

NAME

sed – stream editor for filtering and transforming text

SYNOPSIS

j:Textproc4.1.5-4.1.5.exe [*OPTION*]... {*script-only-if-no-other-script*} [*input-file*]...

DESCRIPTION

Sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as *ed*), *sed* works by making only one pass over the input(s), and is consequently more efficient. But it is *sed*'s ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

-n, --quiet, --silent

suppress automatic printing of pattern space

-e script, --expression=script

add the script to the commands to be executed

-f script-file, --file=script-file

add the contents of script-file to the commands to be executed

--follow-symlinks

follow symlinks when processing in place

-i[SUFFIX], --in-place[=SUFFIX]

edit files in place (makes backup if extension supplied)

-c, --copy

use copy instead of rename when shuffling files in **-i** mode (avoids change of input file ownership)

-l N, --line-length=N

specify the desired line-wrap length for the 'l' command

--posix

disable all GNU extensions.

-r, --regexp-extended

use extended regular expressions in the script.

-s, --separate

consider files as separate rather than as a single continuous long stream.

-u, --unbuffered

load minimal amounts of data from the input files and flush the output buffers more often

--help display this help and exit**--version**

output version information and exit

-T, --textmode use textmode read and write

If no **-e**, **--expression**, **-f**, or **--file** option is given, then the first non-option argument is taken as the sed script to interpret. All remaining arguments are names of input files; if no input files are specified, then the standard input is read.

E-mail bug reports to: bonzini@gnu.org . Be sure to include the word “sed” somewhere in the “Subject:” field.

COMMAND SYNOPSIS

This is just a brief synopsis of *sed* commands to serve as a reminder to those who already know *sed*; other documentation (such as the texinfo document) must be consulted for fuller descriptions.

Zero-address “commands”

: label Label for **b** and **t** commands.

#comment

The comment extends until the next newline (or the end of a **-e** script fragment).

}

The closing bracket of a { } block.

Zero- or One- address commands

= Print the current line number.

a \

text Append *text*, which has each embedded newline preceded by a backslash.

i \

text Insert *text*, which has each embedded newline preceded by a backslash.

q Immediately quit the *sed* script without processing any more input, except that if auto-print is not disabled the current pattern space will be printed.

Q Immediately quit the *sed* script without processing any more input.

r *filename*

Append text read from *filename*.

R *filename*

Append a line read from *filename*.

Commands which accept address ranges

{ Begin a block of commands (end with a }).

b *label* Branch to *label*; if *label* is omitted, branch to end of script.

t *label* If a *s///* has done a successful substitution since the last input line was read and since the last t or T command, then branch to *label*; if *label* is omitted, branch to end of script.

T *label* If no *s///* has done a successful substitution since the last input line was read and since the last t or T command, then branch to *label*; if *label* is omitted, branch to end of script.

c \

text Replace the selected lines with *text*, which has each embedded newline preceded by a backslash.

d Delete pattern space. Start next cycle.

D Delete up to the first embedded newline in the pattern space. Start next cycle, but skip reading from the input if there is still data in the pattern space.

h H Copy/append pattern space to hold space.

g G Copy/append hold space to pattern space.

x Exchange the contents of the hold and pattern spaces.

l List out the current line in a “visually unambiguous” form.

n N Read/append the next line of input into the pattern space.

p Print the current pattern space.

P Print up to the first embedded newline of the current pattern space.

s/regexp/replacement/

Attempt to match *regexp* against the pattern space. If successful, replace that portion matched with *replacement*. The *replacement* may contain the special character **&** to refer to that portion of the pattern space which matched, and the special escapes **\1** through **\9** to refer to the corresponding matching sub-expressions in the *regexp*.

w filename

Write the current pattern space to *filename*.

W filename

Write the first line of the current pattern space to *filename*.

y/source/dest/

Transliterate the characters in the pattern space which appear in *source* to the corresponding character in *dest*.

