Linux Commands

1) cal : The Calendar

cal is a handy tool that you can invoke any time to see the calendar of any specific month or complete year.

Example: 1) $ cal 03 2008   2) $ cal 2014

2) date: Display the System Date

You can display the system date with date command, which shows the date and time.

Arguments of date command:

[m]: - display month (in numeric)
[h]: - display month name
[d]:- display day of the month
[y]:- last two digits of the year
[H], [M], [S]: - hour, minute and second, respectively

Example: 1) $ date   2) $date +%m

When you use multiple format specifies, you must enclose them within quotes, and use a single + symbol before it.

3) echo: Display Messages or a line of text

Syntax: echo [*OPTION*]... [*STRING*]...

Options:

[-n]:- do not output the trailing newline
[-e]:-enable interpretation of the backslash-escaped characters listed below
[-E]: - disable interpretation of those sequences in STRINGs

Without **-E**, the following sequences are recognized and interpolated:

\NNN:- the character whose ASCII code is NNN (octal)
\\**:**- backslash
\a:- alert (BEL)
\b:- backspace
\c:- suppress trailing newline
\f:- form feed
\n:- new line
\r:- carriage return
\t:- horizontal tab
\v:- vertical tab

EXAMPLE:

1) echo command

```
echo "Welcome world of linux"
```

The above command will print as Welcome world of linux

2) To use backspace:

```
echo -e "Welcome \bworld \bof \blinux"
```

The above command will remove space and print as Welcomeworldoflinux

3) To use tab space in echo command

```
echo -e "Welcome \tworld \tof \tlinux"
```

The above command will print as Welcome world of linux.

4) bc: The Calculator

bc is a language that supports arbitrary precision numbers with interactive execution of statements. There are some similarities in the syntax to the C programming language. A standard math library is available by command line option. If requested, the math library is defined before processing any files. **bc** starts by processing code from all the files listed on the command line in the order listed.

Example:

1) bc
   12 + 5 <entre>
   <ctrl-d>
2) bc
   12 * 12; 2^4 <entre>
3) scale = 2
   17/5 <entre>
4) scale=2
   17/5
5) ibase = 2
   11001010
   202

There are special variables, **scale, ibase, obase. ibase and obase** define the conversion base for input and output numbers.

5) script: Record your Session

Records everything printed on your screen. The record is recorded to the filename, if no filename is specified results are recorded to the file transcript.

Syntax: *script [-a] filename*

[-a] :- Append the session record to filename, rather than overwrite it.

Filename: - The name of the file that you wish the results to be stored.
Example: script –a myfile.txt

6) who: Login Details

Display the user name who is currently working on terminal.

Example: $ who <entre>

Using –H with this command will display the column header. Another command is who am I that display a single line of output only, i.e., the login details to the user who invoked this command.

7) Uname: Know Your Machine's Name

If your machine connected with network than it must have name.

Example: $ uname

8) tty: Knowing Your Terminal

Print the file name of the terminal connected to standard input.

Example: $ tty

9) man

The man command is short for manual and provides in depth information about the requested command or allows users to search for commands related to a particular keyword.

Syntax: *man [-] [-k keywords] topic*

[-] Displays the manual without stopping.

[**-k keywords]** Searches for keywords in all of the manuals available.

[topic] Displays the manual for the topic or command typed in.

Example: $ man mkdir

Lists help information on the mkdir command.

10) Info: Read Info documents.

Syntax: info [OPTION]... [MENU-ITEM...]

The first non-option argument, if present, is the menu entry to start from; it is searched for in all dir files along INFOPATH. If it is not present, info merges all dir files and shows the result. Any remaining arguments are treated as the names of menu items relative to the initial node visited.

Example: $ info ls

Gives you the info on the ls command

11) Passwd: Allows you to change your password.

Syntax: passwd

Example: $ passwd <enter>

Entering just passwd would allow you to change the password. After entering passwd you will receive the following three prompts:

Current Password:
New Password:
Confirm New Password:

Each of these prompts must be entered and entered correctly for the password to be successfully changed.

