BASIC COMMANDS
DAY 1

1. cal

With cal command, calendar of a month can be printed.
However the entire year's calendar can also be printed, from the year 1 to 9999 can be displayed with this command.
e.g. $ cal 2004
to stop the screen from scrolling we can use a pipe.

```
$ cal 2004 | more                          #less can also be used for the same purpose
```

2. date

```
$date
Wed Jan  7 14:35:22 IST 2004
```

We can also print date with suitable format specifiers as arguments. Each format is preceded with a + symbol, followed by % operator.

m	prints the month in number
h	prints the month name
d	day of the month (1 to 31)
y	last two digits of the year
H,M & S	hour, minute and second resp.

e.g.

```
$date +"%d %h"
06 January
```

3. **who**

Linux maintains an account of all current users of the system.
who command displays the list of users logged in, their name, their device name and the time since they have logged in.
the terminal names are actually special files representing devices. These files are available in /dev.
-H option with who displays the column headers.
-u option provides a more detailed list.

4. tty

In linux even terminals are treated as files, tty stands for tele type.
The terminal file name is resident in /dev directory.

5. uname

Every machine connected to a network has a name. The u name command with –n option tells the machine name.
Same command when used with –r shows operating system's version number.

6. passwd

passwd command is used to change the password. When we enter a password, the string is encrypted by the system, and the encryption is stored in the file /etc/shadow (file named shadow in /etc directory).

7. echo

echo command is used to display its argument, both quoted and unquoted.

8. tput

tput clear is used to clear the display.
tput can be used with cup argument to position the cursor at row number 10 and column 20.
tput **cup** 10 20

We can also highlight our text by using smso and rmso arguments for highlighting and terminating highlighting resp.
tput **smso**

```
echo “come to the web”
tput rmso
```

9. bc

bc command is used to implement the calculator.

```
$bc
12 + 5 <enter>
17
<ctrl –d >
```

Many calculations can also be performed by using semicolon.
For getting output in decimal point we have to set the scale.

```
scale = 2
17/7
2.42
```

the graphical version of calculator in linux is a **xcalc** command.

10. script

this commands lets us record our login session in a file **typescript** in the current directory. All the commands, their output and the error messages are stored in the file for later viewing.

```
$script
script started, file is a typescript
……
……
$exit
script done, file is typescript
```

script command overwrites any previous typescript that may exist. If you want to append to it, or want to use a different log file, then we can use following options

```
script –a          #append to existing typescript
script logfile     #logs activities to file logfile
```

11. spell and ispell

spell takes the name of the file as an argument and generates a list of all spellings that the program recognizes as mistakes. Spell command is not interactive and we can’t correct the mistakes on line.
ispell is interactive spell check, it highlights the line and the word, suggests some alternatives and offers other options also.

12. ls

To list the names of files available in the directory we use the ls command.

```
$ls
README
chap01
chap02
```

ls displays a list of three files. The files are arranged alphabetically with upper case having precedence over lower.
For detailed information we use ls with an option –l, between thle command and file names.

```
$ ls –l chap*
-rw-r—r--   1 kumar   group   5609    apr    23    09:30   chap01
-rw-r—r--   1 kumar   group   26109   may    14    18:00   chap02
```

The argument beginning with a hyphen is known as an option. Eg. –l used with ls. An option changes the default behaviour of a command.

13. cat

cat command can be used to see the contents of a file (similar to type command in DOS)

$ cat README
remember to terminate every command with <enter> key.
The completion of every command is indicated by the $ prompt

14. exit

exit command ends the session.

