Module - I

1. WHAT IS AN OPERATING SYSTEM (OS)?

An operating system (OS) is an interface between hardware and user. It manages hardware and software resource. It takes the form of a set of software routines that allow users and application programs to access system resources (e.g. the CPU, memory, disks, modems, printers, network cards etc.) in a safe, efficient and abstract way.

For example, an OS ensures safe access to a printer by allowing only one application program to send data directly to the printer at any one time. An OS encourages efficient use of the CPU by suspending programs that are waiting for I/O operations to complete to make way for programs that can use the CPU more productively. An OS also provides convenient abstractions (such as files rather than disk locations) which isolate application programmers and users from the details of the underlying hardware.

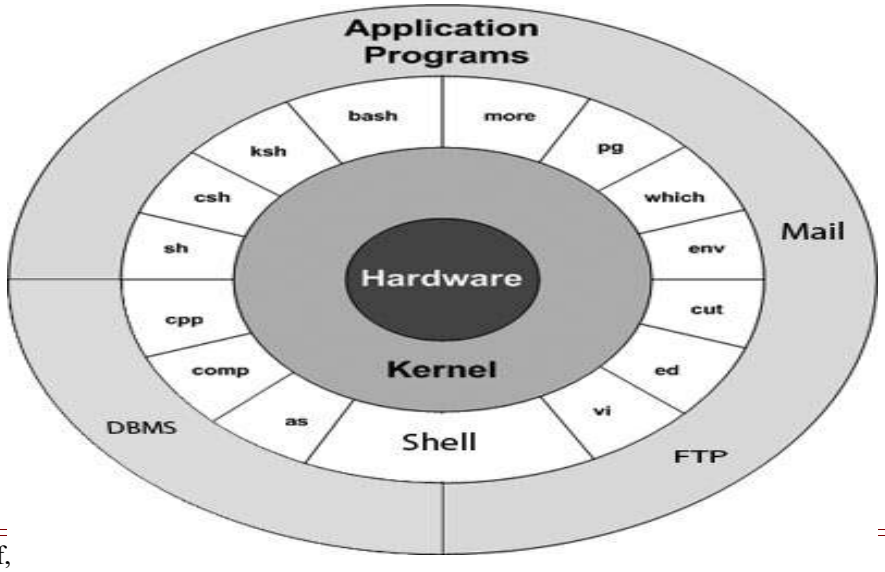
2. BRIEF HISTORY

In the late 1960s, researchers from General Electric, MIT and Bell Labs launched a joint project to develop an ambitious multi-user, multi-tasking OS for mainframe computers known as MULTICS (Multiplexed Information and Computing System). MULTICS failed, but it did inspire Ken Thompson, who was a researcher at Bell Labs, to have a go at writing a simpler operating system himself. He wrote a simpler version of MULTICS on a PDP7 in assembler and called his attempt UNICS (Uniplexed Information and Computing System). Because memory and CPU power were at a premium in those days, UNICS (eventually shortened to UNIX) used short commands to minimize the space needed to store them and the time needed to decode them - hence the tradition of short.

The limitation of UNICS was not portable.In oder to overcome the limitation , Ken Thompson started to work on the development of system using higher level language called B Language.

As B language did not yield expected results,Dennis ritchie developed higher level language called C .Ken Thompson then teamed up with Dennis Ritchie, the author of the first C compiler in 1973. They rewrote the UNIX kernel in C - this was a big step forwards in terms of the system's portability - and released the fifth Edition of UNIX to universities in 1974.

3. UNIX ARCHITECTURE



3.1 Kernel: is the core of operating system. A collection of routines mostly written in C.
It is loaded into memory when the system is booted and communicates directly with the hardware. The kernel manages system memory, processes, decides priorities.

3.2 Shell: interface between Kernel and User. It functions as **command interpreter** i,e it receives and interprets the command from user and interacts with the hardware. There is only one kernel running on the system, there could be several shells in action- one for each user who is logged in.

3.3 Files and Process: file is an array of bytes and it contain virtually anything. Unix considers even the directories and devices as members of file system. The dominant file type is text and behavior of system is mainly controlled by text files.
The second entity is the process, which is the name given to a file when it is executed as a program. Process is simply a time image of an executable file.

3.4 System Calls: Though there are thousands of commands in the unix system, they all use a handful of functions called system calls. User programs that need to access the hardware use the services of the kernel, which performs the job on users behalf. These programs access the kernel through a set of functions called system calls.

Ex: open()-- system call to access both file and device. Write()—system call to write a file.

4. FEATURES OF UNIX

Several features of UNIX have made it popular. Some of them are:

- **Portable:** UNIX can be installed on many hardware platforms. Its widespread use can be traced to the decision to develop it using the C language. Because C programs are easily moved from one hardware environment to another, it is relatively simple to port it to different environments.
- **Multiuser:** The UNIX design allows multiple users to concurrently share hardware and software
- **Multitasking:** UNIX allows a user to run more than one program at a time. In fact more than one program can be running in the background while a user is working foreground.
- **Networking:** While UNIX was developed to be an interactive, multiuser, multitasking system, networking is also incorporated into the heart of the operating system. Access to another system uses a standard communications protocol known as Transmission Control Protocol/Internet Protocol (TCP/IP).
- **Organized File System:** UNIX has a very organized file and directory system that allows users to organize and maintain files.
- **Device Independence:** UNIX treats input/output devices like ordinary files. Input or output to a program can be from any device or file.The source or destination for file input and output is easily controlled through a UNIX design feature called redirection.
- **Utilities:** UNIX provides a rich library of utilities that can be use to increase user productivity.
- **Services:** UNIX also includes the support utilities for system administration and control.

5. THE UNIX ENVIRONMENTS AND UNIX STRUCTURE

There 3 different environments in UNIX

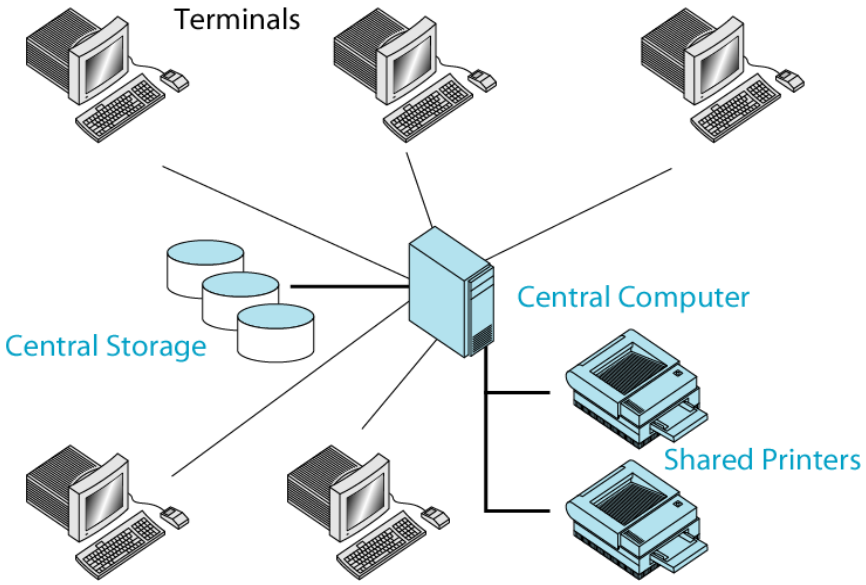
5.1 Personal environment

5. 2. Timesharing environment: Many users connected to one computer

5.3. Client/server environment Computing split between a central computer (server) and users' computers (clients)

5.1 Personal environment originally unix designed as a multiuser environment, many user user are installed UNIX on their personal computers this tends to personal unix system environment

