13 Advanced filter utility –II: sed

13.1 Overview

So far, we discussed many filter utilities. In this chapter, we will discuss multi-purpose filter utility known as **sed**. The term **sed** stands for **stream editor** and was designed by Lee McMohan. Stream editor i.e. **sed** is derived from **ed**, known as line editor. Everything in **sed** is an instruction. The general form of this utility is:

Syntax:

sed [options] instruction [filename(s)]

An *instruction* consists of two components an *address* and a *command* which are enclosed within quotes. The *address* selects/searches the line to be processed or not processed by the *command*. The *command* indicates the action that **sed** is to apply to each input line that matches the address.

13.2 Addresses

An address may be a line number(s), pattern(s) or combination of them. The **sed** supports two types of addresses.

- ✓ Line number.
- ✓ A pattern.

Line address can be a single line, a set of line or range of line. A single line address can be defined by line number or a dollar ($). A dollar is a special symbol and it specifies the last line in the input file. Some of the examples of a single line address are given in table-(a.13):

Table-(a.13): Example of single line address

address	Meaning
5command	It applies *command* on 5th line.
$command	It applies *command* on last line.
15command	It applies *command* on 15th line.

A set of line address allows you to specify more than two lines and may be consecutive or alternate in the input file. A set of line address can be defined by a pattern or regular expression. When you use pattern or regular expression as line address then it must be enclosed within front-slashes. A process of specifying a pattern within the slashes is known as *context address*. Table-(b.13) denotes the example of set of line address used by a pattern.

Table-(b.13): Example of set of line address

/pattern/command	It applies *command* on lines that contains *pattern*.
/^pattern/command	It applies *command* on lines which begins with *pattern*.
/pattern$/command	It applies *command* on lines which ends with *pattern*.

If, a user want to access a set of consecutive lines of input files then range of address is given. You can define range of address by start address followed by comma with no space followed by end address. The start address and end address may be line number or a pattern or combination of them. Table-(c.13) shows example of range addresses:

Table-(c.13): Example of range of address

Address	Meaning
5,10command	It applies *command* on lines between 5 and 10 of input file.
5,$command	It applies *command* on lines between 5 and last line of input file.
2,/pattern/command	It applies *command* on 2nd line up to first occurrences of line which contains *pattern*.
/pattern/,10command	It applies *command* on line which contains *pattern* up to 10th lines of input file.
/pattern1/,/pattern2/command	It applies *command* on those lines which are occurring between *pattern1* and *pattern2* of input files.

NOTE: sed does not change the input file. All modified output is written to standard output and to be saved must be redirected to a file.

13.3 Commands

The **sed** support several commands. Commands are used to apply action on specified lines. They are categorized as follow:

- ✓ Print
- ✓ Quit
- ✓ Line number
- ✓ Modify
- ✓ Files
- ✓ Substitute

(I) print(p) command:

It is denoted by character **p**. It prints selected lines on a standard output. Consider an input file *f1* as follow:

```
$cat f1
unix and shell programming
red hat linux
linux and shell programming
unix operating system
linux is open source
$
```

✓ If a user wants to print top 3-lines of input file then the command is like this:

```
$sed '1,3p' f1
unix and shell programming
unix and shell programming
red hat linux
red hat linux
linux and shell programming
linux and shell programming
unix operating system
linux is open source
$
```

The output shows that by default **sed** utility reads entire file so it prints all lines on the standard output as well as specified lines that are affected by the command **p**. In other words, the addressed lines are printed twice. To overcome the problem of printing duplicate lines, you can use **–n** option whenever you use the p command. Therefore the above command is rewritten as follow:

```
$sed–n '1,3p' f1
unix and shell programming
red hat linux
linux and shell programming
$
```

✓ A special character dollar ($) is used to print last line of an input file. For example, to print the last line of file *f1* then the command is:

```
$sed–n '$p' f1
linux is open source
$
```

✓ A command **p** without any address, displays all lines of input file by default.

```
$sed–n p f1
```

It displays all lines of input file *f1*.

