MODULE 1

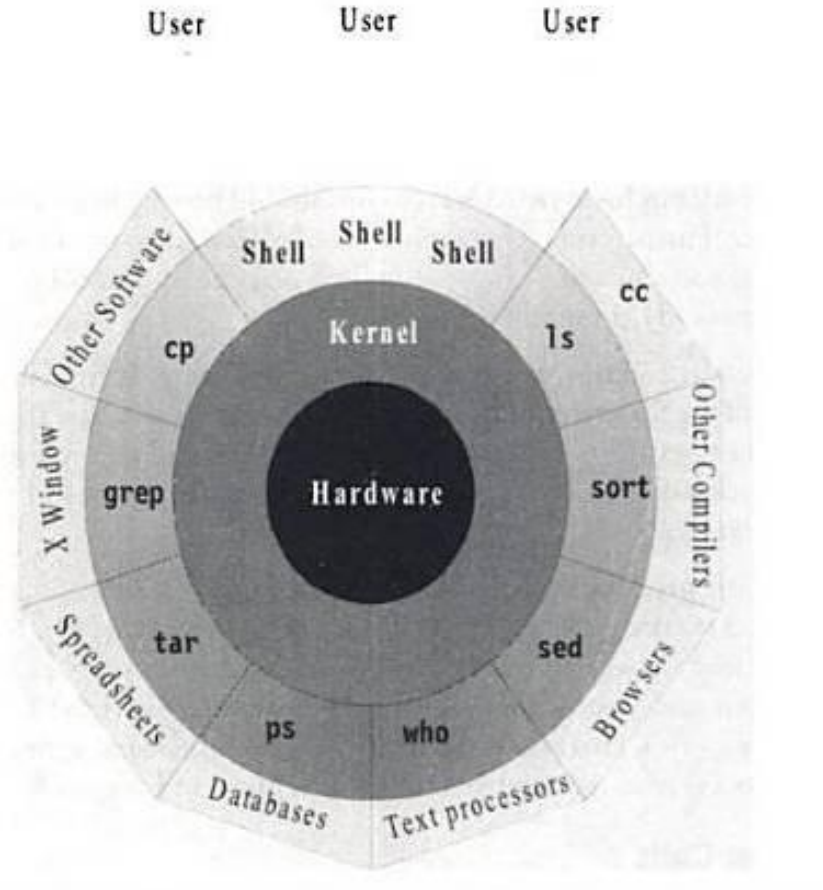
History of UNIX:

- Prior to UNIX, many operating systems ran collections or "**batches**" of operations one at a time.
- This single-user "**batch-processing**" approach did not take advantage of the potential processing power and speed of computers.
- The Unix OS was developed (based on Multics & CTSS operating systems) by **Ken Thompson** at the AT&T Bell Laboratories in 1969. He wanted to create a multi-user operating system to run "space travel" game.
- Ken's philosophy was to create an operating system with commands or "utilities" that would do one thing well (i.e. UNIX). Pipes could be used combine commands.
- The first versions of UNIX were written in "machine-dependent" program such as **PDP-7.**
- In 1973 **Ken Thompson** and **Dennis Ritchie** compiled UNIX in C programming language to make operating system "portable" to other computers systems.
- UNIX became a popular OS among institutions such as colleges & universities through a 4-year "try before you buy" deal.
    - Efficient and inexpensive way of networking
    - promotes Internet use and file-sharing
    - Open system allows for source code to be shared among many programmers - allows for better coordination among programmers
- Students at University of California (in Berkley) further developed the UNIX operating system and introduced the BDS version of UNIX i.e., the free version of UNIX.
- There were versions of UNIX for the Personal Computer (PC), such as XENIX, etc., but they didn't catch on in popularity until Linux was developed in the early 90's.
- Linux operating system developed by programming student **Linus Torvalds**, he wanted to develop Unix-like OS just to experiment with new 386 computers at the time.
- Linus decided to make Linux OS source-code for Linux Kernal open to all:
    - Unlike traditional Operating Systems, anyone can modify and distribute Linux OS (as long as they distribute source code of Linux Kernel)
    - "Competition among Hackers" allow code to be improved and distributed often
    - Many users can spot bugs in the operating system or application if source code is "open"
- Since Linux is a "Unix Work-alike", this OS has a reputation to be a very stable platform for networking (creating at-home servers) and running / maintaining applications.
- Agencies such as **Free Software Foundation** created **GNU** project to provide **free software**.

