Unit 1 ULP

Linux Startup and Shell Programming

Linux operating System:→ An operating System is a system software that helps user to operate the Computer. It acts as an interface between the user and the Computer. without OS we can not do anything. It manages the Computer hardware and provides the environment to the user for execution of programs. It also manages application software installed on your Computer like MS-Word can not work without an operating System.

→ File Management, Process management and memory management are common features found in all OS.

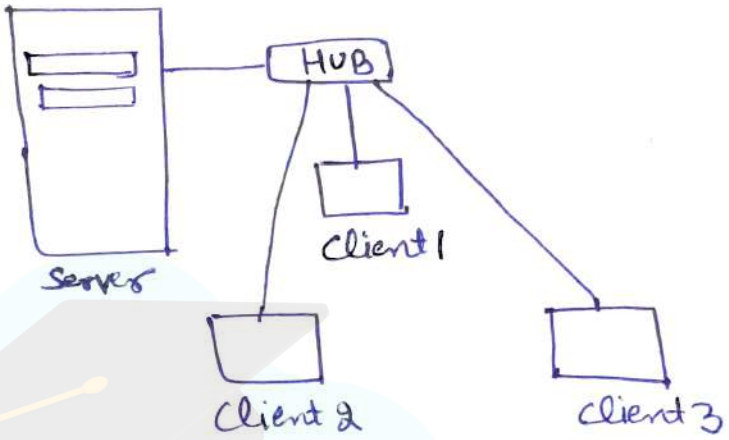
→ Linux adds two more features to that one is multiuser and another is multitasking.

→ Multitasking means that if you are working on Linux O.S you can perform several tasks at the same time. It means if you have given print command to a document you can also edit the another

document at the same time, you do not have to wait for the printing to finish before you start editing.

Multiuser → operating system is basically of two types:

1. Single user operating system.
2. Multi user operating system.



In a single user OS we have to install operating system in the server and all three clients. But if we are using Multi user OS like Linux then we have install operating system in server only and all three clients use OS from server. In Multiuser Every client have a login and password. User1, User2 and User3 can sit on any machine with login and password. But in single user User1, User2 and User3 will be given a computer which will be fixed for him which means user1 can sit only on one client and will have to use same computer.

Single-user OS	Multi-user OS
1. operating system will have to be installed on every computer in a network	1. Operating System will have to be installed only on one Computer (Server) in a network.
2. High Cost	2. Low Cost
3. Extra Hardware required	3. No Extra Hardware required.
4. less secured	4. High Secured

History of Linux: →

History of Linux begins with Unix.

In 1960 a special operating system called MULTICS(Multiplexed Information and Computing Service) was developed at AT&T lab. It had many features like multitasking, user interaction and process management. But MULTICS was not affordable for smaller organisations and can not be installed on minicomputers.

In 1969 Ken Thompson and Denis Ritchie developed Unix at AT&T lab by using many features of MULTICS OS. It was designed to run on minicomputers and it was affordable.

In 1973, Unix was written again by Denis Ritchie and Ken thompson in C programming language. The first version of Unix was distributed free of cost to Computer science department of many universities like California, Berkeley. Berkeley made some changes to Unix and released their own version of Unix with the name BSD (Berkeley Software Distribution).

In 1970 it was adopted by DAPRA.

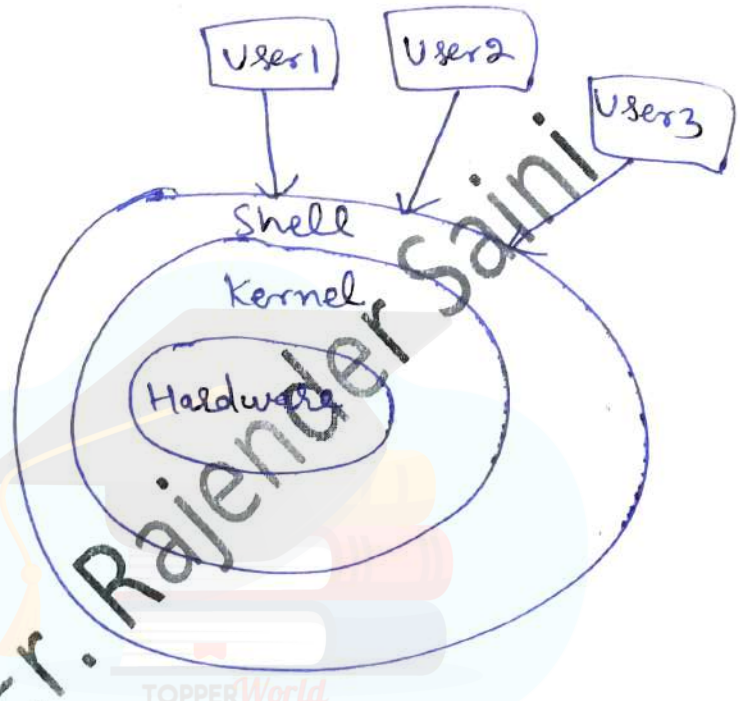
By the mid 1980's there were mainly two competitors of Unix one was AT&T and another was BSD.

Unix developed for the commercial purpose not for education purpose. After some changes Andrew Tenenbaum developed educational version of Unix called Minix. The Minix was modified by a student Linus Torvalds to make PC version of Minix. It was named Linux and in 1991 Linux released version 0.11/Linux was distributed over the internet and programmers keep on adding new features into it. Linux has all os basic features and services like networking, email and web browsers etc.

LINUX Architecture (Layout of Linux) →

The Linux Architecture has three Layers.

→ Hardware
→ Kernel
→ Shell



* Hardware → Hardware includes all the peripherals we use while we are working on the computer. For example keyboard, Mouse and hard disk are the most common peripherals used.

* Kernel → It is the heart of operating System. It is the main program in the linux operating

system that helps to run applications and manages hardware devices such as printers, storage devices etc. It is the core of operating system. When the system is booted it is loaded into memory and communicates directly with the hardware. The application programs access the Kernel through a set of functions known as System calls.