Note:
We have complete flexibility to work in Linux:

1. we can combine commands using pipes (|) and redirection(>).
2. a command can be entered in multiple lines. In that case a secondary prompt appears (>)
3. We can enter commands before previous command has finished. There is a type ahead buffer which stores all these commands for execution after the previous command or program has completed its run.
4. we can have online help using the **man** command. These refer to manual pages. If we want to know about any command we can type man <command name> and the manual pages follow.

man –e option gives online introduction to the command.
If we have no idea about the command to use in given situation we use –k option that accepts the keyword and man gives the name and short description from all manual section the command that will do the job.
$ man –k inode

NAVIGATION OF FILE SYSTEM & HANDLING ORDINARY FILES

A linux file is a store house of information. A file contains a sequence of bytes, it can be source code, or an executable code or anything else for linux it's a file. The concept extends to each and every device also – the hard disk, printer, CD ROM, terminal etc. the shell is also a file and so is the kernel. Linux treats the main memory also as a file. The commands that we use are also files.
There are three categories of files:

- Ordinary files: contain only data
- Directory files: contains other files and directories
- Device files: represents all hardware devices.

Ordinary file

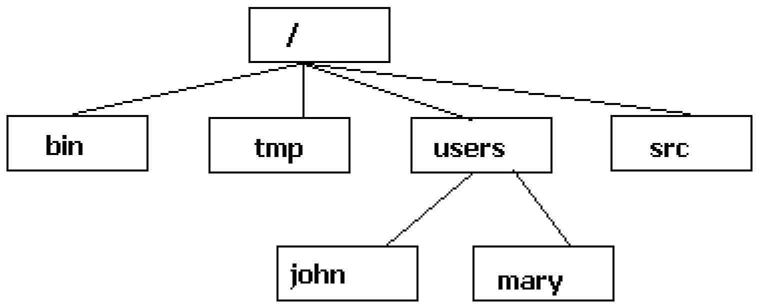
It is a traditional file. It consists of a stream of data resident on some permanent magnetic media. It includes data, source programs, object and executable code, all commands, any files created by user. Commands like cat, ls etc. are treated as ordinary files. This file is also called regular file. The most common type of ordinary file is text file.

Directory file
Directory contains no external data, but keeps some details of the files and subdirectories that it contains. The directory file contains two fields for each file- the name of the file, the identification number (the inode)

Device file
The device file is a special file that contains physical devices like printer, floppy drives , hard disks, terminals etc. Any output directed to the file will be reflected onto the respective physical device associated with the filename. The kernel takes care of this by mapping special filenames to their respective devices.

A file name can have upto 255 characters. Files may or may not have extensions. They can any ASCII character except /. The following are the valid file names:
.last_time   list.   ^V^B^D-++b   -{}[]
but it is recommended only the following characters are used in filenames:
alphabets and numerals
the period (.), the hyphen (-) and the underscore(_)
In unix all files are related to each other. Its is a collection of related files organized in a hierarchical structure.
In this structure there is a supreme, which serves as a reference point for all files. This is called root and is represented by /. Root is actually a directory file and has number of subdirectories.

THE UNIX FILE SYSTEM TREE



COMMANDS FOR FILE SYSTEM

1. pwd

this command displays the present working directory.

2. mkdir dirname

we can create directories with mkdir (**m**ake **dir**ectory) command.

3.cd dirname

cd is the change directory command. If no argument is supplied it reverts to home directory. If argument is supplied which is the pathname of the file.

4. rmdir dirname

rmdir stands for remove directory. We can't delete a directory unless it's empty. Unless we are placed in a directory which is hierarchically above the sub directory we have chosen to remove it can't be deleted.

5. ls

this command is used to obtain a list of all files in the current directory.

Options to ls

-x	displays multi-columnar output
-F	marks executables with * and directories with /
-r	sorts files in reverse order
-a	shows all files including ., .. and those beginning with a dot.
-R	recursive listing of all files in subdirectories.
-l	long listing showing seven attributes of a file.
-t	sorts files by modification time
-u	sorts files by access time (when used with –t option)
-i	shows inode number of a file

6. cat

this command is also used to create a file. Cat command followed by the > character and the file name creates file and puts all the contents entered at command prompt to the file. It is similar to copy con command of DOS. To end the input to the file ctrl-d character is also entered.

