Introduction

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.

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.

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.

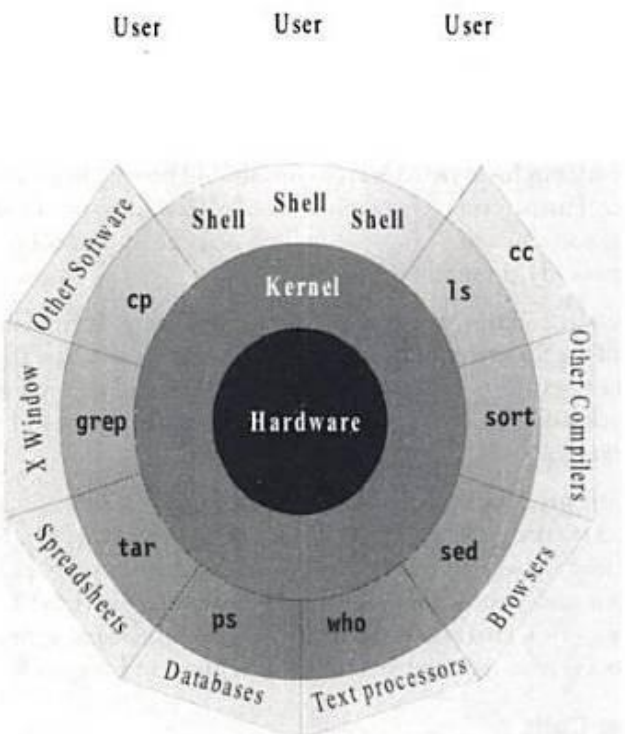
- Most of the time, there's only one kernel running on the system, there could be several shells running – one for each user logged in.
- The shell accepts commands from user, if require rebuilds a user command, and finally communicates with the kernel to see that the command is executed.

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.

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 KIT

  > **$ 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.

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.

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:

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.

COMMAND STRUCTURE

- To understand power of UNIX, user must know syntax of important UNIX commands.
- The general syntax of UNIX command is –
  command [-option(s)] [argument(s)]
- 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 santhosh santhosh 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)
- Examples:

  > **$ pwd** # pwd prints the present current working directory /root

  > **$ who** # who lists currently logged in users santhosh tty7 2013-01-30 09:08

santhosh pts/1 2013-01-30 10:20 (:0)

UNIX Commands Structure

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

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

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

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

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! Historically, UNIX commands have been fairly standard in the way that they use options but there are variations - so be aware!

Since the introduction of UNIX System V, Release 3, any new commands must obey a particular syntax governed by the following rules:

- Command names must be between 2 and 9 characters in length
- Command names must be comprised of lowercase characters and digits
- Option names must be one character in length
- All options are preceded by a hyphen (-)
- Options without arguments may be grouped after the hyphen

- The first option argument, following an option, must be preceded by white space. For example

  -o sfile is valid but **-osfile** is illegal.

- Option arguments are not optional
- If an option takes more than one argument then they must be separated by commas with no spaces, or if spaces are used the string must be included in double quotes (").

  For example, both of the following are acceptable:
  -f past,now,next and **-f "past now next"**

- All options must precede other arguments on the command line
- A double hyphen -- may be used to indicate the end of the option list
- The order of the options are order independent
- The order of arguments may be important
- A single hyphen - is used to mean standard input

You should bear in mind that commands established before System V, Release 3, do not conform to all of the above rules.

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 santhosh santhosh 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.

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

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

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

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

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

12 3 4 5

6 7 89 10 11 12

13 14 1516 17 18 19

20 21 2223 24 25 26

27 28 2930 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

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*

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

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 are 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

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

santhosh tty7 2017-09-04 16:38 (:0)

santhosh123 tty17 2017-09-04 16:38 (:0)

$_

Following command displays the header information with –H option,

$ who -H

NAME LINE TIME COMMENT

santhosh 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 santhosh tty7

2017-09-04 16:38 00:18 1865 (:0)

$_

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.

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

ESSENTIAL SYSTEM ADMINISTRATION

The system administrator is also known as super user 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:

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

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.**

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.

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

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.

