NAME

ed, red – text editor

SYNOPSIS

ed [-GVhs] [-p *string*] [*file*]

red [-GVhs] [-p *string*] [*file*]

DESCRIPTION

ed is a line-oriented text editor. It is used to create, display, modify and otherwise manipulate text files. **red** is a restricted **ed**: it can only edit files in the current directory and cannot execute shell commands.

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

-G Forces backwards compatibility. Affects the commands *'G'*, *'V'*, *'f'*, *'l'*, *'m'*, *'t'*, and *'!!'*.

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

−p *string*
Specifies a command prompt. This may be toggled on and off with the *'P'* command.

file Specifies 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. **ed** 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 semicolon. The value of the first address in a range cannot exceed the value of the 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 semicolon-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,$]*.

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

ˆ*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.

whitespace n

+*n* The *n*th next line, where *n* is a non-negative number. *Whitespace* followed by a number *n* is interpreted as +*n*.

, 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 **ed** 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 A backslash-escaped character *c* other than '{', '}', '(', ')', '<', '>', 'b', 'B', 'w', 'W', '+', and '?' matches itself.

. Matches any single character.

[char-class]
Matches 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 *locale*(5) (not currently supported). See *regex*(3) for an explanation of these constructs.

[^*char-class*]
Matches 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.

\(*re*\) Defines a (possibly null) 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 '\(a.c\)\1' matches the string 'abcabc', but not 'abcadc'. Subexpressions are ordered relative to their left delimiter.

* Matches 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*\}
\{*n,*\}
\{*n*\} Matches 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. If any of these forms occurs first in a regular expression or subexpression, then it is interpreted literally (i.e., the regular expression '\{2\}' matches the string '{2}', and so on).

\<
\> Anchors the single character regular expression or subexpression immediately following it to the beginning (\<) or ending (\>) of a *word*, i.e., in ASCII, a maximal string of alphanumeric characters, including the underscore (_).

The following extended operators are preceded by a backslash (\) to distinguish them from traditional **ed** syntax.

\‘
\’ Unconditionally matches the beginning (\‘) or ending (\’) of a line.

\? Optionally matches the single character regular expression or subexpression immediately preceding it. For example, the regular expression 'a[bd]\?c' matches the strings 'abc', 'adc' and 'ac'. If \? occurs at the beginning of a regular expressions or subexpression, then it matches a literal '?'.

\+ Matches the single character regular expression or subexpression immediately preceding it one or more times. So the regular expression 'a\+' is shorthand for 'aa*'. If \+ occurs at the beginning of a regular expression or subexpression, then it matches a literal '+'.

\b Matches the beginning or ending (null string) of a word. Thus the regular expression '\bhello\b' is equivalent to '\<hello\>'. However, '\b\b' is a valid regular expression whereas '\<\>' is not.

\B Matches (a null string) inside a word.

\w Matches any character in a word.

\W Matches any character not in a word.

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.

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

(.)a Appends text to the buffer after the addressed line, which may be the address 0 (zero). Text is entered in input mode. The current address is set to last line entered.

(.,.)c Changes 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 Deletes 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* Edits *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*
Edits 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* Edits *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* Sets the default filename to *file*. If *file* is not specified, then the default unescaped filename is printed.

(1,$)g*/re/command-list*
Applies *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 edits 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 Toggles 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 Prints an explanation of the last error.

(.)i Inserts 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 Joins 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.

(.)k*lc* Marks 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 Prints the addressed lines unambiguously. If invoked from a terminal, **ed** pauses at the end of each page until a newline is entered. The current address is set to the last line printed.

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

(.,.)p Prints the addressed lines. If invoked from a terminal, **ed** pauses at the end of each page until a newline is entered. The current address is set to the last line printed.

P Toggles 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 Quits ed.

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

($)r *file* Reads *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*
Reads 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*
Replaces 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 Repeats 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(.) Copies (i.e., transfers) 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 Undoes 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*
Applies *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 edits the addressed lines not matching a regular expression *re.* This is similar to the *'G'* command.

(1,$)w *file*
Writes 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*
Writes the addressed lines to *file*, and then executes a *'q'* command.

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

(1,$)W *file*
Appends 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.

(.)x Copies (puts) the contents of the cut buffer to after the addressed line. The current address is set to the last line copied.

(.,.)y Copies (yanks) the addressed lines to the cut buffer. The cut buffer is overwritten by subsequent *'y'*, *'s'*, *'j'*, *'d'*, or *'c'* commands. The current address is unchanged.

(.+1)z*n* Scrolls *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*
Executes *command* via *sh*(1). If the first character of *command* is '!', then it is replaced by text of the previous *'!command'*. **ed** 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.

