UNIX PROGRAMMING

MODULE-1

History of UNIX

In 1969, Ken Thompson, Dennis Ritchie and others started work on what was to become UNIX at AT&T Bell Labs. For ten years, the development of UNIX proceeded at AT&T in numbered versions. V4 (1973) was re-written in C, a major milestone for the operating system's portability among different systems. It became the basis of the first version of UNIX developed at the University of California Berkeley.

Introduction

Operating System

OS is the software that manages the computer's hardware and provides a convenient and safe environment for running programs.

OS acts as an interface between programs and the hardware resources that these programs access.

OS is loaded into memory when a computer is booted and remains active as long as the machine is running.

UNIX is an OS

COMMAND INTERPRETER called SHELL is used in UNIX to interact with user.

Commands are keyed in the form of words, which will be interpreted by Shell.

UNIX OS is security-conscious and the persons who maintain an account with it can only use it. List of accounts is maintained by OS.

UNIX OPERATING SYSTEM

The following are the major features of UNIX Operating System

1. **Multiuser OS**
2. **Multitasking**
3. **File**
4. **Process**

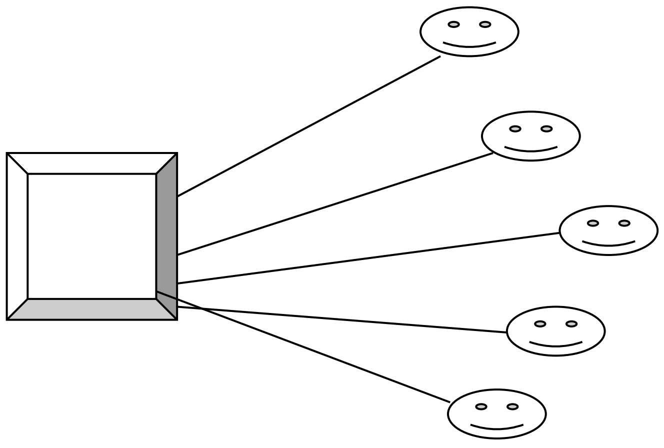
5. **Building Block Approach**
6. **Toolkit**
7. **Pattern Matching**
8. **Programming Facility**
9. **Documentation.**

The UNIX Operating System Consists of

a. **Kernel**:- Which does schedules tasks, manages data/file access and storage enforces security mechanisms and also performs all hardware access
b. **Shell**:- Presents each user with a prompt, interprets commands types by a user executes user commands and supports a custom environment for each user
c. **Utilities:-** Which contains file management (rm, cat, ls, rmdir, mkdir), user management (passwd, chmod, chgrp), process management (kill, ps) and printing (lpr)

1. **Multi-User Operating Systems**

   A multi-user operating system allows more than one user to share the same computer system at the same time. It does this by time-slicing the computer processor at regular intervals between the various users. Is as shown below.



In the above example, there are five users which share the processor hardware and main memory on a time basis.

2. **Multi-Tasking Operating Systems**

   Multi-tasking operating systems permit the use of more than one program to run at once. It does this in the same way as a multi-user system, by rapidly switching the processor between the various programs.

   A multi-user system is also a multi-tasking system. This means that a user can run more than one program at once, using key selection to switch between them. Multi-tasking systems support **foreground** and **background** tasks. A foreground task is one that the user interacts directly with using the keyboard and screen. A background task is one that runs in the background.

3. **File:-**

   The role of the operating system is to keep track of all the programs, allocating resources like disks, memory and printer queues as required. The File hierarchical directory structure, to support the organization and maintenance of files.

   A file is a collection of information, which can be data, an application and documents under UNIX, a file can contain anything. When a file is created, UNIX assigns the file a unique internal number (called an inode).

4. **Processes:-**

   Each program running on a UNIX system is called a process. When a user types a command, UNIX constructs a Process Control Block (PCB) for the process that process. Each process has a PCB that holds its priority, the process state, register information and additional details.

   UNIX provides a set of utilities for managing processes.

   Ps: list processes

   Kill: kill a process

   &: run a process in the background

5. **The building block approach:** -

   Has hundreds of commands which perform simple job, we can combine these commands with the | (pipe).

   Pipes: Sometimes, the use of intermediatary files is undesirable. Pipelining allows to connect two programs together so that the output of one program becomes the input to the next program.

- Allow commands to be combined in a sequential order
- Connects stdout of one program to the stdin of the next program
- The symbol | (vertical bar) represents a pipe
- Any number of commands can be connected in sequence, forming a pipeline

6. **The Unix toolkit:-**

   In order to ensure a consistent environment across UNIX machines, we have come up with a "standard toolkit" to deploy across all of our supported platforms. Some of these are general purpose tools, text manipulation utilities (filters), compilers, interpreters, networking, and system administration tools.

7. **Pattern matching:**

   We need patterns to describe text that is inexact, such as 'all words start with s', 'all words starting with t and ending with m', or 'all 9 digit numbers'. We might be able to list all the possible variations, but it would be impractical and annoying. We use a pattern matching language to describe a pattern. A pattern matching language uses symbols to describe both normal characters (that are matched exactly) and meta characters (that describe special operations, such as alternatives and repeating sections). Patterns are almost always described using what is called a *regular expression*.

