Module-1 Contents

Chapter1

INTRODUCTION

1.1 UNIX ARCHITECTURE

1.1.1 Division of Labor: Kernel and Shell:

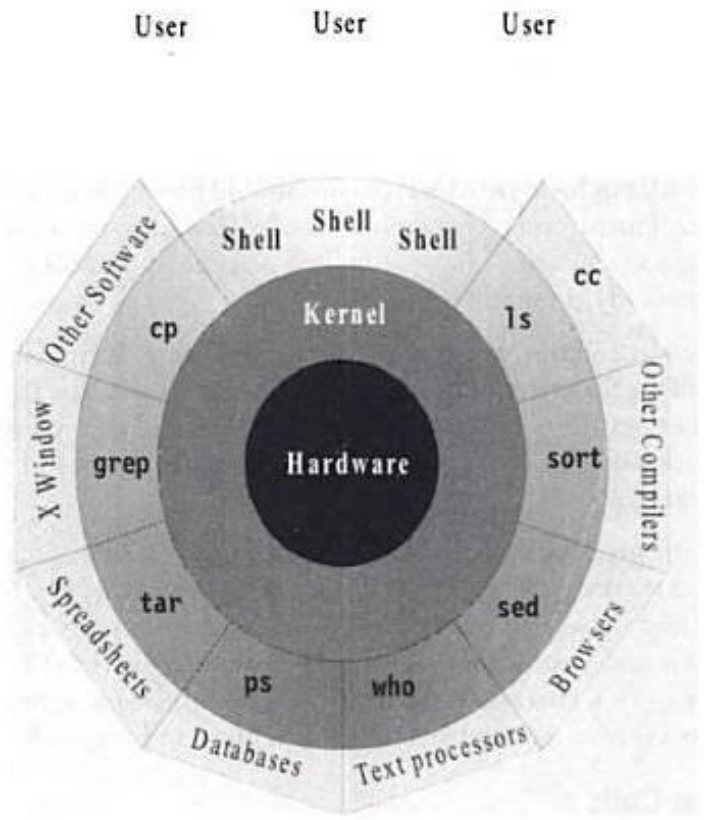

Fig. 1.1 The kernel – shell relationship

- The **kernel** is the core of the operating system - a collection of routines mostly written in C. it is loaded into memory when system is booted and communicates directly with the hardware.
- The applications (user programs) that need to access the hardware use the services of kernel, these programs access the kernel through a set of functions called as system calls.
- The kernel manages the system's memory, schedules processes, and decides their priorities.
- The **shell** which is the outer part of the operating system translates the commands into action (command interpreter).
- Even though there's only one kernel running on the system, there could be several shells in action- one for each user who is logged in.

- When the user enters a command through the keyboard, the sell thoroughly examines the keyboard input for special characters. If it finds any, it rebuilds a simplified command line, and finally communicates with the kernel to see that the command is executed.

1.1.2 The File and Process:

- A **file** is an array of bytes that stores information. It is also related to another file in the sense that both belong to a single hierarchical directory structure.
- UNIX considers even the directories and device as files.
- A **process** is the second abstraction UNIX provides. It can be treated as a time image of an executable file. Like files, processes also belong to a hierarchical structure.

1.1.3 The system Calls:

- These are the set of **function calls** used by programs to access kernel.
- All the UNIX flavors have one thing in common: *they use the same system calls.* Ex: A typical command writes a file with the **write** system call.

1.2 The Features of UNIX:

1.2.1 Multiuser system:

- The system allows multiple users to work on the operating system simultaneously and independently.
- The computer breaks up a unit of time into several segments and each user is allotted a segment, so at any point in time the machine will be doing the job of a single user.

1.2.2. Multitasking system:

- UNIX is a multitasking system. In multitasking environment, a user sees one job running in the foreground; the rest run in the background.
- The user can switch jobs between background and foreground, suspend, or even terminate them.

1.2.3. Building block approach:

- The UNIX commands are developed to perform one simple job.
- It's through the pipes and filters that UNIX implements the small is beautiful philosophy. Today, many UNIX tools are designed with the requirement that the output of one tool be used as input to another.
- By interconnecting a number of tools, user can have a large number of combinations of their

usage.

- For example: ls (listing the files and directories) and wc –w (word count) were used with | (pipe) to count the number of files in your directory.

$ ls | wc –w

- The commands that can be connected in this way are called filters because they filter data in different ways

1.2.4. The UNIX Toolkit:

- To properly exploit the power of UNIX, one should use the host of applications that are shipped with every UNIX system.
- These applications are quite diverse in scope. There are general-purpose tools, text manipulation utilities, compilers and interpreters, networked applications and system administration tools.

1.2.5. Pattern matching:

- UNIX features a very sophisticate pattern matching features.
- For example: by using the ls command with an unusual argument (chap*) lists all the filenames starting with chap and ending with anything.
- (Known as meta-character) is a special character used by the system to indicate that it can match a number of filenames.
- The matching is not confined to filenames only. Some of the most advanced and useful tools also use a special expression called regular expression that is framed with meta- characters.

1.2.6. Programming facilities:

- The UNIX shell is also a programming language; it was designed for a programmer.
- It has all the necessary ingredients, like control structures, loops and variables, that establish it as a powerful programming language in its own right.
- These features are used to design shell scripts.

1.2.7. Documentation:

- The principle online help facility available is the man command, which remains the most important reference for commands and their configuration files.
- Apart from the online documentation there's a vast ocean of UNIX resources available on the internet. There are several newsgroups, blogs and websites that provide information about UNIX.

1.3 The UNIX Environment:

UNIX is used in three different computer environments: Personal environment, Time sharing environment, Client/Server environment.