* Shell → It is a command Interpreter that provides interactive and non-interactive interface between the user and the operating system. With the help of shell the user interact with the operating system. The command entered by the user on command line and interpreted by shell and then sent to the Kernel. The Kernel then accesses the hardware and sent result back to the shell. If command is not valid then error message is display.

User Accounts →

When a Computer is used by many people it is usually necessary to differentiate between them so that their private files can be kept private. This is very important in multiuser operating system.

Thus each users is given a unique username and that name is used to login. An Account is all the files, resources and information belonging to the user.

The Linux Kernel tracks users as mese numbers. Each user is identified by a unique integer, the "user-id" or UID. Because number is faster and easier for a Computer to process than textual names.

There are also a few commands for changing various proprtion of an account like:

→ chfn → changes the file name field.
→ Chsh → changes the login Shell.
→ Passwd → changes the password.

Starting and Shutting Processes →

Whenever the Linux system is booted, a number of processes start executing at time. Though we can not see these processes, they keep executing in the background. These processes are called as daemons.

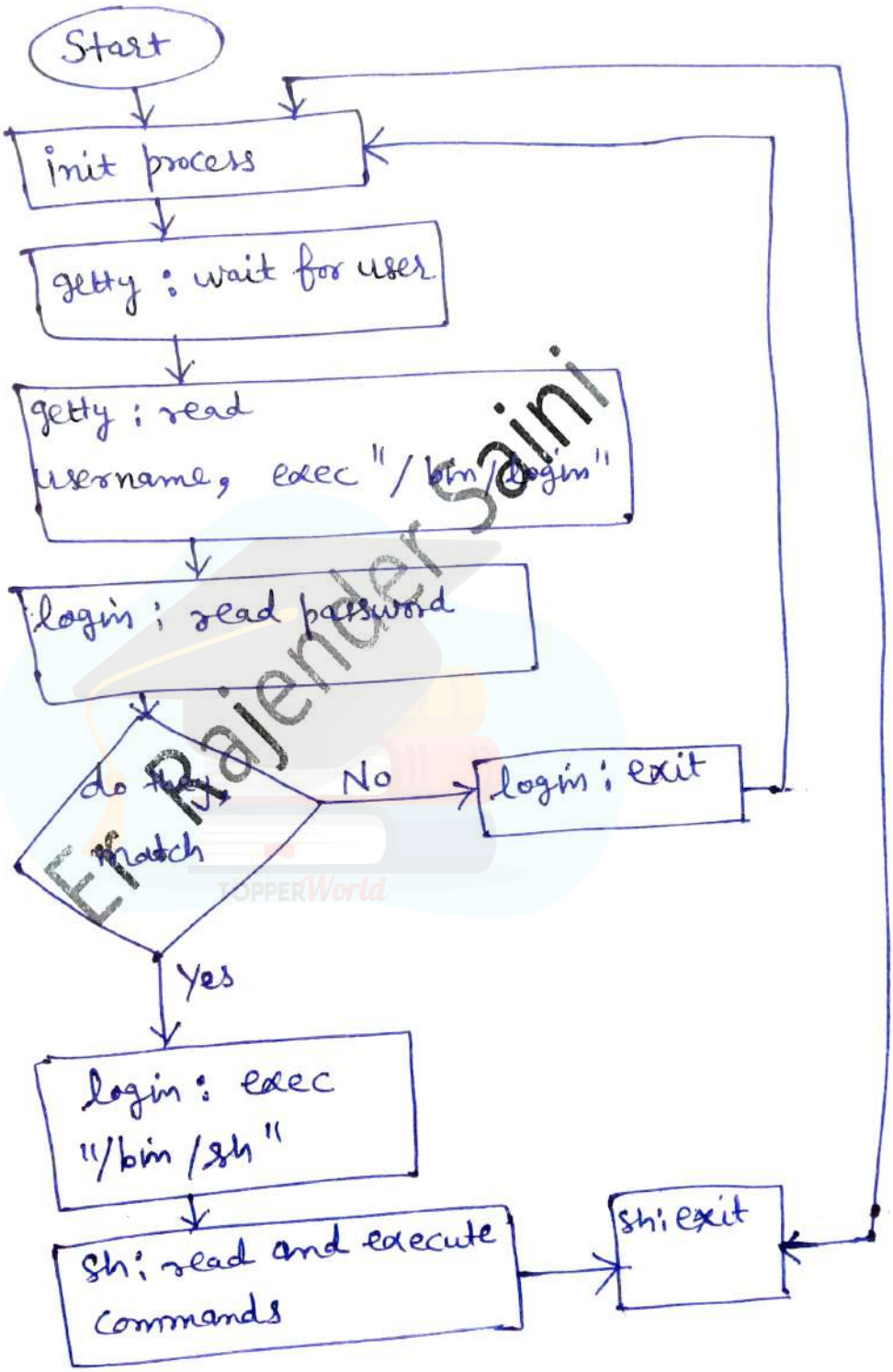
For example, "init" is a daemon which keep executing as long as the system is running. This daemon spawns a lot of children processes and generally called as the ancestor of the the other processes or the system.

First, "init" makes sure that there is a "getty" program for the terminal connection. "getty" listen at the terminal and wait for the user ~~that~~ to notify that he is ready for login.

when it notices a user, "getty" output a welcome message and prompts for the username and finally runs the "login" program.

"Login" gets the username as a parameter and prompts the user for the password.

If these match, login starts the shell.



Init Process

Logging in and Logging out →

You can access a Linux system in two ways:

① either locally (by sitting at the computer in which Linux is installed or

② remotely (by sitting on the client machine on the network in a multiuser environment).

If you are working in a multiuser environment then it is duty of system administrator to provide you a login name. If you are the only user on the system then you will be the system administrator. In that case login name and password will be set at the time of Installation of Linux.

Login name contain alphanumeric character which include alphabets, numbers. The login name must be more than 02 characters long and if it is more than 08 characters then only first 08 characters are considered. The login name must contain only lowercase letters and numbers and must begin with a lowercase letter.

