To understand UNIX, we'll first have to know what an operating system is, why a computer needs operating system and how UNIX is vastly different from other operating systems that came before and after.

What is an operating system?

An **operating system** is the software that manages the computer's hardware and provides a convenient and safe environment for running programs. It acts as an interface between user programs and the hardware resources that these programs access like – processor, memory, hard disk, printer & so on. It is loaded into memory when a computer is booted and remains active as long as the machine is up.

1. THE UNIX OPERATING SYSTEM

- Like DOS and Windows, there's another operating system called UNIX.
- It arrived earlier than the other two, and stayed back late enough to give us the Internet.
- It has practically everything an operating system should have, and several features which other OS never had.
- It runs practically on every Hardware and provided inspiration to the Open Source movement.
- You interact with a UNIX system through a **command interpreter** called the **shell**.
- A command may already exist on the system or it could be one written by user.
- However, the power of UNIX lies in combining these commands in the same way the English language lets you combine words to generate meaningful idea.

2. THE UNIX ARCHITECTURE

- The entire UNIX system is supported by a handful of essentially simple and abstract concepts.
- The success of UNIX, according to Thompson and Ritchie, "lies not so much in new inventions but rather in the full exploitation of a carefully selected fertile ideas, and especially in showing that they can be keys to the implementation of a small and yet powerful operating system".
- UNIX is no longer a small system, but it certainly is a powerful one.
- The UNIX architecture has three important agencies-
    - Division of labor: Kernel and shell
    - The file and process
    - The system calls

- **Division of labor: Kernel and shell**
- The fertile ideas in the development of UNIX has two agencies – kernel and shell.
- The kernel interacts with the machine's hardware.
- The shell interacts with the user.

The Kernel

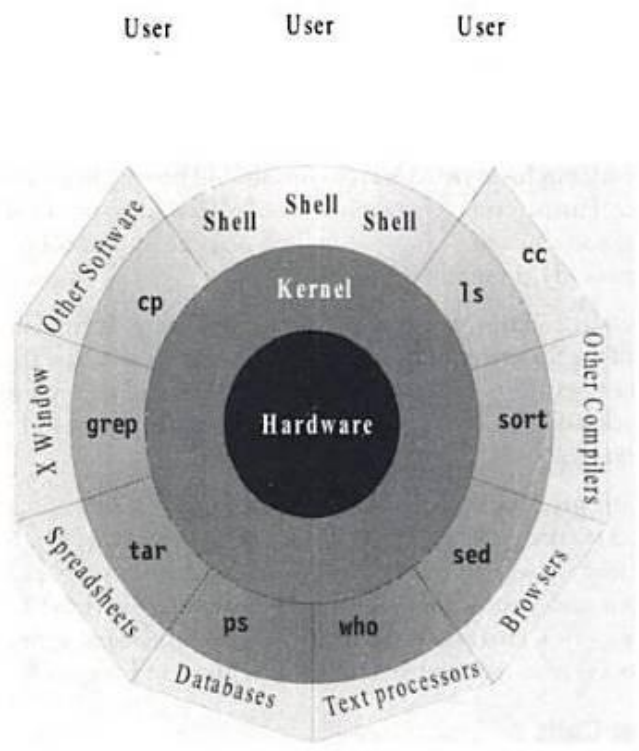
- The core of the operating system - a collection of routines mostly written in C.

- It is loaded into memory when the system is booted and communicates directly with the hardware.
- User programs (the applications) that need to access the hardware use the services of the kernel, which performs the job on the user's behalf.
- These programs access the kernel through a set of functions called system calls.
- Apart from providing support to user's program, kernel also does important housekeeping.
- It manages the system's memory, schedules processes, decides their priorities and so on.
- The kernel has to do a lot of this work even if no user program is running.
- The kernel is also called as the operating system - a programs gateway to the computer's resources.

The Shell

- Computers don't have any capability of translating commands into action.
- That requires a **command interpreter**, also called as the shell.
- Shell is actually interface between the user and the kernel.
- Most of the time, there's only one kernel running on the system, there could be several shells running – one for each user logged in.
- The shell accepts commands from user, if require rebuilds a user command, and finally communicates with the kernel to see that the command is executed.
- **Example:**
  $ echo VTU Belagavi
  #Shell rebuilds echo command by removing multiple spaces
  VTU Belagavi

The following figure shows the kernel-shell Relationship:



The file and process

- Two simple entities support the UNIX – the file and the process.
- ***"Files have places and processes have life"***

The File

- A file is just an array of bytes and can contain virtually anything.
- Every file in UNIX is part of the one file structure provided by UNIX.
- UNIX considers directories and devices as members of the file system.