✓ *Reversing line selection criteria (!):* A user can use negation operator (i.e. !) with any command of **sed** utility. So, selecting first 3-lines means not selecting lines from fourth line to last line of input file. Therefore the command is:

```
$sed -n '4,$!p' file1     OR          sed–n '1,3p' file1
```

✓ To select non-contiguous groups of lines of input file then the command is as follow:

```
$sed–n '1,3p
> 7,9p                 #It select lines 1 to 3, 7 to 9
> $p' f1          # and last line of file f1
```

It displays 1 to 3, 7 to 9 and last lines of input file *f1*.

✓ In addition, a user can get similar output by inclnding inline script/expression using **–e** option. This option allows user to include multiple instruction with **sed** utility. Therefore, the above command is rewritten in a single line as follow:

```
$sed–ne '1,3p' –e '7,9p' –e '$p' file1
```

✓ A user can also use variable substitution in an instruction part. Consider two variables *a* and *b which* contains value 5 and 10 respectively. So, to select lines 5 to 10 then the command is:

```
$sed–n '$a, $b p' f1
```

✓ You can also use pattern and line address in instruction part. For example, consider the command as follow:

```
$sed -ne '1,3p' -e '/hello/p' f1
```

It prints top 3-lines and the lines which contains pattern *hello* of file *f1*.

✓ You can give range of pattern instead of range of line numbers in instruction part. To prints all lines in which it start from the line which contains pattern *unix* up to a line which contains pattern *linux* then the command is as follow:

```
$sed–n '/unix/,/linux/p' f1
```

(ii) quit(q) command:

This command is denoted by a character **q**. It uses single address i.e. it does not allow range of address. It quits after reading up to address line. For example, to quit after 3rd line of file *f1* then the command is:

```
$sed '3q' f1
unix and shell programming
red hat linux
linux and shell programming
$
```

✓ If, you do not specify any address with **q** command then it quits after 1st line of input file.

```
$sed q f1
unix and shell programming
$
```

It prints first line of file *f1*.

✓ It also accept pattern/regular expression as address like this:

```
$sed '/unix/q' f1
unix and shell programming
$
```

It quits from file *f1* on encounter of line that contains *unix* pattern.

(iii) Line number (=) command:

It is denoted by equal sign (i.e. =). It writes line number of addressed line at the beginning of the line. It is similar to **–n** option of **grep** utility, but the difference is that the line number is written on a separate line.

```
$sed '=' f1
         1
         unix and shell programming
         2
         red hat linux
...and so on
$
```

✓ To print only line number then **–n** option is used.

```
$sed–n '=' f1
```

It prints only line number i.e. it does not display content of line.

✓ To print line number of last line of input file then the command is:

```
$sed–n '$=' f1
5
$
```

It is similar to the command **wc –l < f1**. In other words, it displays number of lines in an input file *f1*.

(iv) Modify command:

There are different purposes of this command. It allows you to insert, append, change or delete lines. They do not modify just a part of a line that means they work on entire line. The modify commands are as follow:

(a) Insert command(i):

It is denoted by a character **i**. It inserts one or more lines directly to the output before the address lines.

✓ For example, to insert two lines at the beginning of an input file then the command is as follow:

```
$sed '1i\<enter>
Unix is multi-user multi-tasking operating system<enter>
It provides 3-levels of security'  f1 <enter>
```

Then the output is:

```
Unix is multi-user multi-tasking operating system
It provides 3-levels of security
unix and shell programming
red hat linux
linux and shell programming
unix operating system
linux is open source
$
```

NOTE: The escape character (\) must be immediately followed by a return. If any other character, including a space, follows it, the escape character modifies that character and **sed** will return an error rather than processing your commands.

✓ Insert blank-line before each line of an input file then the command is:

```
$sed 'i\ <enter>        or        $sed 'i\ <enter>
><enter>                          > 'f1
>'f1
```

It inserts blank-lines before each line of input file *f1*.

(b) Append command(a):

It is denoted by a character **a**. It is similar to the insert command except that it writes the text directly to the output after the specified line.

