12 Advanced filter utility –I: grep

12.1 Introduction

Till now, we discussed different filter utilities. In this chapter, we will learn about a very powerful filter utility known as **grep**. The **grep** stands for *g*lobally search a *r*egular *e*xpression and *p*rint it. It is also known as pattern matching utility. It is used to search a file for a particular pattern of characters, and display all records/lines that contain a pattern. The pattern that is searched in the file is referred to as the *regular expression*.

12.2 Pattern matching utility: grep

It is a filter utility that performs various tasks as follow:

- ✓ It scans a file for the occurrences of a pattern and displays lines in which scanned pattern is found.
- ✓ It scans a file for the occurrences of a pattern and displays lines in which scanned pattern does not found.
- ✓ It scans files for the occurrences of a pattern and displays name of files which contains a pattern in them.
- ✓ It also displays count of lines which contains pattern.

The general syntax for the **grep** command is as follows:

Syntax:

```
grep [options] pattern [filename(s)]
```

It is use to select and extract lines from a file and print only those lines that match a given pattern. In the above syntax square bracket indicates optional part. The *filename(s)* and *options* are optional and *pattern* is compulsory in the **grep** command. Here, a *pattern* is a simple string or more complex which contains metacharacters, a special character for pattern matohing. A *pattern* is also known as regular expression. Without a filename **grep** expects standard input. As a line is input, **grep** searches for the regular expression in the line and displays the line if it contains that regular expression. Execution stops when the user indicates end of input by pressing ***<ctrl+ d>***.

- ✓ For example, a user supply a command at shell prompt as follow:

```
$ grep 'unix'
unix and shell programming <enter>
unix and shell programming
red hat linux<enter>
unix OS<enter>
unix OS
<ctrl+ d>
$
```

grep requires an expression to represent the pattern to be searched for, followed by one or more filenames. The first argument is always treated as the expression, and the other arguments are considered as filenames.

Specifying regular expression:

A regular expression is a pattern that describes a set of strings. Regular expressions are constructed analogously to arithmetic expressions, by using various operators to combine smaller expressions.

An expression formed with some special and ordinary characters, which is expanded by a command, and not by the shell to match more than one string. A regular expression is always quoted to prevent its interpretation by the shell. Regular expressions can be used to specify very simple patterns of characters to highly complex ones. Some very simple patterns are shown in table-(a.12):

Table-(a.12): Example of simple regular expression

Regular expression	**Meaning**
A	It displays all lines that contain character "A".
"Unix"	It displays all lines that contains pattern "Unix"

Consider the following examples:

✓ Let us assume that the input file *f1* as follow:

```
$ cat f1
sco Unix
The red hat linux
user name user1
$
```

If a user wish to display lines of file *f1* which contains pattern 'Unix' then the command is as follow:

```
$grep Unix f1
sco Unix
$
```

✓ If you want to locate lines of file *f1* which contains character 'x' then the command is as follow:

```
$ grep x f1
sco Unix
The red hat linux
$
```

More complex regular expressions can be specified by the **grep's** metacharacters, always written in the quotes, shown in table-(b.12).

Table-(b.12): grep metacharacters

Character	**Use**
[...] or [.-.]	It matches any one single character within a square bracket.
^pattern	It matches a *pattern* at the beginning of each line.
pattern$	It matches a *pattern* at the end of each line.
.(dot)	It matches any single character except new-line character.
\ (backslash)	It indicates that **grep** should ignore the special meaning of the character following it in regular expression.
\<pattern	It matches a *pattern* at the beginning of any word in a line.
pattern\>	It matches a *pattern* at the end of any word in a line.
ch*	It matches zero or more occurrences of character *ch*.
ch\{m\}	The preceding character *ch* is occurred m-times.
ch\{m,\}	The preceding character *ch* is occurred at least m times.
ch\{m,n\}	The preceding character *ch* is occurred between m and n times.
\(exp\)	It matches expression *exp* for later referencing with \1, \2...

Consider the following examples which use **grep** metacharacters:

✓ To display lines of file *f1* which contains pattern as *user1* or *user2* or *user3* then the command is as follow:

```
$grep user[123] f1
user name user1
$
```

It displays lines of file *f1* which contains pattern *user1*.

✓ You can display lines which begins with pattern *The* then the command is as follow:

```
$grep '^The' f1
The red hat linux
$
```

It displays lines of file *f1* which start with pattern *The*.

✓ Similarly, if you wish to match a pattern at the end of each line then the command is as follow:

```
$grep 'Unix$' f1
sco Unix
$
```

It displays lines of file *f1* which end with pattern *Unix*.