5.2 Timesharing environment

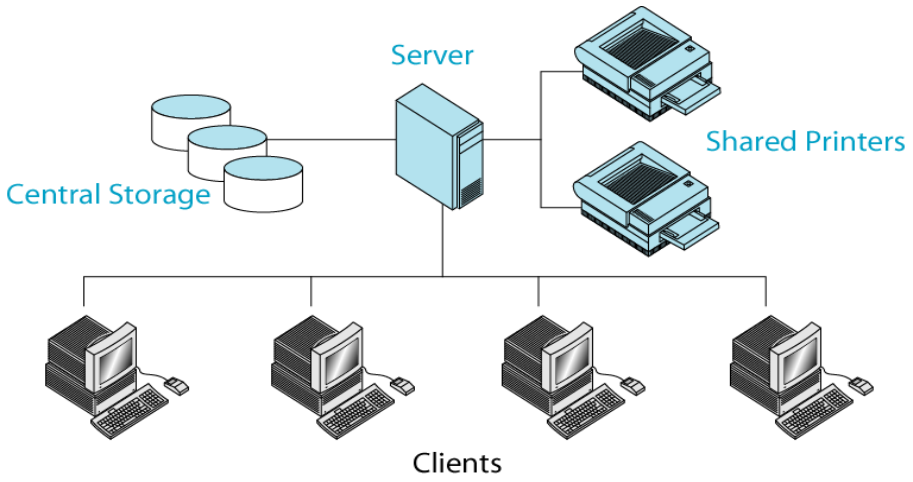


In a time sharing environment, many users are connected to one or more computers. The output devices (such as printers) and auxiliary storage devices (such as disks)are shared by all users .In this environment all of the computing must be done by the central computer. The central computer has several responsibilities

- It must control the shared resources.
- It must manage the shared data
- It must manage the printing data and resource.
- It must handle the computing all task

All of this work tends to keep the central computer busy so, user has to wait more time for get done their work so, it is nonproductive because of slow response.

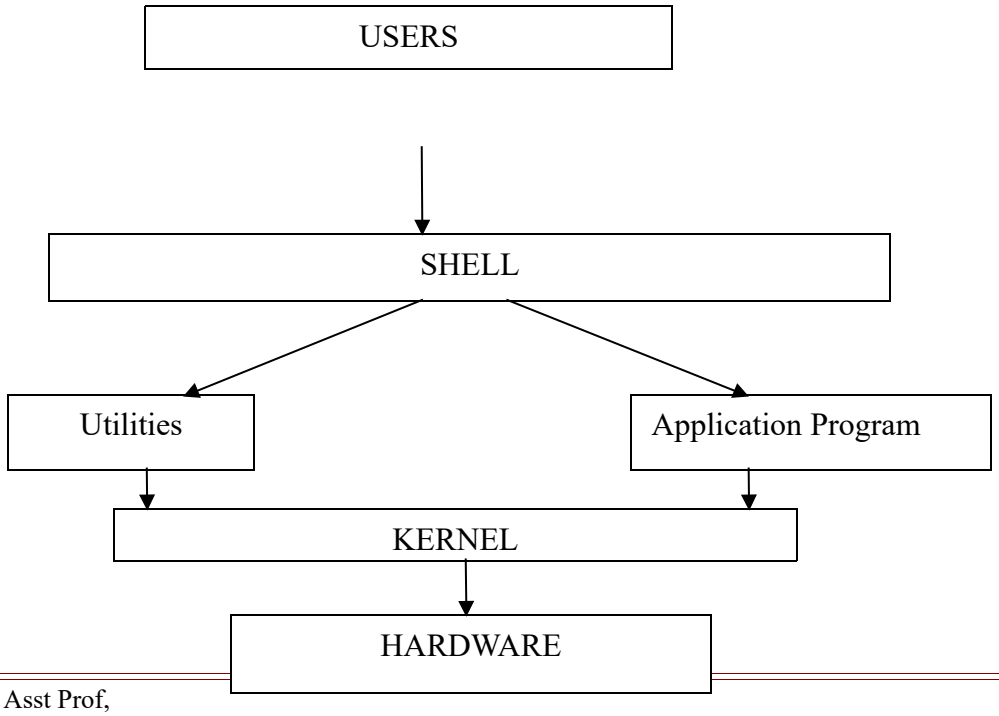
5.3 The Client/Server Environment



- In client server environment splits the computing function between a central computer and users computers.
- User computer are personal computer or workstation central computer assigned to the workstations. In client server environment the users computer or workstation is called client and central computer is called server.
- The central computer , which may be a powerful microcomputer a minicomputer, a central mainframe system is known as server.
- Since work is shared between users computer and the central computer, response time and monitor display are faster and users are more productive.

6.UNIX STRUCTURE

Unix consists of four major components: the kernel, the shell, a standard set of utilities and application programs.



6.1 Kernel: is the heart of UNIX system. It contains two basic parts of the OS: process control and resource management. All other components of the system call on the kernel to perform these services for them.

6.2 Shell: interface between Kernel and User. It functions as **command interpreter** i,e it receives and interprets the command from user and interacts with the hardware. There is only one kernel running on the system, there could be several shells in action- one for each user who is logged in. Shell has two major parts.

a. Interpreter: reads your commands and works with the kernel to execute them.

b. Shell Programming: is a programming capability that allows you to write a shell scripts.

A shell script is a file that contains the shell commands that perform a useful function. It is also known as shell program.

There are three standard shells used in UNIX today.

- **Bourne shell:** developed by steve bourne at AT&T labs, is the oldest.
- **Bash**(Bourne Again shell): An enhanced version of the Bash shell.
- **C shell**: developed in the Berkeley by Bill joy, Its commands look like C statements.
- **Tcsh**: A compatible version of C shell.
- **Korn shell:** developed by David Korn, also of AT&T Labs is the newest and powerful.

6.3 Utilities: A utility is a standard Unix program that provides a support for users. Three common utilities are text editors, search programs and sort programs.

6.4 Applications: are programs that are not a standard part of UNIX. Written by system administrator's professional programmers or users they provide an extended capability to the system.

7. POSIX AND SINGLE UNIX SPECIFICATION

Dennis Ritchie's decision to rewrite UNIX in C didn't quite make UNIX very portable. UNIX fragmentation and absence of a single conforming standard adversely affected the development of portable applications. First ,AT &T created the System V Interface Definition(SVID). Later, X/Guide(XPG). Products conforming to this specification were branded UNIX95, UNIX98 or UNIX03 depending on the version of the specification.

Yet another group of standards, the portable operating system interface for computer environments(POSIX), were developed at the behest of the Institution of Electrical and Electronics Engineers(IEEE). POSIX refers to operating systems 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 C application program interface the system calls. POSIX.2 deals with 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- compliant UNIX machine with minimum modifications. We make reference to POSIX throughout this text, but these references should be

interpreted to mean the SUSV3 as well.

