NAME

ed, **red** - text editor

SYNOPSIS

ed [-] [**-s**] [**-p** *string*] [*file*]
red [-] [**-s**] [**-p** *string*] [*file*]

DESCRIPTION

The **ed** utility is a line-oriented text editor. It is used to create, display, modify and otherwise manipulate text files. When invoked as **red**, the editor runs in "restricted" mode, in which the only difference is that the editor restricts the use of filenames which start with '!' (interpreted as shell commands by **ed**) or contain a '/'. Note that editing outside of the current directory is only prohibited if the user does not have write access to the current directory. If a user has write access to the current directory, then symbolic links can be created in the current directory, in which case **red** will not stop the user from editing the file that the symbolic link points to.

If invoked with a *file* argument, then a copy of *file* is read into the editor's buffer. Changes are made to this copy and not directly to *file* itself. Upon quitting **ed**, any changes not explicitly saved with a *w* command are lost.

Editing is done in two distinct modes: *command* and *input*. When first invoked, **ed** is in command mode. In this mode commands are read from the standard input and executed to manipulate the contents of the editor buffer. A typical command might look like:

,s/*old*/*new*/**g**

which replaces all occurrences of the string *old* with *new*.

When an input command, such as *a* (append), *i* (insert) or *c* (change), is given, **ed** enters input mode. This is the primary means of adding text to a file. In this mode, no commands are available; instead, the standard input is written directly to the editor buffer. Lines consist of text up to and including a *newline* character. Input mode is terminated by entering a single period (.) on a line.

All **ed** commands operate on whole lines or ranges of lines; e.g., the *d* command deletes lines; the *m* command moves lines, and so on. It is possible to modify only a portion of a line by means of replacement, as in the example above. However even here, the *s* command is applied to whole lines at a time.

In general, **ed** commands consist of zero or more line addresses, followed by a single character command and possibly additional parameters; i.e., commands have the structure:

[*address*[,*address*]]*command*[*parameters*]

The address(es) indicate the line or range of lines to be affected by the command. If fewer addresses are given than the command accepts, then default addresses are supplied.

OPTIONS

The following options are available:

-s Suppress diagnostics. This should be used if **ed**'s standard input is from a script.

-p *string*
Specify a command prompt. This may be toggled on and off with the *P* command.

file Specify the name of a file to read. If *file* is prefixed with a bang (!), then it is interpreted as a shell command. In this case, what is read is the standard output of *file* executed via sh(1). To read a file whose name begins with a bang, prefix the name with a backslash (\). The default filename is set to *file* only if it is not prefixed with a bang.

LINE ADDRESSING

An address represents the number of a line in the buffer. The **ed** utility maintains a *current address* which is typically supplied to commands as the default address when none is specified. When a file is first read, the current address is set to the last line of the file. In general, the current address is set to the last line affected by a command.

A line address is constructed from one of the bases in the list below, optionally followed by a numeric offset. The offset may include any combination of digits, operators (i.e., +, - and ^) and whitespace. Addresses are read from left to right, and their values are computed relative to the current address.

One exception to the rule that addresses represent line numbers is the address *0* (zero). This means "before the first line," and is legal wherever it makes sense.

An address range is two addresses separated either by a comma or semi-colon. The value of the first address in a range cannot exceed the value of the second. If only one address is given in a range, then the second address is set to the given address. If an *n*-tuple of addresses is given where $n > 2$, then the corresponding range is determined by the last two addresses in the *n*-tuple. If only one address is expected, then the last address is used.

Each address in a comma-delimited range is interpreted relative to the current address. In a semi-colon-delimited range, the first address is used to set the current address, and the second address is interpreted relative to the first.

The following address symbols are recognized:

. The current line (address) in the buffer.

$ The last line in the buffer.

n The *n*th line in the buffer where *n* is a number in the range *[0,$]*.

- or ^ The previous line. This is equivalent to *-1* and may be repeated with cumulative effect.

-n or ^n
The *n*th previous line, where *n* is a non-negative number.

+ The next line. This is equivalent to *+1* and may be repeated with cumulative effect.

+n The *n*th next line, where *n* is a non-negative number.

, or % The first through last lines in the buffer. This is equivalent to the address range *1,$*.