1.3.1 Personal Environment:

- Although originally designed as a multiuser environment, many users are installing UNIX on their personal computers.
- The trend of personal UNIX systems accelerated in the mid 1990's with the availability ofLINUX, the Apple system X, released in 2001, incorporated UNIX as its kernel.

1.3.2. Time-Sharing Environment:

- In Time-Sharing environment, many users are connected to one or more computers. Their terminals are non-programmable.
- Output devices (such as printers) and auxiliary storage devices (such as disks) are shared by all of the users.
- All the computing must be done by the central computer, it must control the shared resources, it must manage the shared data and printing, and it must also do the computing.
- All these work tend to keep the computer busy and the responses will be slow.
- A typical college lab in which a minicomputer is shared by many students is shown in the figure below:

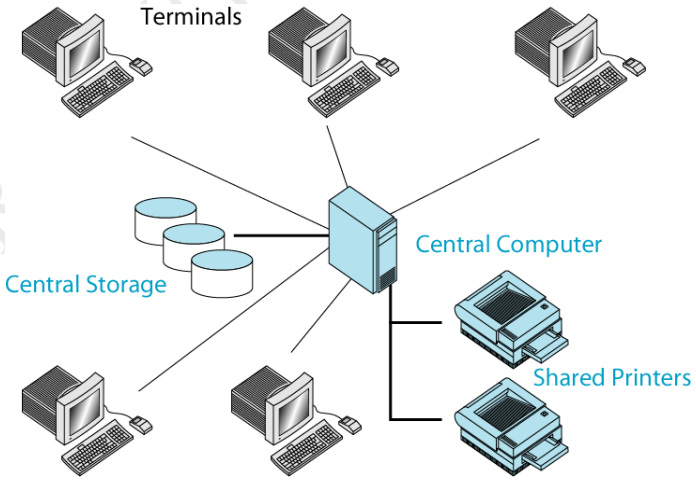

Fig1.2 Time Sharing Environment

1.3.3. Client/Server Environment:

- A Client/Server computing environment splits the computing function between a central computer and users' computers.

- The users are given personal computers or workstations so that some computation responsibility can be moved off the central computer and assigned to the workstations.
- In the Client/Server environment, the users' workstations are called as the Client. The central computer system is known as the Server.
- Since the work is shared between the client and server, the response time and monitor display are faster and users are more productive.
- A typical Client/Server environment is shown in the figure below:

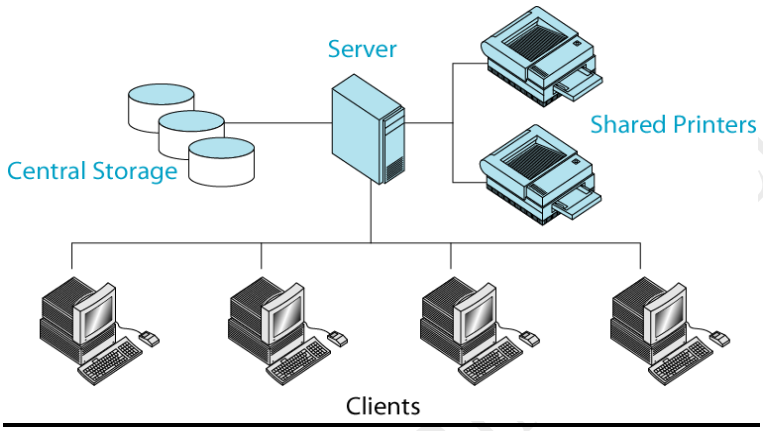

Fig 1.3 Client/Server Environment

1.4 UNIX Structure:

UNIX consists of four major components: the Kernel, the Shell, a standard set of Utilities and Application programs. These components are shown in the figure below:

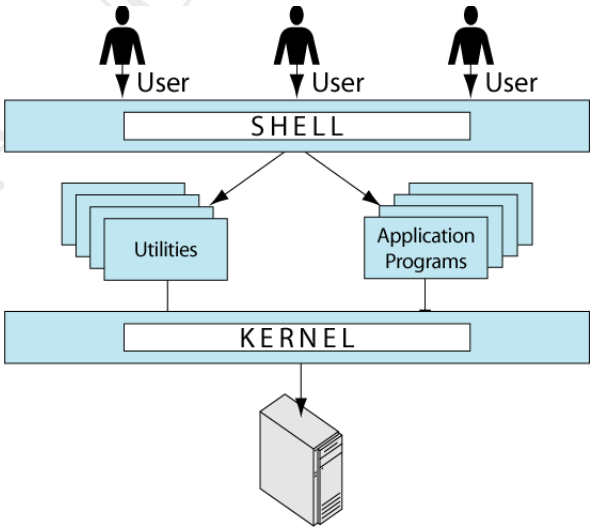

Fig 1.4 Component of UNIX

1.4.1 The Kernel:

- Kernel is the heart (core) of the UNIX operating system.

- It contains most basic parts including process control and resource management.
- All the other components call on the kernel to perform these services for them.