8. **Programming facility:**

   It is also a programming language. It was designed for programmers not end users. It has control structures, loops, variables etc …, these establishes powerful programming language. These features are used to design a shell script.

9. **Documentation:**

   The principle online help facility available is the man command, which is the major reference for commands and their configuration files.

UNIX ARCHITECTURE

1. Kernel and Shell:

The kernel interacts with the machine's hardware, and shell with the user.

The kernel is the core/heart of the Unix Operating System. The collections of routines mostly written in C. these routines are communicates directly with the hardware. It is part of UNIX system that is loaded into memory when the system is booted. The operating system **kernel** is in direct control of the underlying hardware. The kernel provides low-level device, memory and processor management functions (e.g. dealing with interrupts from hardware devices, sharing the processor among multiple programs, allocating memory for programs etc.) Basic hardware-independent kernel services are exposed to higher-level programs through a library of **system calls**. The below diagram shows the kernel shell User relationship.

The shell provides with an interface to the UNIX system. It gathers input from user and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell Prompt:

The prompt, $, which is called command prompt, is issued by the shell. While the prompt is displayed then user can type a command.

The shell reads user input after user press Enter. It determines the command user want executed by looking at the first word of user input. A word is an unbroken set of characters. Spaces and tabs separate words.

i. The Bourne shell. If you are using a Bourne-type shell, the default prompt is the $ character.
ii. The C shell. If you are using a C-type shell, the default prompt is the % character.

There are again various subcategories for Bourne Shell which are listed as follows:

- Bourne shell (sh)
- Korn shell (ksh)
- Bourne Again shell (bash)

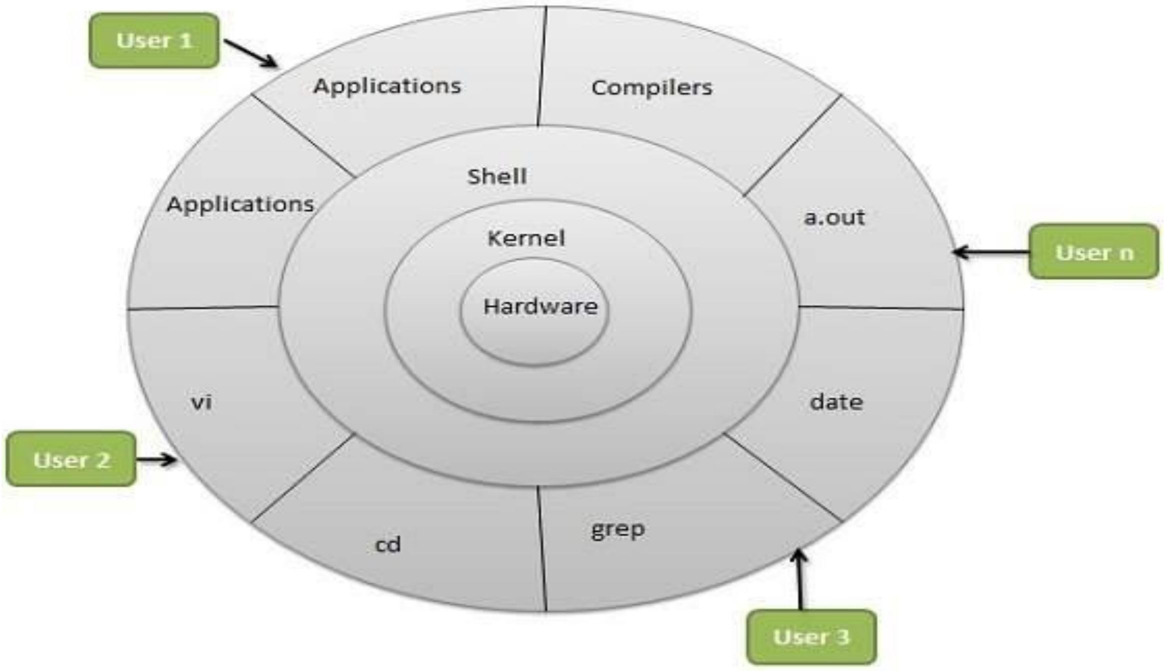


Fig: kernel shell relationship

2. System Calls:

A system call is just what its name implies a request for the operating system to do something on behalf of the user's program. The system calls are functions used in the kernel itself. For the programmer, the system call appears as a normal C function call.
However since a system call executes code in the kernel, there must be a mechanism to change the mode of a process from user mode to kernel mode. Typically UNIX commands writes file with a write() system call, similarly opens with open() system call. These system calls are built into the kernel.

UNIX COMMAND USAGE

Locating a command: -

All UNIX commands are single words like ls, cat, who etc... All these names should be in lowercase.

Example:

```
$LS
Bash: LS: command not found
```

This message is from the Shell

The easiest way to knowing the location of n executable program is to type command.

```
$type ls
ls is /bin/ls
```

Means shell locates the file in the /bin directory.

The PATH environment variable is a colon-delimited list of directories that shell searches through when user enter a command.

