NAME

sed - stream editor

SYNOPSIS

sed [**-Ealnru**] *command* [**-I** *extension*] [**-i** *extension*] [*file* ...]
sed [**-Ealnru**] [**-e** *command*] [**-f** *command_file*] [**-I** *extension*] [**-i** *extension*] [*file* ...]

DESCRIPTION

The **sed** utility reads the specified files, or the standard input if no files are specified, modifying the input as specified by a list of commands. The input is then written to the standard output.

A single command may be specified as the first argument to **sed**. Multiple commands may be specified by using the **-e** or **-f** options. All commands are applied to the input in the order they are specified regardless of their origin.

The following options are available:

-E Interpret regular expressions as extended (modern) regular expressions rather than basic regular expressions (BRE's). The re_format(7) manual page fully describes both formats.

-a The files listed as parameters for the "w" functions are created (or truncated) before any processing begins, by default. The **-a** option causes **sed** to delay opening each file until a command containing the related "w" function is applied to a line of input.

-e *command*
Append the editing commands specified by the *command* argument to the list of commands.

-f *command_file*
Append the editing commands found in the file *command_file* to the list of commands. The editing commands should each be listed on a separate line. The commands are read from the standard input if *command_file* is "-".

-I *extension*
Edit files in-place, saving backups with the specified *extension*. If a zero-length *extension* is given, no backup will be saved. It is not recommended to give a zero-length *extension* when in-place editing files, as you risk corruption or partial content in situations where disk space is exhausted, etc.

Note that in-place editing with **-I** still takes place in a single continuous line address space covering all files, although each file preserves its individuality instead of forming one output

stream. The line counter is never reset between files, address ranges can span file boundaries, and the "$" address matches only the last line of the last file. (See *Sed Addresses*.) That can lead to unexpected results in many cases of in-place editing, where using **-i** is desired.

-i *extension*
Edit files in-place similarly to **-I**, but treat each file independently from other files. In particular, line numbers in each file start at 1, the "$" address matches the last line of the current file, and address ranges are limited to the current file. (See *Sed Addresses*.) The net result is as though each file were edited by a separate **sed** instance.

-l Make output line buffered.

-n By default, each line of input is echoed to the standard output after all of the commands have been applied to it. The **-n** option suppresses this behavior.

-r Same as **-E** for compatibility with GNU sed.

-u Make output unbuffered.

The form of a **sed** command is as follows:

[address[,address]]function[arguments]

Whitespace may be inserted before the first address and the function portions of the command.

Normally, **sed** cyclically copies a line of input, not including its terminating newline character, into a *pattern space*, (unless there is something left after a "D" function), applies all of the commands with addresses that select that pattern space, copies the pattern space to the standard output, appending a newline, and deletes the pattern space.

Some of the functions use a *hold space* to save all or part of the pattern space for subsequent retrieval.

Sed Addresses

An address is not required, but if specified must have one of the following formats:

- a number that counts input lines cumulatively across input files (or in each file independently if a **-i** option is in effect);
- a dollar ("$") character that addresses the last line of input (or the last line of the current file if a **-i** option was specified);

- a context address that consists of a regular expression preceded and followed by a delimiter. The closing delimiter can also optionally be followed by the "I" character, to indicate that the regular expression is to be matched in a case-insensitive way.

A command line with no addresses selects every pattern space.

A command line with one address selects all of the pattern spaces that match the address.

A command line with two addresses selects an inclusive range. This range starts with the first pattern space that matches the first address. The end of the range is the next following pattern space that matches the second address. If the second address is a number less than or equal to the line number first selected, only that line is selected. The number in the second address may be prefixed with a ("+") to specify the number of lines to match after the first pattern. In the case when the second address is a context address, **sed** does not re-match the second address against the pattern space that matched the first address. Starting at the first line following the selected range, **sed** starts looking again for the first address.

Editing commands can be applied to non-selected pattern spaces by use of the exclamation character ("!") function.