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

Unix Files

THE FILE

- The file is the container for storing information.
- Neither a file's size nor its name is stored in file.
- All file attributes such as file type, permissions, links, owner, group owner etc are kept in a separate area of the hard disk, not directly accessible to humans, but only to kernel.
- The UNIX has divided files into three categories:

1. **Ordinary file** – also called as regular file. It contains only data as a stream of characters.
2. **Directory file** – it contains files and other sub-directories.
3. **Device file** – all devices and peripherals are represented by files.

Ordinary File - ordinary file itself can be divided into two types-

1. **Text File** – it contains only printable characters, and you can often view the contents and make sense out of them.
2. Binary file – it contains both printable and unprintable characters that cover entire ASCII range.

Examples- Most Unix commands, executable files, pictures, sound and video files are binary.

Directory File - a directory contains no data but keeps some details of the files and ubdirectories that it contains. A directory file contains an entry for every file and subdirectories that it houses. If you have 20 files in a directory, there will be 20 entries in the directory. Each entry has two components-

- the filename
- a unique identification number for the file or directory (called as inode number).

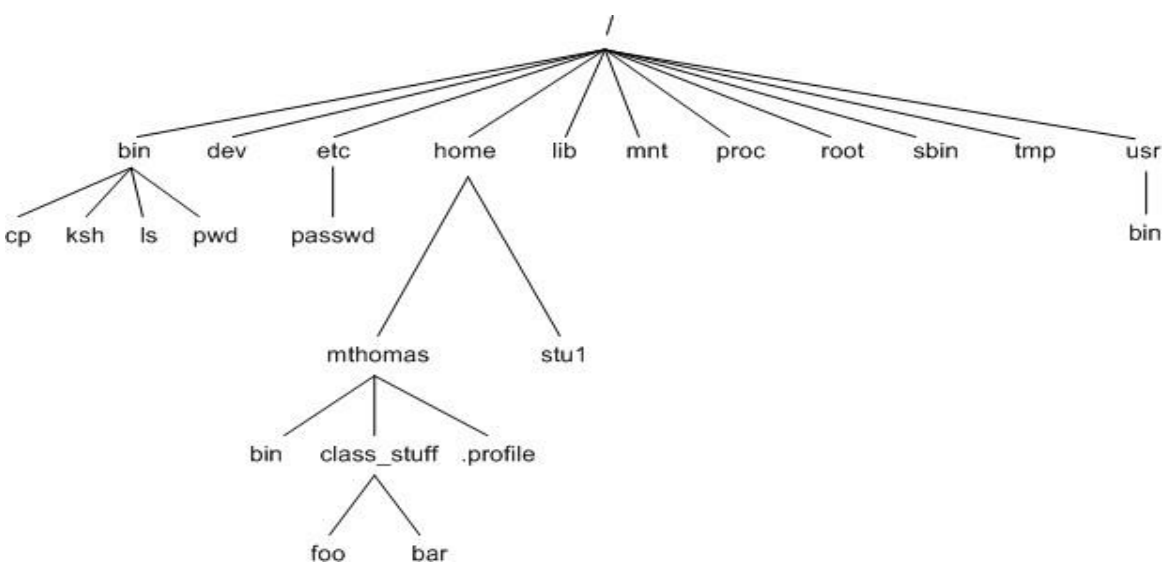
Device File - Installing software from CD-ROM, printing files and backing up data files to tape. All of these activities are performed by reading or writing the file representing the device. Advantage of device file is that some of the commands used to access an ordinary file also work with device file. Device filenames are generally found in a single directory structure, /dev.

WHAT'S IN A (FILE) NAME?

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

THE PARENT-CHILD RELATIONSHIP

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



- The feature of UNIX file system is that there is a top, which serves as the reference point for all files.
- This top is called root and is represented by a / (Front slash).
- The root is actually a directory.
- The root directory (/) has a number of subdirectories under it.

- The subdirectories in turn have more subdirectories and other files under them.
- Every file apart from root, must have a parent, and it should be possible to trace the ultimate parentage of a file to root.
- In parent-child relationship, the parent is always a directory.

