Linux Operating System and Programming
TOPIC # 13

FILTERS USING REGULAR EXPRESSIONS – grep and sed

We often need to search a file for a pattern, either to see the lines containing (or not containing) it or to have it replaced with something else. This chapter discusses two important filters that are specially suited for these tasks – grep and sed. grep takes care of all search requirements we may have. sed goes further and can even manipulate the individual characters in a line. In fact sed can de several things, some of then quite well.
grep – searching for a pattern

It scans the file / input for a pattern and displays lines containing the pattern, the line numbers or filenames where the pattern occurs. It's a command from a special family in UNIX for handling search requirements.

```
grep options pattern filename(s)
grep “sales” emp.lst
```

will display lines containing sales from the file emp.lst. Patterns with and without quotes is possible. It's generally safe to quote the pattern. Quote is mandatory when pattern involves more than one word. It returns the prompt in case the pattern can't be located.

```
grep president emp.lst
```

When grep is used with multiple filenames, it displays the filenames along with the output.

```
grep “director” emp1.lst emp2.lst
```

Where it shows filename followed by the contents
grep options
grep is one of the most important UNIX commands, and we must know the options that POSIX requires grep to support. Linux supports all of these options.

-i	ignores case for matching
-v	doesn't display lines matching expression
-n	displays line numbers along with lines
-c	displays count of number of occurrences
-l	displays list of filenames only
-e exp	specifies expression with this option
-x	matches pattern with entire line
-f file	takes pattrens from file, one per line
-E	treats pattren as an extended RE
-F	matches multiple fixed strings

```
grep -i ‘agarwal’ emp.lst
grep -v ‘director’ emp.lst > otherlist
```

wc -l otherlist will display 11 otherlist

```
grep –n ‘marketing’ emp.lst
grep –c ‘director’ emp.lst
grep –c ‘director’ emp*.lst
```

will print filenames prefixed to the line count

```
grep –l ‘manager’ *.lst
```

will display filenames only

grep –e ‘Agarwal’ –e ‘aggarwal’ –e ‘agrawal’ emp.lst

will print matching multiple patterns

grep –f pattern.lst emp.lst

all the above three patterns are stored in a separate file pattern.lst

“When you know properties of a file or directory and you wish to search for locations of the same, use find command but when based on the content of the files, you wish to search for file names and locations, use grep family commands.”

Basic Regular Expressions (BRE) – An Introduction

It is tedious to specify each pattern separately with the -e option. grep uses an expression of a different type to match a group of similar patterns. If an expression uses meta characters, it is termed a regular expression. Some of the characters used by regular expression are also meaningful to the shell.

BRE character subset

The basic regular expression character subset uses an elaborate meta character set, overshadowing the shell’s wild-cards, and can perform amazing matches.

*	Zero or more occurrences
g*	nothing or g, gg, ggg, etc.
.	A single character
.*	nothing or any number of characters
[pqr]	a single character p, q or r
[c1-c2]	a single character within the ASCII range represented by c1 and c2

The character class

grep supports basic regular expressions (BRE) by default and extended regular expressions (ERE) with the –E option. A regular expression allows a group of characters enclosed within a pair of [ ], in which the match is performed for a single character in the group.

grep “[aA]g[ar][ar]wal” emp.lst

A single pattern has matched two similar strings. The pattern [a-zA-Z0-9] matches a single alphanumeric character. When we use range, make sure that the character on the left of the hyphen has a lower ASCII value than the one on the right. Negating a class (^) (caret) can be used to negate the character class. When the character class begins with this character, all characters other than the ones grouped in the class are matched.

The *

The asterisk refers to the immediately preceding character. * indicates zero or more occurrences of the previous character.

g* nothing or g, gg, ggg, etc.

grep “[aA]gg*[ar][ar]wal” emp.lst

Notice that we don’t require to use –e option three times to get the same output!!!!!

The dot

A dot matches a single character. The shell uses ? Character to indicate that.

.* signifies any number of characters or none

grep “j.*saxena” emp.lst

Specifying Pattern Locations (^ and $)

Most of the regular expression characters are used for matching patterns, but there are two that can match a pattern at the beginning or end of a line. Anchoring a pattern is often necessary when it can occur in more than one place in a line, and we are interested in its occurance only at a particular location.