8. ACCESSING UNIX

There are three methods for accessing the unix

- User ID
- Passwords
- Interactive session

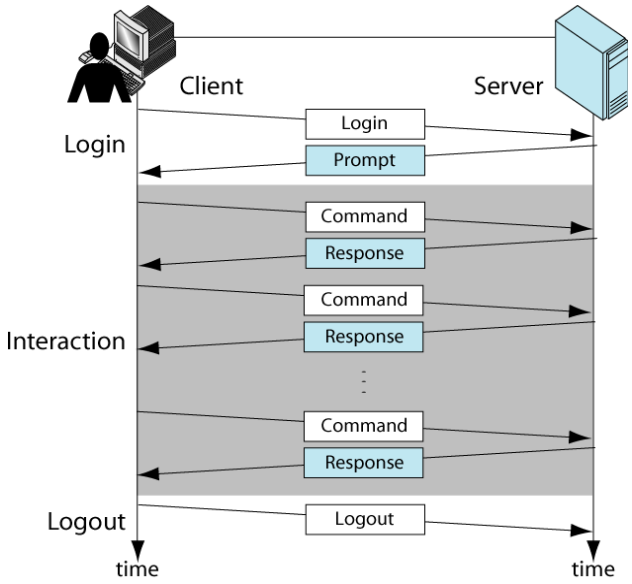
8.1 User ID when you work with your own computer at home, you do not need to log in or be too concerned about who uses the system. When you work in a UNIX environment, security and user control becomes major concerns. Permission comes in the form of account created by sytem admin. you and your account are identified by a special code known as **User id**

8.2 Passwords: A **password** is a word or string of characters used for user authentication to prove identity or access approval to gain access to a resource (example: an access code is a type of password), which is to be kept secret from those not allowed access.

A good password has atleast 6 characters.

It should contain both upper and lower case letters along with atleast one digit and special character.

8.3 Interactive session: contains three steps login , Interaction and logout.



8.3.1 Login

The details of Login process varies from system to system. There is a general pattern to the steps listed below.

a. you must make contact with the system.

If you are working on a local network, always connected to remote server starting the login process is simple as selecting option in a menu.

If you are making the connection from a remote location such as from home to work then you need o use special software for connection such as Telnet.

b. wait for the system login prompt.

login:

c.Type User Id

user id:
d. Type Password.
Password:

8.3.2 Interaction:

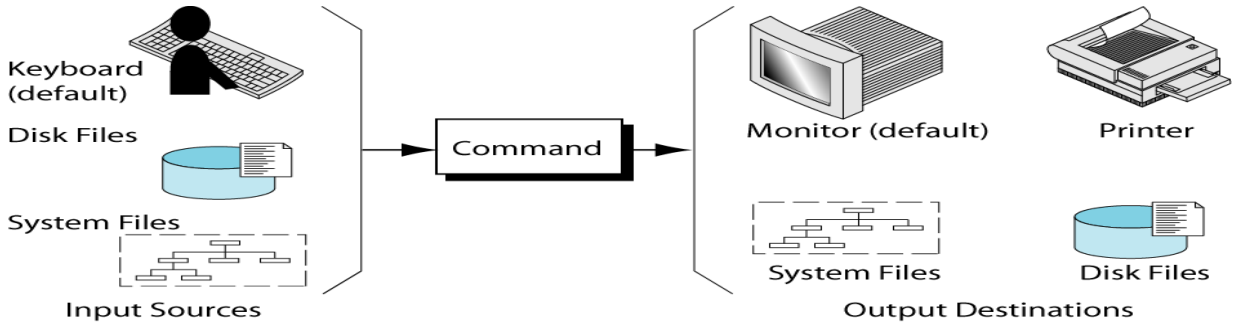
Once you connect to server you can enter commands that allow you to work with the computer.Tyical commands allow you to work with the files-edit copy, sort, process data and print files;send and receive mails etc..

8.3.3 Logout:

Its important to log out when you are done with your work .There are several reasons. First it frees system resources for others who may need to use them.Second it is a security concern.

Typical User Session:
login:gilberg
password:***********
$ ls
File1 file 2…….
$**logout**

9. GENERAL FEATURES OF UNIX COMMANDS/ COMMAND STRUCTURE

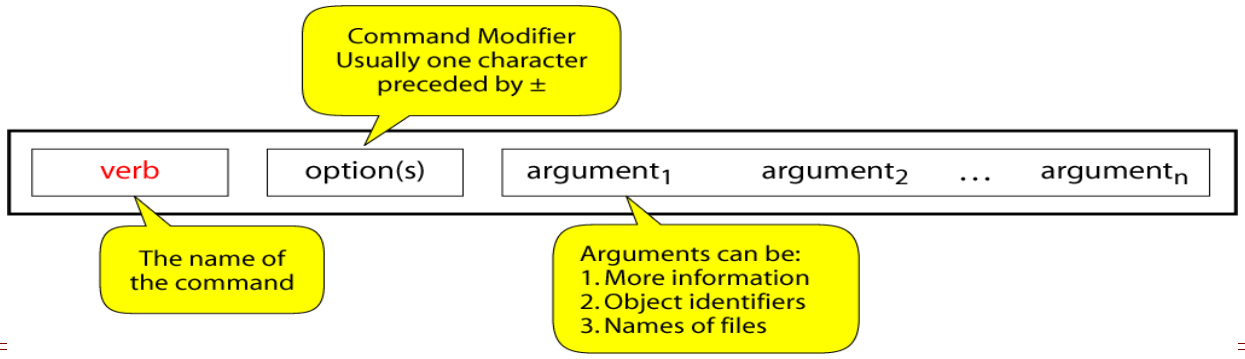


Commands are entered at shell prompt.The components of the command line are:

- the verb;
- any options required by the command
- the command's arguments (if required).

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

```
$verb       [-option(s)]          [argument(s)]
```



9.1 Verb: is the command name. The command indicates what action is to be taken. This action concept gives us the name verb for action .

9.2 option: modifies how the action is applied.

9.3 argument: provides additional information to the command.

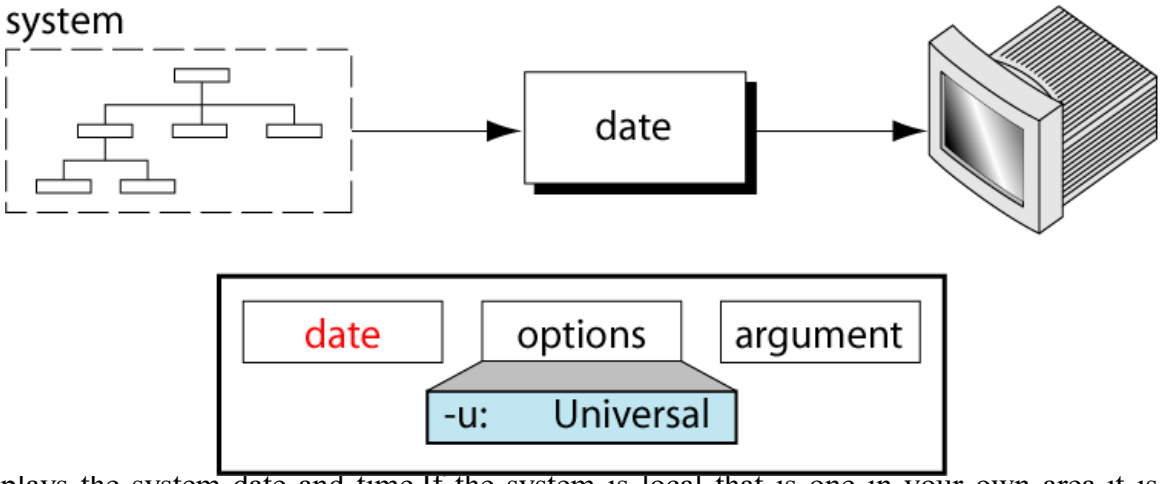
Note: Options MUST come after the command and before any command arguments. Options SHOULD NOT appear after the main argument(s). However, some options can have their own arguments

if **options** are enclosed within the [] then options are not mandatory else it is compulsory

if **arguments** are enclosed within the [] then options are not mandatory else it is compulsory

10. UNDERSTANDING OF SOME BASIC COMMANDS SUCH AS echo, printf, ls, who, date, passwd, cal

10.1 THE DATE COMMAND:



date: displays the system date and time.If the system is local that is one in your own area-it is the current time.If the system is remote, such as across the country the reply will contain the time where the system is physically located.