Password must have atleast 06 characters. It must contain atleast 02 alphabets which can be upper or lowercase.

① zip command :

To zip a file →

$ zip filename.zip filename

filename.zip → Output file as a zipper file

filename → existing file

② To unzip a file

$ unzip filename.zip ↵

output filename →

③ To zip a directory →

$ zip directoryname.zip dirname

directoryname.zip → output directory a a zipper folder

dirname → existing directory

④ To unzip a directory | folder →

$ unzip directoryname.zip

output → directoryname

(5) gzip command

$ gzip file1.C
↑ files or folder

output → file1.C.gz

(6) To uncompress gzip file

6.1 $ gunzip file1.C.gz

6.2 $ gzip -d file1.C.gz

(7) bzip2 command →

$ bzip2 file1.C ↵

file1.C.bz2

(8) uncompress bzip2 →

$ bzip2 -d file1.C.bz2 ↵

output → file1.C

(9) Compress file using lzma →

$ lzma file.c

output file1.c.lzma

(10) uncompress file of lzma →

$ lzma -d file1.c.lzma

output → file1.c

(11) Compress file using x2 format →

$ x2 file1.c

output → file1.c.x2

(12) uncompress x2 file →

$ x2 -d file1.c.x2

output → file1.c

Login | Logout Commands

$ Login

Login : hctm
Password : ___ ↵

$ Logout or exit

$ passwd

Changing password for current user
(Current) Linux password : ___
New password :
: It is based on a dictionary word
: It doesnot contain enough different characters
After time

Passwd : Have exhausted max. no. of retries for servic.

Simple Commands

1. $ Date → Display current System Date

   Tue Aug 25 17:51:15 EDT 2015

   EDT → Eastern Daylight time

2. $ Cal → display calender of current month of year

   $ cal 5 2015

3. $ who

   hctm :0 2015-08-25 17:46

4. $ who am i

5. $ who -Hu → complete information of other users

Name	Line	Time	IDLE	PID	COMMENT
hctm	pts/1	2015-08-25 17:46		3456	(:00)

6. $ date "+ my birthday is %d %b %Y"

   my birthday is 25 Aug 2015

   Year → Y - full year 2015

   y - Abb year 2015

   Month → B - full month

   b → Abb month

   m → numeric month.

day of month - d
Hours - H
minute - M
Second - S
Time - T
Date - D

AM, PM → P → in capital PM
p → in small pm

n → newline
t → tab

7. $echo how are u
how are u
$

⇒ It display the message on the terminal screen.
⇒ It doesnot recognize the blank space
⇒ without an argument puts a blank line.

8. $ tty telepyte
/dev/pts/1 → Linux treats every thing as files, even the terminal connected are represented as files

9. $ tput → contsol the cursor on the screen
10. $tput clear
11. $ tput cup 10 5 → cup is an argument along with 10th row and 5th column.
12. $ tput smso → start bold sequence

11. $ tput rmso → Start normal sequence.

12. $ uname → details of linux System

13. $ uname -r → This option gives the version no. of the machine

14. $ uname -m → machine.

15. $ uname -a → all the details of the System

16. $ bc → It is for calculator and small language for writing numerical programs.

→ After using this command use Ctrl + D for exit

17. $ script → This command is used to record an interactive session

script started, file is typescript

exit
script done, file is typescript

→ We can see the recorded session by

$ more typescript.

18. $ man → format and mannual page for of command.

19. $ stty → display the setting of terminal

20. $ stty -a → print all current setting in human readable form

21. $ Ls → list directory content

22. $ ls [option]... [FILE]....

The option may contain →

-a all (do not ignore entries starting with .)

-A almost all (do not list . and ..)

-m comma separated list of entries

-r reverse order while sorting

-s by size

-t sort by modification time

23. $ cd dirname change directory

[rajender @ localhost ~] $ cd Desktop

[rajender @ localhost Desktop] $

24. $ pwd Present working directory

/home/rajender/Desktop

25. $ mkdir dirname

→ new directories are created by mkdir command

for eg:-

$ mkdir abhi ramesh dipu

$ mkdir a a/b → make a directory with a name and then subdirectory b under the parent directory a

$ mkdir a a/b a/b/c.

26. $ rmdir → remove diredory
directory must be empty.

27. $ rm Filename · remove file

28. $ mv oldfile newfilename
→ moving or renaming filename and directory

29. $ cat filename
→ To see the content of a file

$ cat helloworld.C

30. $ cat > filename ↵ To create a file
type characters —
=
Press Ctrl + D

31. $ cat >> filename
→ To add contents in already existing file

32. $ cat file1 file2 File3 >> file4

→ merge the contents of file1 file2 and File3 in File4 and also create File4 if it is not created yet.

⇒ Wildcard Symbols with ls command

a* Display all filename whose name starts with a

a?. Display file whosename start with a and those file 2 character long.

* sh files end with sh.

33. $ touch new. create empty files with touch command

34. $ cp oldfile newfile

∴ copy the contents of oldfile to newfile If newfile doesnot exist then it it first create destination file

View Long Files

→ PG, More, less Basic commands to see files

→ head and tail to see begining and end of file.

35. $ pg filename → display the contents of a file.

36. $ pg -10 → display max. 10 lines for each window.

37. $ pg +10 → start to display lines from line no. 10.

38. $ more filename.

Show in %

Press spacebar next window

Ctrl + D move forward one screen

Ctrl + B move backward one screen

39. $ less filename

first 24 lines. Then Enter key line by line.

b → backward

f → forward.

40. $ head Display first few lines of one or more files.

$ head file1 file2 file3

==> file1 <<==

abcd

==> file2 <<==

abc

==> file3 <==

abcd.

41. $ head see → It display first 10 lines of see filename

42. $ head -15 sec → display 15 lines

43. $ tail filename 1 2 ..

→ Display last few lines of one or more files.

44. $ tail see → last 10 lines

45. $ tail -15 sec → last 15 lines of the sec file.