12) **logout**
Root or admin user can logout any user forcefully. Logout from your session.
Example: **$ logout**

13) **wc :**Short for word count, wc displays a count of lines, words, and characters in a file.
Syntax: *wc [-c | -m | -C] [-l] [-w] [file ...]*
[-c] Count bytes.
[-m]Count characters.
[-C] Same as -m.
[-l]Count lines.
[-w] Count words delimited by white space characters or new line characters. Delimiting characters are Extended Unix Code (EUC) characters from any code set defined by iswspace()
[file] Name of file to word count.
Example: $ wc myfile.txt

14) **pwd:** Checking your current directory
Display working directory of user.
Example: **$ pwd**

15) **ls:** Lists the contents of a directory.
Syntax: ls [-a] [-d] [-i] [-l] [-r] [-R] [-s] [-t] [-u] [-x] [pathnames]
[-a] Shows you all files, even files that are hidden (these files begin with a dot.)
[-d] Listing of a directory.
[-F] Mark directories with a slash (/), executable files with a trailing asterisk (*).
[-l] Shows you huge amounts of information (permissions, owners, size, and when last modified.)
[-r] Sorts file in reverse order.
[-R] Recursive listing of all files in sub-directories.
[-t] Shows you the files in modification time.
[-u] Sorts file by access time (when used with the –t option).
[-x] Displays files inmulti columnar output.
[-i] Shows inode number of a file.

[pathnames] File or directory to list.

Matching Filenames with Patterns

The file is a container for storing information. An ordinary file or regular file is the most common file type. An ordinary file itself can be divided into two types: Text File and Binary File.

A directory contains no data, but keeps some details of the files and subdirectories that it contains.

File name should contain alphabetic characters, numeric, period(.), hyphen (-) and underscore (_).

16) ps: Reports the process status.
Syntax: ps [-a] [-A] [-c] [-d] [-e] [-f] [-j] [-l] [-L] [-P] [-y]

[-a] List information about all processes most frequently requested: all those except process group leaders and processes not associated with a terminal.
[-A] List information for all processes. Identical to -e, below.
[-e] List information about every process now running.
-f Generates a full listing.
-j Print session ID and process group ID.
-l Generate a long listing.

17) cat: Use for create a file and also used to display the content of the file on the terminal.
For crete a file:
Symbol: cat > filename.txt
Example: $ cat > myfile.txt <entre>
Hi…… Welcome the word of linux with Ubuntu…..
<ctrl –d>
$

Example: $ cat myfile.txt <entre>
This command displays the content of the file.

18) The touch command is the easiest way to create new, empty files. It is also used to change the timestamps (i.e., dates and times of the most recent access and modification) on existing files and directories.

Syntax:
touch [option] file_name(s)

When used without any options, touch creates new files for any file names that are provided as arguments (i.e., input data) if files with such names do not already exist. Touch can create any number of files simultaneously.

Thus, for example, the following command would create three new, empty files named file1, file2 and file3:

```
touch file1 file2 file3
```

A nice feature of touch is that, in contrast to some commands such as cp (which is used to copy files and directories) and mv (which is used to move or rename files and directories), it does not automatically overwrite (i.e., erase the contents of) existing files with the same name. Rather, it merely changes the last access times for such files to the current time.

19) more: Paging Output

The man command displays its output a page at a time. It is used to view a text file one page at a time, press spacebar to go to the next page.

more *filename or command name :* show the document one page at a time

Example: **more -10 filename** will show 10 lines for every page

20) less = is much the same as more command except:

a) You can navigate the page up/down using the less command and not possible in more command.

b) You can search a string in less command. (use /keywordto search)

c) "more" was fairly limited, and additional development on "more" had stopped

d) It uses same functions as vi editor

Example: **less filename or command name**

21) head:

head command is used to display the first ten lines of a file, and also specifies how many lines to display.

