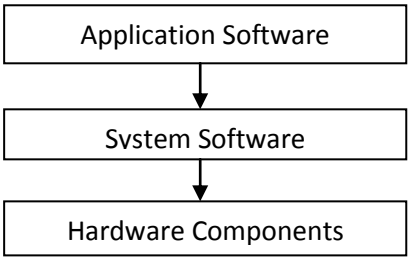
UNIX/LINUX

Software:

Software is a collection of computer programs and related data that provide the instructions for telling a computer what to do and how to do it.

Software is divided into two types.

1. **System S/W:**
   System S/W is a combination of device drivers and operating system.
2. **Application S/W:**
   Application software, also known as an application or an "app", is computer software designed to help the user to perform specific tasks. Application S/W communicates with hardware components trough System S/W.



Device drivers:

It is a hardware program used to communicate with hardware devices.

Every device in system will have its own device driver.

❖ Operating System Overview:

An operating system (O/S) is a set of programs that manage computer Hardware resources and provide common services for application software

. The operating system is the most important type of System software in a computer system. A user cannot run an application program on the computer without an operating system, unless the application program is self booting.

- O/S is a platform where we can run application S/W (c, c++, java, etc..,)
- O/S is an interface between Application S/W and Hardware Components
- O/S provides an environment to run applications. That environment is called “child process”. O/S is called parent of all current running applications.

OS is divided into two types of interfaces

1. **CUI (Character User Interface)**
   Here Work done by commands. Here you are allowed to work only with keyboard. In CUI one task runs at a time.
   Application: MS Dos

2. **GUI (Graphical User Interface)**

Here work done in user friendly environment. Here you are allowed to work with any pointing device like mouse. In GUI more tasks can run simultaneously.

Application: Window

❖ History of Unix/Linux

Before UNIX operating system we have DOS operating system which is machine dependent and single user O/S. By using this O/S wastage of much hardware and time consumed to develop any project.

To overcome the disadvantages of DOS O/S, in 1969 at AT & T Bell labs software team lead by Dennis Ritche, Ken Thomson, Rudd canady and Brian Kernighan developed a project called **MULTICS** (Multiplexed Information Computing System).

MULTICS was developed to share the resources. It was developed only for two users. MULTICS was developed in assembly language. In the same year it is modified to hundreds of users and named as **UNICS** (Uniplexed information Computing System).

In 1972 "C" – language was developed, which is most powerful language.

In 1973 UNICS was rewrote in C- language and renamed as **UNIX.**

❖ Features of UNIX/Linux

- **Multiuser O/S:**

UNIX server supports the multiple users with the help of process based technology. Process based technology works based on time sharing between users and queue.

- **Multi Tasking O/S**

User can perform more than one job at a time. User can run the task as background process by affixing '&' (ampersand) to the command and can run one more job as foreground job; foreground will be given highest priority and next is its background job. So user can run important job as foreground job.

Advantage of Multi tasking is to utilize the maximum CPU time

- **Multiprogramming**

Support more than one program, in memory, at a time. Amounts to multiple user processes on the system.

- **Programming facility**

UNIX provides shell which works like a programming language. It provides the commands, keywords, control statements and operators to develop the shell scripting for system administrators, developers and testers.

- **Open System**

UNIX from the beginning is an open source. User can modify the Unix O/S as per his requirements. User can add new system devices and update complete Unix O/S

- **Portable O/S**

UNIX is an independent of hardware it works with all processors from 8085 to super computer.

- **Software Development Tools**

UNIX offers an excellent variety of tools for software development for all phases, from program editing to maintenance of software.

- **Security**

It provides security for local resources like files and hardware devices

❖ Flavors of UNIX

Many of the proprietary flavors have been designed to run only (or mainly) on proprietary hardware sold by the same company that has developed them. Examples include:

- **AIX** - developed by IBM for use on its mainframe computers
- **BSD/OS** - a commercial version of BSD developed by Wind River for Intel processors
- **HP-UX** - developed by Hewlett-Packard for its HP 9000 series of business servers
- **IRIX** - developed by SGI for applications that use 3-D visualization and virtual reality
- **QNX** - a real time operating system developed by QNX Software Systems primarily for use in embedded systems
- **Solaris** - developed by Sun Microsystems for the SPARC platform and the most widely used proprietary flavor for web servers
- **Tru64** - developed by Compaq for the Alpha processor

Others are developed by groups of volunteers who make them available for free. Among them are:

- **Linux** - the most popular and fastest growing of all the Unix-like operating systems
- **FreeBSD** - the most popular of the BSD systems (all of which are direct descendants of BSD UNIX, which was developed at the University of California at Berkeley)
- **NetBSD** - features the ability to run on more than 50 platforms, ranging from acorn26 to x68k
- **OpenBSD** - may have already attained its goal of becoming the most secure of all computer operating systems
- **Darwin** - the new version of BSD that serves as the core for the Mac OS X

This diversity has had both positive and negative effects. Most importantly, it has resulted in a healthy competition amongst them, which has been a major factor in the rapid improvement of the Unix-like systems as a whole

❖ Comparison of UNIX with Windows

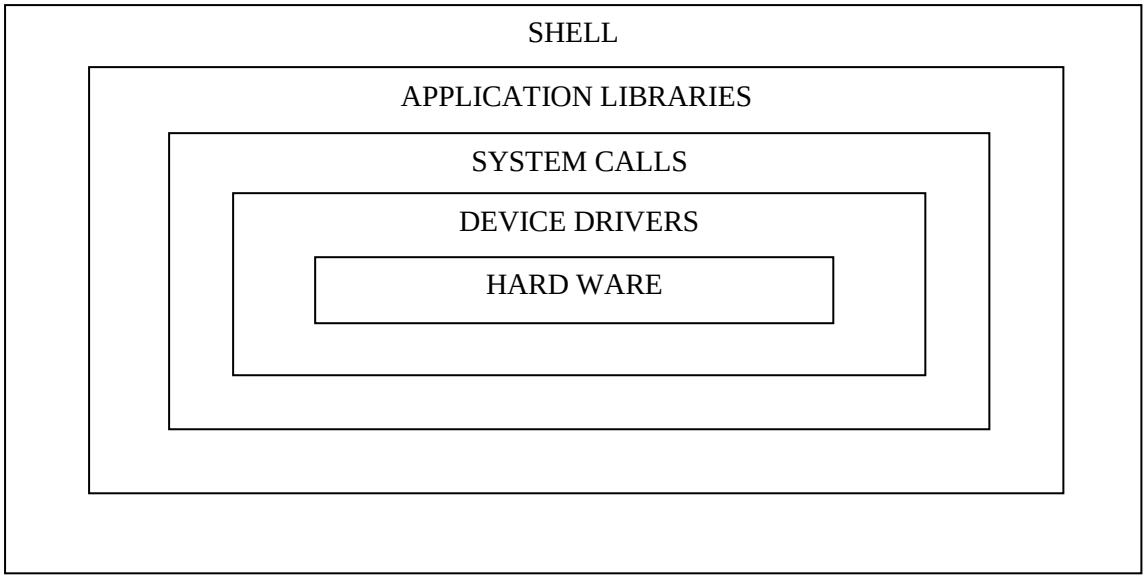
1. UNIX and Windows both Multi user O/S.
2. UNIX and Windows both support multi tasking.
3. UNIX is process based technology. Windows is thread based technology.
4. Process based technology performance is less and it is a heavy weight technology. Thread based technology is a light weight technology and performance is more.
5. UNIX is independent of machines. Windows is an Intel Processor Based.
6. UNIX O/S performs more than Windows with high speed processors in the market.
7. UNIX is an open system, Windows is a closed system.
8. UNIX provides system call access where as windows cannot.
9. UNIX is portable O/S, windows is not.
10. UNIX provides the programming facility where as Windows will not.
11. Unlimited users can work with UNIX O/S, Windows only limited users.
12. UNIX provides security for local resources where as Windows will not provide.
13. UNIX is not user friendly. Windows is User friendly.

❖ Architecture of UNIX

Some key features of the UNIX architecture concept are:

- UNIX systems use a centralized operating system kernel which manages system and process activities.
- All non-kernel software is organized into separate, kernel-managed processes.
- UNIX systems are preemptively multitasking: multiple processes can run at the same time, or within small time slices and nearly at the same time, and any process can be interrupted and moved out of execution by the kernel. This is known as thread management.
- Files are stored on disk in a hierarchical file system, with a single top location throughout the system (root, or "/"), with both files and directories, subdirectories, sub-subdirectories, and so on below it.
- With few exceptions, devices and some types of communications between processes are managed and visible as files or pseudo-files within the file system hierarchy. This is known as everything is a file. However, Linus Torvalds states that this is inaccurate and may be better rephrased as "everything is a stream of bytes".

Structure of UNIX O/S



Shell:

Unix shell is a command-line interpreter or shell that provides a traditional user interface for the Unix operating system and for Unix-like systems. Users direct the operation of the computer by entering commands as text for a command line interpreter to execute or by creating text scripts of one or more such commands.

Shell is an interface between User and Kernel. Shell works like a programming language.

Kernel:

Kernel is collection of system Calls and Device Drivers. In computing, the kernel is the main component of most computer operating systems; it is a bridge between applications and the actual data processing done at the hardware level.

The kernel's responsibilities include managing the system's resources (the communication between hardware and software components).

Usually as a basic component of an operating system, a kernel can provide the lowest-level abstraction layer for the resources (especially processors and I/O devices) that application software must control to perform its function.

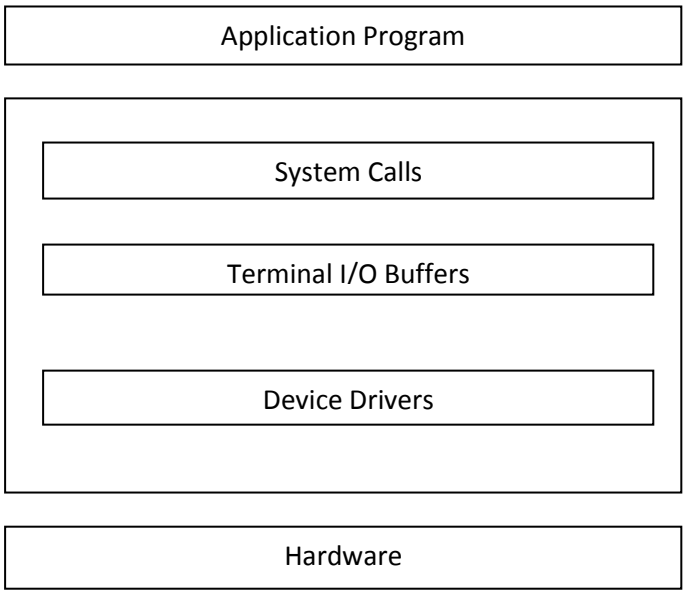
It typically makes these facilities available to application processes through inter-process communication mechanisms and system calls.

Any version of UNIX will have only one kernel and it will be loaded when UNIX is booted.

Kernel process many jobs some important jobs are

1. File Management
2. Time sharing Between users
3. Device Management
4. Process Management
5. Processor Management
6. Memory Management
7. Inter Process Communication
8. Signaling System
9. File Sharing

UNIX Terminal I/O (Kernel Architecture)

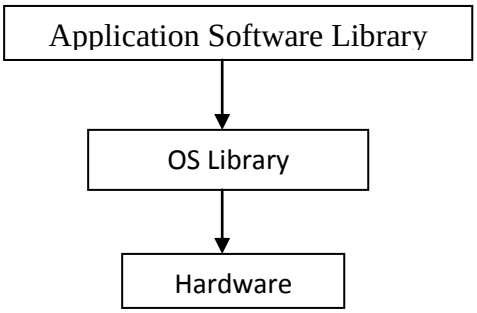


Application Software Library

It is a part of Application Software. It will be generated automatically when the application software installed.

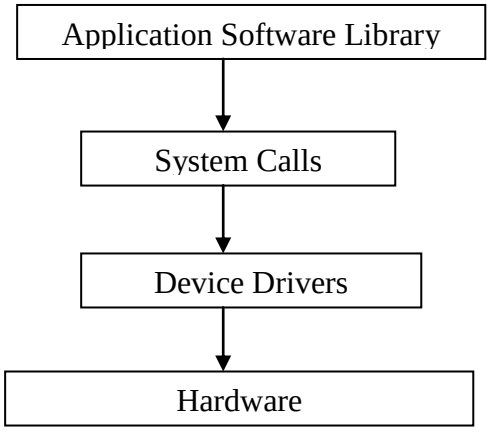
For Example, C language header files, java packages, Oracle packages, etc.

Application Software Library communicates with Hardware components through the OS library.



System Calls

System Calls is a low level function to communicate with Hardware components through the Drivers. It is called as "OS Library"



Note:

UNIX provides 232 System calls. LINUX provides 363 System calls.

Ex:

Open () to activate device

Write () to write

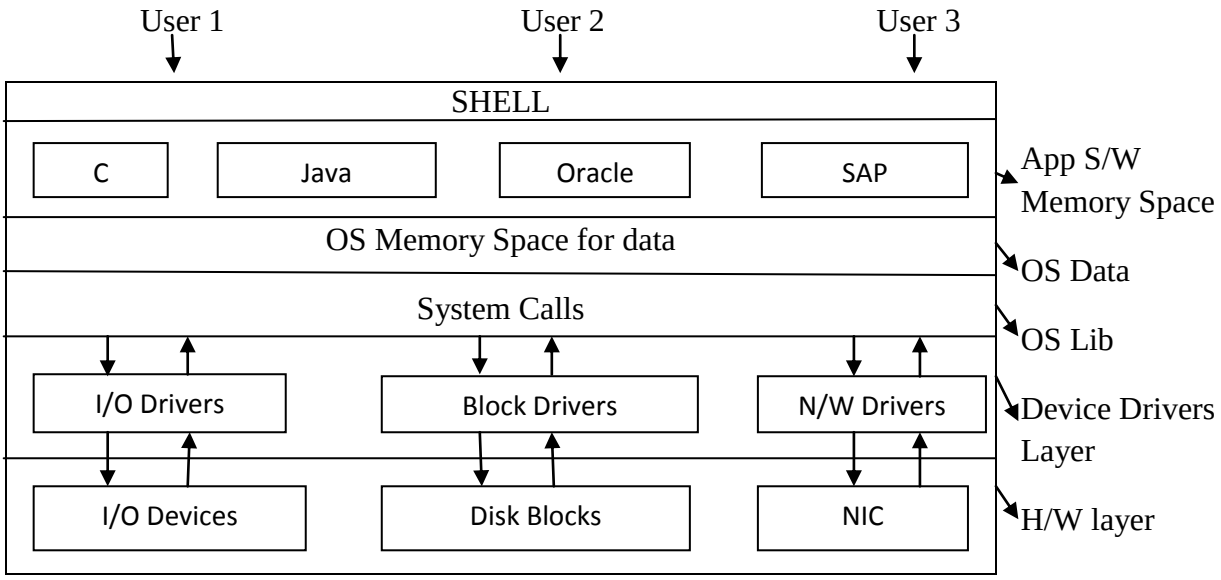
Close () to deactivate device

Fork (), pipe (), etc.