46. $ wc filename

word count

33 65 472 filename

↑ ↑ ↑

Lines words characters

47. $ cmp file1 file2

→ to compare files we use cmp command. If the contents of two or more files are same. and we want to delete files that are identical then use cmp. command.

If files are differ then it will give msg like.

file1 file2 differ : byte 1, line 1

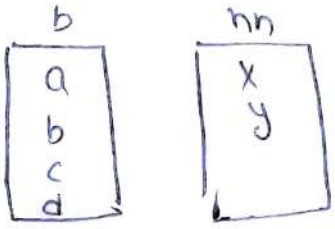
48. $ comm file1 file2

→ common data in two or more files.

49. $ Diff file1 file2 → Difference b/w files and it also tells you the changes you have to made to make two files identical

50. $ diff b nn ↵

```
<a
<b
<c
<d
---
>x
>y
```



51. $ diff nn b

```
<x
<y
---
>a
>b
>c
>d
```

52. $ expr 5 - + - 2

7

53. $ mkdir ravi{1..10} → It make 10 directories in one command with range specified.

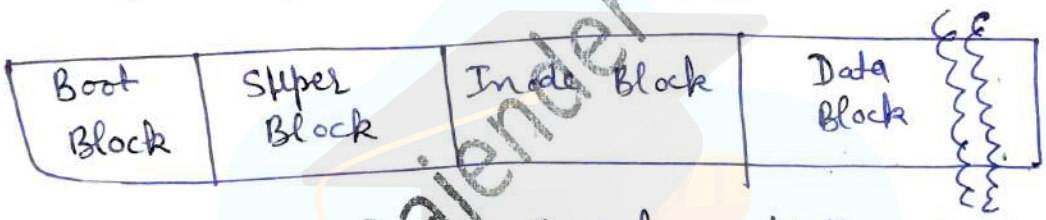
ravi1
ravi2
ravi3
ravi4
ravi5
ravi6
ravi7
ravi8
ravi9
ravi10 } 10 directories with single mkdir command

UNIT-1 ULP

SHELL PROGRAMMING

UNIX FILE SYSTEM → A Filesystem is the method and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organised on the disk. Most UNIX file systems types have a similar general structure.

The Physical organisation of a file system is shown as:



Physical organisation of file systems

① **Boot Block →** The Boot Block contains a small program called the Bootstrap loader. This program is extracted at boot time by BIOS and it is used to start the OS. Every file system has its own boot block. but only one block is put to use when all file systems attached together.

② **Super Block →** The Super Block contains a description of the basic shape and size of the file system. This information is use by

System manager to maintain the file system. The Kernel first read the Super block before Inode and Data block.

It contain following information:

* Magic N Crusher: This allows the mounting s/w to check that this is indeed the Superblock for an EX 72 File System. Current version of EX72 is OXEF 53.
* Revision level → It contains new features Compatibility fields that helps the mounting code to determine which new features used on this file system.
* Mount Count and Maximum Mount Count: These allow the system to determine if the file system should be fully checked. The mount count is incremented each time the file system is mounted and when it equals the maximum mount count, the message of warning is displayed.
* Block Group Number → It holds the copy of Super Block.
* Block Size → The size of the block for this file system in bytes. For example 1024 bytes.

* Block Per Group → The number of blocks in a group.

* Free Block → The number of free blocks in the file system.

* Free Inodes → The number of free Inode in the filesystem.

③ INODE Block → This block contains the inode entry for each file of the file system. All attributes of a file or directory are stored its inode these. The total number of files depends upon the total number of inode.

A new file is created in a file system if and only if a file node is available in the system as well as free data block.

④ Data Block → The Data block follow the inode block. These block need not to be continous. These block contain data stored in either in bytes or group of bytes are called blocks. Holding data byte by byte has a disadvantage of moving disk for every byte. So data is handled in blocks. The size may be either 512 or 1024 bytes.

File system types →

Linux supports several types of filesystems. Important ones are:

Minix → The oldest and to be most reliable, but limited in features (30 characters file names) and 64 MB per file system.

Xia → A modified version of Minix. Add new features.

ext3 → All features of ext2 and recovery time in case of system crash.

ext2 → New version of file system code do not require making the existing filesystems.

ext → older version of ext2.

msdos → Compatibility with MS-DOS (OS/2 and windows NT)

umsdos → Extends the msdos filesystem driver under linux

nfs → A networked filesystem that allows sharing a filesystem between many computers to allow easy access to the files from all of them.

② Unix/Linux Files →

A File is a basic entity used to store and maintain the information on any operating system. It is a stream of bytes. A file can be a simple text file, an image file, a program, a sound file, a video file and so on.

Each file in the file system is given a unique name, which is used to identify the file among thousands of files.

Each filename has two parts separated by a period (.). The part of the filename before period is basename of the file and the part after the period is the extension of the file.

2.1 File Types →

The file is the basic entity in the Linux system. Linux supports mainly four different types of files.

① ordinary files ② Directories
③ link files ④ Special files

ULP – UNIT 1

① **Ordinary Files** → An ordinary file is the most common file type. All programs that we write belong to this type. It is generally devided into two categories namely text file and binary file.

Text File → A Text file contains lines of characters where each line is terminated by a new line character. A Text file occurs only half ASCII range (0 to 127). All C and Java, shell scripts are text files.

Binary File → A Binary file contains both printable and non printable character that range (0 to 255) of ASCII value. For example, object code and executable which are produced by compiling C programs are binary files.
A Unix command is also an example of a Binary file.

② **Directory** → Linux allows you to group files into directories. A directory is a collection of a group of files. It is

always desirable to keep similar type of files in one directory. For example all audio files may be kept in one directory and office related files could be kept in other directory.

In other words a directory allows you to Categorize the information stored on your computer in the form of files.

