DURGA SOFTWARE SOLUTIONS
UNIX

Operating System:- An operating system is the software that manages computers hardware and provides a convenient and safe environment for running programs. It acts as a interface between Programs and the Hardware resources. It is a collection of programs.

Features of OS:- Every operating system commonly have the Following features.

1)Process Management
2)Main memory Management
3)File management
4)Secondary Storage MANAGEMENT
5)Io system Management
6)Networking
7)Protecting System
8)Command Interpreter
9)Program Execution
10)File system Manipulation
11)Communication
12)I/O operations etc..

Introduction to UNIX:- UNIX is an CUI Operating System, it provides all types of computers such as Personal Computers, Micro Computers, Mini Computers, Super Computers etc.. UNIX become more popular because of many factors, including its portability, Machine Independent, Open system, and it can perform wide range of tasks also. It supports Networking, which has become Increasingly Important as the internet has blossomed.

History of UNIX:- Before development of UNIX Operating system at AT&T Bell labs, Software team lead by Ken Thomson, Dennis Ritchie and Rudd canaday worked on MULTICS (Multi Information Computing system) Project .The main theme of the project is to share data between the user at the same time. Actually this project developed for only two users. Based on the same concept in 1969, UNICS (Uniplexed Information Computing System) operating system was developed for 100's of users, and it was

developed in Assembly Language. In 1973, UNIX was developed in C language.

Features of UNIX:-

1)Multi-user

2)Multi Tasking

3)No constant Rebooting

4)Programming Facility

5)Security

6)Hierarchical Directory Structure.

7)Open system

8)Communication

9)Efficiency

10)Machine Independent

11)Help Facility

Features of UNIX:-

1) **Multiuser**:- if more than one user sharing the same system resources(cpu, hard disk, printer) at the same time known as MULTIUSER, windows also supports multi-user but windows servers supports limited no. of users.

2) **Multitasking**:- executing more than one application/job/task simultaneously known as Multitasking. Windows also support multi tasking, but by using windows we can execute limited no. of applications at a time. The main theme of Multi tasking is maximum utilization of CPU time.

3)**No Constant Rebooting**:- For UNIX there is no down time but for windows there must be down time, such that only we say that UNIX is highly available servers.

4)**open system**:- UNIX is more popular, one of the main reason for the popularity is it's an Open System. i.e. UNIX is an Open source code. Any user can modify UNIX source code according to their idea's and

Requirements. Such that only it has n no. of Flavors. UNIX is developed in C language.

5) **Programming Facility**:- UNIX shell is also a Programming language, it was designed for programmer not for end user. It has all the necessary Ingredients like control structures and loops and variables, that establish as a powerful Programming Language. These features are used to design Shell Scripts.

6) **Security**:- UNIX has given 3 levels of security.

a) System level security

b) File level security

c) Encryption Mechanism

System level Security is controlled by system administrator, where as File level security provided by the owner of the file, even the hacker hacks the file we can encrypt the file data such that the hacker cannot see the content.

7) **Hierarchical Directory Structure**:- UNIX uses a hierarchical file structure to store information. This structure has the maximum flexibility in grouping information and It allows for easy maintenance and efficient implementation.

8)**Portability:**- The main advantage of UNIX is it is an Portable Operating system, where as Windows is not Portable such that only it's become more popular and it for that only it is using in real time .it works 8088 processors to super computers.

9) **Efficiency**:- Other operating systems may only work on Advanced hardware or require large amount of memory or disk space. UNIX system

has made significant advances in processor and memory usage techniques, i.e. for UNIX no separate hardware required.

10) **Communication**:-Through UNIX we can establish communication between systems, the communication may be with in the network of a single computer or between two or more such computer networks. The user can easily exchange mail, data, programs Through such networks.

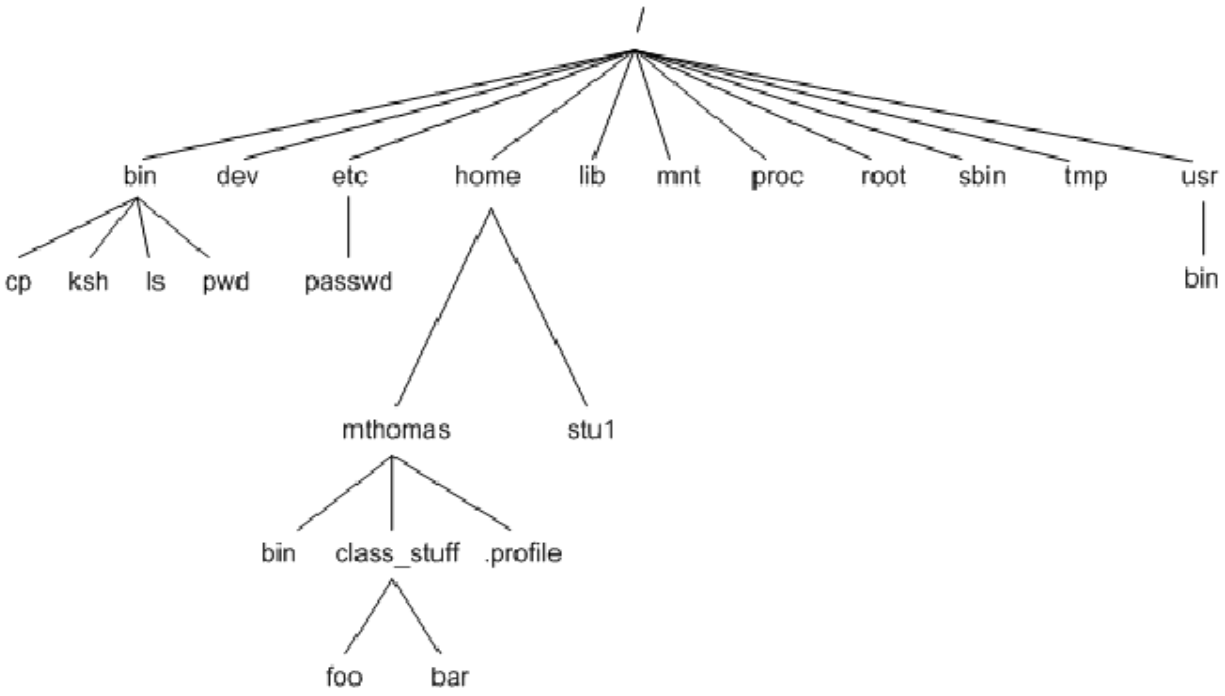
11)_**Documentation (or)Help Facility:-** UNIX provides manual pages for UNIX commands .such as help, info, man, apropos.

Differenfes between UNIX and Windows Operatins systems:

UNIX	WINDOWS
1) UNIX is multi-user o/s	1)windows also multi-user o/s
2) Multi tasking o/s.	2) Multi tasking o/s.
3) To boot the UNIX o/s, 2MB RAM is enough.	3) To boot the windows o/s, 12MB RAM is required.
4) UNIX is process based concept.	4) Window is process thread based Concept.
5) In UNIX, for every user request it creates new process.	5) For number of users request it Creates only one process.
6) Can run more than 1,00,000 Transactions per minute.	6) Maximum number of transactions In windows o/s is 80,000 per Minute.
7) In UNIX, any user process is killed	

it will not effect the other users.	7) If process kills it effects to all users.
8) There is no limit for number no. of users working with UNIX	8) it supports limited no. of users.
9) UNIX is an open system for that only it have number of flavors and no. of vendors.	9) Windows is a closed system.
10) UNIX is a portable o/s	10) No portability
11) UNIX provides the programming facility	11) No programming facility
12) it is CUI, such that it is not user friendly	12) Windows is GUI, such that it is user friendly.

UNIXFilesystem:-



Files:-

From the simplest perspective, everything visible to the user in a UNIX system can be represented as a "file" in the file system - including processes and network connections. Almost all of these items are represented as "files" each having at least one name, an owner, access rights, and other attributes.

The UNIX file system controls the way that information in files and directories is stored on disk and other forms of secondary storage. It controls which users can access what items and how. The file system is

therefore one of the most basic tools for enforcing UNIX security on your system. Information stored in the UNIX file system is arranged as a tree structure of directories and files. The tree is constructed from directories and subdirectories within a single directory, which is called the *root* .

In UNIX There are three different types of files

1. Regular files or Ordinary Files
2. Directory files
3. Special files or device files

Regular Files:- Ordinary files can contain text, data, or program information. An ordinary file cannot contain another file, or directory. An ordinary file can be thought of as a one-dimensional array of bytes.

Directory Files:- we described directories as containers that can hold files, and other directories.

Special Files:- Special files represent input/output (i/o) devices, like a tty (terminal), a disk drive, or a printer. Because UNIX treats such devices as files. Special files can be either character special files (input output) that deal with streams of characters, or block special files, that operate on larger blocks of data. (floppy, disk, CD-Rom, printer).

Directory: Directory is group of files. Directory is divided into two types:

- Root directory - Strictly speaking, there is only one root directory in your system, which is denoted by / (forward slash). It is root of your entire file system and can not be renamed or deleted.
- Sub directory - Directory under root (/) directory is subdirectory which can be created, renamed by the user.