SYNTAX:

The Syntax is

```
head [options] filename
```

OPTIONS:

-n	To specify how many lines you want to display.
-n number	The number option-argument must be a decimal integer whose sign affects the location in the file, measured in lines.
-c number	The number option-argument must be a decimal integer whose sign affects the location in the file, measured in bytes.

EXAMPLE:

1. head index.php

   This command prints the first 10 lines of 'index.php'.

2. head -5 index.php

   The head command displays the first 5 lines of 'index.php'.

3. head -c 5 index.php

   The above command displays the first 5 characters of 'index.php'.

22) tail: tail command is used to display the last or bottom part of the file. By default it displays last 10 lines of a file.

SYNTAX:
The Syntax is
tail [options] filename

OPTIONS:

-l	To specify the units of lines.
-b	To specify the units of blocks.
-n	To specify how many lines you want to display.
-c number	The number option-argument must be a decimal integer whose sign affects the location in the file, measured in bytes.
-n number	The number option-argument must be a decimal integer whose sign affects the location in the file, measured in lines.

EXAMPLE:

1. tail index.php

It displays the last 10 lines of 'index.php'.

2. tail -2 index.php

   It displays the last 2 lines of 'index.php'.

3. tail -n 5 index.php

   It displays the last 5 lines of 'index.php'.

4. tail -c 5 index.php

   It displays the last 5 characters of 'index.php'.

Additional File Management Commands

23) mkdir: Making Directory
Directories are created with the mkdir command. The command is followed by names of the directories to be created.
Example: mkdir abc
mkdir abc xyz pqr (Three directories created)

24) rmdir: Removing Directory
The rmdir command removes directories. You simply have to do this to remove the directories.
Example: rmdir abc
rmdir user/data

25) cd: cd command is used to change the directory.
Syntax: cd [directory | ~ | ./ | ../ | -]
OPTIONS:
-L: Use the physical directory structure.
-P: Forces symbolic links.

cd **..**

This will change to the parent-directory from the current working directory/sub-directory.

cd ~

This command will move to the user's home directory which is "/home/username".

26) cp: cp command copy files from one location to another. If the destination is an existing file, then the file is overwritten; if the destination is an existing directory, the file is copied into the directory (the directory is not overwritten).

Syntax:

```
cp [OPTIONS]... SOURCE DEST
cp [OPTIONS]... SOURCE... DIRECTORY
cp [OPTIONS]... --target-directory=DIRECTORY SOURCE...
```

OPTIONS:

-a	same as -d.
--backup[=CONTROL]	make a backup of each existing destination file
-b	like --backup but does not accept an argument.
-f	if an existing destination file cannot be opened, remove it and try again.
-p	same as --preserve=mode,ownership,timestamps.

EXAMPLE:

1. Copy two files:

```
cp file1 file2
```

The above cp command copies the content of file1.php to file2.php.

2. To backup the copied file:

```
cp -b file1.php file2.php
```

Backup of file1.php will be created with '~' symbol as file2.php~.

3. Copy folder and subfolders:

```
cp -R scripts scripts1
```

The above cp command copy the folder and subfolders from scripts to scripts1.

27) rm: rm linux command is used to remove/delete the file from the directory.

SYNTAX:

```
rm [options..] [file | directory]
```

OPTIONS:

-f	Remove all files in a directory without prompting the user.
-i	Interactive. With this option, rm prompts for confirmation before removing any files.
-r (or) -R	Recursively remove directories and subdirectories in the argument list. The directory will be emptied of files and removed. The user is normally prompted for removal of any write-protected files which the directory contains.

EXAMPLE:

1. To Remove / Delete a file:

```
rm file1.txt
```

Here rm command will remove/delete the file file1.txt.

2. To delete a directory tree:

```
rm -ir tmp
```

This rm command recursively removes the contents of all subdirectories of the tmp directory, prompting you regarding the removal of each file, and then removes the tmp directory itself.

3. To remove more files at once

```
rm file1.txt file2.txt
```

rm command removes file1.txt and file2.txt files at the same time.