1.4.2 The Shell:

- It receives and interprets the commands entered by the user.
- To do any operations in the system, user must give the Shell a command. If the command requires a utility, the shell requests that the kernel execute the utility. If the command requires an application program, the shell request that it be run.
- There are two major parts of a shell. The first is the Interpreter; the second part of the shell is a programming capability that allows you to write a shell script.
- The shells are shown in the figure below:

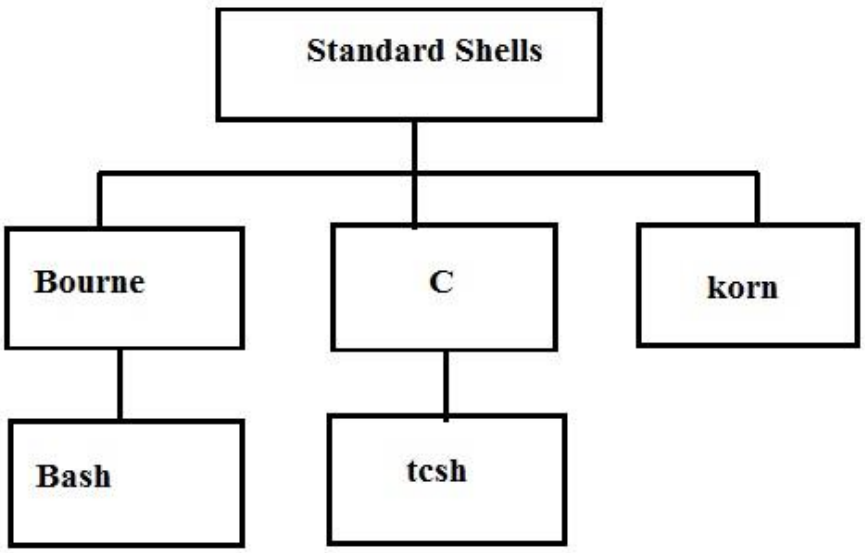

Fig 1.5 Some Standard UNIX Shells

1.4.3 Utilities:

- A Utility is a standard UNIX program that provides a support process for users.
- There are literally hundreds of UNIX utilities. Three common utilities are text editors, search programs, and sort programs.
- For example: vi (text editor), the list (ls) utility displays the files that reside on the disk.

1.4.4 Applications:

- Applications are programs that are not a standard part of UNIX.
- These are written by system administrators, professional programmers or users, they provide an extended capability to the system.
- Many standard utilities started out as application years ago and proved so useful that they are

now part of the system.

1.5 POSIX and the Single UNIX Specification:

- The group of standards, the Portable Operating System Interface for Computer Environments (POSIX), was developed based on the instructions given by Institution of Electrical and Electronics Engineers (IEEE).
- POSIX refers to the operating system in general, but was based on UNIX. Two of the most-cited standards from the POSIX family are known as POSIX.1 and POSIX.2.
- **POSIX.1** specifies the system calls.
- **POSIX.2** specifies the shell and utilities.
- In 2001, a joint initiative of X/Open and IEEE resulted in the unification of the two standards, this is the Single UNIX Specification, Version 3 (**SUSV3**).
- The "Write once, adopt everywhere" approach to this development means that once software has been developed on any POSIX compliant UNIX system, it can be easily ported to another POSIX-complaint UNIX machine with minimum modifications.

1.6 General Features of UNIX commands/ Command Structure:

- A UNIX command is an action request given to the UNIX shell for execution.
- The simplest commands are a single line entered at the command line prompt that cause a program or shell script to be executed.
- **Command Syntax:** the general format or syntax of a command is shown below:

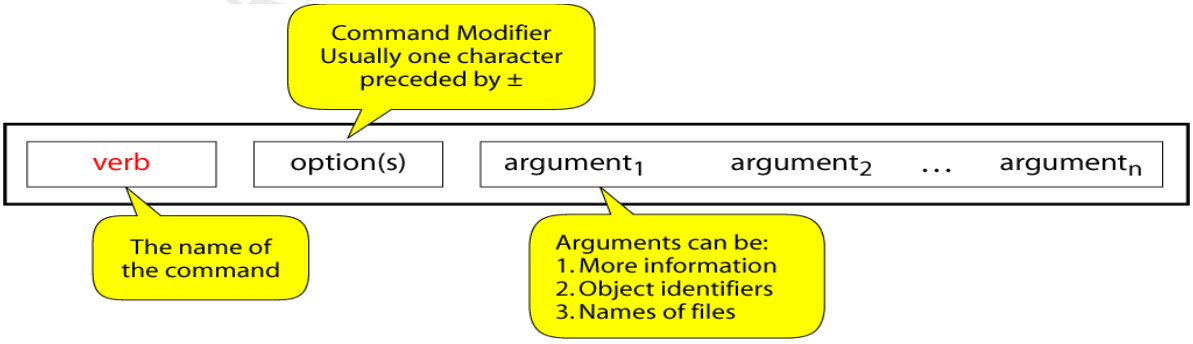

Fig 1.6 Command Syntax

1.6.1 Verb:

- The **Verb** is the command name. The command indicates what action is to be taken

- Command name and options/arguments have to be separated by spaces or tabs to enable the system to interpret them as *words*. Single or multiple spaces can be given to separate the commands and options/arguments.

  Ex:

  $ cat README

  The Shell compresses these multiple spaces to a single space

1.6.2 Options:

- The **Option** modifies how the action is applied. For example, when we display the date, we can use an option to specify if we want the time in GMT or local time.
- Options are usually one character preceded by a minus sign or plus sign.

  Ex:

  $ wc -c filename

  -w is the option to the command wc (-w indicate character count).

- Options can normally be combined with only one – sign, i.e., instead of using

  $ wc –c –w filename

  You might as well use

  $ wc –cw filename

To obtain the same output.

1.6.3 Arguments:

- The **Arguments** provide additional information to the command. For example, when displaying the contents the of a file, an argument can be used to specify the file name.
- Some commands may accept single argument, some may accept multiple arguments and some commands may not accept any arguments.
- Ex:

  $ wcabcdef xyz

  Where abc, def and xyz are the filename arguments to the command wc.