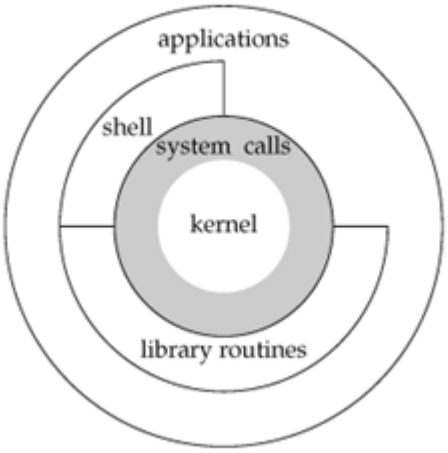
Directory	Description
/	Primary hierarchy root and root directory of the entire file system

	hierarchy.
/bin /bin/	Essential command binaries that need to be available in single user mode; for all users, e.g., cat, ls, cp..
/boot/	Home directory for the root user. files, *e.g.*, kernels, initrd; often a separate partition
/etc/	Host-specific system-wide configuration files
/home/	Users'_____home directories, containing saved files, personal settings, etc.; often a separate partition.
/lib/	Libraries essential for the binaries in /bin/ and /sbin/.
/media/	Mount points for removable media such as CD-ROMs
/mnt/	Temporarily mounted file systems.
/root/	Home directory for the root user.
/sbin/	Essential system binaries, *e.g.*, init, ip, mount.
/tmp/	Temporary files (see also /var/tmp). Often not preserved between system reboots.
/usr/	*Secondary hierarchy* for read-only user data; contains the majority of (multi-)user utilities and applications
/usr/bin/	Non-essential command binaries
/var/	Variable files—files whose content is expected to continually change during normal operation of the

	system—such as logs, spool files, and temporary e-mail files. Sometimes a separate partition.
/var/mail/	Users' mailboxes.
/var/log/	Log files. Various logs.
/dev/	This contains the special(device) files that include terminals, printers, andstorage devices.
/var/lib/	State information. Persistent data modified by programs as they run, *e.g.*, databases, packaging system metadata, etc.
/var/spool/	Spool for tasks waiting to be processed, *e.g.*, print queues and unread mail.
/var/spool/mail/	Deprecated location for users' mailboxes.
/var/adm	Contains system logging and accounting files
/var/news	Contains messages for common interest
/var/opt	It is the root of subtree containing add-on application pavkages
/var/tmp	Is a directory for temporary files
/var/uucp	Contains log and status files for the uucp system

Architecture of UNIX:-

Architecture of the UNIX operating system



shell:- UNIX provides an utility called shell, shell is a command line Interpreter. i.e. what ever the commands we give at the shell prompt it takes the commands and it checks whether executable file available or not for the given command, it available then it checks whether it is user executable command or not, if both are satisfied then shell convert this high level Instruction into Kernel Understandable format, such that we can say that it is an interface between user and kernel.

Kernel:- A kernel is the lowest level of easily replaceable software that interacts with the hardware in your computer. It is responsible for interfacing all of your applications that are running in "user mode" down to the physical hardware, and allowing processes, known as servers, to get information from each other using inter-process communication (IPC).The major functions of the kernel are to manage computer memory, to control access to the computer, to maintain file system, to handle interrupts(signals to terminate execution), to handle errors, to perform input and output services(which allow computers to interact with terminals, storage devices, and printers), and to allocate the resources of the computer(such as the cpu or input/output devices)among users. Programs interact with the kernel

through approximately 100 system calls. System calls tell the kernel to carry out various tasks for the program, such as opening a file, writing to a file, obtaining information about a file, executing a program, terminating a process, changing the priority of a process, and getting the time of day.

Basif Commands:-

$ →user prompt
# →Administrator prompt

1) $logname :-It Displays Current login user name

2) $pwd :- It displays current working directory path

3) $date :-it displays current date and time of the system

4) $cal :-It displays current month calendar

5) $date

6) $clear :- clears the screen

7) $cal 2010 :-it displays 2010 year calendar

8) $date +"%a" -->it gives abbreviated weekday name(thu)

9) $who :- It Displays List of users Connected to the server

o/p:- durgasoft1 ptt july 9 10:00

durgasoft1 ppt6 july 9 10:03

durgasoft →login name
ppt →terminal name
july 9 →date
10:00 →time

10) $who am i :- It displays current user Information

o/p: DurgaSoft tty Def24 9:30

11) $tty :- It displays Current Terminal Name

12)$clear or tput clear :- clears the screen

13)$exit or $logout :- to logout from the current user.

14) #init :-To change the system run levels

a)#init 0 :-to shutdown the system

b)#init 1:- to bring the system to single user mode

c)#init 2 :- To bring the system to multi-user mode with no resource shared

d)#init 3 :- to bring multi-user mode with source shared

e) #init 6 :- Halt and reboot the system to the default run level

15)$banner "UNIX" :- It prints message in Large Letters.

$ banner HELLO!

```
#     #                                  ###
#     #  ######  #       #        ####    ###
#     #  #       #       #       #    #   ###
#######  #####   #       #       #    #    #
#     #  #       #       #       #    #
#     #  #       #       #       #    #   ###
#     #  ######  ######  ######   ####    ###
```

CREATING NEW FILE:-

We can create files by using two commands such as cat and touch.

syn: 1) $cat > filename

---

ctrl +d

$

Eg:-2) $ cat filename (or) cat < filename (used to open a file)

3)$ cat file1 file2 file3 > file4

It create a new file by name file4 with contents of file1 followed by flie2 then followed by file3.If File4 already exists then it overrides the file4 data.

4) syn:- $ cat >> filename

---

ctrl + d

$

touch:- touch can be used to create empty files. by using touch we can create several empty files quickly, but by using touch we cannot insert the data to the files.

Creating a Directory:- In order to properly organize your work, you will want to be able to create sub directories. To create a subdirectory, you use the mkdir command. The command name is an abbreviation for "make directory." You supply the subdirectory name as an argument to the mkdir command, and a subdirectory will be made in the current working directory

Syn:- $mkdir directory name

Changing Directory:- The UNIX le system is arranged into a hierarchy of directories conveniently represented by a tree. As you organize your work, you will need to be able to navigate through the tree. To move to another directory, you use the cd command, short for "change directory." The cd command by itself with no arguments will place you in your home directory—regardless of which directory your current working directory is.

Eg:- durgasoft ~ $ cd abc

Durgasoft/abc $

Removing a directory:-

1)$rmdir directoryname → To remove a directory but directory must be empty.

2)$rm –r directory name:- it deletes recursively entire directory Structure.

PATHNAMES:- If you want to copy a file from one directory to another directory you must specify pathnames. Because of this hierarchy of directories, it is sometimes necessary to specify the path to a file or directory.

Example: /home/1/charles/history/week1.notes and history/week1.notes are both parameters to the week1.notes le

Note:- / is used to separate directory names.

Absolute Pathname:- A pathname that starts with / (the root). Since the path always starts at the root, it is correct regardless of what the current working directory is.

Relative Pathname: -A path that starts from the current working directory. Any pathname that does not start with / is taken to be as a relative pathname.

Example: If the current working directory is /home/1/Charles Then the absolute and relative pathnames to week1.notes respectively are

/home/1/charles/history/week1.note and history/week1.notes

COPYING FILES:- There are many times when you will want to copy a file or directory. For example, the cp command is used to copy a file. The command name cp is an abbreviation for "copy." When you copy a file, you simply create a distinct exact duplicate of the file. This is different from renaming a file.

$cp Sourcefile Destinationfile (here source file must be existing file)

Eg:- cp file1 file2

It copies file1 content to file2.

Rename a File:- The mv command is used to rename a le. The command name mv is an abbreviation for "move." When you rename a le, you simply change the name of the le. The contents of the le are not altered. This is a different process from copying a file.

Eg:- $mv file1 file2

comparison of files:-

cmp: It compares two files character by character, if two files are same then no output will be displayed, otherwise two file are not same.

Eg: $cmp abc xyz

$cmp xyz pqr

Differ xyz pqr : byte 4,line1

```
$cat > abc
Thanks
$cat > xyz
Thanks
$cat > pqr
 Thaks
```

diff:- It displays different lines between two files.

Eg:-$diff file1 file2

Dircmp;- dircmp command compares two directories. If i have two directories in my home directory named dirone and dirtwo and each has 5-10 files in it. Then dircmp dirone dirtwo will return this

Dec 9 16:06 1997 dirone only and dirtwo only Page 1

./fal.txt ./fourth.txt
./dohazaar.txt ./rmt.txt
./four.txt ./te.txt
./junk.txt ./third.txt
./test.txt

Word counts:-

wc:- it counts total no. of lines, words and characters in the given file

Syn:- $wc filename

Eg:- $wc test

10 40 400

2) $wc file1 file 2file3

5 15 60 file1

15 25 80 file2

25 35 90 file3

1)$wc –l filename → it counts no. of lines

2) $wc –w filename→it counts no. of words

3) $wc –c filename → it counts no. of characters

4) $wc –lw filename → it counts no. of lines and words

5)$wc –wc filename → it counts no. of words and characters

6)wc –lc filename → it counts no. of lines and characters.

Listing of files:- The ls command name is an abbreviation for "list." It is used to list your directories and sub directories and files. When ever you are log into your accounts, the system default places you in an area called users home directory.

1) $ ls →it displays Current directory all files and subdirectories in the ascending order based on ASCII values
2) $ls –a -→it list all files along with hidden files
3) $ls –r →it list all files in Reversing order
4) $ls –R →It list all files Recursively
5) $ls –t →it list all files based on date and time of creation
6) $ ls –l →it list all files in long listing format
7) $ls –l filename -→it displays given file information
8) $ls -x : It displays width wise
9) $ls | pg : It displays list of files and directories page's wise
10) $ls -x | pg : It displays list of files and directories pages wise & width wise.
11)$ls -F :It displays files , directories executable files, symbolic files.
12) Sls - i : it displays files and directories along with i-node number
13) Sls – rt : - It displays files and directories based on date & time of creation but in reverse order, i.e. last file to first file.

Wild card Characters:-

1. $ls a* : It displays all file -starting letter is a.

