Module 11: Search and Sort Tools

Unix philosophy is to provide a rich set of generic tools, each with a variety of options. These primitive Unix tools can be combined in imaginative ways (by using pipes) to enhance user productivity. The tool suite also facilitates to build either a user customized application or a more sophisticated and specialised tool.

We shall discuss many primitive tools that are useful in the context of text files. These tools are often called “filters” because these tools help in searching the presence or absence of some specified pattern(s) of text in text files. Tools that fall in this category include ls, grep, and find. For viewing the output from these tools one uses tools like *more*, *less*, *head*, *tail*. The sort tool helps to sort and tools like *wc* help to obtain statistics about files. In this chapter we shall dwell upon each of these tools briefly. We shall also illustrate some typical contexts of usage of these tools.

11.1 *grep, egrep* and *fgrep*

grep stands for general regular expression parser. *egrep* is an enhanced version of *grep*. It allows a greater range of regular expressions to be used in pattern matching. *fgrep* is for fast but fixed string matching. As a tool, *grep* usage is basic and it follows the syntax: *grep* options pattern files with the semantics that search the file(s) for lines with the pattern and options as command modifiers. The following example1 shows how we can list the lines with int declarations in a program called *add.c*. Note that we could use this trick to collate all the declarations from a set of files to make a common include file of definitions.

```
bhatt@SE-0 [~/UPE] >>grep int ./C/add.c
extern int a1; /* First operand */
extern int a2; /* Second operand */
extern int add();
printf("The addition gives %d \n", add());
```

Regular Expression symbol	The effect of choice
c	matches character c
ĉ	matches a line that starts with character c
c$	matches c with the end of line
[]	matches any one of the characters in [ ]
.	matches any character
*	matches zero or more characters
RE*	matches zero, or more, repetitions of RE
RE1RE2	matches two concatenated expressions RE1RE2

Table 11.1: Regular expression options.

RE = A regular expression	The interpretation
RE+	matches one, or more, repetitions of RE
RE?	matches none, or one, occurrence of RE
RE1|RE2	matches occurrence of RE1 or RE2

Table 11.2: Regular expression combinations.

Note: print has int in it !!

In other words, *grep* matches string literals. A little later we will see how we may use options to make partial patterns for intelligent searches. We could have used **.c* to list the lines in all the c programs in that directory. In such a usage it is better to use it as shown in the example below

grep int ./C/.c | more*

This shows the use of a pipe with another tool *more* which is a good screen viewing tool. more offers a one screen at a time view of a long file. As stated in the last chapter, there is a program called less which additionally permits scrolling.

Regular Expression Conventions: Table 11.1 shows many of the grep regular expression conventions. In Table 11.1, RE, RE1, and RE2 denote regular expressions. In practice we may combine Regular Expressions in arbitrary ways as shown in Table 11.2. *egrep* is an enhanced *grep* that allows additionally the above pattern matching capabilities. Note that an RE may be enclosed in parentheses. To practice the above we make a file called testfile with entries as shown. Next, we shall try matching patterns using various options. Below we show a session using our text file called testfile.

aaa

a1a1a1

456

10000001

This is a test file.

bhatt@SE-0 [F] >>grep '[0-9]' testfile

a1a1a1

456

10000001

bhatt@SE-0 [F] >>grep '^4' testfile

456

bhatt@SE-0 [F] >>grep '1$' testfile

a1a1a1

10000001

bhatt@SE-0 [F] >>grep '[A-Z]' testfile

This is a test file.

bhatt@SE-0 [F] >>grep '[0-4]' testfile

a1a1a1

456

10000001

bhatt@SE-0 [F] >>fgrep '000' testfile

10000001

bhatt@SE-0 [F] >>egrep '0..' testfile

10000001

\ The back slash is used to consider a special character literally. This is required when the character used is also a command option as in case of -, * etc.

See the example below where we are matching a period symbol.

bhatt@SE-0 [F] >>grep '\.' testfile

This is a test file.

We may use a character's characteristics as options in grep. The options available are shown in Table 11.3.

bhatt@SE-0 [F] >>grep -v 'a1' testfile

aaa

456

10000001

The option	The effect of choice
-i	to ignore case like in so SO So etc.
-l	asks grep to not list lines, just lists filenames only
-v	select lines that do not match
-w	select lines that contain whole words only

Table 11.3: Choosing match options.

This is a test file.

bhatt@SE-0 [F] >>grep 'aa' testfile

aaa

bhatt@SE-0 [F] >>grep -w 'aa' testfile

bhatt@SE-0 [F] >>grep -w 'aaa' testfile

aaa

bhatt@SE-0 [F] >>grep -l 'aa' testfile

testfile

Context of use: Suppose we wish to list all sub-directories in a certain directory.

ls -l | grep ^d

bhatt@SE-0 [M] >>ls -l | grep ^d

drwxr-xr-x 2 bhatt bhatt 512 Oct 15 13:15 M1

drwxr-xr-x 2 bhatt bhatt 512 Oct 15 12:37 M2

drwxr-xr-x 2 bhatt bhatt 512 Oct 15 12:37 M3

drwxr-xr-x 2 bhatt bhatt 512 Oct 16 09:53 RAND

Suppose we wish to select a certain font and also wish to find out if it is available as a bold font with size 18. We may list these with the instruction shown below.

xlsfonts | grep bold\-18 | more

bhatt@SE-0 [M] >>xlsfonts | grep bold\-18 | more

lucidasans-bold-18

lucidasans-bold-18

lucidasanstypewriter-bold-18

lucidasanstypewriter-bold-18

Suppose we wish to find out at how many terminals a certain user is logged in at the moment. The following command will give us the required information:

who | grep username | wc -l > count

The *wc* with *-l* options gives the count of lines. Also, who|grep will output one line for every line matched with the given pattern (username) in it.