Sed Regular Expressions

The regular expressions used in **sed**, by default, are basic regular expressions (BREs, see re_format(7) for more information), but extended (modern) regular expressions can be used instead if the **-E** flag is given. In addition, **sed** has the following two additions to regular expressions:

1. In a context address, any character other than a backslash ("\") or newline character may be used to delimit the regular expression. The opening delimiter needs to be preceded by a backslash unless it is a slash. For example, the context address \xabcx is equivalent to /abc/. Also, putting a backslash character before the delimiting character within the regular expression causes the character to be treated literally. For example, in the context address \xabc\xdefx, the RE delimiter is an "x" and the second "x" stands for itself, so that the regular expression is "abcxdef".

2. The escape sequence \n matches a newline character embedded in the pattern space. You cannot, however, use a literal newline character in an address or in the substitute command.

One special feature of **sed** regular expressions is that they can default to the last regular expression used. If a regular expression is empty, i.e., just the delimiter characters are specified, the last regular expression encountered is used instead. The last regular expression is defined as the last regular expression used as part of an address or substitute command, and at run-time, not compile-time. For example, the command "/abc/s//XXX/" will substitute "XXX" for the pattern "abc".

Sed Functions

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

The argument *text* consists of one or more lines. To embed a newline in the text, precede it with a backslash. Other backslashes in text are deleted and the following character taken literally.

The "r" and "w" functions take an optional file parameter, which should be separated from the function letter by white space. Each file given as an argument to **sed** is created (or its contents truncated) before any input processing begins.

The "b", "r", "s", "t", "w", "y", "!", and ":" functions all accept additional arguments. The following synopses indicate which arguments have to be separated from the function letters by white space characters.

Two of the functions take a function-list. This is a list of **sed** functions separated by newlines, as follows:

```
{ function
  function
  ...
  function
}
```

The "{" can be preceded by white space and can be followed by white space. The function can be preceded by white space. The terminating "}" must be preceded by a newline, and may also be preceded by white space.

[2addr] function-list
: Execute function-list only when the pattern space is selected.

[1addr]a\
text
: Write *text* to standard output immediately before each attempt to read a line of input, whether by executing the "N" function or by beginning a new cycle.

[2addr]b[label]
: Branch to the ":" function with the specified label. If the label is not specified, branch to the end of the script.

[2addr]c\

text Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, *text* is written to the standard output.

[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 character and start the next cycle.

[2addr]g Replace the contents of the pattern space with the contents of the hold space.

[2addr]G Append a newline character followed by the contents of the hold space to the pattern space.

[2addr]h Replace the contents of the hold space with the contents of the pattern space.

[2addr]H Append a newline character followed by the contents of the pattern space to the hold space.

[1addr]i\
text Write *text* to the standard output.

[2addr]l (The letter ell.) Write the pattern space to the standard output in a visually unambiguous form. This form is as follows:

backslash	\\
alert	\a
form-feed	\f
carriage-return	\r
tab	\t
vertical tab	\v

Nonprintable characters are written as three-digit octal numbers (with a preceding backslash) for each byte in the character (most significant byte first). Long lines are folded, with the point of folding indicated by displaying a backslash followed by a newline. The end of each line is marked with a "$".

[2addr]n Write the pattern space to the standard output if the default output has not been suppressed, and replace the pattern space with the next line of input.

[2addr]N Append the next line of input to the pattern space, using an embedded newline character to separate the appended material from the original contents. Note that the current line number changes.

[2addr]p Write the pattern space to standard output.

[2addr]P Write the pattern space, up to the first newline character to the standard output.

[1addr]q Branch to the end of the script and quit without starting a new cycle.

[1addr]r file

Copy the contents of *file* to the standard output immediately before the next attempt to read a line of input. If *file* cannot be read for any reason, it is silently ignored and no error condition is set.