1.7 Basic Unix Commands

1.7.1 echo: Displaying a message

- The **echo** command copies its arguments back to the terminal.

- **Example1:** displaying a message

```
$ echo Hello World
Hello World

$ echo "Error 105: Invalid Total Sales"
Error 105: Invalid Total Sales
$_
```

- **Example 2:** displaying a message in multiple lines; this can be achieved by specifying the message to printed with double quotes (" "), when *[Enter]* is pressed, secondary prompt will be shown (>).

```
$ echo "UNIX
> Linux
> Ubuntu "
        UNIX
        Linux
        Ubuntu
$ _
```

1.7.2. printf: An alternate to echo

- The **printf** command is an alternate to echo and performs the similar operation performed by echo command.
- Unlike echo it doesn't automatically insert a newline unless the **\n** is used explicitly.
- **Example 1:**

```
$ printf " No filename Entered"
No filename Entered $_
```

(Note: quotes are not mandatory, but it's a good discipline to use them)

- **Example 2:**

```
 $ printf "No filename Entered \n"
No filename Entered
 $_
```

1.7.3. ls: Listing Directory Contents

- ls command lists all filenames in the current directory.
- The filenames in the current directory is arranged in **ASCII collating sequence** (numbers first, uppercase and then lowercase), with one filename in each line.
- **Example 1:**

```
$ ls
08_packets.html
TOC.sh
calendar
dept.lst
```

- Directories often contain many files, and you may simply be interested in only knowing whether a particular file(s) is available. In that case, just use **ls** with the filename(s).
- If the file isn't available, then the system will display the appropriate message.
- **Example 2:**

```
$ ls calendar
calendar
$ lsperl
Perl: No such file or directory
```

1.7.4 who: Who are the users ?

- UNIX maintains an account of all users who are logged on to the system. The **who** command displays an informative listing of these users:

  Example 1:

```
$ who
root      console      Aug  1  07:51 (:0)
kumarpts/10  Aug  1  07:56   (pc123.heavens.com)
sharmapts/6  Aug  1  02:10   (pc125.heavens.com)
sachinpts/14  Aug  1  08:36   (mercury.heavens.com)
```

  - The first column shows the **usernames** (or user-ids) of five users currently working in the system.

- The second column shows the **device names** of their respective terminals. The third, fourth and fifth columns show the **date** and **time of logging in**, the last column shows the **machine name** from where the user has logged in.

- The **–H** option prints the column headers, when combined with the –**u** option, provides a more detailed list.

Example 2:

```
$ who -uH
NAME    LINE          TIME        IDLE   PID    COMMENTS
root      console     Aug 1  07:51    0:48   11040    (:0)
kumarpts/10           Aug 1  07:56    0:33   11200  (pc123.heavens.com)
sachinpts/14          Aug  1  08:36     .     13678 (mercury.heavens.com)
```

- The first five columns are the same as before, the **sixth** one represent the **IDLE TIME**, which indicate how long it has been since there was any activity on the line.
- If the IDLE time is 0:48, it indicates that the user has no activity since 0 hours and 48 minutes.
- If the IDLE time is **.** (dot) it indicates that the user has something in the last minute.
- If the IDLE time is *old* it indicates that the user had no activity over 24 hours.
- The seventh column shows the **process id** (PID) of the process

1.7.5. date : Displaying the system date

- You can display the current date with the **date** command, which shows the **date** and **time** to the nearest second.
- **Example 1:**

```
$ date
Wed  Mar 6  16:22:40  IST  2005
```

- The command can be used with –u option to display the Greenwich Mean Time (GMT):
- **Example 2:**

```
$ date –u
Wed  Apr 3  16:22:40  GMT  2005
```

- The command can be used with suitable format specifiers as arguments. Each format is preceded by the + symbol, followed by the **%** operator, and a single character describing the format (format code).

- **Example 3:**

```
$ date  +%m
06

$ date  +%h
Mar

$ date  + "%h %m"
Mar 06
```

- The list of all the format specifiers is shown in the table below.

Format Code	Explaination
A	Abbreviated weekday name, such as Mon
A	Full weekday name, such as Monday
B	Abbreviated month name, such as Jan
B	Full month name, such as January
D	Day of the month with two digits (leading zeros), such as 01,02
D	Date in format mm//dd/yy,such as 01/01/99
E	Day of themonth with spaces replacing leading zeros, such as 1,2
H	Military time two-digit hour, such as 00,01,...,23
I	Civilian time two digit hour, such as 00,01,....12
J	Julian date (day of the year), such as 001,002,...366
M	Numeric two digit month, such as , 01,02,....12
M	Two digit minute, such as 01,02,....12

N	Newline character
P	Display am or pm
R	Time in format hour: minute:second with am/pm
R	Time in format hour:minute
S	Seconds as decimal number [00-61]
T	Tab character
T	Time in format hour:minute:second
U	Week number of the year, such as 00,01...53
W	Week of year [00-53] with monday being the first day of week
Y	Year within century
Y	Year as ccyy
Z	Time zone name, or no characters if no time zone is determinable

Table 1.1 date command format code

1.7.6. cal: The Calendar

- The calendar command, **cal**, displays the calendar for a specified month or for a year. The general format of cal command is shown below:

cal [[month] year]

- Everything within rectangular brackets is optional, cal can be used without arguments, in which case it displays the calendar of the current month:

```
$ cal
     August 2005
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

- The syntax also tells us that when cal is used with arguments, the month is optional but the year is not.
- For example to see the calendar for the month of March 2006, you need two arguments:

```
$ cal 03 2006
     March 2006
Su Mo Tu We Th Fr Sa
                1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

1.7.7. passwd : Changing your password

- The user password can be changed with the **passwd**command.
- passwd command expects the user to respond three times. First, it prompts for the old password. Next, it checks whether you have entered a valid password, and if you have, it then prompts for the new password, finally the command asks you to reenter the new password.
- When you enter the password, the string is *encrypted* by the system.

```
$ passwd
passwd: Changing password for kumar
Enter login password: ******
New password: *******
Re-enter new password: *******
Passwd (SYSTEM) :passwd successfully changed for kumar
```

1.8 Combining Commands:

- UNIX allows more than one command in the command line.
- Each command has to be separated by a **;** (semicolon)
- **Example 1:** output of wc command will be displayed on the first line followed by the output of date command.

