sed – the Streaming EDitor+

An introduction+, by Michael Paoli

presented:

2021-03-16 BALUG.org
2022-03-17 Verizon

What is sed?

- Streaming EDitor:
  - reads from standard input (stdin) or file(s)
  - uses specified edit script/program to specify what editing to do
  - writes to standard output (stdout)
  - a programming language?

sed - invocation

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

sed - it’s a Streaming EDitor

- [*address*[,*address*]]*function*
  where *function* represents a single character command followed by any applicable arguments. The command can be preceded by blank and/or ; characters, the *function* can be preceded by blanks. A function can be preceded by a '!' character, in which case the function shall be applied if the addresses do not select the pattern space. Zero or more <blank> characters shall be accepted before the '!' character.
- sed uses Basic Regular Expressions (BREs)
- sed functions take between 0 and 2 addresses, here I’ll prefix with digit to show maximum each accepts
- *address* can be given by line number, $ for last line, or /*BRE*/ to match the specific *BRE*, for corresponding line, 2 addresses for corresponding start/stop range(s), and if no address is given where at least one is otherwise required, it defaults to all lines

sed - it’s a Streaming Editor ...

- 1a*append_text* append *append_text* to stdout
- 2c*change_text* change – delete pattern space write *change_text* to stdout, for 2 addresses do so only at end of range
- 2d delete pattern space, start next cycle
- 1i*insert_text* insert *insert_text* to stdout
- 2l list (write) pattern space in visually unambiguous form
- 2n next line, output pattern space if default output not suppressed, next line of input to pattern space

sed - it’s a Streaming Editor ...

- 2p print pattern space to stdout (this is default behavior at end of pattern space processing if not suppressed)
- 1q quit - branch to end of script and quit
- 1r*rfile* read file *rfile* and write it to stdout

sed - it’s a Streaming Editor ...

- *2s/BRE/replacement/flags*
  substitute matched *BRE* with *replacement*. Any character other than \ or newline may be used to delimit *BRE* instead of /. Within *BRE*, delimiter can be used as literal if preceded by \. In *replacement*, & not preceded by \ will be replaced by the matched *BRE*. \n not preceded by \ where n is digit 1-9, will be replaced by corresponding back-reference. Line can be split by substituting newline into it - such newline need be preceded by \.

sed - it’s a Streaming Editor ...

flags (for s function):

- n – nth (where n is 1-9) occurrence
- g – global – all occurrences in pattern space
- p – print if substitution was made
- w *wfile* – append to *wfile* if substitution was made

sed - it’s a Streaming Editor ...

- 2w*wfile* append pattern space to file *wfile*
- 2y/*string1*/*string2*/ replace all occurrences of characters in *string1* with the corresponding characters in *string2*. If a \n appears in *string1* or *string2*, it shall be handled as newline. Any character other than \ or newline may be used to delimit *string1* and *string2* instead of /. Delimiter, if not n, itself can be used as literal character by preceding with \, \\ is handled as a single literal \.
- 1= write line number to stdout
- empty/blank line is ignored
- 0#*comment* ignore # through end of line

sed - it’s a programming language?

- 2{} execute the list of sed commands within {}
- 2b*label* branch to *label* (or end of script if no *label* specified).
- 2D Delete pattern space through first newline and start next cycle with resultant pattern space without reading new input, unless no newline was in pattern space then behave like d
- 2g get from hold to pattern space
- 2G Get from hold append newline and hold to pattern space
- 2h hold pattern space to hold space
- 2H Hold pattern space append newline and pattern space to hold space

sed - it’s a programming language?

- 2N Next line if available append newline and that to pattern space, else branch to end of script and quit without starting new cycle or copying pattern space to stdout
- 2P Print pattern space up to the first newline to stdout
- 2t*label* test if any substitutions have been made since the most recent reading of an input line or execution of a t and if so branch to *label* (or end of script if no *label* specified)
- 2x exchange the pattern and hold space
- 0:*label* Do nothing. This command bears a label to which the b and t commands branch.

sed - it’s a programming language?