Device Drivers

- These are hardware programs developed in Assembly language and C language.
- These are separated by Hardware vendors depending upon components.
- These are to communicate with Hardware components.

Internal Architecture of UNIX



OS drivers to Device Drivers Layer called as OS Private Area. OS doesn't allow the Application programs to access its private area.

But C program can access the OS private area through its pointers.

Combination of both Device Drivers Layer and OS library is called as Kernel.

❖ Filing System Of UNIX

Types of Files in UNIX:

There are mainly 3 types of files in UNIX.

1. Regular or Ordinary files
2. Directory files
3. Device files

- **Regular Files**

  The Regular Files consists of data in either Text format or Binary format.

- **Directory Files**

  These files contain entries of Files.

- **Device Files**

  These are of two types

- **Character Special Files**

  These files only handle character formatted data.

  Ex: stdin, stdout and stderr files
- **Block Special Files**

  Name given to specific blocks of hard disks.

There are three more File Types in UNIX.

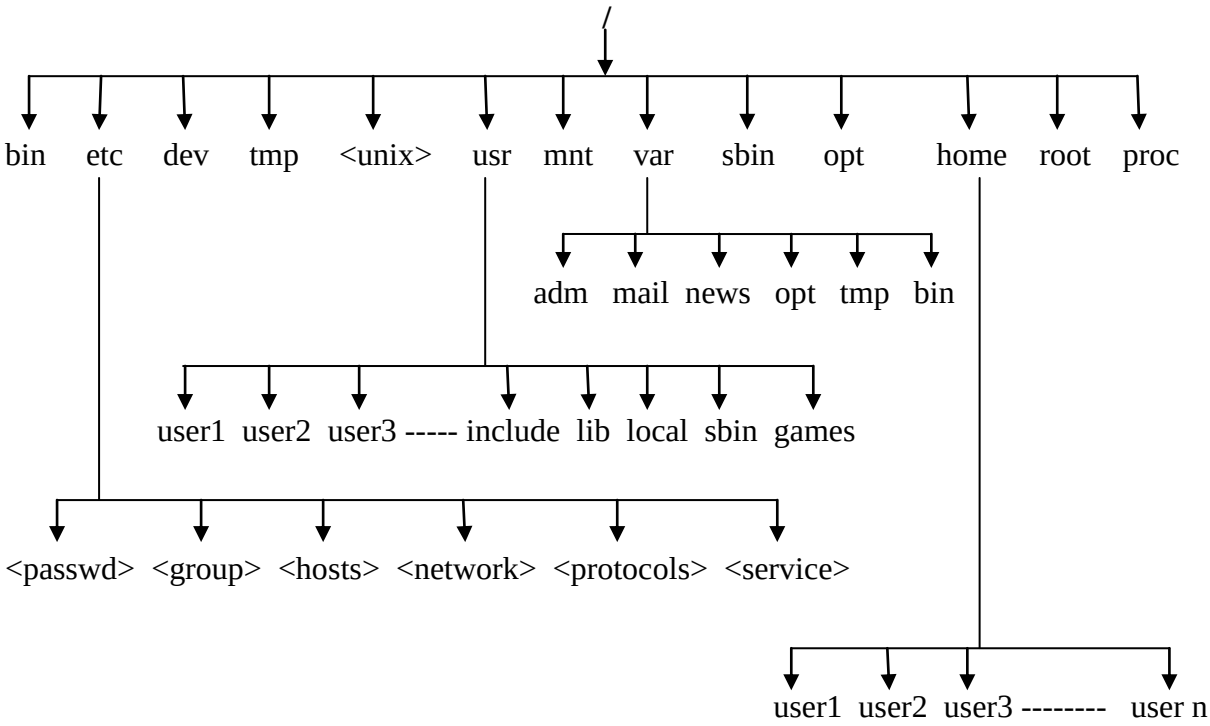
- **FIFO File**

  Used to communicate between two processes running in same system.
- **Socket File**

  Used to communicate between two processes running on different systems in a Network.
- **Symbolic Files**

  It is a link or Pointer for already existing file.

Hierarchy of Filing System



/:

It is the top most directory in UNIX File System hierarchy. This is called as Root Directory. It's a reference point of entire filing system and is system administrator login in UNIX O/S.

/bin:

It contains UNIX Commands.

/etc:

Contains system data files.

/dev:
Device files of UNIX O/S.

/tmp:
Temporary files. These files will be automatically deleted when the UNIX server is shut down.
Eg: log files
/<unix>:
It's a file contains kernel programs (only UNIX O/S).
/usr:
Users working directory.
/mnt:
To mount the system devices like floppy, cd, printers and others.
/var:
It contains variable directories and files. This varies user to user.
/sbin:
Contains system library files like bootable files
Ex: init
/opt:
Contains optional files and directories of system admin (personal files of admin)
/home:
User working directory in LINUX.
/root:
Its system admin login directory in LINUX.
/proc:
Process. It maintains all process id's of current running jobs at the O/S level (all users)

Home, root, proc are the directories only from LINUX & SOLARIS, not provided by UNIX.

/usr/user1, user2, user3….. are the users working directories. These are also called as user home directories.
/usr/include:
Contains C and C++ header files.
/usr/lib:
Contains C and C++ libraries files which contain coding part of header files.
/usr/local:
To install local software.
/usr/sbin:
User bootable files.
/usr/games:
Contains binary files of games.
/var/adm:
Contains admin files of users.
/var/mail:

Mails received from normal users.

/var/news:

Mails received from system admin.

/var/opt:

User optional files.

/var/bin:

User binary files.

/var/tmp:

User temporary files.

/etc/<passwd>:

Maintains the user details like name, passwd, uid, gid, comments, type of shell, home directory.

/etc/<group>:

Maintain the group details like name, passwd, gid and name of the users working under the group.

/etc/<host>:

Maintains host details like IP address and DNS name.

/etc/<network>:

Information about category of network like classes A, B, C, D, and E.

Class A: 0-127, **Class B:** 128-191, **Class C:** 192-223, **Class D:** 224-239, **Class E:** 240-256.

Login Process:

Login: user2
Password: ********

$ → Normal user prompt
# → System admin prompt

Commands:

$ logname → to check present working user.
User2

$ clear → to clear screen

We can also use cntrl+l to clear screen

$exit → to quit from user session.

Help Facility:

Syn:

1. $ man [command name] → displays use and options of the command.

Ex: $ man date

Displays date command usage and options.

$ man cal

$ man man

2. $ pwd → to display the present working directory

/home/user2

3. $ cal → to display calendar (current month calendar)

$ cal 2000 → to display 2000 year calendar.

$ cal 08 2010 → august month of 2010 year calendar

$ cal 78 → 78 year calendar.

$ cal 1 → 1st year calendar.

$ cal 10000 → illegal year usage range upto 1-9999 only

If you pass only one parameter after cal it considers it as year. If you pass two parameters 1st parameter is month and 2nd parameter is year.

4. $ date → to display date (displays system or server's date which it is working)

Options:

$ date +%m → to display month

12

$ date +%h → to display name of the month

December

$ date +%d → to display day of month (1 to 31)

22

$ date +%y → to display last two digits of the year.

11

$ date +%Y → to display four digits of the year.

2011

$ date +%D → to display date in MM/DD/YY format.

12/22/11

$ date +%T → to display time in the format hh:mm:ss

12:30:54

H, M and S → this options are used to display hour, minutes and seconds.

5. **$ who** → to display present working users.

   User2 tty1

   User3 tty3

   User4 tty0

Here tty is the UNIX naming for terminals.

LINUX naming for terminals is Pts1, Pts2 ……

Terminal number is used identify the clients.

6. **$ finger** → displays more information about the users like name of the user, phone number, idle time etc ..,

7. **$ who am i** → displays current working users details

   User2 tty1 2011-11-12 17:20 ip address

8. **$ whoami** → displays current working user without details

   User2

9. **$ tty** → to display terminal type

10. **$ sleep [time in sec]** → to take the shell into sleeping state.

    $ sleep 5

    For 5 seconds the shell will be going to sleep state

11. **Executing multiple commands**

    $ cmd1; cmd2; cmd3; cmd4

Ex: $ ls; sleep 5 ; date

Here first list of the files will be displayed then it goes to sleeping state for 5 seconds and it displays date.

12. Wild card characters

A number of characters are interpreted by the UNIX shell before any other action takes place. These characters are known as wildcard characters. Usually these characters are used in place of filenames or directory names.

*	An asterisk matches any number of characters in a filename, including none.
?	The question mark matches any single character.
[ ]	Brackets enclose a set of characters, any one of which may match a single character at that position.
-	A hyphen used within [ ] denotes a range of characters.
~	A tilde at the beginning of a word expands to the name of your home directory. If you append another user's login name to the character, it refers to that user's home Directory.

Here are some examples:

1. cat c* → displays any file whose name begins with *c* including the file *c*, if it exists.
2. ls *.c → lists all files that have a *.c* extension.
3. cp ../abc?. → copies every file in the parent directory that is four characters long and begins with abc to the working directory. (The names will remain the same)

4. ls abc[34567] → lists every file that begins with abc and has a *3*, *4*, *5*, *6*, or *7* at the end.
5. ls abc[3-7] → does exactly the same thing as the previous example.
6. ls ~ → lists your home directory.
7. ls ~user1 → lists the home directory of the user with the user id user1.
8. $ ls file [1-9] [1-9] → in the filename 5th char should be 1-9 and 6th should be 1-9.
   Ex: file 21 file34 file 56

LINUX Wild card characters

Here are wildcards and regular expressions:

* — Matches all characters

? — Matches one character

* — Matches the * character

\? — Matches the ? Character

\) — Matches the) character

❖ Working with Directories

- **Displaying Directory Contents**

$ ls → To display the present working directory contents.

Options:

$ ls –a → to display all files including hidden files like . (Dot) and .. (Double dot) files

$ ls |pg → to display contents page wise (only UNIX).

$ ls |more → to display contents line by line in UNIX and LINUX.

$ ls –x → to display in width wise.

$ ls –x|more → to display contents width wise and line wise.

$ ls –f → to display only files.

$ ls –F → to display all files including exe files.

$ ls –R → to display including sub directories recursively like tree structure.

$ ls –r → to display in reverse order.

$ ls –d → to display only directories.

$ ls –t → to display based on date and time of creation of files (latest to old files)

$ ls –u → to display based upon last accessed time.

$ ls –s → to display files including number of blocks used by the file and directories.

$ ls –i → to display files including I-node number of files. I-node number provides information about the files.

$ ls –l → to display long list of the files.

Ex:

$ pwd

/home/user2

$ ls –l

d rwxrw_rw_ 3 user2 group1 5436 feb22 14:00 xxx

_ rw_rw_r__ 2 user2 group1 231 oct21 10:00 file1

In above example First character is the type of the file.

_ → Regular File

d → Directory File

c → Character Special File

b → Block Special File

f → FIFO File

s → Socket File

l → Symbolic File

- **Creating a Directory**

  $ mkdir [dir name] → To create a Directory

- **Changing to a Directory**

  $ cd [dir name] → To change into the Directory

- **To Create Multiple Directories**

  $ mkdir [dir1] [dir2] [dir3] ....

- **To move back to Parent Directory**

  $ cd ..

- **To move two levels up from PWD**

  $ cd ../..

- **To move three levels up from PWD**

  $ cd ../../..

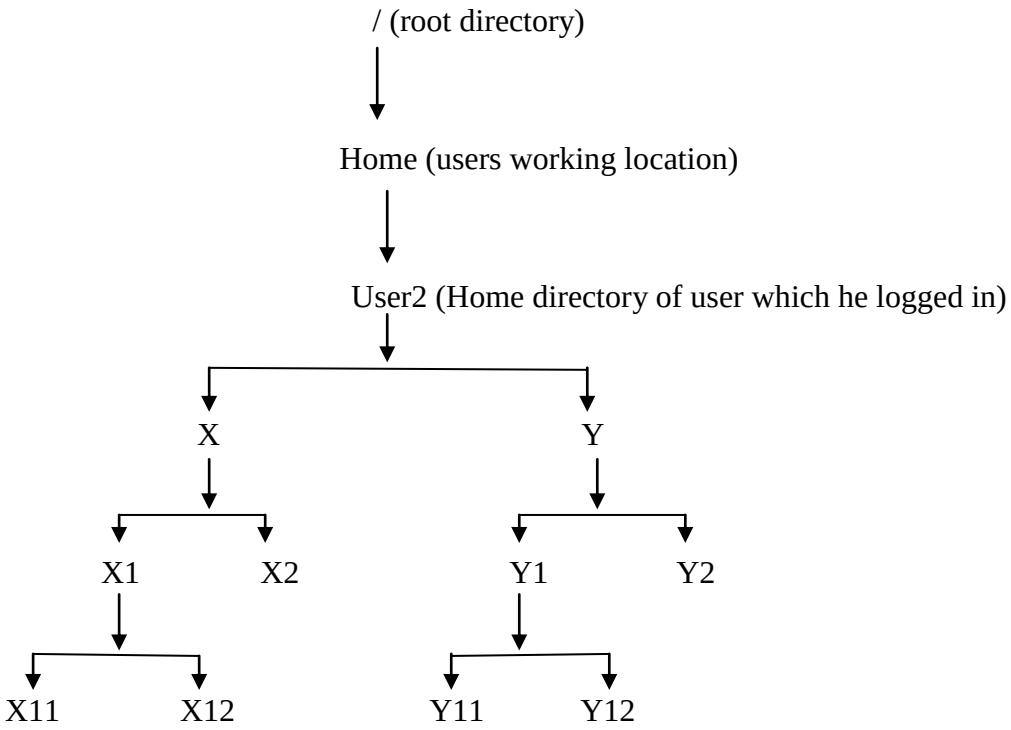
- **To change to Home Directory or User Login Directory**

  $ cd

- **To change to Root Directory**

  $ cd /

Ex: To create following Hierarchy



$ pwd

/home/user2

$ mkdir X

$ mkdir Y

To check whether the Directories created or not use ls command.

