sed

sed ("stream editor") is a Unix utility that parses and transforms text, using a simple, compact programming language. sed was developed from 1973 to 1974 by Lee E. McMahon of Bell Labs,[1] and is available today for most operating systems.[2] sed was based on the scripting features of the interactive editor ed ("editor", 1971) and the earlier qed ("quick editor", 1965–66). sed was one of the earliest tools to support regular expressions, and remains in use for text processing, most notably with the substitution command. Popular alternative tools for plaintext string manipulation and "stream editing" include AWK and Perl.

sed

Paradigm	scripting
Designed by	Lee E. McMahon
First appeared	1974
Implementation language	C
Website	www.gnu.org/software/sed/ (https://www.gnu.org/software/sed/)
Influenced by	
ed	
Influenced	
Perl, AWK	

Contents

History

First appearing in Version 7 Unix,[3] sed is one of the early Unix commands built for command line processing of data files. It evolved as the natural successor to the popular grep command.[4] The original motivation was an analogue of grep (g/re/p) for substitution, hence "g/re/s".[3] Foreseeing that further special-purpose programs for each command would also arise, such as g/re/d, McMahon wrote a general-purpose line-oriented stream editor, which became sed.[4] The syntax for sed, notably the use of `/` for pattern matching, and `s///` for substitution, originated with ed, the precursor to sed, which was in common use at the time,[4] and the regular expression syntax has influenced other languages, notably ECMAScript and Perl. Later, the more powerful language AWK developed, and these functioned as cousins, allowing powerful text processing to be done by shell scripts. sed and AWK are often cited as progenitors and inspiration for Perl, and influenced Perl's syntax and semantics, notably in the matching and substitution operators.

GNU sed added several new features, including in-place editing of files. *Super-sed* is an extended version of sed that includes regular expressions compatible with Perl. Another variant of sed is *minised*, originally reverse-engineered from 4.1BSD sed by Eric S. Raymond and currently maintained by René Rebe. minised was used by the GNU Project until the GNU Project wrote a new version of sed based on the new GNU regular expression library. The current minised contains some extensions to BSD sed but is not as feature-rich as GNU sed. Its advantage is that it is very fast and uses little memory. It is used on embedded systems and is the version of sed provided with Minix.

Mode of operation

sed is a line-oriented text processing utility: it reads text, line by line, from an input stream or file, into an internal buffer called the *pattern space*. Each line read starts a *cycle*. To the pattern space, sed applies one or more operations which have been specified via a *sed script*. sed implements a programming language with about 25 *commands* that specify the operations on the text. For each input line, after running the script, sed ordinarily outputs the pattern space (the line as modified by the script) and begins the cycle again with the next line. Other end-of-script behaviors are available through sed options and script commands, e.g. `d` to delete the pattern space, `q` to quit, `N` to add the next line to the pattern space immediately, and so on. Thus a sed script corresponds to the body of a loop that iterates through the lines of a stream, where the loop itself and the loop variable (the current line number) are implicit and maintained by sed.

The sed script can either be specified on the command line (`-e` option) or read from a separate file (`-f` option). Commands in the sed script may take an optional *address*, in terms of line numbers or regular expressions. The address determines when the command is run. For example, `2d` would only run the `d` (delete) command on the second input line (printing all lines but the second), while `/^ /d` would delete all lines beginning with a space. A separate special buffer, the *hold space*, may be used by a few sed commands to hold and accumulate text between cycles. sed's command language has only two variables (the "hold space" and the "pattern space") and GOTO-like branching functionality; nevertheless, the language is Turing-complete,[5][6] and esoteric sed scripts exist for games such as sokoban, arkanoid,[7] chess,[8] and tetris.[9]

A main loop executes for each line of the input stream, evaluating the sed script on each line of the input. Lines of a sed script are each a pattern-action pair, indicating what pattern to match and which action to perform, which can be recast as a conditional statement. Because the main loop, working variables (pattern space and hold space), input and output streams, and default actions (copy line to pattern space, print pattern space) are implicit, it is possible to write terse one-liner programs. For example, the sed program given by:

```
10q
```

will print the first 10 lines of input, then stop.