2. Sls b*k : It displays all files starting letter is b and ending letter is k.

3. Sls *k : It displays all files ending letter is k.

4. Sls a?k : It displays all three length filenames but starting letter is a.

5. Sls b??k : It displays all 4 length filenames but starting letter is b and ending letter is k.

6. Sls [aeiou]* : It displays all files but first character of the filename to be Listed must be any of the letters given within the square brackets and the

Remaining can be anything:

7. $ls[!aeiou]* : It displays all files whose first character is anything other than a vowel

8. Sis [k-v]*: It displays all files whose starting letter is between k and v.

Size:-size is a command line utility originally written for use with the UNIX-like operating systems. It processes one or more ELF files and its output are the dimensions (in bytes) of the text, data and un initialized sections, and their total.

Common use:

$ size <option> <filename> ...

Here follows some examples on Solaris and syntax may vary on different Operating Systems:

$ size /usr/ccs/bin/size
9066 + 888 + 356 = 10310

Linking Files:- A link is not a kind of file but instead it is a second name for a file. When a file had two links, it is not physically present at two places, but we can refer the file name either of the names. Suppose if you delete a file, (if it does not have any links) we cannot bring back that file (here no recycle bin option like windows).if that deleted file have any links you can get that file by using link file name.

The concept of having several link to a file offers another advantage, If one file is to be shared between several users. Instead of giving each user a separate copy of the same file. LAN creates Links of this file in each user's

directory. This avoids un necessary duplication of the same file contents in different directories.

UNIX os supports two types of Links.

1) Symbolic links (or) Soft links

2) Hard Links

A link can be established between two files or two directories. Here every updation, modification and deletion affected to vice versa.

Hard Links:- Here we can link between two files, it doesn't support link between directories.

Syn:- $ ln [option] soucefilename targetfilename

Eg:- cat > abc

this is abc file

Eg:- ln abc xyz

Eg:- cat abc →this is abc file

cat xyz →this is abc file.

Eg:- cat >> abc

I am appending abc

Eg:- cat abc → this is abc file

I am appending abc

cat xyz → this is abc file

I am appending abc

Unlink:- call unlink function to remove the specified file.

Syn:- unlink filename

Eg:- unlink abc (then only xyz available)

Eg:- cat xyz → this is abc file

I am appending abc

Soft links:- here also we can link between files and directories. Here also every updation affected vice versa.

Eg:- $ ln s-s xyz UNIX (xyz and UNIX are linked here)

Eg:- cat >> UNIX

I am appending to UNIX file

Eg:- cat UNIX → this is abc cile

I am appending abc

I am appending to UNIX file

E.g.:- unlink xyz (here both files are deleted i.e. xy and UNIX)

Note:- In hard link mother file deleted, child file not deleted but in the soft links mother file deleted then child file also deleted.

Linking Directories:-

syn:- $ln –s pqr mno

File permissions:- UNIX os supports three classes for file permissions, such as adding file permissions, deleting file permissions and assigning file permissions. We can add these file permissions in 2 modes.

1) Symbolic mode
2) Absolute mode

Symbolic mode:- In this mode by using the following arithmetical characters we can assign permissions.

+ → adding permission

- →removing permission

= →Assigning permission

When ever you create a file by default it have the following permissions.

-rw- r - - r- -

The first three letters of the permissions field refer to the owner's permissions. The second three to the members of the file's group , and last three to any other users.

r →read

w →write

x →execute

Eg:- $ls –l

-rwxr-xr-x

The first three characters owner of the file can read® it, write(w) it, and execute(x) it, the second group of three characters r-x indicates, that members of the group can read and execute the file but they cannot write it. The last three characters, r-x shows that all other users can read and execute the file but not write to it.

If you have read permissions for a file means you can view its contents, write permissions means you can alter its contents. Execute permissions means that you can run the file as a program.

Permissions for Directories:-

_For directories read permissions allows user to list the contents of the directory. Write permissions allows users to create or remove files or directories inside that directory and execute permissions allows users to change this directory using the cd command.

0 →no permissions
1 →execute
2 →write
3 →write and execute
4 →read
5 →read and execute
6 →read and write
7 →read and write and execute

Chmod :- (changing file access permissions)

Syn :- chmod [who] [+/-/=] [permissions] filename

Note:- u represents user or owner

g for group

o for others

eg:- write a command to add execute permissions to owner of the file

$ chmod u+x filename

Eg:- write a command add execute permissions to owner and add all permissions to group and others.

$chmod u+x,go+rw filename

Eg:- write a command remove read permissions form owner and others.

$chmod uo-x filename

Absolute mode:- In this mode permission presents means one,permission absence means zero.

Eg:- write a command no permissions to owner of the file,

read and write permissions to the group and others.

$chmod 066 filename

Q.write a query to add read and execute permissions to the

Owner of the file,write permissions to group and read permissions to others.

$chmod 524 filename

Q. write a command remove all permissions from all users.

$chmod 000 filename.

Using umask to set permissions :- the chmod command allows you to alter permissions on file by file basis,where as the umask command allows you toodo this automatically when ever you create a file or directory. Every file has default umask ,setting set up either by the system Administrator or their .profile.

The main purpose of umask is changing default file permissions.

According to umask:-
Full permissions for regular files: 666
Full permissions for directory :777

Eg:- $umask 034

Note:- once umask changed then what ever the files created by user having permissions like below.

$cat jaya

-rw- -wx – w –

Changing group of the file:- chgrp command is used to change the group of a file or directory. You must own the file or be a superuser. chgrp [options] newgroup files is syntax of chgrp. Options:

- -h will change the group on symbolif links.

- -R refursively descend through directory Changing group of all files and subdirectories.

The chgrp Command takes two arguments, the name of the new group and name of the file.

Eg 1: S chgrp durgasoft gmng

In above example, chgrp command changes gmng file into durgasoft group.

Note: only the owner of a file (or the super user) can change the group to which this file belongs.

Changing owner of afile or directory:- chown command to change ownership of a file or directory to one or more users. Syntaxis

chown *options newowner files*

Options

- -h will change the owner on symbolic links.
- -R will recursively descend through the directory, including subdirectories and symbolic links.

Eg 1:- Schown durgasoft gmng

In above example, durgasoft is the new owner of gmng file.

Note:- Only the owner of a file (or the superuser) can use chown to change its ownership.

Zip formatted files:- UNIXoperating system supports Creating zip formatted files.

The following threes set of commands supports different flavours of UNIX and linux.

1)**compresss**:- using this command we can add zip format to a specified file.

Syn:- compress [options] [filename]

Eg:- $compress test

Uncompress:- to get the compresses back to its original state, we can use the uncompress utility as shown below.

Eg:- $uncompress test.z (here the test.z is deleted and its original test file is recreated back in its original form).

Zcat:- once the file has been compressed we cannot view by using normal cat command, UNIX provides a utility called zcat.

Eg:-$zcat test.z

2) Pack:- it is also able to compact a file.

Eg:- $pack filename

Unpack:- to get back the original file from packed file there is utility called unpack.

Eg:-$unpack filename

Pcat:- to view contents of a packed file.

Eg:- $pcat filename

3) gzip:- it add zip format to specified file or files.

Syn:- gzip [options] [filename] (default extension is .gz)

gunzip: using this command we are able to remove the specified zip format from the specified file.

Eg:- $gunzip filename

Zcat:- using this command we can see the contents of zip formatted files.

$zcat [options] [filenames]

Split:-

split is a UNIX utility most commonly used to split a file into two or more smaller files.

Usage

The command-syntax is:

split [OPTION] [INPUT [PREFIX]]

The default behavior of split is to generate output files of a fixed size, default 1000 lines. The files are named by appending aa, ab, ac, etc. to output filename. If output filename is not given, the default filename of x is used, for example, xaa, xab, etc. To split filename to parts each 50 MB named partaa, partab, partac,....

Eg:- split -b50M filename part

To join the files back together again use the cat command

cat xaa xab xac > filename or
cat xa[a-c] > filename

Filter Commands:-

grep Command:

The name "grep" means something like "general regular expression parser", but you can think of the grep command as a "search" command for UNIX and Linux systems: it's used to search for text strings and more-complicated regular expressions within one or more files. This first grep command example searches for all occurrences of the text string 'fred' within the "/etc/passwd" file. It will find and print (on the screen) all of the lines in this file that contain the text string fred, including lines that contain usernames like "fred" - and also "alfred".

Eg1:- grep 'fred' /etc/passwd

Eg2 : $ grep durgasoft durgafile

o/p:- Choose your career in durgasoft solutions

Eg2: $ grep "durgasoft solutions" testfile

To search more than word the string should be in double quotes

Using grep for Queries:-

grep is often used to search for information instructured files or simple databases. An example of such a file is "students"?

$ cat students

101 jaya 99 kurnool

102 durga 98 hyd

103 pavan 100 guntur

104 ramu 96 prakasham

You can use grep to find all students in the file that contain the word :

Eg:-$grep jaya students

101 jaya 99 kurnool

Using grep to locate files:-

We can use grep to search in multiple files also. if we search in more than one files to search, then the output will come along with the file name .it includes the name of the file(before each line of output). For example, the following command searches for lines containing the string "jaya" in all of the files in the current directory:

Sgrep jaya *

students : 101 jaya 99 kurnool

students: 103 neeraja 88 durga solutions

sample : jaya is a faculty of durgasoft

durgasoft : jaya is well known to me

Searching for patterns using regular expressions :-

There will be many times when you want to search a le for a pattern. It would be convenient if you could do this without using a text editor. If you could perform such a search from the operating system, you would not have to go through the usual steps of opening the file with a text editor, using the editor's search facility, and then closing the file. UNIX provides the grep command for searching for a pattern. In fact, grep allows you to search an entire directory or even an entire system for a pattern. The grep command is a powerful search mechanism that provides a convenient notation, allowing you to specify complex patterns to search for. The word grep is an acronym for "global regular expression print".