$ ls

X Y

Now Change into X directory to create X1 and X2 directories.

$ cd X

$ pwd

/home/user2/X

Creating multiple directories at once

$ mkdir X1 X2

$ ls

X1 X2

Here X is the parent directory of X1 and X2

$ pwd

/home/user2/X

$ cd X1

$ pwd

/home/user2/X/X1

$ mkdir X11 X12

$ ls

X11 X12

Here X1 is the parent directory of X11 and X12

Now move back to Home Directory

$ cd

$ pwd

/home/user2

$ cd Y

$ pwd

/home/user2/Y

$ mkdir Y1 Y2

$ ls

Y1 Y2

Here Y is the parent directory of Y1 and Y2

$ cd Y1

$ pwd

/home/user2/Y/Y1

$ mkdir Y11 Y12

$ ls

Y11 Y12

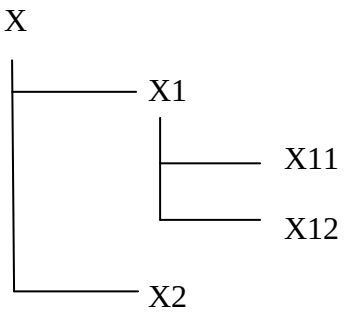
Here Y1 is the parent directory of Y11 and Y12

To display Tree Structure of a Directory

$ tree [dir name]

Ex:

$ tree X

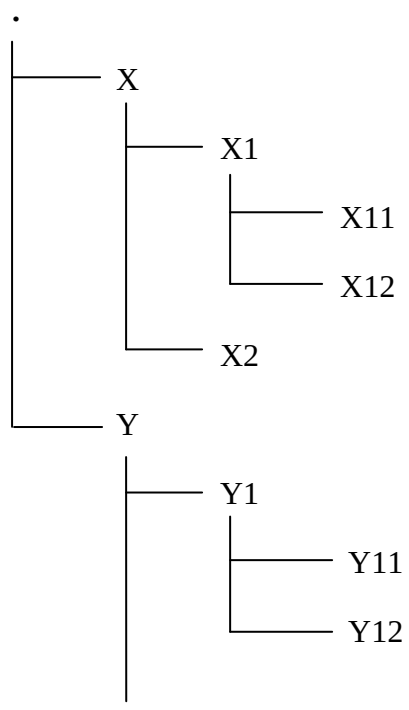


If you enter the Tree command without directory name it shows current working directory tree structure.

S pwd

/home/user2

$ tree



——— Y2

To go to Root Directory

$ cd/

$ pwd

/

To see the Tree structure of Root Directory line by line

$ tree | more

It will display the Tree Structure of Root directory line By line.

- **Renaming the Directory**

  $ mv [old name] [new name]

  $ mv X Z

- **Removing Directory**

  To Delete a Directory it should be Empty

  $ rmdir [dir name]

Ex: $ rmdir x12

To Delete a Directory including Sub Directories. It is a forceful deletion.

$ rm –R [dir name]

Ex: $ rm –R Y

Y dir along with its sub-directories is deleted.

- **Working with Absolute path and Relative path**

  Absolute Path:

  It's a path from Root to Target Directory (Destination)

Ex: $ pwd

/home/user2/X/X1/X11

Now we have to change into Y2 directory in a Y directory

$ cd [absolute path]

$ cd /home/user2/Y/Y2

$ pwd

/home/user2/Y/Y2

$ cd /home/user2/X/X1/X11

$ pwd

/home/user2/X/X1/X11

Relative Path:

It's a path from Current Directory to Target Directory.

Ex: $ pwd

/home/user2/X/X1/X11

Now change into Y2 Directory in Y Directory.

Syn: $ cd [Relative path]

$cd ../../../Y/Y2

$ pwd

/home/user2/Y/Y2

Ex: $ pwd

/home/user2/X/X1/X11

To create an Y21 directory in Y2 directory without moving from present working directory X11.

$ mkdir /home/user2/Y/Y2/Y21 → it's by using Absolute Path.

$ mkdir ../../../Y/Y2/Y21 → it's by using Relative Path.

$ pwd

/home/ user2/X/X1/X11

To check the contents of Y2 directory in present working directory.

$ pwd

/home/ user2/X/X1/X11

$ ls /home/user2/Y/Y2 → by using Absolute Path

$ ls ../../../Y/Y2 → by using Relative Path.

To create complete Hierarchy as above at a time using relative path

$ pwd

/home/user2

$ mkdir X Y X/X1 X/X2 X/X1/X11 X/X1/X12 Y/Y1 Y/Y2 Y/Y1/Y11 Y/Y1/Y12

To remove complete Hierarchy at a time using relative path

$ pwd

/home/user2

$ rmdir X/X1/X11 X/X1/X12 X/X1 X/X2 X Y/Y1/Y11 Y/Y1/Y12 Y/Y1 Y/Y2 Y

It is a reverse path of mkdir to remove complete hierarchy.

- **Moving Directories**

To move the Directory including sub Directories from Source to Destination.

Syn: $ mv [source path] [destination path]

Ex:

$ pwd

/home/user2

To move X1 directory including sub Directories into Y1 directory.

$ mv /home/user2/X/X1 /home/user2/Y/Y1 → Absolute Path

$ mv X/X1 Y/Y1 → Relative Path

- **Copying Directory contents from one location to another location including sub directories**

Syn: $ cp [option] [source path] [destination path]

Options:

-i:

Interactive. Prompt for confirmation whenever the copy would overwrite an existing file. A **y** in answer confirms that the copy should proceed. Any other answer prevents **cp** from overwriting the file.

-p:

Preserve. Duplicate not only the contents of the original file or directory, but also the modification time and permission modes.

-R:

Recursive. If any of the source files are directories, copy the directory along with its files (including any subdirectories and their files); the destination must be a directory.

cp refuses to copy a file onto itself.

Ex: $ pwd

/home/user2

To copy X1 directory into Y1.

$ cp –R /home/user2/X/X1 /home/user2/Y/Y1

(Or)

$ cp –R X/X1 Y/Y1

Note:

The difference between copy and move command is

After copying a file from source to destination the file is available at both source and destinations.

After moving a file from source to destination the file is available only at destination.

❖ Working with Files

- **Creating a File**

Cat command is used to create a file

Syn: $ cat > [file name]

Ex: $ cat >file1

bmnxbcmnxb
cjdbcnbdncbdmns

Enter the data you want and press (control key + d) to save & quit from file
The symbol '>' is used to enter the data into file using cat command.

With **cat** command we can only create a single data file. To create multiple files is not possible through cat command it is possible with **touch** command.

- **Creating multiple files**

Syn: $ touch [file1] [file2] [file3] [file4] …..

Touch command is used to create multiple empty files. The data should be entered later using vi editor.

- **Displaying file contents**

Cat command is also used to display the data in the file.

Syn: $ cat < [file name] or $ cat [file name]

Here the symbol '<' is optional.

Note: With **cat** command we can create file and display data in the file but we cannot modify the data of a file.

Ex: $ cat file1

bmnxbcmnxb
cjdbcnbdncbdmns

- **Displaying contents of multiple files**

With **cat** command multiple files data can be displayed.

Syn: $ cat [file1] [file2] [file3] ….

The data of the files will be displayed sequentially one after the other.

- **Copying files**

To copy the data of one file to another file **cp** command is used.

Syn: $ cp [source file] [destination file]

If the destination file already exist then data of source file overrides the destination file data. If does not exist it creates a new file.

Ex: $ cp file1 file2

If the destination file exist, to get confirmation from user to override the data or not **–i option** is used

Syn: $ cp –i [source file] [destination file] → to get confirmation to override.

If the file exist only it will ask for confirmation otherwise it create new file.

Ex: $ cp –i file1 file2

Overwrite y/n? –

The file is deleted only if option 'y' is entered, other than 'y' any char is entered the will not be deleted.

- **Rename a File**

To rename a file mv command is used.

Syn: $ mv [old name] [new name]

If the new name already existed the old name will be renamed to new name and new name data will be overridden by old name data.

Ex: $ mv file1 filex

Here also to get confirmation we can use **–i** option.

- **Removing a file**

To delete a file **rm** command is used.

Syn: $ rm [file name]

Ex: $ rm file1

- **Comparision of files**

To check the differences between the data of two files **cmp** command is used. But it displays only the first difference.

Syn: $ cmp file1 file2

To display all the differences between the files **diff** command is used.

Syn: $ diff file1 file2

Note: Comparison between the files of different users is possible only when the present working user has the access permission on the other user.

- **Removing multiple files**

To remove multiple files also **rm** command is used.

Syn: $ rm file1 file2 file3 file4 …..

Here all the files which are entered will be deleted. If we want **confirmation from user to delete the files –i option is used**

Syn: $ rm –i file1 file2 file3 file4 ….

Ex:

$ rm –i file1 file2 file3 file4

Remove file1 y/n? y

Remove file1 y/n? n

Remove file1 y/n?

Remove file1 y/n? x

Here only file1 is deleted. Other than option y if you type any character the file will not be deleted.

- **Knowing file types**

To know the type of the file, **file** command is used.

Syn: $ file [file name]

It displays the file type like exe, ascii, ZIP file etc.,

Ex: $ file file1

ASCII text

$ file file.zip

ZIP archive

- **Search for a file**

To search for a file there are two types of commands. They are **locate** and **find**.

Locate command

By using locate command we can search for the file in the entire system (OS).

Syn: $ locate [file name] → in whole filing system

Ex: $ locate file1

/home/user1/x/file1

/home/user2/file1

/home/file1

/file1

Find command

By using find command we can search only in present working directory. To search in other locations by using **–name option** we have to specify the path it has to search in.

Syn: $ find [file name] → only search in the present working directory

$ find –name [file path] → to search for a file in required location

Ex: $ find –name /file1 → to search for file1 in root directory.

$ find –name /home/file3 → to search for file3 in home directory

- **wc**

To count number of lines, words, chars in a file **wc command** is used. By using this command multiple files data can also be counted.

Syn: $ wc [file name]

Ex: $ wc file1

3 20 50 file1 → here 3 lines 20 words and 50 characters

$ wc file1 file2 file3 file4

3 20 50 file1

2 10 20 file2

5 40 30 file3

6 50 100 file4

Options:

-l → To display only lines

-w → To display only words

-c → To display only chars

-lw → To display lines and words

-lc → To display lines and chars

-wc → To display words and chars

- **od**

od – octal dump. It shows the binary format of file.

Options:

-b → to display ascii values of characters in a file

-bc → to display ascii values of characters along with characters.

Ex: $ od -b file1

00060 163 164 158 193

$ od –bc file1

00060 163 164 158 193

a z d c f

- **Compressing File**

Compacts a file so that it is smaller. When compressing a file it will be replaced with a file with the extension .gz, while keeping all the same ownership modes.

gzip is command to compress a file. gzip is any of several software applications used for file compression and decompression. The term usually refers to the GNU Project's implementation, "gzip" standing for GNU zip. It is based on the DEFLATE algorithm, which is a combination of Lempel-Ziv (LZ77) and Huffman coding.

Syn: $ gzip [filename]

After compressing file into zip file then file name will be changed to filename.zip and the original file will not be available.

Ex: $ gzip file1

Result is file1.gz and the original file file1 will not be available.

- **Uncompressing File**

To uncompress the zip file gunzip command is used.

Syn: $ gunzip [filename.gz]

Ex: $ gunzip file1.gz

Result is file1

- **Working with archival file**

Archival file is a pack which contains hierarchy file system.

  - **Creating archival file**

“tar” is command used to create archival file.

Syn: $ tar –cvf [user defined name.tar] [dir name]

Here

c → create new archival file

v → VERBOSE is security protocol

f → specified files

Ex:

$ tar –cvf myarchive.tar . → . refer to all files in current working directory

$ tar -cvf myarchive1.tar x → all files in x directory

$ tar –cvf myarchive2.tar file1 file2 file3 file4 → files to be archived

- **Extracting archive file**

To extract the archive file tar command with option –xvf is used.

Syn: $ tar –xvf [archived file name]

Ex: $ tar –xvf myarchive.tar

Here x stand for extract archive file

- **Creating archive file and compressing**

We can create archive a file and compress it. For doing both actions we use **tar command** with **–czvf option.**

Syn: $ tar –czvf [userdfined name.tar.gz] [directory name]

Ex: $ tar –czvf myarchive.tar.gz . → it will archive and compress the present working directory file system and name as myarchive.tar.gz

$ tar _czvf myarchive1.tar.gz x → x directory file system will be archived and compressed.

- **Uncompressing and Extracting archive file**

By using **tar** command with **–xzvf option** we can uncompress and extract the archive file.

Syn: $ tar –xzvf [archive file name.tar.gz]

Ex: $ tar –xzvf myarchive1.tar.gz

Here archive file is uncompressed first and then Extracted. The result is original file system.

❖ Link Files

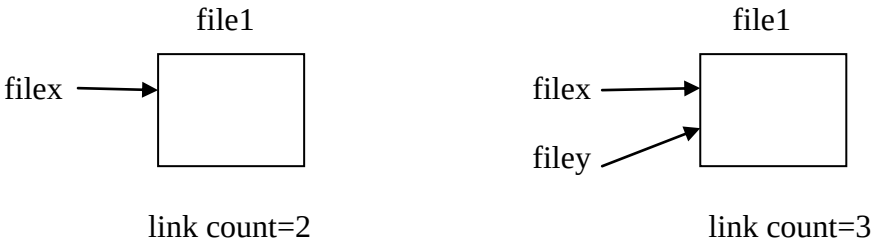
There are two types of link files

1. Hard Link
2. soft Link

1. Hard Link

It is a link file to another file. When the original file is deleted in Hard link, even then we can see the data in link files.

Whenever link file is created on original file then link counters increases.