^	for matching at the beginning of a line
$	for matching at the end of a line

grep “^2” emp.lst

Selects lines where emp_id starting with 2

grep “7…$” emp.lst

Selects lines where emp_salary ranges between 7000 to 7999

grep “^[^2]” emp.lst

Selects lines where emp_id doesn’t start with 2

When meta characters lose their meaning

It is possible that some of these special characters actually exist as part of the text. Sometimes, we need to escape these characters. For example, when looking for a pattern g*, we have to use \

To look for [, we use \[

To look for .*, we use \.*

Extended Regular Expression (ERE) and grep

If current version of grep doesn’t support ERE, then use egrep but without the –E option. -E option treats pattern as an ERE.

+	matches one or more occurrences of the previous character
?	Matches zero or one occurrence of the previous character

b+ matches b, bb, bbb, etc.

b? matches either a single instance of b or nothing

These characters restrict the scope of match as compared to the *

grep –E “[aA]gg?arwal” emp.lst

# ?include +<stdio.h>

The ERE set

ch+	matches one or more occurrences of character ch
ch?	Matches zero or one occurrence of character ch
exp1|exp2	matches exp1 or exp2
(x1|x2)x3	matches x1x3 or x2x3

Matching multiple patterns (|, (and))

grep –E ‘sengupta|dasgupta’ emp.lst

We can locate both without using –e option twice, or

grep –E ‘(sen|das)gupta’ emp.lst

Input Validation via grep with regular expression

```
[jpandya@JMP lg]$ cat emails.txt
test@some.com
test@some.edu
test@some.co.uk
dtssdfasdf
[jpandya@JMP lg]$ grep -Ei '\b[a-z0-9]{1,}@[a-z0-9]*\.(com|net|org|uk|mil|gov|edu)\b' emails.txt
test@some.com
test@some.edu
```

```
test@some.co.uk
[jpandya@JMP lg]$ grep -Eiv '\b[a-z0-9]{1,}@[a-z0-9]*\.(com|net|org|uk|mil|gov|edu)\b' emails.txt
dtssdfasdf
[jpandya@JMP lg]$
```

IP addresses

```
    $ grep -E '\b[0-9]{1,3}(\.[0-9]{1,3}){3}
    \b' patterns
    123.24.45.67
    312.543.121.1
```

MAC addresses

```
   $ grep -Ei '\b[0-9a-f]{2}
   (:[0-9a-f]{2}){5}\b' patterns
   ab:14:ed:41:aa:00
```

Email addresses

```
   $ grep -Ei '\b[a-z0-9]{1,}@[a-z]*\. (com|net|org|uk|mil|gov|edu)\b' patterns
   test@some.com
   test@some.edu
   test@some.co.uk
```

U.S.-based phone numbers

```
    $ grep -E '\b(\(|)[0-9]{3}
    (\)|-|\)-|)[0-9]{3}(-|)[0-9]{4}\b'
     patterns
    (312)-555-1212
    (312) 555-1212
    312-555-1212
    3125551212
```

Social Security numbers

```
    $ grep -E '\b[0-9]{3}( |-|)
    [0-9]{2}( |-|)[0-9]{4}\b' patterns
    333333333
    333 33 3333
    333-33-3333
```

Credit card numbers

For most credit card numbers, this expression works:

```
    $ grep -E '\b[0-9]{4}(( |-|)
    [0-9]{4}){3}\b' patterns
    1234 5678 9012 3456
    1234567890123456
    1234-5678-9012-3456
```

American Express card numbers would be caught by this expression:

```
$ grep -E '\b[0-9]{4}( |-|)
[0-9]{6}( |-|)[0-9]{5}\b' patterns
1234-567890-12345
123456789012345
1234 567890 12345
```

Metacharacter	Name	Matches
Items to match a single character		
.	Dot	Any one character
[...]	Character class	Any character listed in brackets
[^...]	Negated character class	Any character not listed in brackets
\char	Escape character	The character after the slash literally; used when you want to search for a “special” character, such as “$” (i.e., use “\$”)
Items that match a position		
^	Caret	Start of a line
$	Dollar sign	End of a line
\<	Backslash less-than	Start of a word
\>	Backslash greater-than	End of a word
The quantifiers		
?	Question mark	Optional; considered a quantifier ; Matches zero or one occurrence of previous character.
*	Asterisk	Any number (including zero); sometimes used as general wildcard
	Plus	One or more of the preceding expression

+		
{N}	Match exactly	Match exactly N times
{N,}	Match at least	Match at least N times
{min,max}	Specified range	Match between min and max times
Other		
|	Alternation	Matches either expression given
-	Dash	Indicates a range
(...)	Parentheses	Used to limit scope of alternation