$cat > filename – is used to create file (ctrl+d is used to save the file)

$cat filename – is used to display contents of the file

7. cp

the copy command copies a file or a group of files. It creates an exact image of the file on the disk with a different name. The syntax requires at least two filenames to be specified in the command line.

$cp file_source file_dest

if the destination file doesn't exist, it will first be created before copying. If there is only one file to be copied, the destination can be either an ordinary or a directory file.

$cp /usr/Sharma/.profile   .   #destination is current directory.

Interactive copying (-i) the –i option warns the user before overwriting the destination file.

Copying directory structures (-r) copying the entire directory is made possible with –r (recursive) option.

8. rm

this command is used to remove files. This can delete more than one file with a single instruction.

$rm chap1 chap2 chap3

Interactive deletion -i option makes command ask user for confirmation before removing each file.

Recursive deletion –r option removes all files and sub directories.

rm –r * will remove all files in the current directory.

9. mv

this is move command and is used to rename a file or directory and moving a group of files to a different directory. mv doesn't make a copy of file just renames it.

To rename file c01 to man01

$mv c01 man01

A group of files can be moved but only to a directory. the following command moves three files to the progs directory

$mv c01 c02 c03 progs

-i option available for interactive mode

-v option is a verbose option it prints name of each file as action being taken.

10.more
If the file contents are too large for its contents to fit in one screen, it will scroll off the screen. Even pressing Ctrl-s for stopping doesn't work. The **more** command allows the user to view a file, one screen at a time.
eg. $more chap01 #press q to exit

Internal commands of more and less commands

Spacebar	scrolls forward one screen
F	scrolls forward one screen
B	scrolls backward one screen
/pattern	searches for pattern forward
?pattern	searches for pattern backward
n or N	repeats last search
v	starts up vi editor
.	repeats previous command
:n	skips to next file specified in command line.
:p	skips to previous file specified in command line
q	exits from more

11.lpr
lpr option is used to print a file. One has to spool a job along with others in print queue.
lprm is used to delete print jobs from the spool queue.
lpq is used to show the printer spool queue.
lpd is the line printer daemon.

12.file
to know the file types we can execute file command with arguments as the file name.

13.wc
this is used for line, word and character counting in the same order.
-l option counts the number of lines only.
-w option counts the number of word
-c option counts the number of characters.
When used with multiple filenames, wc produces a line for each file as well as total count.

14.split

this command breaks up its input into several equi-line segments. Split, by default breaks up a file into 1000-line pieces
it groups files into xaa, xab,xac….. and upto xaz and then xba,xbb….
We can have 676 files the last ones name would be xzz.
We can specify the size of each file by using number as an option.
split –45 chap   #each file contains 45 lines
we can also select our own primary file name other than x, this can be done by putting the name as the last argument.
split chap small
This creates files smallaa, smallab, ...etc.

15.cmp
used to compare two files. We often require to know whether two files are identical so that one of them can be deleted.
$cmp chap01   chap02
chap01 chap02 differ: char 9, line 1.
The two files are compared byte by byte, the location of first mismatch is echoed on the screen.
-l (list) option gives a detailed list of byte number and the differing bytes in octal for each character that differs in both files.
If two files are identical cmp displays no message it just returns $ prompt

16.comm
comm. Command requires two sorted files, and compares each line of the first file with its corresponding line in the second. It finds what is common.
The command displays three columnar output. 1st column contains lines unique to first file, while second column shows lines unique to the second file. The third column displays two lines common to both files.
We can use options –1, -2, -3 to drop a particular column.

17.Diff f1 f2
it is used to convert one file to another. This command tells which lines in one file have to be changed to make the two files identical.