The UNIX Architecture:

1. Division of Labor: Kernel and Shell :

- 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.
- The Kernel – Shell relationship is shown in the figure below:



The kernel – shell relationship

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.

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.

The Features of UNIX:

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.

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.

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

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.

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.

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**.

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.

The UNIX Environment:

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

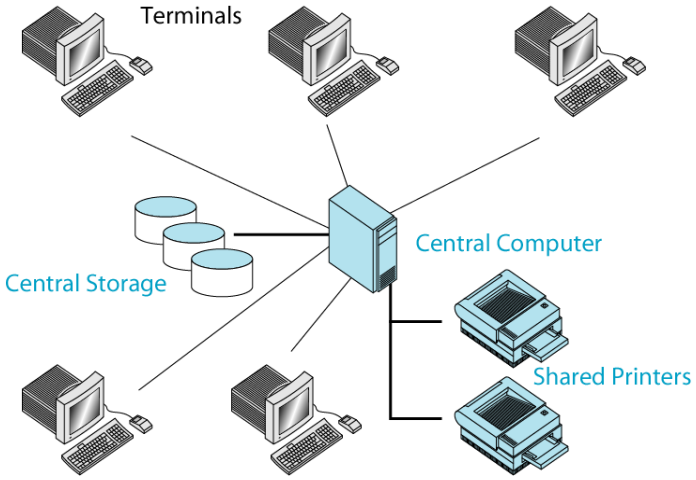
1. Personal Environment:

- 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 of LINUX, the Apple system X, released in 2001, incorporated UNIX as its kernel.

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.

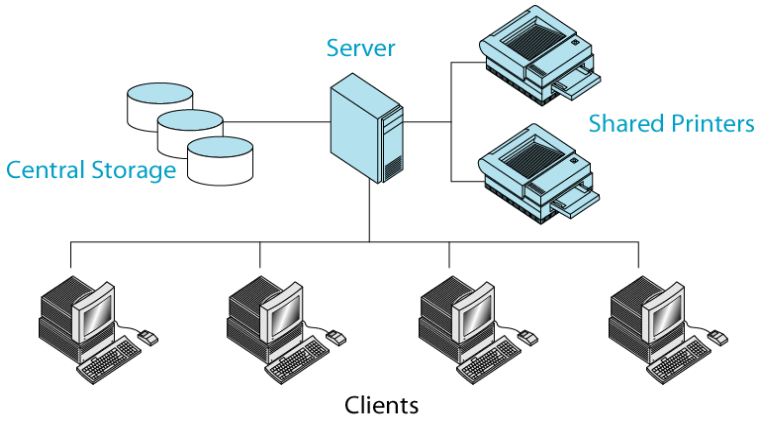
- All the computing must be done by the central computer, it must control the shared resources, it must manage the shared data and printing, and it must also do the computing.
- All these work tend to keep the computer busy and the responses will be slow.
- A typical college lab in which a minicomputer is shared by many students is shown in the figure below:



Time Sharing Environment

3. Client/Server Environment:

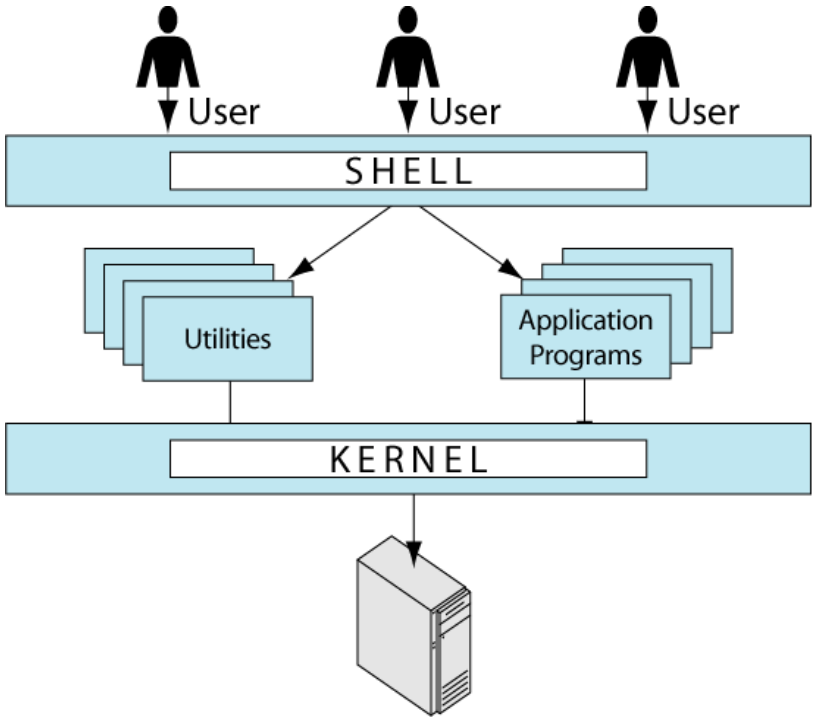
- A Client/Server computing environment splits the computing function between a central computer and users' computers.
- The users are given personal computers or workstations so that some computation responsibility can be moved off the central computer and assigned to the workstations.
- In the Client/Server 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:



Client/Server Environment

UNIX Structure (Components of UNIX):

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:



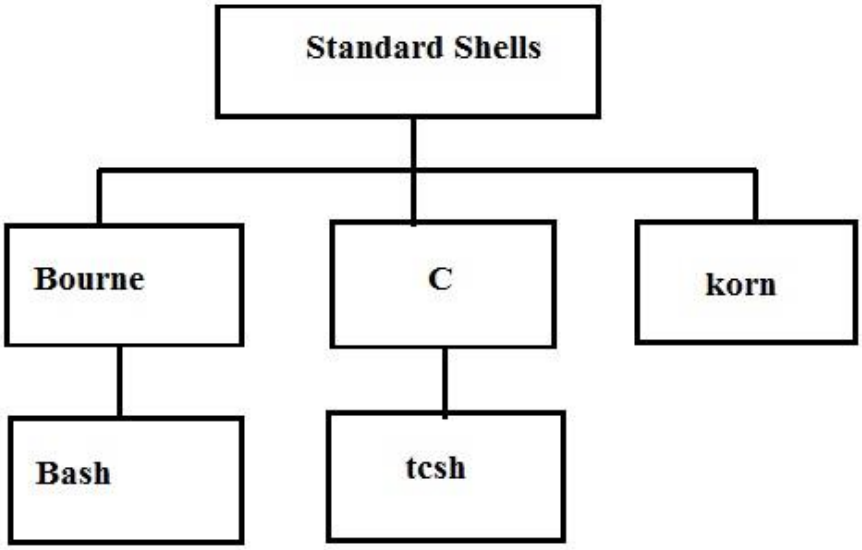
Component of UNIX

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.

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:



Some Standard UNIX Shells

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.

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.

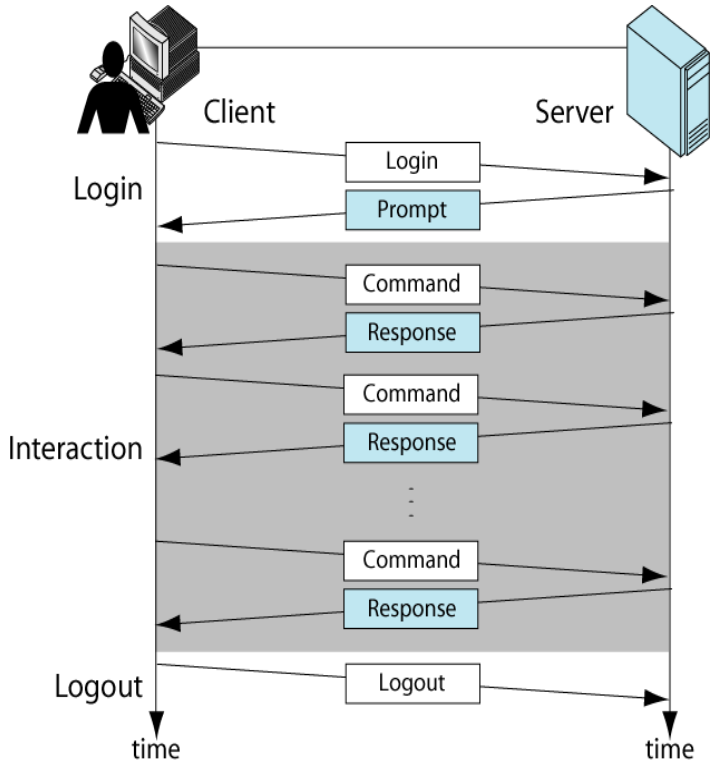
POSIX and the Single UNIX Specification:

- 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.