The HOME VARIABLE: HOME DIRECTORY

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

$echo $HOME

/home/kumar

pwd: CHECKING YOUR CURRENT DIRECTORY

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

$ pwd

/home/kumar

- Like HOME it displays the absolute path.

cd: CHANGING THE CURRENT DIRECTORY

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

$ pwd
/home/kumar
$cd progs
$ pwd
/home/kumar/progs

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

$cd /home/kumar/progs

$ pwd

/home/kumar/progs

$cd /bin

$ pwd

/bin

- cd can also be used without arguments:

$ pwd

/home/kumar/progs

$cd

$ pwd

/home/kumar

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

$cd /home/sharma

$ pwd

/home/sharma

$cd

/home/kumar

mkdir: MAKING DIRECTORIES

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

$mkdir patch

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

$mkdir patch dba doc

- For instance the following command creates a directory tree:

$mkdir progs progs/cprogs progs/javaprogs

- This creates three subdirectories – progs, cprogs and javaprogs under progs.

- The order of specifying arguments is important. You cannot create subdirectories before creation of parent directory.
- For instance following command doesn‘t work

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

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

rmdir: REMOVING A DIRECTORY

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

$rmdir progs

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

$mkdir progs progs/cprogs progs/javaprogs rmdir: directory “progs”: Directory not empty

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

$mkdir progs/cprogs progs/javaprogs progs

- First it removes **cprogs** and **javaprogs** form **progs** directory and then it removes **Progs** from system.
- **rmdir : Things to remember**
- You can‘t remove a directory which is not empty
- You can‘t remove a directory which doesn‘t exist in system.
- You can‘t remove a directory if you don‘t have permission to do so.

ABSOLUTE PATHNAME

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

/etc/passwd

/users/kumar/progs/cprogs

/dev/rdsk/Os3

Example

- date command can executed in two ways as

$date // Relative path

Thu Sep 7 10:20:29 IST 2017

$/bin/date // Absolute path

Thu Sep 7 10:20:29 IST 2017

RELATIVE PATHNAME

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

progs/cprogs rdsk/Os3

Using . and .. in relative path name

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

$pwd

/home/kumar/progs/cprogs

$cd /home/kumar

$pwd

/home/kumar

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

$pwd

/home/kumar/progs/cprogs

$cd ..

$pwd

/home/kumar/progs

- This method is compact and easy when ascending the directory hierarchy. The command **cd ..** Translates to this –change your current directory to parent of current directory‖.
- The relative paths can also be used as:

$pwd

/home/kumar/progs

$cd ../..

$pwd

/home

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

$pwd

/home/kumar/progs/cprogs

$cp ../javaprogs/prog1.java .

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

FILE RELATED COMMANDS

cat: DISPLAYING AND CREATING FILES

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

$ cat cprogram.c # include <stdioh> void main ()

{

Printf(−hello‖);

}

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

$ cat ch1 ch2

It contains the contents of chapter1 It contains the contents of chapter2

In this the contents of the second files are shown immediately after the first file without any header information. So cat concatenates two files- hence its name.

cat OPTIONS

- **Displaying Nonprinting Characters (-v)**

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

- **Numbering Lines (-n)**

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

Using cat to create a file

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

$ cat > new

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

[ctrl-d]

$_

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

$ cat new

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

cp: COPYING A File

The **cp** command copies a file or a group of files. It creates an exact image of the file on the disk with a different name. The **syntax takes two filename** to be specified in the command line.

- When both are ordinary files, **first file is copied to second.**

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

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

  $ cp ch1 cs ch1 retains its name under cs
- cp can also be used with the shorthand notation, .(dot), to signify the current directory as the destination. To copy a file **„new“** from /home/user1 to your current directory, use the following command:

  $cp /home/user1/new new ***destination is a file***

  $cp /home/user1/new . ***destination is the current directory***
- cp command can be used **to copy more than one file with a single invocation of the command.** In this case the last filename must be a directory.
- **Ex: To copy the file ch1,chh2,ch3 to the module , use cp as**

  $ cp ch1 ch2 ch3 module
- The files will have the same name in **module**. If the files are already resident in **module**, they will be overwritten. In the above diagram module directory should already exist and cp doesn‘t able create a directory.
- UNIX system uses * as a shorthand for multiple filenames.

