sed, a stream editor

version 3.02, 28 June 1998
DOC-0473-00

by Ken Pizzini

Published by the Free Software Foundation,
59 Temple Place - Suite 330,
Boston, MA 02111-1307, USA

1 Introduction

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.

2 Invocation

SED may be invoked with the following command-line options:

'`-V`'
'`--version`'
Print out the version of SED that is being run and a copyright notice, then exit.

'`-h`'
'`--help`' Print a usage message briefly summarizing these command-line options and the bug-reporting address, then exit.

'`-n`'
'`--quiet`'
'`--silent`'
By default, SED will print out the pattern space at then end of each cycle through the script. These options disable this automatic printing, and SED will only produce output when explicitly told to via the `p` command.

'`-e` *script*'
'`--expression=`*script*'
Add the commands in *script* to the set of commands to be run while processing the input.

'`-f` *script-file*'
'`--file=`*script-file*'
Add the commands contained in the file *script-file* to the set of commands to be run while processing the input.

If no `-e`, `-f`, `--expression`, or `--file` options are given on the command-line, then the first non-option argument on the command line is taken to be the *script* to be executed.

If any command-line parameters remain after processing the above, these parameters are interpreted as the names of input files to be processed. A file name of `-` refers to the standard input stream. The standard input will processed if no file names are specified.

3 SED Programs

A SED program consists of one or more SED commands, passed in by one or more of the `-e`, `-f`, `--expression`, and `--file` options, or the first non-option argument if zero of these options are used. This document will refer to "the" SED script; this will be understood to mean the in-order catenation of all of the *scripts* and *script-files* passed in.

Each SED command consists of an optional address or address range, followed by a one-character command name and any additional command-specific code.

3.1 Selecting lines with SED

Addresses in a SED script can be in any of the following forms:

'*number*'
: Specifying a line number will match only that line in the input. (Note that SED counts lines continuously across all input files.)

'*first*`~`*step*'
: This GNU extension matches every *step*th line starting with line *first*. In particular, lines will be selected when there exists a non-negative *n* such that the current line-number equals *first* `+` (*n* `*` *step*). Thus, to select the odd-numbered lines, one would use `1~2`; to pick every third line starting with the second, `2~3` would be used; to pick every fifth line starting with the tenth, use `10~5`; and `50~0` is just an obscure way of saying `50`.

'`$`'
: This address matches the last line of the last file of input.

'`/`*regexp*`/`'
: This will select any line which matches the regular expression *regexp*. If *regexp* itself includes any `/` characters, each must be escaped by a backslash (`\`).

'`\%`*regexp*`%`'
: (The `%` may be replaced by any other single character.)

: This also matches the regular expression *regexp*, but allows one to use a different delimiter than `/`. This is particularly useful if the *regexp* itself contains a lot of `/`s, since it avoids the tedious escaping of every `/`. If *regexp* itself includes any delimiter characters, each must be escaped by a backslash (`\`).

'`/`*regexp*`/I`'
'`\%`*regexp*`%I`'
: The `I` modifier to regular-expression matching is a GNU extension which causes the *regexp* to be matched in a case-insensitive manner.

If no addresses are given, then all lines are matched; if one address is given, then only lines matching that address are matched.

An address range can be specified by specifying two addresses separated by a comma (`,`). An address range matches lines starting from where the first address matches, and continues until the second address matches (inclusively). If the second address is a *regexp*, then checking for the ending match will start with the line *following* the line which matched the first address. If the second address is a *number* less than (or equal to) the line matching the first address, then only the one line is matched.

Appending the `!` character to the end of an address specification will negate the sense of the match. That is, if the `!` character follows an address range, then only lines which do *not* match the address range will be selected. This also works for singleton addresses, and, perhaps perversely, for the null address.

3.2 Overview of regular expression syntax

[[I may add a brief overview of regular expressions at a later date; for now see any of the various other documentations for regular expressions, such as the AWK info page.]]

3.3 Where SED buffers data

SED maintains two data buffers: the active *pattern* space, and the auxiliary *hold* space. In "normal" operation, SED reads in one line from the input stream and places it in the pattern space. This pattern space is where text manipulations occur. The hold space is initially empty, but there are commands for moving data between the pattern and hold spaces.