✓ Insert 2-lines at the end of file *f1* then the command is:

```
$sed '$a\<enter>
>unix is portable operating system \<enter>
> It is designed to facilitate programming, text processing and comm.
> 'f1
```

✓ A user want to redirect the output of a command in to another file then the command is:

```
$sed '$a\<enter>
>unix is portable operating system \<enter>
> It is designed to facilitate programming, text processing and comm.
> 'f1 > f1.out
```

It creates an output file *f1.out*.

(c) Change command (c):

A change command is denoted by a character **c**. It replaces addressed/matched line with new text.

✓ To change the first line of file *f1*, you can write command as:

```
$sed '1c\<enter>
> unix & shell programming 'f1
$
```

It replaces 1st line of file *f1* with a text *'unix & shell programming'*.

NOTE: back-slash (\) is compulsory at the end of each line (for new-line character) when we used either a or i or c command.

✓ Insert a text *'introduction'* before 1st line and *'hello world'* after 3rd line then we can use **i** and **a** commands like this:

```
$sed  -e '1i\<enter>
> introduction'  -e '3a\<enter>
>hello world'  file1
```

(d) Delete command (d):
Using the **d** (delete) command, we can simulate **–v** option of **grep** utility to select lines not containing a pattern.

✓ Removes lines of file *f1* that contains *unix* pattern then the command is:

```
$sed '/unix/d' f1          Or      $sed '/unix/!p' f1
red hat linux
linux and shell programming
linux is open source
$
```

It selects lines that do not contain *unix* pattern.

✓ A user can remove blank-line from an input file as follow:

```
$sed '/^$/d' f1
```

It displays non-blank lines of file *f1*.

✓ Following two commands are equivalent.

```
sed '!d' f1              Or      sed 'p' f1
```

It displays all lines of file *f1*.

(v) File command:
File command is used to read or write data to or from other files respectively. There are two types of file command: (i) read file and (ii) write file

(a) readfile command(r fname):
It is denoted by **r fname**. When a user wants to insert common content of a file after specified line of an input file then this command is useful. It reads text from file *fname* and places its content after a specified line of input file.

✓ To append contents of file *f2* after lines of file *f1* that contains *unix* pattern then the command is:

```
$ sed  '/unix/r f2'  f1
```

✓ To insert content of file *f2* at the end of input file *f1* then the command is:

```
$sed  '$ r f2' f1
```

It is similar to **cat f1 f2** command.

✓ To insert content of file *f2* at the beginning of input file *f1* then the command is:

```
$sed '1 r f2' f1
```

✓ To insert contents of file *f3* after each line of input file *f2* except first line then the command is:

```
$sed  '1!r f3' f2
```

(b) Write File command(w fname):

It is denoted by **w fname**. The write file command makes possible to write the selected lines in a separate file.

✓ To write selected lines of input file *f1* to output file *f1.out* then the command is as follow:

$sed –n '/unix/w f1.out' f1

It writes lines of file *f1* to file *f1.out* that contains *unix* pattern.

✓ To write top 5-lines of input file *f1* to output file *f2* then the command is like this:

$sed '1,5w f2' f1

✓ A user can create multiple output files that contain selected lines of input file.

$sed –n '/linux/w lfile <enter>
> /unix/w ufile ' f1
Or
sed -ne '/linux/w lfile' -e '/unix/w ufile' f1

It writes lines that contains pattern *linux* to a file *lfile* and lines that contains pattern ***unix*** to a file *ufile*.

Take instruction from a file: -f option

It is possible to take instruction from a file rather than the command-line. The **–f** option allows us to take instruction from a file. When there are numerous editing instructions to be performed, it will be better to use the **–f** option to accept instructions from a file.

✓ For example, consider an instruction file as follow:

$cat instr.txt
/unix/w ulist
/linux/w llist
$

Now, you can use this instruction using **–f** option of **sed** utility as follow:

$sed –n –f instr.txt f1

It creates two output files *ulist* and *llist* which contains lines having *unix* pattern and *linux* pattern respectively.

✓ A user can also use more than one instruction files by repeating **–f** option with each instruction file like this:

$sed –n –f instr1.txt –f instr2.txt f1

✓ You can combine the **–e** and **–f** options as many times as you want.