(.,.)# Begins a comment; the rest of the line, up to a newline, is ignored. If a line address followed by a semicolon is given, then the current address is set to that address. Otherwise, the current address is unchanged.

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

(.+1)newline
Prints the addressed line, and sets the current address to that line.

FILES

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

SEE ALSO

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

USD:12-13

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

LIMITATIONS

ed 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

DIAGNOSTICS

When an error occurs, if **ed**'s input is from a regular file or here document, then it exits, otherwise it prints a '?' and returns to command mode. An explanation of the last error can be printed with the *'h'* (help) command.

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.

ed exits with 0 if no errors occurred; otherwise >0.

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ed – edit text

SYNOPSIS

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

DESCRIPTION

The *ed* utility is a line-oriented text editor that uses two modes: *command mode* and *input mode*. In command mode the input characters shall be interpreted as commands, and in input mode they shall be interpreted as text. See the EXTENDED DESCRIPTION section.

OPTIONS

The *ed* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

-p *string*
Use *string* as the prompt string when in command mode. By default, there shall be no prompt string.

-s Suppress the writing of byte counts by **e**, **E**, **r**, and **w** commands and of the **'!'** prompt after a !*command*.

OPERANDS

The following operand shall be supported:

file If the *file* argument is given, *ed* shall simulate an **e** command on the file named by the pathname, *file*, before accepting commands from the standard input. If the *file* operand is **'-'**, the results are unspecified.

STDIN

The standard input shall be a text file consisting of commands, as described in the EXTENDED DESCRIPTION section.

INPUT FILES

The input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ed*:

HOME Determine the pathname of the user's home directory.

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL
If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE

Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH
Determine the location of message catalogs for the processing of *LC_MESSAGES* .

ASYNCHRONOUS EVENTS

The *ed* utility shall take the standard action for all signals (see the ASYNCHRONOUS EVENTS section in *Utility Description Defaults*) with the following exceptions:

SIGINT
The *ed* utility shall interrupt its current activity, write the string **"?\n"** to standard output, and return to command mode (see the EXTENDED DESCRIPTION section).

SIGHUP
If the buffer is not empty and has changed since the last write, the *ed* utility shall attempt to write a copy of the buffer in a file. First, the file named **ed.hup** in the current directory shall be used; if that fails, the file named **ed.hup** in the directory named by the *HOME* environment variable shall be used. In any case, the *ed* utility shall exit without returning to command mode.

SIGQUIT
The *ed* utility shall ignore this event.

STDOUT

Various editing commands and the prompting feature (see **-p**) write to standard output, as described in the EXTENDED DESCRIPTION section.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The output files shall be text files whose formats are dependent on the editing commands given.

EXTENDED DESCRIPTION

The *ed* utility shall operate on a copy of the file it is editing; changes made to the copy shall have no effect on the file until a **w** (write) command is given. The copy of the text is called the *buffer*.

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 very often can be omitted. If the **-p** option is specified, the prompt string shall be written to standard output before each command is read.

In general, only one command can appear on a line. Certain commands allow text to be input. 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 shall be recognized; all input is merely collected. Input mode is terminated by entering a line consisting of two characters: a period (**'.'**) followed by a <newline>. This line is not considered part of the input text.

Regular Expressions in ed

The *ed* utility shall support basic regular expressions, as described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.3, Basic Regular Expressions. Since regular expressions in *ed* are always matched against single lines (excluding the terminating <newline>s), never against any larger section of text, there is no way for a regular expression to match a <newline>.

A null RE shall be equivalent to the last RE encountered.

Regular expressions are used in addresses to specify lines, and in some commands (for example, the **s** substitute command) to specify portions of a line to be substituted.

Addresses in ed

Addressing in *ed* relates to the current line. Generally, the current line is the last line affected by a command. The current line number is the address of the current line. If the edit buffer is not empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

Addresses shall be constructed as follows:

1. The period character (**'.'**) shall address the current line.

2. The dollar sign character (**'$'**) shall address the last line of the edit buffer.

3. The positive decimal number *n* shall address the *n*th line of the edit buffer.

4. The apostrophe-x character pair (**"'x"**) shall address the line marked with the mark name character *x*, which shall be a lowercase letter from the portable character set. It shall be an error if the character has not been set to mark a line or if the line that was marked is not currently present in the edit buffer.

5. A BRE enclosed by slash characters (**'/'**) shall address the first line found by searching forwards from the line following the current line toward the end of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of slash characters shall address the next line for which the line excluding the terminating <newline> matches the last BRE encountered. In addition, the second slash can be omitted at the end of a command line. Within the BRE, a backslash-slash pair (**"\/"**) shall represent a literal slash instead of the BRE delimiter. If necessary, the search shall wrap around to the beginning of the buffer and continue up to and including the current line, so that the entire buffer is searched.