Program files (executables) are kept in many different places on the Unix system. Path tells the UNIX shell where to look on the system when user requests a particular program. To find out what path it is, at the UNIX shell prompt, enter:

```
$echo $PATH
```

Path will look something like the following.

```
/usr2/username/bin:/usr/local/bin:/usr/bin:.
```

Above contains Three directories separated by a colon (:) , and finally the local directory, indicated by the . (a period).

INTERNAL AND EXTERNAL COMMANDS:

Internal commands are the commands that are executed directly by the shell. These commands will not have a separate process running for each. External commands are the commands that are executed by the kernel. These commands will have a process id running for it.

Since **ls** is a program or file having an independent existence in the **/bin** directory or **/user/bin**, it is branded as an external command. Most commands are external in nature, but there are some which are not really found anywhere, and some which are normally not executed even if they are in one of the directories specified by PATH. Let's see with example-

```
$type echo
Echo is a shell built in
```

echo is not an external command in the same that, when you type echo, the system won't look in its **PATH** to locate it (even if it is there in /bin). Rather, it will execute it from its own set of built-in commands there are stored as separate files. These built in commands, of which echo is a member, are known as internal commands.

```
$type ls
ls is /bin/ls
```

We have been attributing all this hunting work to "system" as, if there is such a thing as a system. The agency that actually does all this work is knows as **shell**. Which starts running when log in , and dies when you log out.

POSIX and the Single UNIX specification

Early UNIX OS did not confine to portability characteristic of programs.

"Write once execute anywhere"

Later, Portable Operating System Interface for Computer Environments (POSIX) was developed by IEEE (Institute of Electrical and Electronics Engineers), which referred to operating systems in general based on UNIX.

Two of the most standards from POSIX family are known as POSIX.1 and POSIX.2. POSIX.1 specifies the C applications program interface - System calls. POSIX.2 deals with shell and utilities.

A joint initiative of X/Open and IEEE resulted in the unification of the two standards, termed as Single UNIX Specification, Version 3 (SUSV 3), "Write once, adopt everywhere" means once program has been developed in an POSIX compliant system, it can be executed in another system which must also be POSIX compliant with slight modifications.

COMMAND STRUCTURE:

To give a command to a UNIX system, type the name of the command, along with any associated information, such as a filename, and press the <Return> key. The typed line is called the command line and UNIX uses a special program, called the shell or the command line interpreter, to interpret what you have typed into what you want to do. The components of the command line are:

- The command;
- Any options required by the command
- The command's arguments (if required).

For example, the general form of a UNIX commands is:

Command [-option(s)] [argument(s)]

Options:

There is a special type argument that's mostly used with a – sign. Example:

```
ls –l
```

here –l is an argument of ls by definition, but it is a special type of argument called option. There must be white space between command and options. Suppose if you use command with wrong option, the shell locates the command but finds wrong option ie

```
$ls –z note
ls: illegal option - - z
```

Either we can use the options separately or combine

```
$ls –l  -a  -t
```

Is same as

```
$ls  -lat
```

Exceptions:

Exceptions are those they are not accept any argument (pwd) and some may or may not specify with any argument (who). The command ls can also use without argument.

FLEXIBILITY OF USING COMMANDS:

UNIX provides certain degree of flexibility like:

➢ Combining commands

UNIX allows combining one or more commands. Each command must be separated with ;(semicolon). For example

i.	Wc note ; ls –l note	Here we combining wc and ls –l
ii.	(wc note ; ls –l note) > new	here output is redirect to file called new

➢ A command line can overflow or be split into multiple lines

A command often keyed in but the terminal width is only 80 characters. If it exceeds then command may overflow s to the next line at this point we need to split into multiple lines. In this case shell issues Secondary prompt.

```
$echo “this is
>a three line                              >is secondary prompt
>text message
```

➢ Entering a command before previous command has finished

Subsequent commands can be entered at the keyboard without waiting for the prompt. The command entered isn’t seen by the shell as its input when it is busy running another program. The input remains stored in a buffer.

man - man formats and displays the on-line manual pages. If you specify section, man only looks in that section of the manual. Name is normally the name of the manual page, which is typically the name of a command, function, or file.

OPTIONS

-C config_file : Specify the configuration file to use; the default is /etc/man.config.

-M path : Specify the list of directories to search for man pages. Separate the directories with colons. An empty list is the same as not specifying -M at all

-B: Specify which browser to use on HTML files. This option overrides the BROWSER environment variable. By default, man uses /usr/bin/

-H: Specify a command that renders HTML files as text. This option overrides the HTMLPAGER environment variable. By default, man uses /bin/cat

-S section_list: List is a colon separated list of manual sections to search.

-a By default, man will exit after displaying the first manual page it finds. Using this option forces man to display all the manual pages that match name, not just the first.

-c Reformat the source man page, even when an up-to-date cat page exists. This can be meaningful if the cat page was formatted for a screen with a different number of columns, or if the preformatted page is corrupted.

-d Don't actually display the man pages, but do print gobs of debugging information.

-D Both display and print debugging info.