Usage

Substitution command

The following example shows a typical, and the most common, use of sed: substitution. This usage was indeed the original motivation for sed:[4]

```
sed 's/regexp/replacement/g' inputFileName > outputFileName
```

In some versions of sed, the expression must be preceded by `-e` to indicate that an expression follows. The `s` stands for substitute, while the `g` stands for global, which means that all matching occurrences in the line would be replaced. The regular expression (i.e. pattern) to be searched is placed after the first delimiting symbol (slash here) and the replacement follows the second symbol. Slash (`/`) is the conventional symbol, originating in the character for "search" in ed, but any other could be used to make syntax more readable if it does not occur in the pattern or replacement; this is useful to avoid "leaning toothpick syndrome".

The substitution command, which originates in search-and-replace in ed, implements simple parsing and templating. The `regexp` provides both pattern matching and saving text via sub-expressions, while the `replacement` can be either literal text, or a format string containing the characters `&` for "entire match" or the special escape sequences `\1` through `\9` for the *n*th saved sub-expression. For example, `sed -r "s/(cat|dog)s?/\1s/g"` replaces all occurrences of "cat" or "dog" with "cats" or "dogs", without duplicating an existing "s": `(cat|dog)` is the 1st (and only) saved sub-expression in the regexp, and `\1` in the format string substitutes this into the output.

Other sed commands

Besides substitution, other forms of simple processing are possible, using some 25 sed commands. For example, the following uses the *d* command to filter out lines that only contain spaces, or only contain the end of line character:

```
sed '/^ *$/d' inputFileName
```

This example uses some of the following regular expression metacharacters (sed supports the full range of regular expressions):

- The caret (^) matches the beginning of the line.
- The dollar sign ($) matches the end of the line.
- The asterisk (*) matches zero or more occurrences of the previous character.
- The plus (+) matches one or more occurrence(s) of the previous character.
- The question mark (?) matches zero or one occurrence of the previous character.
- The dot (.) matches exactly one character.

Complex sed constructs are possible, allowing it to serve as a simple, but highly specialized, programming language. Flow of control, for example, can be managed by the use of a label (a colon followed by a string) and the branch instruction `b`. An instruction `b` followed by a valid label name will move processing to the block following that label.

sed used as a filter

Under Unix, sed is often used as a filter in a pipeline:

```
generateData | sed 's/x/y/g'
```

That is, a program such as "generateData" generates data, and then sed makes the small change of replacing *x* with *y*. For example:

```
$ echo xyz xyz | sed 's/x/y/g'
yyz yyz
```

[notes 1]

File-based sed scripts

It is often useful to put several sed commands, one command per line, into a script file such as `subst.sed`, and then use the `-f` option to run the commands (such as `s/x/y/g`) from the file:

```
sed -f subst.sed inputFileName > outputFileName
```

Any number of commands may be placed into the script file, and using a script file also avoids problems with shell escaping or substitutions.

Such a script file may be made directly executable from the command line by prepending it with a "shebang line" containing the sed command and assigning the executable permission to the file. For example, a file `subst.sed` can be created with contents:

```
#!/bin/sed -f
s/x/y/g
```

The file may then be made executable by the current user with the `chmod` command:

```
chmod u+x subst.sed
```

The file may then be executed directly from the command line:

```
subst.sed inputFileName > outputFileName
```

In-place editing

The `-i` option, introduced in GNU sed, allows in-place editing of files (actually, a temporary output file is created in the background, and then the original file is replaced by the temporary file). For example:

```
sed -i 's/abc/def/' fileName
```

Examples

Hello, world! example

```
# convert input text stream to "Hello, world!"
s/.*/Hello, world!/
q
```

This "Hello, world!" script is in a file (e.g., script.txt) and invoked with `sed -f script.txt inputFileName`, where "inputFileName" is the input text file. The script changes "inputFileName" line #1 to "Hello, world!" and then quits, printing the result before sed exits. Any input lines past line #1 are not read, and not printed. So the sole output is "Hello, world!".