$sed –ne '/linux/p' –f instr.txt –f instr2.txt file1

(vi) Substitute command(s):

It is denoted by a character **s**. It scans a line for search pattern and substitutes it with replacement string. This is the most powerful command in **sed** utility. This command is similar to the search and replace feature found in text editor. This feature provides us to add, delete or change text in one or more lines. The format of the substitution command is as follow:

[address or scanned_pattern] s/search_pattern/replace_string/[flag(s)]

If the *address* is not specified, the substitution will be performs for all lines containing first occurrences of *search_ pattern*. A *search_pattern* may be a regular expression or literal string. Both *search_pattern* and *replace_string* are delimited by slash (/). The *replace_string* is a string that consists of either ordinary characters or an atom or meta-characters or combination of them. Only a back reference atom and meta-characters such as ampersand (&) and back slash (\) can be used in a *replace_string*. We will discuss these tokens later on in this section.

✓ To replaces first occurrences of word *unix* in each line by word *linux* in a file *f1* then the command is:

```
$sed 's/unix/linux/' f1
```

✓ To replaces first occurrences of word *unix* in each line by word *linux* in top 5-lines of file *f1* then the command is:

```
$sed '1,5s/unix/linux/' f1
```

Flag (g):
We know that the following command replaces first occurrences of the *unix* by *linux* in each line of an input file *f1* then the command is as follow:

```
$sed -n 's/unix/linux/p' f1
```

To replace all occurrences *unix* with *linux*, a user need to use the g(global) flag at the end of the instruction. This is referred to as *global substitution.* A global (g) flag replaces all occurrences of *search_pattern* with *replace_string*.

✓ For example, to replace all occurrences of *unix* with *linux* in each line of file *f1* then the command is:

```
$sed 's/unix/linux/g' f1
```

✓ A user can also use context address in *address* part.

```
$sed -n '/unix/,/linux/s/hello/bye/gp' f1
```

It replaces all occurrences of *hello* with *bye* in selected lines. Here replacement occurs between the start line which contains a pattern *unix* up to the line which contains a pattern *linux*.

Remembered pattern:
Sometimes, an address pattern is similar to *search_pattern* in other words scanned pattern and *search_pattern* are same then we can ignore *search_pattern* in an instruction part. For example, user wishes replace word *unix* with word *linux* in those lines of file *f1* that contains *unix* pattern then the command is:

```
$sed '/unix/s/unix/linux/' f1
```

In this example, both address pattern and *search_pattern* are same. So, if you ignore *search_pattern* then the above command is re-written as follow:

```
$sed '/unix/s//linux/' f1
```

The two front slashes (i.e. //) represents an empty or null regular expression which is interpreted as the *scanned_pattern* and *search_pattern* are the same. We will call it the *remembered pattern*.
Therefore, another alternative to write the above command is like this:

```
$sed 's/unix/linux/' file1
```

✓ However, when a user can use // in the *replace_string* then it removes the pattern from the output. For example, to remove all *unix* words from file *f1* then the command is:

```
$sed 's/unix//g' f1        Or        $sed '/unix/s///g' file
```

✓ Sometimes, an address pattern, *search_pattern* and *replace_string* may also be different string. For example, consider the following command:

$sed –n '/The unix/s/unix/UNIX/gp' f1

It selects lines that contains pattern *The unix* (i.e. address pattern) and replace each *unix* (i.e. *search_pattern*) with *UNIX* (i.e *replace_string*).

Moreover, **sed** utility also uses regular expression in a *search_pattern* to be substituted. Some of them are listed bellow:

✓ Repeat pattern
✓ Braces Regular Expression (BRE) or Interval Regular Expression (IRE)
✓ Tagged Regular Expression (TRE)

Repeat pattern:
There might be a situation when a *search_pattern* occurs in a *replace_string*. To repeat *search_pattern* in *replace_string* a special meta-character ampersand (i.e. &) is used. For example, to replace pattern *director* with *in-charge director* then we can write command as follow:

$sed 's/director/ in-charge &/' f1

Other alternatives are as follow:

$sed 's/director/ in-charge director/' file1
$sed '/director/s// in-charge &/' file1

In above example, & (ampersand) is known as repeat pattern operator that expands to the entire *search_pattern*.