To create hard link to file1 by using a link file filex **ln command** is used

$ ln file1 filex

$ ls –l

--------2---------file1
--------2---------filex
Here 2 is the link counter number

$ ln file1 filey

$ ls –l

----------3--------file1
----------3--------filex
----------3--------filey

Here 3 is the link counter number

$ rm file1

Even original file is deleted data will not be deleted. data will be remained in link files. Whenever a link file or original file is deleted just link counter decreases. When link counter reaches to 1 the data will be deleted.

If any changes made in original file data that affects the link file data and vice versa. since all link files refer to the same data.

2. Soft Links

Soft link is also a link to the original file. but when the original file is deleted link file will not work.

To create a soft link on a file use "-s" option with **ln command**

$ ln –s file1 filex

$ ls –l

-l----------1----------file1
-l----------1----------filex → file1
Here l indicates symbolic link file

$ ln –s file1 filey

----------1----------file1
-l ---------1----------filex → file1
-l ---------1----------filey → file1

In soft link there will be no increment of link counter number.

when the file1 is deleted other link files on file1 will not work.

❖ FTP (File Transfer Protocol)

File Transfer Protocol (FTP) is a network protocol used to copy a file from one computer to another over the Internet or LAN. FTP follows a client-server architecture which utilizes separate control and data connections between the ftp client and server.

To connect to FTP server enter **ftp command** in user mode. Now $ prompt changes to ftp prompt there open the server connection by using ip address and password.

Syn: $ ftp

ftp > open 192.14.35.76

Password: ********

To close the server connections **close command** is used.

ftp> close

237 goodbye

To come out from ftp to $ prompt **bye command** is used.

ftp> bye

$-

Commands used in ftp

ftp> verbose → Verbose mode gets off.

ftp> pwd → To display server's current working directory

ftp> lcd → To display client's current working directory.

ftp> ls → To display server's directory contents.

ftp>dir → To display server's directory contents.

ftp> mkdir [dir name] → To create a directory on server.

ftp> cd [dir name] → To change a directory on server.

ftp> rmdir [dir name] → To delete a directory from server.

ftp> delete [file name] → To delete file on server.

ftp> mdelte file1 file2.. → To delete multiple files on server.

ftp> binary → To set the transfer mode as binary

ftp> put [filename] → To upload the file

ftp> mput file1 file2.. → To upload multiple files

ftp> mput *.cpp → To upload all the files with extension .cpp

ftp> get [file name] → To download the file

ftp> mget *.cpp → To download the multiple files with extension .cpp.

ftp> disconnect → To disconnect from server.

ftp> quit → To quit from ftp prompt

ftp> bye → disconnecting from server and quit from ftp prompt.

ncftp → Anonymous ftp

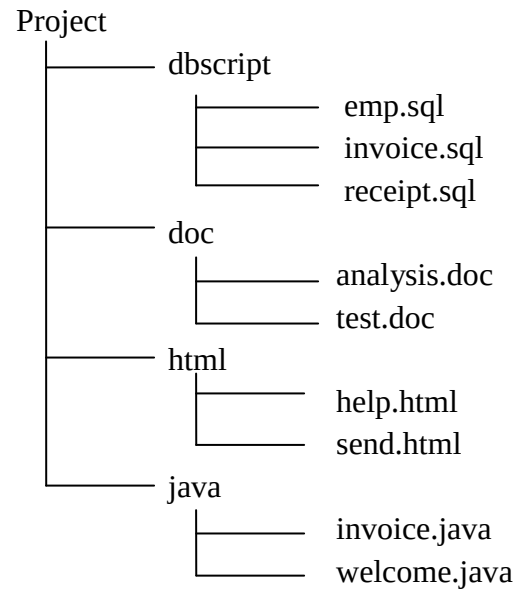
$ ncftp 192.65.78.90 → To connect to the ftp site without UID and PWD.

Ex: Uploading and downloading project builts in the FTP server.

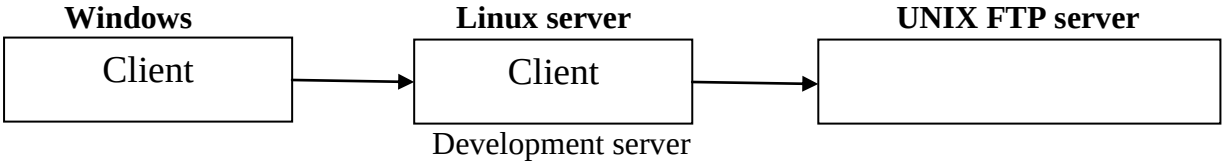
Requirement:

There is a directory named project in one of the system connected to ftp server. By using FTP protocol upload the project into server. From the server another client who is also connected to the server will download the project.

$ tree project



Procedure for uploading project to server:



Step 1: pack the above project

$ tar –cvf javaproject.tar project

$ ls

Javaproject.tar

Step 2: compress the pack using zip

$ gzip javaproject.tar

$ ls

Javaproject.tar.gz

Step 3: connect to FTP server

$ ftp

ftp> open 102.45.23.11

Connected to 192.45.23.11

Name : admin

Password: admin

Displays a message 340 login successfully (here 340 is ftp number)

ftp> pwd

253 "/var/www/html/upload" → it is server present working directory also called as configured directory

If you want to find the local current working directory i.e., client directory.

ftp> lcd

254 "/home/user1"

Now create a directory in server

ftp> mkdir javaproj

257 "/var/www/html/upload/javaproj" created

Change into the directory

ftp> cd javaproj

259 directory successfully changed.

Now set file transfer mode as binary.

ftp> bin

200 switching to binary mode

Provide security by making verbose mode off

ftp> verbose

verbose mode off

ftp> hash

hash mark prints # mark for 1024 bytes of data transfer.

To go to local directory from the server temporarily

ftp> !

$

To return back to ftp enter exit

$ exit

ftp>

now upload the javaproject.tar.gz to server by using put command

ftp> put javaproject.tar.gz

(# symbol indicates successful transter of file)

Check the contents of server

ftp> ls

javaproject.tar.gz

To disconnect from server

ftp> bye

$

Procedure to download project from server

Start FTP server as above

Now check present working directory

ftp> pwd

243 “/var/www/html/upload”

ftp> ls

ftp> cd javaproj

ftp> ls

javaproject.tar.gz

set file transfer mode as binary

ftp> bin

set security mode

ftp> verbose

ftp> hash

ftp> get javaproject.tar.gz

#

ftp> bye

$

After downloading unzip and extract the project

$ gunzip javaproject.tar.gz

$ ls

javaproject.tar

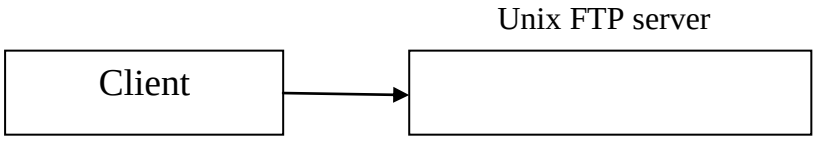
$ tar –xvf javaproject.tar

$ ls

Project

In above process the project uploading and downloading is done from client to Unix FTP server through Linux server which is a development server as another client.

We can also upload and download the project directly from windows as client to UNIX server without Linux server.



Procedure

Go to command prompt in windows

C:\ ftp

ftp> user name: admin

Password: admin

Repeat same as above process to upload and download project

Here when we provide hash function for every 2kb of data transfer # mark will be printed.

❖ Working vi editor

vi editor is used to modify the file contents.

Vi editor → visual editor

Opening file

$ vi [file name]

If the file is not found the editor creates a new file.

To modify the file contents vi editor provides three types of modes

1. Command mode
2. input mode
3. exit mode

1. Command mode

It is default mode. In this mode we can execute the commands of vi editor. If you are in another mode in order to get into command mode press “Esc” key.

Command to modify the file contents

A → To append the text at the end of the line.

I → To insert the text at the beginning of the line.

a → To append the text to the right of the cursor position.

i → To insert the text to the left of the cursor position.

Cursor Navigations

l → To move right

h → To move left

j → To move down

k → To move up

0 (zero) → Works like home key

$ → Works like end key.

b → goes to beginning of previous word

4b → moves four words backward

w → goes to beginning of next word

5w → five words forward

e → goes to end of the word

3e → goes to end of third word

Ctrl + f → go to next page (page down)

Ctrl + d → go to previous page (page up)

gg → to move beginning of the file

G → end of the file

LINUX supports the following functional keys.

Arrow keys like left, right, up and down.

Home, End, Page Up, Page Down keys.

But UNIX does not support the functional keys. In LINUX work as we work in notepad.

Commands to deletion of text

dd → To delete a line

4dd → To delete 4 lines

cc → to clear the text in a line with out deleting a line so we can enter the data in that line.

dw → to delete a word

3dw → to delete 3 words

x → to delete a character

5x → to delete 5 characters

u → Undo the transaction

Opening Lines

O → to open a line above the cursor position

o → to open a line above the cursor position

Copying or Yanking lines

yy → to copy a line

4yy → to copy 4 lines

p → paste a text

yy and p → copy and paste

dd and p → cut and paste

2. Input mode

In this mode user can modify the file contents. In this mode if user tries to execute any command, it won't treat like command it is treated like text. To treat as command user has to go to command mode by pressing "Esc" key

In order get into input mode press "i" key

3. Exit mode

In this mode user can quit from the editor. If you are in input mode then to go to exit mode first you have to go to command mode then to go to exit mode is possible.

: → To get into exit mode as showed at the bottom of the editor.

:w → To save the file

:q → To quit from the editor

:wq → To save and quit

:q! → To quit without saving the file

:set nu → To set the line numbers

:set nonu → To remove the line numbers in vi editor

:! [unix command] → To execute unix command

Ex: :!date → Execute date command

After execution of command press enter key to come back to vi editor and do work.

:w 10 → To move to 10^{th} line

:w [file name] → save as some file

❖ Compilation of Programs

- **Compiling C Program**

$ vi hello.c

#include<stdio.h>

Main()

{

printf ("\n hello world")

printf("\n hello")

}

Go to command mode and then go to exit mode, save and quit.

Now compile to the program cc is the command with –c option

$ cc –c hello.c

Result file is hello.o (object file)

An object file is generated after compilation. Now generate a exe file from object file. In LINUX there is no extension for exe file it was just shown in green color.

Creating exe file

To create a exe file cc is the command with –o option. Here we have to enter user defined name for exe file.

$ cc –o myhello hello.o

Result is myhello exe file

Executing

$ ./myhello

Hello world

Hello

Here ‘ .’ (dot) refers to current working directory

If you want to execute the exe files with out using path then go to .bash_profile in home directory and set the path to current working directory as below

Go to user login directory (home directory)

$ cd

$ pwd

/home/user1

$ vi .bash_profile

PATH=$PATH:./

EXPORT PATH

:wq

Assign the permissions to .bash_profile

$ chmod 777 .bash_profile

Now, relogin again and execute the exe file

When ever a new user created this .bash_profile will be created automatically in Linux OS. For Unix OS, the file will be only ‘.profile’

Execution of exe file with out ./

$ myhello

Hello world

Hello

- **Compiling C++ Program**

'g++' is a command to compile the C++ program with option '–c'

$ g++ -c hello.cpp

Result file is hello.o (Object file)

Creation of exe file

To create an exe file from object file 'g++' command with option '-o' is used

$ g++ -o myhello1 hello.o

Result file is myhello1 exe file

Executing

$ myhello1

Already we have set the path as current working directory in above. So, the exe file will be executed without path. If not, set the path as above.

- **Compiling Java Program**

' javac' is the command to compile java program

$ javac Test.java

Result file is Test.class

When we compile a java program it will generate '.class' file.

Executing Java Program

$ java Test

Setting PATH and CLASSPATH for Java

Install java in '/usr/local' directory and then open the .bash_profile file in the home directory

$ pwd
/home/user1

$ vi .bash_profile

PATH=/bin/usr/local/j2sdk1.5/bin:./:$PATH

CLASSPATH=/usr/local/j2sdk1.5/lib/tools.jar:.:%CLASSPATH%

Export PATH CLASSPATH

:wq (Save and Quit)

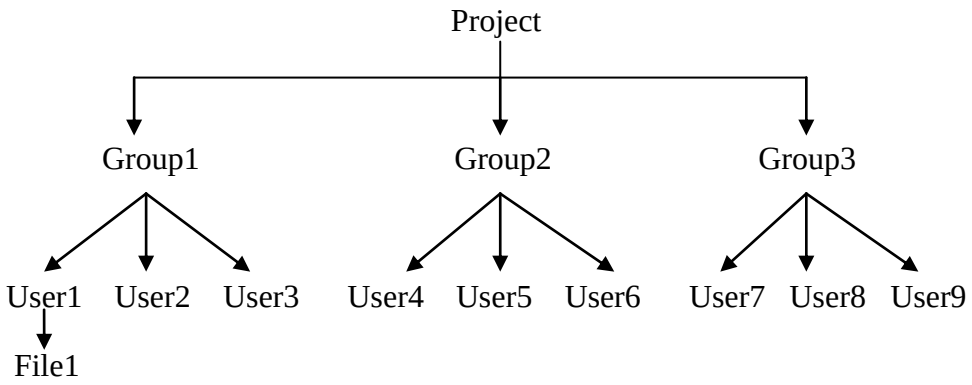
Now, logout and login again

In linux, no need to install java. Java is in-built in Linux Operating System.

❖ Working with File Permissions

In UNIX/LINUX groups and users are created. Users are assigned to a group in the time of creation of users. If not assigned the Default group name of the file is user group.

For every directory and files in file system has permissions to read, write and execute for users, group and others. In Default some permission are assigned to users, group and others at the time of creation of directory and files. As an owner of the file the permissions can be changed by user and system admin can also change the permissions.



Here for file1 the user1 will be the owner. Group will be the group1. The rest of groups (group2, group3) and users of those groups comes under Others.

$ ls –l file1

-rwxrw-rw- 3 user2 group1 6451 dec 12:00 file1

Here, rwxrw-rw- are file permissions of file1 to the user, group and others

Here, user has read, write, and execute permissions

Groups have only read and write permissions. Others have also read and write permissions

Changing File permissions

‘chmod’ is a command to change the file permission

To change the file permissions of a file current user must be owner of the file or system admin can change the permissions.