Addresses

Sed commands can be given with no addresses, in which case the command will be executed for all input lines; with one address, in which case the command will only be executed for input lines which match that address; or with two addresses, in which case the command will be executed for all input lines which match the inclusive range of lines starting from the first address and continuing to the second address. Three things to note about address ranges: the syntax is *addr1,addr2* (i.e., the addresses are separated by a comma); the line which *addr1* matched will always be accepted, even if *addr2* selects an earlier line; and if *addr2* is a *regexp*, it will not be tested against the line that *addr1* matched.

After the address (or address-range), and before the command, a **!** may be inserted, which specifies that the command shall only be executed if the address (or address-range) does **not** match.

The following address types are supported:

number Match only the specified line *number*.

first~step

Match every *step*'th line starting with line *first*. For example, “*sed -n 1~2p*” will print all the odd-numbered lines in the input stream, and the address *2~5* will match every fifth line, starting with the second. (This is an extension.)

\$ Match the last line.

/regexp/

Match lines matching the regular expression *regexp*.

\cregexpc

Match lines matching the regular expression *regexp*. The **c** may be any character.

GNU *sed* also supports some special 2-address forms:

0,addr2

Start out in "matched first address" state, until *addr2* is found. This is similar to *1,addr2*, except that if *addr2* matches the very first line of input the *0,addr2* form will be at the end of its range, whereas the *1,addr2* form will still be at the beginning of its range.

addr1,+N

Will match *addr1* and the *N* lines following *addr1*.

addr1,~N

Will match *addr1* and the lines following *addr1* until the next line whose input line number is a multiple of *N*.

REGULAR EXPRESSIONS

POSIX.2 BREs *should* be supported, but they aren't completely because of performance problems. The **\n** sequence in a regular expression matches the newline character, and similarly for **\a**, **\t**, and other sequences.

BUGS

E-mail bug reports to **bonzini@gnu.org**. Be sure to include the word “sed” somewhere in the “Subject:” field. Also, please include the output of “*sed --version*” in the body of your report if at all possible.

COPYRIGHT

Copyright © 2003 Free Software Foundation, Inc.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE, to the extent permitted by law.

SEE ALSO

awk(1), **ed(1)**, **grep(1)**, **tr(1)**, **perlre(1)**, sed.info, any of various books on *sed*, the *sed* FAQ (<http://sed.sf.net/grabbag/tutorials/sedfaq.txt>), <http://sed.sf.net/grabbag/>.

The full documentation for **sed** is maintained as a Texinfo manual. If the **info** and **sed** programs are properly installed at your site, the command

info sed

should give you access to the complete manual.

NAME

sed – stream editor

SYNOPSIS

sed [-n] *script*[*file*...]

sed [-n][-e *script*]...[-f *script_file*]...[*file*...]

DESCRIPTION

The *sed* utility is a stream editor that shall read one or more text files, make editing changes according to a script of editing commands, and write the results to standard output. The script shall be obtained from either the *script* operand string or a combination of the option-arguments from the **-e** *script* and **-f** *script_file* options.

OPTIONS

The *sed* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines, except that the order of presentation of the **-e** and **-f** options is significant.

The following options shall be supported:

-e *script*

Add the editing commands specified by the *script* option-argument to the end of the script of editing commands. The *script* option-argument shall have the same properties as the *script* operand, described in the OPERANDS section.

-f *script_file*

Add the editing commands in the file *script_file* to the end of the script.

-n

Suppress the default output (in which each line, after it is examined for editing, is written to standard output). Only lines explicitly selected for output are written.

Multiple **-e** and **-f** options may be specified. All commands shall be added to the script in the order specified, regardless of their origin.

OPERANDS

The following operands shall be supported:

file A pathname of a file whose contents are read and edited. If multiple *file* operands are specified, the named files shall be read in the order specified and the concatenation shall be edited. If no *file* operands are specified, the standard input shall be used.

script A string to be used as the script of editing commands. The application shall not present a *script* that violates the restrictions of a text file except that the final character need not be a <newline>.

STDIN

The standard input shall be used only if no *file* operands are specified. See the INPUT FILES section.

INPUT FILES

The input files shall be text files. The *script_files* named by the **-f** option shall consist of editing commands.

ENVIRONMENT VARIABLES

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

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.

NLSPATH

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

ASYNCHRONOUS EVENTS

Default.