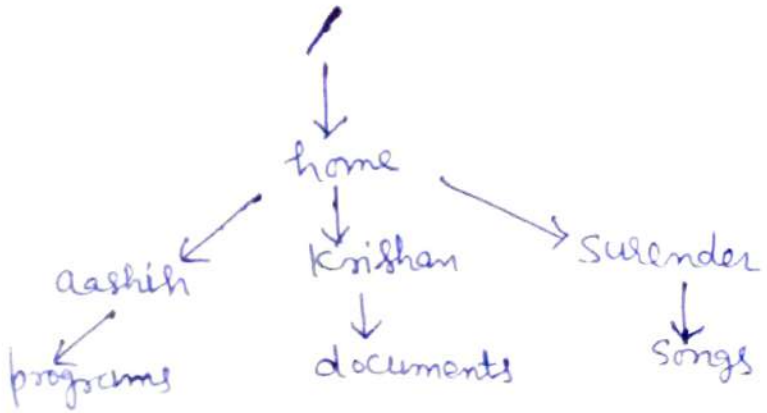
③ Links → A Link is basically not a file but it is just second name for a file. Suppose if two users want to share the information in a file. They can have separate copies of the same file.

$ ln chapter1 /home/ravi/abc/chap1

Now we can access the chapter1 from the directory /home/ravi/abc/chap1 with the name chap1. If you change some text in chapter1 then those changes can be seen in chap1 also.

④ **Special Files →** Linux also has some special files known as device files. Each file represents a device. It may be a terminal, a printer, a cd-rom and other peripherals connected to the computer. These files are treated in much the same way as the ordinary files. These files can be created and changed. It means read and write operation can be performed on the device files as on the ordinary files.

2.2 Hierarchical File Structure → We know that directories can contain other directories which in turn can contain other directories. Linux file system is called as Hierarchical file structure.



2.3 Difference between Linux and Windows

File Structure →

① In Linux root directory of the file system is represented by / (slash). In windows root is always represented by \ (back slash).

/home/ishan — Linux

c:\ishan — windows

② If you want to use options with command in Linux, then it is specified by - minus sign sign whereas \ is used in windows.

$ ls -l] linux

c:\dir \p] windows

③ In windows a file can have an extension of two to three characters whereas no limit on the no. of characters in Linux.

④ In Linux one file can have more than one extensions like book.tar.gz whereas in Windows one file can have only one extension. like book.doc.

INODE AND ITS STRUCTURE

- Inode stands for index node. As soon as the file is created, the kernel allocates a unique inode number to that file.
- An inode number is a positive number whose maximum value depends on the total inode numbers of the file system.
- An inode describes which blocks of data a file occupies as well as the access rights of the file, file modification times and type of the file.

The Inode structure is & known below :

1) **File type :** This field shows the type associated with each file. If this field contains a hyphen "–" Here it is an ordinary file, if it contains a letter "d" here it is a directory file.

2) **Link Information:** This field shows the number of links associated with file.

3) **UID of the owner:** This field displays the UID or user identification number of the owner of the file.

4) **GID of the owner:** The owner of a file also belongs to some group. However the group does not have same permissions as that after owner who created the file.

5) **Size in bytes:** This field shows the size of the file in bytes that is the amount of data it contains.

6) **Access time:** This field shows the time when the file was last accessed for reading the contents. The contents are not modified in any way.

7) **Modification time:** This field contains the last modification time of the file. A file is said to be modified if its contents have changed in any way. If only the permissions or ownership of the file is changed then the modification time of the files remains unchanged.

8) Time the inode was last changed: This field contains the time when any of the fields of inode change such as access time, UID and many more.

9) Generation Number: This field contains the number which is generated when an inode is created.

10) Address of data blocks: This field contains the pointers to the blocks that contains the data the inode is describing. Also, the inode value of a file can be known by using "ls" command with "-i" option as shown below:

```
$ ls -i ishan.txt
106 ishan.txt
$
```


File type
Link Information
UID of the owner
GID of the owner
size is bytes
Access time
modification time
Time the inode was last changed
Generation number
Address of first 10 blocks and than of 3 indirect blocks

Shell as command processor

Whenever we log on to a Unix machine, we see a prompt. This prompt can be either $ or #. Infact a program is running everytime we make a login to the system. This program is called as shell.

- Whenever user issue a command, the shell acquires that information, parses and rebuilds the command line and finally leaves the execution work to the kernel.
- The shell is generally is one of the three states namely waiting, sleeping and waking. The shell keeps sleeping for most of the time.
- Whenever a user issues a command, the shell wakes up and perform the following activities:

(1) It parse the command line and identifies each and every word and removes additional spaces if present.

(2) Looks for special variables in the command line like $$ which generally begin with a $ symbol

(3) It then creates a simplified command line and passes it for execution. The shell then gives to the waiting state until the command is executed and waits for its completion.

(4) After the command is executed successfully, the shell prompt reappears and shell returns to the waiting state waiting for the user to enter new commands.