The Login Prompt:

The interactive session contains three steps: login, interaction and logout. The interactive session is shown in the figure below:



Interactive Session

Login:

The general pattern for login is listed as follows:

1. You must make contact with the system.
    - If you are working on a local network and are always connected to remote server, starting the **login** process is as simple as selecting an option in a menu.
    - On the other hand, if you are making a connection from the remote location, then you will need to use special connection software, often referred as **Telnet** software.
2. Wait for system **login prompt.**
    - Once you have connected to the server, you must wait for the server to ask you to identify yourself
    - This is usually just a login prompt, sometimes with the name of the system. A typical login prompt is :

Login:

3. Type user id.
    - Enter your user id. Note that UNIX is a case-sensitive system. This means that the uppercase letter A is different than the lowercase a.
    - Usually, your user id is all lowercase. If it contains both uppercase and lowercase, you must type exactly as it was given to you

Login: username

4. Type your password.
    - After you enter your user id, the system will prompt you for your password.
    - As you type your password, it will not be displayed on the screen; the system displays an asterisk for each letter of the password. Some systems don't even display the asterisks; they just leave the screen blank.

Password: ******

- If you do everything correctly, you will see the shell prompt. If you make a mistake, the system will give you a error message

Interaction:

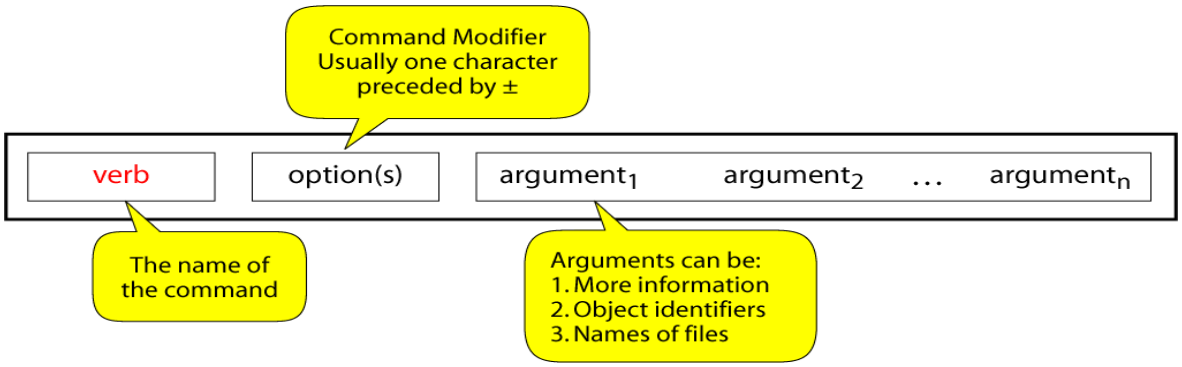
- Once you connect to the server, you can enter commands that allow you to work with computer.
- The result will be displayed on the terminal for all the valid user commands.

Logout:

- It is very important to logout when you are through with the system. There are several reasons for this.
- First, it frees system resources for others who may need to use them.
- Some unauthorized person may walk up and gain access to the system, and your files.

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:



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.

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.

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:

  $ wc abc def xyz

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

Basic Commands

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
 $ _
```

---

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
$_
```

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
emp.lst
usdsk06x
usdsk07x
usdsk08x
```

- 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

$ ls perl
Perl: No such file or directory
```

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)
kumar    pts/10      Aug  1  07:56  (pc123.heavens.com)
sharma   pts/6       Aug  1  02:10  (pc125.heavens.com)
sachin   pts/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)
kumar   pts/10      Aug 1  07:56   0:33   11200  (pc123.heavens.com)
sachin  pts/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.

---

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

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
```

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
```

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.

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.

The type Command: Knowing the Type of a Command and locating it

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

```
$ type date
date is hashed  (/usr/bin/date)
```

- When the user execute the date 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.**

The man command:

- UNIX offers an online help facility in the **man** command.
- **man** displays the documentation often called the **man documentation** of practically every command on the system.
- A man page is divided into a number of compulsory and optional sections. Every command doesn't have all sections, but the first three are generally seen in all man pages.
- **NAME** presents a one-line introduction to the command.
- **SYNOPSIS** shows the syntax used by the command.
- **DESCRIPTION** provides a detailed description.
- The SYNOPSIS follows certain conventions and rules which every user must understand:
    - If the command argument is enclosed in rectangular brackets [ ], then it is optional; otherwise, the argument is required.
    - The ellipsis (set of three dots) implies that there can be more instances of preceding word. The expression [file . . .] signifies more than one filename as argument.