The rules and symbols used for forming regular expression for ed and vi can also be used with grep to search for patterns.

```
$grep -l 'delay' /fode/*.f
```

The above command searches for those files that end with a '.f' (within

the /fode directory) and in which the text 'delay' is present. It only returns the names of these files and not the lines where it found the string.

Using grep with pipes

$ls -l | grep rwxrwxrwx

As you must be knowing ls -l displays the directory listing for any directory. The grep rwxrwxrwx part of the command extracts only those lines which display the files having their read,write,execute permissions set for user, group and others also. Thus instead of getting a listing of all the files in the directory, you would only see those files that have their r,w,x permissions set for all everybody.

$ grep '^#' /home/durgasoft/script1

The above command would display those lines (from the file /home/durgasoft/script1) that begin with a '#'. The term '^#' means that # should be present as the first character on a line.

fgrep Command :- fgrep searches files for one or more pattern arguments. It does not use regular expressions; instead, it does direft string comparison to find matching lines of text in the input.

The fgrep command prints all lines matching a particular pattern.

Eg:- $fgrep "jaya
>ramu" students

o/p:- 101 jaya 99 kurnool
104 ramu 96 prakasham

By the way, you should notice here that when you give fgrep multiple search targets, each one must be on a separate line.Note that you have to put the search string in quotation marks when it contains several targets.

The fgrep command does not accept regular expressions. The target must be text strings.

egrep:- The egrep command is the most powerful member of the grep command .You can use it like fgrep to search for multiple targets. Like grep , it allows you to use regular expressions to specify targets, but it provides a More powerful set of regular expressions than grep.

The egrep command accepts all of'the basic legular expressions recognized by grep. You can tell egrep to search for several targets in two ways: by putting them on separate lines as in fgrep or by separating them with the vertical bar or pipe symbol (|).

For example, the following command uses the pipe symbol to tell egrep to Search for entries for dkd,kurnool ahd,hyd in the file students:

eg:- $egrep "(dkd|Kurnool|hyd)" students

101 jaya 99 kurnool

102 durga 98 hyd

Sort:-

The sort command can be used for sorting the contents of a file. Apart form Sorting file, the sort can merge multiple sorted files and store the result in the specified output file. While sorting the sort command bases it comparisons on the first character in each line in the file. If the first character of two lines is same then the second character in each line is compared and so on i.e. The sorting done according to the ASCII sequence.

Syn:- S sort filename

E.g. 1: S sort testfile

This would sort the contents of test_file and display the sorted output on the

screen. If we want we can sort the contents of several files at one shot as in:

Ssort testl test2 test3

This would sort the three files and displays the sorted ouput.

Instead of displaying the sorted output on the screen we can store it in a file by saying,

Ssort -oresult test l test2 test3

The above command sorts the three files test1,test2,test3and saves the result in a file called result. If the files have already been sorted and we just want to merge them we can use.

Options for sort:-

- -b ignores leading spaces and tabs.
- -f checks whether files are already sorted.
- -d ignores punctuation.
- -i ignores non-printing characters.
- -n sorts in arithmetic order.
- -o*file* put output in a file.
- +m[-m] skips n fields before sorting, and sort upto field position m.
- -r reverse the order of sort.
- -u identical lines in input file apear only one time in output.

Cut Command:

Like sort, cut is also a filter. It picks up a required number of character or fields from the specified file. Say you have a large database of students information. Suppose if you want to view only a few selected fields, such as student id and marks, then cut is the answer.

$cut-f 1,3 students

it displays the following output. In students file the first field is student id and third field is marks.

```
o/p:-  101   99
       102   98
       103   100
       104   96
```

Eg:- $cut –f 1-3 students

```
o/p:-  101    jaya     99   kurnool
       102    durga    98    hyd
       103    pavan    100   guntur
       104    ramu     96    prakasham
```

Eg:- $cut –f 1-8 students

```
o/p:-  101 jaya
       102 durg
       103 pava
       104 ramu
```

Input/Output/Error Redirection:-

Whenever any program is executed (i.e. when the user types a command) the program has 3 important files to work with. They are standard input, standard output, and standard error. These are 3 files that are always open when a program runs.

Output Redirection:-

The most common use of Redirection is to redirect the output (that normally goes to the terminal) from a command to a file instead. This is known as Output Redirection. This is generally used when you get a lot of output when you execute your program. Often you see that screens scroll past very

rapidly. You could get all the output in a file and then even transfer that file elsewhere or mail it to someone.

The way to redirect the output is by using the ' > ' operator in shell command you enter. This is shown below. The ' > ' symbol is known as the output redirection operator. Any command that outputs its results to the screen can have its output sent to a file.

Input Redirection:-

Input Redirection is not as popular as Output Redirection. Since most of the times you would expect the input to be typed at the keyboard. But when it is used effeftively , Input Redirection can be of great use. The general use of Input Redirection is when you have some kind of file, which you have ready and now you would like to use some command on that file.

You can use Input Redirection by typing the ' < ' operator. An excellent example of Input Redirection has been shown below.

Error Redirection:-

This is a very popular feature that many UNIX users are happy to learn. In case you have worked with UNIX for some time, you must have realised that for a lot of commands you type you get a lot of error messages. And you are not really bothered about those error messages. For example whenever I perform a search for a file, I always get a lot of *permission denied* error messages. There may be ways to fix those things. But the simplest way is to redirect the error messages elsewhere so that it doesn't bother me. In my case I know that errors I get while searching for files would be of no use to me. Here is a way to redirect the error messages .

File Descriptor	Descriptor points to

0	Standard Input (Generally Keyboard)
1	Standard output (Generally Display/Screen)
2	Standard Error Ouput (Generally Display/Screen)

The symbol >> adds output from a command to the end of a file without deleting the information already in the file.

E.g. Scat testl > test2

On executing this command , the symbol > sends testl file contents into test2

file instead of sending to output screen.

E.g. Scat testl » test2

on executing this command , the symbol » sends file1 contents to file2 then it allows the user to data into test2 file.

Piping:- The purpose of this pipes is to introduce you to the way that you can construct powerful UNIX command lines by combining UNIX commands. UNIX commands alone are powerful, but when you combine them together, you can accomplish complex tasks with ease. The way you combine UNIX commands is through using pipes and filters.

The symbol | is the UNIX pipe symbol that is used on the command line. What it means is that the standard output of the command to the left of the pipe gets sent as standard input of the command to the right of the pipe. Note that this functions a lot like the > symbol used to redirect the standard output of a command to a *file*. However, the pipe is different because it is used to pass the output of a command to *another command*, not a file.

E.g. Sls | wc-l

On executing this output as a input to we takes ls command output and redirects output as input to wc –l commands and and display no of lines on output screen.

E.g. Sls | wc -l > file1

On executing this command, the pipe takes la command output and redirects the output as a input to wc -l commands and counts no of lines and sends output to file1(filename).

Viewing Long Files:-

more command:- The more command provides a convenient way to view the contents of a file one screenful at a time. For example, entering the command

Syn:- more filename

displays one screenful of content of the given file name. Hitting the Spacebar brings u p the next screenful of text, and typing q "quits" the more command and brings you back to the UNIX prompt.

pg:- Pg displays a text file on a FRT one screenful at once. After each Page, a prompt is displayed. The user may then either press the newline Key to view the next page or one of the keys described below.

less COMMAND:- less command is used to display text in the terminal screen. It just prints the text in the given file, you cannot edit or manipulate the text here. To display the file from the specified line, enter the line number followed by colon(:). It allows Forward and backward movement in the file.

SYNTAX:

The Syntaxis

less [options] filename

OPTIONS:

-f Clear screen before displaying.
+n Starts up the file from the given number.

Communication Commands:-

Write command let u have a two way communication with any person who is currently logged in. write command uses login name of the recipient as argument, and text of the message from standard Input.

$ write nit

hey how r u gmng.

We have UNIX class today.

Then Ctrl d

There are two prerequisites for a smooth write operation:

(a) The recipient must be logged in, else an error message is inevitable.

(b) The recipient must have given permission for message to reach his or her terminal. This is done by saying at the $ prompt

$ mesg -y

if you are expecting nothing of consequence and do not wish to be disturbed by any other you can deny permission to your terminal by saying

$mesg –n

mail command:-

Using mail you can send files and receive mails from your Team members. You can even send and receive mail from people outside your organization. If you and they use network computers, By using mail command. mail can be sent to users who have logged in currently or even to users who haven't logged in currently.

Sending mail:

$ mail durgasoft

subject: from durgasoft solutions

Durgasoft offering Android course also.

ctrl+d

→To send mail to multiple persons

$ mail tecnol tecno2 tecno3

subject: from durgasoft solutions

Durgasoft offering Android course also.

It contains following options

+ →to view next message

- →to view previous message

&2 →to see second mail

&3 → to see third mail

&1-3 →it displays 1st and 2nd and 3rd mails

q →quit

d →it is used to delete current used mail

d 5 →to delete 5th mail

d 1-5 →delete mails from 1-5

& $ →displays last message

$ = →displays current message number

. →displays current message

u →undo last deleted message

Receiving mail:-

$ mail (press enter)

"/usr/spool/mail/durga" 2messages 1new 1unread

N 2 durgasoft mon dec 24 12:00 8/243 java

U 1 India sat apr 21 11:09 9/146 UNIX

Options:-

& 1 →shows first message

& d1→deletes 1st message

&d 1-4 →deletes 1-4 messages

& = →displays current message no

& . →displays currently worked message

mail –f:- it will take to the secondary inbox.

Finding Files:-

Introduction:-

The find command helps-you locate files in the file system. With the find Command, you can search through any part of the file system, looking for all files with a particular name. An example of a common problem that find can help solve is locating a file that you have misplaced. For example, if you want to find a file called durgasoft file but you can't remember where you put it, you can use find to search for it through all or part of your directory system. The find command searches through the contents of one or more directories, including all of their sub directories. You have to tell find in which directory to start its search. To search through all your directories, for example, tell find to start in your login directory.