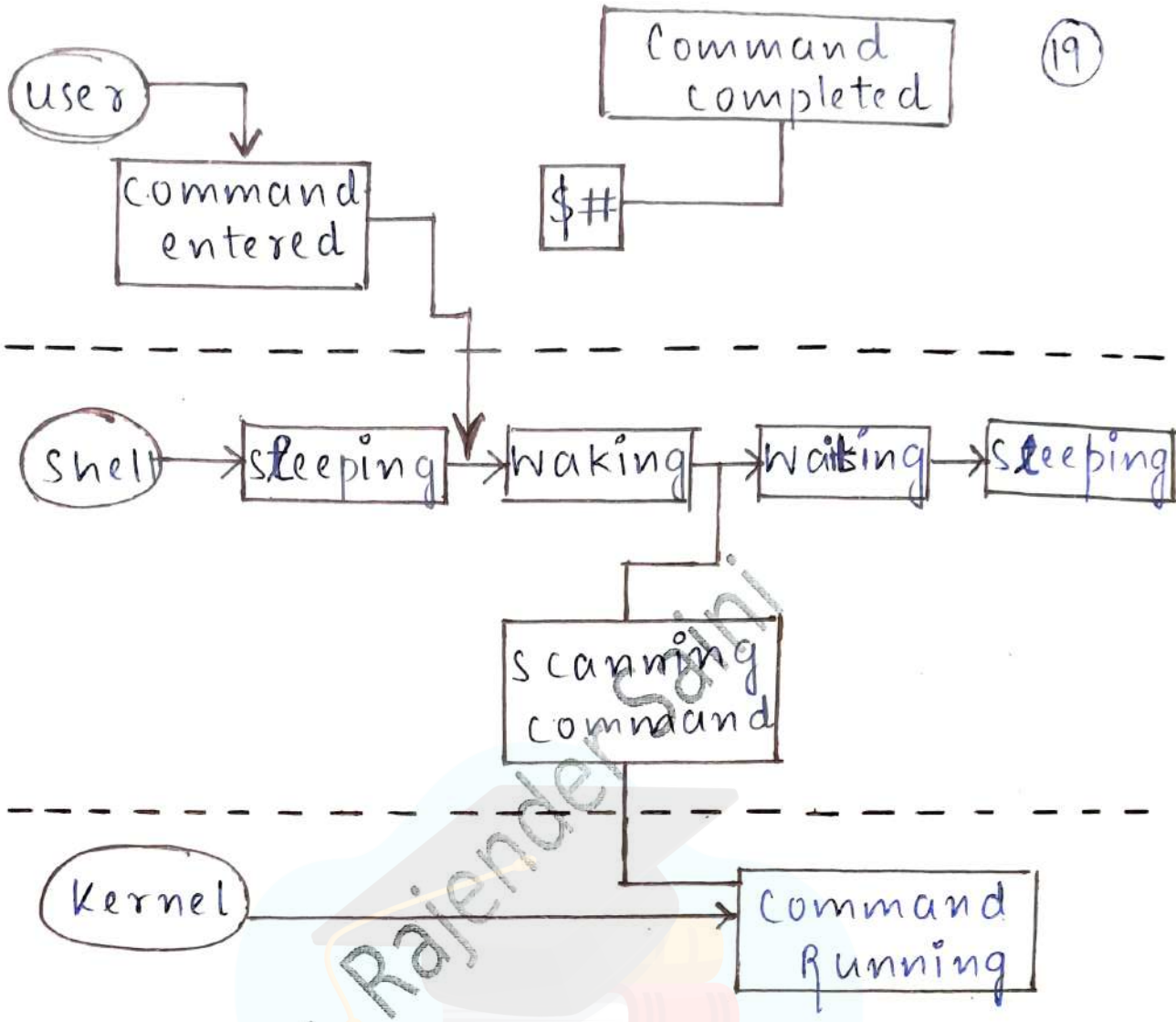


Fig → shell as a command processor

Shell variables and customizing shell Envit.

Shell like any other high level language provides you the facility of variables that can be used to hold the piece of information. Basically a variable is the name given to a memory location where the information is stored.

There are 2 types of variables in Linux:

- System Variables
- User defined Variables

(A) System Variables

- These variables are used by Linux itself. You can use these variables for own purpose and change the information stored in them.
- System variables are also known as Environmental variables because they set the compnting environment.
- System variables are always defined in capital letters. To check the values of system variables, just type the set command on shell prompt like :⟶

```
$ set
CDPATH =: /home/sue: /home/sue/db: /home/sue/ progs
HOME = /home/sue
LOGNAME = sue
MAIL = /var/mail/sue
MAILCHECK = 30
MAIL PATH = /var/mail/sue : /home/sue/rje
MAILRC = /home/sue/mail/.mailrc
MBOX = /home/sue/mail/mbox
PATH = /usr/bin: /usr/ibin : /home/sue/bin
PS1 =:
PS2 = ...
SHELL = /usr/bin/sh
TERM = AT386-M
TZ = EST5EDT
```

(B) User defined variables

- The user in the unix environment also has the right of creating shell variables on command line.
- Variables are case-sensitive and capitalized by default. It is however advisable to use a lower case name for user defined variables.
- However, we are free to use the names we want. Variables can also contain digits but a name starting with a digit is not allowed.

To set a variable in a shell, we use :→

VARNAME = "value"

Putting spaces around the equal to sign will cause errors. It is however a good practice to quote content string when assigning values to variables. Consider the example shown :→

1. $ MYVAR = "2"
   $ echo $ MYVAR
   2

2. $ first_name = "ishan"
   $ echo $ first_name
   ishan

Syntax of user defined variable is :

variable name = value

'value' is assigned to given 'variable name'

and value must be on right side of = sign.

Examples:

```
$ no = 10    # This is ok

$ 10 = no    # Error, Not ok, value must
               be on right side of = sign.
```

To defined variable 'abhi' having value 'Bus'.

```
$ abhi = Bus
```

To defined valiable called n having value 10.

```
$ n = 10
```

Types of Shells → when you login to the Linux system, you are entering into the shell. Originally UNIX system shell sh was written by Steven Bourne and it was known as Bourne shell. other shells like C shell (csh), advanced C shell (tcsh), Korn shell (ksh) and Bourne again shell (bash).

The default shell of Linux is bash.

bash :– The bash also known as Bourne Again Shell is an advanced version of the Bourne shell which was used in UNIX systems. It is most command and freeware shell. Because bash uses all the features of the bourne shell that was designed earlier therefore every shell program written in Bourne Shell will also run under the bash shell.

tcsh → It is the advanced version of original C shell which was developed by Bill Joy. It is named as C shell because most of the part of this shell is designed in C programming language. It has features like history list,

and job control were also added. The tcsh is a superset of csh. It means all the commands of csh will also run under tcsh.

pdksh → The pdksh (Public domain Korn Shell) simply known as Ksh (Korn Shell) was developed in 1982 by David Korn. It is based on original Unix shell sh. It adds many features of C shell. It also adds its own features to make the shell more easy to use. The one important point to be noted that shell programs written in bash will run under Korn shell without making any changes to them.

How many shells installed on your system?

```
$ cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/bin/tcsh
/bin/csh
/bin/zsh
$
```

What is your Shell ?

```
$ echo $SHELL
/bin/bash
$
```

This command will show you the current shell of linux system.