6. A BRE enclosed by question-mark characters (**'?'**) shall address the first line found by searching backwards from the line preceding the current line toward the beginning of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of question-mark characters (**"??"**) shall address the previous line for which the line excluding the terminating <newline> matches the last BRE encountered. In addition, the second question-mark can be omitted at the end of a command line. Within the BRE, a backslash-question-mark pair (**"\?"**) shall represent a literal question mark instead of the BRE delimiter. If necessary, the search shall wrap around to the end of the buffer and continue up to and including the current line, so that the entire buffer is searched.

7. A plus-sign (**'+'**) or hyphen character (**'-'**) followed by a decimal number shall address the current line plus or minus the number. A plus-sign or hyphen character not followed by a decimal number shall address the current line plus or minus 1.

Addresses can be followed by zero or more address offsets, optionally <blank>-separated. Address offsets are constructed as follows:

* A plus-sign or hyphen character followed by a decimal number shall add or subtract, respectively, the indicated number of lines to or from the address. A plus-sign or hyphen character not followed by a decimal number shall add or subtract 1 to or from the address.

* A decimal number shall add the indicated number of lines to the address.

It shall not be an error for an intermediate address value to be less than zero or greater than the last line in the edit buffer. It shall be an error for the final address value to be less than zero or greater than the last line in the edit buffer. It shall be an error if a search for a BRE fails to find a matching line.

Commands accept zero, one, or two addresses. If more than the required number of addresses are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than the required number of addresses are provided to a command, the addresses specified first shall be evaluated and then discarded until the maximum number of valid addresses remain, for the specified command.

Addresses shall be separated from each other by a comma (**','**) or semicolon character (**';'**). In the case of a semicolon separator, the current line (**'.'**) shall be set to the first address, and only then will the second address be calculated. This feature can be used to determine the starting line for forwards and backwards searches; see rules 5. and 6.

Addresses can be omitted on either side of the comma or semicolon separator, in which case the resulting address pairs shall be as follows:

Specified	Resulting
,	1 , $,
addr	1 , addr
addr ,	addr , addr
;	. ; $
; addr	. ; addr
addr ;	addr ; addr

Any <blank>s included between addresses, address separators, or address offsets shall be ignored.

Commands in ed

In the following list of *ed* commands, the default addresses are shown in parentheses. The number of addresses shown in the default shall be the number expected by the command. The parentheses are not part of the address; they show that the given addresses are the default.

It is generally invalid for more than one command to appear on a line. However, any command (except **e**, **E**, **f**, **q**, **Q**, **r**, **w**, and **!**) can be suffixed by the letter **l**, **n**, or **p**; in which case, except for the **l**, **n**, and **p** commands, the command shall be executed and then the new current line shall be written as described below under the **l**, **n**, and **p** commands. When an **l**, **n**, or **p** suffix is used with an **l**, **n**, or **p** command, the command shall write to standard output as described below, but it is unspecified whether the suffix writes the current line again in the requested format or whether the suffix has no effect. For example, the **pl** command (base **p** command with an **l** suffix) shall either write just the current line or write it twice-once as specified for **p** and once as specified for **l**. Also, the **g**, **G**, **v**, and **V** commands shall take a command as a parameter.

Each address component can be preceded by zero or more <blank>s. The command letter can be preceded by zero or more <blank>s. If a suffix letter (**l**, **n**, or **p**) is given, the application shall ensure that it immediately follows the command.

The **e**, **E**, **f**, **r**, and **w** commands shall take an optional *file* parameter, separated from the command letter by one or more <blank>s.

If changes have been made in the buffer since the last **w** command that wrote the entire buffer, *ed* shall warn the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands. The *ed* utility shall write the string:

"?\n"

(followed by an explanatory message if *help mode* has been enabled via the **H** command) to standard output and shall continue in command mode with the current line number unchanged. If the **e** or **q** command is repeated with no intervening command, it shall take effect.

If a terminal disconnect is detected:

* If the buffer is not empty and has changed since the last write, the *ed* utility shall attempt to write a copy of the buffer to a file named **ed.hup** in the current directory. If this write fails, *ed* shall attempt to write a copy of the buffer to a filename **ed.hup** in the directory named by the *HOME* environment variable. If both these attempts fail, *ed* shall exit without saving the buffer.

* The *ed* utility shall not write the file to the currently remembered pathname or return to command mode, and shall terminate with a non-zero exit status.

If an end-of-file is detected on standard input:

* If the *ed* utility is in input mode, *ed* shall terminate input mode and return to command mode. It is unspecified if any partially entered lines (that is, input text without a terminating <newline>) are discarded from the input text.