10 | grep Pocket Reference

Metacharacter	Name	Matches
\1, \2, ...	Backreference	Matches text previously matched within parentheses (e.g., first set, second set, etc.)
\b	Word boundary	Batches characters that typically mark the end of a word (e.g., space, period, etc.)
\B	Backslash	This is an alternative to using “\\” to match a backslash, used for readability
\w	Word character	This is used to match any “word” character (i.e., any letter, number, and the underscore character)
\W	Non-word character	This matches any character that isn’t used in words (i.e., not a letter, number, or underscore)
\`	Start of buffer	Matches the start of a buffer sent to grep
\'	End of buffer	Matches the end of a buffer sent to grep

sed – The Stream Editor

sed is a multipurpose tool which combines the work of several filters . sed uses instructions to act on text. An instruction combines an address for selecting lines, with an action to be taken on them.

```
sed options 'address action' file(s)
```

sed supports only the BRE set. Address specifies either one line number to select a single line or a set of two lines, to select a group of contiguous lines . action specifies print, insert, delete, substitute the text.

sed processes several instructions in a sequential manner. Each instruction operates on the output of the previous instruction. In this context, two options are relevant, and probably they are the only ones we will use with sed – the –e option that lets us use multiple instructions, and the –f option to take instructions from a file. Both options are used by grep in identical manner.

Line Addressing

```
sed '3q' emp.lst
```

Just similar to head –n 3 emp.lst. Selects first three lines and quits

```
sed –n '1,2p' emp.lst
```

p prints selected lines as well as all lines. To suppress this behavior, we use –n whenever we use p command

```
sed –n '$p' emp.lst
```

Selects last line of the file

```
sed –n '9,11p' emp.lst
```

Selecting lines from anywhere of the file, between lines from 9 to 11

```
sed –n '1,2p
7,9p
$p' emp.lst
```

Selecting multiple groups of lines

```
sed –n '3,$!p' emp.lst
```

Negating the action, just same as 1,2p

Using Multiple Instructions (-e and –f)

There is adequate scope of using the –e and –f options whenever sed is used with multiple instructions.

```
sed –n –e '1,2p' –e '7,9p' –e '$p' emp.lst
```

Let us consider,

```
cat instr.fil
1,2p
7,9p
$p
```

-f option to direct the sed to take its instructions from the file

```
sed –n –f instr.fil emp.lst
```

We can combine and use –e and –f options as many times as we want

```
sed –n –f instr.fil1 –f instr.fil2 emp.lst
sed –n –e '/saxena/p' –f instr.fil1 –f instr.fil2 emp.lst
```

Context Addressing

We can specify one or more patterns to locate lines

```
sed –n '/director/p' emp.lst
```

We can also specify a comma-separated pair of context addresses to select a group of lines.

```
sed –n '/dasgupta/,/saxena/p' emp.lst
```

Line and context addresses can also be mixed

```
sed –n '1,/dasgupta/p' emp.lst
```

Using regular expressions

Context addresses also uses regular expressions.

```
Sed –n '/[aA]gg*[ar][ar]wal/p' emp.lst
```

Selects all agarwals.

```
Sed –n '/sa[kx]s*ena/p
            /gupta/p' emp.lst
```

Selects saxenas and gupta.

We can also use ^ and $, as part of the regular expression syntax .

```
sed –n '/50…..$/p' emp.lst
```

Selects all people born in the year 1950.

Writing Selected Lines to a File (w)

We can use w command to write the selected lines to a separate file.

```
sed –n '/director/w dlist' emp.lst
```

Saves the lines of directors in dlist file

```
sed –n '/director/w dlist
              /manager/w mlist
              /executive/w elist' emp.lst
```

Splits the file among three files

```
sed –n '1,500w foo1
501,$w foo2' foo.main
```

Line addressing also is possible. Saves first 500 lines in foo1 and the rest in foo2

Text Editing

sed supports inserting (i), appending (a), changing (c) and deleting (d) commands for the text.

```
$ sed '1i\
> #include <stdio.h>\
> #include <unistd.h>
> 'foo.c > $$
```

Will add two include lines in the beginning of foo.c file. Sed identifies the line with out the \ as the last line of input. Redirected to $$ temporary file. This technique has to be followed when using the a and c commands also. To insert a blank line after each line of the file is printed (double spacing text), we have,

```
sed 'a\
' emp.lst
```

Deleting lines (d)

```
sed '/director/d' emp.lst > olist        or
sed –n '/director/!p' emp.lst > olist
```

Selects all lines except those containing director, and saves them in olist

Note that –n option not to be used with d

Substitution (s)

Substitution is the most important feature of sed, and this is one job that sed does exceedingly well.

```
        [address]s/expression1/expression2/flags
```

