_2 },
      { "item 3", 0, 0, 0, 0, item_3 },
      { 0 }
};
NMenu b3 = { imenu };
(void)hmenuhit(&b3, 3);
```

NAME

strinsure, strinsert, strdelete, strzero, setmuxbuf, getmuxbuf, movstring − dynamic strings in mux

SYNOPSIS

```
#include <jerq.h>

strinsure(s, n); String *s;

strinsert(d, i, s); String *d, *s;

strdelete(d, i, j); String *d;

strzero(d); String *d;

setmuxbuf(s); String *s;

getmuxbuf(d); String *d;

movstring(n, sp, dp); char *sp, *dp;
```

DESCRIPTION

These functions manipulate strings represented in the following form.

```
struct String {
        char *s;
        short n;
        short size;
};
```

The string proper occupies the first `n` characters of a data block of `size` characters pointed to by `s`. Initially both `size` and `s` should be 0. Strings are always counted, not terminated by `\0`. The functions obtain space as needed from *gcalloc*; see *alloc*(9.3). Thus a `String` structure should never be copied.

Strinsure arranges that *s*`->size`≥*n*. It must be called before any operation that could overflow the current size.

Strinsert inserts a copy of source *s* into destination *d* beginning at character *i* (counted from 0), adding *s*`->n` to *d*`->n`.

Strdelete removes characters *i* through *j*−1 from string *d*, subtracting *j*−*i*, which must be nonnegative, from *d*`->n`.

Strzero frees the memory associated with *d* and sets both *d*`->n` and *d*`->size` to zero.

Setmuxbuf copies string *s* into the snarf buffer maintained by *mux*(9.1); *getmuxbuf* copies from the snarf buffer into *d*.

Movstr copies a block of *n* characters beginning at *sp* to a block beginning at *dp*. If *n* is negative it copies −*n* characters ending at *sp*−*1* to a block ending at *dp*−1. Notice that *movstring* does not operate on `String`s.

SEE ALSO

libc(9.3)

NAME

string, defont, strwidth, infont, outfont, getfont − text and font operations

SYNOPSIS

```
#include <jerq.h>
#include <font.h>

Point string(ft, s, b, p, f) Font *ft; char *s; Bitmap *b; Point p; Code
f;

extern Font defont;

int strwidth(ft, s) Font *ft; char *s;

Font *infont(inch) int (*inch)();

int outfont(ft, ouch) Font *ft; int (*ouch)();

void ffree(ft) Font *ft;

#include <jerqio.h>

Font *getfont(file) char *file;
```

DESCRIPTION

String draws the null-terminated string *s* using characters from font *ft* in Bitmap *b* at Point *p*, with Code *f*. The return value is the location of the first character *after s*; passed to another call to *string*, the two strings will be concatenated. The characters are drawn such that the `origin` point of the bounding rectangle of a maximum height character lies at *p*. Therefore, a character drawn on the screen at (0,0) will occupy the upper-leftmost character position on the screen. *String* draws characters as they are in the font. No special action is taken for control characters such as tabs or newlines.

The global *defont* is the name of the standard font (not a pointer to it).

Strwidth returns the width in pixels of the null-terminated string *s*, interpreted in the Font **ft*. The height of a character string is simply `ft->`*height.*

Infont creates a font by reading the byte-wise binary representation returned by successive calls to *inch*. It returns 0 on error. *Inch* must return successive bytes of the Unix file representation of the font, and −1 at end-of-file. *Outfont* calls the routine *ouch* to write successive bytes of the binary representation of font *ft*. It returns −1 on error, as must *ouch* . For programs running under *jx*, *getfont* returns a pointer to a font read from the named *file*, essentially by calling *infont* with argument routine *getc*. It returns 0 on error. *Ffree* frees a font allocated by *infont* or *getfont*.

NAME

thinkchar, thinknchars, thinkflush, thinkmap, thinkabort − ThinkJet routines

SYNOPSIS