28) mv: mv command which is short for move. It is used to move/rename file from one directory to another. mv command is different from cp command as it completely removes the file from the source and moves to the directory specified, where cp command just copies the content from one file to another.

SYNTAX:

```
mv [-f] [-i] oldname newname
```

OPTIONS:

-f	This will not prompt before overwriting (equivalent to --reply=yes).

mv -f will move the file(s) without prompting even if it is writing over an existing target.

-i Prompts before overwriting another file.

EXAMPLE:

1. To Rename / Move a file:

```
mv file1.txt file2.txt
```

   This command renames file1.txt as file2.txt

2. To move a directory

```
mv hscripts  tmp
```

   In the above line mv command moves all the files, directories and sub-directories from hscripts folder/directory to tmp directory if the tmp directory already exists. If there is no tmp directory it rename's the hscripts directory as tmp directory.

3. To Move multiple files/More files into another directory

```
mv file1.txt tmp/file2.txt newdir
```

   This command moves the files file1.txt from the current directory and file2.txt from the tmp folder/directory to newdir.

29) diff: diff command is used to find differences between two files.

SYNTAX:

```
diff [options..] from-file to-file
```

EXAMPLE:

Lets create two files file1.txt and file2.txt and let it have the following data.

Data in file1.txt	Data in file2.txt
HIOX TEST	HIOX TEST
hscripts.com	HSCRIPTS.com
with friend ship	with friend ship
hiox india	

Compare files ignoring white space:

```
diff -w file1.txt file2.txt
```

This command will compare the file file1.txt with file2.txt ignoring white/blank space and it will produce the following output.

```
2c2
        < hscripts.com
        ---
        > HSCRIPTS.com
        4d3
        < Hioxindia.com
```

30) comm: What is Common?

This command displays what is common between two files. It will display three columnar output in which first column contains unique lines of first file, and the second column shows unique to the second file. The third column displays common lines from both the files.

Example: comm. file1 file2

31) cmp: Comparing two files

The two files compared byte by byte, and the location of the first mismatch is enchoed to the screen. If two files are identical, cmp displays no message, but simply returns the prompt.

Example: cmp file1 file2

32) chmod:

chmod command allows you to alter / Change access rights to files and directories.

SYNTAX:

The Syntax is

chmod [options] [MODE] FileName

File Permission

#	File Permission
0	none
1	execute only

2	write only
3	write and execute
4	read only
5	read and execute
6	read and write
7	set all permissions

OPTIONS:

-c	Displays names of only those files whose permissions are being changed
-f	Suppress most error messages
-R	Change files and directories recursively
-v	Output version information and exit.

EXAMPLE:

1. To view your files with what permission they are:

   ```
   ls -alt
   ```

   This command is used to view your files with what permission they are.

2. To make a file readable and writable by the group and others.

   ```
   chmod 066 file1.txt
   ```

3. To allow everyone to read, write, and execute the file

   ```
   chmod 777 file1.txt
   ```

33) Piping and redirecting

Standard input and output constitute two separate streams that can be individually manipulated by the shell. There are two ways to redirecting output.

```
Example: ls | wc or wc –l <user.txt
         ls | wc –l  > linecount.txt
```

34)

tr:- Translating Characters

The tr filter manipulates individual characters in line.

Note - tr takes input only from standard input; it does not take a filename as argument.

Syntax: tr [options] expression1 expression2 standard input

Options:

[-d] Deleting Characters

[-s] eliminate all redundant spaces

[-c] complements the set of characters in the expression.

Example:

1) tr –d ' | / ' < xyz.txt | head –n 3
2) tr –s ' ' < xyz.txt | head –n 3
3) tr –cd ' | / ' < xyz.txt

34) cut: cut command is used to cut out selected fields of each line of a file. The cut command uses delimiters to determine where to split fields.

SYNTAX:

The Syntax is

cut [options]

OPTIONS:

-c	Specifies character positions.
-b	Specifies byte positions.
-d flags	Specifies the delimiters and fields.

EXAMPLE:

Lets create a file file1.txt and let it have the following data:

Data in file1.txt

This is, an example program,for cut command.