<u>GENERAL PURPOSE UTILITIES</u>

1. **cal:- D**isplays a calendar

   Synopsis : cal [[*month] year*]

Cal displays a simple calendar. If arguments are not specified, the current month is displayed. For example

```
$ cal
 March 2013
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
```

```
$ cal 01 2013
    January 2013
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 options are as follows:

-1	Display single month output. (This is the default.)
-3	Display prev/current/next month output.
-s	Display Sunday as the first day of the week. (This is the default.)
-m	Display Monday as the first day of the week.
-y	Display a calendar for the current year.

Date: displays system date

For example:

```
$ date
Sun Mar  3 01:46:44 IST 2013

$ date +%m
03

$ date +%h
Mar

$ date +"%h %m"
Mar 03
```

2. **echo:** This command is used to display messages on the terminal, we can use in two ways:

- To display a message (like echo some text)
- To evaluate shell variables(like echo $SHELL)

It is an external command.

Escaping sequence generally two character string beginning with a backslash (\ ). For example

```
$echo "Enter filename: \c"

Enter filename:  $  -                    prompt and cursor in same line
```

Also escape sequence can represent octal, number etc for example

```
$echo '\07'

....beep heard...
```

Below table shows the escape sequence used by echo and printf

Escape sequence	meaning
\\	Backslash
\v	**Vertical tab**
\t	**Horizontal tab**
\r	carriage return
\n	new line
\f	form feed
\c	suppress trailing newline
\b	backspace
\a	alert (BEL)

3. **printf :** An alternative to echo

```
$printf "no file \n"

No file

$ _

$printf "my current shell is %s\n" $SHELL                 No comma
```

```
my current shell is /bin/bash

$ _
```

By using printf we can convert data from one form to another is as shown below example

```
$printf ”The val of 255 is %o in octal and %x in hexadecimal\n” 255 255

The val of 255 is 377 in octal and ff in hexadecimal
```

4. bc: The calculator

When we invoke bc without argument, the cursor keeps on blinking and nothing seems to happen until keyed something. Use [ctrl-d] to quit.

```
$bc
12+5
17[ctrl-d]
$ _
```

bc performs only integer computation and truncates the decimal point.
ie 9/5 to 1
To enable floating point then use scaling ie

```
Scale=2                         truncates to 2 decimal places
17/7
2.42
```

Can also coverts binary to decimal ie

```
Ibase=2
11001010
202                             output decimal base-10
```

Reverse is also possible

```
Obase=2
14
1110                            binary of 14
```

5. ***Script***

script - make typescript of terminal session . Script makes a typescript of everything printed on your terminal. It is useful for students who need a hardcopy record of an interactive session as proof of an assignment, as the typescript file can be printed out later with lpr(1). If the argument file is given, script saves all dialogue in file. If no file name is given, the typescript is saved in the file typescript. For example

```
$ script

Script started, file is typescript                /default name

To exit

$ exit

Script done, file is typescript            /back to login shell

$ _
```

6. **Change Password:**

All Unix systems require passwords to help ensure that users files and data remain your own and that the system itself is secure from hackers. Here are the steps to change your password:

1. To start, type **passwd** at command prompt as shown below.

2. Enter your old password the one you're currently using.

3. Type in your new password. Always keep your password complex enough so that no body can guess it. But make sure, you remember it.

4. You would need to verify the password by typing it again.

```
$ passwd

Changing password for anjan

(current) Unix password:******
```

```
New UNIX password:*******

Retype new UNIX password:*******

passwd: all authentication tokens updated successfully

$
```

7. **who** - show who is logged on, Print information about users who are currently logged in.

options

-a, --all	same as -b -d --login -p -r -t -T -u
-d, --dead	print dead processes
-H, --heading	print line of column headings
-l, --login	print system login processes

```
$ who

anjan    tty1        2013-03-02 04:05 (:0)

anjan    pts/0       2013-03-02 04:08 (:0.0)

$ who -Hu

NAME     LINE        TIME                    IDLE          PID            COMMENT

anjan    tty1        2013-03-02 04:05       old           1731                 (:0)

anjan    pts/0       2013-03-02 04:08       .              2246                 (:0.0)
```

. implies it started just one minute before the command was invoked.

```
$ who am i

anjan    pts/0       2013-03-02 04:08 (:0.0)
```

8. **tty** - print the file name of the terminal connected to standard input

```
$ tty
/dev/pts/0
```

stty - change and print terminal line settings, Print or change terminal characteristics.

-a, --all	print all current settings in human-readable form
-g, --save	print all current settings in a stty-readable form
-F, --file=DEVICE	open and use the specified DEVICE instead of stdin

9. **Sty: display and setting terminal characteristics**