The input for the date is the system itself.The date is actually maintained in the computer as a part of OS.The date command sends its response to monitor.

$date

```
Sun Aug 28 13:28:39 IST 2016
```

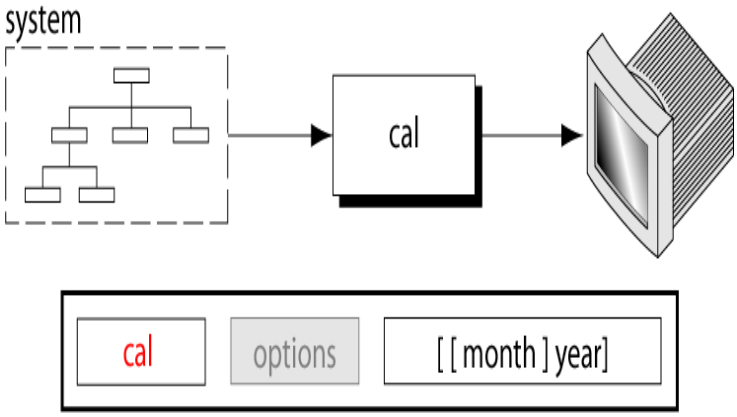
$date -u

```
Sun Aug 28 08:02:13 UTC 2016
```

```
$ date "+Today's date is %D and Time is %T"
Today's date is 08/28/16 and Time is 13:33:57
```

Format code	Explanation
a	Abbreviated weekday name such as Mon
A	Full weekday name such as Monday
B	Full month name such as January
b	Abbreviated month name such as jan
d	Day of the month with two digits leading zeros (01,02…31)
e	Day of the month with spaces replacing zeros(1,2, ….31)
D	Date in the format (mm/dd/yy)
H	Military time-two digit hour
I	Civilian time two digit hour
j	Julian date(day of the year) 001…366
M	Two digit minute such as 00,01,…59
m	Numeric two digit month 00,01,…12
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	Time in format hour:min:second
U	Week number of the year
W	Week of year[00-53], with Monday being first day of the week.
Y	Year as ccyy(4 digits)
Z	Time zone name

10.2 cal: The CALENDAR COMMAND



A single parameter specifies the 4 digit year (1 - 9999) to be displayed.
Two parameters denote the Month (1 - 12) and Year (1 - 9999). If arguments are not specified, the current month is displayed. A year starts on 01 Jan.

- To display current month's calendar

  $ cal

Output :

```
    April 2016
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
```

- To display feb 2015 calendar

  $ cal 2 2015

Output :

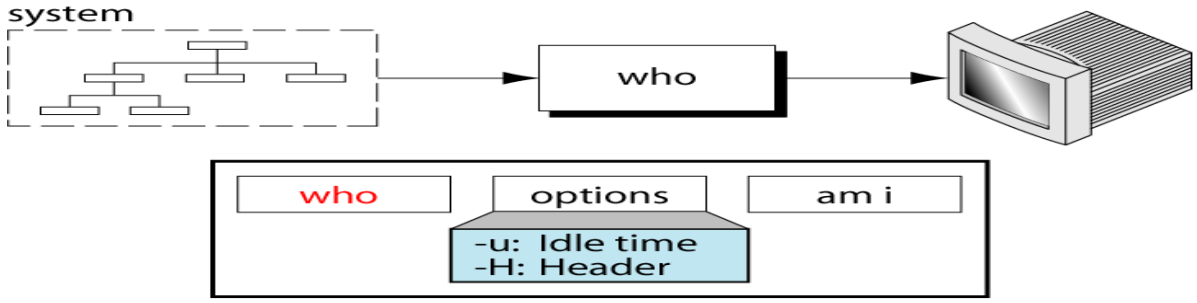
```
     February 2015
Su Mo Tu We Th Fr Sa
1  2  3  4  5  6  7
8  9  10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

- To display complete year calendar.

  $ cal -y

10.3 who: THE Who is online Command

The **who** command prints information about all users who are currently logged in.



who [*OPTION*]... [*FILE*] [am i]

-u –Idle time: Print the idle time for each user, and the process ID.

-H – HEADING : Print a line of column headings.

$who

Displays the username, terminal, and time and date of all currently logged-in sessions.

```
vizion   tty7         2016-08-28 13:28 (:0)
vizion   pts/0        2016-08-28 13:28 (:0.0)
```

$who am i

Displays the same information, but only for the terminal session where the command was issued, for example:

```
vizion   pts/0        2016-08-28 13:28 (:0.0)
```

$who –u

Indicates how long it has been since there was any activity on the line. This is known as **Idle time.** It also returns the process id for the user.

```
vizion   tty7         2016-08-28 13:28   .          2420 (:0)
vizion   pts/0        2016-08-28 13:28   .          2623 (:0.0)
```

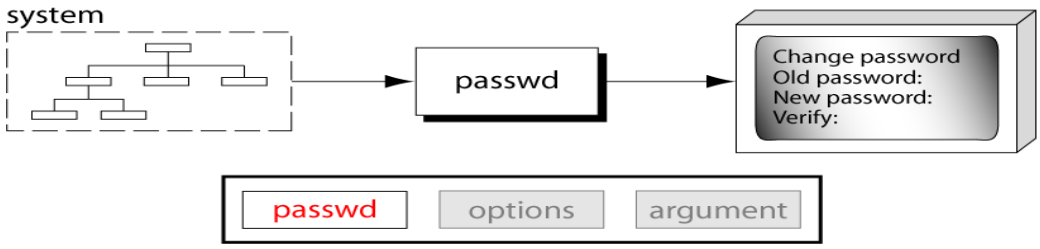
$who -uH

Displays "all" information, and headers above each column of data, for example:

```
NAME       LINE         TIME             IDLE          PID COMMENT  EXIT
                      2016-08-28 13:26              1262 id=l5    term=0 exit=
0
LOGIN      tty1         2016-08-28 13:26              2303 id=1
LOGIN      tty2         2016-08-28 13:26              2304 id=2
LOGIN      tty3         2016-08-28 13:26              2305 id=3
LOGIN      tty4         2016-08-28 13:26              2306 id=4
LOGIN      tty5         2016-08-28 13:26              2307 id=5
LOGIN      tty6         2016-08-28 13:26              2308 id=6
                      2016-08-28 13:26              2309 id=x
vizion   + tty7         2016-08-28 13:28   .          2420 (:0)
vizion   + pts/0        2016-08-28 13:28   .          2623 (:0.0)
```

10.4 passwd command.: The change Password(PASSWD) COMMAND

The **passwd** command is used to change the password of a user account. A normal user can run **passwd** to change their own password, and a system administrator (the superuser) can use **passwd** to change another user's password, or define how that account's password can be used or changed.



passwd syntax
passwd [*OPTION*] [*USER*]

$passwd
Running passwd with no options will change the password of the account running the command. You will first be prompted to enter the account's current password:

- (current) UNIX password:
  If it is correct, you will then be asked to enter a new password:
- Enter new UNIX password:
  ...and to enter the same password again, to verify it:
- Retype new UNIX password:
  If the passwords match, the password will be changed.

The **passwd** command changes passwords for user accounts. A normal user can only change the password for their own account, but the superuser can change the password for any account. **Passwd** can also change or reset the account's validity period — how much time can pass before the password expires and must be changed.

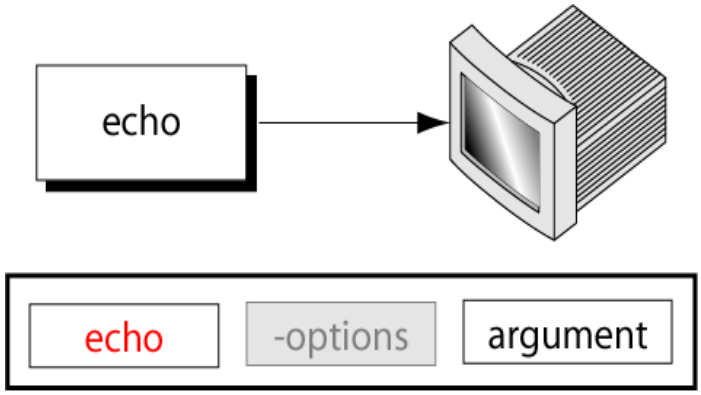
Before a normal user can change their own password, they must first enter their current password for verification. (The superuser can bypass this step when changing another user's password.)
After the current password has been verified, **passwd** checks to see if the user is allowed to change their password at this time. If not, **passwd** refuses to continue, and exits.
Otherwise, the user is then prompted twice for a replacement password. Both entries must match for **passwd** to continue.

Rules for giving passwords
a) must be >=6 characters long,
b) must contain 2 out of 3 of
upper-case letters,
lower-case letters,
non-letters (digits, punct)
c) may not be a dictionary word or too similar to your name

10.5 echo: Print message command

The **echo** command used to display a message.

- To display the diagnostic messages on the terminal or to issue prompts for taking user input
- To evaluate shell variable.



$ echo [*SHORT-OPTION*]... [*STRING*]...

Options

-n	Do not output a trailing newline.
-e	Enable interpretation of backslash escape sequences (see below for a list of these).
-E	Disable interpretation of backslash escape sequences (this is the default).
--help	Display a help message and exit.
version	Output version information and exit.

If you specify the **-e** option, the following escape sequences are recognized:

\\	A literal backslash character ("\").
\a	An alert (The BELL character).
\b	Backspace.
\c	Produce no further output after this.
\e	The escape character; equivalent to pressing the escape key.
\f	A form feed.
\n	A newline.
\r	A carriage return.
\t	A horizontal tab.
\v	A vertical tab.

Ex 1: **$echo Hello world**
Output :Hello world // Outputs the following text
Ex 2:

$**x=10**
$echo "The value of x is $x"
Entering these two commands will output the following text:
Output :The value of x is 10.

Ex 3:
$echo -e "Here**b**the**b**spaces**b**are**b**backspaced**"**
Outputs the following text:
Output :Herthspacearbackspaced //one character before backslash is deleted.

Ex 4:

$echo –e "Unix **n** is **n** a **n** os**"**

Output :$Unix
is
a
Os //prints in a new line

Ex 5:
$echo –e "unix **t** is **t** a **t** os**"**
Output :unix is a os //**t** –prints after 8 spaces

Ex 6:
$echo –e "unix **r** is a os**"**
Output : is a os //moving cursor to beginning of line.
//words before \r is deleted.

Ex 7:
$echo –e "unix **c** is a os"
Output :unix // cursor in same line.
//words after \c deleted.
Ex 8:
$echo $SHELL
Output :/bin/bash //prints the shell name

10.6 printf command-alternative to echo command

$printf "No file entered"
Output :No file entered

$printf My current shell is %s\n" $SHELL
Output :My current shell is /bin/bash //Here %s act as a placeholder for the value $SHELL

%s-String **%f-**Floating point number
%d-Decimal integer **%x**-Hexadecimal integer
%o-octal integer

10.7 THE MAN COMMAND

The man command knowing more about Unix commands and using Unix online manual pages

man is the system's manual viewer; it can be used to display manual pages, scroll up and down, search for occurrences of specific text, and other useful functions.

Each argument given to **man** is normally the name of a program, utility or function. The manual page associated with each of these arguments is then found and displayed. A section number, if provided, will direct **man** to look only in that section of the manual. The default action is to search in all of the available sections, following a pre-defined order and to show only the first page found, even if page exists in several sections.

A man page is divided into a number of compulsory optional sections. Every command doesn't have all sections, but the first three(NAME,SYNOPSIS and DESCRIPTION) are seen in all man pages.

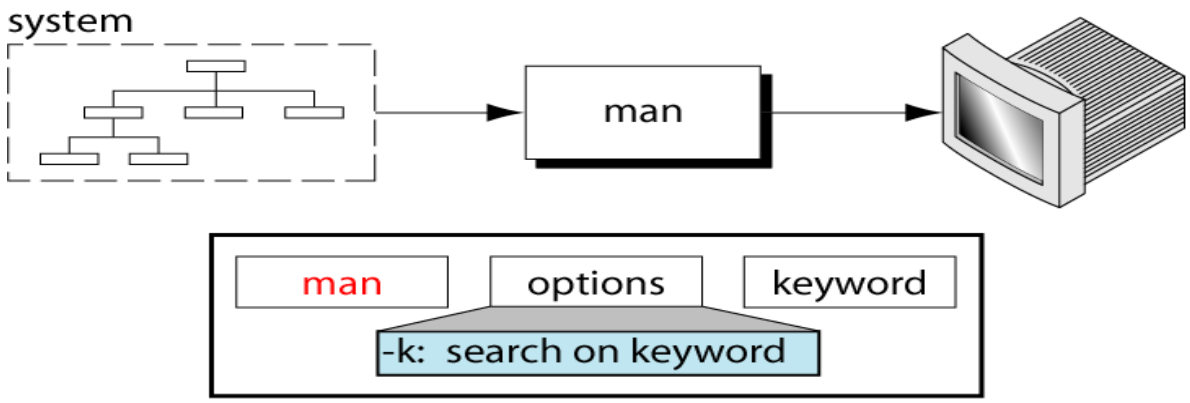
NAME presents the online introduction to the command

SYNOPSIS shows the syntax used by the command

DESCRIPTION provides a detailed information.

man syntax

$man [option] command name



Options

-K,--global-apropos	Search for text in all manual pages. This is a brute-force search, and is likely to take some time; if you can, you should specify a section to reduce the number of pages that need to be searched. Search terms may be simple strings (the default), or regular expressions if the **--regex** option is used.

Section Numbers

The section numbers of the manual are listed below. While reading documentation, if you see a command name followed by a number in parentheses, the number refers to one of these sections. For example, **man** is the documentation of **man** found in section number **1**. Some commands may have documentation in more than one section, so the numbers after the command name may direct you to the correct section to find a specific type of information.

The section numbers, and the topics they cover, are as follows:

Section number	Description
1	Executable programs or shell commands
2	System calls (functions provided by the kernel)
3	Library calls (functions within program libraries)
4	Special files (usually found in /dev)
5	File formats and conventions eg /etc/passwd
6	Games
7	Miscellaneous (including macro packages and conventions), e.g. man, groff
8	System administration commands (usually only for root)
9	Kernel routines [Non standard]

Man Examples

$man man

View the manual page for the **man** command.

$man –s4 passwd

This displays the documentation for a configuration file from the section 4. Even this information is present in the section 1 it won't display section 1 information.

10.8 whatis command: displays short manual page descriptions.

Each manual page has a short description available within it. **whatis** searches the manual page names and displays the manual page descriptions of any name matched.

$whatis who

Display a description of what **who** is.