STDOUT

The input files shall be written to standard output, with the editing commands specified in the script applied. If the **-n** option is specified, only those input lines selected by the script shall be written to standard output.

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 *script* shall consist of editing commands of the following form:

[address[,address]]function

where *function* represents a single-character command verb from the list in Editing Commands in sed , followed by any applicable arguments.

The command can be preceded by <blank>s and/or semicolons. The function can be preceded by <blank>s. These optional characters shall have no effect.

In default operation, *sed* cyclically shall append a line of input, less its terminating <newline>, into the pattern space. Normally the pattern space will be empty, unless a **D** command terminated the last cycle. The *sed* utility shall then apply in sequence all commands whose addresses select that pattern space, and at the end of the script copy the pattern space to standard output (except when **-n** is specified) and delete the pattern space. Whenever the pattern space is written to standard output or a named file, *sed* shall immediately follow it with a <newline>.

Some of the editing commands use a hold space to save all or part of the pattern space for subsequent retrieval. The pattern and hold spaces shall each be able to hold at least 8192 bytes.

Addresses in sed

An address is either a decimal number that counts input lines cumulatively across files, a '\$' character that addresses the last line of input, or a context address (which consists of a BRE, as described in Regular Expressions in sed , preceded and followed by a delimiter, usually a slash).

An editing command with no addresses shall select every pattern space.

An editing command with one address shall select each pattern space that matches the address.

An editing command with two addresses shall select the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line shall be selected.) Starting at the first line following the selected range, *sed* shall look again for the first address. Thereafter, the process shall be repeated. Omitting either or both of the address components in the following form

produces undefined results:

[*address*[,*address*]]

Regular Expressions in sed

The *sed* utility shall support the BREs described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.3, Basic Regular Expressions, with the following additions:

- * In a context address, the construction "**\cBREc**", where *c* is any character other than backslash or <newline>, shall be identical to **"/BRE/"**. If the character designated by *c* appears following a backslash, then it shall be considered to be that literal character, which shall not terminate the BRE. For example, in the context address "**\xabc\xdefx**", the second *x* stands for itself, so that the BRE is **"abcxdef"**.
- * The escape sequence **'\n'** shall match a <newline> embedded in the pattern space. A literal <newline> shall not be used in the BRE of a context address or in the substitute function.
- * If an RE is empty (that is, no pattern is specified) *sed* shall behave as if the last RE used in the last command applied (either as an address or as part of a substitute command) was specified.

Editing Commands in sed

In the following list of editing commands, the maximum number of permissible addresses for each function is indicated by [*0addr*], [*1addr*], or [*2addr*], representing zero, one, or two addresses.

The argument *text* shall consist of one or more lines. Each embedded <newline> in the text shall be preceded by a backslash. Other backslashes in text shall be removed, and the following character shall be treated literally.

The **r** and **w** command verbs, and the *w* flag to the **s** command, take an optional *rfile* (or *wfile*) parameter, separated from the command verb letter or flag by one or more <blank>s; implementations may allow zero separation as an extension.

The argument *rfile* or the argument *wfile* shall terminate the editing command. Each *wfile* shall be created before processing begins. Implementations shall support at least ten *wfile* arguments in the script; the actual number (greater than or equal to 10) that is supported by the implementation is unspecified. The use of the *wfile* parameter shall cause that file to be initially created, if it does not exist, or shall replace the contents of an existing file.

The **b**, **r**, **s**, **t**, **w**, **y**, and **:** command verbs shall accept additional arguments. The following synopses indicate which arguments shall be separated from the command verbs by a single <space>.

The **a** and **r** commands schedule text for later output. The text specified for the **a** command, and the contents of the file specified for the **r** command, shall be written to standard output just before the next attempt to fetch a line of input when executing the **N** or **n** commands, or when reaching the end of the script. If written when reaching the end of the script, and the **-n** option was not specified, the text shall be written after copying the pattern space to standard output. The contents of the file specified for the **r** command shall be as of the time the output is written, not the time the **r** command is applied. The text shall be output in the order in which the **a** and **r** commands were applied to the input.