Ques to practise
Create Directories

Yourname->Course -> mca , bca, bba
Mca - > subjects-> python, java, dbms
Bca -> subjects -> IT, c,c++
Bba-> subjects -> hr,marketing, sales

Create Files
Course -> mca , bca, bba
Mca - > subjects-> python->**pythonlabfile**, java->**javalabfile**, dbms->**dbmslabfile**
Bca -> subjects -> IT, c,c++
Bba-> subjects -> hr ->**hrlabfile**,marketing, sales->**saleslabfile**

THE SHELL
DAY 2

Shell is the agency that sits between the user and the linux system. When we log into a linux system, we see series of messages followed by $ prompt. This is nothing but the shell. It is actually the **sh** command that is present in /bin directory. it is the bash shell.
Shell swings into action when we enter something from keyboard. When we issue a command, shell is the first agency to acquire the information. The following activities are typically performed by the shell in each cycle.

- It issues the $ prompt and waits for you to enter a command.
- After a command has been entered, the shell scans the command line for some special characters and then rebuilds the command line after process is complete.
- The command is then passed on the kernel for execution, and the shell waits for its completion.
- The $ prompt appears, and the shell waits for you to enter the next command.

PATTERN MATCHING – THE WILD CARDS

1. **The * and ?**

The * known as the metacharacter, is one of the characters of shell's special set. This character matches any number of characters (including none). When this character is appended to string chap, the pattern chap* expands into all files, in which first four characters constitute the string chap. * may occur anywhere in a file name and not merely at the end. Thus, **chap** matches all following filenames:
chap, newchap chap03 chap03.txt

e.g. wc * command makes a word count of each and every file in the current directory.

the next metacharacter is the ?, which matches a single character. When used with same string chap?, the shell matches all five character filenames beginning with chap. Placing another ? at the end of this string creates the pattern chap??, where chap is followed by two characters.

2. The character class [ ]

Shell provides a character class that is very restrictive in terms of compact expressions. The character class uses two more metacharacters represented by a pair of brackets []. You can have as many characters inside this enclosure as we want, but matching takes place for only a single character in the class. For eg. A single character expression, taking the values 1, 2 or 4, can be represented by the expression [124].

```
$ls –x chap0[124]
```

range specification is also possible inside the class with a – (hyphen)

e.g. `$ls –x  chap[x-z]`

```
chapx chapy chapz
```

3. Negating the character class

Placing ! (bang) at the beginning of the class reverses the matching character i.e. it matches all other characters except the ones in the class.

Eg. The pattern [!A-Za-z]* matches all files where the first character is not alphabetic.

***.**[!Z] all files except those ending with Z

4. when wild cards lose their meaning

some of wild card characters have different meanings depending on where they are placed in the pattern. The metacharacters * and ? lose their meaning when used inside the class and are matched literally. Similarly – and ! also lose their significance when placed outside the class or at wrong places inside the class.

5. matching the dot

the * doesn't match all the files beginning with a . (dot) or the / o a pathname. If you want to list all the hidden files in your directories having at least three characters after the dot, then dot must be matched explicitly.

```
$ls –x .???*
.exrc     .news_time          .profile
```

if file name contains a dot anywhere but at the beginning, it need not be matched explicitly.

```
$ls –x emp*lst
emp.lst      empl.lst          emp22lst         emp2.lst
```

6. using rm with *

when using shell metacharacters especially **,** if instead of typing rm chap* we type rm chap * it will remove file chap and then all files in the directory. this is because shell treats * as separate argument.

ESCAPING –THE BACKSLASH (\ )

(\) is used to remove the special meaning of any metacharacter placed after it.

e.g. if we name a file named chap* and then try to use ls command with it or rm command it will take into account this file and all files beginning with *. So removal of this file will remove all files beginning with chap.
For negating the meaning of special characters we should use \ before the metacharacter.

```
 ls –x chap\*
chap*
 rm chap\*
ls –x chap\*
chap* not found
```

similarly with [ ] and other special characters.

Suppose we have a long chain of commands we can split the command line by hitting enter key, but only after \ escapes the key.

```
$ wc –l chap04 note \<enter>
>unit01  <enter>
```

the appearance of > (secondary prompt) indicates the command is not complete.