```
$ stty
speed 38400 baud; line = 0;
eol = M-^?; eol2 = M-^?; swtch = M-^?;
ixany iutf8
$ stty -a
speed 38400 baud; rows 39; columns 125; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = M-^?; eol2 = M-^?; swtch = M-^?; start
= ^Q; stop = ^S;  susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts -ignbrk brkint -ignpar -parmrk -inpck -
istrip -inlcr -igncr icrnl ixon -ixoff -iuclc ixany imaxbel iutf8 opost -olcuc -ocrnl onlcr -onocr -
onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0 isig icanon iexten echo echoe echok -echonl -noflsh -
xcase -tostop -echoprt echoctl echoke
```

10. echo

NOTE: In Bash shell to see the effect of escape sequences echo command must be coded with -e option

Ex: echo -e “Hai,\t it is over\n”

ASCII characters can also be represented by their octal values. echo interprets a number as octal when it is preceded by \0.

Ex: echo -e ‘\07’

Generates a beep sound.

Escape Sequence	Significance
\a	Bell
\b	Backspace
\c	No newline
\f	Form feed
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\\	Backslash
\0n	ASCII character represented by the octal value n, where n can’t exceed 0377 (decimal value 255)

UNIX FILE SYSTEM

Introduction to the Unix File System:-

The UNIX file system is a methodology for logically organizing and storing large quantities of data such that the system is easy to manage. A **file** can be informally defined as a collection of data, which can be logically viewed as a stream of bytes A file is the smallest unit of storage in the Unix file system.

File Types: UNIX files can be one of the following type.

1. ***Regular file***
2. ***Directory***
3. ***FIFO file***
4. ***Device file***
5. ***Symbolic link file***
6. ***Socket file***

Regular file : A regular file contains data, it can be text or binary files. They are executable. If we want create regular file use any of the following.

Create: vi, cat, ex etc…

If we want delete regular file use of the following.

Remove: rm

Directory: The most common special file is the directory. The layout of a directory file is defined by the filesystems used. As several filesystems, both native and non-native, are available under UNIX, there is not one directory file layout.

A directory is marked with a **d** as the first letter in the mode field in the output of `` `ls -l ``

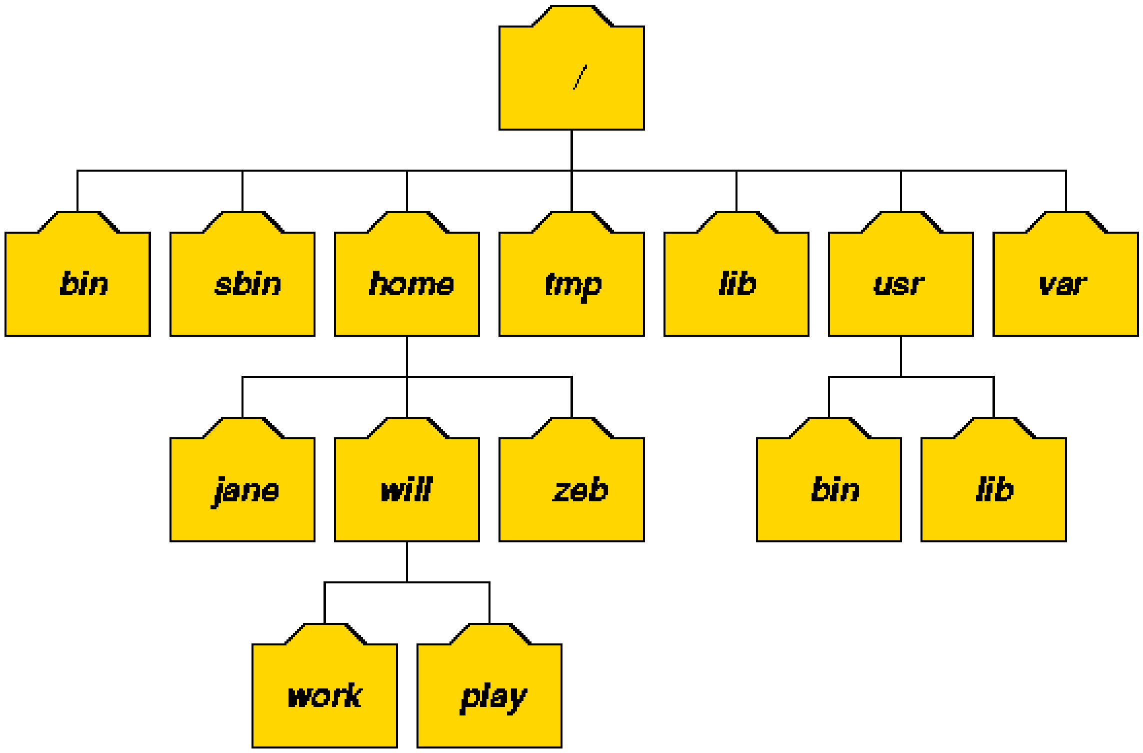
If we want create directory file use the following.

Create: mkdir

If we want delete regular file use of the following.

Remove: rmdir

Below figure illustrates layout of directory file.



Device file: these file names are generally found inside a single directory structure,/dev. They are two types of device files.

To create: mknod devicename majornumber minornumber

1. Character device file
2. Block device file

Character device file: devices that transfer data on a byte-by-byte basis (e.g. modems and printers). Specified by ‘c’.

To create :

```
mknod   c     100      120
```

Block device file: devices which transfer data in blocks (e.g. hard disks), specified by the symbol ‘b’

To create:

```
mknod   b     100       120
```

FIFO File: Is a special pipe device file which provides buffer for two or more processes to communicate by writing data to and reading data from buffer. It also supports IPC(Inter process communication)

Creation : mkfifo <file name>

Symbolic link file: It contains pathname and also it reference to another file.

Creation : ln –s existingfile   newfile

FILE NAME: can contains up to 255 characters . may or may not have extensions. Can consists of any ASCII characters(except /) and NULL character. Names are very much case sensitive. The file name may contain

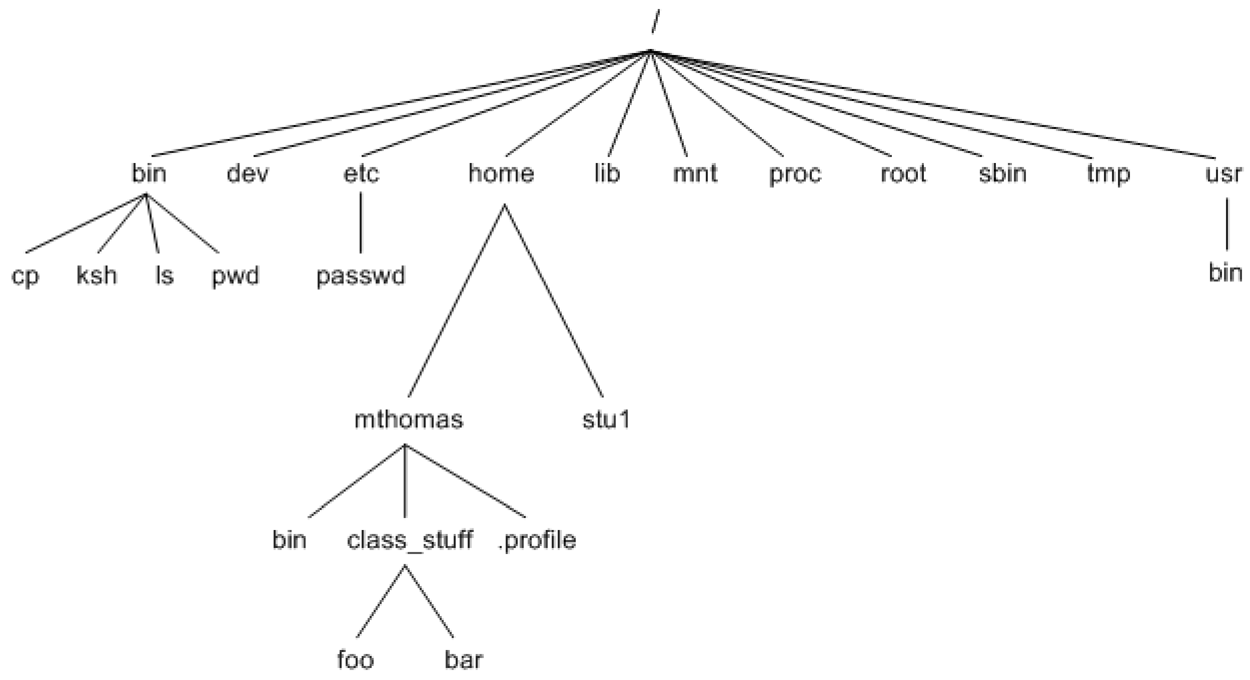
- Alphabets and numbers
- The period(.), hyphen(-) and underscore(_)
- Can have many dots embedded in it

THE PARENT CHILD RELATIONSHIP**:** The UNIX filesystem is laid out as a hierarchical tree structure which is anchored at a special top-level directory known as the root (designated by a slash '/'). Because of the tree structure, a directory can have many child directories, but only one parent directory. Is as shown below

The root directory has number of subdirectories under it, or more files under it. From above example

- Under root subdirectory called home and sharma
- Under home subdirectory called kumar
- Under kumar subdirectory called progs , safe and file called login.sql

Above kumar is the parent for login.sql, progs and safe, home is the parent for kumar so on



1. **The home variable:** The home directory

When log on to unix system, unix automatically place in directory called home directory. It is created by the system when the user account opened. If user log in as anjan then

```
$ echo $HOME
/home/anjan                    /  represents the root directory
```

The above is the absolute pathname, which is simply sequence of directory names separated by slashes.(/)

2. **Pwd: checking your current directory**

It will give in which directory you are working with, because in UNIX you can move around different directories.

```
$ pwd
/home/anjan                    anjan is your current directory
```

3. **cd: This command changes your current directory location. By default, your Unix login session begins in your home directory.** To switch to a subdirectory (of the current directory) named myfiles, enter:

```
cd myfiles
```

To switch to a directory named /home/dvader/empire_docs, enter:

cd /home/dvader/empire_docs

To move to the parent directory of the current directory, enter:

cd ..

To move to the root directory, enter:

cd /

To return to your home directory, enter:

Cd

4. **mkdir:** This command will make a new subdirectory. To create a subdirectory named mystuff in the current directory, enter:

mkdir mystuff

To create a subdirectory named morestuff in the existing directory named /tmp, enter:

mkdir /tmp/morestuff

Note: To make a subdirectory in a particular directory, you must have permission to write to that directory.

5. **Rmdir:** This command will remove a subdirectory. To remove a subdirectory named oldstuff, enter:

rmdir oldstuff

Note: The directory you specify for removal must be empty. To clean it out, switch to the directory and use the ls and rm commands to inspect and delete files.

6. **mv: Move Files and Rename Files**

This command performs two distinct functions.

It renames a file (or directory)   It moves a group of files to a different directory.

mv will not create a copy of file, it merely renames it. No additional disk space is consumed during renaming.

$ mv t.txt   T.TXT

If the destination file does not exist, it will be created. By default, mv does not prompt for overwriting the destination file if it exists.

Group of files can be moved to a directory.

