NAME

ed − text editor

SYNOPSIS

ed [−] [−x] [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* commands. If **−x** is present, an *x* command is simulated first to handle an encrypted file.

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* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer for each invocation of*ed*.

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 very 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; 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 substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character* REs match a *single* character:

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

1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. ., *, [, and \ (period, asterisk, 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* RE (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 is special at the *end* of an entire RE (see 3.2 below).

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

1.3 A period (.) is a one-character RE 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 RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE 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 right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., []a−f] matches either a right square bracket (]) or one of the letters **a** through **f** inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

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

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

2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly 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 RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression ^\(.*\)\1$ matches a line consisting of two repeated appearances of the same string.

Finally, an *entire* RE 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 RE constrains that RE to match an *initial* segment of a line.

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

The null RE standing alone (e.g., //) is equivalent to the last RE 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 each 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 RE 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 RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched.

6. A RE enclosed in question marks (?) 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 RE. If necessary, the search wraps around to the end of the buffer and continues up to and including 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 to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address − refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to −.) Moreover, trailing + and − characters have a cumulative effect, so −− refers to the current line less 2.
10. For convenience, a comma (,) stands for the address pair **1,$**, while a semicolon (;) stands for the pair **.,$**.

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, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6. 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; they 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 (except *e*, *f*, *r*, or *w*) 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: it causes the "appended" text to be placed at the beginning of the buffer.

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was 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 currently-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*, *r*, and *w* commands. If the name used in an *e* command begins with !, the rest of the line is taken to be a shell (*sh*(1)) command to be read from. Such a command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E *name*

The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

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/*RE***/***command list*

In the *g*lobal command, the first step is to mark every line that matches the given RE. 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*, *G*, *v*, and *V*) are *not* permitted in the *command list*.

(1,$)G/*RE***/**

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the global commands *q*, *G*, *v*, and *V*) may be input and is executed. 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 the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *h*elp command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous ? if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)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. Address 0 is not legal for this command.

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If only one address is given, this command does nothing.

(.)k*x*

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

(.,.)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 be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)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; . is left at the last line moved.

(.,.)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*, *f*, *r*, or *w*; for example, *dp* deletes the current line and prints the new current line.

P

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

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

Q

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

($)r *name*

The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-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. If the name used in an *r* command begins with !, the rest of the line is taken to be a shell (*sh*(1)) command to be read from. Such a command is *not* remembered as the current file name.

(.,.)s/*RE*/*replacement*/ or
(.,.)s/*RE*/*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, matched strings are replaced under control of *range*. *Range* can appear in one of four ways. If it is empty only the first occurence of the matched string is replaced. If *range* is "g", all matches are replaced. If a single number appears only the match that number from the left is replaced. If a pair of numbers separated by a "," appears, the first is a starting point and the second is a count. No error occurs if the number of matches is less than the second number. Instead of a number, a "$" may be used to refer to the last possible match on a line. 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 string matching the RE 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 RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character ^ is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The ^ loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

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

The *u*ndo command reverses the effect of the last *s* command. The *u* command affects only the last line changed by the most recent *s* command. Some commands will cause the last substitution to be forgotten and the *u*ndo command will not work.

(1,$)v/*RE*/*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 RE.

(1,$)V/*RE*/

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 RE.

(1,$)w *name*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-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 currently-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. If the name used in a *w* command begins with !, the rest of the line is taken to be a shell (*sh*(1)) command to be written to. Such a command is *not* remembered as the current file name.

x

A key string is demanded from the standard input. Later *e*, *r*, and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

($)=

The line number of the addressed line is typed; . is unchanged by this command.

!*shell command*

The remainder of the line after the ! is sent to the UNIX shell (*sh*(1)) to be interpreted as a command. Within the text of the command, the character % is replaced with the current filename and ! is replaced with the text of the previous command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(.+1)<new-line>
An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to .+1p; it is useful for stepping forward through the buffer.

The editor has a limited macro capability. Macros are defined at the beginning of *any* line, even in the append mode. The following is a correct macro definition:

\\C=anystring

Macro definitions are intercepted at the getchar() level of the editor and are recognized by the following sequence: <newline><backslash><backslash><uppercase_letter><=>. The "anystring" is any string, including one with escaped newlines. Thus, it could be a series of commands to the editor, including multiple append sequences. A macro is invoked whenever a string of the form <backslash><uppercase_letter> is seen. If the preceding sequence is preceded by a backslash, translation is turned off. Thus to get the sequence "\C" in to the editor when the "\C" macro is defined, type "\\C". Legal macro names are all upper case letters.

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.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line. Files (e.g., *a.out*) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

FILES

/tmp/e#	temporary; # is the process number.
ed.hup	work is saved here if the terminal is hung up.

DIAGNOSTICS

?	for command errors.
?*name*	for an inaccessible file.
?TMP	for overflow of temporary file.
	(use the *h*elp and *H*elp commands for more detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer into the remembered name, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *q* or *e* commands: it prints ? and allows one to continue editing. A second *q* or *e* command at this point will take effect. The − command-line option inhibits this feature.

SEE ALSO

crypt(1), grep(1), sed(1), sh(1).
A Tutorial Introduction to the UNIX Text Editor by B. W. Kernighan.
Advanced Editing on UNIX by B. W. Kernighan.

BUGS

Escaped new-lines do not work in the replacement string of a *s* command that is part of the *command list* of a *g* or a *v* command.
A *!* command cannot be subject to a *g* or a *v* command.
The sequence \n in a RE does not match any character.
The *l* command mishandles DEL.
Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.