```
#include <jerq.h>
#include <thinkclient.h>

int thinkchar(c) int c;

int thinknchars(n, p) int n; char *p;

int thinkflush()

int thinkmap(b, r) Bitmap *b; Rectangle r;

int thinkabort()
```

DESCRIPTION

These macros provide access to the routines used internally by *thinkblt*(9.1). *Thinkchar* and *thinknchars* send characters to the printer; characters are buffered so that *thinkflush* must be called after the last transmission. *Thinkmap* sends all or part of a bitmap (it calls *thinkflush* automatically). *Thinkabort* stops transmission as quickly as possible, throwing away any characters that may be queued up.

SEE ALSO

thinkblt(9.1), *newproc*(9.2), *types*(9.5)

DIAGNOSTICS

The routines return zero on success, a positive value on failure, and a negative value if *thinkblt*(9.1) is not loaded.

NAME

jioctl − mux ioctl requests

SYNOPSIS

```
#include "/usr/jerq/include/jioctl.h"

ioctl(fd, request, 0)

ioctl(fd, JWINSIZE, win)
struct winsize *win;
```

DESCRIPTION

Mux(9.1) supports several *ioctl*(2) requests for Unix programs attached to layers. The requests are:

`JMUX` returns 0 if file descriptor *fd* is connected to a *mux* layer, −1 otherwise.

`JTERM` resets the layer connected to *fd* to the default terminal program.

`JBOOT` initiates the down-load protocol to replace the layer's terminal program. Usually called by *32ld*(9.1).

`JZOMBOOT`
is the same as *JBOOT*, but disables execution of the program when the download is complete (see the `-z` flag of *32ld*).

`JWINSIZE`
returns, in the location pointed to by the third argument, a structure describing the size of the layer connected to *fd*, with character `0` being the unit of size. The structure is:

```
struct winsize {
      char  bytesx, bytesy;  /* size in characters */
      short bitsx, bitsy;    /* size in pixels */
};
```

`JEXIT` causes *mux* to exit.

SEE ALSO

32ld(9.1), *mux*(9.1), *ioctl*(2)

NAME

mouse − jerq mouse user interface

DESCRIPTION

Most jerq programs use the mouse for control, either by pointing at things on the screen or by making selections from a menu. The mouse buttons are different from keys on a keyboard in that events are reported when a button is released (let ‘up’) as well as depressed (pressed ‘down’). It therefore matters not only *where* and *when* a button is pressed, but for how long. For example, menus are drawn when a button is depressed, and remain displayed as long as the button is held down. While the button is down, moving the cursor over the menu highlights entries in the menu; the entry (possibly none) under the cursor when the button is *released* is the selection returned to the program. Large menus also present a ‘scroll bar’ on the left side of the menu. Moving the mouse inside the scroll bar chooses which subset of the available entries are displayed and therefore selectable.

There is a convention about how the buttons are used. The left button (button 1) is used to point: selecting which layer to work in, which file inside the editor, some text in the file, etc. The middle button (button 2) produces a menu of actions related to the selection: remove the selected text, replace it, etc. The right button (button 3) presents a menu of global, program-wide actions: pick up a new file, rearrange the files on the screen, etc. Programs follow this convention well enough that an unfamiliar program can often be learned simply by trying it. The main violators of the convention are drawing programs, which use button 1 to draw things and button 2 to undraw them, but this is also a consistent convention.

The mouse cursor is usually an arrow pointing at a pixel, but programs often change the cursor to an iconic representation of the program’s state. The most common cursors are:

arrow standard cursor

coffee cup
: Program will be busy for a while.

rectangle and arrow
: Program expects a rectangle to be ‘swept out’ by pressing a button (usually 3) at one corner and releasing at the diagonally opposite corner.

gunsight
: Program expects an object to be selected by pointing at it and pressing a button (usually 3).

upside-down mouse
: Program is thinking; the mouse is inoperative.

NAME

bitfile − format of bitmap file

DESCRIPTION

Binary files produced by *blitblt*(9.1) and other bitmap-generating programs are formatted as follows:

Byte no.	Description
0, 1:	Zero.
2, 3:	*x*-coordinate of the rectangle origin (low-order byte, high-order byte).
4, 5:	*Y*-coordinate of the rectangle origin (low-order byte, high-order byte).
6, 7:	*x*-coordinate of the rectangle corner (low-order byte, high-order byte).
8, 9:	*Y*-coordinate of the rectangle corner (low-order byte, high-order byte).
remainder:	Compressed raster data. Each raster is exclusive-or'd with the previous one, and zero-extended (if necessary) to a 16-bit boundary. It is then encoded into byte sequences, each of which consists of a control byte followed by two or more data bytes:

Control	Data
n (< 127)	2×*n* bytes of raster data, running from left to right.
`0x80+`*n*	2 bytes of raster data, to be replicated from left to right *n* times.

There are also two ASCII formats in current use. Textures and 16×16 icons, as created by *icon*(9.1), are encoded as a `Texture` declaration with initializer, to be copied unchanged into C program source; see *types*(9.5). Faces and other large icons are without any surrounding C syntax. In either case, each scan line of the bitmap is a comma-separated list of C-style short hexadecimal constants; scan lines are separated by newlines.

SEE ALSO

blitblt(9.1), *icon*(9.1), *types*(9.5), *vismon*(9.1)

NAME

faced − network face server

SYNOPSIS

`/usr/net/face.go`

DESCRIPTION

The network face server provides a database of 48×48 bit icons and other facial representations. It is implemented as a network file system similar to *netfs*(8).

The file system, conventionally mounted on `/n/face`, has a fixed three-level hierarchy. The first level is a machine name, the second level a user name, and the third level a resolution. Thus the file `/n/face/kwee/pjw/48x48x1` is the standard face icon (for user pjw) on machine kwee:



Many local users also have 512×512 byte high-resolution faces, named `512x512x8`. Other resolutions may also be present for a particular face. One-bit images are stored in the format used by *icon*(9.1); eight-bit images are arrays of bytes. The directories for machines sharing a user community, such as those on a Datakit node, are linked together and given a name appropriate to the community. For example, `/n/face/kwee` is a link to `/n/face/astro`.

To access the face for a mail name *machine*`!`*uid* take the result of the first successful open from the following list of files:

```
/n/face/machine/uid/48x48x1
/n/face/misc./uid/48x48x1
/n/face/machine/unknown/48x48x1
/n/face/misc./unknown/48x48x1
```

The directory `misc`. holds faces for generic users such as `root` and `uucp`. The face server is made available on a machine by running `/usr/net/face.go` from *rc*(8).

The face server data is administered by a pair of ASCII files that associate related machines and faces. The machine table `machine.tab` attaches machines to communities; in it the line

```
kwee=astro
```

puts machine `kwee` in community `astro`. The people table `people.tab` associates a machine/user pair in the face server with a file in one of the source directories `/n/kwee/usr/jerq/icon/face48` or `/n/kwee/t0/face/512x512x8`. Thus

```
astro/pjw=pjweinberger
```

causes the images stored in source files named `pjweinberger` to be available in the face server in directory `/n/face/astro/pjw`. As well, each disk file used by the face server is linked (by its original name) into the directory `/n/face/48x48x1` or `/n/face/512x512x8` for easy access to all the images.

FILES

`/n/kwee/usr/jerq/icon/face48`	directory of low resolution faces
`/n/kwee/t0/face/512x512x8`	directory of high resolution faces
`/n/kwee/usr/net/face/people.tab`	people/file equivalences
`/n/kwee/usr/net/face/machine.tab`	machine/community equivalences

SEE ALSO

netfs(8), *face*(9.7), *icon*(9.1), *vismon*(9.1)

BUGS

After updating the tables, an indeterminate time may pass before the new faces are available.
All face server files are unwritable.

NAME

movies − graphics movie file formats

DESCRIPTION

Movie files are generated by *preflicks* and used by *fflicks*; see *flicks*(9.1). The format of a movie files is:

```
struct Header {
      unsigned char version;
      short header_length;
      short nr_frames;
      unsigned char nr_tables;
      struct LOOKUP_TABLE {
            short number_of_entries;
            struct {
                  short count;
                  unsigned char value;
            } table[256];
      } Table[nr_tables];
};
struct Frame {
      short width, height;
      short compacted_length;
      unsigned char which_table;
      unsigned char data[compacted_length];
} Frame[nr_frames];
```

Each `short` in the above structure is present as a two-byte number in the file, most significant byte first. Each `unsigned char` is a single byte.

`version`
: software version number, to ensure compatibility between producer and consumer of the file.

`header_length`
: total length in bytes of the lookup table(s) used to encode the file plus three bytes (the next three that follow).

`nr_frames`
: total number of movie frames in the file.

`nr_tables`
: number of lookup tables.

`nr_entries`
: number of entries in the lookup table (maximum 256).

`count value` pixel value and a count of how many times that value is to be repeated.

Immediately following the lookup tables begin the frames encoded in an indirect run-length code. Each frame is described by `width`, `height`, and the `compacted_length` of the frame in bytes. The frame is coded in raster-scan order as a sequence of indexes into the table numbered `which_table` (counting from 0).

FILES

_movie

SEE ALSO

flicks(9.1), *pico*(1), *rebecca*(9.1)

NAME

font − jerq font layouts

SYNOPSIS

```
#include <jerq.h>
#include <font.h>

typedef struct Fontchar Fontchar;
typedef struct Font Font;

extern Font defont;
```

DESCRIPTION

A *Font* is a character set, stored as a single Bitmap with the characters placed side-by-side. It is described by the following data structures.

```
typedef struct Fontchar {
        short x;                /* left edge of bits */
        unsigned char top;      /* first non-zero scan-line */
        unsigned char bottom;   /* last non-zero scan-line */
        char left;              /* offset of baseline */
        unsigned char width;    /* width of baseline */
} Fontchar;
typedef struct Font {
        short n;                /* number of chars in font */
        char height;            /* height of bitmap */
        char ascent;            /* top of bitmap to baseline */
        long unused;
        Bitmap *bits;           /* where the characters are */
        Fontchar info[n+1];     /* n+1 character descriptors */
} Font;
```

Characters in `bits` abut exactly, so the displayed width of the character *c* is `Font.info[`*c*`+1].x - Font.info[`*c*`].x`. The first `left` columns of pixels in a character overlap the previous character. The upper left corner of the nonempty columns appears at (`x`,0) in the bitmap. `Width` is the distance to move horizontally after drawing a character. The font bitmap has a fixed `height`; parameters `top` and `bottom` may speed up the copying of a character.

Characters are positioned by their upper left corners.

Fonts are stored on disk in binary with byte order that of the terminal. First in the file is the Font structure with `bits` elided. The data for the bitmap follows. The header for the bitmap must be inferred from `Font.height` and `Font.info[Font.n].x`.

EXAMPLES