* If the *ed* utility is in command mode, it shall act as if a **q** command had been entered.

If the closing delimiter of an RE or of a replacement string (for example, **'/'**) in a **g**, **G**, **s**, **v**, or **V** command would be the last character before a <newline>, that delimiter can be omitted, in which case the addressed line shall be written. For example, the following pairs of commands are equivalent:

s/s1/s2 s/s1/s2/p
g/s1 g/s1/p
?s1 ?s1?

If an invalid command is entered, *ed* shall write the string:

"?\n"

(followed by an explanatory message if *help mode* has been enabled via the **H** command) to standard output and shall continue in command mode with the current line number unchanged.

Append Command

Synopsis:

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

The **a** command shall read the given text and append it after the addressed line; the current line number shall become the address of the last inserted line or, if there were none, the addressed line. Address 0 shall be valid for this command; it shall cause the appended text to be placed at the beginning of the buffer.

Change Command

Synopsis:

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

The **c** command shall delete the addressed lines, then accept input text that replaces these lines; the current line shall be set to the address of the last line input; or, if there were none, at the line after the last line deleted; if the lines deleted were originally at the end of the buffer, the current line number shall be set to the address of the new last line; if no lines remain in the buffer, the current line number shall be set to zero. Address 0 shall be valid for this command; it shall be interpreted as if address 1 were specified.

Delete Command

Synopsis:

(.,.)d

The **d** command shall delete the addressed lines from the buffer. The address of the line after the last line deleted shall become the current line number; if the lines deleted were originally at the end of the buffer, the current line number shall be set to the address of the new last line; if no lines remain in the buffer, the current line number shall be set to zero.

Edit Command

Synopsis:

e [*file***]**

The **e** command shall delete the entire contents of the buffer and then read in the file named by the pathname *file*. The current line number shall be set to the address of the last line of the buffer. If no pathname is given, the currently remembered pathname, if any, shall be used (see the **f** command). The number of bytes read shall be written to standard output, unless the **-s** option was specified, in the

following format:

"%d\n", <*number of bytes read***>**

The name *file* shall be remembered for possible use as a default pathname in subsequent **e**, **E**, **r**, and **w** commands. If *file* is replaced by **'!'**, the rest of the line shall be taken to be a shell command line whose output is to be read. Such a shell command line shall not be remembered as the current *file*. All marks shall be discarded upon the completion of a successful **e** command. If the buffer has changed since the last time the entire buffer was written, the user shall be warned, as described previously.

Edit Without Checking Command

Synopsis:

E [*file***]**

The **E** command shall possess all properties and restrictions of the **e** command except that the editor shall not check to see whether any changes have been made to the buffer since the last **w** command.

Filename Command

Synopsis:

f [*file***]**

If *file* is given, the **f** command shall change the currently remembered pathname to *file*; whether the name is changed or not, it shall then write the (possibly new) currently remembered pathname to the standard output in the following format:

"%s\n", <*pathname***>**

The current line number shall be unchanged.

Global Command

Synopsis:

(1,$)g/*RE***/***command list*

In the **g** command, the first step shall be to mark every line for which the line excluding the terminating <newline> matches the given RE. Then, going sequentially from the beginning of the file to the end of the file, the given *command list* shall be executed for each marked line, with the current line number set to the address of that line. Any line modified by the *command list* shall be unmarked. When the **g** command completes, the current line number shall have the value assigned by the last command in the *command list*. If there were no matching lines, the current line number shall not be changed. A single command or the first of a list of commands shall appear on the same line as the global command. All lines of a multi-line list except the last line shall be ended with a backslash preceding the terminating <newline>; the **a**, **i**, and **c** commands and associated input are permitted. The **'.'** terminating input mode can be omitted if it would be the last line of the *command list*. An empty *command list* shall be equivalent to the **p** command. The use of the **g**, **G**, **v**, **V**, and **!** commands in the *command list* produces undefined results. Any character other than <space> or <newline> can be used instead of a slash to delimit the RE. Within the RE, the RE delimiter itself can be used as a literal character if it is preceded by a backslash.

Interactive Global Command

Synopsis:

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

In the **G** command, the first step shall be to mark every line for which the line excluding the terminating <newline> matches the given RE. Then, for every such line, that line shall be written, the current line number shall be set to the address of that line, and any one command (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) shall be read and executed. A <newline> shall act as a null command (causing no action to be taken on the current line); an **'&'** shall cause the re-execution of the most recent non-null command executed within the current invocation of **G**. Note that the commands input as part of the execution of the **G** command can address and affect any lines in the buffer. Any line modified by the command shall be unmarked. The final value of the current line number shall be the value set by the last command successfully executed. (Note that the last command successfully executed shall be the **G** command itself if a command fails or the null command is specified.) If there were no matching lines, the current line number shall not be changed. The **G** command can be terminated by a SIGINT signal. Any character other than <space> or <newline> can be used instead of a slash to delimit the RE and the replacement. Within the RE, the RE delimiter itself can be used as a literal character if it is preceded by a backslash.