3.4 Often used commands

If you use SED at all, you will quite likely want to know these commands.

'`#`'
: [No addresses allowed.]

  The `#` "command" begins a comment; the comment continues until the next newline.

  If you are concerned about portability, be aware that some implementations of SED (which are not POSIX.2 conformant) may only support a single one-line comment, and then only when the very first character of the script is a `#`.

  Warning: if the first two characters of the SED script are `#n`, then the `-n` (no-autoprint) option is forced. If you want to put a comment in the first line of your script and that comment begins with the letter 'n' and you do not want this behavior, then be sure to either use a capital 'N', or place at least one space before the 'n'.

'`s/`*regexp*`/`*replacement*`/`*flags*'
: (The `/` characters may be uniformly replaced by any other single character within any given `s` command.)

  The `/` character (or whatever other character is used in its stead) can appear in the *regexp* or *replacement* only if it is preceded by a `\` character. Also newlines may appear in the *regexp* using the two character sequence `\n`.

  The `s` command attempts to match the pattern space against the supplied *regexp*. If the match is successful, then that portion of the pattern space which was matched is replaced with *replacement*.

  The *replacement* can contain `\`*n* (*n* being a number from 1 to 9, inclusive) references, which refer to the portion of the match which is contained between the *n*th `\(` and its matching `\)`. Also, the *replacement* can contain unescaped `&`

characters which will reference the whole matched portion of the pattern space. To include a literal `\`, `&`, or newline in the final replacement, be sure to precede the desired `\`, `&`, or newline in the *replacement* with a `\`.

The `s` command can be followed with zero or more of the following *flags*:

‘`g`’
: Apply the replacement to *all* matches to the *regexp*, not just the first.

‘`p`’
: If the substitution was made, then print the new pattern space.

‘*number*’
: Only replace the *number*th match of the *regexp*.

‘`w` *file-name*’
: If the substitution was made, then write out the result to the named file.

‘`I`’
: (This is a GNU extension.)

  Match *regexp* in a case-insensitive manner.

‘`q`’
: [At most one address allowed.]

  Exit SED without processing any more commands or input. Note that the current pattern space is printed if auto-print is not disabled.

‘`d`’
: Delete the pattern space; immediately start next cycle.

‘`p`’
: Print out the pattern space (to the standard output). This command is usually only used in conjunction with the `-n` command-line option.

  Note: some implementations of SED, such as this one, will double-print lines when auto-print is not disabled and the `p` command is given. Other implementations will only print the line once. Both ways conform with the POSIX.2 standard, and so neither way can be considered to be in error. Portable SED scripts should thus avoid relying on either behavior; either use the `-n` option and explicitly print what you want, or avoid use of the `p` command (and also the `p` flag to the `s` command).

‘`n`’
: If auto-print is not disabled, print the pattern space, then, regardless, replace the pattern space with the next line of input. If there is no more input then SED exits without processing any more commands.

‘`{` *commands* `}`’
: A group of commands may be enclosed between `{` and `}` characters. (The `}` must appear in a zero-address command context.) This is particularly useful when you want a group of commands to be triggered by a single address (or address-range) match.

3.5 Less frequently used commands

Though perhaps less frequently used than those in the previous section, some very small yet useful SED scripts can be built with these commands.

‘`y/`*source-chars*`/`*dest-chars*`/`’
: (The `/` characters may be uniformly replaced by any other single character within any given `y` command.)

Transliterate any characters in the pattern space which match any of the *source-chars* with the corresponding character in *dest-chars*.

Instances of the `/` (or whatever other character is used in its stead), `\`, or newlines can appear in the *source-chars* or *dest-chars* lists, provide that each instance is escaped by a `\`. The *source-chars* and *dest-chars* lists *must* contain the same number of characters (after de-escaping).

‘`a\`’
‘*text*’ [At most one address allowed.]

Queue the lines of text which follow this command (each but the last ending with a `\`, which will be removed from the output) to be output at the end of the current cycle, or when the next input line is read.