✓ You can use dot. to match any character in a line. For example, consider a file *f2* as follow:

```
$ cat f2
Unix and shell programming
                                        #blank line contains only new-line character
red hat linux


Unix OS
vb.net
program and process
$
```

Here, file *f2* contains blank and non-blank line. If you wish to remove blank line from the output then the command is:

```
$ grep '.' f2
Unix and shell programming
red hat linux
Unix OS
vb.net
program and process
$
```

It displays all lines which contains any character in a line except blank-line (contains only new line character).

✓ You can protect special meaning of **grep** metacharacter using back-slash. For example, a user wish to display lines which contains '.' character anywhere in a line then the command is as follow:

```
$ grep '\.' f2
vb.net
$
```

It displays lines which contains '.' in a line.

✓ To display lines of file *f2* which contains pattern 'program' then the command is as follow:

```
$ grep 'program' f2
Unix and shell programming
```

```
program and process
$
```

But, if you want to display lines of file *f2* which contains word 'program' that means it is not a part of any string then the command is as follow:

```
$ grep '\<program\>' f2
program and process
$
```

- ✓ The * (asterisk) refers to the immediately preceding character. It matches zero or more occurrences of previous character. The pattern **a*** matches a null string, single character '**a**' and any number of **as**.

  i.e. (nothing) a aa aaa aaaa .....

- ✓ A user can locate lines which contains characters repeated more than one times then the command is:

```
$ grep 'mm*' f2
Unix and shell programming
program and process
$
```

It locates lines in which character 'm' repeated one or more times.

- ✓ You can display lines of file *f1* which contains exact 8 characters then the command is as follow:

```
$ grep '^.\{8\}$' f1
sco Unix
$
```

- ✓ To display lines of input file which contains characters between 5 and 15 then the command is like this:

```
$ grep '^.\{5,15\}$' f2
red hat linux
Unix OS
vb.net
$
```

It displays lines of file *f2* that contains character between 5 and 15.

- ✓ To display lines which contains pattern at the beginning of line would occur in the same line anywhere then you can use save operator with back references as follow:

```
$ grep '^\(.\).*\1' f2
program and process
$
```

It displays lines of file *f2* which contains any character occur at the beginning of line would also occur anywhere in the same line. The output shows that 1st character 'p' occur in the same line therefore we get such output.

- ✓ **grep** is silent and simply returns the prompt when a pattern is not found in a file.

```
$grep hello f2
$                          # No hello found
```

It displays nothing that means *hello* pattern do not present in file *f2*.

- ✓ **grep** also accept output of other command. For example, a user want to display filenames of working directory having permission read and write to owner, group and other user then the command is like this:

```
$ ls -l|grep '^.rw-rw-rw-'
-rw-rw-rw- 1 bharat   bharat      43   Apr  3 17:25   f1
-rw-rw-rw- 2 bca1     tybcasem5   77   Apr  4 11:02   f1.ln
-rw-rw-rw- 2 bca1     tybcasem5   77   Apr  4 11:02   f2
```

```
-rw-rw-rw- 1 bharat    bharat          34      Jul 18 2013   f3
$
```

✓ When **grep** is used with a series of strings, it interprets the first argument as the pattern and the rest as filenames along with the output. For example, consider a command as follow:

```
$grep red hat linux
```

It indicates that argument *red* is considered as pattern and other arguments *hat* and *linux* are considered as filenames.

✓ Quote is compulsory when a pattern contains more than one word. For example, consider the following command:

```
grep "hello world" filename
```

✓ Quote is also compulsory when a pattern contains special characters that can be interpreted by search utility (i.e. **grep**) not by the shell. You can generally use either single or double quotes, but if command substitution or variable evaluation is involved, you must use double quotes.
Consider an example which contains variable substitution in double-quote as follow:

```
$a=1
$grep "$a" f1
```

It prints all lines of file *f1* that contains 1. Consider another example which uses command substitution in double-quote as follow:

```
$grep "`echo if`" f1
```

It prints all lines of file *f1* which contains pattern *if* in line.

Options:
The **grep** utility can be used with many options, a few of which are discussed below:
(i)-c (count): It prints count of matching lines for each input file.

- ✓ For example, a command is follow:

```
$grep -c '.' f2
5
$
```

It counts all non-empty lines of file *f2.*

- ✓ Consider another command as follow:

```
$grep -c '^$' f2
2
$
```

It counts all empty lines (consist of only new-line character) of file *f2.*

(ii)-l (list): It displays only the names of files in which a pattern has been found.

- ✓ For example, consider a command as follow:

```
$ grep –l '.' *
```

It displays names of all files of current directory that contains any character in it.

- ✓ You can print names of all files of current directory that contains pattern *echo* anywhere in a file then the command is as follow:

```
$grep –l 'echo' *
```

(iii) ***–n (number):*** It can be used to display the line numbers containing the pattern, along with the lines.