[2addr]s/regular expression/replacement/flags

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

An ampersand ("&") appearing in the replacement is replaced by the string matching the RE. The special meaning of "&" in this context can be suppressed by preceding it by a backslash. The string "\#", where "#" is a digit, is replaced by the text matched by the corresponding backreference expression (see re_format(7)).

A line can be split by substituting a newline character into it. To specify a newline character in the replacement string, precede it with a backslash.

The value of *flags* in the substitute function is zero or more of the following:

N Make the substitution only for the *N*'th occurrence of the regular expression in the pattern space.

g Make the substitution for all non-overlapping matches of the regular expression, not just the first one.

p Write the pattern space to standard output if a replacement was made. If the replacement string is identical to that which it replaces, it is still considered to have been a replacement.

w *file* Append the pattern space to *file* if a replacement was made. If the replacement string is identical to that which it replaces, it is still considered to have been a replacement.

i or I Match the regular expression in a case-insensitive way.

[2addr]t [label]

Branch to the ":" function bearing the label if any substitutions have been made since the most recent reading of an input line or execution of a "t" function. If no label is specified, branch to the end of the script.

[2addr]w *file*

Append the pattern space to the *file*.

[2addr]x Swap the contents of the pattern and hold spaces.

[2addr]y/string1/string2/

Replace all occurrences of characters in *string1* in the pattern space with the corresponding characters from *string2*. Any character other than a backslash or newline can be used instead of a slash to delimit the strings. Within *string1* and *string2*, a backslash followed by any character other than a newline is that literal character, and a backslash followed by an ‘‘n’’ is replaced by a newline character.

[2addr]!function

[2addr]!function-list

Apply the function or function-list only to the lines that are *not* selected by the address(es).

[0addr]:label

This function does nothing; it bears a label to which the "b" and "t" commands may branch.

[1addr]= Write the line number to the standard output followed by a newline character.

[0addr] Empty lines are ignored.

[0addr]# The "#" and the remainder of the line are ignored (treated as a comment), with the single exception that if the first two characters in the file are "#n", the default output is suppressed. This is the same as specifying the **-n** option on the command line.

ENVIRONMENT

The COLUMNS, LANG, LC_ALL, LC_CTYPE and LC_COLLATE environment variables affect the execution of **sed** as described in environ(7).

EXIT STATUS

The **sed** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

Replace ‘bar’ with ‘baz’ when piped from another command:

```
echo "An alternate word, like bar, is sometimes used in examples." | sed ’s/bar/baz/’
```

Using backlashes can sometimes be hard to read and follow:

```
echo "/home/example" | sed  ’s/\/home\/example/\/usr\/local\/example/’
```

Using a different separator can be handy when working with paths:

```
echo "/home/example" | sed ’s#/home/example#/usr/local/example#’
```

Replace all occurrences of ‘foo’ with ‘bar’ in the file *test.txt*, without creating a backup of the file:

```
sed -i ’’ -e ’s/foo/bar/g’ test.txt
```

SEE ALSO

awk(1), ed(1), grep(1), regex(3), re_format(7)

Lee E. McMahon, *SED -- A Non-interactive Text Editor*, *AT&T Bell Laboratories*, Computing Science Technical Report, 77, January 1979.

STANDARDS

The **sed** utility is expected to be a superset of the IEEE Std 1003.2 ("POSIX.2") specification.

The **-E**, **-I**, **-a** and **-i** options, the special meaning of **-f -**, the prefixing "+" in the second member of an address range, as well as the "I" flag to the address regular expression and substitution command are non-standard FreeBSD extensions and may not be available on other operating systems.

HISTORY

A **sed** command, written by L. E. McMahon, appeared in Version 7 AT&T UNIX.

AUTHORS

Diomidis D. Spinellis <*dds@FreeBSD.org*>

BUGS

Multibyte characters containing a byte with value 0x5C (ASCII ‘\’) may be incorrectly treated as line continuation characters in arguments to the "a", "c" and "i" commands. Multibyte characters cannot be used as delimiters with the "s" and "y" commands.