```
Fontchar *i = f->info + c;
bitblt(f->bits, Rect(i->x, i->top, (i+1)->x, i->bottom),
        &display, Pt(p.x+i->left, p.y+i->top), fc);
p.x += i->width;
```

Copy character *c* from font *f* to point *p* with Code `F_XOR` or `F_OR`.

For Code `F_STORE`, use `Rect(i->x, 0, (i+1)->x, f->height)`.

SEE ALSO

jf(9.1), *string*(9.3), *getfont*(9.1)

BUGS

The `unused` field is used, by *getfont*(9.1).

NAME

pads − user interface package

DESCRIPTION

Pads is a mouse-based interface for browsing a network of windows.

Button 1 points. Pointing at a window makes it current, with a heavy border; pointing at a line of text makes it current, inverts its video, and scrolls it to the middle of the window. A scroll bar at the left of each window shows how much of the text of a window is visible; pointing into the scroll region controls what text is displayed.

Button 2 has a menu of operations that apply to the current line. Operations above the ~~~~~ separator are specific to each line; operations below the separator are generic line operations:

`cut` Remove the line.

`sever` Remove the line and all lines above it.

`fold` If lines pass the right margin, continue them on following lines.

`truncate` Truncate lines at the right margin.

Button 3 has a menu of window-level operations, and is in three parts. Below the lower separator is a list of windows; selecting one makes it current. They appear in front-to-back screen order, current at the top. Operations above the upper separator are specific to each window; operations between the separators are generic window operations:

`reshape`
`move`
`close` Like `reshape`, `move`, and `delete` in *mux*(9.1).

`fold`
`truncate` apply to all lines in the window.

Keyboard characters accumulate at the bottom of the layer. If the current line accepts input, it flashes with each keystroke; otherwise, if the current window accepts input, its border flashes. Carriage return is ignored until a line or window accepts the text, whereupon the input line is sent to the line or window. The ESC key substitutes the *mux*(9.1) global snarf buffer.

If the first character of a line from the keyboard is < or > the remainder of the line is interpreted as a shell command. For <, each line of the command's standard output is sent to the line or window, as though it had come from the keyboard. For >, the line or lines of the window become the command's standard input. Each line or window that accepts keyboard input produces some help in response to `?`. Special cursor icons occasionally appear:

arrow-dot-dot-dot
: The host is completing an operation; the terminal is ready asynchronously.

exclamation mark
: Confirm a dangerous menu selection by pressing that menu's button again.

SEE ALSO

T. A. Cargill, *Pads Programming Guide*

NAME

Word, Point, Rectangle, Bitmap, Texture, Pt, Rect, Rpt, display, Drect, Jrect − graphics data types

SYNOPSIS

```
#include <jerq.h>

typedef int Word;
typedef struct Point Point;
typedef struct Rectangle Rectangle;
typedef struct Bitmap Bitmap;
typedef struct Texture Texture;

extern Bitmap display;
extern Rectangle Drect, Jrect;

Point Pt(x, y) int x, y;

Rectangle Rect(x0, y0, x1, y1) int x0, y0, x1, y1;

Rectangle Rpt() Point p0, p1;
```

DESCRIPTION

A `Word` is a 32-bit integer, and is the unit of storage used in the graphics software.

A `Point` is a location in a Bitmap (see below), such as the display, and is defined as:

```
typedef struct Point {
        short x;
        short y;
} Point;
```

The coordinate system has *x* increasing to the right and *y* increasing down. All objects and operators in the graphics world live in the same coordinate space—that of the display bitmap.

A `Rectangle` is a rectangular area in a Bitmap.

```
typedef struct Rectangle {
        Point origin;       /* upper left */
        Point corner;       /* lower right */
} Rectangle;
```

By definition, `origin.x <= corner.x` and `origin.y <= corner.y`. By convention, the right (maximum *x*) and bottom (maximum *y*) edges are excluded from the represented rectangle, so abutting rectangles have no points in common. Thus, `corner` contains the coordinates of the first point beyond the rectangle. The image on the display is contained in the Rectangle `{0, 0, XMAX, YMAX}`, where `XMAX`=800 and `YMAX`=1024.

A `Bitmap` holds a rectangular image, stored in contiguous memory starting at *base*.

```
typedef struct Bitmap {
        Word    *base;          /* pointer to start of data */
        unsigned width;         /* width in Words of total data area */
        Rectangle rect;         /* rectangle in data area, screen coords */
} Bitmap;
```

Each `width` Words of memory form a scan-line of the image, and `rect` defines the coordinate system inside the `Bitmap`: `rect.origin` is the location in the Bitmap of the upper-leftmost point in the image. The coordinate system is arranged so *x* positions equal to 0 mod 32 are in the leftmost bit of a Word.

A `Texture` is a 16×16 dot bit pattern.