I/O Redirection and file descriptors

- **> output redirection**
- **< input redirection**
- **<< append**

As you know I/O redirectors are used to send output of command to file or to read input from file. Consider following example

```
$ cat > myf
This is my file
^D (press CTRL + D to save file)
```

Above command send output of cat command to myf file

```
$ cal
```

Above command prints calendar on screen, but if you wish to store this calendar to file then give command

```
$ cal > mycal
```

The cal command send output to mycal file. This is called *output redirection.*

```
$ sort
10
-20
11
2
^D
-20
2
10
11
```

sort command takes input from keyboard and then sorts the number and prints (send) output to screen itself. If you wish to take input from file (for sort command) give command as follows:

```
$ cat > nos
10
-20
11
2
^D
$ sort < nos
-20
2
10
11
```

First you created the file *nos* using cat command, then *nos* file given as input to *sort* command which prints sorted numbers. This is called *input redirection*.
In Linux (And in C programming Language) your keyboard, screen etc are all treated as files. Following are name of such files

Standard File	File Descriptors number	Use	Example
Stdin	0	as Standard input	Keyboard
Stdout	1	as Standard output	Screen
Stderr	2	as Standard error	Screen

By default in Linux every program has three files associated with it, (when we start our program these three files are automatically opened by your shell). The use of first two files (i.e. stdin and stdout) , are already seen by us. The last file stderr (numbered as 2) is used by our program to print error on screen. You can redirect the output from a file descriptor directly to file with following syntax
Syntax:
file-descriptor-number>filename
Examples: (Assemums the file **bad_file_name111** does not exists)

```
$ rm bad_file_name111
rm: cannot remove `bad_file_name111': No such file or directory
```

Above command gives error as output, since you don't have file. Now if we try to redirect this error-output to file, it can not be send (redirect) to file, try as follows:

```
$ rm bad_file_name111 > er
```

Still it prints output on stderr as *rm: cannot remove `bad_file_name111': No such file or directory*, And if you see er file as **$ cat er** , this file is empty, since output is send to error device and you can not redirect it to copy this error-output to your file '*er*'. To overcome this problem you have to use following command:

Redirection of Standard output/input i.e. Input - Output redirection

Mostly all command gives output on screen or take input from keyboard, but in Linux (and in other OSs also) it's possible to send output to file or to read input from file.
For e.g.
$ ls command gives output to screen; to send output to file of ls command give command

$ ls > filename
It means put output of ls command to filename.
There are three main redirection symbols **>,>>,<**
(1) > Redirector Symbol
Syntax:
Linux-command > filename
To output Linux-commands result (output of command or shell script) to file. Note that if file already exist, it will be overwritten else new file is created. For e.g. To send output of ls command give
$ ls > myfiles
Now if '**myfiles**' file exist in your current directory it will be overwritten without any type of warning.
(2) >> Redirector Symbol
Syntax:
Linux-command >> filename
To output Linux-commands result (output of command or shell script) to END of file. Note that if file exist , it will be opened and new information/data will be written to END of file, without losing previous information/data, And if file is not exist, then new file is created. For e.g. To send output of date command to already exist file give command
$ date >> myfiles
(3) < Redirector Symbol
Syntax:
Linux-command < filename
To take input to Linux-command from file instead of key-board. For e.g. To take input for cat command give
$ cat < myfiles
You can also use above redirectors simultaneously as follows
Create text file sname as follows
$cat > sname
vivek
ashish
zebra
babu
Press CTRL + D to save.
Now issue following command.
$ sort < sname > sorted_names
$ cat sorted_names

```
ashish
babu
vivek
zebra
```

In above example sort (**$ sort < sname > sorted_names**) command takes input from sname file and output of sort command (i.e. sorted names) is redirected to sorted_names file.

Try one more example to clear your idea:

$ tr [a-z] [A-Z] < sname > cap_names
$ cat cap_names

```
VIVEK
ASHISH
ZEBRA
BABU
```

tr command is used to translate all lower case characters to upper-case letters. It take input from sname file, and tr's output is redirected to cap_names file.