$ wc note ; date

- **Example 2:** user may like to group them together within parenthesis and redirect the output to a disk file.

$ (wc note ; date) >newlist

- When a command line contains a semicolon, the shell understands that the command on each side of it needs to be processed separately.

1.9 Internal and External commands:

- **External command** is a program or a file having an independent existence in the **/bin** directory (or /usr/bin) of the system.
- Example for external commands are : **ls, date**
- **Internal command** is the Shell built-in commands that are not stored as separate files.
- Example for internal commands are : **echo, printf**
- The command type can be determined by using the **type** command.

1.10 The type Command

- The easiest way of knowing the location of an executable program is to use the **type** command.
- Example :

```
$ typels
ls is /bin/ls
```

- When the user execute the ls command, the shell locates this file in the **/bin** directory and makes arrangement to execute it.
- There are few commands which cannot be located in the file system i.e., the path of the command is not displayed when it is used with **type** command.
- Example:

```
$ type echo
echo is shell builtin
```

- Using **type** command it is possible to determine the command type (internal or external).
- If the output of type command displays the location of the program in the system, then such command are branded as an **External command**, ex: **ls.**
- If the **type** command output doesn't specify the location path, instead displays the message "*shell builtin*" then such commands are called as **Internal command,** ex: **echo.**

1.11 The root login:

- The UNIX system provides a special login name for the exclusive use of the administrator; it is called **root**.
- This account comes with every system. Its password is generally set at the time of installation of the system and has to be used on logging in:

  login: **root**
  password: ********** *[Enter]*
  #_
- The prompt of root is #
- Once you log in as root, you are placed in root's home directory. Depending on the system, this directory could be **/** or **/root**.
- Most administrative commands are resident in **/sbin** and **/usr/sbin**.

1.12 Becoming the super user: su

- Any user can acquire superuser status with the su command if he/she knows the root password.
- For example, the user Juliet (with home directory /home/Juliet) becomes a superuser in this way:

  $ su

  Password: ********** *[Enter]*

  #_
- The current working directory doesn't change, the # prompt indicates that Juliet now has the power of the superuser.
- To be in root's home directory on superuser login, use **su –l**.
- ***Creating a User's environment:***su when used with a -, recreates the user's environment without taking the login-password route
- For example :

  su – henry
- Temporarily creates henry's environment. This mode is terminated by using **exit**.

  $cd /home/kumar/progs

  $ pwd

  /home/kumar/progs

  $cd /bin

$ pwd

/bin

- cd can also be used without arguments:

$ pwd

/home/kumar/progs

$cd

$ pwd

/home/kumar

- cd without argument changes the working directory to home directory.

$cd /home/sharma

$ pwd

/home/sharma

$cd

/home/kumar

Chapter-2

UNIX FILES

2.1 THE FILE

- The file is the container for storing information.
- Neither a file's size nor its name is stored in file.
- All file attributes such as file type, permissions, links, owner, group owner etc are kept in a separate area of the hard disk, not directly accessible to humans, but only to kernel.
- The UNIX has divided files into three categories:
    1. Ordinary file – also called as regular file. It contains only data as a stream of characters.
    2. Directory file – it contains files and other sub-directories.
    3. Device file – all devices and peripherals are represented by files.

2.1.1 Ordinary File:

- Ordinary file itself can be divided into two types-
    1. **Text File** – it contains only printable characters, and you can often view the contents and make sense out of them.
    2. **Binary file** – it contains both printable and unprintable characters that cover entire ASCII range.Examples- Most Unix commands, executable files, pictures, sound and video files are binary.

2.1.2 Directory File:

- A directory contains no data but keeps some details of the files and subdirectories that it contains.
- A directory file contains an entry for every file and subdirectories that it houses.
- If you have 20 files in a directory, there will be 20 entries in the directory.
- Each entry has two components-
    1. The filename
    2. A unique identification number for the file or directory (called as inode number).

2.1.3 Device File:

- Installing software from CD-ROM, printing files and backing up data files to tape. All of these activities are performed by reading or writing the file representing the device.

- Advantage of device file is that some of the commands used to access an ordinary file also work with device file. Device filenames are generally found in a single directory structure, /dev.

2.2 Naming Files:

File name in UNIX can have the following:

- A filename can consist up to 255 characters.
- File may or may not have extensions, and consist of any ASCII character expect the / & NULL character.
- Users are permitted to use control characters or other unprintable characters in a filename.
- Examples - .last_time list. @#$%*abcda.b.c.d.e
- But, it is recommended that only the following characters be used in filenames-
    - Alphabetic characters and numerals
    - The period(.), hyphen(-) and underscore(_).

2.3 The Parent-Child Relationship:

- The files in UNIX are related to one another.
- The file system in UNIX is a collection of all of these files (ordinary, directory and device files) organized in a hierarchical (an inverted tree) structure as shown in below figure.