How to change login shell ?

```
$ chsh
changing shell for root.
New shell [/bin/bash]: /bin/tcsh
Shell changed
$
```

we have also used other command for changing shell as

```
$ chsh -s /bin/tcsh
changing shell for root.
Shell changed
$
```

-s → set shell as the login shell

Shell scripts →

A Program written in shell programming language or shell is known as shell script or simply a shell Program. A shell script is defined as a sequence of commands written in an ordinary file. It may also includes conditional statements and loop control structures.

How to write and Execute shell script:

with the help of Cat command or VI editor we can make a shell program as.

```
$ cat > raj
who
pwd
ctrl +d
$
```

This script has two commands who and pwd.

Execution of Script:

```
$ ./raj
Permission denied.
$
```

Permission of execution of this file is denied beca

when a new file is created, it has permission as

-rw- r-- r--

with `$ ls -l raj` command we can check these permission. It means user or owner has not execution permission of shell program. So we need to change the permission as

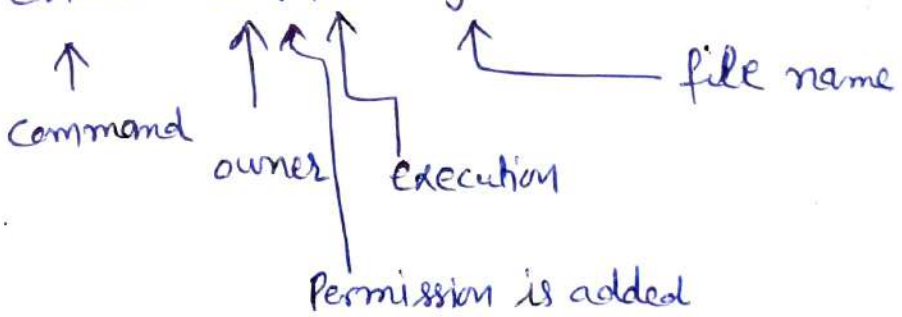
-rwx r-- r-- with chmod command

by using `744` or `u+x`

```
$ chmod 744 raj
```

or

```
chmod u+x raj
```




```
$
```

After this command file raj has file permission as

`-rwx r-- r--`

To execute the raj script. we have

```
$ ./raj
root    tty1    2016-09-07  20:15
root    tty2    2016-09-07  20:16

/root/Desktop

$
```

```
- rwx  rwx  rwx
  |    |    |
  user group other
  or
  owner
```

\+ Permission given

\- Permission not given

rw- → 110

r-- → 100

rwx → 111

Command Substitution → In general, we have seen how a Pipe can be used to connect commands by obtaining its standard input from the standard output of other command.

```
$ who
root    tty1    2016-09-07 20:15
root    tty2    2016-09-07 20:16

$ grep tty1
— —
$
```

After Applying Pipe (|) command with who and grep command, we get the output as follows.

```
$ who | grep tty1
root tty1   2016-09-07  20:15
$
```

The Shell also enables one or more command arguments to be obtained from the standard output of another command. This is called as Command Substitution.

UNIT 2 - ULP

Regular Expression and Filters →

In Unix/Linux, we often need to search one or more records from a database or one or more lines from a text file. For example, such a search can include searching or finding a:

① line having a certain word in a file.
② record based on data like name or roll no.
③ line that contain words in a specified pattern like aaa, bbb and so on.

Patterns: → Patterns are nothing but simply a string of characters. These string of characters that represent certain patterns and used for searching words/lines are called as Regular Expression.

With Regular Expressions we can define the patterns we are looking for and let the regular expression engine do the search for us. Some filters also employ the use of regular expression

like grep, egrep, sed and fgrep.

A **filter** is a program that reads its input from the standard input, processes it and write its output back to the standard output.

Regular Expressions are made up of a number of things. These include:

→ Normal characters that match exactly the same character in the input.

→ character classes that match exactly single character in the class.

→ Special characters include * (asterisk), ^ (caret), $ (dollar) and many more.

In writing Regular expressions, it is necessary to see what type of patterns are present in our data.

```
$ ls -l
drwx--wx--x  3 root root 1024 Feb 10.8.10
-rw---w---x  1 ishan ishan 864 June 11.3:40
```

Unit 2- ULP

Character Classes and Bracket Expressions →

often, there arise situations where we need to match a character from a given set of characters.

In Unix/Linux, from characters out of which only a single character is matched is called a character class. with character classes, we can specify characters individually or give a range of allowable characters like [a-d].

Also we negate a character class indicating which characters are not acceptable instead of which are acceptable.

Character classes are presented within a pair of square bracket like [range of characters].

For example, if a user wants to select only those lines that contain the word "raj" followed by any digit 1, 2, or 3 then the search pattern will be raj[1 2 3]. This pattern can also be written as:

eg: [1-3] where "-" specify range of characters in the set. So, [1-3] means any of the characters [1, 2, 3].

A Bracket expression is a list of characters enclosed by '[' and ']'. It matches any single character in the list; if the first character of the list is '^', then it matches any character not in the list.

Within a Bracket Expression, a range expression consists of two characters separated by a hyphen(-).