Syntax: $ chmod [category] [operation] [permission] [filename]

Category	Operation	Permission
u → User	+ → Add	r → Read
g → Group	- → Remove	w → Write
o → Others		x → Execute

Ex 1:

$ ls –l file2
-rw-r-xr-- 3 user1 group1 7421 dec 11:30 file2

Requirement:

User → adding "x"
Group → removing "x"
Others → adding "w"

$ chmod u+x, g-x, o+w file2

$ ls –l file2
-rwxr--rw- 3 user1 group1 7421 dec 12:30 file2

Ex 2:

$ ls –l file3
-r--rwx rwx 3 user1 group1 9422 dec 01:30 file3

Requirement:

User → adding "wx"

Group → removing "wx"

Others → removing "rw"

$ chmod u+wx, g-wx, o-rw file3

$ ls –l file3
-rwx r-- --x 3 user1 group1 6492 dec 01:30 file2

Ex 3:

$ ls –l file4

-r-xrw- -wx 3 user1 group1 6492 dec 02:30 file3

Requirement:

User → adding "w" and removing "x"

Group → removing "w" and adding "x"

Others → adding "r" and removing "x"

As above it is not possible to perform two operations at a time

We need to execute two commands.

$ chmod u+w, g-w, o+r file4

$ chmod u-x, g+x, o-x file4

Alternative to these two commands is

$ chmod u+w-x, g-w+x, o+r-x file4
(Or)
$ chmod u+w, u-x, g-w, g+x, o+r, o-x file4

$ ls –l file4

-rw-r-x rw- 3 user1 group1 7421 dec 02:30 file2

Using Octal Notations to change file permissions

Below are the octal notations for giving permissions

Read → 4
Write → 2
Execute → 1

All → 7

For combination of permissions below octal notations are used

Read and Write → 6
Read and execute → 5
Write and Execute → 3

Ex:

$ chmod 645 file1

In the above command

To the user → 6 (read and write)
Group → 4 (read)
Others → 5 (read and execute)

These permissions are given to all. These are latest permissions.
The previous permissions of the file are removed and latest permissions will be added.

Ex 4:
$ ls –l file2
- r-x rw- -wx 3 user2 group2 5432 dec 09:00 file2

Requirement:

User → adding "w" and removing "x"
Group → removing "w" and adding "x"
Others → adding "r" and removing "x"

$ chmod 656 file2

$ ls –l file2
- rw- r-x rw- 3 user2 group2 5432 dec 10:00 file2

$ chmod 777 [filename] → all permissions to all

$ chmod –R 777 [dir name] → changing permissions of a directory including sub-directory contents.

$ chmod 65 file1 → $ chmod 065 file1

$ chmod 5 file1 → $ chmod 005 file1

$ chmod 0 file1 → No permissions to all.

$ chmod file1 → syntax error

Permissions:

- When there is **Write permission** to the user who is the owner of the file, then he can forcefully overwrite the file using **'!'** symbol.

: wq! The file will be forcefully overridden and saved and quit.

- **Executable permissions** are only applicable for directories, executable files and shell scripts.
- When there is an executable permission for a file, it is shown in green color.

$ ls –l x

Shows the contents of x directory and their permissions.

$ ls –l

Shows the permissions of directories.

Directory permissions are:

Read: with this permission we can see the contents of the directory.

Write: with this permission we can create or delete files in the directory.

Execute: with this permission user can change into that directory. Without this permission user cannot change into that directory.

Note: To modify a file you should have both read and write permissions.

umask: Default file and directory permissions

To assign all the permissions to a regular file the notation is 666. Since a regular file will not have executable permission.

To assign all the permissions to a directory the notation is 777.

Initial permissions of the files and directories depends upon current umask value (when a file is created newly the default permissions are given depending on umask value)

To check the umask value, **umask command** is used.

$ umask

0022

Here first zero is the sticky bit value; second zero for users; 2 for group; another 2 for others

Umask value tells that not to grant specified permissions.

For example as above umask value is 022 it means

The default permissions of a file is 644 (666-022=644) and for directories 755 (777-022=755)

After that permissions of files and directories can be changed by using **chmod command.**

To change umask value

$ umask 000 → by this mask value all the permissions will be assigned to files and directories.

Only logged users can change the umask value, once the user logged out from user session the changes made in umask value will be erased and default umask value will be stored again i.e., the changed umask value is applicable for only current session.

To change the default umask value permanently, go to .bash_profile file

$ vi .bash_profile

Umask 033

:wq

From here at the time of login of this user the default umask value is 033

❖ Working with Shell

UNIX provides three types of shells.

1. **Bourne shell** → It is developed by Steve Bourne

In Bourne shell there are two types of shells

i) bsh → it is older version
ii) bash → it is latest version shell and default shell and its full form is Bourne again shell.

2. **K-shell**

It provides only one shell called ksh

3. **C-shell**

It also provides only one shell called csh

Default shell is bourne shell and it's a mostly used shell.

To change shell

$ ksh → to chage to ksh from existing shell.

Note: with ksh ctrl+l (clear screen) will not work.

ksh commands for clearing screen is

$ clear → to clear screen

$ logout → to logout of the shell.

$ csh → to change to c shell

csh is somewhat similar to bash shell.

$ bsh → to change to bourne shell

Now bsh shell was not used. No command will work in bsh shell. To come out of bsh shell press ctrl + l

$ bash → to change to bash shell

Bash shell is the latest shell. It is mostly used in real time.

Bourne shell will provide the commands, keywords, control statements and operators to develop the shell script.

❖ Shell commands

There are two kinds of commands that come with a Linux / UNIX operating system: Shell Commands and Linux/Unix Commands.

Linux/Unix Commands

These are located in special directories for binary files, such as /user/bin. Directories that contain these Linux / UNIX commands are listed in the search path, which the shells use to find them.

Shell Commands

Shell Commands are part of a shell. In other words, each shell (e.g., C Shell, Bourne Shell and Korn Shell) has a set of commands built into its program. Though shell commands may vary from one shell to another, the commands within each shell stay the same across Linux / UNIX distributions and variants.

Some of shell commands are echo and printf

Ex: $ echo “Hello world”
Hello world

Ex: $ printf Hello world
Hello $-

Printf command read the data up to space or enter key. Printf is provided by only LINUX not UNIX.

Printf will read multiple words only if we use double quotations. For echo no need of quotations to read multiple words

Ex: $ printf “Hello world”
Hello world $-

Ex: $ echo Hello world
Hello world

Echo command; insert the new line at the end of data.

Printf command will not insert the new line at the end of the data. To insert the new line provide \n at the end of the data.

$ printf "Hello world \n"
Hello world
$-

❖ Standard files of UNIX

There are 3 types of standard files in UNIX.

1. Std input file

2. Std output file

3. Std error file

Std input file → the file (or stream) representing input, which is connected to keyboard.

Std output file → the file (or stream) representing output, which is connected to display.

Std error file → the file (or stream) representing error message, which is connected to display.

Standard Input file

Keyboard, the default source

Ex: $ wc
Hello world
How are you
^d
2 6 20

Here in above example data is taken from keyboard & displaying output. Here default source is keyboard and destination is screen.

With the help of left redirection symbol '<' we can assign the files as sources.

Ex: $ wc < file1
5 15 45 file1

Here file1 is the source. '<' symbol works as redirection to the left.
Here wc counts the file1 contents and displays output on screen.

Ex: $ cat < file1
Display the contents of the file1.

Standard output file

Default std output file is screen.

$ cat file1
Display the contents of the file1.

Here default output is terminal.

$ date

Current date is displayed on screen.

Here also with the help of right redirection operator '>' we can store the output in a file

Ex: $ date > filex

No output will be displayed the date will be stored in filex. Filex will be the output file.

$ cal > filex

If the file already exists it silently overrides the data. If not exist create a new file.

To append the data into file use **'>>' symbol.**

Standard error file

The default standard error file is screen.

Ex: $ cat filey

Cat: cannot open filey

Above is the error displayed as cat cannot open the file which is not existed.

Ex: $ cat filey > filex

$ cat filex

There will be no output in filex since using '>' we cannot send the error output to file

To redirect error message to file

Ex: $ cat filey 2> filex → using 2> we redirect error message to file

$ cat x

Cat: cannot open filey

Here

- **0 → used for standard input**
- **1 → used for standard output**
- **2 → used for standard error**

Terminal file

Ex: $ cat < file1 >/dev/tty

Writing ouput to terminal

$ tty

/dev/pts/3 (pts/3 is the name given to the current login)

We can send messages to other terminals as above

Ex: $ date > /dev/pts/4

Here the date has been send to the pts/4 terminal user

Wall is also a command to broad cast message to all users from current working user.

Shell variables

$ set → to display all environment variables

Command History

.bash_history file maintains the commands history

$! 1 → first command

$! 5 → fifth command

$ pwd
/home/user1

To change the home directory location to x

Ex: $ HOME = /home/user1/x

$ pwd
/home/user1/x

We can define our own variables as below

$ a=10

This a=10 will be stored in environment variable

$ c=a

The value of c is stored as only ‘a’ but not 10 in environment variables.

$ b=20

In shell there are no data types everything is a character

Reading data from variable memory

‘$’ is an operator to read the value from variable memory.

Ex: $ echo a=$a and b=$b
a=10 and b=20

Ex: $ echo $a $b
10 20

$ a=50

$ b=60

Any changes made on the variable that will affect on the variable location also

Ex: $ echo a=$a and b=$b
a=50 and b=60

String Variables

We can store string variables as below

Ex: $ name=Hyderabad

$ echo my location= $name
My location=Hyderabad

Ex: $ name=hello world

$ echo my name=$name
My name=hello

It will not take multiple words into consideration with double quotes. Since ‘=’ operator will read the data upto space.

Ex: $ name=”hello world”

$ echo my name=$name
My name=hello world

Ex: $ path=/home/user2/x

$ cd $path

$ pwd
/home/user2/x

Ex: $ cmd=cal

$ $cmd
Cal will be displayed.

These variables are local to the current user session. To make these variables available to all users' sessions, export the variable into global memory space.

Syn: export variable name

Ex:

$ PATH=/home/user2/local/j2sdk 1.5/bin:.:$PATH

$ CLASSPATH=/home/user2/local/j2sdk 1.5/lib/tools.jar :.:$CLASSPATH

$ export PATH

$ export CLASSPATH

Command substitution into the text

Command quotations " **`cmd`** " are used to substitute command into the text.

Quotation key is available below the "Esc" key in the keyboard.

Ex: $ echo present login user name is `logname`
Present login name is user1

Ex: $ echo today date is `date % + d`
Today date is 13/12/11

Ex: $ echo my present working directory is `pwd`

my present working directory is /home/user2

Change the $ prompt

To change the dollar $ prompt and to assign user defined prompt

$ ps1="user defined"

❖ Filters

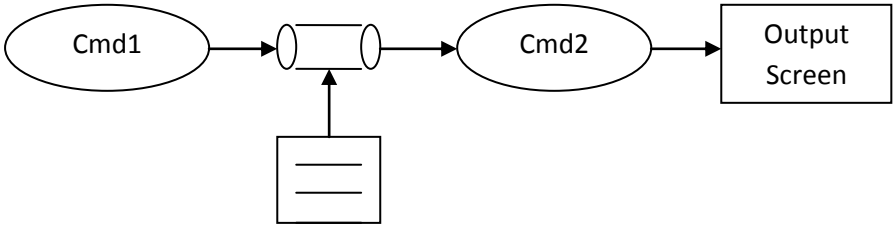
UNIX/LINUX provides following filters

Pipe, head, tail, tr, tee, grep commands

Working with pipe

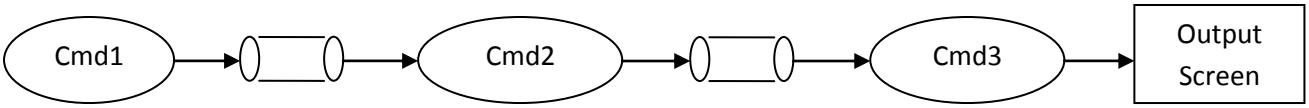
Pipe is used to communicate between two process called as "inter process communication".

$ cmd1 | cmd2



The result of cmd1 will be return to pipe temporary file. From this file cmd2 will read data and send to output screen.

Note: multiple pipes can be used.



Ex: $ ls –l | wc –l
148

Ex: $ ls –l | wc –l | wc –l
1

Ex: $ ls –l | wc –l | wc –l | wc –l
1

The result of ls –l will be return to pipe from there the output data is counted by wc and is return to pipe from there the output data is counted and result is displayed on output screen.

Ex: $ echo my file has `cat<file1 | wc –l` lines
My file has 45 lines

Ex: $ my present working directory contains `ls –l | wc –l` files
My present working directory contains 34 files

Ex: $ the number of present working users are `who | wc –l`
The numbers of present working users are 5

Head command: Display beginning lines

Syn: $ head –n [file name] → to display first n lines in a file

Ex: $ head –4 file1 → it displays first 4 lines of file1

Tail command: Display ending lines

Syn: $ tail –n [file name] → to display last n lines in a file

$ tail +n [file name] → to display from nth line to end of the file

Note: "+n" option works only with UNIX not LINUX

Ex: $ tail –4 file1 → it displays last 4 lines of file1

$ tail +4 file1 → it displays from 4 line to end of the file1

Using head and Tail commands

Ex:

$ cat filex

1 Bfjhdbjh
2 Bhjdbf
3 Ksdjnjkcvhkdsj

----
10 Hsvbcsfjkshdsjkfh

In above filex there is 10 lines of data.

To get the data from 4th line to 8th line

$ tail +4 filex | head -5
From 4th line to 8th line the data is displayed from filex.

(or)

$ head -8 filex | tail -5

With the head command first 8 lines are retrived from there using tail command last 5 lines i.e, from 4th line to 8th line of data is displayed from filex.

Use case of head and tail

Head and Tail commands are used to split the data files into number of files (multiple files).

Requirement

Emp.dat file contains 10 lakhs of records. It should split into 10 files. Each file should contain 1 lakh records. This is an real time requirement.

$ head -100000 emp.dat > emp1.dat

First 1 lakh records stored in emp1.dat

$ head -200000 emp.dat | tail -100000 > emp2.dat

Next 1 lakh records stored in emp2.dat

$ head -300000 emp.dat | tail -100000 > emp3.dat

Next 1 lakh records stored in emp3.dat

$ tail -100000 emp.dat > emp10.dat

Last 1 lakh records stored in emp10.dat

Translation command (tr)

This 'tr' command is used to replace the character.