The Process

- The process is the name given to the file when it is executed as a program (Process is program under execution).
- We can say process is an "time image" of an executable file.
- We also treat process as a living organism which have parents, children and are born and die.

The System Calls

- The UNIX system-comprising the kernel, shell and applications-is written in C.
- Though there are several commands that use functions called **system calls** to communicate with the kernel.
- All UNIX flavors have one thing in common – they use the same system calls.

3. FEATURES OF UNIX

UNIX is an operating system, so it has all the features an operating system is expected to have.

- A Multiuser System
- A Multitasking System
- The building-block approach
- The UNIX toolkit
- Pattern Matching
- Programming Facility
- Documentation

A Multiuser System

- UNIX is a multiprogramming system, it permits multiple programs to run and compete for the attention of the CPU.
- This can happen in two ways:
- Multiple users can run separate jobs
- A single user can also run multiple jobs

A Multitasking System

- A single user can also run multiple tasks concurrently.
- UNIX is a multitasking system.

- It is usual for a user to edit a file, print another one on the printer, send email to a friend and browse www - all without leaving any of applications.
- The kernel is designed to handle a user's multiple needs.
- In a multitasking environment, a user sees one job running in the foreground; the rest run in the background.
- User can switch jobs between background and foreground, suspend, or even terminate them.

The Building-block Approach

- The designer never attempted to pack too many features into a few tools.
- Instead, they felt "**small is beautiful**", and developed a few hundred commands each of which performed one simple job.
- UNIX offers the | (filters) to combine various simple tools to carry out complex jobs.
- Example:
  - **$ cat note** #cat displays the file contents
    WELCOME TO HIT
  - **$ cat note | wc** #wc counts number of lines, words & characters in the file
    1 3 15

The UNIX Toolkit

- Kernel itself doesn't do much useful task for users.
- UNIX offers facility to add and remove many applications as and when required.
- Tools include general purpose tools, text manipulation tools, compilers, interpreters, networked applications and system administration tools.

Pattern Matching

- UNIX features very sophisticated pattern matching features.
- Example: The * (zero or more occurrences of characters) is a special character used by system to indicate that it can match a number of filenames.

Programming Facility

- The UNIX shell is also a programming language; it was designed for programmer, not for end user.
- It has all the necessary ingredients, like control structures, loops and variables, that establish powerful programming language.
- This features are used to design shell scripts – programs that can also invoke UNIX commands.
- Many of the system's functions can be controlled and automated by using these shell scripts.

Documentation

- The principal on-line help facility available is the man command, which remains the most important references for commands and their configuration files.
- Apart from the man documentation, there's a vast ocean of UNIX resources available on the Internet.

4. POSIX AND THE SINGLE UNIX SPECIFICATION

- Dennis Ritchie's decision to rewrite UNIX in C didn't make UNIX portable.
- UNIX fragmentation and absence of single standard affected the development of portable applications.
- First, AT&T creates the System V Interface Definition (SVID).
- Later, X/Open – a consortium of vendors and users, created the X/Open Portability Guide (XPG).
- Products of this specification were UNIX95, UNIX98 and UNIX03.
- Yet another group of standards, the **POSIX** (Portable Operating System for Computer Environment) were developed by **IEEE** (Institute of Electrical and Electronics Engineers).
- Two of the most important standards from POSIX are:
- **POSIX.1** – Specifies the C application program interface – the system calls (Kernel).
- **POSIX.2** – Deals with the Shell and utilities.
- In 2001, a joint initiative of X/Open and IEEE resulted in the unification of 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 machine it can be easily ported to another POSIX compliant machine with minimum or no modification.

5. LOCATING COMMANDS

- The UNIX is command based system i.e.,- things happens because the user enters commands in.
- UNIX commands are seldom more than four characters long.
- All UNIX commands are single words like – cat, ls, pwd, date, mkdir, rmdir, cd, grep etc.
- The command names are all in **lowercase**.