✓ Display the list of files of working directory which have write permission set to either group or others then the command is as follow:

$ls –l|grep "^.\{5,8\}w"

BRE or IRE
Sometimes, a user wishes to print those lines that containing characters that occurs number of times in a line or locate fixed length of lines. This is possible with BRE or IRE. So we can define BRE or IRE as it is an expression that consists of character and a single or pair of numbers enclosed within a pair of escape curly braces (i.e. \{ \}).

This expression derived from **ed**, and takes the four forms as follow:

(i) ch\{m\} : It indicates that character *ch* occurs m-times.
(ii) ch\{m,n\} : It indicates that character *ch* occurs between m and n times.
(iii) ch\{m,\} : It indicates that character *ch* occurs at least m-times
(iv) ch\{,n\} : It indicates that character *ch* occurs at most n-times.

A character *ch* may be a single character regular expression. It can be ordinary character, dot (.) or character class followed by a pair of escaped curly braces, containing either a single number m, or a range of numbers lying between m and n to determine the number of times the character *ch* can occur. The value of m and n cannot exceed 255.

✓ Instead of write 50-dots to locate lines having more than 50-characters, we can use IRE as follow:

$sed –n '/.\{51\}/p' f1

It prints all lines longer than 50 characters. Here the expression \{51\} specifies that the any character (i.e. dot for any character) has to occur 51 times.

✓ To display all lines having length between 51 and 100 characters then the command is:

```
$sed -n '/^.\{51,100\}$/p'f1
```

✓ Display all lines having length of at least 50-characters.

```
$sed–n '/.\{50,\}/p'f1 Or      sed -n '/.\{50\}/p' f1
```

✓ Display a lines that consist of only alphabets then the command is:

```
$sed–n '/^[a-zA-Z]\{1,\}$/p'f1
```

✓ To replace all consecutive spaces by single space, use the regular expression as follows:

```
$sed–n 's/[ ]\{2,\}/ /gp'f1
```

Here only affected lines will be displayed.

✓ If we omit **–n** option & **p** command then it display affected lines as well as unaffected lines and the command is as follow:

```
$sed 's/[ ]\{2,\}/ /g'f1
```

Tagged Regular expression (TRE):
We know that remember pattern repeats entire *search_pattern* in *replace_string*. To repeat just a part of the *search_pattern* in *replace_string* TRE is used. TRE is an expression which groups a *search_pattern* with a pair of escape parenthesis (i. e. \(\)) and represents these group in *replace_string* with back reference (i.e. \1 up to \9). That means, the first group is represented as \1, second group is \2 and so forth.

✓ For example, a user wishes to replace the word *new line* by *new-line*. Then the command is

```
$echo "new line" | sed 's/\(new\) \(line\)/\1-\2/'
```

Here, we have two tagged patterns *\(new\)* and *\(line\)* in the *search_pattern*. They are automatically reproduced in the *replace_string* back references \1 and \2, respectively. Each escaped pattern is called a Tagged Regular Expression (TRE).

✓ To convert date in the format mm/dd/yy in to dd-mm-yy then we can write command as

```
$sed 's/\(..\)\/\(..\)\/\(..\)/\2-\1-\3/'f1
```

✓ To replace 'Unix shell programming' with '*Unix & shell programming*' then the command is:

```
$sed 's/Unix shell programming/Unix \& shell programming/'f1
                    Or
$sed 's/\(Unix\) \(shell programming\)/\1 \& \2/'f1
```

The *search_pattern* of **sed** utility uses only a subset of the regular expression atoms and patterns. The allowable atoms are listed in table-(d.13).

Table-(d.13): atoms used by sed utility

Atoms	Allowed
Character	v
Dot	v
Class	v
Anchors	^ and $
Back Reference	v

The last column of table shows that **sed** utility supports all atoms except two anchors \< and \>.

Similarly, a list of operators supported by sed utility is given in table-(e.13).

Table-(e.13): operators used by sed utility

Operators	Allowed
Sequence	v
Repetition	v
Alternation	X
Group	X
Save	v

The second column of table shows that **sed** utility supports all operators except group and alternation.