  Ex: $ cp ch* usp Copies all the files beginning with ch

cp options

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

  $ cp -i ch1 unit1

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

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

Copying directory structure (-R) :

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

  $ cp -R usp newusp

  $ cp -R class newclass

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

rm: DELETING FILES

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

$ rm mod1 mod2 mod3

- Can remove two chapters from usp directory without having to cd

  Ex: $rm usp/marks ds/marks

- To remove all file in a directory use ***

  $ rm *

- Removes all files from that directory

rm options

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

  $ rm -i ch1 ch2

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

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

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

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

  $ rm -r * ***Works as rmdir***

- It deletes all files in the current directory and all its subdirectories.
- **Forcing Removal (-f): rm** prompts for removal if a file is **write-protected.** The **–f** option overrides this minor protection and forces removal.

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

mv: RENAMING FILES

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

  Ex: To rename the file csb as csa we can use the following command

  $ mv csb csa

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

  $ mv ch1 ch2 ch3 module

- Can also used to rename directory

$ mv rename newname

- mv replaces the filename in the existing directory entry with the new name. It doesn't create a copy of the file; it renames it
- Group of files can be moved to a directory
- mv chp1 chap2 chap3 **unix**

more : PAGING OUTPUT

- To view the file ch1, we can use more command along with the filename, it is used for display

$ more odfile press q to exit

- this file is an example for od command ^d used as an interrupt key ^e indicates the end of file.
- It displays the contents of ch1 on the screen, one page at a time. If the file contents is more it will show the filename and percentage of the file that has been viewed:
  ----More--- (15%)

Navigation

- f or Spacebar: to scroll forward a page at a time
- b to move back one page

Using more in pipeline

- The **ls** outpu won‘t fit on the screen if there are too many files, So the command can be used like this:

ls | more

- The pipeline of two commands where the output of two commands, where the output of one is used as the input of the other.

wc: COUNTING LINES,WORDS AND CHARACTERS

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

$ wc ofile

4 20 97 ofile

- Line: Any group of characters not containing a newline
- Word: group of characters not containing a space, tab or newline

- Character: smallest unit of information, and includes a space, tab and newline
- **wc** offers 3 options to make a specific count. –l option counts only number of lines, - w and –c options count words and characters, respectively.

  $ wc -l ofile 4 ofile

  $ wc -w ofile 20 ofile
- Multiple filenames, **wc** produces a line for each file, as well as a total count.

  $ wc -c ofile file 97 ofile

  15 file

  112 total

od: DISPLAYING DATA IN OCTAL

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

  $ more ofile

  this file is an example for od command

  ^d used as an interrupt key
- -b option displays this value for each character separately.
- Each line displays 16 bytes of data in octal, preceded by the **offset in the file of the first byte in the line.**

$ od –b file

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

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

$ od –bc file

od -bc ofile

```
0000000 164 150 151 163 040 146 151 154 145 040 151 163 040 141 156 040
          t   h   i   s       f   i   l   e       i   s       a   n

0000020 145 170 141 155 160 154 145 040 146 157 162 040 157 144 040 143
          e   x   a   m   p   l   e       f   o   r       o   d       c

0000040 157 155 155 141 156 144 012 136 144 040 165 163 145 144 040 141
          o   m   m   a   n   d  \n   ^   d       u   s   e   d       a

0000060 163 040 141 156 040 151 156 164 145 162 162 165 160 164 040 153
          s       a   n       i   n   t   e   r   r   u   p   t       k

0000100 145 171
          e   y
```

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

BASIC FILE ATTRIBUTES

The UNIX file system allows the user to access other files not belonging to them and without infringing on security. A file has a number of attributes (properties) that are stored in the inode. In this chapter, we discuss,

- ls –l to display file attributes (properties)
- Listing of a specific directory
- Ownership and group ownership
- Different file permissions

LISTING FILE ATTRIBUTES

ls command is used to obtain a list of all filenames in the current directory. The output in UNIX lingo is often referred to as the listing. Sometimes we combine this option with other options for displaying other attributes, or ordering the list in a different sequence. ls look up the file‘s inode to fetch its attributes. **It lists seven attributes of all files in the current directory and they are:**