Help Command

Synopsis:

h

The **h** command shall write a short message to standard output that explains the reason for the most recent **'?'** notification. The current line number shall be unchanged.

Help-Mode Command

Synopsis:

H

The **H** command shall cause *ed* to enter a mode in which help messages (see the **h** command) shall be written to standard output for all subsequent **'?'** notifications. The **H** command alternately shall turn this mode on and off; it is initially off. If the help-mode is being turned on, the **H** command also explains the previous **'?'** notification, if there was one. The current line number shall be unchanged.

Insert Command

Synopsis:

(.)i
<*text*>
.

The **i** command shall insert the given text before the addressed line; the current line is set to the last inserted line or, if there was none, to the addressed line. This command differs from the **a** command only in the placement of the input text. Address 0 shall be valid for this command; it shall be interpreted as if address 1 were specified.

Join Command

Synopsis:

(.,.+1)j

The **j** command shall join contiguous lines by removing the appropriate <newline>s. If exactly one address is given, this command shall do nothing. If lines are joined, the current line number shall be set to the address of the joined line; otherwise, the current line number shall be unchanged.

Mark Command

Synopsis:

(.)k*x*

The **k** command shall mark the addressed line with name *x*, which the application shall ensure is a lowercase letter from the portable character set. The address **"'x"** shall then refer to this line; the current line number shall be unchanged.

List Command

Synopsis:

(.,.)l

The **l** command shall write to standard output the addressed lines in a visually unambiguous form. The characters listed in the Base Definitions volume of IEEE Std 1003.1-2001, Table 5-1, Escape Sequences and Associated Actions (**'\\'**, **'\a'**, **'\b'**, **'\f'**, **'\r'**, **'\t'**, **'\v'**) shall be written as the corresponding escape sequence; the **'\n'** in that table is not applicable. Non-printable characters not in the table shall be written as one three-digit octal number (with a preceding backslash character) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than nine bits, the format used for non-printable characters is implementation-defined.

Long lines shall be folded, with the point of folding indicated by <newline> preceded by a backslash; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line shall be marked with a **'$'**, and **'$'** characters within the text shall be written with a preceding backslash. An **l** command can be appended to any other command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**. The current line number shall be set to the address of the last line written.

Move Command

Synopsis:

(.,.)m*address*

The **m** command shall reposition the addressed lines after the line addressed by *address*. Address 0 shall be valid for *address* and cause the addressed lines to be moved to the beginning of the buffer. It shall be an error if address *address* falls within the range of moved lines. The current line number shall be set to the address of the last line moved.

Number Command

Synopsis:

(.,.)n

The **n** command shall write to standard output the addressed lines, preceding each line by its line number and a <tab>; the current line number shall be set to the address of the last line written. The **n** command can be appended to any command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

Print Command

Synopsis:

(.,.)p

The **p** command shall write to standard output the addressed lines; the current line number shall be set to the address of the last line written. The **p** command can be appended to any command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

Prompt Command

Synopsis:

P

The **P** command shall cause *ed* to prompt with an asterisk (**'*'**) (or *string*, if **-p** is specified) for all subsequent commands. The **P** command alternatively shall turn this mode on and off; it shall be initially on if the **-p** option is specified; otherwise, off. The current line number shall be unchanged.

Quit Command

Synopsis:

q

The **q** command shall cause *ed* to exit. If the buffer has changed since the last time the entire buffer was written, the user shall be warned, as described previously.

Quit Without Checking Command

Synopsis:

Q

The **Q** command shall cause *ed* to exit without checking whether changes have been made in the buffer since the last **w** command.

Read Command

Synopsis:

($)r [*file***]**

The **r** command shall read in the file named by the pathname *file* and append it after the addressed line. If no *file* argument is given, the currently remembered pathname, if any, shall be used (see the **e** and **f** commands). The currently remembered pathname shall not be changed unless there is no remembered pathname. Address 0 shall be valid for **r** and shall cause the file to be read at the beginning of the buffer. If the read is successful, and **-s** was not specified, the number of bytes read shall be written to standard output in the following format:

"%d\n", <*number of bytes read***>**

The current line number shall be set to the address of the last line read in. If *file* is replaced by **'!'**, the rest of the line shall be taken to be a shell command line whose output is to be read. Such a shell command line shall not be remembered as the current pathname.