- If you can find the character | in any of these areas, it means that only one of the options shown on either side of the pipe can be used.

- The **OPTIONS** section list all the options used by the command.
- The **EXIT STATUS** section lists possible error conditions and their numeric representation.
- Example of a man page for **wc** command is shown below:

```
User Commands                                wc(1)
NAME
     wc - display a count of lines, words and characters in a file
SYNOPSIS
     wc [ -c | -m  | -C ]  [ -lw ]  [ file ... ]
DESCRIPTION
     The wc utility  reads  one  or  more  input  files and,  by
     default,  writes the number of newline characters, words and
     bytes contained in  each input file to the standard output.
     The utility also writes a total count for all named  files,
     if more than one input file is specified.
     wc considers a word to be a non-zero-length string  of  char-
     acters  delimited by white space (for example, SPACE, TAB ).
     See iswspace(3C) or isspace(3C).
OPTIONS
     The following options are supported:
     -c        Count bytes.
     -m        Count characters.
     -C        Same as -m.
     -l        Count lines.
     -w        Count words delimited by white space characters or new
               line  characters.  Delimiting  characters are Extended
               Unix Code (EUC) characters from any code  set  defined
               by iswspace().
     If  no option is specified the default is -lwc (count  lines,
     words, and  bytes.)
OPERANDS
     The following operand is supported:
     file  A path name of an input file. If no file operands  are
           specified, the standard input will be used.
USAGE
     See largefile(5) for the description of the behavior of wc when
     encountering files greater than or equal to 2 Gbyte (2 **31 bytes).
EXIT STATUS
     The following exit values are returned:
     0     Successful completion.
     >0    An error occurred.
SEE ALSO
     cksum(1),      isspace(3C),      iswalpha(3C),       iswspace(3C),
     setlocale(3C), attributes(5), environ(5), largefile(5)
```

man –k : man with the keyword option

- The user may not remember all the commands present in the system. Then the user can search the command using the keyword option available with man.
- The keyword option is **–k**, Displays the header lines of the command that contain any of the keywords.
- **Example:**

- The user wants the manual pages of a command that performs sorting operation, but the user is unsure about the name of the command.
- In such a scenario, the user can use the keyword sort to identify the commands that perform sorting related operations.

```
$ man –k sort
alphasort (3)      - Scan a directory for matching entries
apt-sortpkgs (1) - Utility to sort package index files
FcFontSort (3)   - Return list of matching fonts
qsort (3)          - sort an array
qsort_r (3)        - sort an array
sort (1)           - sort lines of text files
tsort (1)          - perform topological sort
versionsort (3)    - Scan a directory for matching entries
```

whatis

- The man command gives the detailed description about the command, if the user doesn't require the detailed description but needs only the brief description about the command then the user can use **whatis** command.
- whatis command gives a one-line description about the command
- **Example:**

```
$ whatis wc
wc – display a count of words, characters and lines in a file
```

The more command:

- **more** is a command to view the content of a text file one screen at a time.
- Command syntax is : **more [options] [filename]**
- If no file name is provided, more looks for input from the standard input.
- To navigate through more pages, use the following keys:
  - f, *[ctrl-f]* or spacebar — scroll forward one screen
  - b or *[ctrl-b]* — scroll backward one screen
  - j — one line up
  - k — one line down
- **more** can be used with pipeline. Eg : **ls | more.**

tty: Knowing the user terminal

- **tty** (teletype) command tells you the filename of the terminal you are using.
- The command is simple and needs no argument

```
$ tty
/dev/pts/10
```

- The terminal name is **10** (a file named 10) resident in the **pts** directory. This directory in turn is under the **/dev** directory.

---

stty : Displaying and Setting terminal characteristics

- Different terminals have different characteristics. For example the command interruption cannot be possible with your system.
- Sometimes the user may like to change the settings to match the ones used at the previous place of work.
- **stty** command both displays and changes settings.
- The –a (all) option displays the current terminal settings.
- **Example:**

```
$ stty –a
speed 38400 baud; rows 24; columns 80; 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
```

- The output shows the baud rate (the speed) of the terminal- in this case 38,400.
- The other keywords take two forms:
  - ***Keyword = value***
  - ***Keyword*** or ***–keyword***. The – prefix implies that the option is turned off.
- The setting intr = ^C signifies that *[ctrl – c]* interrupts a program. The erase character is *[ctrl –h]*, the kill character is *[ctrl – u]* and eof (end of file) character is set to *[ctrl – d]*.