Just similar to the syntax of substitution in vi editor, we use it in sed also .

```
        sed ‘s/|/:/’ emp.lst | head –n 2
2233:a.k.shukla |gm |sales |12/12/52|6000
9876:jai sharma |director|production|12/03/50|7000
```

Only the first instance of | in a line has been replaced. We need to use the g (global) flag to replace all the pipes.

```
        sed ‘s/|/:/g’ emp.lst | head –n 2
```

We can limit the vertical boundaries too by specifying an address (for first three lines only).

```
        sed ‘1,3s/|/:/g’ emp.lst
```

Replace the word director with member in the first five lines of emp.lst

```
        sed ‘1,5s/director/member/’ emp.lst
```

sed also uses regular expressions for patterns to be substituted. To replace all occurrence of agarwal, aggarwal and agrawal with simply Agarwal, we have,

```
        sed ‘s/[Aa]gg*[ar][ar]wal/Agarwal/g’ emp.lst
```

We can also use ^ and $ with the same meaning. To add 2 prefix to all emp-ids,

```
        sed ‘s/^/2/’ emp.lst | head –n 1
22233 | a.k.shukla | gm | sales | 12/12/52 | 6000
```

To add .00 suffix to all salary,

```
        sed ‘s/$/.00/’ emp.lst | head –n 1
2233 | a.k.shukla | gm | sales | 12/12/52 | 6000.00
```

Performing multiple substitutions

```
        sed ‘s/<I>/<EM>/g
        s/<B>/<STRONG>/g
        s/<U>/<EM>/g’ form.html
```

An instruction processes the output of the previous instruction, as sed is a stream editor and works on data stream

```
        sed ‘s/<I>/<EM>/g
        s/<EM>/<STRONG>/g’ form.html
```

When a ‘g’ is used at the end of a substitution instruction, the change is performed globally along the line. Without it, only the left most occurrence is replaced . When there are a group of instructions to execute, you should place these instructions in a file instea d and use sed with the –f option.

Compressing multiple spaces

```
    sed ‘s/*|/|/g’ emp.lst | tee empn.lst | head –n 3
2233|a.k.shukla|g.m|sales|12/12/52|6000
9876|jai sharma|director|production|12/03/50|7000
5678|sumit chakrobarty|dgm|mrking|19/04/43|6000
```

The remembered patterns

Consider the below three lines which does the same job

```
        sed ‘s/director/member/’ emp.lst
        sed ‘/director/s//member/’ emp.lst
        sed ‘/director/s/director/member/’ emp.lst
```

The // representing an empty regular expression is interpreted to mean that the search and substituted patterns are the same

```
sed 's/|//g' emp.lst          removes every | from file
```

Basic Regular Expressions (BRE) – Revisited

Three more additional types of expressions are:

- The repeated patterns - &
- The interval regular expression (IRE) – { }
- The tagged regular expression (TRE) – ( )

The repeated patterns - &

To make the entire source pattern appear at the destination also

```
sed 's/director/executive director/' emp.lst
sed 's/director/executive &/' emp.ls t
sed '/director/s//executive &/' emp.lst
```

Replaces director with executive director where & is a repeated pattern

The interval RE - { }

sed and grep uses IRE that uses an integer to specify the number of characters preceding a pattern. The IRE uses an escaped pair of curly braces and takes three forms:

- ch\{m\} – the ch can occur m times
- ch\{m,n\} – ch can occur between m and n times
- ch\{m,\} – ch can occur at least m times

The value of m and n can't exceed 255. Let teledir.txt maintains landline and mobile phone numbers. To select only mobile numbers, use IRE to indicate that a numerical can occur 10 times.

```
grep '[0-9]\{10\}' teledir.txt
```

Line length between 101 and 150

```
grep '^.\{101,150\}$' foo
```

Line length at least 101

```
sed –n '/.{101,\}/p' foo
```

The Tagged Regular Expression (TRE)

You have to identify the segments of a line that you wish to extract and enclose each segment with a matched pair of escaped parenthesis. If we need to extract a number, \([0-9]*\). If we need to extract non alphabetic characters,

```
\([^a-zA-Z]*\)
```

Every grouped pattern automatically acquires the numeric label n, where n signifies the nth group from the left.

```
sed 's/ \ (a-z]*\) *\ ([a-z]*\) / \2, \1/' teledir.txt
```

To get surname first followed by a , and then the name and rest of the line. sed does not use compulsorily a / to delimit patterns for substitution. We can use only any character provided it doesn't occur in the entire command line. Choosing a different delimiter has allowed us to get away without escaping the / which actually occurs in the pattern.