11.2 Using *find*

find is used when a user, or a system administrator, needs to determine the location of a certain file under a specified subtree in a file hierarchy. The syntax and use of *find* is:

find path expression action

where path identifies the subtree, expression helps to identify the file and the action specifies the action one wishes to take. Let us now see a few typical usages.

- List all the files in the current directory.

  bhatt@SE-0 [F] >>find . -print /* only path and action specified */

  .

  /ReadMe

  ./testfile

- Finding files which have been created after a certain other file was created.

  bhatt@SE-0 [F] >>find . -newer testfile

  .

  /ReadMe

  There is an option mtime to find modified files in a certain period over a number of days.

- List all the files which match the partial pattern test. One should use only shell meta-characters for partial matches.

  *bhatt@SE-0 [F] >>find . -name test**

  ./testfile

  *bhatt@SE-0 [F] >>find ../../ -name test**

  ../../COURSES/OS/PROCESS/testfile

  ../../UPE/F/testfile

- I have a file called linkedfile with a link to testfile. The find command can be used to find the links.

  bhatt@SE-0 [F] >>find ./ -links 2

  ./

  ./testfile

  ./linkedfile

- Finding out the subdirectories under a directory.

  bhatt@SE-0 [F] >>find ../M -type d

  ../M

  ../M/M1

  ../M/M2

  ../M/M3

  ../M/RAND

- Finding files owned by a certain user.

  bhatt@SE-0 [F] >>find /home/georg/ASM-WB -user georg -type d

  /home/georg/ASM-WB

  /home/georg/ASM-WB/examples

  /home/georg/ASM-WB/examples/Library

  /home/georg/ASM-WB/examples/SimpleLang

/home/georg/ASM-WB/examples/InstructionSet

/home/georg/ASM-WB/Projects

/home/georg/ASM-WB/FirstTry

The file type options are: f for text file, c for character special file, b for blocked files and p for pipes.

Strings: Sometimes one may need to examine if there are ascii strings in a certain object or a binary file. This can be done using a string command with the syntax :

string binaryfileName | more

As an example see its use below2:

bhatt@SE-0 [F] >>strings ../M/RAND/main | more

The value of seed is %d at A

The value is %f at B

ctags and etags: These commands are useful in the context when one wishes to look up patterns like c function calls. You may look up man pages for its description if you are a power user of c.

11.2.1 Sort Tool

For sort tool Unix treats each line in a text file as data. In other words, it basically sorts lines of text in text file. It is possible to give an option to list lines in ascending or descending order. We shall demonstrate its usage through examples given below.

The sort option	The effect of choice
-r	get a sort in decreasing order of value
-b	ignore leading blank spaces
-f	fold uppercase to lower case for comparison
-i	ignore characters outsides ASCII range
-n	perform numeric comparison. A number may have leading blanks

Table 11.4: Sort options.

bhatt@SE-0 [F] >>sort

aaa

bbb

aba

^d terminates the input ....see the output below

aaa

aba

bbb

bhatt@SE-0 [F] >>sort testfile

10000001

456

This is a test file.

a1a1a1

aaa

(Use -r option for descending order).

bhatt@SE-0 [F] >>sort testfile -o outfile

bhatt@SE-0 [F] >>

One can see the outfile for sorted output. *sort* repeats all identical lines. It helps to use a filter *uniq* to get sorted output with unique lines. Let us now modify our testfile to have repetition of a few lines and then use *uniq* as shown below.

bhatt@SE-0 [F] >>sort testfile|uniq|more

10000001

456

This is a test file.

a1a1a1

aaa

In table 11.4 we list the options that are available with *sort. sort* can also be used to merge files. Next, we will split a file and then show the use of merge. Of course, the usage is in the context of merge-sort.

One often uses filtering commands like *sort, grep* etc. in conjunction with *wc, more, head* and *tail* commands available in Unix. System administrators use *who* in conjunction with *grep, sort, find* to track of terminal usage and also for lost or damaged files.

split: *split* command helps one to split a file into smaller sized segments. For instance, if we *split* ReadMe file with the following command :

split -l 20 ReadMe seg

Upon execution we get a set of files segaa, segab, ...etc. each with 20 lines in it. (Check the line count using *wc*). Now merge using sorted segaa with segab.

sort -m segaa segab > check

A clever way to merge all the split files is to use *cat* as shown below:

cat seg > check*

The file check should have 40 lines in it. Clearly, *split* and *merge* would be useful to support merge-sort and for assembling a set of smaller files that can be sent over a network using e-mail whenever there are restrictions on the size of attached files.

In the next module we shall learn about the AWK tool in Unix. Evolution of AWK is a very good illustration of how more powerful tools can be built. AWK evolves from the (seemingly modest) generic tool *grep*!!