Changing the settings:

- **echoe : whether backspacing should erase character**
  - The backspacing over a character sometimes removes it from sight and sometimes doesn't. This is decided by the keyword **echoe**.
  - **echoe** (enabled) removes the character from sight.
  - **-echoe** (disabled) doesn't remove a character from sight.
  - The setting **stty echoe** enables echoe.
  - The setting **stty –echoe** disables echoe.

- **echo: Entering a password through shell script**
  - The **echo** setting has to be manipulated to let shell programs accept a password-like string that must not be displayed on the screen.
  - By default the option is turned on, but you can turn it off by using the **stty –echo** command. With this setting, keyboard entry is not echoed.
  - **stty echo** setting turns on keyboard input.

- **Changing the Interrupt key (intr)**
  - **stty** also sets the functions for some of the keys. For instance if you like to use ***[ctrl –c]*** as the interrupt key, then you have to use **stty intr \^c**
  - The keyword **intr** is followed by a space, the **\ (backslash**) character, a **^ (caret)**, and finally the character **c**. This is the way the stty indicates the system that interrupt character is ***[ctrl –c].***

- **Changing the End-of-File key (eof)**
  - ***[ctrl –d]*** is used to terminate input to the command which takes input from the keyboard (for example **bc**).
  - You changes this setting using **stty,** instead of ***[ctrl –d]*** you can use ***[ctrl –a]*** as the eof character: **stty eof \^a**
  - **[ctrl –a]** will now terminate input for those commands that expect input from the keyboard.

- **When everything else fails (sane)**
  - stty also provides another argument to set the terminal characteristics to values that will work on most terminals.
  - Use the word sane as the single argument to the command: **stty sane**
  - Restores *sanity* to the terminal.

Managing the non uniform behavior of the terminals and keyboard:

- Terminals and keyboards have no uniform behavioral pattern.
- Terminal settings directly impact keyboard operation.
- The following table shows which key to press when things don't quite work as expected.

Keystroke / Command	Function
[ctrl –h]	Erases text
[ctrl –c] or [Delete]	Interrupts a command
[ctrl-d]	Terminates login session or a program that expects its input from the keyboard (eof character)
[ctrl –s]	Stops scrolling of the screen output and locks keyboard
[ctrl-q]	Resumes scrolling of screen output and unlocks keyboard
[ctrl-u]	Kills the command line without executing it
[ctrl-\]	Kills the 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
[ctrl-m] [ctrl-j]	Alternative to *[Enter]*
Stty sane	Restores terminal to normal status.

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**.

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**.

The etc/passwd and etc/shadow files:

- All user information except the password encryption is 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**.
- The user information in /etc/passwd file contains seven fields, they are:
  - **Username** – The name you use to log on to a UNIX system
  - **Password**- No longer stores the password encryption but contains an x.
  - **UID**- The user's unique numerical identification.
  - **GID**- The user's numerical group identification.
  - **Comment**- User details, example her name, address and so forth.
  - **Home directory**-The user's home directory.
  - **Login shell**- The login shell for the user.
- For every line in /etc/passwd, there's corresponding entry in /etc/shadow. The relevant line in this file will be of the form

Username : Encrypted password

- Example:

Oracle : PR1lfDdAgJ66dSdf2:::::::

- It is impossible to generate the password from this encryption.

---

useradd: Adding a User

- The **useradd** command adds new users to the system. All parameters related to the user have to be provided in the command line itself:

```
# useradd  -u  210  -g dba  -c "The DBMS" –d /home/oracle –s /bin/ksh –m oracle
#_
```

- This creates the user oracle with a UID of 210 and group name dba. The home directory is /home/oracle, and the user will use the Korn shell.
- The –m option ensures that the home directory is created if it doesn't already exist.
- The line **useradd** creates in /etc/passwd is shown below:

  oracle : x : 210 : 241 : The DBMS : /home/oracle : /bin/ksh
- You now have to set the new user's password with the command

  passwd oracle
- Once all this is done, the oracle user account is ready to use.

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 changes the login shell for the user oracle: (changing the login shell to Bash shell)

  usermod –s /bin/bash oracle
- Users are removed from the system with **userdel.** The following command removes the user oracle from the system:

  userdel oracle
- This removes all entries pertaining to oracle from **etc/passwd** and **etc/shadow**.

****