Substitute Command

Synopsis:

(.,.)s/*RE***/***replacement***/***flags*

The **s** command shall search each addressed line for an occurrence of the specified RE and replace either the first or all (non-overlapped) matched strings with the *replacement*; see the following description of the **g** suffix. It is an error if the substitution fails on every addressed line. Any character other than <space> or <newline> can be used instead of a slash to delimit the RE and the replacement. Within the RE, the RE delimiter itself can be used as a literal character if it is preceded by a backslash. The current line shall be set to the address of the last line on which a substitution occurred.

An ampersand (**'&'**) appearing in the *replacement* shall be replaced by the string matching the RE on the current line. The special meaning of **'&'** in this context can be suppressed by preceding it by backslash. As a more general feature, the characters **'\n'**, where *n* is a digit, shall be replaced by the text matched by the corresponding back-reference expression. When the character **'%'** is the only character in the *replacement*, the *replacement* used in the most recent substitute command shall be used as the *replacement* in the current substitute command; if there was no previous substitute command, the use of **'%'** in this manner shall be an error. The **'%'** shall lose its special meaning when it is in a replacement string of more than one character or is preceded by a backslash. For each backslash (**'\'**) encountered in scanning *replacement* from beginning to end, the following character shall lose its special meaning (if any). It is unspecified what special meaning is given to any character other than **'&'**, **'\'**, **'%'**, or digits.

A line can be split by substituting a <newline> into it. The application shall ensure it escapes the <newline> in the *replacement* by preceding it by backslash. Such substitution cannot be done as part of a **g** or **v** *command list*. The current line number shall be set to the address of the last line on which a substitution is performed. If no substitution is performed, the current line number shall be unchanged. If a line is split, a substitution shall be considered to have been performed on each of the new lines for the purpose of determining the new current line number. A substitution shall be considered to have been performed even if the replacement string is identical to the string that it replaces.

The application shall ensure that the value of *flags* is zero or more of:

count Substitute for the *count*th occurrence only of the RE found on each addressed line.

g Globally substitute for all non-overlapping instances of the RE rather than just the first one. If both **g** and *count* are specified, the results are unspecified.

l Write to standard output the final line in which a substitution was made. The line shall be written in the format specified for the **l** command.

n Write to standard output the final line in which a substitution was made. The line shall be written in the format specified for the **n** command.

p Write to standard output the final line in which a substitution was made. The line shall be written in the format specified for the **p** command.

Copy Command

Synopsis:

(.,.)t*address*

The **t** command shall be equivalent to the **m** command, except that a copy of the addressed lines shall be placed after address *address* (which can be 0); the current line number shall be set to the address of the last line added.

Undo Command

Synopsis:

u

The **u** command shall nullify the effect of the most recent command that modified anything in the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, **G**, or **V** command. All changes made to the buffer by a **g**, **G**, **v**, or **V** global command shall be undone as a single change; if no changes were made by the global command (such as with **g**/RE/ **p**), the **u** command shall have no effect. The current line number shall be set to the value it had immediately before the command being undone started.

Global Non-Matched Command

Synopsis:

(1,$)v/*RE***/***command list*

This command shall be equivalent to the global command **g** except that the lines that are marked during the first step shall be those for which the line excluding the terminating <newline> does not match the RE.

Interactive Global Not-Matched Command

Synopsis:

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

This command shall be equivalent to the interactive global command **G** except that the lines that are marked during the first step shall be those for which the line excluding the terminating <newline> does not match the RE.

Write Command

Synopsis:

(1,$)w [*file***]**

The **w** command shall write the addressed lines into the file named by the pathname *file*. The command shall create the file, if it does not exist, or shall replace the contents of the existing file. The currently remembered pathname shall not be changed unless there is no remembered pathname. If no pathname is given, the currently remembered pathname, if any, shall be used (see the **e** and **f** commands); the current line number shall be unchanged. If the command is successful, the number of bytes written shall be written to standard output, unless the **-s** option was specified, in the following format:

"%d\n", <*number of bytes written***>**

If *file* begins with **'!'**, the rest of the line shall be taken to be a shell command line whose standard input shall be the addressed lines. Such a shell command line shall not be remembered as the current pathname. This usage of the write command with **'!'** shall not be considered as a "last **w** command that wrote the entire buffer", as described previously; thus, this alone shall not prevent the warning to the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands.

Line Number Command

Synopsis:

($)=

The line number of the addressed line shall be written to standard output in the following format:

"%d\n", <*line number***>**

The current line number shall be unchanged by this command.