Command verbs other than **{**, **a**, **b**, **c**, **i**, **r**, **t**, **w**, **:**, and **#** can be followed by a semicolon, optional <blank>s, and another command verb. However, when the **s** command verb is used with the *w* flag, following it with another command in this manner produces undefined results.

A function can be preceded by one or more **'!'** characters, in which case the function shall be applied if the addresses do not select the pattern space. Zero or more <blank>s shall be accepted before the first **'!'** character. It is unspecified whether <blank>s can follow a **'!'** character, and conforming applications shall not follow a **'!'** character with <blank>s.

[*2addr*] {*function*

function

...

} Execute a list of *sed* functions only when the pattern space is selected. The list of *sed* functions shall be surrounded by braces and separated by <newline>s, and conform to the following

rules. The braces can be preceded or followed by <blank>s. The functions can be preceded by <blank>s, but shall not be followed by <blank>s. The <right-brace> shall be preceded by a <newline> and can be preceded or followed by <blank>s.

[*1addr*]**a**\

text Write *text* to standard output as described previously.

[*2addr*]**b** [*label*]

Branch to the **:** function bearing the *label*. If *label* is not specified, branch to the end of the script. The implementation shall support *labels* recognized as unique up to at least 8 characters; the actual length (greater than or equal to 8) that shall be supported by the implementation is unspecified. It is unspecified whether exceeding a label length causes an error or a silent truncation.

[*2addr*]**c**\

text Delete the pattern space. With a 0 or 1 address or at the end of a 2-address range, place *text* on the output and start the next cycle.

[*2addr*]**d**

Delete the pattern space and start the next cycle.

[*2addr*]**D**

Delete the initial segment of the pattern space through the first <newline> and start the next cycle.

[*2addr*]**g**

Replace the contents of the pattern space by the contents of the hold space.

[*2addr*]**G**

Append to the pattern space a <newline> followed by the contents of the hold space.

[*2addr*]**h**

Replace the contents of the hold space with the contents of the pattern space.

[*2addr*]**H**

Append to the hold space a <newline> followed by the contents of the pattern space.

[*1addr*]**i**\

text Write *text* to standard output.

[*2addr*]**l**

(The letter ell.) Write the pattern space to standard output 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 that table shall be written as one three-digit octal number (with a preceding backslash) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than 9 bits, the format used for non-printable characters is implementation-defined.

Long lines shall be folded, with the point of folding indicated by writing a backslash followed by a <newline>; 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 `'$'`.

[*2addr*]**n**

Write the pattern space to standard output if the default output has not been suppressed, and replace the pattern space with the next line of input, less its terminating <newline>.

If no next line of input is available, the **n** command verb shall branch to the end of the script and quit without starting a new cycle.

[*2addr*]**N**

Append the next line of input, less its terminating <newline>, to the pattern space, using an embedded <newline> to separate the appended material from the original material. Note that the current line number changes.

If no next line of input is available, the **N** command verb shall branch to the end of the script and quit without starting a new cycle or copying the pattern space to standard output.

[2addr]**p**

Write the pattern space to standard output.

[2addr]**P**

Write the pattern space, up to the first <newline>, to standard output.

[1addr]**q**

Branch to the end of the script and quit without starting a new cycle.

[1addr]**r** *rfile*

Copy the contents of *rfile* to standard output as described previously. If *rfile* does not exist or cannot be read, it shall be treated as if it were an empty file, causing no error condition.

[2addr]**s**/BRE/replacement/flags

Substitute the replacement string for instances of the BRE in the pattern space. Any character other than backslash or <newline> can be used instead of a slash to delimit the BRE and the replacement. Within the BRE and the replacement, the BRE delimiter itself can be used as a literal character if it is preceded by a backslash.

The replacement string shall be scanned from beginning to end. An ampersand ('&') appearing in the replacement shall be replaced by the string matching the BRE. The special meaning of '&' in this context can be suppressed by preceding it by a backslash. The characters "\n", where *n* is a digit, shall be replaced by the text matched by the corresponding backreference expression. The special meaning of "\n" where *n* is a digit in this context, can be suppressed by preceding it by a backslash. For each other backslash ('\ ') encountered, the following character shall lose its special meaning (if any). The meaning of a '\ ' immediately followed by any character other than '&', '\ ', a digit, or the delimiter character used for this command, is unspecified.