1. cut -c1-3 text.txt

Output:

Thi

Cut the first three letters from the above line.

2. cut -d, -f1,2 text.txt

Output:

This is, an example program

The above command is used to split the fields using delimiter and cut the first two fields.

35) paste:
paste command is used to paste the content from one file to another file. It is also used to set column format for each line.

SYNTAX:
The Syntax is

```
paste [options]
```

OPTIONS:

-s	Paste one file at a time instead of in parallel.
-d	Reuse characters from LIST instead of TABs .

EXAMPLE:

2. `paste test.txt>test1.txt`

   Paste the content from 'test.txt' file to 'test1.txt' file.

3. `ls | paste - - - -`

   List all files and directories in four columns for each line.

36) sort: sort command is used to sort the lines in a text file.

SYNTAX:
The Syntax is

```
sort [options] filename
```

OPTIONS:

-r	Sorts in reverse order.
-u	If line is duplicated display only once.
-o filename	Sends sorted output to a file.

EXAMPLE:

1. sort test.txt

   Sorts the 'test.txt'file and prints result in the screen.

2. sort -r test.txt

   Sorts the 'test.txt' file in reverse order and prints result in the screen.

37) uniq:- Locate repeated and nonrepeated lines

When you concatenate or merge files, you will find duplicate entries. To avoid this problem uniq command help you.

Since uniq requires a sorted file as input, the general procedure is to sort a file and pipe its output to uniq.

Options:

[-u] selects only lines that are not repeated.

[-d] selects only one copy of the repeated lines.

[-c] displays the frequency of occurrence of all lines, along with the lines.

Example:

1) cut –d"|" –f3 xyz.txt | sort | uniq –c
2) uniq xyz.txt

38) grep:

grep command selects and prints the lines from a file which matches a given string or pattern.

SYNTAX:

The Syntax is

```
grep [options] pattern [file]
```

OPTIONS:

-A	Print num lines of text that occur after the matching line.
-a	Don't suppress output lines with binary data, treat as text.
-b	Print the byte offset of input file before each line of output.
-c	Print's the count of line matched.
-d action	Define action for accessing the directories **read** read all files in the directories. **skip** skip directories.

	recurse recursively read all files and directories
-e pattern	Search for pattern.
-h	Print matched lines but not filenames.
-i	Ignore changes in case; consider upper- and lower-case letters equivalent.
-n	Print line and line number.
-q	Prints in quite mode, prints nothing.
-r	Recursively read all files in directories and in subdirectories found.
-v	Prints all the lines that do not match.
-V	Print Version.
-w	Match on whole word only.

You can also use Patterns for search operation.

.	Matches single character.
*	Wild Character.
^	Starting with.
$	Ending with.

EXAMPLE:

Lets assume that we have a file file1.txt and it has the following data.

```
hscripts has many valuable free scripts
It is the parent site of www.forums.hscripts.com
hscripts include free tutorials and free gif images
Purchase scripts from us
A webmaster/web master resource website
```

1. To print all lines containing hscripts :

   ```
   grep 'hscripts' file1.txt
   ```

   The output will be.

   ```
   hscripts has many valuable free scripts
   It is the parent site of www.forums.hscripts.com
   hscripts include free tutorials and free gif images
   ```

2. To print the count of line that matches hscripts.

   ```
   grep -c 'hscripts' file1.txt
   ```

   The output will be.

   ```
   3
   ```

3. To print the lines that starts as hscripts.

   ```
   grep '^hscripts' file1.txt
   ```

   The output will be.

   ```
   hscripts has many valuable free scripts
   hscripts include free tutorials and free gif images
   ```

4. To Search the files in HEC directory which has the string "include":

   ```
   grep -c 'include' HEC/*
   ```

   The above command will print the file name and count of line that matched the string "include"

   Sample output:

   ```
   HEC/admin.php:3

   HEC/auth.php:1
   HEC/calendar.php:3

   HEC/checklogin.php:0
   HEC/colors.php:0
   ```

1) **uniq:-** Locate repeated and nonrepeated lines

   When you concatenate or merge files, you will find duplicate entries. To avoid this problem uniq command help you.

