

Search and Sort Tools

Prof P.C.P. Bhatt



grep, egrep, And fgrep

- **grep** stands for general regular expression parser.
- **egrep** enhanced version of grep.
- **fgrep** fast but fixed string matching.
- Syntax for grep

grep options pattern files

Search the files(s) for lines with the pattern and
options as command modifier



Example to show how we can list the lines

- The following example 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());
```

NOTE print has int in it ... so we got an extra line ☺



Example to show how we can list the lines (Contd..)

- 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.



Regular Expression

- c : matches character c
- c : matches a line that starts with character c
- $c\$$: matches a line that ends with character c
- $[]$: matches any one of the character in $[]$
- $.$: matches any character
- $*$: matches zero or more, repetitions of RE
- RE^* : matches zero or more repetitions of RE
- $RE1RE2$: matches two concatenated expressions
 $RE1RE2$



Regular Expression Combinations

RE+ : matches one, or more, repetitions
of RE

RE? : matches none, or one, occurrence
of RE

RE1 | RE2 : matches occurrence of RE1 or RE2



Regular Expression : Examples

- 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 test file.

```
aaa
```

```
alalal
```

```
456
```

```
10000001
```

```
This is a test file
```




Regular Expression : Examples (Contd...)

```
bhatt@SE-0 [F] >>grep '[0-9]' testfile  
alalal  
456  
10000001
```

```
bhatt@SE-0 [F] >>grep '^4' testfile  
456
```

```
bhatt@SE-0 [F] >>grep 'l$' testfile  
alalal  
10000001
```




Regular Expression : Examples (Contd...)

```
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
```



Choosing Match Options

- i : matches one, or more, repetitions of RE
- l : matches none, or one, occurrence of RE
- v : select lines that do not match
- w : select lines that contain whole words only

To find out at *how many terminals* a user is logged in at the moment.

```
who | grep username | wc -l > count
```



Options

- *bhatt@SE-0 [F] >>grep -v 'a1' testfile*
aaa
456
10000001
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



Options : Examples - 1

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
```



Options : Examples – 2

- 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
```



Using find

Syntax

find *path_expression* *action*

File type options:

- f text file
- c character special file
- b blocked files
- p pipes

string command to *find* if there are *ascii strings* in a certain object or binary file

eg: string binaryFileName | more



Few Typical Usages - 1

- List all the files in the current directory.

```
bhatt@SE-0 [F] >>find . -print  
  » ./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.



Few Typical Usages - 2

- 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
```



Few Typical Usages - 3

- Finding out the subdirectories under a directory.

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

```
» ../M
```

```
» ../M/M1
```

```
» ../M/M2
```

```
» ../M/M3
```

```
» ../M/RAND
```



Few Typical Usages - 4

- 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*



Sort Tool

Unix treats each line in a text file as data.

- `sort` : note that *identical lines* from the input text are repeated in the sorted output.
- `uniq` : to get sorted output with *unique lines*.
- `ctags` and `etags` : commands are useful for looking for *patterns* like a c-function call.
- `split` : command *splits a file* into smaller sized segments
- `merge` : command *support merge sort* and for assembling a set of smaller files.



Sort Tool : Example - 1

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

aaa

bbb

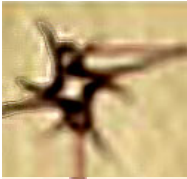
aba

^d terminates the inputsee the output below

aaa

aba

bbb



Sort Tool : Example - 2

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

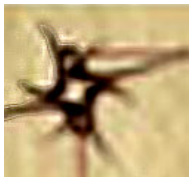
```
10000001
```

```
456
```

```
This is a test file.
```

```
alalal
```

```
aaa
```



Sort Tool : Example - 3

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.  
alalal  
aaa
```




Sort Options

- r : To get a sort in *decreasing order*
- b : To *ignore leading blank spaces*
- f : To *fold uppercase* to lowercase for comparison
- i : To *ignore characters* outside the ASCII range
- n : To *perform numeric comparison*. A number may have leading blanks



Split Command

- *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 .