NAME

ed − text editor

SYNOPSIS

ed [−] [+] [name]

DESCRIPTION

Ed is the standard text editor.

If the *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 '−' suppresses the printing of character counts by *e, r,* and *w* (or *z*) commands.

Ed operates on a copy of the file it is editing; changes made in the copy have no effect on the file until a *w* or *z* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer.* There is only one buffer.

If changes have been made in the buffer since the last *w* or *z* command, *ed* warns the user if an attempt is made to destroy *ed's* buffer via the *q* or *e* commands. *Ed* prints 'q?' or 'e?', respectively, and allows one to continue editing. A second*q* or *e* command at this point will tak e effect. This warning feature may be inhibited by specifying the '+' option (e.g., ed + file). The '−' option also inhibits this feature.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single character *command,* possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that 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 te xt, 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; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be replaced. 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. The regular expressions allowed by *ed* are constructed as follows:

The following *one-character regular expressions* match a single character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character regular expression that matches itself.

1.2 A backslash '\' followed by any special character is a one-character regular expression that matches the special character itself. The special characters are:

a. '**.**', '*', '+', '[', and '\' (period, asterisk, plus sign, left square bracket, and backslash, respectively), which are always special, except when they appear within square brackets '[ ]' (see 1.4 below).

b. '^' (caret or circumflex), which is special at the beginning of an *entire regular expression* (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets '[ ]' (see 1.4 below).

c. '$' (currency symbol), which special at the end of an entire regular expression (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire regular expression, which is special for that regular expression (for example, see how '/' is used in the *g* command, below.)

1.3 A period '**.**' is a one-character regular expression that matches any character except the new-line character.

1.4 A non-empty string of characters enclosed in square brackets '[ ]' is a one-character regular expression that matches *any one* character in that string. If, however, the first character of the string is a circumflex '^', the one-character regular expression matches any character *except* new-line and the remaining characters in the string. The '^' has this special meaning *only* if it occurs first in the string. The minus '−' may be used to indicate a range of consecutive ASCII characters; for example, [0−9] is equivalent to [0123456789]. The '−' loses this special meaning if it occurs first (after an initial '^', if any) or last in the string. The ']' does not terminate such a string when it occurs first (after an initial '^', if any), in it, e.g., '[ ]a]' matches either a right square bracket ']' or the letter 'a'. The five characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *regular expressions* from *one-character regular expressions:*

2.1 A one-character regular expression is a regular expression that matches whatever the one-character regular expression matches.

2.2 A one-character regular expression followed by an asterisk '*' is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, this regular expression matches as many occurrences as possible.

2.3 A one-character regular expression followed by a plus '+' is a regular expression that matches *one* or more occurrences of the one-character regular expression. If there is any choice, this regular expression matches as many occurrences as possible.

2.4 A one-character regular expression followed by '\{*m*\}', '\{*m*,\}', or '\{*m*,*n*\}' is a re gular expression that matches a *range* of occurrences of the one-character regular expression. The values of *m* and *n* must be non-negative integers less than 256; '\{*m*\}' matches e xactly *m* occurrences; '\{ *m*,\}' matches at least *m* occurrences; '\{ *m*,*n*\}' matches any number of occurrences between *m* and *n* inclusive. Whenever a choice exists, the regular expression matches as many occurrences as possible.

2.5 The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.

2.6 A regular expression enclosed between the character sequences '\(' and '\)' is a regular expression that matches whatever the unadorned regular expression matches; this construction has side effects discussed under the *s* command, below.

Finally, an *entire regular expression* may be constrained to match only an initial segment or final segment of a line (or both):

3.1 A circumflex '^' at the beginning of an entire regular expression constrains that regular expression to match an *initial* segment of a line.

3.2 A currency symbol '$' at the end of an entire regular expression constrains that regular expression to match a *final* segment of a line. The construction *^entire regular expression$* constrains the entire regular expression to match the entire line.

The null regular expression standing alone (e.g., '//') is equivalent to the last regular expression encountered.

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; the exact effect on the current line is discussed under the description of the command. Addresses are constructed as follows:

1. The character '**.**' 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 mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A regular expression enclosed by slashes '/' addresses the first line found by searching forward from the line *following* the current line 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 and continues through the current line, so that the entire buffer is searched.
6. A regular expression enclosed in queries '?' addresses the first line found by searching backward from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the regular expression. If necessary the search wraps around to the end of the buffer and continues through the current line.
7. An address followed by a plus sign '+' or a minus sign '−' followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. 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'.
9. If an address ends with '+' or '−', then 1 is added or subtracted, respectively. As a consequence of this rule and of rule 8, the address '−' refers to the line preceding the current line. Moreover, trailing '+' and '−' characters have a cumulative effect, so '−−' refers to the current line less 2.
10. To maintain compatibility with earlier versions of the editor, the character '^' in addresses is entirely equivalent to '−'.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma '**,**'. They may also be separated by a semicolon '**;**'. In the latter case, the current line '**.**' is set to the first address before the second address is interpreted. This feature can be used to determine the starting line for forward and backward searches (see items 5. and 6. in the list above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, 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.

It is generally illegal for more than one command to appear on a line. However, any command may be suffixed by 'p' or by 'l', in which case the current line is either printed or listed, respectively, as discussed below under the *p* and *l* commands.

(**.**)a
<text>

The *a*ppend command reads the given text and appends it after the addressed line. '**.**' is left at the last inserted line; or, if there were none, at the addressed line. Address '0' is legal for this command: text is placed at the beginning of the buffer.

