

Unix / Linux - Regular Expressions with SED

In this chapter, we will discuss in detail about regular expressions with SED in Unix.

A regular expression is a string that can be used to describe several sequences of characters. Regular expressions are used by several different Unix commands, including **ed**, **sed**, **awk**, **grep**, and to a more limited extent, **vi**.

Here **SED** stands for **s**tream **e**ditor. This stream-oriented editor was created exclusively for executing scripts. Thus, all the input you feed into it passes through and goes to STDOUT and it does not change the input file.

Invoking sed

Before we start, let us ensure we have a local copy of **/etc/passwd** text file to work with **sed**.

As mentioned previously, **sed** can be invoked by sending data through a pipe to it as follows –

```
$ cat /etc/passwd | sed
Usage: sed [OPTION]... {script-other-script} [input-file]...

-n, --quiet, --silent
            suppress automatic printing of pattern space
-e script, --expression = script
.....
```

The **cat** command dumps the contents of **/etc/passwd** to **sed** through the pipe into **sed**'s pattern space. The pattern space is the internal work buffer that **sed** uses for its operations.

The sed General Syntax

Following is the general syntax for **sed** –

```
/pattern/action
```

Here, **pattern** is a regular expression, and **action** is one of the commands given in the following table. If **pattern** is omitted, **action** is performed for every line as we have seen above.

The slash character (/) that surrounds the pattern are required because they are used as delimiters.

Sr.No.	Range & Description
1	p Prints the line
2	d Deletes the line
3	s/pattern1/pattern2/ Substitutes the first occurrence of pattern1 with pattern2

Deleting All Lines with sed

We will now understand how to delete all lines with sed. Invoke sed again; but the sed is now supposed to use the **editing command delete line**, denoted by the single letter **d** –

```
$ cat /etc/passwd | sed 'd'
$
```

Instead of invoking sed by sending a file to it through a pipe, the sed can be instructed to read the data from a file, as in the following example.

The following command does exactly the same as in the previous example, without the cat command –

```
$ sed -e 'd' /etc/passwd
$
```

The sed Addresses

The sed also supports addresses. Addresses are either particular locations in a file or a range where a particular editing command should be applied. When the sed encounters no addresses, it performs its operations on every line in the file.

The following command adds a basic address to the sed command you've been using –

```
$ cat /etc/passwd | sed '1d' | more
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
```

```
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
$
```

Notice that the number 1 is added before the **delete edit** command. This instructs the sed to perform the editing command on the first line of the file. In this example, the sed will delete the first line of **/etc/passwd** and print the rest of the file.

The sed Address Ranges

We will now understand how to work with **the sed address ranges**. So what if you want to remove more than one line from a file? You can specify an address range with sed as follows –

```
$ cat /etc/passwd | sed '1, 5d' |more
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
$
```

The above command will be applied on all the lines starting from 1 through 5. This deletes the first five lines.

Try out the following address ranges –

Sr.No.	Range & Description
1	'4,10d' Lines starting from the 4 th till the 10 th are deleted
2	'10,4d' Only 10 th line is deleted, because the sed does not work in reverse direction
3	'4,+5d' This matches line 4 in the file, deletes that line, continues to delete the next five lines, and then ceases its deletion and prints the rest
4	'2,5!d' This deletes everything except starting from 2 nd till 5 th line
5	'1~3d' This deletes the first line, steps over the next three lines, and then deletes the fourth line. Sed continues to apply this pattern until the end of the file.
6	'2~2d' This tells sed to delete the second line, step over the next line, delete the next line, and repeat until the end of the file is reached
7	'4,10p' Lines starting from 4 th till 10 th are printed
8	'4,d' This generates the syntax error
9	','10d' This would also generate syntax error

Note – While using the **p** action, you should use the **-n** option to avoid repetition of line printing. Check the difference in between the following two commands –

```
$ cat /etc/passwd | sed -n '1,3p'
```

Check the above command without -n as follows –

```
$ cat /etc/passwd | sed '1,3p'
```

The Substitution Command

The substitution command, denoted by **s**, will substitute any string that you specify with any other string that you specify.

To substitute one string with another, the sed needs to have the information on where the first string ends and the substitution string begins. For this, we proceed with bookending the two strings with the forward slash (/) character.