The example emphasizes many key characteristics of sed:

- Typical sed programs are rather short and simple.
- sed scripts can have comments (the line starting with the # symbol).
- The `s` (substitute) command is the most important sed command.
- sed allows simple programming, with commands such as `q` (quit).
- sed uses regular expressions, such as `.*` (zero or more of any character).

Other simple examples

Below follow various sed scripts; these can be executed by passing as an argument to sed, or put in a separate file and executed via `-f` or by making the script itself executable.

To replace any instance of a certain word in a file with "REDACTED", such as an IRC password, and save the result:

```
sed -i s/yourpassword/REDACTED/ ./status.chat.log
```

To delete any line containing the word "yourword" (the *address* is '/yourword/'):

```
/yourword/ d
```

To delete all instances of the word "yourword":

```
s/yourword//g
```

To delete two words from a file simultaneously:

```
s/firstword//g
s/secondword//g
```

To express the previous example on one line, such as when entering at the command line, one may join two commands via the semicolon:

```
sed "s/firstword//g; s/secondword//g" inputFileName
```

Multiline processing example

In the next example, sed, which usually only works on one line, removes newlines from sentences where the second line starts with one space. Consider the following text:

```
This is my dog,
 whose name is Frank.
This is my fish,
whose name is George.
This is my goat,
 whose name is Adam.
```

The sed script below will turn the text above into the following text. Note that the script affects only the input lines that start with a space:

```
This is my dog, whose name is Frank.
This is my fish,
whose name is George.
This is my goat, whose name is Adam.
```

The script is:

```
 N
 s/\n / /
 P
 D
```

This is explained as:

- (`N`) add the next line to the pattern space;
- (`s/\n / /`) find a new line followed by a space, replace with one space;
- (`P`) print the top line of the pattern space;
- (`D`) delete the top line from the pattern space and run the script again.

This can be expressed on a single line via semicolons:

```
sed 'N; s/\n / /; P; D' inputFileName
```

Limitations and alternatives

While simple and limited, sed is sufficiently powerful for a large number of purposes. For more sophisticated processing, more powerful languages such as AWK or Perl are used instead. These are particularly used if transforming a line in a way more complicated than a regex extracting and template replacement, though arbitrarily complicated transforms are in principle possible by using the hold buffer.

Conversely, for simpler operations, specialized Unix utilities such as grep (print lines matching a pattern), head (print the first part of a file), tail (print the last part of a file), and tr (translate or delete characters) are often preferable. For the specific tasks they are designed to carry out, such specialized utilities are usually simpler, clearer, and faster than a more general solution such as sed.

The ed/sed commands and syntax continue to be used in descendent programs, such as the text editors vi and vim. An analog to ed/sed is sam/ssam, where sam is the Plan 9 editor, and ssam is a stream interface to it, yielding functionality similar to sed.

See also

- List of Unix commands
- AWK
- tr (Unix)
- Turing tarpit

Notes

1. In command line use, the quotes around the expression are not required, and are only necessary if the shell would otherwise not interpret the expression as a single word (token). For the script `s/x/y/g` there is no ambiguity, so `generateData | sed s/x/y/g` works correctly. However, quotes are usually included for clarity, and are often necessary, notably for whitespace (e.g., `'s/x x/y y/'`). Most often single quotes are used, to avoid having the shell interpret $ as a shell variable. Double quotes are used, such as `"s/$1/$2/g"`, to allow the shell to substitute for a command line argument or other shell variable.