Let us see examples on find command.

Eg1:-finding files by using name.

1. **Find Files Using Name**:-

This is a basic usage of the find command. This example finds all files with name MyCProgram.c in the current directory and all its sub-directories.

```
# find -name "MyCProgram.c"
./backup/MyCProgram.c
./MyCProgram.c
```

2. **Find Files Using Name and Ignoring Case**:-

This is a basic usage of the find command. This example finds all files with name MyCProgram.c (ignoring the case) in the current directory and all its sub-directories.

```
# find -iname "MyCProgram.c"
./mycprogram.c
./backup/mycprogram.c
./backup/MyCProgram.c
./MyCProgram.c
```

3) Limit Search to Specific Directory Level Using mindepth and maxdepth

Find the passwd file under all sub-directories starting from root directory.

```
# find / -name passwd
./usr/share/doc/nss_ldap-253/pam.d/passwd
./usr/bin/passwd
./etc/pam.d/passwd
./etc/passwd
```

4) Finding files which has been modified less than one day in UNIX:

find. -mtime -1

5) List all the files and directories in the box which holds the 777 permission in UNIX?

find . -perm 777 –print

6) Find all empty files (zero byte file) in your home directory and its subdirectory

Most files of the following command output will be lock-files and place holders created by other applications.

```
# find ~ -empty
```

Vacation:-

vacation command is used when you are out of office. It returns a mail message to sender announcing that you are on vacation. to disable this feature,type mail-F " " .

vacation *options :*

- -d will append the date to the logfile.
- -F user will forward mail to user when unable to send mail to mailfile.

ftp command (protocol):- The FTP (File Transfer Protofol) utility program is commonly used for copying files to and from other computers. These computers may be at the same site or at different sites thousands of miles apart. FTP is a general protofol that works on UNIX systems as well as a variety of other (non-UNIX) systems.

Syntax:-

ftp options hostname

ShutdownCOMMAND:-

Shutdown command can only be exefuted by root. To gracefully bring down a system, shutdown command is used.

alias:-

syn:- alias newname command

Allows you to create an easy-to-remember substitution for a complex command or a pathname. The alias command is supported by ksh and bash, but not sh.

Eg:- The example causes ls -l to be executed when the command ll is entered:

% alias ll='ls -l'

du:- the "du" command in UNIX tells you how much space a directory or set of directories and the files inside are taking up. Let's say you want to know how much space you are taking up in your home directory, and all the subdirectories you've made off your home directory. You would simply log into your accounts, and as your first command, just type 'du'. The output wll look something like this:

yossman.net# du

906 ./mail
8 ./.nfftp
yossman.net#

head:- it displays top lines of the file.

Syn:- head [options] [filename]

Options:-

-c →bytes,it prints first 'n' bytes.

-n →lines,it prints first 'n' lines.

Eg:- $head -4 big → displays 4 files.

tail:-it displays last lines of files.

-c →bytes,it prints last 'n' bytes.

-n →lines,it prints last 'n' lines.

Eg:- $tail -3 big →it prints last 3 lines.

lp(line printer):- print files.

Syn:- lp [options] [filenames]

It has the following options

-d →destination,print files to the named printer

-n →no. ofcopies

-q →priority,sets the job priority

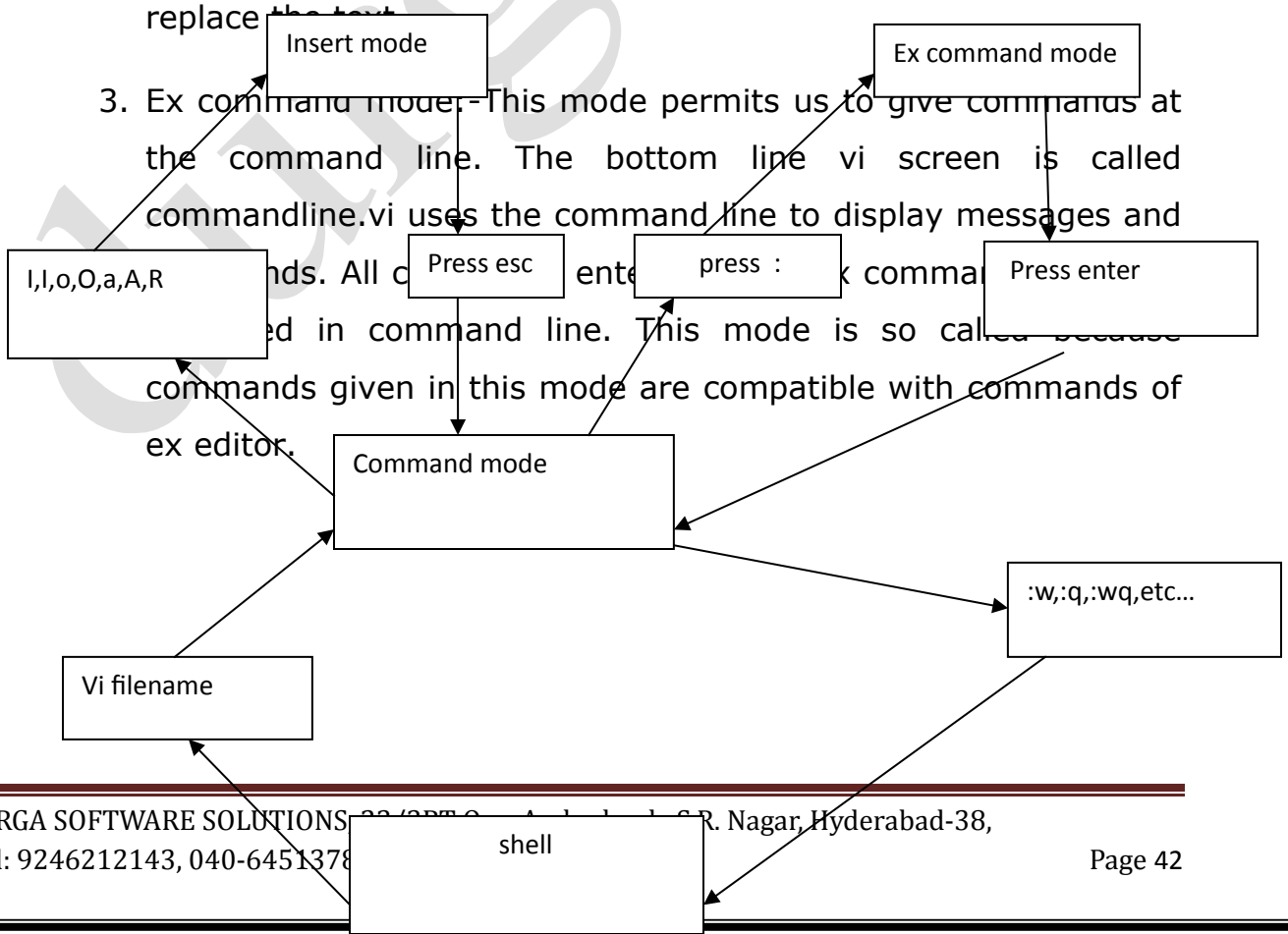
Editor:- A file which supports only asfii values called as editor.No editor supports other than Ascii values i.e. numerics,Alphabet symbol.

ed:- this is known as line editor,means a user can make any changes at line level only.we cannot move cursor to the character position.

ex:- it is also a line editor for UNIX systems.The original ex was an advanced version of the Standard UNIX editor ed.

Vi edior:- The default editor that comes with the UNIX operating system is called vi (visual editor). [Alternate editors for UNIX environments include pico and emacs, a product of GNU.The UNIX vi editor is a full screen editor and has three modes of operation:

1. command mode :- command mode treats every key as a command.
2. Insert mode:- In this mode we can enter the text, modify, replace the text
3. Ex command mode:-This mode permits us to give commands at the command line. The bottom line vi screen is called commandline.vi uses the command line to display messages and nds. All c ente k comma ed in command line. This mode is so called because commands given in this mode are compatible with commands of ex editor.



Moving the Cursor *deleting text:-*

Keys pressed	Effect
h	left one character
l or <Space>	right one character
k	up one line
j or <Enter>	down one line
b	left one word
w	right one word
(	start of sentence
)	end of sentence
{	start of paragraph
}	end of paragraph
1G	top of file
nG	line ***n***
G	end of file
<Ctrl>W	first character of insertion
<Ctrl>U	up ½ screen
<Ctrl>D	down ½ screen
<Ctrl>B	up one screen

Keys pressed	Text deleted
x	character under cursor
12x	12 characters
X	character to left of cursor
dw	word
3dw	three words
d0	to beginning of line
d$	to end of line
dd	current line
5dd	five lines
d{	to beginning of paragraph
d}	to end of paragraph
:1,. d	to beginning of file
:.,$ d	to end of file
:1,$ d	whole file

Screen manipulation:-

^f	*move forward one screen*
^b	*move backward one screen*
^d	*move down (forward) one half screen*
^u	*move up (back) one half screen*
^l	*redraws the screen*
^r	*redraws the screen, removing deleted lines*

Command	Function
w	Moves cursor to the right,to the first character of the next word
B	Moves cursor back to the first character of the previous word
E	Moves the cursor to the end of the current word

Inserting or Adding Text:-

The following commands allow you to insert and add text. Each of these commands puts the vi editor into insert mode; thus, the <Esc> key must be pressed to terminate the entry of text and to put the vi editor back into command mode.