- **Example:**

  $ LS

  bash: LS: command not found

- All UNIX commands are files containing programs, mainly written in C.
- All UNIX commands(files) are stored in directories(folders).
- If you want to know the location of executable program (or command), use **type** command-

- **Example:**

  $ type date

  date is /bin/date

- When you execute **date** command, the shell locates this file in the **/bin** directory and makes arrangements to execute it.

The PATH

- The sequence of directories that the shell searches to look for a command is specified in its own PATH variable.

- **Example:**

  $ echo $PATH
  /bin: /usr/bin: /usr/local/bin: /usr/ccs/bin: /usr/local/java/bin:

6. INTERNAL AND EXTERNAL COMMANDS

- When the shell execute command(file) from its own set of built-in commands that are not stored as separate files in /bin directory, it is called internal command.
- If the command (file) has an independence existence in the /bin directory, it is called external command.

- **Examples:**

  $ type echo # echo is an internal command
  echo is shell built-in

  $ type ls # ls is an external command
  ls is /bin/ls

- If the command exists both as an internal and external one, shell execute internal command only.
- Internal commands will have top priority compare to external command of same name.

7. COMMAND STRUCTURE

- To understand power of UNIX, user must know syntax of important UNIX commands.
- The general syntax of UNIX command is -

  command arguments
- Commands and arguments have to be separated by spaces or tabs to enable the system to interpret them as words.
- UNIX arguments range from the simple to the complex.
- Arguments may consist of options, expressions, instructions and filenames etc.

Options

- Options are special type of arguments mostly used with a minus(-) sign.
- An option is normally preceded by a minus(-) sign to distinguish it from filenames or other arguments.
- **Example:**

  $ ls -l note # -l option list all the attributes of the file note
  -rwxrwxrwx 1 mahesh mahesh 811 Jan 27 12:20 note

$ ls -z note # Message from ls, not from shell
ls: illegal option –z

- Options can normally be combined with only one (-) sign, i.e., instead of using **$ ls -l -a -t -d** you might as well use **$ ls -latd.**

Filename Arguments

- Many UNIX commands use a filename as argument so the command can take input from the file.
- If a command uses a filename as argument, it will generally be its last argument.
- It's also quite common to see many commands working with multiple filenames as arguments.
- The command with its arguments and options is known as the command line.
- This line can be considered complete only after the user has hit [Enter].
- The complete line is then fed to the shell as its input for interpretation and execution.
- Examples:
  - **$ ls -lat chap01 chap02 chap03** # Multiple filenames as arguments
  - **$ rm chap01 chap02**
  - **$ cp chap01 chap01.bak**

Exceptions

- There are some commands that don't accept any arguments.
- There are also some commands that may or may not be specified with arguments.
- The ls command can run without arguments (ls), with only options (ls -l) and also with only filenames like- (ls chap01 chap02)
- Examples:
  - **$ pwd** # pwd prints the current working directory
    /root
  - **$ who** # who lists currently logged in users
    mahesh tty7 2013-01-30 09:08
    mahesh pts/1 2013-01-30 10:20 (:0)

8. FLEXIBILITY OF COMMAND USAGE

- The UNIX system provides certain degree of flexibility in the usage of commands.
- A command can often be entered in more than one way.
- Shell allow the following type of command usage
  - Combining Commands
  - A command line can overflow or Be split into multiple lines
  - Entering a command before previous command has finished

Combining Commands

- UNIX allows you to specify more than one command in the single command line.
- Example:
  - **$ (wc note; ls -l note)** #Two commands combined here using **;** & parenthesis
    2 3 16 note

```
-rw-rw-r-- 1 mahesh mahesh 16 Jan 30 09:35 note
```

- **$ ls | wc** #Two commands combined here using filter

```
115 166 1227
```

A Command Line can Overflow or Be split into Multiple Lines

- UNIX terminal width is restricted to maximum **80 characters**.
- Shell allows command line to overflow or be split into multiple lines.
- Example:

```
$ echo "This is                  # $ first prompt
> a three-line                  # > Second prompt
> text message"                 #Command line ends here

This is
a three-line
text message
```

Entering a Command Before Previous Command Has Finished

- UNIX provides a full-duplex terminal which lets you type a command at any time, and rest assured that the system will interpret it.
- When you run a long program, the prompt won't appear until program execution is complete.
- Subsequent commands can be entered at the keyboard without waiting for prompt.
- The input remains stored in a buffer maintained by kernel for all keyboard input.
- The command is passed on to the shell for interpretation after the previous program has completed.