```
typedef struct {
        Word bits[16];
```

```
} Texture;
```

Textures are aligned to absolute display positions, so adjacent areas colored with the same Texture mesh smoothly.

The functions *Pt*, *Rect* and *Rpt* construct geometrical data types from their components. Since they are implemented as macros, they only work in function argument lists.

The global *display* is a Bitmap describing the display area of the process. *Drect* is a Rectangle defining, in screen coordinates, the display area available to the program (inside the layer's border). *Jrect* is the Rectangle `{0, 0, XMAX, YMAX}`.

NAME

face, mugs − show faces, make face icons from pictures

SYNOPSIS

`face` *machine*`!`*user file* ...

`mugs` [`-a`]

DESCRIPTION

Face displays the 48×48 bit icons specified by its arguments. If an argument contains an exclamation mark, it is assumed to be a machine and user pair and is looked up in the face file system, *faced*(9.5); otherwise it is taken to be a file name. If the file does not exist and contains no slashes, it is looked up in `/n/face/48x48x1`.

When *face*'s layer is full, it waits for a character to be typed before continuing.

Mugs interactively converts grey-scale images in the form of *picfile*(5) into 48×48 icons for display by *face* and *vismon*(9.1). It prompts for the name of a picture file, displaying a large approximation to the original picture and a matrix of 48×48 icons of varying contrast and brightness. Button 1 selects one of the 48×48's. Button 3 presents a menu with entries:

`window`
: Select a square window in the large picture using button 3. Touch down at the top and center of the square and slide around to adjust its size. Appropriately cropped 48×48's will be displayed.

`in`
: Zoom in on a smaller part of contrast-brightness space, displaying an array of 48×48's that look more-or-less like the selected one. Repeated `in`s will zoom in farther.

`out`
: Opposite of `in`.

`save`
: Type in the name of a file in which to save the currently selected 48×48.

`read`
: Type in the name of a picture file containing the next face to process.

`exit`
: Confirm with button 3.

Option `-a` indicates that picture files have non-square pixels with aspect ratio 1.25, as produced by the ITI frame-grabber attached to kwee. Normally pixels are assumed to be square.

EXAMPLES

`face /n/face/coma/*/48x48x1`
: All the users of coma.

SEE ALSO

faced(9.5), *icon*(9.1), *vismon*(9.1), *imscan*(1), *picfile*(5)

NAME

reader − electronic retrieval of typeset documents

SYNOPSIS

`reader` *name*

DESCRIPTION

Reader presents the named paper on a 5620 terminal in a form designed for readability, not for similarity to the printed version. The *name* is a pathname for a manuscript in the *papers*(7) database with any final `.d` elided) or the name of a *troff*(1) input file. Mouse button 1 selects subheads; button 3 moves forward (‘more’) or backward (‘less’). The program exits completely on button 2, or tentatively (to avoid downloading upon reexecution) on button 3 (‘done’).

When the text in a screen overlaps text in a previous screen, a tick mark in the bar (not a scroll bar) at the left of the screen shows where new material begins.

Fully installed papers in the database, which appear as directories suffixed `.d`, have been preprocessed so that *reader* can present figures and complex equations. In *troff* input, it understands straightforward text and *eqn*(1), the macro packages *ms*(6), *mm*, and *me*, but cannot handle arbitrary motions such as appear in figures and complex equations.

FILES

`/n/bowell/pap/Titles`	titles, authors and installation dates
`/n/bowell/pap/*org`	membership list
`/n/bowell/pap/`*center*`/`*department*`/`*author*`/`*papername*[`.d`]	

SEE ALSO

troff(1), *proof*(9.1), *docsubmit*(1), *papers*(7)

BUGS

Button 1 knows only already-read subheads unless the paper has been preprocessed.
Reader can only handle papers written in *troff* with standard (`−ms`, `−mm`, `−me`) macro packages.