SED - The Stream Editor

Orr Dunkelman

orrd@vipe.technion.ac.il

Introduction

Sed is a stream editor. It gets input stream (file or pipe) and manipulates it to an output stream.

Sed is very useful when one need to manipulate a text stream quickly, with one iteration over the text (sed gets an input stream, and thus cannot return backward). Thus Sed is used in many places:

- Changing some definition in many files (like IP or computer's name)
- Adding text in the beginning or end of files (signatures or timestamps)
- Replacement of a names/titles/etc.

Introduction (cont')

Sed has three spaces - the input space, the pattern space and the hold space. The input space can be read only once (read and discard). The pattern space is what is usually would be sent to the output stream (one can avoid that). The hold space is a working buffer.

Sed's command line is:

sed $<$ *options* $>$ $<$ *script* $>$ $<$ *stream* $>$

Sed's Options

-n
–quiet
–silent - By default sed prints out the pattern space once it ends a cycle through the script. These options disable the default behavior.

-e $< script >$
-expression=$< script >$
-f $< script\_file >$
–file=$< script\_file >$ - This tells sed to add the commands in the script files to the commands to execute.

If no script file is given, sed assumes the first non-option argument to be the script to execute.

Sed's Commands - General Commands

Sed has a programming language which is easy to learn and perform.

: **label**	defines a label for other commands.
# comment	defines a comment. This line is ignored.
{ }	starts and ends a block of command.
=	print the current line number.
q	quit sed. If the auto-print was not disabled, the pattern space would be printted.

Sed’s Commands - Text Field Commands

a /

text append text.

i /

text insert text.

c /

text replace the selected lines with text.

Note that text may contain newlines. This is done by adding a / before the newline character.

Sed's Commands - Files Commands

r <u>filename</u>	append text from <u>filename</u>
w <u>filename</u>	save pattern space in <u>filename</u>

Sed's Commands - Spaces Commands

Sed has two spaces to work with - the pattern space and the hold space. Each new line is read into the pattern space, and when cycle ends, the pattern space is printed (unless the auto-print was disabled by *-n*.

d	delete pattern space (start a new cycle).
D	delete the first line in pattern space (note that pattern space might not be empty) and start a new cycle.

Sed's Commands - Spaces Commands (cont')

h	copy pattern space to hold space.
H	append pattern space to hold space.
g	copy hold space to pattern space.
G	append hold space to pattern space.
x	exchange hold space and pattern space contents.

Sed's Commands - Pattern Space Commands

l	output the pattern space in a special format Note that in the sed's manual from 7/5/98 there is mistake. The *l* command output the pattern space.
n	read the next line into the pattern space.
N	append the next line into the pattern space.
p	print the pattern space.
P	print first line of the pattern space.

Sed's Commands - The Real Stuff

So far we saw how to manipulate the lines, we didn't do anything intelligent. It is possible in sed to replace one text in another. The replacing can be from idiotic substitutions (upper-lower case) to very sophisticated (increment all integers by 1).

Basic substitutions can be done using the command *y/source/dest/*. Each character is searched in source and its corresponding character in dest is written.

For example *y/ab/BA* will change all a's to B's, and all b's to A's. This resembles the operation of the *tr* command.

Sed's Commands - The Real Stuff (cont')

The command for replacing more than on a letter-to-letter base is *s/what we want to replace/the new thing/*. For example s/thna/than/ would fix all the typos of thna to than in the given text.

But what if we want to replace all the places "Fsck you" appears to "F— you" for any capitalization of the original fsck?

We need a smart search option. This is enabled using Regular Expressions (actually, sed supports Basic Regular Expressions, BRE).

Basic Regular Expression

Regular Expression are a method of representing a group of strings which has some mathematical property. Most of the relevant life related groups of strings has this property.

Basic regular expression is composed of characters and special symbols which all has a meaning. The easiest way to define a regular expression is in a recursive manner:

character *c* - is a regular expression which represent the character *c*

. - any character.

*c** - zero or more occurrences of *c*.

c\\+ - one or more occurrences of *c*.

c\\? - zero or one occurrence of *c*.