A line can be split by substituting a <newline> into it. The application shall escape the <newline> in the replacement by preceding it by a backslash. A substitution shall be considered to have been performed even if the replacement string is identical to the string that it replaces. Any backslash used to alter the default meaning of a subsequent character shall be discarded from the BRE or the replacement before evaluating the BRE or using the replacement.

The value of *flags* shall be zero or more of:

n

Substitute for the *n*th occurrence only of the BRE found within the pattern space.

g

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

p

Write the pattern space to standard output if a replacement was made.

w *wfile*

Write. Append the pattern space to *wfile* if a replacement was made. A conforming application shall precede the *wfile* argument with one or more <blank>s. If the **w** flag is not the last flag value given in a concatenation of multiple flag values, the results are undefined.

[2addr]**t** [*label*]

Test. Branch to the : command verb bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is not specified, branch to the end of the script.

[2addr]**w** *wfile*

Append (write) the pattern space to *wfile*.

[2addr]x

Exchange the contents of the pattern and hold spaces.

[2addr]y/string1/string2/

Replace all occurrences of characters in *string1* with the corresponding characters in *string2*. If a backslash followed by an 'n' appear in *string1* or *string2*, the two characters shall be handled as a single <newline>. If the number of characters in *string1* and *string2* are not equal, or if any of the characters in *string1* appear more than once, the results are undefined. Any character other than backslash or <newline> can be used instead of slash to delimit the strings. If the delimiter is not *n*, within *string1* and *string2*, the delimiter itself can be used as a literal character if it is preceded by a backslash. If a backslash character is immediately followed by a backslash character in *string1* or *string2*, the two backslash characters shall be counted as a single literal backslash character. The meaning of a backslash followed by any character that is not 'n', a backslash, or the delimiter character is undefined.

[0addr]:label

Do nothing. This command bears a *label* to which the **b** and **t** commands branch.

[1addr]=

Write the following to standard output:

```
"%d\n", <current line number>
```

[0addr]

Ignore this empty command.

[0addr]#

Ignore the '#' and the remainder of the line (treat them as a comment), with the single exception that if the first two characters in the script are "#n", the default output shall be suppressed; this shall be the equivalent of specifying **-n** on the command line.

EXIT STATUS

The following exit values shall be returned:

- | | |
|----|------------------------|
| 0 | Successful completion. |
| >0 | An error occurred. |

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

Regular expressions match entire strings, not just individual lines, but a <newline> is matched by '\n' in a *sed* RE; a <newline> is not allowed by the general definition of regular expression in IEEE Std 1003.1-2001. Also note that '\n' cannot be used to match a <newline> at the end of an arbitrary input line; <newline>s appear in the pattern space as a result of the **N** editing command.

EXAMPLES

This *sed* script simulates the BSD *cat -s* command, squeezing excess blank lines from standard input.

```
sed -n '
# Write non-empty lines.
./ {
    p
    d
}
# Write a single empty line, then look for more empty lines.
/^$/ p
# Get next line, discard the held <newline> (empty line),
```

```

# and look for more empty lines.
:Empty
/^$/ {
    N
    s!./!
    b Empty
}
# Write the non-empty line before going back to search
# for the first in a set of empty lines.
p

```

RATIONALE

This volume of IEEE Std 1003.1-2001 requires implementations to support at least ten distinct *wfiles*, matching historical practice on many implementations. Implementations are encouraged to support more, but conforming applications should not exceed this limit.

The exit status codes specified here are different from those in System V. System V returns 2 for garbled *sed* commands, but returns zero with its usage message or if the input file could not be opened. The standard developers considered this to be a bug.

The manner in which the **l** command writes non-printable characters was changed to avoid the historical backspace-overstrike method, and other requirements to achieve unambiguous output were added. See the RATIONALE for *ed* for details of the format chosen, which is the same as that chosen for *sed*.