- ✓ If you want to print line number before matched line then the command is as follow:

```
$ grep -n 'Unix' f2
1:Unix and shell programming
```

```
5:Unix OS
$
```

It prints two column output, each column delimited by colon (:). In the 1st column, line number will be displayed and 2nd column contains content of matched lines.

✓ You can give more than one filename as input files as follow:

```
$ grep -n 'Unix' f1 f2
f1:1:sco Unix
f2:1:Unix and shell programming
f2:5:Unix OS
$
```

It prints output in three columns, each column delimited by colon (:). The 1st field contains name of file, 2nd column contains line number and last column contains content of matched line.

(iv) -v (inverse) The –v option select all but not the lines containing the pattern.

✓ Sometimes, a user is interested only on unmatched lines then he used –v option as follow:

```
$ grep -cv '^$' f2
5
$
```

It counts all non-empty lines (contains only new-line character) of file *f2*.

✓ Consider another command as follow:

```
$ grep -v 'Unix' f1
The red hat linux
user name user1
$
```

It displays lines of file f1 which do not contains *Unix* pattern.

(v)-i (ignore): It ignores case in pattern matching.

✓ For example, you want to print lines that contains pattern *unix* in any case then the command is as follow:

```
$ grep -i 'unix' f1
sco Unix
$
```

It displays lines of file *f1* having *unix* pattern in any case.

(vi) -h (hide): It omits filenames when handling multiple files.

✓ For example, consider an example as follow:

```
$ grep -h 'Unix' f1 f2
sco Unix
Unix and shell programming
Unix OS
$
```

It displays lines of files *f1* and *f2* which contains pattern *Unix*. It does not display filename before matched line i.e. it hides name of a file.

•

(vii)-e RegExp : You can specity regular expression with this option. You can use this option multiple times.

✓ For example, you want to locates lines of file which contains pattern either *Unix* or *linux* then the command is as follow:

```
$ grep -e 'Unix' -e 'linux' f2
```

```
Unix and shell programming
red hat linux
Unix OS
$
```

(viii) -f fname: A list of strings to be match is stored in file *fname*.

✓ For example, consider a *patfile* as follow:

```
$ cat patfile
Unix
linux
$
```

It contains list of pattern in a separate lines. Now, we want to locate lines of file *f1* that contains any of the pattern given in file *patfile* then the command is as follow:

```
$ grep -f patfile f1
sco Unix
The red hat linux
$
```

NOTE: (i) Complement a class Regular expressions. the ^ (caret) is used at the beginning within the character class, while the shell uses the ! (exclamation mark) at the beginning within the character class.

(ii) The ^ (caret) use at the beginning of pattern (i.e. ^pattern) then it matches pattern at the beginning of each line, at any other location (e.g. a^b), it matches itself literally.

(iii) The . (dot) and * lose their meaning when placed inside the character class. The * also matched literally (lose its meaning) if it is the first character of the expression.

(iv) The * has significance in regular expression only. if it is preceded by a character. If it is the first character in a regular expression, then it matches itself.

12.3 grep family

There is a small family of **grep** utility which includes **egrep** and **fgrep**. These two utilities operate in a similar way to **grep** but each has its own particular usage, and there are small differences in the way that each work. Both utilities search for specific pattern in either the standard input stream or a series of input files supplied at command-line.

12.3.1 egrep

egrep stands for extended **grep**. It was invented by Alfred Aho. It extends **grep**'s pattern-matching capabilities in two major ways.

✓ It admits alternates

✓ It enables regular expressions to be bracketed/grouped using the pair of parenthesis (i.e. (…)), also known as factoring.

It offers all the options and regular expression metacharacters of **grep**, but its most useful feature is the facility to specify more than one pattern for search. While **grep** uses some more characters that are not recognized by **egrep**, **egrep** includes some additional extended metacharacters not used by either **grep** or **sed** utilities that are given in table-(c.12).

1950
9624166 ₹

Table-(c.12): egrep's extended metacharacters

Expression	Meaning
ch+	It matches one or more occurrences of character *ch*.
ch?	It matches zero or one occurrences of character *ch*.
exp1| exp2	It matches expression *exp1* or *exp2*.
(*x1|x2*)*x3*	It matches expression *x1x3* or *x2x3*.

Let us consider the following examples which uses extended metacharacter:

✓ To display lines which contains any character that occur one or more time then the command is as follow:

```
$ egrep m+ f2
Unix and shell programming
program and process
$
```

It prints lines of file *f2* that contains character '*m*' occur one or more times.