Syn: $ tr "old char" "new char" < [file name]

tr command replace the "old char" with "new char". It will display the replaced output but it will not shoe any affect on the original file.

Ex: $ tr "a" "z" < file1

Here in file1 data the char "a" is replaced with char "z" and displayed on output screen.

To redirect the replaced output to another file use redirection operator. So we can store the replaced output

Ex: $ tr "a" "z" < file1 > filex → To redirect the replaced output to filex

Silently redirects the replaced output to filex

"tee" command

It works along with pipe symbol. It acts as right redirection operator '>' but it displays the output on the screen as well as send to output file.

Ex:

$ cat < file1 | tee filex

$ date | tee filex

$ ls –l | tee filex

$ who | tee filex

$ tr “a” “z” < file1 | tee filex

Grep command: pattern searching

Grep command is used to search a particular pattern from the files

Syn:

$ grep “searching pattern” [file name]

Grep command displays the lines in which searching pattern is found.

$ grep –n “searching pattern” [file name]

Displays the lines along with line numbers in which searching pattern is found.

$ grep –v “searching pattern” [file name]

Displays the lines in which searching pattern is not found i.e., expect the lines containing the pattern will be displayed.

$ grep –vn “searching pattern” [file name]

Displays the lines along with line numbers in which searching pattern is not found

$ grep –c “searching pattern” [file name]

It displays the number of lines on which pattern is found

$ grep –i “searching pattern” [file name]

Grep command is case sensitive so to ignore the case sensitive **‘–i’** option is used.

$ grep –e “searching pattern” –e “searching pattern” [file name]

To search for multiple patterns

$ grep “pattern” [file1] [file2] [file3]…..

Searching of pattern from multiple files

$ grep “pattern” *

To search pattern from all files in current working directory

Ex:

$ grep “hello” file1

---------hello----------
----------hello---------

$ grep –n “hello” file1

1:---------hello----------
3:----------hello---------

$ grep –v “hello” file1

Except the lines containing “hello” are displayed

$ grep –vn “hello” file1

2:---
4:---

$ grep –c “hello” file1

2

$ grep –i“hello” file1

---------hello----------
----------hello---------
--------HELLO----------

$ grep –e “hello” –e “hai” filex

---------hello----------
----------hai----------
----------hello----------
---------hai----------

$ grep –ie “hello” –e “hai” filex

---------hello----------
----------hai----------
----------hello----------
-----HELLO----------
---------hai----------

$ grep “hello” file1 file2 file3

File1:---------hello----------
File1:----------------hai----------
File2:-------------hello------------
File3:---------hai-----------------

$ grep “hello” *.dat

To search for hello pattern in all .dat files

Use case for grep command

Grep is used to segregate (categorization) of data

Requirement

To get all sales dept records and store into one file

$ grep “sales” emp.dat > sales.dat

To get clerks information working under sales dept

$ grep “sales” emp.dat | grep “clerk” > clerk.dat

❖ Advanced Filters

Sort, cut, paste, uniq are the advanced filters

- **Sort**

Sort command is used to sort the data files based on particular fields. Default sort is based on first field and will be in ascending order.

Syn: sort [file name]

Sort based on particular fields:

$ sort –t “:” –k 1, 2, 3… [Data file name]

Here “t” indicates field terminator. “k” indicates key (field number)

Sort based on 2nd field

$ sort –t “:” –k 2 [Data file name]

Sort in Descending order (reverse order) based on 2nd field

$ sort –rt “:” –k 2 [Data file name]

Ex:

To retrieve the record of Employee getting max Sal. Here empno is 1st field and sal is 3rd field

$ sort –rt ":" –k 3 emp.dat | head -1

Sort based on multiple fields (1st & 3rd field)

$ sort –t ":" –k 3, 3 –k 1 emp.dat

start at stop at

In Default when any two values are equal in the sorting fields then it goes for next field for comparison. So to stop the comparison at that field above type of syntax is used.

- **Cut**
  Cut command is used to get the particular fields of data file.

Syn: $ cut –d ":" –f 1, 2, 3... [Data file name]

Here "d" indicates delimiter (or) separator, f indicates field number.

Ex:

To get ename which is 2nd field and age of employee which is 6th field

$ cut –d ":" –f 2,6 emp.dat

aaa:25
bbb:32

To get Max sal

$ sort –rt ":" –k 3 emp.dat | head -1 | cut –d ":" –f 3

60000

Write a script to get all max Sal records

$ vi maxsal.sh

maxsal=`sort –rt ":" –k 3 emp.dat | head -1 | cut –d ":" –f 3`
grep $maxsal emp.dat
-
-
:wq

$ sh maxsal.sh

1002: aaaa: 9800: edp: director: 34

3214: bbbb: 9800: psy: manager: 32

- **Paste**

To paste the data of two files side by side “paste” command is used

Syn: $ paste [file1] [file2]

To change order of the fields:

To get following order

step1: get 1, 3, 5 fields and store into one file

$ cut –d “:” –f 1, 3, 5 emp.dat > emplist1.dat

step2: get 2, 4, 6 fields and store into another file

$ cut –d “:” –f 2, 4, 6 emp.dat > emplist2.dat

step3: use paste command to paste the files side by side

$ paste emplist1.dat emplist2.dat>resultemp.dat

In this case space is provided between the fields of two files

$ paste emplist1.dat:emplist2.dat>resultemp.dat

In this case “:” is provided between the fields of two files

Now you can check using “cat” command.

Uniq command

$ uniq [file name]

It is used to filter the duplicate lines in the file. For every duplicate line it will read nly one line

❖ Writing shell scripting

$ vi addr.sh → .sh extension is optional

With extension the keywords and data is highlighted in different colors.Without extension all the keywords will be as simple text. With extension we can identify the shell scripting files.

$ vi addr

```
echo managing director

echo xxx training institute

echo ameerpet

echo Hyderabad

-

-

:wq
```

Executing shell program

$ sh addr.sh
(Or)
$ sh addr

Shell is a command interpreter; it interprets the script line by line. If there is an error in any line, except the error line all other lines will be displayed as output.

Program to display variable

$ vi var.sh

```
A=10

B=20

echo A=$A and B=$B

-

-

:wq
```

$ sh var.sh

A=10 and B=20

Reading variables from input buffer

"read" is a command to read data from keyboard.

Syn: read var1 var2 var3...

"read" command will read input data up to space or tab or enter key.

Ex:

Read a b

Input → 10 20

Then 'a' value is 10 and 'b' value is 20

Program to read and display two numbers

$ vi num.sh

```
echo enter two numbers

read a b

your numbers are $a and $b

-

:wq
```

$ sh num.sh

Enter two numbers
30 40
Your numbers are 30 and 40

- **Escape characters**

Character	Task
\007	Alert Bell (generates beep sound)
\b	Back space
\c	End of line
\n	New line
\r	Carriage return (beginning of the same line)
\t	Horizantal tab

\v	Vertical tab
\”	Double quote
\’	Single quote
\\	Back slash
\$	Dollar
#	comment

Programs to read and display employee information

Printf “enter emp details \n”

Printf “enter emp num:”
Read enum

Printf “enter name:”
Read ename

Printf “enter salary:”
Read salary

Printf “enter dept:”
Read dept

Printf “enter age:”
Read age

Printf “\n emp num: $enum”
Printf “\n emp name: $ename”
Printf “\n salary: $salary”
Printf “\n dept: $dept”
Printf “\n age: $age”

To append data to the file
Printf “$enum: $ename: $salary: $dept: $age\n”>>empfile.dat

- **Operators**

1. Arithmetic operators

+ → addition
- → subtraction
* → multiplication
/ → division

% → modulo (remainder)

2. Relational operators

Numeric Comparisons

-gt → greater than
-ge → greater than or equal to
-lt → less than
-le → less than or equal to
-eq → is equal to
-ne → not equal to

Sting Comparisons

s1==s2 → is equal to
s1!= s2 → not equal to
-n s1 → return true if s1 is not null
-z s1 → return true if s1 is null

3. Logical Operators

-a → And
-o → Or
-n → Not

Working with arithmetic operators

"expr" is a command to perform mathematical operations

Syn:

expr $ var1 + $ var2 + $ var3 + $ var4 ……..

Ex:

a=10
b=20
c=a+b → value of 'c' is a+b
c=$a+$b → value of 'c' is 10+20
c=expr $a + $b → value of 'c' is expr
c=`expr 4a + $b` → value of 'c' is 30

Note: Here also to perform mathematical operations command quotes which are below the Esc key are compulsory. Otherwise 'expr' will be treated as string not command.

Program to add, subtract, multiply two numbers

$ vi math.sh

```
echo enter two numbers
read a b
c=`expr $a + $b`
```

```
echo addition =$c
c=`expr $a - $b`
echo subtraction =$c
c=`expr $a \* $b`
echo multiplication = $c
```

```
$ sh math.sh

enter two numbers
100      2
addition = 102
subtraction = 98
multiplication = 200
```

Note: After '=' assignment operator don't give space.

As '*' is a wild card character in Unix, to convert it as multiplication symbol place back slash '\' before '*'

- **Control Statements**

If statement

1. simple - if
2. else – if
3. nested – if
4. ladder – if

1. Simple – if

```
if [ condition ]
then
   ---------------
   ---------------
fi
```

If condition is true execute otherwise end if.
Space after 'if' and after and before square brackets is compulsory.

2. else – if

```
if [ condition ]
then
   ---------------
   ---------------
else
   ---------------
   ---------------
fi
```

3. nested – if

```
if [ condition ]
then
   ---------------
   ---------------
        if [ condition ]
        then
          ---------------
          ---------------
                if [ condition ]
                then
                  ---------------
                  ---------------
                else
                  ---------------
                  ---------------
                fi
        else
            ---------------
            ---------------
        fi
else
  ---------------
   ---------------
fi
```

4. Ladder – if

```
if [ condition ]
then
   ---------------
   ---------------
elif [ condition ]
then
   ---------------
   ---------------
elif [ condition ]
   ---------------
   ---------------
elif [ condition ]
then
   ---------------
   ---------------
else
```

```
    ---------------
    ---------------
fi
```

"Ladder if" is reverse of "nested if". "Nested if" for positive logic & "ladder if" is for negative logic.

Program to find the greater of two numbers

```
echo enter two numbers
read a   b
if [ $a –gt b ]
then
echo greater num=$a
else
echo greater num=$b
fi
```

Program to find the greater of three numbers

```
echo enter three numbers
read a   b   c
if [ $a –gt b ]
then
        if [ $a –gt c ]
        then
           echo greater num=$a
        else
           echo greater num=$c
        fi
elif [ $b –gt $c ]
then
    echo greater num=$b
else
    echo greater num=$c
fi
```

Program to find result of a student. If all subject marks are greater than or equal to 40 then result is pass otherwise fail.

```
echo enter three subject numbers
read m1   m2   m3
if [ $m1 –ge 40 ]
then
        if [ $m2 –ge 40 ]
        then
```

```
                    if [ $m3 –ge 40 ]
                    then
                        echo pass
                    else
                        echo fail
                    fi
               else
                   echo fail
               fi
          else
             echo fail
          fi
```

Same as above program using “and” Operator

```
echo enter three subject numbers
read m1   m2   m3
if [ $m1 –ge 40 –a $m2 –ge 40 –a $m3 –ge 40 ]
then
     echo pass
else
    echo fail
fi
```

Same as above program using “ladder if”

```
echo enter three subject numbers
read m1   m2   m3
if [ $m1 –lt 40 ]
 then
      echo fail
elif [ $m2 –lt 40 ]
then
      echo fail
elif [ $m3 –lt 40 ]
then
      echo fail
else
      echo pass
fi
```

Same as above program using “or” Operator

```
echo enter three subject numbers
read m1   m2   m3
if [ $m1 –lt 40 –o $m2 –lt 40 –o $m3 –lt 40 ]
then
```

```
    echo fail
else
    echo pass
fi
```

Program to read the score of cricket bats man and display comment on the score

```
echo enter the score of  batsman
read score
if [ $score –eq 0 ]
then
   echo DUCK OUT
elif [ $score –lt 50 ]
then
   echo normal score
elif [ $score –lt 100]
then
  echo half centure
else
  echo century
fi
```

- **Working with file permissions**

Test	True if
-e file	file exist
-f file	file exist and is a regular file
-d file	file exist and is a directory file
-r file	file exist and is a readable file
-w file	file exist and is a writable file
-x file	file exist and is a executable file
-s file	file exist and has size greater than zero bytes
-L file	file exist and is a symbolic link file
f1 –nt f2	f1 is newer than f2 (check date and time of creation)
f1 –ot f2	f1 is older than f2 (check date and time of creation)
f1 –et f2	f1 is linked f2

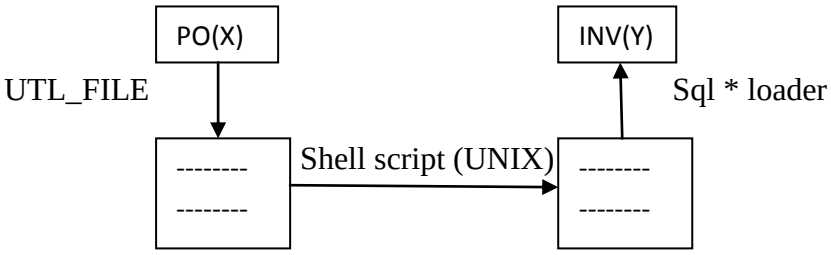
Program o test type of the file

```
echo enter the file name
read fname
if [ -f $fname ]
then
    echo regular file
elif [ -d $fname ]
then
    echo directory file
else
   echo $fname is not found
fi
```

Program to test the file permissions

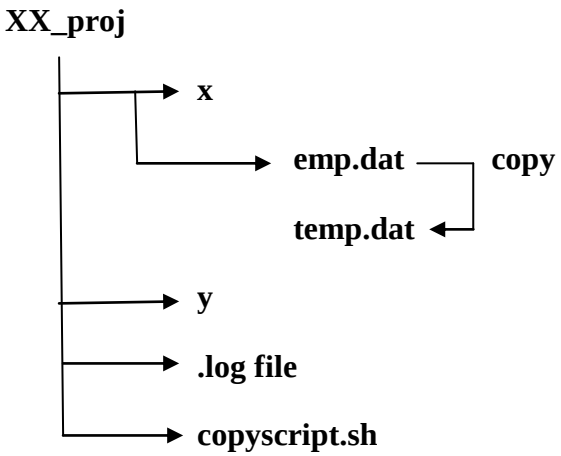
```
echo enter the file name
read fname
if [ -f $fname ]
then
        if [ -r $fname ]
        then
          cat $fname
        else
           echo $fname read permissions denied
        fi
        elif [ -d $fname ]
        then
                if [ -r $fname ]
                then
                  ls –l $fname
                else
                   echo $fname read permissions denied
                fi
else
  echo $fname not found
```