Future Point : Try following command and find out most important point:

$ sort > new_sorted_names < sname
$ cat new_sorted_names

BASIC FILE ATTRIBUTES
DAY 3

- ls –l option displays the permissions associated with each file

```
-rw-r-xr--     1      kumar    metal   19514          may 10 13:45  chap01

permission   links    owner    group   size of file in bytes   last modification time
```

- ls –ld

d optin is used to list only directory attributes.
drwxr-xr-x
1st character in the 1st column is d for directory, - for ordinary file, b or c for device files

- **FILE PERMISSIONS**

There is three tired file protection system that determines the access rights that you have for the file. Each tier represents a category and consists of a string of r, w, x
r – read permission i.e. we can read file using cat command
w – write permission i.e. we can direct some output to it
x- execute permission i.e. file can be executed as a program

-rwxr-xr- -
first group (-rwx)

it is an ordinary file contains all three permissions. Three permission valid for the **OWNER** of the file.(kumar)
Second group (r-x)
Absence of write permission by **GROUP OWNER** of file. Group owner is metal
Third group (r - -)
Applicable to **OTHERS** i.e. who are neither kumar nor belong to metal group. Can only read the file.

- **chmod**

change file permissions using this. It sets the three permissions for all three categories of users.
Syntax
chmod category operation permission filename(s)

category	operation	permission
u – user	+ -> assign	r – read
g-group	- -> remove	w-write
o-others	= -> absolute	x-execute
a-all		

$chmod u+x start will add execute permission for the user or the owner of the file.

$chmod ugo +x start = $chmod a+x start = chmod +x start

e.g. $chmod u-x,go+r start;ls –l start

chmod ugo=r start will remove all other permissions on start only put read permissions for all categories of users.

Octal notation

Shorthand notation for chmod
r – 4 , w – 2, x – 1

$chmod u=rwx, g=rw, o=x start
is same as
$chmod 761 start
ugo

note: default permissions of directory is 755 or drwxr-xr-x

$chmod –R a+x progs
where progs is a directory. –R option makes all files and subdirectories in the directory progs as executable by all users.

<u>Simple filters</u>
<u>DAY 4</u>

1. **head**

displays the beginning of the file.
It by default displays 10 lines/records of a file

- $head –3 emp.lst

displays first three records of file emp.lst

- $head –1 emp.lst | wc –c

finds the record length of the file.

- $vi `ls –t | head –1`

edits the last modified file using vi editor
it is also possible to pick up bytes

```
head –c 512 emp.lst          #first 512 characters
head –c 1b emp.lst           #first 512 characters
head –c 2m emp.lst           #first two
blocks(1MB each)
```

2. **tail**

displays the end of the file works similar to head
$tail –5 emp.lst
displays last 5 lines/records of the file emp.lst

3. **cut**

it slits the file vertically.cut identifies both fields and columns
e.g.
$cut –c 6-22, 24-32 shortlist
   will cut columns 6-22 and 24-32 of file shortlist

if in the file we are using '|' as the delimiter for fields and we want to cut the fields
$cut –d \| -f 2,3 shortlist > cutlist1
   -d for delimiter and –f for fields. This command will cut 2nd and 3rd fields
$cut –d "|" –f 1,4- shortlist > cutlist2
   this option will cut 1st and 4th – last fields of the file shortlist and place in cutlist2

4. paste

syntax :
$paste file1 file2
this will concatenate two files vertically. By default paste used tab character for pasting files. We can specify delimiter using –d option.
$paste –d\| file1 file2

if file 2 doesn’t exist data can be supplied through standard input

```
$ cut –d “|” –f 1,4- file1 | paste –d “|” > file2-
```

we can also reverse the order of pasting by altering the location of – sign.

```
$cut –d “|” -f 1,4- file1 | paste –d “|” –file2
```

5. sort

this is used for ordering the files on the basis of first character of each line in the file.