*	i	*insert text before cursor, until <Esc> hit*
	I	*insert text at beginning of current line, until <Esc> hit*
*	a	*append text after cursor, until <Esc> hit*
	A	*append text to end of current line, until <Esc> hit*
*	o	*open and put text in a new line below current line, until <Esc> hit*
*	O	*open and put text in a new line above current line, until <Esc> hit*

Changing Text

The following commands allow you to modify text.

Keys pressed	Text changed or replaced
cw	word
3cw	three words
cc	current line
5cc	five lines
r	current character only
R	current character and those to its right
s	current character
S	current line
~	switch between lowercase and uppercase

Cutting and pasting:-

Positioning by character:-

Command	Function
w	Yanks word from cursor position
Yy	Yanks line from cursor position
y$	Yanks line from cursor position of line
y0	Yanks line from cursor position to beginning of line
Dgg	It deletes current position to begining of the file
dG	It deletes current position to end of the file
U	Undo
Zz	Save and quit
P	Paste below the cursor
P(capital)	Paste above the cursor

Command	function
0	Moves cursor to the beginning of the current line
$	Moves cursor to the end of the current line
H	Moves cursor to the one character left
l	Moves cursor to the one character right

Ex command mode commands:-

Positioning by lines:-

Command	Functionality
:w	Save with out quit
:w filename	Save with given filename
:q!	Quit with out save
:wq	Save and quit
:n	Moves to specified line
:$	Moves to last line
:set nu	To set line numbers
:set nonu	to remove line numbers
:!<cmd>	To execute UNIX commands
:/UNIX/	It searches for the UNIX word from top to bottom
:?UNIX?	it searches for the word UNIX form bottom to top
:1,$ s/UNIX/linux/gi	It replaces UNIX with linix in the file globally (i-ignore case)
:1,$ s/^/UNIX	It adda UNIX at beginning of each file
:1,$ s/$/UNIX	It adds UNIX at end of each linc
:3d	It deletes 3rd line
:4,7d	it deletes 4th line to 7th line

Command	function
+	Moves cursor down to the beginning of next line
-	Moves cursor up to the beginning of previous line
J	Moves cursor down one line from its cur ent position,in the same column
K	Moves cursor up one line from its present porition,in the same column

SHELL SCRIPTING:-

Shell programming is integral part of UNIX operating systems. Shell is command line user interface to UNIX operating system, User have an option of picking an interface on UNIX such as ksh, csh, or default sh., these are called shells (interface). Shell programming is used to automate many tasks. Shell programming is not a programming language in the truest sense of

word since it is not compiled but rather an interpreted language. UNIX was written in C language and thus c language is integral part of UNIX and available on all versions. Shells, like ksh and csh are popular shells on UNIX although there are 5 or 6 different shells available.

SHELL:- Shell is a user program or it's environment provided for user interaction. Shell is an command language interpreter that executes commands read from the standard input device (keyboard) or from a file. Shell is not part of system kernel, but uses the system kernel to execute programs, create files etc.

UNIX provides an utility called shell , shell is a command line Interpreter. i.e. whatever the commands we give at the shell prompt it takes the commands and it checks whether executable file available or not for the given command, it available then it checks whether it is user executable command or not, if both are satisfied then shell convert this high level Instruction into Kernel Understandable format, such that we can say that it is an interface between user and kernel.

Shell script file means if any file contain set of commands with in it. If any file contain set of commands then that file can be an executable file. Shell scripts can be used to execute the set of commands at a time, Executing commands separately will consume much more time. Using th e shell script we can reduce this time at a greater extent.

Uses of shell scripting:-

1. Automating your daily tasks, for example, you a may want to back up all Your programs at the end of day. This can be done using a shell script.
2. Automating repetitive tasks, for example, the repetitive task of compiling a c program, linking it with some libraries and executing the executable

Code can be assigned to a shell scripting.

3. Customizing your work environments, for example, every time you log in if you want to see the current date, a welcome message and the list of 3 users who have logged in you can write a shell script for the same.

4. Performing same operation on many files. example, you may want to Replace a string printf with a myprintf in all the c programs present in a directory.

5. we can create our own commands.

6. It saves lot of work.

Types of shells:-

Bourne shell:-It was developed by Stephen Bourne. It have good features for I/O control, and often used for writing scripts, and it is not well suited for interactive users. Other shells based on Bourne may be suited for interactive users. Default prompt is $.we can execute file by using sh at shell prompt.

C shell:- it was designed by Bill Joy at the university of California at Berkley, the c shell was named because much of its syntax parallel to that of C language. I/O more awkward than Bourne shell. Nice for interactive use. Shell prompt is %.

Korn shell:- It was developed by Durgasoft korn.It was developed based on the Bourne shell. To execute a script file,we use ksh.The default prompt for Korn shell is $.

Almost all UNIX implementations offer the Bourne shell as part of their standard configuration, it is smaller than the other two shells and therefore more efficient for most shell processing, it lacks features offered by the c and the korn shell. All shell programs written for the Bourne shell are likely to work with the korn shell, the reverse may not be true, this is so since the facilities like arrays, command aliasing and history mechanism available in the korn shell are not supported by the bourne shell.

Shell variable:-variable is an identifier, whose value is Changing during execution of the program.

In UNIX we have 2 types of variables.

1.System defined variables or environmental variables

2.User Defined Vriables

System defined variables:-

HOME=/home/durgasoft

would set the home directory to /home/durgasoft. This is perfect in case your login name is durgasoft and you have been given a directory named /home/durgasoft . In fase you don't want this to be your home directory but some other one you could indicate so by typing the new directory name. The HOME directory is always the directory that you are put in when you login.

There are many advantages of using the HOME variable. You can always reach your home directory by only typing ' cd ' at the prompt, irrespective of which directory you are presently within. This would immediately transfer you to your HOME directory. Besides in case you write scripts that have $HOME present in them to refer to the current HOME directory, these scripts can be used by other users as well since $HOME in their case would refer to their home directories.

PATH=/usr:/bin/:usr/local/bin:.

This is a very important environment variable. This sets the path that the shell would be looking at when it has to execute any program. It would search in all the directories that are present in the above line. Remember

that entries are separated by a ' : '. You can add any number of directories to this list. The above 3 directories entered is just an example.

PATH=/newdirectory

This would replace the current PATH value with the new value only. What you would want is to append the new directory to the existing PATH value. For that to happen you should type

PS1=durgasoft

PS1 is the shell prompt. It defines what you want your shell prompt to look like. By default it looks like a ' $ ' in bash shell. The above case would replace the default ' $ ' with a new 'durgasoft'. Hence an ls command would looksomethinglike

durgasoft $ ls

All your commands would now be typed at a ' boss ' prompt instead of a ' $ ' prompt.

PS2= PS2The secondary prompt (Input prompt), The default value is "'> '"

SHELL=/bin/bash

This tells where the program that represents your shell is to be counted. In case you typed /bin/ksh in the above, then your bash shell would be replaced with the ksh shell (korn shell). So in case you are not happy with the bash shell, you could replace the bash with some other shell.

LOGNAME=durgasoft

The LOGNAME is automatically set for you as the same as your login name. This variable is used in case you want to use your own login name in any script. This is the simplest way of getting your login name from within a script. Thus in case you use $LOGNAME in any script the script would work for all users since the LOGNAME always holds the name of the current user.

TERM→ Your login terminal type. echo $TERM
SHELL→ Set path to login shell.

User defined variables:-

Rules for Creating user defined variables:-

1. The first character of a variable name should be alphabet or underscore
2. No commas or blanks are allowed within a variable name
3. Variables names should be of any reasonable length
4. Variable names are case sensitive, that is name,Name,nAme,name are all Different variable names.
5. Variable name shouldn't be a reserve word

echo:- Displaying Line of text on console.

```
$echo "HelloWorld"
Hello World
```

```
Eg:- 1)$a=100
     $echo $a
        o/p:- 100
     2)k=`cat stud`
       echo $k
```

o/p:- it displays stud file details.

Shell Keywords:-

echo	case
if	wait
read	esac
else	eval
Set	break
fl	continue
unset	exit
while	umask
readonly	return
do	shift
	done
	export

Eg:-2) echo "today's calendar is:" `cal`

3)echo "display current login users" : `who`

Constant variable:-we can declare constant variables by using "readonly" keyword

Syn:- $readonly a

Whenever you made a variable as read-only variable then we cannot change that value. If we want to delete a variable then we have to use "unset" keyword.

Syn:- $unset a

null variable:-while declaring variables if we not give values for that variables those type of variables are known null variables.

Eg:- $n= or $n='' or $n=""

SHELL SCRIPTING PROGRAMS:-

1) Write a shell script to display the following content.

I am learning

I am learning first shell script program

o/p:- vi test1

```
echo I am learning
echo " I am learning first shell script program"
:wq(save and quit)
```

Execution:-At the shell prompt we can execute by using following command in Bourne shell.

```
$ sh test1
Or $chmod 755 first
$./test1
```

2) write a pgm for accepting two numbers and display the numbers.

```
$vi test2
   echo "enter number:\c"
  read a
echo "enter another number:\c"
  read b
echo "a value is:$a"
echo "b value is:$b"
:wq(save and quit)
$sh test2
```

Read is the special keyword to read a value.