Real time example:



Data file (emp.dat)

Requirement



Writing a script to move the data file from one directory to another directory with all validations.

```
if [ -d x ]
then
        if [ -d y ]
        then
                if [ -w y ]
                then
                        if [ -x x ]
                        then
                          cd x
                # echo present location is `pwd`
                                if [ -f emp.dat ]
                                then
                                   cp emp.dat   temp.dat
                                   mv temp.dat   ../y
                                   echo operation successfully completed
                                 # rm emp.dat
                                else
```

```
                    echo operation failed
                    Timestamp=’date+%h-%d-%H-%M-%S’
                    filename=$Timestamp
                    date>>../$filename
                   echo emp.dat not found >> ../$filename
                  fi
            else
              chmod u+x  x
              cd x
            # echo present location is `pwd`
                 if [ -f emp.dat ]
                    then
                       cp emp.dat   temp.dat
                       mv temp.dat   ../y
                       echo operation successfully completed
                   # rm emp.dat
                    else
                      echo operation failed
                      Timestamp=’date+%h-%d-%H-%M-%S’
                      filename=$Timestamp
                      date>>../$filename
                      echo emp.dat not found >> ../$filename
                   fi
                cd..
                chmod   u-x   x
                fi
          else
             echo operation failed
             Timestamp=’date+%h-%d-%H-%M-%S’
             filename=$Timestamp
             date>>$filename
             echo no writable permission on y dir >>  $filename
          fi
      else
        echo operation failed
        Timestamp=’date+%h-%d-%H-%M-%S’
        filename=$Timestamp
        date>>$filename
        echo y dir not found >>$filename
      fi
else
  echo operation failed
  Timestamp=’date+%h-%d-%H-%M-%S’
  filename=$Timestamp
```

```
    date>>$filename
    echo x dir not found>>  $filename
fi
```

- **Case Statements**

```
case $variable in
        const 1 )
                ------------
                ------------;;
        const 2 )
                ------------
                ------------;;
        const 3 )
                ------------
                ------------;;
        const * )
                ------------
                ------------;;
esac
```

‘;;’ is optional for last block i.e., ‘* case’ which is default case. Here ‘;;’ acts as a break.

Example

```
echo enter a num
read num
case $num in
        1 )
           echo one;;
        2 )
          echo two;;
        3 )
          echo three;;
        * )
          echo default
esac
echo end
```

Menu Programming

```
echo Commands menu
echo 1. list of files
echo 2. today date
echo 3. year calendar
echo 4. current working users
```

```
echo enter your choice
read choice
case $choice in
        1 )
             ls –l;;
        2 )
             date;;
        3 )
             echo enter the year
             read y
             cal $y;;
        4 )
            who;;
        * )
            echo wrong choice
esac
```

Note: Here in cases if you want you can execute scripts also.

Program to validate the number

```
echo enter a number
read num
case $num in
        [1-9] )
             echo valid single digit number
            sh script1.sh
        [1-9][0-9] )
              echo valid two digit number
        [1-9][0-9][0-9] )
              echo valid three digit number
        * )
             echo invalid number
esac
```

- **Loops**

1. while
2. until
3. for

Working with while

Syn:

Condition is true, loop will be repeated. If false the loop will be terminated.

```
while [ condition ]
do
      ------------
      ------------
done
```

Here if the condition is true, the statement executes until the condition become false.

The loop will be terminated when the condition is false.

Working with Until

The loop continues until the condition is true. If the condition is true the loop terminates.

Syn:

```
 until [ condition ]
do
    -------------
    -------------
done
```

Program to display numbers from 1 to n using while and until

```
echo enter the limit
read n
i=1
while [ $i –le $n ]    (or)  until [ $i –gt $n]
do
echo $i
i=`expr $i + 1`
done
```

Program to find sum of even numbers and odd numbers from 1 to n

```
echo enter the limit
read n
```

```
even=0
odd=0
i=1
while [ $i –le $n ]
do
        if [ `expr $i % 2` -eq 0 ]
        then
           even=`expr $even + $i`
        else
           odd=` expr $odd + $i `
        fi
i=` expr i + 1 `
done
        echo even sum =$even and odd sum =$odd
```

Program to find recourse of a given number

```
echo enter a number
read n
rev=0
while [ $n –gt 0 ]
do
r=`expr $n % 10`
rev=`expr $rev \* 10 + $r`
n=`expr $n / 10`
done
echo reverse $ rev
```

Exit command → to terminate the program
Break command → to terminate the loop

Syntax

```
while [ condition ]
do
        -----------
        -----------
        if [ condition ]
        then
           break
        fi
        --------------
        -------------
done

until [ condition ]
```

```
do
        ----------
        ----------
        if [ condition ]
        then
           break
        fi
        ----------
        ----------
done
```

Program to check the given number is prime or not using “exit” command

```
echo enter a number
read n
i=2
while [ $i –le `expr $n / 2` ]
do
        if [ `expr $n % $i` -eq 0 ]
        then
                echo not a prime number
                exit
        fi
   i=`expr $i + 1`
done
echo prime number
```

Program to check the given number is prime or not using “break” statement

```
echo enter a number
read n
i=2
flag=0
while [ $i –lt `expr $n / 2` ]
do
        if [ `expr $n % $i` -eq 0 ]
        then
                flag=1
                break
        fi
   i=`expr $i + 1`
done

if [ $flag –eq 0 ]
then
    echo prime number
```

```
else
    echo not a prime number
fi
```

Here flag is a temporary variable which holds true (1) or false (0) situations in real time.

True command → to make the condition as true. To set the undefined loop

```
while true
do
-------------
-------------
done
```

False command → to make the condition as false. To set the undefined loop

```
until false
do
-------------
-------------
done
```

Ex:

```
while true
do
        clear
        printf" \n\n\n\n\n\t\t\t hello world"
        sleep 2
        clear
        printf" \n\n\n\n\n\t\t\t welcome to hyderabad"
        sleep 2
done
```

Note: Press 'ctrl + c' to terminate from undefined loop.

```
until false
do
        clear
        printf" \n\n\n\n\n\t\t\t hello world"
        sleep 2
        clear
        printf" \n\n\n\n\n\t\t\t welcome to hyderabad"
        sleep 2
done
```

- **Nested loops**

Syn:

```
while [ condition ]
do
        --------------
        while [ condition ]
       do
              --------------
              while [ condition ]
              do
                     --------------
              done
              --------------
        done
        -----------------
done
```

Program to design the digital clock. 0 : 0 : 0 to 23 : 59 : 59 when reaches 24 hrs stop the clock and run the scripts with in time limit.

```
h=0
while [ $h –lt 24 ]
do
        if [ `expr $h % 2 –eq 0` ]
        then
            sh script1.sh
        # to run the script for every 2 hrs
        fi
        if [ `expr $h % 3 –eq 0` ]
        then
          sh script2.sh
       # to run the script for every 3 hrs
       fi
       m=0
       while [ $m –lt 60 ]
       do
            s=0
            while [ $s –lt 60 ]
            do
                clear
                printf" \n\n\n\n\n\t\t\t $h : $m : $s"
                sleep 1
                s=`expr $s + 1`
```

```
                done
                m=`expr $m + 1`
        done
        h=`expr $h + 1`
done
```

To terminate from this program before 24 hrs press ‘ctrl+l’ otherwise it will be running up to 24 hrs.

Program to display system time

This program will run 365 days. it will not terminate since its true condition will never fail.

```
while true
do
        hrs=’date+%H’
        min=’date+%M’
        sec=’date+%S’
# to run the particular script at particular amount of time
        if [ $hrs –eq 10 –a $min –eq 15 ]
        then
                sh script1.sh
        fi
        if [ hrs –eq 15 –a $min –eq 40 ]
        then
                sh script2.sh
        fi
# the above scripts will run every day at 10 hr 15 min and 15 hr 40 min.
        clear
        printf” \n\n\n\n\n\t\t\t $hrs : $min : $sec”
        sleep 1
done
```

Note: Alternative for this program is cron tab. cron tab is used for job scheduling.

- **Working with “for” loop**

Syn:

```
for variable in const1  const2 const3 ……
do
        ----------
        ----------
done
```

Program to check the constants

```
a=10
b=20
c=30
d=40
for i in $a $b $c $d
do
        echo $i
done
```

Program to display multiple files

```
for fname in file1  file2   file3   file4
do
        echo $fname contents are
        cat < $fname
done
```

Program to redirect file data and display

```
for fname in file1  file2   file3   file4
do
        cat < $fname > filex
done
```

Program to concatenation of the files and display

```
if [ -f filex ]
then
    rm filex
fi
for fname in file1  file2   file3   file4
do
        cat < $fname >>filex
done
```

Remove the filex before concatenation since if filex is existed with data, now the 4 files data will be appended to the old data.

Write a script to copy all files in required directory

$ vi filecopy.sh

```
echo Enter the dir to which do you want to copy the files
read dname
        if [ -d $danme ]
```

```
then
        if [ -w $dname ]
        then
                for fname in *
                do
                        if [-f $fname]
                        then
                            cp $fname $dname
                            echo copying $fname into $dname
                        fi
                done
        else
             echo $dname dir write permission defined, failed to copy
        fi
else
       echo $dname dir not found, failed to copy
fi
```

Script to count number of files, directories & other files in a current working directory

```
filecount=0
dircount=0
othcount=0
for fname in *
do
        if [ -f $fname ]
        then
            filecount=`expr $filecount+1`
            elif [ -d $fname ]
            then
                dircount=`expr $dircount+1`
            else
                othcount=`expr $othcount+1`
        fi
done
echo total number of files : $filecount
echo total number of dirs : $dircount
echo total number of other files : $othcount
```

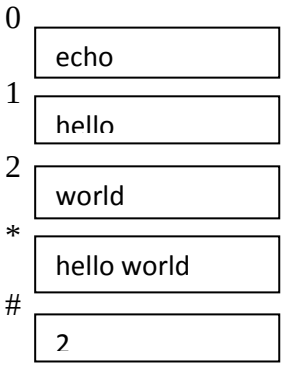
Working with command line arguments (or) positional parameters

Here variables will be generated to store the command & parameters. Next variables store the parameters.

Command line parameters are used to develop user own tools and commands.

Ex:

$ echo hello world



* → store all parameters

→ stores the number of arguments.

To get the count of parameters: $#

To get all the parameters: $*

Creation of user tools (services)

$ put hello world

Here put is a user defined command which have to work as "echo"

STEP 1:

$ vi put

```
for str in $*
do
        printf"$str"
done
echo   → to place cursor to the next line
```

STEP 2:

Assign the execute permission to the program file.

$ chmod u+x put

Execution

$ ./put hello world

hello world

Note: "./" indicate current working directory. Shell looks at current working directory only. Without "./", to execute the program set the path at .bash_profile file as "./"

$ echo $PATH → displays the path

We can set the path in .bash_profile file. after setting path "logout" from shell and "login" again.

$ put hello world

Now shell looks whole system.

Setting path to execute shell script at command prompt without using sh command

Step1:

$pwd

$cd → go to login directory

$ vi .bash_profile

export PATH=----------:./

Now logout and login again.

Now you can execute shell programs without sh command.

Program to create a file using command line arguments

By using "cat" command if we are creating a file, if the file name already exist the new data

Silently overrides old data. So there is loss of data.

So creating a command named "create" to create a file without loss of data.

To create a file without loss of data, $# (count of parameters) should be equal to 1 otherwise syntax error and stop the program since only one file can be created at once. Next check the existence of file. if file exist don't create the file.

$ create [file name]

STEP 1:

$ vi create

```
if [ $# -ne 1 ]
then
        echo syntax error. usage: create [file name]
        exit
fi
if [ -f $1 ]
then
        echo $1 is an already existed file so cannot create file
elif [ -d $1 ]
then
        echo $1 is a already existed directory. Cannot create.
else
        cat >$1
        echo successfully created
fi
```

STEP 2:

$ chmod u+x create

Execution

$ ./create file1

Program to display contents of multiple files

Creating a command named “disp” to display contents of multiple files.

$ disp file1 file2 file3 file4 …

STEP 1:

$ vi disp

```
for fname in $*
do
if [ -f $fname]
then
        if [ -r $fname]
        then
                echo $fname contents are
                cat $fname
        else
                echo $fname read permission denied
        fi
```

```
elif [ -d $fname ]
then
        echo $fname is a directory cannot display
else
        echo $fname not found
fi
done
```

STEP 2:

$ chmod u+x disp

Execution

$ ./disp file1 file2 file3 file4

Program to copy one file to another file

Creating a command named “mycopy” to copy one file to another file

$ mycopy [source file] [destination file]

STEP 1:

$ vi mycopy

```
if [ $# -ne 2 ]
then
        echo syntax error usage mycopy [source file] [destination file]
exit
fi

if [ -f $1 ]
then
        if [ -r $1 ]
        then
                if [ -f $2 ]
                then
                        if [ -w $2 ]
                        then
                            cp $1 $2
                        else
                            echo $2 write permission denied
                        fi
                elif [ -d $2 ]
                then
                   echo $2 is a directory cannot copy
                else
```

```
                    cp $1 $2
                    echo copied successfully
                fi
            else
                echo $1 read permission denied
            fi
        elif [ -d $1 ]
        then
          echo $1 is a directory cannot copy
        else
            echo $1 not found
        fi
```

Step 2 and 3 as above programs

❖ Shell Processes

To display the current running processes **ps command with option –f is** used.