References

1. "The sed FAQ, Section 2.1" (http://sed.sourceforge.net/sedfaq2.html#s2.1). Retrieved 2013-05-21.
2. "The sed FAQ, Section 2.2" (http://sed.sourceforge.net/sedfaq2.html#s2.2). Retrieved 2013-05-21.
3. McIlroy, M. D. (1987). *A Research Unix reader: annotated excerpts from the Programmer's Manual, 1971–1986* (http://www.cs.dartmouth.edu/~doug/reader.pdf) (PDF) (Technical report). CSTR. Bell Labs. 139.
4. "On the Early History and Impact of Unix" (http://www.columbia.edu/~rh120/ch001j.c11). "A while later a demand arose for another special-purpose program, gres, for substitution: g/re/s. Lee McMahon undertook to write it, and soon foresaw that there would be no end to the family: g/re/d, g/re/a, etc. As his concept developed it became sed…"
5. "Implementation of a Turing Machine as Sed Script" (http://sed.sourceforge.net/grabbag/scripts/turing.txt).
6. "Turing.sed" (http://sed.sourceforge.net/grabbag/scripts/turing.sed).
7. "The $SED Home - gamez" (http://sed.sourceforge.net/#gamez).
8. "bolknote/SedChess" (https://github.com/bolknote/SedChess). *GitHub*.

9. "Sedtris, a Tetris game written for sed" (https://github.com/uuner/sedtris).

Further reading

- Bell Lab's Eighth Edition (circa 1985) Unix sed(1) manual page (http://man.cat-v.org/unix_8th/1/sed)
- GNU sed documentation (http://www.gnu.org/software/sed/manual/) or the manual page (http://www.unix.com/man-page/linux/1/sed/)
- Dale Dougherty & Arnold Robbins (March 1997). *sed & awk* (http://shop.oreilly.com/product/9781565922259.do) (2nd ed.). O'Reilly. ISBN 1-56592-225-5.
- Arnold Robbins (June 2002). *sed and awk Pocket Reference* (http://shop.oreilly.com/product/9780596003524.do) (2nd ed.). O'Reilly. ISBN 0-596-00352-8.
- Peter Patsis (December 1998). *UNIX AWK and SED Programmer's Interactive Workbook (UNIX Interactive Workbook)*. Prentice Hall. ISBN 0-13-082675-8.
- Daniel Goldman (February 2013). *Definitive Guide to sed* (https://www.ehdp.com/press/sed-book/). EHDP Press. ISBN 978-1-939824-00-4.
- Sourceforge.net (http://sed.sourceforge.net/sedfaq.html), the sed FAQ (March, 2003)

External links

- `sed (https://www.opengroup.org/onlinepubs/9699919799/utilities/sed.html)` – Commands & Utilities Reference, The Single UNIX Specification, Issue 7 from The Open Group
- `sed(1) (https://9p.io/magic/man2html/1/sed)` – Plan 9 Programmer's Manual, Volume 1

Tutorials

- Sed - An Introduction and Tutorial (http://www.grymoire.com/Unix/Sed.html), by Bruce Barnett
- SED -- A Non-interactive Text Editor (1974) (http://sed.sourceforge.net/grabbag/tutorials/sed_mcmahon.txt), by Lee E. McMahon
- 31+ Examples For Sed Linux Command In Text Manipulation (https://likegeeks.com/sed-linux/), by Mokhtar Ebrahim

Examples

- Major sources for sed scripts, files, usage (http://sed.sourceforge.net)
- Roger Chang's SED and Shell Scripts (2012) (http://main.rtfiber.com.tw/~changyj/sed/)
- Top 'sed' commands – Usage examples (http://www.shell-fu.org/lister.php?tag=sed)
- Sed command examples in Unix & Linux (http://www.techsakh.com/2016/05/12/20160512use-sed-command-linux-unix/)

Other links

- GNU sed homepage (https://www.gnu.org/software/sed/) (includes GNU sed manual)
- sed the Stream Editor (2004) (http://www.pement.org/sed/) (Eric Pement)

- sed-users Yahoo discussion group (https://groups.yahoo.com/neo/groups/sed-users/info)
- minised sed implementation (https://exactcode.com/opensource/minised/) original author Eric_S._Raymond, maintained by ExactCODE.