; The current through last lines in the buffer. This is equivalent to the address range *.,$*.

/re/ The next line containing the regular expression *re*. The search wraps to the beginning of the buffer and continues down to the current line, if necessary. // repeats the last search.

?re? The previous line containing the regular expression *re*. The search wraps to the end of the buffer and continues up to the current line, if necessary. ?? repeats the last search.

'lc The line previously marked by a *k* (mark) command, where *lc* is a lower case letter.

REGULAR EXPRESSIONS

Regular expressions are patterns used in selecting text. For example, the command:

g/*string*/

prints all lines containing *string*. Regular expressions are also used by the *s* command for selecting old text to be replaced with new.

In addition to a specifying string literals, regular expressions can represent classes of strings. Strings thus represented are said to be matched by the corresponding regular expression. If it is possible for a regular expression to match several strings in a line, then the left-most longest match is the one selected.

The following symbols are used in constructing regular expressions:

c Any character *c* not listed below, including ‘{’, ‘}’, ‘(’, ‘)’, ‘<’ and ‘>’, matches itself.

\c Any backslash-escaped character *c*, except for ‘{’, ‘}’, ‘(’, ‘)’, ‘<’ and ‘>’, matches itself.

. Match any single character.

[char-class]
Match any single character in *char-class*. To include a ‘]’ in *char-class*, it must be the first character. A range of characters may be specified by separating the end characters of the range with a ‘-’, e.g., ‘a-z’ specifies the lower case characters. The following literal expressions can also be used in *char-class* to specify sets of characters:

[:alnum:] [:cntrl:] [:lower:] [:space:]
[:alpha:] [:digit:] [:print:] [:upper:]
[:blank:] [:graph:] [:punct:] [:xdigit:]

If ‘-’ appears as the first or last character of *char-class*, then it matches itself. All other characters in *char-class* match themselves.

Patterns in *char-class* of the form:

[.*col-elm*.] or,
[=*col-elm*=]

where *col-elm* is a *collating element* are interpreted according to the current locale settings (not currently supported). See regex(3) and re_format(7) for an explanation of these constructs.

[^char-class]
Match any single character, other than newline, not in *char-class*. *Char-class* is defined as above.

^ If ^ is the first character of a regular expression, then it anchors the regular expression to the beginning of a line. Otherwise, it matches itself.

$ If *$* is the last character of a regular expression, it anchors the regular expression to the end of a line. Otherwise, it matches itself.

\< Anchor the single character regular expression or subexpression immediately following it to the beginning of a word. (This may not be available)

\> Anchor the single character regular expression or subexpression immediately following it to the end of a word. (This may not be available)

\(re\) Define a subexpression *re*. Subexpressions may be nested. A subsequent backreference of the form *n*, where *n* is a number in the range [1,9], expands to the text matched by the *n*th subexpression. For example, the regular expression '\(.*\)\1' matches any string consisting of identical adjacent substrings. Subexpressions are ordered relative to their left delimiter.

* Match the single character regular expression or subexpression immediately preceding it zero or more times. If * is the first character of a regular expression or subexpression, then it matches itself. The * operator sometimes yields unexpected results. For example, the regular expression 'b*' matches the beginning of the string 'abbb' (as opposed to the substring 'bbb'), since a null match is the only left-most match.

\{n,m\} or \{n,\} or \{n\}
Match the single character regular expression or subexpression immediately preceding it at least *n* and at most *m* times. If *m* is omitted, then it matches at least *n* times. If the comma is also omitted, then it matches exactly *n* times.

Additional regular expression operators may be defined depending on the particular regex(3) implementation.

COMMANDS

All **ed** commands are single characters, though some require additional parameters. If a command's parameters extend over several lines, then each line except for the last must be terminated with a backslash (\).

In general, at most one command is allowed per line. However, most commands accept a print suffix, which is any of *p* (print), *l* (list), or *n* (enumerate), to print the last line affected by the command.

An interrupt (typically ^C) has the effect of aborting the current command and returning the editor to command mode.

The **ed** utility recognizes the following commands. The commands are shown together with the default address or address range supplied if none is specified (in parenthesis).

(.)a Append text to the buffer after the addressed line. Text is entered in input mode. The current address is set to last line entered.