$ ps –f

UID	PID	PPID	C	STIME	TTY	TIME	CMD
User2	367	291	0	12:45:12	console	0:00	vi file1
User2	456	1	0	10:30:12	console	0:00	-bash
User2	336	342	0	11:30:12	console	0:00	/usr/bin/bash –l

Other options of "ps" command

-e or –A → all processes including user and system processes

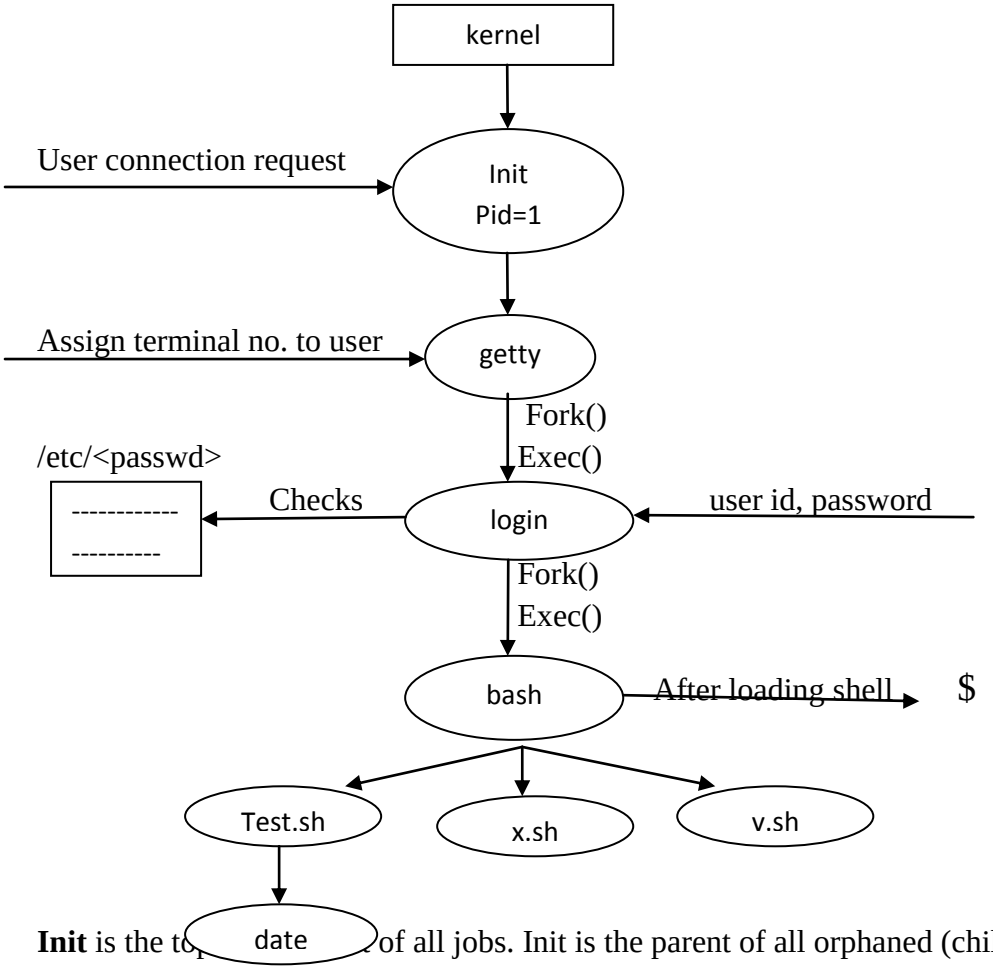
-u → processes of users only

-a → all users processes

-l → long list including memory information

-t → processes running on terminal term

Shell processes creation



Init is the top of all jobs. Init is the parent of all orphaned (child without parent) childs.

Fork : creates a child process and imports all parent process.

Exec: To load and start execution of application in the child process.

Wait: The patent then executes the wait system call to wait for the child process to complete.

Running the jobs in the back ground

To run the job in back ground assign '&' after the job

$ date &
550

$ lp file1 & ->printer command
551

$ cat >file1 &
552

$ mysql &
553

$ starttomcat &
554

$ ps –f
Process id of background working jobs will be displayed (multitasking).

Killing processes

To kill the jobs those are working background in current working session

$ kill pid1 pid2 pid3

Eg:

$ kill 550 551 552 553 554

❖ Job Scheduling (Cron tabs)

Cron tab is used for job scheduling.

$ crontab –e

e stands for edit

# min	hrs	days	months	weeks	shell script name
# [0-59]	[0-23]	[1-31]	[1-12]	[0-6]	sh scriptname.sh
10	11	15	08	*	sh script1.sh
30	14	10	01	*	sh script2.sh

:wq

In above job scheduling the script1.sh runs at 15th of august month at 11 hrs 10 mins. in weeks * indicates any week day is possible.

Script2.sh runs at 10th of january month at 14 hrs 30 mins of any week day.

$ vi test.sh

date >> myfile.dat

# min	hrs	days	months	weeks	shell script name
# [0-59]	[0-23]	[1-31]	[1-12]	[0-6]	sh scriptname.sh
*	*	*	*	*	sh test.sh

:wq

Removing Scripts from the cron tab

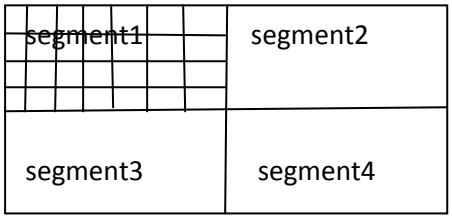
$ crontab –r

All data will be removed

❖ Storage of Files

All the files are stored permanently in UNIX hard disk. The Hard disk is divided into 4 types of segments.

Each segment consists of blocks



Segment 1 consists of Bootable Blocks
Segment 2 consists of File Blocks
Segment 1 consists of i-node Blocks
Segment 1 consists of Data Blocks

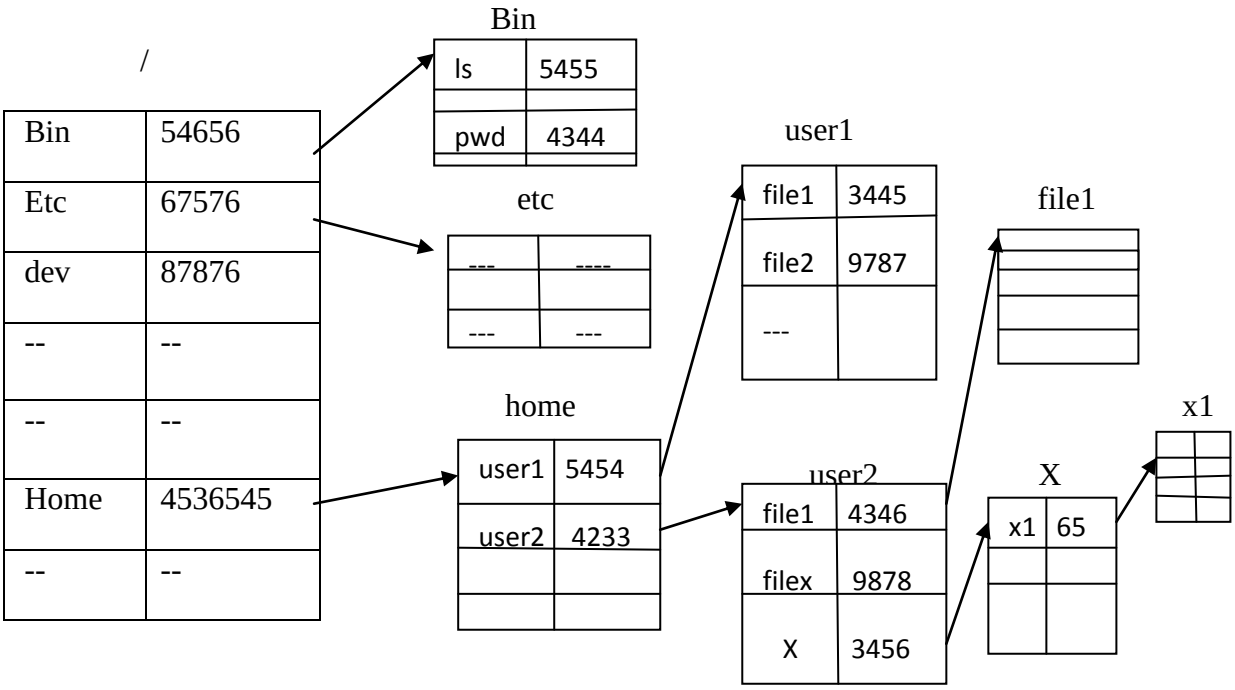
Size of each block is 1024 bytes (1 kb)

- **Bootable Blocks**

This Blocks contains all bootable files of UNIX OS. 1st Block is called MBR (Master Bootable Record). Booting of UNIX starts with MBR. Bootable files are responsible to load the kernel into memory.

- **File Blocks**

This blocks maintains hierarchy of UNIX filing system. With the help of i-node numbers the file is searched.



$ vi /

Root directory contents will be displayed.

- **i-node Blocks**

These blocks contain i-node numbers of the files. i-node number is a unique id for the file. Every file has separate i-node number.

I-node number is a address of memory space in the i-node blocks where the information about the file is stored.

I-node memory space provides following information about the file.

Type of the file

- Permissions of the file
- Number of links of the file
- Owner name and Group name of the file
- Size of the file
- Date and time of creation of the file
- Date and time of modification of the file
- Date and time of last access of the file

- **Data Blocks**

This Blocks contains actual data of the file. Files data storage is random in the data blocks.

❖ UNIX Administrator

→ Administrator prompt

init 0 → To shut down the server.

init 1 → To reboot the server

The above commands are conman for UNIX, LINUX and Solaris

Acquiring super users' status

"su" is the command to get super user status.

Syn: # su [user name]

Password: ********

Only admin can act as super user.

Ex:

su user1

$ cd /home/user1

$ pwd

/home/user1

If any file is created by admin as super user in user1 directory. The owner of the file will be user1 only.

$ cat >file1

$ ls –l
-rw-r—r—1 user1 group1 --------

"exit" is command to come back to '#' prompt

$ exit

Admin can create a file directly by changing into user's directory. Now root will the owner of the file.

cd /home/user1

cat >file2
---

ls –l

-rw-r—r— 1 user1 group1 file1 --------
-rw-rwxr— 1 root root file2 --------

Setting the system date

Syn: # date MMDDhhmmYYYY

Ex: # date 010612502012

To display date

date

thu jan 06 12:50:00 IST 2012

UNIX will understand the century till the year 2038

Communicating with users

Wall is the command to communicate with the users. It is used to send message to all users connected to that server.

wall

The server will shut down today at 12:00 hrs

Please complete your work before that time

[ctrl+d]

Ulimit

Setting limits to file size

The “ulimit” command imposes a restriction on the maximum size of the file that a user is permitted to create.

ulimit 12754 → in 512 – byte blocks

Place the statement in /etc/profile so that every user has to work within these limit.

USER MANAGEMENT

groupadd: adding a group

groupadd –g 300 group1

Here 300 is the group id for group1

Group information will be stored in /etc/<group>

$ vi group

You can get all the details of groups

useradd: Adding a User

Syn:

useradd –u [UID] -g [group name] -c “comment” –d [home directory] -s [shell] –m [user name]

Ex:

useradd –u 4001 -g group1 -c “developers” –d /home/user1 -s /bin/bash –m user1

With “-m” option the user is forcefully created

User’s information will be stored in /etc/<passwd>

$ vi passwd

All the information of users will be displayed.

usermod: Modifying User

usermod -s /bin/ksh user1 → to modify the shell

Can modify everything of user except home directory.

userdel: Deleting user’s

userdel [user name]

Deleting group

To delete a group first of all associated users with that group must be deleted.

groupdel [group name]

Maintaining security

By providing password to the users we can maintain the security

Syn: # passwd [user name]

Ex:

passwd user1

It asks to type the password. give the password

Changing Owner of the file

chown [new owner name] [file name]

Changing Group name

chgrp [new group name] [file name]

shutdown: SHUTDOWN the system

It is a LINUX command to shutdown the system

shutdown 17:30 → to shutdown at 17:30 hrs

shutdown –r now → shutdown immediately and reboot (restart)

shutdown –h now → shutdown immediately and hault (complete shut down)

df: Reporting Free Space

df → for all

df –k / /usr → reports on / and /usr file system

df –h / /usr → for more information

du: Disk usage

du → for all

du –s /usr

Device Blocks

ls –l /dev

❖ Working with oracle on UNIX/LINUX server

After login into LINUX

$ dbstart → it will start database

$ sqlplus scott/tiger@orcl

SQL>

to go back to OS again

SQL> !

$

To get back to oracle type “exit” command

SQL>

Work with sql queries here.

❖ AWK

AWK was developed by \

1. Aho
2. Weinburg
3. Kernighan

AWK is the combination of head, tail, cut and grep commands

Syn:

$ awk '/pattern/{action}' [data file]

Ex:

$ awk '/sales/{print}' emp.dat

Only the records of sales dept will be displayed

Options:

$ awk -F ":" '/pattern/{action}' [data file]

Ex:

$ awk -F ":" '{print $0}' emp.dat

$0 → for all fields
$1 → first field
$2 → second field
-
-

To get multiple fields
Syn:

$ awk -F ":" '/pattern/{print $1, $2, $3 ..}' [data file]

Ex:

$ awk -F ":" {print $1, $2, $3, $6, $4, $5}' emp.dat

Operators using with AWK

>, <, >=, <=, ==, !=

To get 1st record

$ awk -F ":" 'NR==1{print}' emp.dat

To get first 4 records

$ awk -F ":" 'NR<=1{print}' emp.dat

To get 5th record to the end of the file

$ awk -F ":" 'NR>=5{print}' emp.dat

To get particular record

$ awk -F ":" 'NR==1; NR==3;NR==5{print}' emp.dat

To get 3rd record to the 6th file

$ awk -F ":" 'NR==3,NR==6{print}' emp.dat

To get particular fields of particular records

$ awk -F ":" 'NR==3,NR==6{print $1,$2,$5}' emp.dat

Records of employee Sal more than 7000

$ awk -F ":" '$3>=7000{print}' emp.dat

Depending length of the field retrieving records

$ awk -F ":" 'length($2)>=5{print}' emp.dat

❖ Real Time Tools

Winscp tool:

Double click on winscp tool. A window is opened; give the IP address at host name & user name & password as admin. Click on Login button.

A window is opened with two set ups i.e., client and server.

You can upload & download files easily by drag and dropping the files between both setups by setting the transfer mode as binary.

Note: for small amount of data file transfers we can use this tool.

Toad:

With the Help of Toad tool we can work on Oracle Sql queries.Write a query in the working space and place the cursor anywhere on the statement and press (ctrl+enter key) to execute the query.

Otherwise select the statement and at above icons there is execute icon select it and execute.