The following command substitutes the first occurrence on a line of the string **root** with the string **amrood**.

```
$ cat /etc/passwd | sed 's/root/amrood/'
amrood:x:0:0:root user:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
.....
```

It is very important to note that sed substitutes only the first occurrence on a line. If the string root occurs more than once on a line only the first match will be replaced.

For the sed to perform a global substitution, add the letter **g** to the end of the command as follows –

```
$ cat /etc/passwd | sed 's/root/amrood/g'
amrood:x:0:0:amrood user:/amrood:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
.....
```

Substitution Flags

There are a number of other useful flags that can be passed in addition to the **g** flag, and you can specify more than one at a time.

Sr.No.	Flag & Description
1	g Replaces all matches, not just the first match
2	NUMBER Replaces only NUMBER th match
3	p If substitution was made, then prints the pattern space
4	w FILENAME If substitution was made, then writes result to FILENAME
5	I or i Matches in a case-insensitive manner
6	M or m In addition to the normal behavior of the special regular expression characters ^ and \$, this flag causes ^ to match the empty string after a newline and \$ to match the empty string before a newline

Using an Alternative String Separator

Suppose you have to do a substitution on a string that includes the forward slash character. In this case, you can specify a different separator by providing the designated character after the **s**.

```
$ cat /etc/passwd | sed 's:/root:/amrood:g'
amrood:x:0:0:amrood user:/amrood:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

In the above example, we have used **:** as the **delimiter** instead of slash **/** because we were trying to search **/root** instead of the simple root.

Replacing with Empty Space

Use an empty substitution string to delete the root string from the **/etc/passwd** file entirely –

```
$ cat /etc/passwd | sed 's/root//g'
:x:0:0:::/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

Address Substitution

If you want to substitute the string **sh** with the string **quiet** only on line 10, you can specify it as follows –

```
$ cat /etc/passwd | sed '10s/sh/quiet/g'
root:x:0:0:root user:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/quiet
```

Similarly, to do an address range substitution, you could do something like the following –

```
$ cat /etc/passwd | sed '1,5s/sh/quiet/g'
root:x:0:0:root user:/root:/bin/quiet
daemon:x:1:1:daemon:/usr/sbin:/bin/quiet
bin:x:2:2:bin:/bin:/bin/quiet
sys:x:3:3:sys:/dev:/bin/quiet
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

As you can see from the output, the first five lines had the string **sh** changed to **quiet**, but the rest of the lines were left untouched.

The Matching Command

You would use the **p** option along with the **-n** option to print all the matching lines as follows –

```
$ cat testing | sed -n '/root/p'
root:x:0:0:root user:/root:/bin/sh
[root@ip-72-167-112-17 amrood]# vi testing
root:x:0:0:root user:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

```
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

Using Regular Expression

While matching patterns, you can use the regular expression which provides more flexibility.

Check the following example which matches all the lines starting with *daemon* and then deletes them –

```
$ cat testing | sed '/^daemon/d'
root:x:0:0:root user:/root:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

Following is the example which deletes all the lines ending with **sh** –

```
$ cat testing | sed '/sh$/d'
sync:x:4:65534:sync:/bin:/bin/sync
```

The following table lists four special characters that are very useful in regular expressions.

Sr.No.	Character & Description
1	^ Matches the beginning of lines
2	\$ Matches the end of lines
3	. Matches any single character
4	* Matches zero or more occurrences of the previous character
5	[chars] Matches any one of the characters given in chars, where chars is a sequence of characters. You can use the - character to indicate a range of characters.

Matching Characters

Look at a few more expressions to demonstrate the use of **metacharacters**. For example, the following pattern –

Sr.No.	Expression & Description
1	/a.c/ Matches lines that contain strings such as a+c , a-c , abc , match , and a3c
2	/a*c/ Matches the same strings along with strings such as ace , yacc , and arctic
3	/[tT]he/ Matches the string The and the
4	/^\$/ Matches blank lines
5	/^.*\$/ Matches an entire line whatever it is
6	/ */ Matches one or more spaces
7	/^\$/ Matches blank lines

Following table shows some frequently used sets of characters –

Sr.No.	Set & Description
1	[a-z] Matches a single lowercase letter
2	[A-Z] Matches a single uppercase letter
3	[a-zA-Z] Matches a single letter
4	[0-9] Matches a single number
5	[a-zA-Z0-9] Matches a single letter or number

Character Class Keywords

Some special keywords are commonly available to **regexps**, especially GNU utilities that employ **regexps**. These are very useful for sed regular expressions as they simplify things and enhance readability.