Shell Escape Command

Synopsis:

!*command*

The remainder of the line after the **'!'** shall be sent to the command interpreter to be interpreted as a shell command line. Within the text of that shell command line, the unescaped character **'%'** shall be replaced with the remembered pathname; if a **'!'** appears as the first character of the command, it shall be replaced with the text of the previous shell command executed via **'!'** . Thus, **"!!"** shall repeat the previous !*command*. If any replacements of **'%'** or **'!'** are performed, the modified line shall be written to the standard output before *command* is executed. The **!** command shall write:

"!\n"

to standard output upon completion, unless the **-s** option is specified. The current line number shall be unchanged.

Null Command

Synopsis:

(.+1)

An address alone on a line shall cause the addressed line to be written. A <newline> alone shall be equivalent to **"+1p"** . The current line number shall be set to the address of the written line.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion without any file or command errors.

>0 An error occurred.

CONSEQUENCES OF ERRORS

When an error in the input script is encountered, or when an error is detected that is a consequence of the data (not) present in the file or due to an external condition such as a read or write error:

* If the standard input is a terminal device file, all input shall be flushed, and a new command read.
* If the standard input is a regular file, *ed* shall terminate with a non-zero exit status.

The following sections are informative.

APPLICATION USAGE

Because of the extremely terse nature of the default error messages, the prudent script writer begins the *ed* input commands with an **H** command, so that if any errors do occur at least some clue as to the cause is made available.

In previous versions, an obsolescent **-** option was described. This is no longer specified. Applications should use the **-s** option. Using **-** as a *file* operand now produces unspecified results. This allows implementations to continue to support the former required behavior.

EXAMPLES

None.

RATIONALE

The initial description of this utility was adapted from the SVID. It contains some features not found in Version 7 or BSD-derived systems. Some of the differences between the POSIX and BSD *ed* utilities include, but need not be limited to:

* The BSD **-** option does not suppress the **'!'** prompt after a **!** command.
* BSD does not support the special meanings of the **'%'** and **'!'** characters within a **!** command.
* BSD does not support the *addresses* **';'** and **','** .
* BSD allows the command/suffix pairs **pp**, **ll**, and so on, which are unspecified in this volume of IEEE Std 1003.1-2001.
* BSD does not support the **'!'** character part of the **e**, **r**, or **w** commands.
* A failed **g** command in BSD sets the line number to the last line searched if there are no matches.
* BSD does not default the *command list* to the **p** command.
* BSD does not support the **G**, **h**, **H**, **n**, or **V** commands.

* On BSD, if there is no inserted text, the insert command changes the current line to the referenced line -1; that is, the line before the specified line.
* On BSD, the *join* command with only a single address changes the current line to that address.
* BSD does not support the **P** command; moreover, in BSD it is synonymous with the **p** command.
* BSD does not support the *undo* of the commands **j**, **m**, **r**, **s**, or **t**.
* The Version 7 *ed* command **W**, and the BSD *ed* commands **W**, **wq**, and **z** are not present in this volume of IEEE Std 1003.1-2001.

The **-s** option was added to allow the functionality of the now withdrawn **-** option in a manner compatible with the Utility Syntax Guidelines.

In early proposals there was a limit, {ED_FILE_MAX}, that described the historical limitations of some *ed* utilities in their handling of large files; some of these have had problems with files larger than 100000 bytes. It was this limitation that prompted much of the desire to include a *split* command in this volume of IEEE Std 1003.1-2001. Since this limit was removed, this volume of IEEE Std 1003.1-2001 requires that implementations document the file size limits imposed by *ed* in the conformance document. The limit {ED_LINE_MAX} was also removed; therefore, the global limit {LINE_MAX} is used for input and output lines.

The manner in which the **l** command writes non-printable characters was changed to avoid the historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous because most terminals simply replace overstruck characters, making the **l** format not useful for its intended purpose of unambiguously understanding the content of the line. The historical backslash escapes were also ambiguous. (The string **"a\0011"** could represent a line containing those six characters or a line containing the three characters **'a'**, a byte with a binary value of 1, and a 1.) In the format required here, a backslash appearing in the line is written as **"\\"** so that the output is truly unambiguous. The method of marking the ends of lines was adopted from the *ex* editor and is required for any line ending in <space>s; the **'$'** is placed on all lines so that a real **'$'** at the end of a line cannot be misinterpreted.

Systems with bytes too large to fit into three octal digits must devise other means of displaying non-printable characters. Consideration was given to requiring that the number of octal digits be large enough to hold a byte, but this seemed to be too confusing for applications on the vast majority of systems where three digits are adequate. It would be theoretically possible for the application to use the *getconf* utility to find out the CHAR_BIT value and deal with such an algorithm; however, there is really no portable way that an application can use the octal values of the bytes across various coded character sets, so the additional specification was not worthwhile.