How `sed` Works

`sed` maintains two data buffers: the active *pattern* space, and the auxiliary *hold* space. Both are initially empty.

`sed` operates by performing the following cycle on each line of input: first, `sed` reads one line from the input stream, removes any trailing newline, and places it in the pattern space. Then commands are executed; each command can have an address associated to it: addresses are a kind of condition code, and a command is only executed if the condition is verified before the command is to be executed.

When the end of the script is reached, unless the `-n` option is in use, the contents of pattern space are printed out to the output stream, adding back the trailing newline if it was removed.[3] Then the next cycle starts for the next input line.

Unless special commands (like ‘`D`’) are used, the pattern space is deleted between two cycles. The hold space, on the other hand, keeps its data between cycles (see commands ‘`h`’, ‘`H`’, ‘`x`’, ‘`g`’, ‘`G`’ to move data between both buffers).

```
sed 's/one/ONE/' < test

[jpandya@JMP ~]$ cat test
one two three, one two three
four three two one
one hundred

[jpandya@JMP ~]$ sed 's/one/ONE/' < test
ONE two three, one two three
four three two ONE
ONE hundred

[jpandya@JMP ~]$ sed 's/one/ONE/g' < test
ONE two three, ONE two three
four three two ONE
ONE hundred
[jpandya@JMP ~]$

echo day | sed 's/day/night/'
```

In place replacement
You can modify a file in place with `sed`, if you're sure of what you're doing.

```
[jpandya@JMP ~]$ cat test
one two three, one two three
four three two one
one hundred
[jpandya@JMP ~]$ sed -i'.bak' 's/one/ONE/g' test
[jpandya@JMP ~]$ cat test
ONE two three, ONE two three
four three two ONE
ONE hundred
[jpandya@JMP ~]$ cat test.bak
one two three, one two three
four three two one
```

```
one hundred
[jpandya@JMP ~]$
```

Let's take a look at the `d` (delete) command and address ranges. Let's say that you want to delete lines 10 through 100 in a file:

`sed -i'' -e'10,100d'` *filename*

That tells sed to edit the file in place, then to delete the range 10 through 100.

Insert a line with 'i'

You can insert a new line before the pattern with the "i" command:

```
#!/bin/sh
sed '
/WORD/ i\
Add this line before every line with WORD
```

Change a line with 'c'

You can change the current line with a new line.

```
#!/bin/sh
sed '
/WORD/ c\
Replace the current line with the line
'
```

Append a line with 'a'

The "a" command appends a line after the range or pattern. This example will add a line after every line with "WORD:"

```
#!/bin/sh
sed '
/WORD/ a\
Add this line after every line with WORD
'
```

```
[jpandya@JMP ~]$ cat sedswap.data
10
20
[jpandya@JMP ~]$ sed -n '{h;n;G;p;}' ./sedswap.data
20
10
[jpandya@JMP ~]$
```

Every time a line is read, it is temporarily stored in the hold space(h). Then, the next line(n) is read into the pattern space. The line in the hold space is concatenated(G) with the pattern space. And the pattern space is printed(p).

Another one using sed. This is a little tricky.

```
$ sed -n '{h;${p;q;};n;G;p;}' file


Solaris


Linux


Ubuntu


AIX


Fedora
```

Every time a line is read, it is temporarily stored in the hold space(h). Then, the next line(n) is read into the pattern space. The line in the hold space is concatenated(G) with the pattern space. And the pattern space is printed(p).

${p;q} - This is needed especially to handle files which has an odd number of lines. When the file has odd number of lines, there is no next line to read, simply print the line in the pattern space(p) and quit(q).

Aim: Display the pattern “firstname lastname” in “lastname, firstname” manner using sed.
Hint: Every grouped pattern (used within parentheses) automatically acquires the numeric label n, where n signifies the nth group from the left.

```
[jpandya@JMP ~]$ cat names.txt
jigar pandya
james towery
michael won
[jpandya@JMP ~]$ sed 's/\([a-z]*\) \([a-z]*\)/\2 \1/' names.txt
pandya jigar
towery james
won michael
[jpandya@JMP ~]$ sed 's/\([a-z]*\) \([a-z]*\)/\2, \1/' names.txt
pandya, jigar
```

```
towery, james
won, michael
[jpandya@JMP ~]$
```

Courtesy:

Source: Sumitabha Das, “UNIX – Concepts and Applications”, 4th edition, Tata McGraw Hill, 2006
http://www.gnu.org/software/sed/manual/sed.html
http://www.theunixschool.com/2012/06/swap-every-2-lines-in-file.html

Contributors (CE Department, DDU):
- Prof. Jigar M. Pandya