- **File type and Permissions**
  - The file type and its permissions are associated with each file.
- **Links**
  - Links indicate the number of file names maintained by the system. This does not mean that there are so many copies of the file.
- **Ownership**
  - File is created by the owner. The one who creates the file is the owner of that file.
- **Group ownership**
  - Every user is attached to a group owner. Every member of that group can access the file depending on the permission assigned.
- **File size**
  - File size in bytes is displayed. It is the number of character in the file rather than the actual size occupied on disk.
- **Last Modification date and time**
  - Last modification time is the next field. If you change only the permissions or ownership of the file, the modification time remains unchanged. If at least one character is added or removed from the file then this field will be updated.
- **File name**
  - In the last field, it displays the file name.

For example,

$ ls -l

total 72

-rw-r--r-- 1 kumar metal 19514 may 10 13:45 chap01

-rw-r--r-- 2 kumar metal 19555 may 10 15:45 chap02

drwxr-xr-x 2 kumar metal 512 may 09 12:55 helpdir

drwxr-xr-x 3 kumar metal 512 may 09 11:05 progs

Listing Directory Attributes

$ls -d

This command will not list all subdirectories in the current directory .

For example, $ls –ld helpdir progs

drwxr-xr-x 2 kumar metal 512 may 9 10:31 helpdir

drwxr-xr-x 2 kumar metal 512 may 9 09:57 progs

- Directories are easily identified in the listing by the first character of the first column, which here shows a d.
- The significance of the attributes of a directory differs a good deal from an ordinary file.
- To see the attributes of a directory rather than the files contained in it, use ls –ld with the directory name. Note that simply using ls –d will not list all subdirectories in the current directory. Strange though it may seem, ls has no option to list only directories.

File Ownership

- When you create a file, you become its owner. Every owner is attached to a group owner. Several users may belong to a single group, but the privileges of the group are set by the owner of the file and not by the group members. When the system administrator creates a user account, he has to assign these parameters to the user:

The user-id (UID) – both its name and numeric representation The group-id (GID) – both its name and numeric representation

File Permissions

UNIX follows a three-tiered file protection system that determines a file's access rights. It is displayed in the following format: Filetype owner (rwx) groupowner (rwx) others (rwx)

For Example: **-rwxr-xr-- 1 kumar metal 20500 may 10 19:21 chap02**

rwx r-x r-- owner/user group owner others

- The first group has all three permissions. The file is readable, writable and executable by the owner of the file.
- The second group has a hyphen in the middle slot, which indicates the absence of write permission by the group owner of the file.
- The third group has the write and execute bits absent. This set of permissions is applicable to others.
- You can set different permissions for the three categories of users – owner, group and others. It's important that you understand them because a little learning here can be a dangerous thing. Faulty file permission is a sure recipe for disaster.

CHANGING FILE PERMISSIONS

A file or a directory is created with a default set of permissions, which can be determined by umask. Let us assume that the file permission for the created file is -rw-r-- r--. Using chmod command, we can change the file permissions and allow the owner to execute his file.

The command can be used in two ways:

- In a **relative** manner by specifying the changes to the current permissions
- In an **absolute** manner by specifying the final permissions

Relative Permissions

- chmod only changes the permissions specified in the command line and leaves the other permissions unchanged.
- Its **syntax** is: **chmod category operation permission filename(s)**
- chmod takes an expression as its argument which contains:
- user category (user, group, others)
- operation to be performed (assign or remove a permission)
- type of permission (read, write, execute)

- **Category : u – user g – group o – others a - all (ugo)**
- **operations : + assign - remove = absolute**
- **permissions: r – read w – write x - execute**

Let us discuss some examples:

- Initially,

  -rw-r—r-- 1 kumar metal 1906 sep 23:38 xstart

  $chmod u+x xstart

  -rwxr—r-- 1 kumar metal 1906 sep 23:38 xstart
- The command assigns (+) execute (x) permission to the user (u), other permissions remain unchanged.