- So, it’s got logical grouping {}, conditional (t) and unconditional (b) branching.
- It doesn’t have general variables, but it has the pattern and hold spaces and functions to specifically utilize newlines within (DgGhHNPx), so they can very effectively be used as a pair of stacks

References and Examples

- sed per POSIX
- man(1) pages:
  - sed(1) from Debian
  - sed(1) from UNIX Seventh Edition (1979)
- GNU: sed: examples
- SourceForge: sed: books scripts games tools sedlovers
- Wikipedia: sed: examples, links to: examples tutorials
- Some of Michael Paoli’s stuff on sed

Examples

```
$ echo -e '1\n2\n3' | sed -e '2iInsert
2aAppend'
1
Insert
2
Append
3
$ echo -e '1\n2\n3' | sed -e '2,/Zebra/s/.*/Before>&<After/'
1
Before>2<After
Before>3<After
$
```

Examples

```
$ sed -ne '/^\(.\)\(.\).\2\1$/{
G;/\(\n[^\n]*\)\{5\}/{s/\n$//;s/\n/ /g;p;q};h
}' /usr/share/dict/words
madam ma'am level kayak civic
$ echo 'fJ3qnGmzbX' | sed -e 's/[a-zA-Z]/&(<-- 5th letter )/5'
fJ3qnG(<-- 5th letter )mzbX
$ ip a s | sed -ne 's!^ *inet6 \([0-9:a-f]*\)/[0-9]* scope global *$!\1!p'
2001:470:1f05:19e::2
2001:470:1f05:19e::3
2001:470:1f05:19e::4
2001:470:1f05:19e::5
2001:470:1f05:19e::6
2001:470:1f05:19e::7
2001:470:1f04:19e::2
$
```

Examples

```
$ type ttt
ttt is hashed (/home/m/michael/bin/ttt)
$ sed -ne '1{p;q}' /home/m/michael/bin/ttt
#!/usr/bin/env -S sed -nf
$ ttt
?
Help: Tic-Tac-Toe: Positions are numbered 1-9 on 3x3 board:
1|2|3 Players are X and O and alternate turns between X and O, playing
-+-+- one position per turn.  Three in a row, horizontally, vertically,
4|5|6 or diagonally wins.  X always goes first.  Players alternate X and
-+-+- O between games.
7|8|9 Enter:
1-9 - (just one digit) to make your move
N - Next game
P - Print current game positions ? basic status
Q - Quit
R - Restart game

 | |  1|2|3
-+-+- -+-+-
 | |  4|5|6
-+-+- -+-+-
 | |  7|8|9 1-9NPQR?:
```

Examples

```
1
X| |
-+-+-
 | |
-+-+-
 | |
X| |  1|2|3
-+-+- -+-+-
 |O|  4|5|6
-+-+- -+-+-
 | |  7|8|9 1-9NPQR?:
2
X|X|
-+-+-
 |O|
-+-+-
 | |
X|X|O 1|2|3
-+-+- -+-+-
 |O|  4|5|6
-+-+- -+-+-
 | |  7|8|9 1-9NPQR?:
```

```
7
X|X|O
-+-+-
 |O|
-+-+-
X| |
X|X|O 1|2|3
-+-+- -+-+-
O|O|  4|5|6
-+-+- -+-+-
X| |  7|8|9 1-9NPQR?:
6
X|X|O
-+-+-
O|O|X
-+-+-
X| |
X|X|O 1|2|3
-+-+- -+-+-
O|O|X 4|5|6
-+-+- -+-+-
X|O|  7|8|9 1-9NPQR?:
```

```
9
X|X|O
-+-+-
O|O|X
-+-+-
X|O|X
Tie game!
X| |  1|2|3
-+-+- -+-+-
 | |  4|5|6
-+-+- -+-+-
 | |  7|8|9 1-9NPQR?:
```