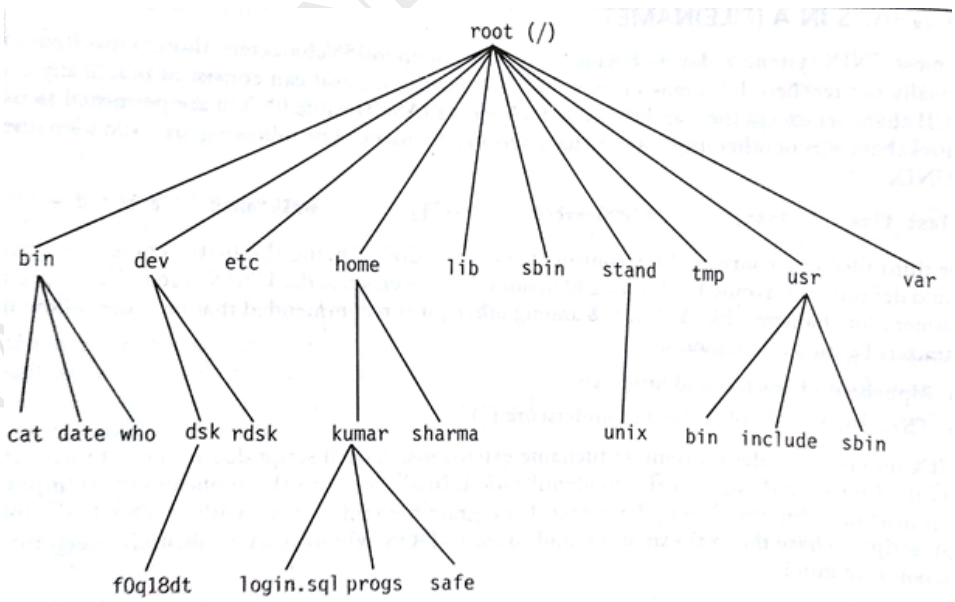

Fig2.1 UNIX File System Tree

- The feature of UNIX file system is that there is a top, which serves as the reference point for all files.

- This top is called root and is represented by a / (Front slash).
- The root is actually a directory.
- The root directory (/) has a number of subdirectories under it.
- The subdirectories in turn have more subdirectories and other files under them.
- Every file apart from root, must have a parent, and it should be possible to trace the ultimate parentage of a file to root.
- In parent-child relationship, the parent is always a directory.

2.4 The HOME Variable and the PATH Variable

2.4.1 HOME: Displaying Home Directory

- When you logon to the system, UNIX places you in a directory called home directory.
- It is created by the system when the user account is created.
- If a user login using the login name kumar, user will land up in a directory that could have the path name /home/kumar.
- The shell variable HOME knows the home directory.

$ echo $HOME

/home/kumar

2.4.2 PATH:

- List of directories searched by shell to locate the external command present in the system.
- **$ echo $PATH**
  /usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/home/kumar/.local/bin:/home/kumar/bin
- the value of the PATH variable contains the absolute path of the list of directories, each path is separated by a delimiter **:**
- When a command is typed in the command prompt, the shell searches the executable code of the command in the list of directories specified in the PATH command.
- If the command is not found in these list then the shell displays an error message.
- It is possible to add a new path to the PATH variable, show in the example below

$ PATH=$PATH:/opt/bin

- This will add the path /opt/bin to PATH.

$ echo $PATH

/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/home/kumar/.local/bin:/home/kumar/bin:/opt/bin

- The changes made to the PATH variable will be applicable only for that session.

2.5 DIRECTORY COMMANDS

2.5.1 pwd: CHECKING YOUR CURRENT DIRECTORY

- Any time user can know the current working directory using pwd command.

$ pwd

/home/kumar

- Like HOME it displays the absolute path.

2.5.2 cd: CHANGING THE CURRENT DIRECTORY

- User can move around the UNIX file system using cd (change directory) command.
- When used with the argument, it changes the current directory to the directory specified as argument, progs:

$ pwd

/home/kumar

$cd progs

$ pwd

/home/kumar/progs

- Here we are using the relative pathname of progs directory. The same can be done with the absolute pathname also.

2.5.3 mkdir: MAKING DIRECTORIES

- Directories are created with mkdir (make directory) command. The command is followed by names of the directories to be created. A directory patch is created under current directory like this:

$ mkdir patch

- You can create a number of subdirectories with one mkdir command:

  $mkdir patch dba doc

- For instance the following command creates a directory tree:

  $mkdirprogsprogs/cprogsprogs/javaprogs

- This creates three subdirectories – progs, cprogs and javaprogs under progs.
- The order of specifying arguments is important. You cannot create subdirectories before creation of parent directory.
- For instance following command doesn‘t work

  $mkdirprogs/cprogsprogs/javaprogsprogs

  mkdir: Failed to make directory “progs/cprogs”; No such directory mkdir: Failed to make directory “progs/javaprogs”; No such directory

- System refuses to create a directory due to fallowing reasons:
  - The directory is already exists.
  - There may be ordinary file by that name in the current directory.
  - User doesn‘t have permission to create directory.

2.5.4 rmdir: REMOVING A DIRECTORY

- The rmdir (remove directory) command removes the directories. You have to do this to remove progs:

  $ rmdirprogs

- If progs is empty directory then it will be removed form system.
- rmdir expect the arguments reverse of mkdir.
- Following command used with mkdir fails with rmdir

  $ rmdirprogsprogs/cprogsprogs/javaprogs

  rmdir: directory “progs”: Directory not empty

- First subdirectories need to be removed from the system then parent.
- Following command works with rmdir

  $ rmdirprogs/cprogsprogs/javaprogsprogs

- First it removes cprogs and javaprogs form progs directory and then it removes Progs from system.
- rmdir : Things to remember

- You can‘t remove a directory which is not empty
- You can‘t remove a directory which doesn‘t exist in system.
- You can‘t remove a directory if you don‘t have permission to do so.