9. man: BROWSING THE MANUAL PAGES ON-LINE

- The syntax of UNIX commands can be confusing – even to the expert.
- User may not remember either the command or the required option that will perform a specific job.
- UNIX offers an on-line help facility in the man command.
- Man displays the documentation of practically every command on the system.
- For example, to seek help on the wc command, simply run man with wc as argument.
- man presents the first page and pauses.
- It does this by sending its output to a pager program, which displays this output one page at a time.
- The pager is actually a UNIX command, and man is preconfigured to be used with a specific pager.
- UNIX systems currently use the following pager programs-
  - more
  - less
- Finally, to quit the pager, and ultimately man, **press q**. You'll be returned to the shell's prompt.

Navigation and Search

- There are following two navigation commands that often used across UNIX implementations.
    - Spacebar or f – Advances the display by one screen of text at a time
    - b – which moves back one screen

10. UNDERSTANDING THE man DOCUMENTATION

- Vendors organize the man documentation differently, but in general you could see eight sections of the UNIX manual.

Understanding a man Page

- A man page is divided into a number of compulsory and optional sections.
- Every command doesn't have all sections, but first three **- NAME, SYNOPSIS and DESCRIPTION** are seen.
- **NAME** presents a one-line introduction to the command.
- **SYNOPSIS** shows the syntax used by the command.
- **DESCRIPTION** provides a detailed description.
- The SYNOPSIS section is one that we need to examine closely.
- Example:

$ man wc #Help on the wc command

User Commands wc(1)

NAME

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

SYNOPSIS

wc [-c | -m | -C] [-lw] [files....]

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.

OPTIONS

The following options are supported:

-c Count Bytes

-m Count Characters

-l Count Lines

-w Count Words

OPERANDS

The following operands are supported:

File- a path name of an input file. If no file operand are specified, the standard input will be used.

USAGE

See largefiles(5) for the behavious of wc when encountering files greater than or equal to 2 Gbyte.

EXIT STATUS

0 Successful Completion

> 0 An Error Occurred

SEE ALSO

isspace(3C), iswalpha(3C), iswspace(3C), setlocale(3C), attributes(5), environ(5), largefile(5)

Using man to understand man

- Since man is also an UNIX command like ls or cat.
- User will also like to know how man itself is used.
- Use the same command to view its own documentation:
- Example:

$ man man #Viewing man pages with man

11. FURTHER HELP WITH man -K, apropos AND whatis

- When man is used with option, it searches a summary database and prints a one-line description of the command.
- Example:
  - **$ man -k awk** #To know what awk does

    awk (1) - pattern scanning and text processing language
    mawk (1) - pattern scanning and text processing language
    nawk (1) - pattern scanning and text processing language

  - **$ apropos awk** #Same as $ man -k awk

    awk (1) - pattern scanning and text processing language
    mawk (1) - pattern scanning and text processing language
    nawk (1) - pattern scanning and text processing language

  - **$ whatis awk** #Lists one-line description of command and same as $man -f awk

    awk (1) - pattern scanning and text processing language

12. MANAGING THE NONUNIFORM BEHAVIOUR OF TERMINALS AND KEYBOARDS: WHEN THINGS GO WRONG

Terminals and keyboards have no uniform behavioral pattern. Terminal settings directly impact the keyboard operation.

Backspacing Doesn't work : Consider a word 'passwd' is misspelled as password, and when backspace key is pressed it produces some characters ^H^H^H ; backspacing is not working. It happens when user login to the remote machine whose terminal settings are different from your local one. So can use these following keys to work:

[ctrl-h] or [delete]

Killing a Line: If the command line contains a many mistakes, its possible to kill the line altogether without executing it.