(.,.)c Change lines in the buffer. The addressed lines are deleted from the buffer, and text is appended

in their place. Text is entered in input mode. The current address is set to last line entered.

(.,.)d Delete the addressed lines from the buffer. If there is a line after the deleted range, then the current address is set to this line. Otherwise the current address is set to the line before the deleted range.

e *file* Edit *file*, and sets the default filename. If *file* is not specified, then the default filename is used. Any lines in the buffer are deleted before the new file is read. The current address is set to the last line read.

e *!command*

Edit the standard output of *!command*, (see *!command* below). The default filename is unchanged. Any lines in the buffer are deleted before the output of *command* is read. The current address is set to the last line read.

E *file* Edit *file* unconditionally. This is similar to the *e* command, except that unwritten changes are discarded without warning. The current address is set to the last line read.

f *file* Set the default filename to *file*. If *file* is not specified, then the default unescaped filename is printed.

(1,$)g/re/command-list

Apply *command-list* to each of the addressed lines matching a regular expression *re*. The current address is set to the line currently matched before *command-list* is executed. At the end of the *g* command, the current address is set to the last line affected by *command-list*.

Each command in *command-list* must be on a separate line, and every line except for the last must be terminated by a backslash (\). Any commands are allowed, except for *g*, *G*, *v*, and *V*. A newline alone in *command-list* is equivalent to a *p* command.

(1,$)G/re/

Interactively edit the addressed lines matching a regular expression *re*. For each matching line, the line is printed, the current address is set, and the user is prompted to enter a *command-list*. At the end of the *G* command, the current address is set to the last line affected by (the last) *command-list*.

The format of *command-list* is the same as that of the *g* command. A newline alone acts as a null command list. A single ‘&’ repeats the last non-null command list.

H Toggle the printing of error explanations. By default, explanations are not printed. It is

recommended that ed scripts begin with this command to aid in debugging.

h Print an explanation of the last error.

(.)i Insert text in the buffer before the current line. Text is entered in input mode. The current address is set to the last line entered.

(.,.+1)j
Join the addressed lines. The addressed lines are deleted from the buffer and replaced by a single line containing their joined text. The current address is set to the resultant line.

(.)klc Mark a line with a lower case letter *lc*. The line can then be addressed as *'lc* (i.e., a single quote followed by *lc*) in subsequent commands. The mark is not cleared until the line is deleted or otherwise modified.

(.,.)l Print the addressed lines unambiguously. If a single line fills more than one screen (as might be the case when viewing a binary file, for instance), a "--More--" prompt is printed on the last line. The **ed** utility waits until the RETURN key is pressed before displaying the next screen. The current address is set to the last line printed.

(.,.)m(.)
Move lines in the buffer. The addressed lines are moved to after the right-hand destination address, which may be the address *0* (zero). The current address is set to the last line moved.

(.,.)n Print the addressed lines along with their line numbers. The current address is set to the last line printed.

(.,.)p Print the addressed lines. The current address is set to the last line printed.

P Toggle the command prompt on and off. Unless a prompt was specified by with command-line option **-p** *string*, the command prompt is by default turned off.

q Quit **ed**.

Q Quit **ed** unconditionally. This is similar to the *q* command, except that unwritten changes are discarded without warning.

($)r *file*
Read *file* to after the addressed line. If *file* is not specified, then the default filename is used. If there was no default filename prior to the command, then the default filename is set to *file*.

Otherwise, the default filename is unchanged. The current address is set to the last line read.

($)r *!command*

Read to after the addressed line the standard output of *!command*, (see the *!command* below). The default filename is unchanged. The current address is set to the last line read.

(.,.)s/re/replacement/

(.,.)s/re/replacement/g

(.,.)s/re/replacement/n

Replace text in the addressed lines matching a regular expression *re* with *replacement*. By default, only the first match in each line is replaced. If the *g* (global) suffix is given, then every match to be replaced. The *n* suffix, where *n* is a positive number, causes only the *n*th match to be replaced. It is an error if no substitutions are performed on any of the addressed lines. The current address is set the last line affected.

Re and *replacement* may be delimited by any character other than space and newline (see the *s* command below). If one or two of the last delimiters is omitted, then the last line affected is printed as though the print suffix *p* were specified.