-r option reverses the sort order

-o option stores output in a file given as argument

-c checks whether file is sorted or not. If sorted displays prompt otherwise the first disorder found.

-t option sorts using fields.

-n option sorts numeric fields

-u purges duplicate records and shows only unique records

e.g. `cut –d “|” –f 3 emp.lst |sort –u |tee desg.lst`

-m option merges files provided the original files are sorted individually sort –m a1 a2 a3

e.g. $sort –t\| +1 shortlist
sorts on 2nd field of file shortlist
note: we can also sort on secondary keys
$sort –t \| +2 –3 +1 shortlist.
We can sort on specific columns in a field.
$sort -t “|” +4.6-4.8 short list

6. uniq command

this command searches/fetches one copy of each record and writes it to standard output. It requires sorted file as input.
$sort filename | uniq > uniqlist
$cut –d “|” -f3 emp.lst | sort | uniq –u

7. nl filename

it is used for adding line numbers to a file

8. tr (tr “|” “-“ < filename)
tr “| “-“ < filename > newfile

tr option is used for translating characters.
$tr options expression1 expression2 standard input this filters manipulates individual characters in a file

$tr '|/' '~-' < emp.lst | head –3
this option replaces | by ~ and / by – in the first three lines and displays on output
-d option when used with tr deletes a particular character in a file
$tr –d '|/' <emp.lst
deletes all | and / characters.
-s option squeezes multiple consecutive occurrences of its argument to single character.
$tr –s "chartoreplace" < emp.lst | head –3
-c it complements the set of characters in the expression
e.g. to delete all characters except |, / we can combine –d and –c
$tr –cd "|/" < emp.lst

GREP COMMNADS
DAY 5

1. **grep** is the most popular, it scans a file for the occurrence of the pattern, and can display the selected pattern, the line numbers in which they are found, or the filenames where the pattern occurs. Grep

can also select lines not containing the pattern.

Syntax: grep options pattern filenames

If a pattern is not found prompt returns.

OPTIONS

a) **counting occurrences (-c)** to count the number of occurrences of a particular pattern. This option will not display the lines at all.

b) **Displaying line numbers(-n)** to display the line numbers containing the pattern along with the lines.

c) **Deleting lines (-v)** this option selects all but the lines containing the pattern.

e.g. grep –v ‘director’ emp.lst > other list

d) **displaying filenames (-I)** displays only the file names of files where a pattern has been found.

e) **Ignore case (-i)** this option ignores case for pattern matching

f)**–An** displays line and n lines after matching line.

g) –**Bn** displays line and n lines before matching line.

h) –**n** displays line and n lines above and before matching line.

Symbols	significance
*	matches zero or more occurrences of previous character
.	matches a single character
[pqr]	matches a single character p,q or r
[0-9]	matches a single character within the ascii range represented by 0 to 9
[^pqr]	matches a single character which is not p,q or r
^pat	matches pattern pat at the beginning of the line.
pat$	matches pattern pat at the end of the line.

e.g. grep “[aA]g[ar][ar]wal” filename

agarwal

agrawal

egrep (extending grep)
it offers all the options of grep, its most useful feature is that it facilitates to specify more than one pattern for search separated by a | (pipe)

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

```
$ egrep –I ‘agg?[ar]+wal’ emp.lst
aggarwal
agarwal
agrawal
```

```
$egrep ‘sengupta|dasgupta’    emp.lst
$egrep ‘(sen|das)gupta’ emp.lst
```

egrep offers option –f (file) to take multiple patterns from the file.

e.g. $cat pat.lst
admin|accounts|sales
$egrep –f pat.lst emp.lst

fgrep (multiple string searching)
like egrep fgrep accepts multiple patterns both from command line and file but unlike grep and egrep it doesn't accept regular expressions. It is much faster, and it should be used when using fixed strings.

```
$cat pat1.lst
sales
personnel
admin
$fgrep –f pat1.lst emp.lst
```