[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 doesn't seem to complete. In this case to interrupt the program and bring back the prompt can use either of the two sequences:

[ctrl-c] or [delete]

Terminating a command's input: cat command uses an argument representing the filename. If filename is omitted and simple press enter

$ cat[Enter]

Command waits for user to enter something. Even if you enter any text you should know how to terminate it. For commands that expect user input, enter a **[ctrl-d]** to bring back the prompt:

$ cat
[ctrl-d]

$_

The keyboard is lacked: When this happens you are not able to key in anything. It could probably be due to accidental pressing of the key sequence **[ctrl-s].** Press **[ctrl-q]** to release the lock and restore normal keyboard operation. To resume scrolling, press **[ctrl-q].**

The [Enter] key doesn't work: This key is used to complete the command line. If it doesn't work 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. To restore the original setting press **stty sane**

The following table lists keyboard commands to try when things go wrong.

Keystroke or 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

	keyboard
[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
[Ctrl-\]	Kills running program but creates a core file containing the memory
	image of the program
[Ctrl-z]	Suspends process and returns shell prompt; use fg to resume job
[Ctrl-j]	Alternative to [Enter]
[Ctrl-m]	Alternative to [Enter]
stty sane	Restores terminal to normal status

13. cal: THE COMMAND

cal command can be used to see the calendar of any specific month or a complete year.

Syntax:

cal [[month] year]

Everything within the rectangular box in optional. So cal can be used without any arguments, in which case it displays the calendar of the current month

```
$ cal
   September 2017
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
```

The syntax show that cal can be used with arguments, the month is optional but year is not. To see the calendar of month August 2017, we need to use two arguments as shown below,

```
$ cal 8 2017
    August 2017
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
```

You can’t hold the calendar of a year in a single screen page;it scrolls off rapidly before you can use [ctrl-s] to make it pause. To make cal pause using paegr using the | symbol to connect them.

$cal 2017 | more

14. date: DISPLAYING THE SYSTEM DATE

One can display the current date with the date command, which shows the date and time to the nearest second:

$ date
Mon Sep 4 16:40:02 IST 2017

The command can also be used with suitable format specifiers as arguments. Each symbol is preceded by the + symbol, followed by the % operator, and a single character describing the format. For instance, you can print only the month using the format +%m:

$date +%m
09

Or

the month name name:

$ date +%h
Aug

Or

You can combine them in one command:

$ date + "%h %m"
Aug 08

There are many other format specifiers, and the useful ones are listed below:

- d – The day of month (1 - 31)
- y – The last two digits of the year.
- H, M and S – The hour, minute and second, respectively.
- D – The date in the format *mm/dd/yy*
- T – The time in the format *hh:mm:ss*

15. echo: Displaying the Message

echo command is used is shell scripts to display a message on the terminal, or to issue a prompt for taking user input.

$ echo "Enter your name:\c"
Enter your name:$_

$echo $SHELL
/usr/bin/bash

Echo can be used with different escape sequences

Constant	Meaning
‘a’	Audible Alert (Bell)
‘b’	Back Space
‘f’	Form Feed
‘n’	New Line
‘r’	Carriage Return
‘t’	Horizontal Tab
‘v’	Vertical Tab
‘\’	Backslash
‘\0n’	ASCII character represented by the octal value n

16. printf: AN ALTERNATIVE TO ECHO

The printf command is available on most modern UNIX systems, and is the one we can use instead of echo. The command in the smplest form can be used in the same way as echo:

```
$ printf "Enter your name\n"
Enter your name
$_
```

printf also accepts all escape sequences used by echo, but unlike echo, it doesn’t automatically insert newline unless the \n is used explicitly. printf also uses formatted strings in the same way the C language function of the same name uses them:

```
$ printf "My current shell is %s\n" $SHELL
My current shell is /bin/bash
$_
```

The %s format string acts as a placeholder for the value of $SHELL, and printf replaces %s with the value of $SHELL. %s is the standard format used for printing strings. printf uses many of the formats used by C’s printf function. Here ara some of the commonly used ones:

- %s – String
- %30s – As above but printed in a space 30 characters wide
- %d – Decimal integer
- %6d - As above but printed in a space 30 characters wide
- %o – Octal integer
- %x – Hexadecimal integer
- %f – Floating point number

Example:

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

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

17. who: WHO ARE THE USERS

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

```
$ who
mahesh   tty7        2017-09-04 16:38 (:0)
mahesh123   tty17        2017-09-04 16:38 (:0)
$_
```

Following command displays the header information with –H option,

```
$ who -H
NAME     LINE        TIME           COMMENT
mahesh   tty7        2017-09-04 16:38 (:0)
$_
```

-u option is used with who command displays detailed information of users:

```
$ who -Hu
NAME     LINE        TIME           IDLE         PID COMMENT
mahesh   tty7        2017-09-04 16:38 00:18       1865 (:0)
$_
```

18. tty: KNOWING YOUR TERMINAL

Since UNIX treats even terminals as files. tty s used display the current terminal used by user.

```
$ tty
/dev/pts/11
$_
```

Terminal filename is 11 resident in pts directory. This directory in turn is under the /dev directory.