2.6 ABSOLUTE AND RELATIVE PATHNAME

2.6.1 ABSOLUTE PATHNAME

- Directories are arranged in a hierarchy with root (/) at the top. The position of any file within the hierarchy is described by its pathname.
- Elements of a pathname are separated by a /. A pathname is absolute, if it is described in relation to root, thus absolute pathnames always begin with a /.
- Following are some examples of absolute filenames.

  /etc/passwd

  /users/kumar/progs/cprogs

  /dev/rdsk/Os3

- Example: date command can executed in two ways as

  $date // Relative path

  Thu Sep 7 10:20:29 IST 2017

  $/bin/date // Absolute path

  Thu Sep 7 10:20:29 IST 2017

2.6.2 RELATIVE PATHNAME

- A pathname can also be relative to your current working directory.
- Relative pathnames never begin with /.
- Relative to user home directory, some pathnames might look like this –**progs/cprogsrdsk/Os3**

2.7 Using Dot .and Double Dot .. in relative path name

- User can move from working directory /home/kumar/progs/cprogs to home directory /home/kumar using cd command like

  $ pwd

/home/kumar/progs/cprogs

$ cd /home/kumar

$ pwd

/home/kumar

- Navigation becomes easy by using common ancestor.
- **.** (a single dot) - This represents the current directory
- **..** (two dots) - This represents the parent directory
- Assume user is currently placed in /home/kumar/progs/cprogs

$ pwd

/home/kumar/progs/cprogs

$ cd ..

$ pwd

/home/kumar/progs

- This method is compact and easy when ascending the directory hierarchy. The command cd .. Translates to this ―change your current directory to parent of current directory.

- The relative paths can also be used as:

$ pwd

/home/kumar/progs

$ cd ../..

$ pwd

/home

- The following command copies the file prog1.java present in javaprogs, which is present is parent of current directory to current directory.

$ pwd

/home/kumar/progs/cprogs

$ cp ../javaprogs/prog1.java .

- Now prog1.java is copied to cprogs under progs directory.

2.8 FILE RELATED COMMANDS

2.8.1 cat: DISPLAYING AND CREATING FILES

- cat command is used to display the contents of a small file on the terminal.

$ cathello.c

include <stdioh> void main ()

{

printf("hello");

}

- As like other files cat accepts more than one filename as arguments

$ cat ch1 ch2

- It contains the contents of ch1 It contains the contents of ch2
- In this the contents of the second files are shown immediately after the first file without any header information. So cat concatenates two files- hence its name.
- **cat OPTIONS**
  - **Displaying Nonprinting Characters (-v)**

    cat without any option it will display text files. Nonprinting ASCII characters can be displayed with –v option.

  - **Numbering Lines (-n)**

    -n option numbers lines. This numbering option helps programmer in debugging programs.

- **Using cat to create a file:** cat is also useful for creating a file. Enter the command cat, followed by > character and the filename.

$ cat> new

- This is a new file which contains some text, just to Add some contents to the file new.
- When the command line is terminated with [Enter], the prompt vanishes. Cat now waits to take input from the user. Enter few lines; press [ctrl-d] to signify the end of input to the system to display the file contents of new use file name with cat command.

$ cat new

- This is a new file which contains some text, just to Add some contents to the file new.

2.8.2 cp: COPYING A File

- The cp command copies a file or a group of files. It creates an exact image of the file on the disk with a different name. The syntax takes two filename to be specified in the command line.
- When both are ordinary files, first file is copied to second.

$ cpcsacsb

- If the destination file (csb) doesn‘t exist, it will first be created beforecopying takes place. If not it will simply be overwritten without any warning from the system.
- Example to show two ways of copying files to the cs directory:

$ cp ch1 cs/module1 #ch1 copied to module1 under cs

$ cp ch1 cs #ch1 retains its name under cs

- cp command can be used to copy more than one file with a single invocation of the command. In this case the last filename must be a directory.
- Ex: To copy the file ch1,chh2,ch3 to the module , use cp as

$ cp ch1 ch2 ch3 module

- The files will have the same name in module. If the files are already resident in module, they will be overwritten. In the above diagram module directoryshould already exist and cp doesn‘t able create a directory.
- UNIX system uses * as a shorthand for multiple filenames.

$ cpch* usp # Copies all the files beginning with ch

cp options

- **Interactive Copying(-i) :** The –i option warns the user before overwriting the destination file, If unit 1 exists, cp prompts for response

$ cp -i ch1 unit1

$ cp: overwrite unit1 (yes/no)? Y

A y at this prompt overwrites the file, any other response leaves it uncopied.

- **Copying directory structure (-R) :** It performs recursive behavior command can descend a directory and examine all files in its subdirectories. -R : behaves recursively to copy an entire

directory structure

$ cp -R uspnewusp

$ cp -R class newclass

If the newclass/newusp doesn‘t exist, cp creates it along with the associated sub directories.

2.8.3 rm: DELETING FILES

- The rm command deletes one or more files.
- Ex: Following command deletes three files:

$ rm mod1 mod2 mod3

Can remove two chapters from usp directory without having to cd

$ rmusp/marks ds/marks

- To remove all file in a directory use *

$ rm *

Removes all files from that directory

- **rm options**
  - **Interactive Deletion (-i) :** Ask the user confirmation before removing each file:

$ rm -i ch1 ch2

rm: remove ch1 (yes/no)? ? **y**

rm: remove ch1 (yes/no)? ? **n** [Enter]

A _y‘ removes the file (ch1) any other response like n or any other key leave the file undeleted.