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

The *c*hange 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 deleted.

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

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

becomes the current line.

e name

The *e*dit 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. If no file name is given, the remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *name* is remembered for possible use as a default file name in subsequent *e* or *r* or *w* or *z* commands.

f name

If *name* is given, the *f*ilename command changes the currently remembered file name to *name*; otherwise, it prints the currently remembered file name.

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

In the *g*lobal command, the first step is to mark every line that 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 a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a '\'; *a, i,* and *c* commands and associated input are permitted; the '**.**' terminating input mode may be omitted if it would be the last line of the *command list*. The (global) commands (*g, v, G,* and *V*) are *not* permitted in the *command list*.

(**.**) h

The date as returned by *date*(I) is appended after the addressed line.

(**.**) i
<text>

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

(**.** , **.**+1) j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters.

(**.**) k*x*

The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. The address form '*´x*' then addresses this line.

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

The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab, backspace*) are represented by (hopefully) mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are folded. An *l* command may also be appended to any other command.

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

The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address '0' is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file; it is an error if address *a* falls within the range of moved lines. The last line so moved becomes the current line.

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

The *p*rint command prints the addressed lines; '**.**' is left at the last line printed. The *p* command may be appended to any other command (e.g., '*dp*' deletes the current line and prints the new current line).

q

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

($) r name

The *r*ead 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 *name* is the very first file name mentioned since *ed* was invoked. 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 set to the last line read in.

(. , .) s/regular expression/replacement/ or,
(. , .) s/regular expression/replacement/g

The *s*ubstitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* 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 on which a substitution occurred.

An ampersand '&' appearing in the *replacement* is replaced by the string matching the regular expression on the current line. The special meaning of '&' in this context may be suppressed by preceding it by '\'. As a more general feature, the characters '*n*', where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified regular expression enclosed between '\(' and '\)'. When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by '\'. Such substitution cannot be done as part of a *g* or *v* command list.

(. , .) t*a*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be '0'); '**.**' is left at the last line of the copy.

u

This command reverses the effect of the last *s* command. The *u* command affects only the last line changed by the most recent *s* command.

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

This command is the same as the global command *g* except that the *command list* is executed with '**.**' initially set to every line that does *not* match the *regular expression*.

(1,$) w name
(1,$) z name

The *w*rite command writes the addressed lines onto the named file. If the file does not exist, it is created with mode 644 (readable by everyone, writable by you). The remembered file name is *not* changed unless *name* is the very first file name mentioned since *ed* was invoked. 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 *z* command is identical to *w* but, on most keyboards, the 'z' key is farther from the 'q' key than is the 'w' key.

(1,$) G/regular expression/

In the interactive *G*lobal command, the first step is to mark every line that matches the given *regular expression*. Then, for every such line, that line is printed, '**.**' is changed to that line, and any *one* command, other than a global command (*g, v, G,* and *V*), must be input. After the execution of that command, the next marked line is printed, and so on. A new-line acts as a null command; an '&' causes the re-execution of the most recent command executed within this invocation of *G*. Note that the commands input after the *G* command prints each marked line may address and affect *any* lines in the buffer. The*G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

P

The editor will prompt with a '*' for all subsequent commands. This command alternately turns the mode on and off; it is initially off.

Q

The editor exits without checking if changes have been made in the buffer since the last *w* or *z*

command.

(1,$) V/regular expression/
This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the *regular expression*.

($) =
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 the UNIX shell (*sh*(I)) to be interpreted as a command; '**.**' is unchanged.

(**.**+1) <new-line>
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 or BREAK) is sent, *ed* prints a '?' and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

Ed allows the user to include, in the first line of each text file, a specification to control the line length and the tab-to-space conversion. For example, <:t5,10,15s72:> sets tabs at columns 5, 10, and 15; it will also truncate the *printing* of all lines to a length of 72 characters and warn when a truncation has occurred. For the specification to take effect, the user's terminal must be in *echo* and −*tabs* modes (see *stty*(I)). Only the 't' and 's' parameters may be used as described in*fspec*(V). If the 's' parameter is used, all referenced lines are checked for maximum length on file read and write operations and on line print operations. Appropriate diagnostics are generated. Truncation occurs *only* on printing.

If the user attempts a *w* or *z* command and the destination file system does not have enough space available, a diagnostic is printed with an error number (i.e. ''NO SPACE: e1''). *Ed* will not perform the write. The *UNIX* command ''help e1'' (see *help*(I)) prints out a full description of what to do. *Help* should be e xecuted before leaving the editor (e.g., ''!help e1'').

FILES

/tmp/e#, temporary; '#' is the process number (in octal).

DIAGNOSTICS

'?' for errors in commands; 'TMP?' for temporary file (buffer) overflow; *help*(I) error numbers in all other cases. Commands in error should be re-entered properly. On temporary file overflow, the buffer should be written to a file and then an *e* command executed on that file. This will re-initialize the buffer; note that if the buffer overflows during the execution of a command that, in the absence of the TMP? diagnostic, would have done several changes, only some of the changes may have been done. Help error messages are self-explanatory.

SEE ALSO

A Tutorial Introduction to the UNIX Text Editor by B. W. Kernighan.
Advanced Editing on UNIX by B. W. Kernighan.

BUGS

If the *s* command succeeds on (i.e., modifies) a line that was marked by a *g, v, G,* or *V* command, then that mark is effectively removed. The editor deletes all ASCII *null* characters whenever it reads text into the buffer.