```
who                  (1p)  - display who is on the system
who                  (1)   - show who is logged on
```

$whatis cal

```
cal                  (1)   - displays a calendar
cal                  (1p)  - print a calendar
```

10.9 apropos command :

searches the manual pages for a keyword or regular expression. Each manual page has a short description included with it. **apropos** searches these descriptions for instances of *keyword*.

$apropos find

aa_find_mountpoint (2) - find where the apparmor interface filesystem is mounted
chkdupexe (1) - find duplicate executables
ffs (3) - find first bit set in a word

11. ls COMMAND

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.

Syntax

The syntax for the **ls** command is:

```
ls [options] [names]
```

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.

Option	Description
-x	Displays files as rows across the screen.
-1	Displays each entry on a line.

To show long listing information about the file/directory.

$ls -l

```
total 1340
-rwxrwxr-x 1 vizion vizion    6961 2015-09-17 16:17 a.out
-rw-rw-r-- 1 vizion vizion       0 2016-08-08 16:03 cal
drwxr-xr-x 2 vizion vizion    4096 2016-08-28 13:29 Desktop
drwxr-xr-x 2 vizion vizion    4096 2008-03-06 13:08 Documents
drwxr-xr-x 2 vizion vizion    4096 2008-03-06 13:08 Download
-rw-rw-r-- 1 vizion vizion    1129 2016-08-11 00:49 ecos.c
drwxrwxr-x 2 vizion vizion    4096 2016-08-24 05:53 fedora
-rw-rw-r-- 1 vizion vizion      29 2016-08-10 13:22 ff.c
```

a.Field 1:

- **1st Character – File Type:** First character specifies the type of the file. In the example above the hyphen (-) in the 1st character indicates that this is a normal file. Following are the possible file type options in the 1st character of the ls -l output.
  - ✓ **Field Explanation**
  - ✓ – normal file
  - ✓ d directory
  - ✓ s socket file
  - ✓ l link file
- **2nd to 9th character -- File Permissions:** Next 9 character specifies the files permission. Each 3 characters refers to the read, write, execute permissions for owner, group and other.

b.Field 2 – Number of links: Second field specifies the number of links for that file. In this example, 1 indicates only one link to this file ecos.c and 2 links to directory named fedora

c.Field 3 – Owner: Third field specifies owner of the file. In this example, ecos.c file is owned by username 'vizion'.

d. Field 4 – Group: Fourth field specifies the group of the file. In this example, the file ecos.c belongs to "vizion' group.

e. Field 5 – Size: Fifth field specifies the size of file. In this example, '1129' indicates the ecos.c file size.

f. Field 6 – Last modified date & time: Sixth field specifies the date and time of the last modification of the file.

g.Field 7 – File name: The last field is the name of the file.

Display Directory Information Using ls -ld

When you use "ls -l" you will get the details of directories content. But if you want the details of directory then you can use -d option as., For example, if you use ls -l /etc will display all the files under etc directory. But, if you want to display the information about the /etc/ directory, use -ld option as shown below.

```
$ ls -l /etc
total 3344
-rw-r--r--   1 root root   15276 Oct  5  2004 a2ps.cfg
-rw-r--r--   1 root root    2562 Oct  5  2004 a2ps-site.cfg
drwxr-xr-x   4 root root    4096 Feb  2  2007 acpi
-rw-r--r--   1 root root      48 Feb  8  2008 adjtime
drwxr-xr-x   4 root root    4096 Feb  2  2007 alchemist
$ ls -ld /etc
drwxr-xr-x 21 root root 4096 Jun 15 07:02 /etc
```

12. FLEXIBILITY OF COMMAND USAGE

COMBINING COMMANDS

UNIX allows you to specify more than one command in the command line. Each command has to be separated from the other by a ; (semicolon).

Example

$wc note; ls -l note

tThe above command first it displays the line count, word cont and byte or character count along with this it also display the details of note file.

When you learn to redirect the output of these commands you may even like to group them together within parentheses .

Example :

$(wc note ; ls -l note) > newlist

The combined output of the two commands is now sent to the file newlist. Whitespace is provided here only for better readability. You might reduce a few keystrokes like this

$wc note;ls -l note)>newlist

When a command line contains a semicolon, the shell understands that the command on each side of it needs to be processed separately. The ; here is known as a metacharacter, and you'll come across several metacharaters that have special meaning to the shell.

A command line can overflow or be split into multiple lines

A command is often keyed in. though the terminal width is restricted to 80 characters, that doesn't

prevent you from entering a command, or a sequence of them, in one line even though the total width may exceed 80 characters. The command simply overflows to the next line though it is stil in a single logical line.

Sometimes, you'll find it necessary or desirable to split a long command line into multiple lines. In that case, the shell issues a **secondary prompt,** usually >, to indicate to you that the command line isn't complete. This is easily shown with the echo command:

$echo "this is
> a three-line
>text message"
output:
this is
a three-line
text message

13. MEANING OF INTERNAL AND EXTERNAL COMMANDS

UNIX commands are classified into two types

- Internal Commands - Ex: echo
- External Commands - Ex: ls, cat

Internal Command:

Internal commands are something which is built into the shell. For the shell built in commands, the execution speed is really high. It is because no process needs to be spawned for executing it.

- For example, when using the "cd" command, no process is created. The current directory simply gets changed on executing it.

External Command:

External commands are not built into the shell. These are executable present in a separate file. When an external command has to be executed, a new process has to be spawned and the command gets executed.

- For example, when you execute the "cat" command, which usually is at /usr/bin, the executable /usr/bin/cat gets executed.

How to find out whether a command is internal or external?

***type* command:**

```
$ type cd
cd is a shell builtin
$ type cat
cat is /bin/cat
```

For the internal commands, the type command will clearly say its shell built-in, however for the external commands, it gives the path of the command from where it is executed.

THE TYPE COMMAND: knowing the type of a command and locating it.

type - Display information about command type.

The **type** command is a shell built-in that displays the kind of command the shell will execute, given a particular command name. It works like this: type *command*
where "command" is the name of the command you want to examine. Here are some examples:
$type type
Output: type is a shell built-in

$type ls
Output: ls is aliased to 'ls –color=tty'

$type cp
Output: cp is /bin/cp
Here we see the results for three different commands. Notice that the one for ls (taken from a Fedora system) and how the ls command is actually an alias for the ls command with the "-- color=tty" option added. Now we know why the output from ls is displayed in color!

14. THE MORE COMMAND

The more command and using it with other commands

The **more** command displays the file called *name* in the screen. The RETURN key displays the next line of the file. The spacebar displays the next screen of the file.

Syntax

The syntax for the **more** command is:

```
more [options] [files]
```

Option	Description
-c	Page through the file by clearing the window. (not scrolling).
-d	Displays "*Press space to continue, 'q' to quit*"
-f	Count logical lines rather than screen lines (wrapping text)
-l	Ignores form feed (^L) characters.
-r	Displayall control characters.
-s	Displays multiple blank lines as one blank line.
-u	Does not displayunderline characters and backspace (^H).
-w	Waits for a user to press a keybefore exiting.
-n	Displays n lines per window.
+*num*	Displays the file starting at line number *num*.
+/*pattern*	Displays the file starting at two lines before the *pattern*.

$more ecos.c

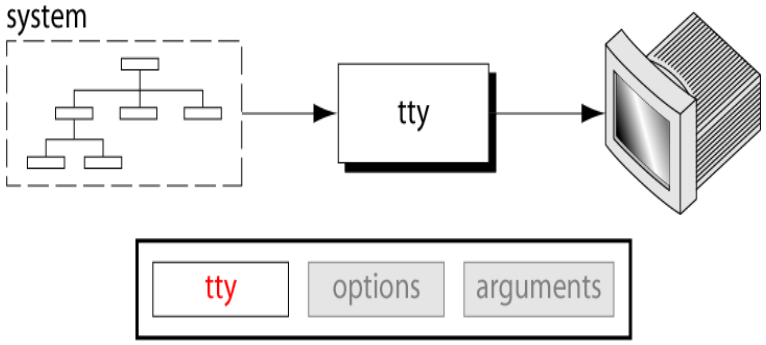
```
#include<stdio.h>
#include<stdlib.h>
#include "kapi.h"
#include<pthread.h>
#define RES_MAX 10
#define PROC_MAX 8

cyg_mutex_t res_lock= CYG_MUTEX_INITIALIZER;
cyg_cond_t res_wait= CYG_COND_INITIALIZER;
res_t res_pool[PROC_MAX];
int res_count =PROC_MAX;
void res_init()
{
int i;
for(i=0; i< PROC_MAX;i++)
res_pool[i]=i+1;
}
res_t res_allocate()
{
res_t res;
cyg_mutex_lock(&res_lock);
while(res_count==0)
cyg_cond_wait(&res_wait, &res_lock);
--More--(39%)
```

-- more –(39%) prompt is seen at bottom left corner of the screen. At this prompt you can press a key to perform navigation.

15. KNOWING THE USER TERMINAL

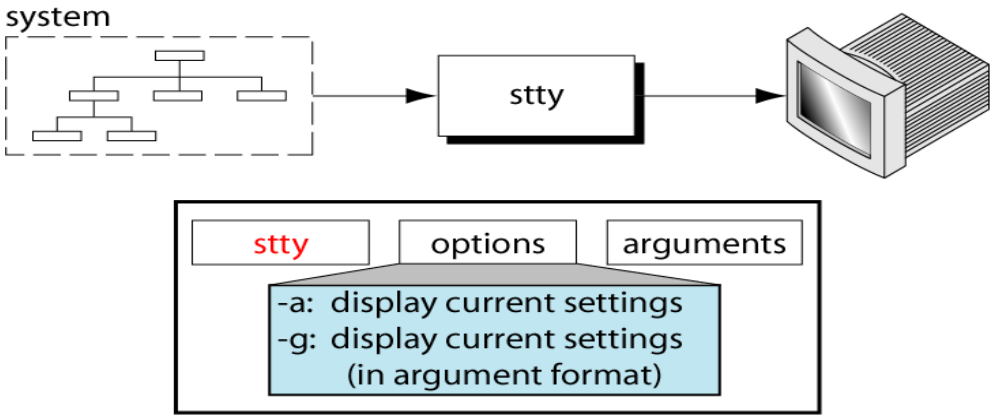
For knowing the user terminal have one command called **tty** (teletype) command.
This command is simple and needs no arguments.



$tty
output is /dev/pts/10
The terminal filename is 10 resident in the pts directory. This directory in turns is under the /dev directory. You can use tty in a shell script to control the behavior of the script depending on the terminal it is invoked from.

15.1 Displaying its characteristics and setting characteristics

For displaying and setting characteristics in unix uses the command called **stty**



15.1.1 Set terminal without option or arguments.
$stty
Output: Speed 900 baud;

Line =1;intr =’A’; …….

It shows the current common settings of your terminal

15.1.2 Set terminal with options only

Set terminal can be used with two options – (-a and –g)
-a displays the current terminal option settings
-g displays the selected settings in a format that can be used as an argument to another set terminal command.

15.1.3 Set terminal with arguments

- **Set erase and kill(ek):** The ek argument sets the default erase(Delete key –Ctrl+h)and kill (ctrl+c) to their defaults.
- **Set erase key(erase):**By default the Erase key is Ctrl+h on the terminal. It deleted the previous harcter typed.We can reconfigure the keyboard to use another key as the delete key with the erase argument as shown below
  $stty erase ^e
- **Set terminal to general configuration(sane):**
  $stty sane
  Sane argument sets the terminal configuration to reasonable settings.
- **Set kill(kill)**
  $stty kill 9
  The kill deletes a whole line.
- **Set Interrupt key(intr)**
  $stty intr 9
  The interrupt key suspends or interrupts a command.

15.2 Other useful commands:
15.2.1 SCRIPT COMMAND- record session

Syntax
script options arguments

- Used to record an interactive session ,when you want to start recording key the command as script . **$script**

- To stop recording, key the command as exit. **$exit**

Example:

```
[vizion@localhost ~]$ script
Script started, file is typescript
[vizion@localhost ~]$ date
Sun Aug 28 16:33:55 IST 2016
[vizion@localhost ~]$ who
vizion   tty7         2016-08-28 13:28 (:0)
vizion   pts/0        2016-08-28 13:28 (:0.0)
vizion   pts/1        2016-08-28 15:29 (:0.0)
[vizion@localhost ~]$ exit
exit
Script done, file is typescript
[vizion@localhost ~]$
```

To view the recorded script
$gedit filename
$gedit typescript

```
Script started on Sun 28 Aug 2016 04:33:52 PM IST
]0;vizion@localhost:~[?1034h[vizion@localhost ~]$ date
Sun Aug 28 16:33:55 IST 2016
]0;vizion@localhost:~[vizion@localhost ~]$ who
vizion   tty7         2016-08-28 13:28 (:0)
vizion   pts/0        2016-08-28 13:28 (:0.0)
vizion   pts/1        2016-08-28 15:29 (:0.0)
]0;vizion@localhost:~[vizion@localhost ~]$ exit
exit

Script done on Sun 28 Aug 2016 04:33:59 PM IST
```

15.2.2 clear Command: clears the screen

$clear
Clears the screen and puts the cursor at top.

15.2.3 uname command

$uname options arguments

Options:
-a :all
-n:name of the system
-s:name of the OS
-r:software release

$uname
or
$uname –s

Output: Linux

$uname –n
Output: localhost.localdomain

$uname –r
Output:2.6.23.1-42.fc8

$uname –a
Output: Linux localhost.localdomain2.6.23.1-42.fc8

16.MANAGING THE NON UNIFORM BEHAVIOR OF TERMINALS AND KEY-BOARDS

Terminals and keyboards have no uniform behavioral pattern. Terminal settings directly impact key-board operation.
Backspacing Doesn't Work: consider that you misspelled passwd as password, and when you pressed the backspace key to erase the last three characters, you saw this
$password^H^H^H.
Backspacing is not working here; that's why you see the symbol ^H every time you press the key.

Killing a Line: if the command line contains many mistakes, you could prefer to kill the line altogether without executing it. In that case, use [Ctrl-u].
The line-kill character erases everything in the line and returns the cursor to the beginning of the line.

Interrupting a Command: sometimes, a program goes on running for an hour and doesnot seem to complete. You can interrupt the program and bring back the prompt by using either of the two sequence [Ctrl-c] or [Delete]

Terminating a Command's Input You know that the **cat** command is used with an argument representing the filename *(1.4.10)*. What happens if you omit the filename and simply press *[Enter]*?