For example, the characters **a through z** and the characters **A through Z**, constitute one such class of characters that has the keyword **[:alpha:]**

Using the alphabet character class keyword, this command prints only those lines in the **/etc/syslog.conf** file that start with a letter of the alphabet –

```
$ cat /etc/syslog.conf | sed -n '/^[:alpha:]/p'
authpriv.*          /var/log/secure
mail.*               -/var/log/maillog
cron.*               /var/log/cron
uucp,news.crit       /var/log/spooler
local7.*             /var/log/boot.log
```

The following table is a complete list of the available character class keywords in GNU sed.

Sr.No.	Character Class & Description
1	[[:alnum:]] Alphanumeric [a-z A-Z 0-9]
2	[[:alpha:]] Alphabetic [a-z A-Z]
3	[[:blank:]] Blank characters (spaces or tabs)
4	[[:cntrl:]] Control characters
5	[[:digit:]] Numbers [0-9]
6	[[:graph:]] Any visible characters (excludes whitespace)
7	[[:lower:]] Lowercase letters [a-z]
8	[[:print:]] Printable characters (non-control characters)
9	[[:punct:]] Punctuation characters
10	[[:space:]] Whitespace
11	[[:upper:]] Uppercase letters [A-Z]
12	[[:xdigit:]]

Hex digits [0-9 a-f A-F]

Aampersand Referencing

The **sed metacharacter &** represents the contents of the pattern that was matched. For instance, say you have a file called **phone.txt** full of phone numbers, such as the following –

```
5555551212
5555551213
5555551214
6665551215
6665551216
7775551217
```

You want to make the **area code** (the first three digits) surrounded by parentheses for easier reading. To do this, you can use the ampersand replacement character –

```
$ sed -e 's/^[[:digit:]][[:digit:]][[:digit:]]/(&)/g' phone.txt
(555)5551212
(555)5551213
(555)5551214
(666)5551215

(666)5551216
(777)5551217
```

Here in the pattern part you are matching the first 3 digits and then using **&** you are replacing those 3 digits with the surrounding **parentheses**.

Using Multiple sed Commands

You can use multiple sed commands in a single sed command as follows –

```
$ sed -e 'command1' -e 'command2' ... -e 'commandN' files
```

Here **command1** through **commandN** are sed commands of the type discussed previously. These commands are applied to each of the lines in the list of files given by files.

Using the same mechanism, we can write the above phone number example as follows –

```
$ sed -e 's/^[[:digit:]]\{3\}/(&)/g' \
-e 's/)[[:digit:]]\{3\}/&-/g' phone.txt
(555)555-1212
(555)555-1213
(555)555-1214
(666)555-1215
```

```
(666)555-1216  
(777)555-1217
```

Note – In the above example, instead of repeating the character class keyword `[[:digit:]]` three times, we replaced it with `\{3\}`, which means the preceding regular expression is matched three times. We have also used `\` to give line break and this has to be removed before the command is run.

Back References

The **ampersand metacharacter** is useful, but even more useful is the ability to define specific regions in regular expressions. These special regions can be used as reference in your replacement strings. By defining specific parts of a regular expression, you can then refer back to those parts with a special reference character.

To do **back references**, you have to first define a region and then refer back to that region. To define a region, you insert **backslashed parentheses** around each region of interest. The first region that you surround with backslashes is then referenced by `\1`, the second region by `\2`, and so on.

Assuming **phone.txt** has the following text –

```
(555)555-1212  
(555)555-1213  
(555)555-1214  
(666)555-1215  
(666)555-1216  
(777)555-1217
```

Try the following command –

```
$ cat phone.txt | sed 's/\(.*\)\\\(.*-\\\)\\(.*$\\)/Area \\  
code: \1 Second: \2 Third: \3/'  
Area code: (555) Second: 555- Third: 1212  
Area code: (555) Second: 555- Third: 1213  
Area code: (555) Second: 555- Third: 1214  
Area code: (666) Second: 555- Third: 1215  
Area code: (666) Second: 555- Third: 1216  
Area code: (777) Second: 555- Third: 1217
```

Note – In the above example, each regular expression inside the parenthesis would be back referenced by `\1`, `\2` and so on. We have used `\` to give line break here. This should be removed before running the command.