The description of how a NUL is written was removed. The NUL character cannot be in text files, and this volume of IEEE Std 1003.1-2001 should not dictate behavior in the case of undefined, erroneous input.

Unlike some of the other editing utilities, the filenames accepted by the **E**, **e**, **R**, and **r** commands are not patterns.

Early proposals stated that the **-p** option worked only when standard input was associated with a terminal device. This has been changed to conform to historical implementations, thereby allowing applications to interpose themselves between a user and the *ed* utility.

The form of the substitute command that uses the **n** suffix was limited in some historical documentation (where this was described incorrectly as "backreferencing"). This limit has been omitted because there is no reason why an editor processing lines of {LINE_MAX} length should have this restriction. The command **s/x/X/2047** should be able to substitute the 2047th occurrence of **'x'** on a line.

The use of printing commands with printing suffixes (such as **pn**, **lp**, and so on) was made unspecified because BSD-based systems allow this, whereas System V does not.

Some BSD-based systems exit immediately upon receipt of end-of-file if all of the lines in the file have been deleted. Since this volume of IEEE Std 1003.1-2001 refers to the **q** command in this instance, such behavior is not allowed.

Some historical implementations returned exit status zero even if command errors had occurred; this is not allowed by this volume of IEEE Std 1003.1-2001.

Some historical implementations contained a bug that allowed a single period to be entered in input mode as <backslash> <period> <newline>. This is not allowed by *ed* because there is no description of escaping any of the characters in input mode; backslashes are entered into the buffer exactly as typed. The typical method of entering a single period has been to precede it with another character and then use the substitute command to delete that character.

It is difficult under some modes of some versions of historical operating system terminal drivers to distinguish between an end-of-file condition and terminal disconnect. IEEE Std 1003.1-2001 does not require implementations to distinguish between the two situations, which permits historical implementations of the *ed* utility on historical platforms to conform. Implementations are encouraged to distinguish between the two, if possible, and take appropriate action on terminal disconnect.

Historically, *ed* accepted a zero address for the **a** and **r** commands in order to insert text at the start of the edit buffer. When the buffer was empty the command **.=** returned zero. IEEE Std 1003.1-2001 requires conformance to historical practice.

For consistency with the **a** and **r** commands and better user functionality, the **i** and **c** commands must also accept an address of 0, in which case 0*i* is treated as 1*i* and likewise for the **c** command.

All of the following are valid addresses:

+++ Three lines after the current line.

***/pattern/*-**
One line before the next occurrence of pattern.

-2 Two lines before the current line.

3 ---- 2 Line one (note the intermediate negative address).

1 2 3 Line six.

Any number of addresses can be provided to commands taking addresses; for example, **"1,2,3,4,5p"** prints lines 4 and 5, because two is the greatest valid number of addresses accepted by the **print** command. This, in combination with the semicolon delimiter, permits users to create commands based on ordered patterns in the file. For example, the command **"3;/foo/;+2p"** will display the first line after line 3 that contains the pattern *foo*, plus the next two lines. Note that the address **"3;"** must still be evaluated before being discarded, because the search origin for the **"/foo/"** command depends on this.

Historically, *ed* disallowed address chains, as discussed above, consisting solely of comma or semicolon separators; for example, **",,"** or **";;"** were considered an error. For consistency of address specification, this restriction is removed. The following table lists some of the address forms now possible:

Address	Addr1	Addr2	Status	Comment
7,	7	7	Historical	
7,5,	5	5	Historical	
7,5,9	5	9	Historical	
7,9	7	9	Historical	
7,+	7	8	Historical	
,	1	$	Historical	
,7	1	7	Extension	
,,	$	$	Extension	
,;	$	$	Extension	
7;	7	7	Historical	
7;5;	5	5	Historical	
7;5;9	5	9	Historical	
7;5,9	5	9	Historical	
7;$;4	$	4	Historical	Valid, but erroneous.
7;9	7	9	Historical	
7;+	7	8	Historical	
;	.	$	Historical	
;7	.	7	Extension	
;;	$	$	Extension	
;,	$	$	Extension	

Historically, values could be added to addresses by including them after one or more <blank>s; for

example, **"3 - 5p"** wrote the seventh line of the file, and **"/foo/ 5"** was the same as **"5 /foo/"** . However, only absolute values could be added; for example, **"5 /foo/"** was an error. IEEE Std 1003.1-2001 requires conformance to historical practice.

Historically, *ed* accepted the **'^'** character as an address, in which case it was identical to the hyphen character. IEEE Std 1003.1-2001 does not require or prohibit this behavior.

FUTURE DIRECTIONS

None.

SEE ALSO

Utility Description Defaults, *ex*, *sed*, *sh*, *vi*

COPYRIGHT