   Since uniq requires a sorted file as input, the general procedure is to sort a file and pipe its output to uniq.

   Options:

   [-u] selects only lines that are not repeated.

   [-d] selects only one copy of the repeated lines.

   [-c] displays the frequency of occurrence of all lines, along with the lines.

   Example:

   3) cut –d”|” –f3 xyz.txt | sort | uniq –c
   4) uniq xyz.txt

Redirecting Standard Output: Every print and printf statement can be separately redirected with the > and | symbols.

The filename or command that follows these symbols is enclosed within double quotes. Default separated by white space. But –F option can be used if the columns are separated by another character.

Variable and Expressions: Expression comprise strings, numbers, variables and entities that are built by combining them with operators (x+5)*12 is an expression.
awk also allows the use of user defined variables but without declaring them.
awk variables do not use the $ either ion assignment or in evalution.
e.g: x=”5”
print x
A user define variable needs no initialization to zero or a null string.
awk provides no operator for concatenating strings. Strings are concatenated by simle placing them side by side.
e.g: x=”sum” ; y=”moon”
print x y or print x “ ‘ “ y

Comparison Operator: awk uses the || and && logical operators in the same sense as used by C.
Example: awk –F “|” ‘$3 == “director” || $3 == “chairman” ‘emp.txt

Regular Expression operators: ~ & !~ are the awk regular expression which offers these operators to match and negate a match.

Example:
$2 ~ /[cC]ho[wu]dh?ury/ || $2 ~ / sa[xk]s?ena / (Matches second field)
$2 ~ /[cC]ho[wu]dh?ury | sa[xk]s?ena / (same as above)
$3 !~ / director | chairman (Neither director nor chairman)

Number Comparison: awk can also handle numbers- both integer and floating type and make relation tests on them.
Example: print those people details whose basic pay greater than 7500.
awk –F”|” ‘$6 > 7500 {printf “%-20s %-12s %s \n”, $2, $3, $6}’ emp.txt

Number Processing: awk can perform computations on number using the arithmetic operators +, -, *, / and %.

Example: awk –F”|” ‘$3==”director” {printf “%-20s %-12s %d %d %d \n”, $2, $3, $6, $6*0.4, $6*0.15}’ emp.txt

Variables: While awk has certain built in variables like NR and $0, it also permits user to use variables of their choice.
Example: awk –F”|” ‘$3=”director” && $6 >6700 {count = count +1 printf “%3d %-20s %-12s %d \n”, count, $2, $3, $6}’ emp.txt

-f Option:
You should hold large awk programs in separate files and provide them with the .awk extension for identification.
$cat empawk.awk
$3==”director” {printf “%-20s %-12s %d %d %d \n”, $2, $3, $6, $6*0.4, $6*0.15}

To run this program you have to type:
awk -F”|” -f empawk.awk emp.txt
-f option storing awk program in file.

The BEGIN and END Sections: awk statements are usually applied to all lines selected by the address, and if there are no addresses, then they are applied to every line of input. But if you have to print before processing the first line, for example, a heading, then the BEGIN section can be used quite gainfully. Similarly the END section is used when you want to print after processing is over.
For example,
awkdemo.awk

```
BEGIN {
Printf “\t\t Employee Details\n\n”
}
$6 > 8500 {
count++; total+= $6
printf “%3d %-20s %-12s %d\n”, count,$2,$3,$6
}
END {
Printf “\n\t The average basic pay is %6d\n”, total/count
}
```

Built-in Variables

awk has several built in variables. They all are assigning automatically, though it is also possible for a user to reassign some of them. Following are some variables.

FS Variable: awk uses a contiguous string of space as the default field delimiter. FS redefines this field separator, which in the sample database happens to be the |. When used at all, it must occur in the BEGIN section so that the body of the program knows its value before it starts processing:
BEGIN { FS=”|”}
This is an alternative to the –F option.