3)write a shell script to accept two numbers and perform arithmetic operations.

```
Vi test3
echo -e "enter number:\c"
read a
echo -e "enter another number:\c"
read b
f=`expr $a + $b `
echo " sumis:$f"
f=`expr $a - $b `
echo " sumis:$f"
f=`expr $a \* $b `
echo " sumis:$f"
f=`expr $a / $b `
echo " sumis:$f"
f=`expr $a % $b `
echo " sumis:$f"
:wq(save and quit)
$sh test3
```

4)write a shell script to accept a number and find square of that number.

```
Vi test4
Echo "enter a number"
read num
a=num
num=`expr $num * $num`
echo "square of given number $a is:$num"
:wq(save and quit)
$sh test5
```

5)write a shell script to know how many no. ofusers logged in.

```
Vi test6
echo `who|wc –l` users are currently logged in
:wq
```

6)write a shell script to accept a file and delete it.

```
echo "enter a file"
read fname
rm fname
echo "$fname is deleted"
```

7)write a shell script to know present working directory and loginusername and no. ofusers working in my organization,present date.

```
vi test7
```

```
pwd
logname
echo `cat /etc/passwd/ | wc –l`
date
```

8)write a shell script to accept a file and open it contents.

```
echo "enter a file"
        read fn
        cat $fn
    echo "file opened successfully"
```

OPERATORS:-

1.Arithmetical operators

2.Relational operators

3)boolean operators

4)file test operators

Arithmetical operators:-There are following arithmetic operators supported by Bourne Shell. Assume variable a holds 10 and variable b holds 20 then:

operator	description	example
+	Addition - Adds values on either side of the operator	`expr $a + $b` will give 30
-	Subtraction Subtracts right hand operand from lefthandoperand	`expr $a - $b` will give 30
*	Multiplication - Multiplies values on either side of the operator	`expr $a * $b` will give 200
/	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
%	Divides left hand operand by right hand	`expr $b % $a` will

	operand and returns remainder	give 0
=	Assignment - Assign right operand in left operand a=$b would assign value of b into a	a=$b would assign value of b into a
==	Compares two numbers, if both are same then returns true.	[$a == $b] would return false
!=	Compares two numbers, if both are different then returns true	[$a != $b] would return true

—

Relational Operators:- Bourne Shell supports following relational operators which are specific to numeric values. These operators would not work for string values unless their value is numeric.

For example, following operators would work to check a relation between 10 and 20 as well as in between "10" and "20" but not in between "ten" and "twenty". Assume variable a holds 10 and variable b holds 20 then:

Operator	description	example
-eq	Checks if the value of two operands are equal or not, if yes then condition becomes true	[$a -eq $b] is not true
-ne	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true	[$a -ne $b] is true
-gt	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	[$a -gt $b] is not true

-lt	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true	[$a -ge $b] is not true
-le	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	[$a -le $b] is true.

Logical Operators or Boolean operators:- There are following boolean operators supported by Bourne Shell.Assume variable a holds 10 and variable b holds 20 then:

operators	description	example
!	This is logical negation. This inverts a true condition into false and vice versa	[! false] is true
-o	logical OR. If one of the operands is true then condition would be true	[$a -lt 20 -o $b -gt 100] is true
-a	This is logical AND. If both the operands are true then condition would be true otherwise it would be false	[$a -lt 20 -a $b -gt 100] is false.

9)write a script to accept a string and find out its length.

```
Vi test9
echo "enter a string"
read str
l=`echo str |wc –c`
echo length of given string is `expr $l – 1`
```

10)write a script to accept two float numbers and perform addition and subtraction operations.

```
echo "enter a no"
read a
echo "enter another no"
read b
echo addition of two floats is: ` echo $a + $b | bc `
echo addition of two floats is: ` echo $a - $b | bc `
```

FLOW CONTROL STAMENTS:-

1)if:- The **if** command allows you to conditionally execute a command or set of commands depending upon whether some expression is true. There are two forms.

Syn:-

```
if [ logical expression ]
then
(statements)
else
(statements)
fi
```

or

```
if UNIX command
then
(statements)
else
(statements)
fi
```

while:- In programming, a loop means a part of a program that can be executed two or more times in succession. The while statement is the

simplest looping statement . It repeatedly executes a statement as long as a condition is true. It looks like this:

Syn:-while [condition]
do
(statements)
done

for:- The for statement makes it more convenient to count iterations of a loop. The general form of the for statement looks like this:

syn:- for variable in vlue1,value2,…..valuen
do
(statements)
Done

Until:- If the resulting value is *false*, given *statement(s)* are executed. If command is true then no statement would be not executed and program would jump to the next line after done statement.

Syn:-until [condition]
do
(statements)
done

eg:- #!/bin/sh
a=0
until [! $a -lt 10]
do
echo $a
a=`expr $a + 1`
done

Beside pgm gives output like below.

```
0
1
2
3
4
5
6
7
8
9
```

The case statement:-

case is a flow control construct that provides for multi-way branching based on patterns.Program flow is controlled on the basis of the *word*given. This *word* is compared with each pattern in order until a match is countd, at which point the associated *command*(*s*) are executed.

```
case word in
pattern1) command(s)
;;
pattern2) command(s)
;;
patternN) command(s)
;;
esac
```

When all the commands are executed control is passed to the first statement after the esac. Each list of commands must end with a double semi-colon (;;).

The break statement:-

The break statement is used to terminate the execution of the entire loop, after completing the execution of all of the lines of code up to the break statement.

Syntax:

The following break statement would be used to come out of a loop:

```
Eg:- a=0
 while [ $a -lt 10 ]
 do
   echo $a
   if [ $a -eq 5 ]
   then
      break
   fi
   a=`expr $a + 1`
done
```

```
The beside program produces
the following output
0
1
2
3
4
5
```

The continue statement:-

The continue statement is similar to the break command, except that it causes the current iteration of the loop to exit, rather than the entire loop.

This statement is useful when an error has occurred but you want to try to execute the next iteration of the loop.

Syn:- continue

```
Eg:- NUMS="1 2 3 4 5 6 7"

for NUM in $NUMS
do
   Q=`expr $NUM % 2`
   if [ $Q -eq 0 ]
   then
      echo "Number is an even number!!"
      continue
   fi
```

```
  echo "Countd odd number"
done
```

11) write a script accept a number and check the given even or odd

```
        echo "enter a number"
         read n
       if  [  ` expr $n %  2  `  -eq  0  ]
        then
         echo "$n is even"
       else
          echo " $n is odd"
          fi
```

12)write a script to accept three no's and find biggest number among them.

```
    echo "enter a no"
  read a
echo "enter another no"
read b
echo "enter again another no"
 read c
if [ $a  -gt  $b  -a  $a  -gt  $c ]
then
  echo "$a is greater"
else
  if  [ $b  -gt  $c ]
  then
     echo "$b is greater"
else
    echo "$c is greater"
  fi
fi
```

13)write a program to findout factorial for a given number.

```
echo "enter a no"
read n
```

```
i=1
while [ $n –gt 0 ]
do
i=`expr  $n \* $i`
n=`expr $n – 1`
done
echo  "factorial of given no is" $i
```

14)write a shell script to accept numbers form user and find out biggest number.

```
echo "enter a number"
read num
echo "enter a number"
read num
big=num
i=2
while [ $i  –le  $n ]
do
echo "enter a number"
read num1
 if  [ $num  -gt  $big ]
then
  big=$num1
fi
i=`expr $i + 1`
done
echo "biggest number is:$big"
```

15)write a shell script to accept a numbers and findout smallest among them.

```
echo  for how many numbers u want to find smallest number
```

```
read n
echo "enter a number"
read num
small=num
i=2
while [ $i  –le  $n ]
do
echo "enter a number"
read num1
 if  [ $num  -lt $small ]
then
  small=$num1
fi
i=`expr $i + 1`
done
echo "biggest number is:$small"
```

16)write a shell script to find out fibinacci series.

```
   f0=0
  f1=1
echo "enter range"
read num
echo  $f0
echo $f1
while [ $num –gt 0 ]
do
f2=`expr  $f0 + $f1`
echo $f2
f0=$f1
f1=$f2
```

```
num=`expr $num -1`
done
```

17)write a shell script to findout whether given number is prime or not.

```
 i=2
  count=0
echo "enter a number"
read num
while [ $num –ge $i ]
 do
if  [  `expr  $num  % $i`  -eq 0 ]
then
 count=`expr $count + 1`
  fi
i=`expr  $i  + i`
done
if [ $count  -gt 1 ]
then
echo "$num not a prime"
else
echo "$num is a prime"
fi
```

18)write a shell script to find out SIMPLE INTEREST.

```
echo "enter principle"
read p
echo "enter rate iof interest"
read r
echo "enter time interval"
read t
Si=`expr $p \* $t  \* $r /100`
```

```
echo "simple interest is:"$si
```

19)write a script to accept a file and display no. oflines and words in that.

```
echo  "enter a file"
read fname
echo `wc –l $fname`
echo `wc –w $fname
```

20)write a script to find out multiplication of a particular number.

```
  echo "enter a number"
  read num
i=1
while [ $i  -le 10 ]
do
j=`expr  $num \* $i`
echo  "$num \* $i" = $j
i=`expr  $i + 1`
done
```

21)write a shell script to accept students marks and show whether fail or pass.

```
echo "enter three subject marks:"
read m 1 m2 m3
if[$ml-ge 35  -a $m2 –gt  40  -a $m3 –ge 40 ]
then
  echo "student passed"
else
 echo"student failed"
fi
```

22)write a script to display day in a short form.

```
  c=`date |cut –d”,” –f 1`
echo $c
```

23) write a script accept a string and check the given string is empty or not.

```
echo enter a string
read str
if  [  -z  $str  ]
then
echo "string is empty"
else
echo   “given string is not empty"
fi
```

24) write a shell script to accept a filename and check the given file is a file or a directory

```
echo "enter a file name:"
read fn
if  [ -e  $fn  ]
 then
       if  [  -f  $fname ]
          then
            echo “ $fn is file”
      else
                  if  [ -d $fname ]
                then
                    echo $fn  is a directory”
                 else
                    echo $fn is not directory
                         fi
       fi
else
```

```
  echo  $fn doesnot exist
fi
```

25) write a program to compare two strings

```
echo "enter first string:"
read strl
echo "enter second string:"
read str2
if  test  "$strl"  =  "$str2"
then
echo "both strings are equal"
else
echo "strings are not equal"
fi
```