This volume of IEEE Std 1003.1-2001 requires implementations to provide pattern and hold spaces of at least 8192 bytes, larger than the 4000 bytes spaces used by some historical implementations, but less than the 20480 bytes limit used in an early proposal. Implementations are encouraged to allocate dynamically larger pattern and hold spaces as needed.

The requirements for acceptance of <blank>s and <space>s in command lines has been made more explicit than in early proposals to describe clearly the historical practice and to remove confusion about the phrase "protect initial blanks [*sic*] and tabs from the stripping that is done on every script line" that appears in much of the historical documentation of the *sed* utility description of text. (Not all implementations are known to have stripped <blank>s from text lines, although they all have allowed leading <blank>s preceding the address on a command line.)

The treatment of '#' comments differs from the SVID which only allows a comment as the first line of the script, but matches BSD-derived implementations. The comment character is treated as a command, and it has the same properties in terms of being accepted with leading <blank>s; the BSD implementation has historically supported this.

Early proposals required that a *script_file* have at least one non-comment line. Some historical implementations have behaved in unexpected ways if this were not the case. The standard developers considered that this was incorrect behavior and that application developers should not have to avoid this feature. A correct implementation of this volume of IEEE Std 1003.1-2001 shall permit *script_files* that consist only of comment lines.

Early proposals indicated that if **-e** and **-f** options were intermixed, all **-e** options were processed before any **-f** options. This has been changed to process them in the order presented because it matches historical practice and is more intuitive.

The treatment of the **p** flag to the **s** command differs between System V and BSD-based systems when the default output is suppressed. In the two examples:

```

echo a | sed 's/a/A/p'
echo a | sed -n 's/a/A/p'

```

this volume of IEEE Std 1003.1-2001, BSD, System V documentation, and the SVID indicate that the first example should write two lines with **A**, whereas the second should write one. Some System V systems write the **A** only once in both examples because the **p** flag is ignored if the **-n** option is not specified.

This is a case of a diametrical difference between systems that could not be reconciled through the compromise of declaring the behavior to be unspecified. The SVID/BSD/System V documentation

behavior was adopted for this volume of IEEE Std 1003.1-2001 because:

- * No known documentation for any historic system describes the interaction between the **p** flag and the **-n** option.
- * The selected behavior is more correct as there is no technical justification for any interaction between the **p** flag and the **-n** option. A relationship between **-n** and the **p** flag might imply that they are only used together, but this ignores valid scripts that interrupt the cyclical nature of the processing through the use of the **D**, **d**, **q**, or branching commands. Such scripts rely on the **p** suffix to write the pattern space because they do not make use of the default output at the "bottom" of the script.
- * Because the **-n** option makes the **p** flag unnecessary, any interaction would only be useful if *sed* scripts were written to run both with and without the **-n** option. This is believed to be unlikely. It is even more unlikely that programmers have coded the **p** flag expecting it to be unnecessary. Because the interaction was not documented, the likelihood of a programmer discovering the interaction and depending on it is further decreased.
- * Finally, scripts that break under the specified behavior produce too much output instead of too little, which is easier to diagnose and correct.

The form of the substitute command that uses the **n** suffix was limited to the first 512 matches in an early proposal. This limit has been removed because there is no reason an editor processing lines of {LINE_MAX} length should have this restriction. The command **s/a/A/2047** should be able to substitute the 2047th occurrence of **a** on a line.

The **b**, **t**, and **:** commands are documented to ignore leading white space, but no mention is made of trailing white space. Historical implementations of *sed* assigned different locations to the labels **'x'** and **"x "**. This is not useful, and leads to subtle programming errors, but it is historical practice, and changing it could theoretically break working scripts. Implementors are encouraged to provide warning messages about labels that are never used or jumps to labels that do not exist.

Historically, the *sed* **!** and **}** editing commands did not permit multiple commands on a single line using a semicolon as a command delimiter. Implementations are permitted, but not required, to support this extension.

FUTURE DIRECTIONS

None.

SEE ALSO

awk, *ed*, *grep*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1, 2003 Edition, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 6, Copyright (C) 2001-2003 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at <http://www.open-group.org/unix/online.html> .