An unescaped '&' in *replacement* is replaced by the currently matched text. The character sequence *\m*, where *m* is a number in the range [1,9], is replaced by the *m th* backreference expression of the matched text. If *replacement* consists of a single '%', then *replacement* from the last substitution is used. Newlines may be embedded in *replacement* if they are escaped with a backslash (\).

(.,.)s Repeat the last substitution. This form of the *s* command accepts a count suffix *n*, or any combination of the characters *r*, *g*, and *p*. If a count suffix *n* is given, then only the *n*th match is replaced. The *r* suffix causes the regular expression of the last search to be used instead of the that of the last substitution. The *g* suffix toggles the global suffix of the last substitution. The *p* suffix toggles the print suffix of the last substitution The current address is set to the last line affected.

(.,.)t(.)

Copy (i.e., transfer) the addressed lines to after the right-hand destination address, which may be the address *0* (zero). The current address is set to the last line copied.

u Undo the last command and restores the current address to what it was before the command. The global commands *g*, *G*, *v*, and *V*. are treated as a single command by undo. *u* is its own inverse.

(1,$)v/re/command-list

Apply *command-list* to each of the addressed lines not matching a regular expression *re*. This is similar to the *g* command.

(1,$)V/re/

Interactively edit the addressed lines not matching a regular expression *re*. This is similar to the *G* command.

(1,$)w *file*

Write the addressed lines to *file*. Any previous contents of *file* is lost without warning. If there is no default filename, then the default filename is set to *file*, otherwise it is unchanged. If no filename is specified, then the default filename is used. The current address is unchanged.

(1,$)wq *file*

Write the addressed lines to *file*, and then executes a *q* command.

(1,$)w *!command*

Write the addressed lines to the standard input of *!command*, (see the *!command* below). The default filename and current address are unchanged.

(1,$)W *file*

Append the addressed lines to the end of *file*. This is similar to the *w* command, expect that the previous contents of file is not clobbered. The current address is unchanged.

(.+1)zn

Scroll *n* lines at a time starting at addressed line. If *n* is not specified, then the current window size is used. The current address is set to the last line printed.

!command

Execute *command* via sh(1). If the first character of *command* is '!', then it is replaced by text of the previous *!command*. The **ed** utility does not process *command* for backslash (\) escapes. However, an unescaped *%* is replaced by the default filename. When the shell returns from execution, a '!' is printed to the standard output. The current line is unchanged.

($)= Print the line number of the addressed line.

(.+1)newline

Print the addressed line, and sets the current address to that line.

FILES

*/tmp/ed.** buffer file
ed.hup the file to which **ed** attempts to write the buffer if the terminal hangs up

DIAGNOSTICS

When an error occurs, **ed** prints a '?' and either returns to command mode or exits if its input is from a script. An explanation of the last error can be printed with the *h* (help) command.

Since the *g* (global) command masks any errors from failed searches and substitutions, it can be used to perform conditional operations in scripts; e.g.,

g/*old*/**s**//*new*/

replaces any occurrences of *old* with *new*. If the *u* (undo) command occurs in a global command list, then the command list is executed only once.

If diagnostics are not disabled, attempting to quit **ed** or edit another file before writing a modified buffer results in an error. If the command is entered a second time, it succeeds, but any changes to the buffer are lost.

SEE ALSO

sed(1), sh(1), vi(1), regex(3)

USD:12-13

B. W. Kernighan and P. J. Plauger, *Software Tools in Pascal*, 1981, Addison-Wesley.

B. W. Kernighan, *A Tutorial Introduction to the UNIX Text Editor*.

B. W. Kernighan, *Advanced Editing on UNIX*.

LIMITATIONS

The **ed** utility processes *file* arguments for backslash escapes, i.e., in a filename, any characters preceded by a backslash (\) are interpreted literally.

If a text (non-binary) file is not terminated by a newline character, then **ed** appends one on reading/writing it. In the case of a binary file, **ed** does not append a newline on reading/writing.

per line overhead: 4 ints

HISTORY

An **ed** command appeared in Version 1 AT&T UNIX.

BUGS

The **ed** utility does not recognize multibyte characters.