Regular Expression in Unix / Linux

A regular expression is a string that can be used to describe different sequences (arrangement) of characters. Regular Expression is often used by various Unix commands, including ed, sed, awk, grep and micro domains.

A regular expression is a string that can be used to describe different sequences (arrangement) of characters. Regular Expression is often used by various Unix commands, including ed, sed, awk, grep and **micro** domains.

This chapter will show you how to use the Regular Expression with **sed** .

Here sed, short for **s** tream **e** ditor is a string navigation editor that is created exclusively for individual scripts. So all the input data you enter is passed from STDOUT and it does not change the input file.

Call sed in Unix / Linux

Before we begin, make sure you have an internal copy of the / etc / passwd file to work with **sed** .

As mentioned earlier, sed can be called by sending data through a pipe to it as follows:

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

The cat command dumps the contents of / etc / passwd to sed via pipe in sed's sample space. The sample space is the inner work area that sed uses to carry out its work.

Sed general syntax in Unix / Linux

Here is the general syntax for sed:

```
/ pattern / action
```

Here, the **pattern** is a Regular Expression, which action is one of the commands provided in the following table. If the pattern is omitted, the action is executed for every line as we looked above.

The slash character that surrounds the pattern is required because they are used as limit markers.

Sequence Description Print line d Delete line s / pattern1 / pattern2 / Replace the event of pattern 1 with the event in pattern 2.

Delete all lines with sed

Call sed again, but this time tell sed to use the edit command to delete the lines, denoted by a single character d:

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

Instead of calling sed by sending a file to it via a pipe, you can instruct sed to read the data from a file, as in the following example.

The following command does exactly the same thing as mentioned above. You try to practice it without using the cat command.

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

SED positioning in Unix / Linux

Sed also understands something called addresses. The address is either a separate location in a file or a range that a particular edit command should be applied to. When sed encounters something that is not located, it performs its operations on every line in the file.

The following command adds a basic address to the sed command you are using:

```
$ cat / etc / passwd |  sed '1d' |  more daemon : x : 1 : 1 : daemon : / usr /
```

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

Sed address ranges in Unix / Linux

So what if you want to remove more than one line from a file? You can define a range of addresses with sed as follows:

```
$ cat / etc / passwd |  sed '1, 5d' |  more games : x : 5 : 60 : games : / usr
```

The above command will apply on all lines from line 1 to line 5. So it deletes the first 5 lines.

Try practicing on the ranges provided below:

Range Description '4,10d' The line starting from 4 to 10 is deleted. '10, 4d 'Only the 10th line is deleted, because sed does not work in the opposite direction. '4, + 5d' It will match line 4 in the file, delete that line, continue to delete the next 5 lines, and then stop the deletion and print the rest of the file. '2.5! D' It will delete everything except from line 2 to line 5. '1 ~ 3d' It will delete the first line, step through the next 3 lines, and then delete the 4th line. Sed Continue to apply this deletion pattern to the end of the file. '2 ~ 2d' It tells sed to delete the second line, step through the next line, delete the next line, and repeat the previous step until the file is read. '4.10p' Starting from the 4th to 10th lines printed. '4, d' This is an error syntax. ', 10d' This is an error syntax.
Note : While using **p** action, you should use the **-n** function to avoid repeating the line printing. You try to check the difference between the following two commands:

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

Check the above command without using the -n function as follows:

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

Replace command (Unix command) in Unix / Linux

The replacement command, represented by s, replaces any string you specify with any other string.

To replace one string with another, you need a few ways to tell sed where your first string ends and the replacement string starts. This is traditionally done by separating 2 strings with a slash.

The following command replaces the first event on a line of the root **string** with **the amrood string** .

```
$ cat / etc / passwd |  sed 's / root / amrood /' amrood : x : 0 : 0 : root use
```

It is important to remember that sed only replaces the first event on a line. If more than one line occurs on the root string, only the first line will be replaced.

To tell sed to perform a full replacement, add the character g to the end of the command as follows:

```
$ cat / etc / passwd |  sed 's / root / amrood / g' amrood : x : 0 : 0 : amrooc
```

Substitution Flag (Substitution Flag) in Unix / Linux

There are a number of useful Flag uses that can be used to add g marks, and you can identify it more than once.

Flag DescriptionG Replace all matches (replacement strings), not just with the first match. NUMBER Replace only matching NUMBER number. p If the replacement is created, print the sample space. w FILENAME If the replacement is created, write the result to FILENAME. I or i Create a match in the distinction between typography (uppercase and lowercase). M or m In the normal processing mode of special Expression characters ^ and $, this flag causes ^ to match an empty string after a new line and $ matches a blank string before a new line .

Use an alternative string in Unix / Linux

You can find yourself to make a replacement on a string that includes a slash character. In this case, you can define another operator by providing the character specified after the letter s:

```
$ cat / etc / passwd |  sed 's: / root: / amrood: g' amrood : x : 0 : 0 : amroo
```

In the above example we used the limit sign: as instead of the slash / because we are trying to find / root instead of a single root.