```
$ mv  t.txt  t1.txt  t2.txt  progs      3 files will be moved to directory named progs
```

Directory name can also be renamed using mv

```
$mv  progs  programs       considering progs as directory name, it will be renamed as programs
```

7. **rm: Deleting Files**

rm command is used to delete one or more files. It will not prompt user before deletion, hence must be used with utmost care.

```
$ rm t.txt  t1.txt  t2.txt        deletes the 3 files that are mentioned in the command
```

A file once deleted cannot be recovered.

File name in rm command can be prefixed with path too in order to delete files.

```
$ rm  test/t.xt   test/t1.txt    test/t2.txt
```

All files in a directory can be deleted at once $ rm *

Deleting files from a directory, depends on the permission that is associated with the directory for a particular user.

rm

Interactive Deletion (-i)

-i option associated with rm, prompts for user confirmation before a file is deleted.

```
$ rm -i t.txt

remove t.txt ?
```

Response as y, removes the file, any other response leaves the file un-deleted.

Recursive Deletion (-r)

Usually, rm command will not remove subdirectory within the PWD, but by using option -r, subdirectories can also be deleted.

```
$ rm -r  *    deletes all files and subdirectories in PWD, without any prompt
```

```
$ rm -i -r *  prompts the user upon each file/subdirectory deletion.
```

Rm : **Forcing removal (-f)**

Attempt to remove the files without prompting for confirmation, regardless of the file's permissions.

If the file does not exist, do not display a diagnostic message or modify the exit status to reflect an error.

```
$ rm -f res.txt // assume res.txt does not exist

$ echo $?    // to print the return value of the previous process

1

$ rm res.txt ; echo $?

0
```

-f option overrides any previous -i options.

```
$ rm -i -f  t      file ‘t‘ will be deleted without prompting user for deletion.
```

NOTE: Make sure before rm * is used, be doubly sure before using rm -rf *. The first command removes only ordinary files in pwd. The second one removes everything - files and directories alike.

8. **cp: Copying a File**

cp command copies a file or a group files. It creates an exact image of the file on disk with a different name.

Syntax requires at least two file names in the command line. When both are ordinary files, the first is copied to the second.

```
$ cp   src.txt   dest.txt    contents of src.txt will be copied to dest.txt
```

If destination file does not exist, it will first be created before copying.

If destination file exists, its contents will be overwritten without any warning.

If there is only one file to be copied, the destination can be either a file in a directory (wherein the file mentioned will have path associated with it) or directory. If directory name is mentioned source file will be copied to the mentioned directory.

```
$ cp src.txt    test1/dest.txt       dest.txt will be created in test1 and src.txt contents gets copied

$ cp src.txt   test1                 src.txt file will be copied to test1 subdirectory
```

10. **The root login**

root user / superuser / administrator, is a special user account in UNIX, used for system administration.

It is the most privileged user on the Linux system and it has access to all commands and files.

root user can do many things an ordinary user cannot, such as installing new software, changing the ownership of files, and managing other user accounts.

It is not recommended to use **root** for ordinary tasks, such as browsing the web, writing texts, e.g. A simple mistake can cause problems with the entire system, for example if you mistype a command.

It is advisable to create a normal user account for such tasks.

If root permissions are needed, the *su* and *sudo* commands can be used.

su Command

su command, which is short for *substitute user* or *switch user*, enables the current user to act as another user during the current login session.

Syntax: su [*options*] [*username*]

If no username is specified, **su** defaults to becoming the superuser (root).

user will be prompted for a password, if password is appropriate, login will be successful. Invalid passwords produce an error message.

Ex: Consider two users in a unix system termed as ‘dos’ and ‘unix’, if currently a user has logged in as ‘dos’, he can change his identity using su command to ‘unix’, provided the password of unix is known to him.

```
dos$ su unix

Password: *****

unix$  lsa
```

wc: Counting Lines, Words and Characters

```
$ wc infile

2      5     19  infile
```

A **line** is any group of characters not containing a newline.

A **word** is a group of characters not containing a space, tab or newline.

A **character** is the smallest unit of information, and includes a space, tab and newline.

wc offers 3 options to make a specific count.

-l option to count only number of lines	wc -l infile
-w option to count only number of words	wc -w infile
-c option to count only number of characters	wc -c infile

When used with multiple filenames, wc produces a line for each file, as well as total count.

```
$ wc infile  t   c

4    5    6   infile

6    7    8   t

3    4   10   c
```

od: Displaying data in octal (Octal Decimal hex ASCII dump)

Files in UNIX containing non-printing characters can be displayed using this command, od command prints these characters in readable form (ASCII octal values).

```
$ cat > infile

I

Am the

wc command^d          file infile is made up of escape sequences

$ od infile

0000000   005151  066541  072040  062550  073412  020143  067543  066555
```

```
0000020    067141  000144

0000023
```

Each line displays 16 bytes of data in octal, preceded by the offset in the file of the first byte in the line.

In order to understand the output -b and -c arguments will be used.