Basic Regular Expression (cont')

c[i,j] - between i to j occurrences of c. One of i or j can be omitted, and than 0 or infinite, respectively replace the border.

[abc] - is a regular expression which represent or a or b or c.

[a-z] - is a regular expression which represent all lower case characters.

A concatenation of basic regular expressions is also a basic regular expression. The new expression can be viewed as a demand that all the conditions are satisfied. For example *[abc][0-9]*DOD\+* represent all the strings that starts with a, b or c, followed by no up to infinite number of digits followed by DO and one or more D's, like a156DODDD.

Basic Regular Expression (cont')

Basic regular expression might also contain some reference to the relative location in line.

In basic regular expressions ^ represent the beginning of the line, and $ the end of the line. For example ^.*\[,15\]A* represent the last A out of the first 16 characters. Note that once there are two possible matching of a string to a regular expression the longer one is considered (as a convention). Otherwise, the above statement on the regular expression might not be right (it might not have been the last A in the first 16 characters).

Asking for a RE or another RE can be done using \| (thus RE_1 \| RE_2 is one of the two RE).

Back to Sed - Substitute command

Sometimes we want that fn[0-9]\[0,3\]jb would become nf[0-9]\[,3\]jb, and all strings of the form two letters, up to 3 digits jb would replace the first two letters.

We can write this request as 299 substitution rules, or as one using the fact that \1 to \9 represent the sub-expressions of a basic regular expression.

Therefore, the substitution rule would be:

```
s/\([a-z]\)\([a-z]\)\([0-9]\)\[,3\]jb/ \2\1\3jb/
```

Sed's Commands - Substitute options

The substitute command can substitute all, one or several occurrences of the basic regular expression. The syntax is *s/old RE/new RE/option*, where *option* can be one of:

g	replace all occurences
	(defualt is to replace only the first).
p	write the pattern space only if substitution occured.
w <u>filename</u>	like *p* but is written to <u>filename</u>
d	replace the d'th occurence (d is a digit).

Sed's Commands - Labels

Labels are used in order to enhance sed's power.

b label	branch to label. If the label is omitted the jump is to the end of the script (end cycle).
t lable	jump to label if a successfull substitution had occured since last input line was read and last *t* command

This enables loops (like replace all leading 9's to something else).

Sed's Work Area

Sed can work on all, some, one (or less) lines for each file.

number	the follwonig command(s) would affect only line number.
first,last	match all lines between (include) lines first to last.
first~step	match every step'ts line starting with first. For example $1 \sim 2p$ would mean to print all odd-numbered lines.

Sed’s Work Area (cont’)

$	match the last line (EOF).
/<u>**regexp**</u>/	match lines matching the regular expression.
/**c<u>regexp</u>c**/	match lines matching the regular expression.
	(c can be any character).

Combining the methods is acceptable and used. For example /BR/,30 would affect all lines containing BR before (include) line 30.

One can also request to do something if the condition is not satisfied, using the ! operator. Thus, 2,31! would mean the command is applicable for all lines but 2 to 31.

Example of a Sed Script

The following script is reversing the order of characters in each line:

```
/./!b      # If empty line continue to read next

s/!/!!/g   # replace all ! to !! for markers
s/^/-!-/   # mark beginning of line
s/$/-!-/   # mark the end of line

:a
s/-!-\([^!]\|!!\)\(.*\)\([ ^!]\|!!\)-!-/\3-!-\2-!-\1/
ta

s/-!-//g
s/!!/!/g
```

Errata

There is an error in the previous script. Different from regular programming language, if there was a successful substitute and afterward an unsuccessful one, the *t* command would be executed!

The solution is to clear the “t-flag” by adding ta in after the :a. This way, if there was a substitution before it would be removed, and than the s/// command would be executed, and update the “t-flag” if a substitute would occur.

References

[1] Carlos Jorge G.uarte, *do-it-with-sed*,
http://www.dbnet.ece.ntua.gr/∼george/sed/sedtut_1.html.

[2] Eric Pement *FAQ about SED, the stream editor*,
http://www.ptug.org/sed/sedfaq.htm/