```
$ cat[Enter]
```

Nothing happens; the command simply waits for you to enter something. Even if you do some text entry, you must know how to terminate your input. For commands that expect user input, enter a *[Ctrl-d]* to bring back the prompt:

```
$ cat
[Ctrl-d]                                    The end-of-file or eof character
$ _
```

This is another important key sequence; we'll often refer to *[Ctrl-d]* as the *eof* or end-of-file character. Sometimes pressing the interrupt key also works in this situation.

The Keyboard is Locked When this happens, you won't be able to key in anything. It could probably be due to accidental pressing of the key sequence *[Ctrl-s]*. Try using *[Ctrl-q]* to release the lock and restore normal keyboard operation. These two sequences are actually used by the system to control the flow of command output.

At times, you may consciously like to use *[Ctrl-s]* and *[Ctrl-q]*. If the display from a command is scrolling too fast for you to see on the terminal, you can halt the output temporarily by pressing *[Ctrl-s]*. To resume scrolling, press *[Ctrl-q]*. With modern hardware where the output scrolls off very fast, this facility is now practically ineffective, but it pays to know what they do because inadvertent pressing of *[Ctrl-s]* can lock your terminal.

The [Enter] Key Doesn't Work This key is used to complete the command line. If it doesn't work, you can use either *[Ctrl-j]* or *[Ctrl-m]*. These key sequences generate the linefeed and carriage return characters, respectively.

The Terminal Behaves in an Erratic Manner Your terminal settings could be disturbed; it may display everything in uppercase or simply garbage when you press the printable keys. Try using the command **stty sane** to restore sanity. Since the *[Enter]* key may not work either in these situations, use *[Ctrl-j]* or *[Ctrl-m]* to simulate *[Enter]*.

Also keep in mind that some UNIX programs (like **mailx**) are interactive and have their own set of internal commands (those understood only by the program). These commands have specific key sequences for termination. You may not remember them, so try using **q**, **quit**, **exit** or *[Ctrl-d]*; one of them might just work.

Table 2.2 Keyboard Commands to Try When Things Go Wrong

Keystroke or Command	*Function*
[Ctrl-h]	Erases text (The *erase* character)
[Ctrl-c] or *[Delete]*	Interrupts a command (The *interrupt* character)
[Ctrl-d]	Terminates login session or a program that expects its input from the keyboard (The *eof* character)
[Ctrl-s]	Stops scrolling of screen output and locks keyboard
[Ctrl-q]	Resumes scrolling of screen output and unlocks keyboard
[Ctrl-u]	Kills command line without executing it (The *line-kill* character)
[Ctrl-\]	Kills running command but creates a core file containing the memory image of the program (The *quit* character)
[Ctrl-z]	Suspends process and returns shell prompt; use **fg** to resume job (The *suspend* character)
[Ctrl-j]	Alternative to *[Enter]*
[Ctrl-m]	As above
`stty sane`	Restores terminal to normal status (a UNIX command)

17. THE ROOT LOGIN

Root: the system administrator's login. The unix system provides a special login name for the exclusive use of the administrator, it is called **root**. This account doesn't need to be separately created but 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: ******

The prompt of the root is # other users(non privileged user) either $ or **%**

Once you login as root, you are place in root's home directory. Depending on the system, this could be **/** or **/root**

BECOMING THE SUPER USER: SU COMMAND

Any user can acquire super user status with the **su** command if she knows the root password.

Example, the user GMIT becomes a super user in this way

$su

Password:****

#pwd

/home/GMIT

Though the current directory does not change the # prompt indicates the GMIT now has powers of a super user. To be in root's home directory on superuser login, use **su –l.**

Creating a user's Environment: users often rush to the administrator with the complaint that a program has stopped running. The administrator first tries running it in a simulated environment.

su, when used with a -, recreates the user's environment without taking the login password route:

$su – GMIT

This sequences executes GMIT's .profile and temporarily creates GMIT's environment. **Su** runs a separate sub shell, so this mode is terminated by hitting [Ctrl-d] or using **exit.**

THE /ETC/PASSWD AND /ETC/SHADOW FILES

All user information except the password encryption is now stored in /etc/passwd. This file contained the password once, the reason why it continues to be known by that name. the encryption itself is stored in /etc/shadow. This is now the control file used by **passwd** to ascertain the legitimacy of a user's password.

There are seven fields here and their significance is noted below(in order they appear in /etc/passwd)

- user name : the name you use to log on to a UNIX system(GMIT)
- Password : No longer stores the password encryption but contains an X or *
- UID : The user's numerical identification. No two users should have the same UID.
- GID : The user's numerical group identification. This number is also the third field in /etc/group.
- Comment or GCOS : User details e.g. her name, address and so forth.
- Home directory : The directory where the user ends up logging in. the login program reads this field to set the variable HOME.
- Login shell : The first program executed after logging in.

18. USER MANAGEMENT

Commands to add modify and delete users

The term *user* in UNIX is not meant to be only a person; it can represent a project or an application as well. A group of users performing similar functions may use the same username to use the system. It's thus quite common to have usernames like marketing, accounts, and so forth, for the creation and maintenance of user accounts, UNIX provides three commands **useradd, usermod** and **userdel.**

When opening a user account, you have to associate the user with a group. A group usually has more than one member with a different set of privileges. People working on a common project should be able to read one another's files, which is possible only if they belong to same group.

- Creating a user involves defining the following parameters
- A user identification number(UID) and username.
- A group identification number (GID) and group name.
- The home directory
- The login shell
- The mailbox in /var/mail
- The password

Most of these parameters are found in a single line identification the user in /etc/passwd. We will now create a group for a user and then add that user to the system

18.1 Groupadd: adding a group

The **groupadd** command creates a new group account using the values specified on the command line plus the default values from the system. The new group will be entered into the system files as needed.

groupadd syntax

groupadd [*options*] *group*

-g, --gid *GID*	The numerical value of the group's ID. This value must be unique, unless the **-o** option is used. The value must be non-negative. The default is to use the smallest ID value greater than or equal to **GID_MIN** and greater than every other group. See also the **-r** option and the **GID_MAX** description.

Note: For this command to work you must have superuser rights or be logged in as root.

```
#groupadd  gmit
```

The above example would create a new group called "GMIT". This new group could then have users added to it using the useradd command.

#groupadd –g 241 gmit

The above example creates a new group, gmit with a GID of 241.
The command places this entry in /etc/group. Once an entry for the group has been made, you are now ready to add a user of tis group to the system.

18.2 Useradd: Adding a User:

The **useradd** command adds new user to the system.

useradd syntax

useradd [*options*] *LOGIN*

here useradd is a command

option

-g, **--gid** *GROUP*	The group name or number of the user's initial login group. The group name must exist. A group number must refer to an already existing group.

Note: For these commands to work you must have superuser rights or be logged in as root.

useradd newperson

Creates **newperson** as a new user. Once the new user has been added, you would need to use the **passwd** command to assign a password to the account.
Once a user has been created, you can modify any of the user settings, such as the user's home directory, using the **usermod** command.

$useradd –u 210 –g gmit –s /bin/bash –m kumar

From the above example it creates the **kumar** user under the **gmit** group with 210 UID number with bash shell as working shell environment.

18.3 Usermod and userdel: modifying and removing users

Usermod is used for modifying some of the parameters set with **useradd**. Users sometimes need to change their login shell, and the following command line sets bash as the login shell for the user.

$usermod –s /bin/ksh kumar

From the above command user changed their kernel from bash to ksh.

Users are removed from the system with **userdel**. The following command removes the user kumar from the system.

$userdel kumar