```
$ od -bc infile

0000000   151 012 141 155 040 164 150 145 012 167 143 040 143 157 155 155

            i   \n   a   m        t   h   e   \n   w   c        c   o   m   m

0000020   141 156 144

            a   n   d

0000023
```

Each line is having two displays.

Octal representation are displayed in the first line.

The printable characters and escape sequences are displayed in the second line.

-b for displaying octal bytes

-c for c-style escape characters.

ABSOLUTE AND RELATIVE PATH

What is an absolute path?

An absolute path is defined as the specifying the location of a file or directory from the root directory(/). In other words we can say absolute path is a complete path from start of actual filesystem from / directory.

Some examples of absolute path:

/var/ftp/pub

/etc/samba.smb.conf

/boot/grub/grub.conf

If you see all these paths started from / directory which is a root directory for every Linux/Unix machines.

What is the relative path?

Relative path is defined as path related to the present working directory(pwd). Suppose I am located in /var/log and I want to change directory to /var/log/kernel. I can use relative path concept to change directory to kernel

changing directory to /var/log/kernel by using relative path concept.

pwd

/var/log

cd kernel

Note: If you observe there is no / before kernel which indicates it's a relative directory to present working directory.

Changing directory to /var/log/kernel using absolute path concept.

cd /var/log/kernel

Example1: Present location is /abc/xyz, I am want to remove /abc/xyz/read/hello.txt file.

Using relative path:

rm read/hello.txt

Using absolute path:

rm /abc/xyz/read/hello.txt

Using . and .. in relative Pathname

An invisible file is one whose first character is the dot or period character (.). UNIX programs (including the shell) use most of these files to store configuration information.

- ✓ Single dot **.**: This represents current directory.
- ✓ Double dot **..**: This represents parent directory.

Assume that you are placed in /home/anjan/progs/data/text, then use .. cd to move parent directory.

```
$pwd
/home/anjan/progs/data/text                              Moves one level up
$cd ..
/home/anjan/progs/data
```

If you use cd ../.. then

```
$pwd
/home/anjan/progs/data
$cd ../..                                                Moves two level up
$pwd
/home/anjan
```

Shell allows us to access the current directory and the parent directory by using the single dot and the double dots respectively.

Relative path also uses these dots to represent the current directory and the parent directory respectively. With the use of these dots, relative path can be built for any file or directory from the current directory.

Ex:

```
$ pwd
/home/kumar/kumar1
$ ls ../../../
lists the files present in root directory
```

Ex:

```
$ pwd
/home/kumar
$ cat > ./kumar1/t.txt
ABCD^Z
$ cat > ./kumar1/t.txt
ABCD
```

ls: listing directory contents

The **ls** command lists all files in the directory that match the *name*. If name is left blank, it will list all of the files in the directory.

The syntax for the **ls** command is: ls [options] [names]

Either we can use with or without arguments.

Example without arguments:

```
$ls
09_packets.html
ABC.sh
Program1.c
Progs
```

Here it displays in the order numeric, uppercase and then lowercase.

Option	**Description**
-a	Displays all files.
-b	Displays nonprinting characters in octal.
-c	Displays files by file timestamp.
-C	Displays files in a columnar format (default)
-d	Displays only directories.
-f	Interprets each *name* as a directory, not a file.
-F	Flags filenames.
-g	Displays the long format listing, but exclude the owner name.
-i	Displays the inode for each file.
-l	Displays the long format listing.
-L	Displays the file or directory referenced by a symbolic link.

-m	Displays the names as a comma-separated list.
-n	Displays the long format listing, with GID and UID numbers.
-o	Displays the long format listing, but excludes group name.
-p	Displays directories with **/**
-q	Displays all nonprinting characters as **?**
-r	Displays files in reverse order.
-R	Displays subdirectories as well.
-t	Displays newest files first. (based on timestamp)
-u	Displays files by the file access time.
-x	Displays files as rows across the screen.
-1	Displays each entry on a line.

ls options

output in multiple columns(-x) : if we have multiple files better to display the file names in multiple columns,to this we need to use ls –x option.

Identifying directories and executables (-F) : to identify directories and executable files we use this option. We can combine this with –x option also .

Showing hidden files (-a) : ls doesn't list all files in a directory .There are certain hidden files found in home directory.to display this use –a option.

Listing directory contents : to list two different directory contents ,for example dir1,dir2

ls –x dir1 dir2 : it displays directories and files of the both.

Recursive Listing :the –R option displays files and sub directories in a directory tree. This is done recursively until there are no sub directories left.

THE UNIX FILE SYSTEM :

UNIX file system contains the following

- /bin and /usr/bin – these are the directories where all the commonly used unix commands found.
- /sbin and /usr/sbin – the directories that only system administrator can access will be stored in these.
- /etc – this directory contains configuration files of the system.
- /dev – this directory contains all device files.
- /lib and /usr/lib – this directory contains all library files.
- /usr/include – this directory contains all header files used in c program.
- /usr/share/man – this is where the man pages are stored.

User also work with their own files .these files are available in the following way

- /tmp – the directories where users are allowed to create temporary files.
- /var – the variable part of the file system.contains all print jobs and outgoing and incoming mail.
- /home – on many systems users are housed.