✓ If you want to locates lines which contains one or more patterns then you can use alternate metacharacter as follow:

```
$ egrep 'Unix|linux' f1
sco Unix
The red hat linux
$
```

It displays lines of file *f1* which contains pattern either *Unix* or *linux*.

✓ Sometimes, you want to display lines which contains either *software* or *hardware* then the command is as follow:

```
$egrep "(soft|hard)ware" f1
```

NOTE: In **egrep**, if a pattern contains some special characters then it must be quoted.

-f option: Storing pattern in a file

egrep provides a facility to take patterns from a file. If there are number of pattern that you have to match; **egrep** offers the –f (file) option to take such patterns from the file. For example, a file *patfile* contains patterns in which each pattern is delimited by '|' as follow:

```
$ cat patfile
Unix|linux
$
```

Now, you can execute **egrep** with the **–f** option in this way:

```
$egrep -f patfile f1
sco Unix
The red hat linux
$
```

Here, the command takes the pattern/expression from file *patfile* and display matched lines of file *f1*.

2.3.2 fgrep

fgrep stands for fixed/fast **grep**. The **fgrep** utility can normally only search for fixed strings i.e. character string without embedded metacharacters. However, some implementations of the **fgrep** utility allow it to be used with a few metacharacters – check your version to make sure. **fgrep** accepts multiple patterns, both from the command line and a file, but unlike **grep** and **egrep**, does not accept regular expressions. So, if the pattern to be search is a simple string, or a group of them, **fgrep** is recommended. It is arguably faster than **grep** and **egrep**, and should be used when using fixed strings.

Alternative patterns in **fgrep** are specified by separating one pattern from another using the new-line character. This is unlike in **egrep**, which uses the '|' to delimit two expressions. You may either specify these patterns in the command line itself, or store them in a file.

✓ For example consider a file *patfile* which contains list of pattern delimited by new-line character as follow:

```
$ cat patfile
Unix
linux
$
```

We can use this file using **–f** option as follow:

```
$ fgrep -f patfile f1
sco Unix
The red hat linux
$
```

✓ You can achieved same output without using file *patfile* by supplying patterns at command-line as follow:

```
$ fgrep 'Unix <enter>
> linux' f1 <enter>
sco Unix
The red hat linux
$
```

✓ The disadvantage with **grep** family is that none of them has separate facilities to identify fields. This limitation is overcome by **awk** utility.

Limitation of grep family:

The **grep** family has following limitation.

✓ It cannot be used to add, delete or change a line.

✓ It cannot be used to print only part of a line.

✓ It cannot read only part of a file.

✓ It cannot select a line based on the contents of the previous or the next line. There is only one buffer, and it holds only the current line.

Following table-(d.12) shows the atoms used in regular expression by **grep** family:

Table-(d.12): Atoms used by grep family

Atoms	grep	fgrep	egrep
Character	✓	✓	✓
Dot	✓	X	✓
Class	✓	X	✓
Anchors	✓	X	✓
Back Reference	✓	X	✓

As shown in table-(d.12), both **grep** and **egrep** utilities allows all the atoms in regular expression whereas **fgrep** utility supports only character atom.
Similarly, table-(e.12) shows the operators used in regular expression by **grep** family:

Table-(e.12): Operators used by grep family

Operators	grep	fgrep	egrep
Sequence	✓	✓	✓
Repetition	✓	X	✓
Alternation	X	X	✓
Group	X	X	✓
Save	✓	X	✓

Table-(e.12) indicates that **grep** utility supports sequence, repetition and save operators, **egrep** utility supports all operators but **fgrep** utility supports only sequence operator.

Exercise

Answer the following questions:

1. How will you remove blank lines from a file?
2. Explain the meaning of regular expression metacharacters used with **grep** utility.
3. How can you locates lines that begins with '*' character?
4. Explain any one pattern matching utility with proper illustration.
5. Differentiate between **grep**, **egrep** and **fgrep**.
6. What is the advantage of **fgrep** compare to **grep** and **egrep**?
7. Give the meaning of all the extended metacharacters used by **egrep**.
8. Write a note of **egrep**.
9. What is the significance of quote when a regular expression is enclosed in a quote?
10. Explain different role of ^ (caret) in regular expression with proper illustration.

Frame a grep command for the following:

1. Display all lines of file f1 that is longer than 20 characters.
2. Display only the name of all regular files of working directory.
3. Locate lines of file f2 that begins and ends with a dot (.) and containing anything between them.
4. Display all lines of file f1 which contains two or more $ symbol at end of line.
5. Display all lines of file f2 which contains two or more ^ symbol at beginning of line.
6. Display all lines of file f1 which contains spaces and tabs only (not blank-lines).
7. Display all lines of file f2 that contains * character in a line.