OFS: When you used the print statement with comma-separated arguments, each argument was separated from the other by a space. This is awk’s default **O**utput **F**ield **S**eparator, and can be reassigned using the variable OFS in the BEGIN section:
BEGIN { OFS=”~” }
When you reassign this variable with a ~ (tilde), awk will use this character for delimiting the print arguments.

NF: NF comes in quite handy for cleaning up a database of lines that don’t contain the right number of fields.

FILENAME: FILENAME stores the name of the current file being processed. Like grep and sed, awk can also handle multiple filenames in the command line.

Control Flow
awk has practically all the features of a modern programming language. It has conditional structures and loops. They all execute a body of statements depending on the success or failure of the control command.

The **if statement** can be used when the && and || are found to be inadequate for certain tasks.

```
if (condition id true) {
        atatements
} else {
        statements
}
```

```
Example: if ($6 < 6000) {
        hra = 0.50*$6
        da = 0.25*$6
} else {
        hra = 0.40*$6
        da = 1000
}
```

for loop: This form also consists of three components same as C language.

```
for ( k=1 ; k<= 9 ; k++ )
```

while Loop: The while loop has a similar role to play; it repeatedly iterates the loop until the control command succeeds.

```
k = 0
while (k < (55 – length($0))/2) {
        printf "%s" , " "
        k++
}
```

Shell Script Programming

Shell variables are of two types- local and environment. PATH, HOME and SHELL are Environment variable. They are so called because they are available in the user's total environment- the sub-shells that run shell script, and mail commands and editors.

Environment Variables: Environment variables control the behavior of the system. They determine the environment in which you work. If they are not set properly, you may not be able to use some commands without a pathname.

PATH: The PATH variable instructs the shell about the route it should follow to locate any executable command.

HOME: When you log in, normally places you in a directory named after login name. This directory is called the home directory and is available in the variable HOME.

PS1: The shell has two prompts: PS1 and PS2. Normally, PS1 and PS2 in the Bourne shell are set to the character $ and >, respectively.
While the $ is the primary prompt string for no privileged users, the system administrator uses the # as the prompt.

SHELL: SHELL tells you the shell you are using.

TERM: TERM indicates the terminal type that is used. Every terminal has certain characteristics that are defined in a separate control file in the terminfo directory.

LOGNAME: this variable shows your username. When you wander around in the file system.

IFS: IFS contains a string of characters that are used as word separators in the command line. The string normally consists of the space, tab and newline characters.

MAIL: absolute pathname of user's mailbox file.

MAILCHECK: Mail checking interval for incoming mail.

ALIASES: Bash and Korn support the use of aliases that let you assign shot hand names to frequently used commands.
A user often uses the ls –l command, so if you don't have the ll command or alias on your system, you can create an alias:

```
alias ll='ls –l'
```

MISCELLANEOUS Features
Before moving on to the shell's initialization script, lets consider a few features available in Bash and Korn which you'll find quite useful.

- **Using set –o:** The set statement by default displays the variables in the current shell, but in bash and Korn, it can make several environment settings with the –o option.

  This option followed by a keyword, takes care of some of the common hazards faces by users, like overwriting files and accidental logging out.

- ✓ **File Overwriting (noclobber) :** The shell's > symbol overwrites(clobber) an existing file, and to prevent such accidental overwriting, you have to use the noclobber argument.
  set –o noclobber

  To override this protection feature, use the | after the >.
  example: head –n 5 emp.lst >| abc.txt

- ✓ **Accidental Logging Out(ignoreeof):** Users often inadvertently press [ctrl+d] with intent to terminate standard input, but end up logging out of the system. The ignoreeof keyword offers protection from accidental logging out:
  set –o ignoreeof

  A set option is turned off with set +o keyword.

- **Tilde Substitution:** The ~ acts as a shorthand representation of the home directory. When the ~ is followed by a login name, the shell understands the entire argument as the absolute pathname of the login directory.
  ex: cd ~sharma