‘`i\`’
‘*text*’ [At most one address allowed.]

Immediately output the lines of text which follow this command (each but the last ending with a `\`, which will be removed from the output).

‘`c\`’
‘*text*’ Delete the lines matching the address or address-range, and output the lines of text which follow this command (each but the last ending with a `\`, which will be removed from the output) in place of the last line (or in place of each line, if no addresses were specified). A new cycle is started after this command is done, since the pattern space will have been deleted.

‘`=`’ [At most one address allowed.]

Print out the current input line number (with a trailing newline).

‘`l`’ Print the pattern space in an unambiguous form: non-printable characters (and the `\` character) are printed in C-style escaped form; long lines are split, with a trailing `\` character to indicate the split; the end of each line is marked with a `$`.

‘`r` *filename*’
[At most one address allowed.]

Queue the contents of *filename* to be read and inserted into the output stream at the end of the current cycle, or when the next input line is read. Note that if *filename* cannot be read, it is treated as if it were an empty file, without any error indication.

‘`w` *filename*’
Write the pattern space to *filename*. The *filename* will be created (or truncated) before the first input line is read; all `w` commands (including instances of `w` flag on successful `s` commands) which refer to the same *filename* are output through the same FILE stream.

‘`D`’ Delete text in the pattern space up to the first newline. If any text is left, restart cycle with the resultant pattern space (without reading a new line of input), otherwise start a normal new cycle.

'`N`' Add a newline to the pattern space, then append the next line of input to the pattern space. If there is no more input then SED exits without processing any more commands.

'`P`' Print out the portion of the pattern space up to the first newline.

'`h`' Replace the contents of the hold space with the contents of the pattern space.

'`H`' Append a newline to the contents of the hold space, and then append the contents of the pattern space to that of the hold space.

'`g`' Replace the contents of the pattern space with the contents of the hold space.

'`G`' Append a newline to the contents of the pattern space, and then append the contents of the hold space to that of the pattern space.

'`x`' Exchange the contents of the hold and pattern spaces.

3.6 Commands for die-hard SED programmers

In most cases, use of these commands indicates that you are probably better off programming in something like PERL. But occasionally one is committed to sticking with SED, and these commands can enable one to write quite convoluted scripts.

'`:` *label*' [No addresses allowed.]

Specify the location of *label* for the `b` and `t` commands. In all other respects, a no-op.

'`b` *label*' Unconditionally branch to *label*. The *label* may be omitted, in which case the next cycle is started.

'`t` *label*' Branch to *label* only if there has been a successful `s`ubstitution since the last input line was read or `t` branch was taken. The *label* may be omitted, in which case the next cycle is started.

4 Some sample scripts

[[Not this release, sorry. But check out the scripts in the testsuite directory, and the amazing dc.sed script in the top-level directory of this distribution.]]

5 About the (non-)limitations on line length

For those who want to write portable SED scripts, be aware that some implementations have been known to limit line lengths (for the pattern and hold spaces) to be no more than 4000 bytes. The POSIX.2 standard specifies that conforming SED implementations shall support at least 8192 byte line lengths. GNU SED has no built-in limit on line length; as long as SED can malloc() more (virtual) memory, it will allow lines as long as you care to feed it (or construct within it).

6 Other resources for learning about SED

In addition to several books that have been written about SED (either specifically or as chapters in books which discuss shell programming), one can find out more about SED (including suggestions of a few books) from the FAQ for the seders mailing list, available from any of:

```
http://www.dbnet.ece.ntua.gr/~george/sed/sedfaq.html
http://www.ptug.org/sed/sedfaq.htm
http://www.wollery.demon.co.uk/sedtut10.txt
```

There is an informal "seders" mailing list manually maintained by Al Aab. To subscribe, send e-mail to `af137@torfree.net` with a brief description of your interest.

7 Reporting bugs

Email bug reports to `bug-gnu-utils@gnu.org`. Be sure to include the word "sed" somewhere in the "Subject:" field.

Concept Index

This is a general index of all issues discussed in this manual, with the exception of the SED commands and command-line options.

Command and Option Index

This is an alphabetical list of all SED commands and command-line opions.

Table of Contents