26) Write a program to accept a file and append data to that file

```
echo "enter a filename"
read $fname
if  [ -f $fname ]
then
if [ -w $fname ]
then
echo "enter data to file "
cat»$fhame
else
chmod u+w $fname
echo "enter data to file "
cat»$fname
fi
else
echo "pla enter a valid file name"
```

```
fi
```

27)write a script whether a file has user executable permissions or not.

```
  echo "enter a file"
  read fn
  if [ -r $fn  -a  -x $fn ]
 then
   echo $fn has execute permission
  else
   echo $fn has no execute permission
  fi
```

case EXAMPLES:-

28)

```
echo "enter a number between 1 to 4:\c"
        read num
      case $num in
1 )echo "ENTER 1" ;;
2)echo "ENTER  2" ;;
3)echo "ENTER 3" ;;
4)echo "ENTER 4" ;;
*)echo "invalid number, enter number between 1-4"
esac
```

29) Write a program to check given character is upper case alphabet or lower
case alphabet or digit or special character.

```
echo "enter a single character"
read choice
case $choice  in
[a-z])echo "you entered a small case alphabet"  ;;
```

```
[A-Z])echo "you entered a upper case alphabet"  ;;
[0-9])echo "you entered a digit"  ;;
 *)echo "you entered a special character" ;;
esac
```

30) write a script accept a month and display quarter of the month.

```
    echo  "enter a month[jan]"
     read mon
   case  $mon  in
   [Jj][fF][mM]*|[Ff][Ee][Bb]*|[Mm][Aa][Rr]*)echo "first quarter"
   apr|mar|jun)echo "second quarter"
    jul|aug|sep)echo"third quarter"
    oct|nov|dec)echo"fourth quarter"
  *) echo "invalid month
     esac
```

31) write a menu driven program which has following options

1 .contents of current directory
2.list of users who have currently logged in
3.present working directory
4.calendar
5.date
6.exit

o/p:-

```
echo "Main Menu
echo  "1.contents of a directory"
echo"2.1istofusers
echo "3.present working directory
echo "4.display calendar
echo "5.date"
```

```
echo "6.exit"
echo "enter your choice:\c"
read choice
case Schoice in c
1) ls -x
2) who
3)pwd
4)cal
5)date
6)exit
esac
```

32)write a shell script to copy a given file to another directory.

```
echo "enter a filename"
read fn
  if  [  -e  $fn ]
then
cp  $fn  ~/jaya/loka/
directory
else
  echo $fn is not exist
fi
```

o/p:-it copies given file to loka directory.loka directory presents in jaya directory,jaya directory presents in user's home directory.

33)write a pgm by using until.

```
     i=10
   until [ i –le 1 ]
do
    echo  $i
```

```
i=`expr  $i – 1`
done
```

34)write a shell script to print even numbers form the given range.

```
  echo "enter a start value"
   read st
echo  "enter a end value"
  read st1
    while [ $st –le  $st1 ]
      do
    if  [ `expr $st % 2` -eq 0 ]
   then
     echo "$st is even number"
  fi
done
```

35) write a shell script to print odd numbers form the given range.

```
  echo "enter a start value"
   read st
echo  "enter a end value"
  read st1
    while [ $st –le  $st1 ]
      do
    if  [ `expr $st % 2` -gt 0 ]
   then
     echo "$st is odd number"
  fi
done
```

36) example of sleep

#sleep is used for to stop the executing for specified no of seconds

```
While true
do
clear
tput cup 5 8
echo "welcome to durgasoft"
Sleep2
clear
tput cup 5 8
echo  "durgasoft offering many courses"
sleep2
done
```

37) write a shell script to display contents of a file.

```
While true
do
echo "enter a file name to open"
read fn
If [  -e $fn   –a   –f   $fn  -a –r  $fn  ]
then
cat $fn
break
else
continue
fi
done
```

for loop;-

38)write a shell script to display numbers using for loop

o/p:- for i in 1 2 3 4 5

```
do
echo  $i
 done
```

39) Write a script retrieve details of employees who are receiving salary more than 6000 in deptno 10 from emp file and insert into emp1 file.

```
#101, lokesh,10000,10
#102,venkat,5000,20
#103,anuja,777777000,30
#104,priya,9000,10

for i in cat emp
do
j= echo  $i  |  cut  -d"."  -f  3
k= echo  $i  |  cut  -d"."  -f  4
if [ $j –ge  6000  -a  $k  -eq  10 ]
then
echo $i >>emp1
fi
done
```

40)write a shell script to find out how many files are there in my directory.

```
      echo "enter a directory"
       read  dir
count=0
if [ -d $dir ]
 then
       for i in $dir
        do
        if  [  -e $i  -a  -f  $i ]
```

```
            then
             count =`expr $count + 1`
         fi
    done
else
    echo "$fn is not a directory"
  echo no. files in given directory are :$count
fi
```

41)write a shell script to find out multiplication table of the given number.

```
      echo  "enter a number"
      read n
    i=1
     while [  $i  -le  10 ]
       do
      echo  $n  *  $i= `expr  $n \* $i`
       i=`expr $i + 1`
       done
```

42)write a shell script to display all file names and directory names in a current directory.

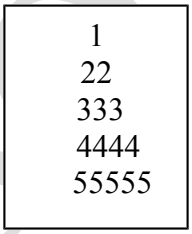
```
       for f in 'ls –l'
       do
          if  [ -e $f  -a –f  $f  ]
           then
            echo  $f is a file
          else
          if [ -d $f ]
              then
                    echo "$f is a directory"
           fi
```

```
        fi
    done
```

43)write a shell script to display 10 digit mobile number in a given file.

```
    echo "enter a file"
      read fn
    if [ -f $fn]
        then
  echo `egrep –o "/<[0-9]{10}\>"  $fn `
      fi
```

44) write a shell script like beside format.

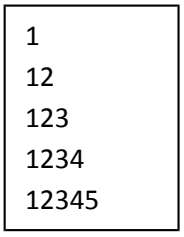


o/p:-

```
for (( i=1; i<=5; i++ ))
do
   for (( j=1; j<=i;  j++ ))
   do
    echo -n "$i"
   done
   echo ""
done
```

45)write a shell script like beside format.

o/p:-

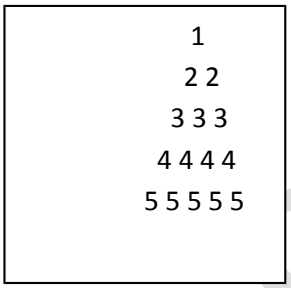



```
for (( i=1; i<=5; i++ ))
do
   for (( j=1; j<=i;  j++ ))
   do
    echo -n "$j"
   done
   echo ""
```

done

46)write a shell script like beside format.

o/p:-




```
for (( i=1; i<=MAX_NO; i++ ))
do
   for (( s=MAX_NO; s>=i; s-- ))
   do
      echo -n " "
   done
   for (( j=1; j<=i;  j++ ))
   do
    echo -n " $i"
   done
   echo ""
done
```

```
for (( i=1; i<=MAX_NO; i++ ))
do
   for (( s=MAX_NO; s>=i; s-- ))
   do
      echo -n " "
   done
   for (( j=1; j<=i;  j++ ))
   do
    echo -n " ."
   done
   echo ""
done
```

47)write a shell script to convert all lower case letters to upper case in a file.

```
echo "enter a file name"
read fn
if [ -f $fn ]
then
tr "[a-z]" "[A-Z]" < $fn
else
echo "plz enter a valid filename"
fi
```

COMMAND LINE ARGUMENTS:-The arguments which are passed at the command prompt can be called as command line arguments. The arguments whatever we pass at the command prompt shell puts each argument into special variables. For this purpose shell is using some predefined variables. These can be called as variables of shell. They are

The Special variables hold parameters values. The Special variables are

$0,$1,$2,$3,$4,$5,$6,$7,$8,$9, $#,$*,$?,$$.

---

48)write a shell script to perform addition of two numbers.

```
for k in $*
do
echo `expr $1 + $2 `
done
```

49)write a shell script to perform remove operation of a given file

```
for j in $1
do
if [ -j –eq 1]
then
if [ -f $j ]
then
```

```
    rm $j
else
  echo "file not exist"
  fi
else
 echo  invalid no. ofarguments
fi
done
```

Functions:- function is a subroutine ,which can be used to perform specific task,by using functions we can reduce redundancy of code.

Syn:-

Declaring a function:-function functionname ()

{

}

Calling a function:- functionname value1 value2….value n

```
 Eg:-50)generate_list ()
       {
          echo "one two three"
       }
for word in $(generate_list)  # Let "word" grab output of function.
do
 echo "$word"
done
```

Job Control:- Jobs Are of 2 types.

1)Fore Ground Job

2)Back Ground Job

By default every job comes under foreground job, we can make foreground job as Back ground job.

Foreground vs. Background

A foreground job can receive keyboard input and signals such as Control-C from the controlling terminal, background jobs cannot. If the login session is disconnected, foreground jobs are terminated by a hang-up signal, while backgrounds jobs are not.

Putting a Job in the Background

Jobs can be put in the background either by initiating them in the background or by stopping a foreground job and then specifying that it be continued in the background. The way a job is initiated in the background is by putting an ampersand (&) at the end of the command line.

Useful Commands--Summary

control-z	Stop (don't kill) the foreground job, and then return to the shell
jobs	Check the status of jobs in the current session
ps -u	Check the status of processes, including those from other sessions.
kill -9 123	Kill a process, by specifying its process id (PID) number
bg	Run the most recently stopped job in the background
fg	Bring most recently backgrounded job to the foreground
fg 1	Bring a job to foreground by specifying its job number after the percent sign