NAME

ed – editor

SYNOPSIS

ed [–] [name]

DESCRIPTION

Ed is the standard text editor.

If a *name* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed's* buffer so that it can be edited. The optional – simulates an **os** command (see below) which suppresses the printing of characters counts by *e*, *r*, and *w* commands.

Ed operates on a copy of any file it is editing; changes made in the copy 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*. There is only one 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. Every command which requires addresses has default addresses, so that the addresses can often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in 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 is an expression which specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. The regular expressions allowed by *ed* are constructed as follows:

1. An ordinary character (not one of those discussed below) is a regular expression and matches that character.
2. A circumflex '^' at the beginning of a regular expression matches the null character at the beginning of a line.
3. A currency symbol '$' at the end of a regular expression matches the null character at the end of a line.
4. A period '**.**' matches any character but a new-line character.
5. A regular expression followed by an asterisk '*' matches any number of adjacent occurrences (including zero) of the regular expression it follows.
6. A string of characters enclosed in square brackets '[ ]' matches any character in the string but no others. If, however, the first character of the string is a circumflex '^' the regular expression matches any character but new-line and the characters in the string.
7. The concatenation of regular expressions is a regular expression which matches the concatenation of the strings matched by the components of the regular expression.
8. The null regular expression standing alone is equivalent to 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.

Addresses are constructed as follows. 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 by each command is discussed under the description of the command.

1. The character '**.**' addresses the current line.
2. The character '^' addresses the line immediately before the current line.
3. The character '$' addresses the last line of the buffer.
4. A decimal number *n* addresses the *n*-th line of the buffer.
5. ''*x*' addresses the line associated (marked) with the mark name character *x* which must be a printable character. Lines are marked with the *k* command described below.
6. A regular expression enclosed in slashes '/' addresses the first line found by searching toward the end of the buffer and stopping at the first line containing a string matching the regular expression. If necessary the search wraps around to the beginning of the buffer.
7. A regular expression enclosed in queries '?' addresses the first line found by searching toward the beginning of the buffer and stopping at the first line found containing a string matching the regular expression. If necessary the search wraps around to the end of the buffer.
8. 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.

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. This feature can be used to determine the starting line for forward and backward searches ('/', '?'). 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.

As mentioned, it is generally illegal for more than one command to appear on a line. However, any command may be suffixed by 'p' (for 'print'). In that case, the current line is printed after the command is complete.

(**.**)a
<text>
.

The append command reads the given text and appends it after the addressed line. '**.**' 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.

(**.**,**.**)c
<text>
.

The change command deletes the addressed lines, then accepts input text which replaces these lines. '**.**' is left at the last line input; if there were none, it is left at the first line not changed.

(**.**,**.**) d

The delete command deletes the addressed lines from the buffer. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e filename

The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in. '**.**' is set to the last line of the buffer. The number of characters read is typed. 'filename' is remembered for possible use as a default file name in a subsequent *r* or *w* command.

f filename

The filename command prints the currently remembered file name. If 'filename' is given, the currently remembered file name is changed to 'filename'.

(1,$) g/regular expression/command list

In the global command, the first step is to mark every line which matches the given regular expression. Then for every such line, the given command list is executed with '**.**' 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 be ended with '\'. *A, i,* and *c* commands and associated input are permitted; the '**.**' terminating input mode may be omitted if it would be on the last line of the command list. The (global) commands, *g,* and *v,* are not permitted in the command list.

(**.**) i
<text>
.

This command inserts the given text before the addressed line. '**.**' is left at the last line input; if there were none, at the addressed line. This command differs from the *a* command only in the placement of the text.

(**.**) k*x*

The mark command associates or marks the addressed line with the single character mark name *x.* The ten most recent mark names are remembered. The current mark names may be printed with the *n* command.

(**.**,**.**) l

The list command will print the addressed lines in a way that is unambiguous: Non-graphic characters are printed in octal, prefixed characters are overstruck with a circumflex, and long lines are folded.

(**.**,**.**) m*a*

The move command will reposition the addressed lines after the line addressed by *a.* The last of the moved lines becomes the current line.

n

The *n* command will print the current mark names.

os
ov

After *os* character counts printed by *e, r,* and *w* are suppressed. *Ov* turns them back on.

(**.**,**.**) p

The print command prints the addressed lines. '**.**' is left at the last line printed. The *p* command *may* be placed on the same line after any command.

q

The quit command causes *ed* to exit. No automatic write of a file is done.

($) r filename

The read command reads in the given file after the addressed line. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). The remembered file name is not changed unless 'filename' is the very first file name mentioned. Address '0' is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed. '**.**' is left at the last line read in from the file.

(.,.)s/regular expression/replacement/ or,
(.,.)s/regular expression/replacement/g
The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all matched strings are replaced by the replacement specified, if the global replacement indicator 'g' appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or new-line may be used instead of '/' to delimit the regular expression and the replacement. '**.**' is left at the last line substituted.

An ampersand '&' appearing in the replacement is replaced by the regular expression that was matched. The special meaning of '&' in this context may be suppressed by preceding it by '\'.

(1,$)v/regular expression/command list
This command is the same as the global command except that the command list is executed with '**.**' initially set to every line *except* those matching the regular expression.

(1,$)w filename
The write command writes the addressed lines onto the given file. If the file does not exist, it is created mode 666 (readable and writeable by everyone). The remembered file name is *not* changed unless 'filename' is the very first file name mentioned. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). '**.**' is unchanged. If the command is successful, the number of characters written is typed.

($)=
The line number of the addressed line is typed. '**.**' is unchanged by this command.

!UNIX command
The remainder of the line after the '!' is sent to UNIX to be interpreted as a command. '**.**' is unchanged. The entire shell syntax is not recognized. See msh(VII) for the restrictions.

(.+1)<newline>
An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to '.+1p'; it is useful for stepping through text.

If an interrupt signal (ASCII DEL) is sent, *ed* will print a '?' and return to its command level.

If invoked with the command name '–', (see init(VII)) *ed* will sign on with the message 'Editing system' and print '*' as the command level prompt character.

Ed has size limitations on the maximum number of lines that can be edited, on the maximum number of characters in a line, in a global's command list, in a remembered file name, and in the size of the temporary file. The current sizes are: 4000 lines per file, 512 characters per line, 256 characters per global command list, 64 characters per file name, and 64K characters in the temporary file (see BUGS).

FILES

/tmp/etm?, temporary
/etc/msh, to implement the '!' command.

DIAGNOSTICS

'?' for errors in commands; 'TMP' for temporary file overflow.

SEE ALSO

A Tutorial Introduction to the ED Text Editor (internal memorandum)

BUGS

The temporary file can grow to no more than 64K bytes.