  - **Recursive deletion (-r or -R):** It performs a recursive search for all directories and files within these subdirectories. At each stage it deletes everything it finds.

$ rm -r * #Works as rmdir

It deletes all files in the current directory and all its subdirectories.

  - **Forcing Removal (-f):**rm prompts for removal if a file is write-protected. The –f option overrides this minor protection and forces removal.

rm -rf* Deletes everything in the current directory and below

2.8.4 mv: RENAMING FILES

- The mv command renames (moves) files. The main two functions are:
- It renames a file(or directory)
- It moves a group of files to different directory
- It doesn't create a copy of the file; it merely renames it. No additional space is consumed on disk during renaming.
- Ex: To rename the file csb as csa we can use the following command

  $ mvcsbcsa

- If the destination file doesn‘t exist in the current directory, it will be created. Or else it will just rename the specified file in mv command.
- A group of files can be moved to a directory.
- Ex: Moves three files ch1,ch2,ch3 to the directory module

  $ mv ch1 ch2 ch3 module

- Can also used to rename directory

  $ mv rename newname

- mv replaces the filename in the existing directory entry with the new name. It doesn't create a copy of the file; it renames it
- Group of files can be moved to a directory

  $ mv chp1 chap2 chap3 unix

2.8.5 wc: COUNTING LINES,WORDS AND CHARACTERS

- wc command performs Word counting including counting of lines and characters in a specified file. It takes one or more filename as arguments and displays a four columnar output.

  $ wcofile

  4 20 97 ofile

- **Line:** Any group of characters not containing a newline
- **Word:** group of characters not containing a space, tab or newline.
- **Character:** smallest unit of information, and includes a space, tab and newline
- wc offers 3 options to make a specific count. –l option counts only number of lines, - w and –c

options count words and characters, respectively.

$ wc -l ofile 4 ofile

$ wc -w ofile 20 ofile

- Multiple filenames, wc produces a line for each file, as well as a total count.

$ wc -c ofile file 97 ofile 15 file

112 total

2.8.6 od: DISPLAYING DATA IN OCTAL

- od command displays the contents of executable files in a ASCII octal value.

$ moreofile

this file is an example for od command

^d used as an interrupt key

- -b option displays this value for each character separately.
- Each line displays 16 bytes of data in octal, preceded by the offset in the file of the first byte in the line.

$ od –b file

```
0000000 164 150 151 163 040 146 151 154 145 040 151 163 040 141 156 040
0000020 145 170 141 155 160 154 145 040 146 157 162 040 157 144 040 143
0000040 157 155 155 141 156 144 012 136 144 040 165 163 145 144 040 141
0000060 163 040 141 156 040 151 156 164 145 162 162 165 160 164 040 153
0000100 145 171
```

- -c character option
- Now it shows the printable characters and its corresponding ASCII octal representation

$ od –bc file

```
od -bc file
0000000 164 150 151 163 040 146 151 154 145 040 151 163
t h i s f i l e I s a n
0000020 145 170 141 155 160 154 145 040 146 157 162 040 157
e x a m p l e f o r o d c
0000040 157 155 155 141 156 144 012 136 144 040 165 163 145 144
o m m a n d \n ^ d u s e d a
0000060 163 040 141 156 040 151 156 164 145 162 162 165 160
s a n i n t e r r u p t k
0000100 145 171
e y
```

- Some of the representation:
- The tab character, [ctrl-i], is shown as \t and the octal vlaue 011
- The bell character , [ctrl-g] is shown as 007, some system show it as \a
- The form feed character,[ctrl-l], is shown as \f and 014
- The LF character, [ctrl-j], is shown as \n and 012
- od makes the newline character visible too.

Web / Video Resources

1. UNIX Architecture, Features of UNIX, Components of UNIX, POSIX standards and UNIX Environment
   https://youtu.be/jvwq9_mzafs
2. command structure, types of command and basic UNIX command

https://youtu.be/oH3j9FoDuqA

3. combining commands, root login, su, concept of files
   https://youtu.be/yjTBiAViPW0
4. HOME, PATH, Pathnames used in UNIX commands
   https://youtu.be/AXNWnK3yHB4
5. Directory Commands

https://youtu.be/GbN8jG6aFI4

6. File Related commands

https://youtu.be/X6O3CciyzTQ

REFERENCES

[1] Sumitabha Das., Unix Concepts and Applications., 4thEdition., Tata McGraw Hill

QUESTION BANK

1. Explain the UNIX architecture with a neat diagram.
2. Discuss the salient features of UNIX operating system.
3. Explain the different computing environment in UNIX.
4. Explain the UNIX structure with a neat diagram.
5. Write a note on POSIX and single UNIX specification.
6. Explain the command structure with an example.
7. Explain the following commands with an example.
   - i) echo
   - ii) printf
   - iii) cat
   - iv) who
   - v) date
   - vi) cal
   - vii) passwd

8. Discuss the difference between internal and external command.
9. Explain the root login and su command.
10. What is a file? Explain the different types of files.
11. Explain the parent-child relationship in UNIX file system.
12. Discuss HOME and PATH variable.
13. Explain the following commands with suitable example.
    i) mkdir
    ii) rmdir
    iii) cd
    iv) pwd
14. Discuss the difference between absolute and relative pathnames.
15. Explain the following commands with suitable example.
    i) cat
    ii) rm
    iii) mv
    iv) wc
    v) od
    vi) cp