19. stty: DISPLAYING AND SETTING TERMINAL CHARACTERISTICS

Different terminals have different characteristics. For instance command interruption may not be possible with [Ctrl-c] on your system. Sometime you may like to change the settings to match the ones used at your previous place of work.

stty uses a very large number of keywords, but we all consider only a handful of them. The –a option displays the current settings. The trimmed output is presented below:

```
$ stty -a
speed 38400 baud; rows 24; columns 116; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>; swtch
= <undef>; start = ^Q;
stop = ^S; susp = ^Z
 iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt echoctl echoke
```

Changing the settings

echoe – backspacing to erase the character

To remove the backspacing we can use the following command

$stty –echoe

To remove echo command to work we can use the following command

$stty –echo

Changing the interrupt key (intr)

To change the interrupt setting one can use the following command

$stty intr \^c

Changing the end-of-File key (eof)

To change the end-of-File key setting one can use the following command

$stty eof \^a

[ctrl-a] will now terminate input for those commands that expects input from the keyboard when invoked in a particular way.

When everything Else fail (sane)

stty also provides another argument to set the terminal characteristics to value that will work on most terminals. Use the word sane as a single argument to the command:

$stty sane #restores sanity to the terminal

20. ESSENTIAL SYSTEM ADMINISTRATION

The system administrator is also known as superuser or root user. The job of system administration involves the management of the entire system- ranging from maintaining user accounts, security and managing disk space to performing backups.

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:

1. Becoming the super at login time

Login: root
Password: ******* [Enter]**
 # -

The prompt of root is #

Once you login as a root you are placed in **root's home directory.** Depending on the system, this directory could be / or /root.
Administrative commands are resident in /sbin and /usr/sbin in modern systems and in older system it resides in /etc.

Roots PATH list includes detailed path, for example:
/sbin:/bin:/usr/sbin:/usr/bin:/usr/dt/bin

2. Becoming the super user using *su* command

su: Acquiring superuser status

Any user can acquire superuser status with the su command if they know the root password. For example, the user ***abc*** becomes a superuser in this way.

$ su
Password: ******** #Password of root user**

#pwd
/home/abc

Though the current directory doesn't change, the # prompt indicates that ***abc*** now has powers of a superuser. To be in root's home directory on superuser login, use **su –l.**

Creating a user's Environment:

User's 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** command when used with a **–** (minus), recreates the user's environment without the login-password route:

$su – abc

This sequence executes **abc's .**profile and temporarily creates **abc's** environment. su runs in a separate sub-shell, so this mode is terminated by hitting **[ctrl-d]** or using **exit.**

21. USER MANAGEMENT

For the creation and maintenance of user accounts, UNIX provides three commands - **useradd, usermod, userdel.**

When opening a user account, you have to associate the user with a group. Group usually has more than one member with a different set of privileges. Creating a group involves defining the following parameters:

- A User identification number (UID) and username
- A group identification number (GID) and groupname
- The home directory
- The login shell
- The mailbox in /var/mail
- The password

All these fields are found in a single line identifying the user in /etc/passwd file.

1. groupadd: Adding a group

If the user is to be placed in a new group, an entry for the group has to be created first in the **/etc/group**. User will always have one primary group and one or more **supplementary groups.**

/etc/group file contains all of the named groups of the system, and a few lines of this file have the structure as follow:

root:x:0:root
bin:x:2:
lp:x:7:
usp:x:18: Ram, krishn, image
class:x:1000:

Each line contains **four colon-delimited fields.**

- The first field indicates the **group ownership.**
- The second field **Group password** but it is hardly used today; its either blank or x.
- The third field shows the **GID** (Group ID -1000).
- The last field contains a list of comma-delimited **usernames** for whom this is the supplementary group. Blank at this position indicates not the supplementary group for any user.