Change position with empty space in Unix / Linux

Use an empty replacement string to delete the root string from the file / etc / passwd:

```
$ cat / etc / passwd |  sed 's / root // g' : x : 0 : 0 :: /: / bin / sh daemo
```

Locate replacement in Unix / Linux

If you want to replace the string sh with the string quiet only on the 10th line, you can specify it as follows:

```
$ cat / etc / passwd |  sed '10s / sh / quiet / g' root : x : 0 : 0 : root use
```

In the same way, to change a range of addresses, you can do the same job:

```
$ cat / etc / passwd |  sed '1.5s / sh / quiet / g' root : x : 0 : 0 : root us
```

As you can see from the output, the first 5 lines of the sh string have been changed by string quiet, but the remaining lines are not touched.

Matching command in Unix / Linux

You will use function p in parallel with the -n function to print all matching lines as follows:

```
$ cat testing |  sed - n '/ root / p' root : x : 0 : 0 : root user : / root: /
```

Use Regular Expression in Unix / Linux

While matching with patterns, you can use Regular Expressions that provide more flexibility for you.

Check the following example that matches all lines starting with the daemon and then deleting them:

```
$ cat testing |  sed '/ ^ daemon / d' root : x : 0 : 0 : root user : / root: /
```

Following is the example that will delete all lines ending with **sh** :

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

The table below lists four special characters that are very useful in regular expressions:

Character Description ^ Matches the beginning of the lines. $ Matches the end of the lines. . Matches any single character. * Matches with 0 or more events of the previous character. [chars] Matches any character given in chars (is a layout of characters). You can use the - character to indicate a sequence of characters.

Matching characters in Unix / Linux

You look at the table below listing some other Expressions that explain the use of metacharacters. For example, the following samples:

Expression Description / ac / Match lines that contain strings like a + c, ac, abc, match and a3c. / a * c / Matches the same strings as the given string like ace, yacc, and arctic. / [TT] he / Matches the string The and the. / ^ $ / Matches with empty lines. /^.*$/ Matches with a whole line regardless of the line. / * / Matches with one or more spaces. / ^ $ / Matches with empty lines.
Below is a table listing some of the character layout sets that are commonly used:

Setting Description [az] Matches a single letter. [AZ] Matches a single-letter letter. [a-zA-Z] Matches with a single letter. [0-9] Matches with some applications. [a-zA-Z0-9] Matches with a single number or letter.

Character class keywords in Unix / Linux

Some special keywords are often available for regular expressions, especially GNU utilities that use regular expressions. This is very useful for Expression Sed Regulars when they simplify tasks and improve its readability.

For example, characters from a to z as well as characters from A to Z delegate a character class as keywords [[: alpha:]]

Using the alphabetic class keyword, this command only prints lines in the /etc/syslog.conf file that starts with an alphabetic character.

```
$ cat / etc / syslog .  conf |  sed - n '/ ^ [[: alpha:]] / p' authpriv . * / v
```

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

Character class Description [[: alnum:]] Alphabetical-number [az AZ 0-9] [[: alpha:]] Alphabet [az AZ] [[: blank:]] Space characters (spaces or tabs) [[: cntrl:]] Control characters [[: digit:]] Numbers [0-9] [[: graph:]] Any visible characters (except white spaces) [[: lower:]] Lowercase characters [az] [[: print:]] Printable characters (non-control characters) [[: punct:]] Signals self-punctuation [[: space:]] White space [[: upper:]] Uppercase characters [AZ] [[: xdigit:]] Hexadecimal digits [0-9 af AF]

Reference & in Unix / Linux

Super characters & in sed represent the content of the pattern that is matched. For example, suppose you have a file called phone.txt filled with phone numbers, as follows:

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

You want to create a region code (the first 3 digits) surrounded by parentheses to make it easier to read. To do this, you can use the alternate character &, like:

```
$ sed - e 's / ^ [[: digit:]] [[: digit:]] [[: digit:]] / (&) / g' phone .  tx
```

Here in the sample section you are matching the first 3 digits, and then use & you are swapping the 3 digits with parentheses around.

Use many sed commands in Unix / Linux

You can use many sed commands in a sed command as follows:

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

Here, command1 to commandN is the sed command types discussed above. These commands are applied to each line in a list of files provided by the files.

Using the same technique, we can write the example phone number above as follows:

```
$ sed - e 's / ^ [[: digit:]] {3} / (&) / g' - e 's /) [[: digit:]] {3} / & - /
```

Note : In the above example, instead of repeating the keyword class [[: digit:]] 3 times, you change it with {3}, which means to match the Regular Expression 3 times in advance. Here I use to get down the line, you should

remove it before running this command.

Reverse reference in Unix / Linux

Super characters & are useful, but more useful is the ability to define specific areas in a regular expression so that you can reference them in relocate strings. By defining specific parts of a Regular Expression, you can then review these sections with a special reference character.

To reverse reference, you must first define an area and then review that area. To define an area, you insert parentheses in a backslash around each area you are interested in. The first area that you surround with a backslash is then referenced by 1, the second area by 2, and continues.

Suppose phone.txt has the following data:

```
( 555 ) 555 - 1212 ( 555 ) 555 - 1213 ( 555 ) 555 - 1214 ( 666 ) 555 - 1215 ( 
```

Now try the following command:

```
$ cat phone .  txt |  sed 's /(.*))(.*-)(.*$)/ Area code: 1 Second: 2 Third: 3
```

Note : In the above example, each Regular Expression within parentheses will be referenced by 1, 2, . Here, I used to get down the line, you should remove them before running the command.

According to Tutorialspoint

Previous article: Select loop in Shell

Next article: Basics of File System in Unix / Linux