Big Data Programming COMP47470

Regex, grep and sed

Anthony VENTRESQUE

Wednesday 23rd January, 2019

Table of contents

Quotes

Types of quotes

There are four (4) types of quotes in bash:

- Command substitution quotes: `` `...` ``
- Weak quotes: `"..."`
- Strong quotes: `'...'`
- Character quotes: `\`\"\'`

Examples

```
$> filename='myFile.txt'
$> echo 'text in a file' > $filename
$> echo `cat $filename`
text in a file
$> echo "cat $filename"
cat myFile.txt
$> echo 'cat $filename'
cat \$filename
$> echo \' cat $filename\'
'cat myFile.txt'
$> echo \' cat \$filename\'
'cat $filename'
```

Regular Expressions: regex

Definition

A regular expression is a sequence of characters that form a **pattern**.
e.g. `^[hH]ello[1-5]*`

Regex are useful for searching words matching a certain patterns.

Regex a **case sensitive**

A regex is made of three types of elements:

Anchors are used to specify the position of the pattern in a line

Character sets represent the characters that should be matched

Modifiers let us specify how many times the previous character set should be repeated

Anchors

There are two anchors:

- ^represents the beginning of the line
  e.g. `^pattern`
- $ represents the end of the line
  e.g. `pattern$`

If no anchors are used in the regex then it will match the pattern anywhere in a line.

Examples

Regex	String	Match?
^Anthony	Anthony is the lecturer	Match: Anthony is the lecturer
^Anthony	Is Anthony the lecturer	No match
lecturer$	Anthony is the lecturer	Match: Anthony is the lecturer
lecturer$	The lecturer is Anthony	No match
Anthony	Is Anthony the lecturer	Match: Is Anthony the lecturer
lecturer	The lecturer is Anthony	The lecturer is Anthony

Character sets

A character set is formed by a sequence of characters (e.g. a, d, 4, 5, #, \$, etc).

You can also use the following constructs in a character set:

- . to match any character
  e.g. .ard.assword12
- [characters] to match any characters in the defined subset
  e.g. hard[A-Za-z]as[zs]word[1-9]2
- [^characters] to match any character expect those in the defined subset
  e.g. hardPassword[^2-9]2

Modifiers

Modifiers can be applied to any character set and indicate how many times a character set appears. The following modifiers can be used in a regex:

- * indicates that the character set appears zero or more times
- + indicates that the character set appears one or more times
- ? indicates that the character set appears zero or one time

N.B. + and ? are only available when using **extended regular expressions**. By default only * is available.

Examples

Regex	String	Match?
Hel*o	Hello Anthony	Match: Hello Anthony
Hel*o	Hellllllllo Anthony	Match: Hellllllllo Anthony
Hel*o	Heo Anthony	Match: Heo Anthony
Hel+o	Hello Anthony	Match: Hello Anthony
Hel+o	Hellllllllo Anthony	Match: Hellllllllo Anthony
Hel+o	Heo Anthony	No match
Hel?o	Hello Anthony	No match
Hel?o	Hellllllllo Anthony	No match
Hel?o	Heo Anthony	Match: Heo Anthony

grep

Syntax

Grep prints out the lines from its input that contain a string matching a given pattern. The input can be given as a file name that grep will then read or as a string though the standard input.

Examples of grep syntax:

```
#Returns the lines containing pattern "root"
grep root /etc/passwd
#Returns the lines containing pattern "^ro*t"
grep '^ro*t' /etc/passwd
#Returns the lines containing pattern "root:x:0:[0-9]"
cat /etc/passwd | grep 'root:x:0:[0-9]'
```

Combining patterns

You can combine multiple patters in a single grep command using the **or** operator: `|`.

- pattern1 **or** pattern2

  ```
  grep 'pattern1|pattern2' filename
  grep -e pattern1 -e 'pattern2'
  ```

- pattern1 **and** pattern2: there is not and operator but it can be simulated with and or:

  ```
  grep 'pattern1.*pattern2|pattern2.*pattern1' fName
  ```

Options

Several options are available in grep:

-c	only returns the number of matching lines
-v	inverted search: returns the lines that do not contain the pattern
-i	ignores case
-n	returns the line numbers as well as the matching lines
-E	uses extended rehular expressions
others	see the man page

Stream editor: sed

Syntax

`sed` is a powerful tool that lets you edit a stream, so that is what you would use if you wanted to modify a file. Here we will only discuss one of its many functions: the **substitution** function.

The syntax for the substitution function is `/s/pattern/replacement/` . This replaces a string that matches the pattern (a regex) with the replacement string.

The "/" delimiter can be replaced with other characters such as "_", ";", "#", "|" or even a space " " (provided your pattern is between quotes).

Examples

```
$> echo "dog" | sed s/dog/cat/
cat
$> sed s/dog/cat/ dogFile
cat #assuming the file dogFile only contains the text "d
$> echo "hellllllo" | sed s_hel*o_hello_
hello
$> echo "boring" | sed 's boring fun '
fun
$> echo "Every day is pay day" | sed s/day/night/
Every night is pay day
```

N.B. by default sed only replace the first string that matches the pattern for each line.

More sophisticated replacements

The replacement string does not have to be just a literal. We can reuse (parts of) the matched string in the replacement string using special characters:

- Using & in the replacement string will insert the whole matched string
- Using \1 to \9 will insert the string matching the 1^{st} to 9^{th} capturing group of the pattern. A capturing group is defined by enclosing a part of the pattern between escaped parentheses (\(and \)). The parentheses should not be escaped when using extended regex.

Examples

```
$> echo "user123" | sed 's/user[0-9]*/<b>&<\/b>/'
<b>user123</b>
$> echo "COMP47470" | sed 's/COMP\([0-9]*\)/CS\1/'
CS30640
$> echo "one two" | sed ‘s/\([^ ]*\) \([^ ]*\)/\2 \1/ ‘
two one
```

Flags

As we have seen before, `sed` only replaces the first string matching the pattern for each line by default. We can use flags to change that behaviour and control which matching string is replaced by sed.

The syntax is `/s/pattern/replacement/flag`

The flag can be a number n, and sed will only replace the n^{th} string for each line, or `g` (for global), and sed will replace all matching strings.

Examples

```
$> echo "Every day is pay day" | sed s/day/night/2
Every day is pay night
$> echo "Every day is pay day" | sed s/day/night/g
Every night is pay night
```