To create a new group usp with GID 2015, use groupadd command as follow,

#groupadd -g 2015 usp ***#2015 is the GID for usp***

The command places this entry in **/etc/group,** which can also be inserted manually.

usp:x:2015:

Once an entry for the group has been made, then we can add a user of this group to system

2. Useradd: Adding a user

The useradd command adds a new users to the system. All details to the user are provided in the command line:

 # useradd -u 999 -g class -c "Unix and shell programming" -d /home/usp -s /bin/ksh -m usp
#_

- This creates the user usp with UID 999 and group name class.
- The home directory is /home/usp and user will use the Korn shell.
- The -m option ensure that the home directory is created if it doesn't already exist and copies a sample .profile and .kshrc to the user's home directory.

The line useradd creates in **/etc/passwd** as:

usp:x:999:2015: Unix and shell programming:/home/usp:/bin/ksh

Useradd sets up the user's mailbox and sets the MAIL variable to point to that location(/var/mail). Can set new users password with the command **passwd usp.**

/etc/passwd and /etc/shadow: User profile

All user information is stored in /etc/passwd except the password encryption; it's stored in **/etc/passwd**

/etc/shadow

Encrypted password is stored in **/etc/shadow** file, this is the control file used by passwd to ascertain the legitimacy of a user's password.
For every line in **/etc/passwd,** there's a corresponding entry in **/etc/shadow.** The relevant line in this file could look like:

usp:ggytai749sditjm:999:::::::

This password encryption is stored in the second field. It's impossible to generate password from this encryption. This file is made unreadable for all users for security reason. Only superuser can access this file.

Each field in the shadow file is separated with ":" colon characters, and are as follows:

- Username, up to 8 characters. Case-sensitive, usually all lowercase. A direct match to the username in the /etc/passwd file.
- Password, 13 character encrypted. A blank entry (eg. ::) indicates a password is not required to log in (usually a bad idea), and a ``*'' entry (eg. :*:) indicates the account has been disabled.
- The number of days (since January 1, 1970) since the password was last changed.
- The number of days before password may be changed (0 indicates it may be changed at any time)
- The number of days after which password *must* be changed (99999 indicates user can keep his or her password unchanged for many, many years)
- The number of days to warn user of an expiring password (7 for a full week)
- The number of days after password expires that account is disabled
- The number of days since January 1, 1970 that an account has been disabled
- A reserved field for possible future use

/etc/passwd

There are total seven fields in each line of the /etc/passwd file. Their significance are as follows:

- **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 numerical identification No two users should have the same UID. **ls** command prints the owner's name by matching the UID obtained from the inode with this field.
- **GID:** The user's numerical group identification
- **Comments or GCOS**: User details, name address. This name is used at the front of the email address for this user

- **Home directory**: The directory where the user ends up on logging in. The **login** program reads this field to set the variable HOME
- **Login shell**: The first program executed after logging in This is usually the shell (/bin/ksh). Login sets the variable SHELL by reading this entry, and also **fork-execs** the shell process.

3. usermod and userdel: Modifying and removing Users

usermod is used for modifying some of the parameters set with useradd. Any parameters can be modified by specifying corresponding options to usermod command.
For example sometimes user need to change their login shell. Command to set **Bash** as the login shell for the user usp is:

#usermod -s /bin/bash usp #changes user usp's shell from ksh to bash shell

Users are removed from the system with the **userdel** Command to delete user usp from the system is:

#userdel usp #deletes user usp from system

Removes all entries pertaining to **usp** from **/etc/passwd, etc/group** and **/etc/shadow**
The user's home directory doesn't get deleted in the process and has to be removed separately if required.

Summary:

Managing Users and Groups

Command	Description
useradd	Adds accounts to the system.
usermod	Modifies account attributes.
userdel	Deletes accounts from the system.
groupadd	Adds groups to the system.

Option	Description
-g GID	The numerical value of the group's ID.
groupname	Actaul group name to be created.

Option	Description
-d homedir	Specifies home directory for the account.
-g groupname	Specifies a group account for this account.
-m	Creates the home directory if it doesn't exist.
-s shell	Specifies the default shell for this account.
-u userid	You can specify a user id for this account.
Account name	Actual account name to be created