CST8177

sed

The Stream Editor

The original editor for Unix was called **ed**, short for editor. By today's standards, **ed** was very primitive.

Soon, **sed** was derived from it, especially for use in scripts, since being able to edit a file under script control is very powerful. Of course, **sed** also works great on the command line.

sed is called the stream editor since it works from **stdin** or from input files, and writes its output to **stdout**. That is, it works on a stream of data through **stdin**/**stdout**.

Therefore:

sed - stream editor for filtering and transforming text

```
sed [OPTION]... [{sed script}] [input-file]...
```

The options need not be used. Some of the common ones include:

`-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

`-r, --regexp-extended`

use extended regular expressions in the script.

Once of the most common uses for **sed** is to substitute one string for another, using a regular expression:

```
stream -> | sed 's/bad/good/g' | stream ->
```

This will substitute **good** for **bad** in every occurrence (**g** is for global) in the data stream.

The instruction to **sed** is **s**, for substitute. The **/** is to be used as a separator, and is just the character that follows the instruction - it can be any single character (except newline) that cannot be found in the instruction. Both **/** and **+** are common; these do the same thing:

```
stream -> | sed 's+bad+good+g' | stream ->
stream -> | sed 'sXbadXgoodXg' | stream ->
stream -> | sed 's@bad@good@g' | stream ->
```

The first item **/bad/** is the regular expression to be searched for. The second **/good/** is the substitute value. And the trailing **g** causes every **/bad/** to be replaced instead of only the first on each line.

You can combine several operations into a single sed:

```
... | sed 's/bad/good/g; s+red+green+' | ...
```

This will substitute **good** for **bad** in every occurrence (**g** is for global) in the data stream and replace the first **red** per line with **green**. Note well the semicolon.

You can also do the same with the **-e**:

```
sed -e 's/bad/good/g' -e 's+red+green+'
```

A collection of **sed** commands is known as a **sed** script, and is **not** the same as a bash script:

```
[Prompt]$ ls -l sedscr
-rw-rw-r--. 1 allisor allisor 26 (date) sedscr
[Prompt]$ cat sedscr
s/bad/good/g
s+red+green+
[Prompt]$ echo bad red bad red | sed -f ./sedscr
good green good red
```

sed command format

Commands for **sed** are in one of three forms:

1. with an optional multi-line address:

```
[address]command
```

2. with an optional single line address:

```
[line address]command
```

3. as a group with a required address:

```
address {
    command1
    command2
    command3
    ...
}
```

The third form is usually only found in **sed** scripts.

Addresses

You can address a single line with a line number. For example, to delete the first line of a file:

```
[Prompt]$ cat file1
line1
line2
[Prompt]$ sed '1d' file1
line2
[Prompt]$ sed 'd' file1  # note this!
[Prompt]$
```

The delete command is **d**, and the first example uses the address 1 with it to delete the first line. Note especially the second example, where the default is all lines!

A range of lines use comma (file2 has 3 lines):

```
[Prompt]$ sed '1,2d' file2
line3
```

Addresses, continued

You can also address lines with a regex, which must be enclosed in forward slashes **/.../**:

```
[Prompt]$ cat file2
first line
second line
line3
[Prompt]$ sed '/^line/d' file2
first line
second line
```

You can also combine line numbers and a regex as in **'1,/^$/d'** where all lines from the start to the first empty line will be deleted.

The last line of the file can be represented by **$** so that **2,$** would delete all but the first line:

```
[Prompt]$ sed '2,$d' file2
first line
```

Substitute command

[address]s!regex!replacement!flags

address	as above
s	the substitute command itself
!	the argument separator
regex	the basic or extended (with -r) regular expression
replacement	the replacement string
flags	optional: **n, g, p, w file**

You've already seen the global flag, **g**. The others are **n**, a number, requesting that the nth instance of the **regex** be replaced (default 1), **p** to print (for example, if using **-n**), and **w file** to write to the named **file** in addition to **stdout** (unless **-n**).

Substitute regex and replacement

The regex is the same as **grep**. By using the **-r** option with **sed**, you can use extended regex instead of basic regex.

You can also use tags (remember tags?) to repeat parts of the regex match into the replacement string.

The replacement string is often just a string, but you can reference tags by **\1**, **\2**, etc. Where the nth tag is taken from the matching characters of the regex:

```
Prompt$ echo "a.b 3.4" | \  # European decimal
              sed -r 's!([0-9]+)\.([0-9]+)!\1,\2!'
a.b 3,4
```

You can also use **&** in the replacement string, which uses the whole of the match in the replacement. To convert integers to be floating-point (real) values:

```
Prompt$ echo "a.b 3 4" | sed -r 's![0-9]+!&.0!g'
a.b 3.0 4.0
```

Escape **&** with **** if you do not want this to happen.

Some commands

No address

comment to the end of the line, quote, or **-e**

Zero or one address

a text append text after the current line

i text insert text before the current line

Both append and insert require that you escape all embedded newline characters.

r filename append text read from **filename**

Samples

```
[Prompt]$ cat file1
aaa
bbb
ccc
[Prompt]$ sed '2,3d' file1
aaa
[Prompt]$ sed '#2,3d' file1
aaa
bbb
ccc
```

Samples

```
[Prompt]$ sed '2ixxx' file1
aaa
xxx
bbb
ccc
[Prompt]$ sed '2ayyy' file1
aaa
bbb
yyy
ccc
```

Samples

```
[Prompt]$ cat file2
ddd
eee
fff
[Prompt$ sed '$rfile2' file1
aaa
bbb
ccc
ddd
eee
fff
```

Some commands with Address ranges

c text	replace the selected lines with text, newlines as in append/insert above
d	delete lines (default: all lines)
l	list the current line in a "visually unambiguous" form
s/regex/repl/	substitute **repl** for **regex**
y/from/to/	translate the characters in **from** to the corresponding character in **to** (like **tr**)

Samples

```
[Prompt]$ cat file3
A test line
^B
[Prompt]$ sed -n 'l' file3
A test line$
^B \002$

[Prompt]$ sed '2d' file3
A test line

[Prompt]$ echo abcdef | sed 'y/abc/123/'
123def
[Prompt]$ echo abcdef | sed 'y/def/456/'
abc456
[Prompt]$ echo abcdef | sed 'y/ace/789/'
7b8d9f
```