  $chmod ugo+x xstart or chmod a+x xstart or chmod +x xstart

  $ls –l xstart

  -rwxr-xr-x 1 kumar metal 1906 sep 23:38 xstart
- chmod accepts multiple file names in command line

  $chmod u+x note note1 note3

- Let initially,

  -rwxr-xr-x 1 kumar metal 1906 sep 23:38 xstart

  $chmod go-r xstart
- Then, it becomes

  $ls –l xstart

  -rwx—x--x 1 kumar metal 1906 sep 23:38 xstart

Absolute Permissions

Here, we need not to know the current file permissions. We can set all nine permissions explicitly. A string of three octal digits is used as an expression. The permission can be represented by one octal digit for each category. For each category, we add octal digits. If we represent the permissions of each category by one octal digit, this is how the permission can be represented:

Read permission – 4 (octal 100)

Write permission – 2 (octal 010)

Execute permission – 1 (octal 001)

Octal	Permissions	Significance
0	---	no permissions
1	--x	execute only
2	-w-	write only
3	-wx	write and execute
4	r--	read only
5	r-x	read and execute
6	rw-	read and write
7	rwx	read, write and execute

We have three categories and three permissions for each category, so three octal digits can describe a file‘s permissions completely. The most significant digit represents user and the least one represents others. chmod can use this three-digit string as the expression.

Using relative permission, we have,

$chmod a+rw xstart

Using absolute permission, we have,

$chmod 666 xstart

$chmod 644 xstart

$chmod 761 xstart

Will assign all permissions to the owner, read and write permissions for the group and only execute permission to the others.

- 777 signify all permissions for all categories, but still we can prevent a file from being deleted.
- 000 signifies absence of all permissions for all categories, but still we can delete a file.
- It is the directory permissions that determine whether a file can be deleted or not.
- Only owner can change the file permissions. User cannot change other user‘s file‘s permissions.
- But the system administrator can do anything.

The Security Implications

- Let the default permission for the file xstart is

  -rw-r—r- -

  $chmod u-rw, go-r xstart or chmod 000 xstart

- This is simply useless but still the user can delete this file.

  On the other hand,

  $chmod a+rwx xstart or chmod 777 xstart

  -rwxrwxrwx

- The UNIX system by default, never allows this situation as you can never have a secure system. Hence, directory permissions also play a very vital role here .

We can use chmod Recursively.

$chmod -R a+x shell_scripts

This makes all the files and subdirectories found in the shell_scripts directory, executable by all users. When you know the shell meta characters well, you will appreciate that the * doesn‘t match filenames beginning with a dot. The dot is generally a safer but note that both commands change the permissions of directories also.

Directory Permissions

It is possible that a file cannot be accessed even though it has read permission, and can be removed even when it is write protected. The default permissions of a directory are,

rwxr-xr-x (755)

- A directory must never be writable by group and others .

Example: **$mkdir c_progs**

$ls –ld c_progs

drwxr-xr-x 2 kumar metal 512 may 9 09:57 c_progs

- If a directory has write permission for group and others also, be assured that every user can remove every file in the directory. As a rule, you must not make directories universally writable unless you have definite reasons to do so.

Changing File Ownership

- Usually, on BSD and AT&T systems, there are two commands meant to change the ownership of a file or directory. Let kumar be the owner and metal be the group owner. If sharma copies a file of kumar, then sharma will become its owner and he can manipulate the attributes.
- chown changing file owner and chgrp changing group owner
- On BSD, only system administrator can use chown
- On other systems, only the owner can change both

chown

- Changing ownership requires super user permission, so use su command

$ls -l note

-rwxr-----x 1 kumar metal 347 may 10 20:30 note

$chown sharma note; ls -l note

-rwxr-----x 1 sharma metal 347 may 10 20:30 note

- Once ownership of the file has been given away to sharma, the user file permissions that previously applied to Kumar now apply to sharma. Thus, Kumar can no longer edit note since there is no write privilege for group and others. He cannot get back the ownership either. But he can copy the file to his own directory, in which case he becomes the owner of the copy.

chgrp

- This command changes the file‘s group owner. No super user permission is required.

#ls –l dept.lst

-rw-r—r-- 1 kumar metal 139 jun 8 16:43 dept.lst

#chgrp dba dept.lst; ls –l dept.lst

-rw-r—r-- 1 kumar dba 139 Jun 8 16:43 dept.lst