Some of character classes in Unix include:

① [0-9] → matches any of digits between 0 to 9

② [a-z] → matches any lower case alphabet

③ [A-Z] → matches any upper case alphabet

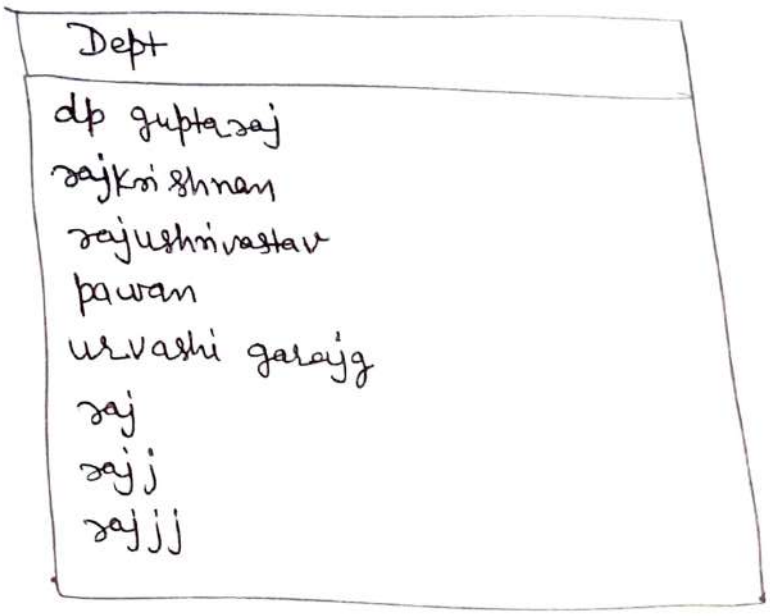
④ [A-Z a-z 0-9] → This characters class combines all the above characters classes and matches any alphanumeric character.

To negate a character class '^' caret symbol is used.

Quantifiers → whenever we looking for data in Regular expressions, we often need to specify how many times characters appear. For example, we might want the pattern matches to match those lines in files which contain one or more occurrences of the word "raj". Quantifiers provide a way to do this. They work by specifying how many times the preceding character is supposed to be matched. Some of the most frequently used quantifiers are:

① **The Asterisk character (*)** → It is used to match multiple characters. It stands for zero or more occurrences of preceding character. For example, the expression "a*" matches ^(null), a, aa, aaa... and so on. Here null stands for zero occurrences of character "a".

② **Dot character (.)** → The (.) Dot is used to match a single character except a new line character. (.) Dot is used in regular expression whereas in case of shell operator "?" is used to match a single character.



$ grep " raj* " dept

dp guptaraj
raj krishanan
rajushrivastav
urvashi garajg
raj
rajj
rajjj

$ grep " raj. " dept

dp guptaraj
raj krishanan
rajushrivastav
urvashi garajg
rajj
rajjj

③ Plus character (+) → The + character is similar to (*) except that (+) character matches one or more occurrences of preceding character.

④ Caret character (^) → This quantifier is used to search those words that start with a specified pattern. And Also it is used for negate the symbol.

Example1: "^a" matches "a" at the start of line
Example2: "[^0-9]" matches any non digit.

```
$ grep "^raj" dept
rajKrishnan
rajushrivastav
raj
rajj
rajjj
$
```

output

⑤ $ Dollar character → It match or search those words that end with specified pattern.

```
$ grep "raj$" dept
raj
$
```

output

⑥ Braces { } → Curly bracket are used as range.

Example: "a {2,3}" matches "aa" or "aaa".

⑦ $ ls -l | grep "^d"

output:

drwxrwxr-x. 2 rajender rajender 4096 Sep 27 15:14 aar

This command display the all directories under the current working directory.

⑧ $ ls -l | grep "^-rwx"

```
-rwxr--r--. 1 rajender ravi.sh
-rwxrw-r--. 1 rajender largest
-rwx------. 1 rajender ravi
-rwx------. 1 rajender kabhi.c
```

This command display all the files which have rwx file permission for user..

Introduction to Grep →

Grep stands for Global regular expression print. Grep is a filter which is capable for receiving its input from standard input device and sending its output to standard output. The family of grep consist of 03 commands.

① grep
② egrep (extended grep)
③ fgrep (fixed grep)

① Grep → This command is used to search or extract lines or records from an input file. Grep is global regular expression print. Here, global means that the entire file is searched for specified pattern or patterns.

The syntax of grep command is;

$ grep [options] pattern filename1 filename2

① Searching Single word → Suppose we want to search raj from the file "dept".

The command would be;

```
$ grep raj dept
rajgupta
Urvashi gargraj
raj
rajj
$
```

② Searching multiword string → For searching multiword strings, it is essential to quote the pattern to be searched. If we do not do this then only first word is treated as pattern and rest is file names.

For example we want to search "Urvashi gargraj" from the file dept. Then command is

```
$ grep urvashi gargraj dept
grep: gargraj : No such file or directory
dept: urvashi gargraj
$
```

(3) Grep options →

(3.1) Inverse option (-v) → It will prints those lines that do not contains specified patterns.

```
$ grep -v "dp" teachers
```

output:

```
Rajender
Pawan
Mukesh
Anil
```

(3.2) Ignore option (-i) → Grep distinguish between uppercase and lowercase. With -i option it will prints all patterns without considering the case.

```
$ grep -i "DP" teachers
```

output:

```
dp gupta
```

(3.3) Filename option (-l) → It print the filename in which specified pattern is present,

$ grep -l "dp" student teachers for.c
file1 file2 file3

output:

teachers

(3.4) Count occurrances (-c) →

The -c option with grep command will count the occurrances of records.

$ grep -c "dp" teachers

output:

01

(3.5) Line number option (-n) → This option will print out the line number with the record that contain the specified pattern.

$ grep -n "for" largest.c for
file 1 file 2

output:

largest.c : 11 : Press 1 for addition
largest.c : 12 : Press 2 for substraction
for : 1 : for a in 1 2 3 4 5

(3.6) Multiple pattern search option (-e) →

This pattern option allow us to search multiple patterns within a single command.

$ grep -e "dp" -e "rk" teachers

output:

dp gupta
rk bakshi

Q: Display the file in current directory that contain the string "If" is either in uppercase or in lowercase letter.

```
$ ls -l | grep -i "IF"
```

```
-rwx rw- rwx . 1 rajender rajender sep 28 21:31 if.pl
-rwx --- r-- . 1 rajender rajender sep 29 20:30 If.c
-rwx --x rw- . 1 rajender rajender sep 11 11:29 IF.txt
```

Introduction to EGREP → Egrep stands for Extended grep because it include two additional quantifiers "?" and "+". This command is the most powerful command of grep family. It is also includes search for multiple patterns. For searching multiple pattern, the pipe ~~simp~~ "|" character is used.


```
$ grep -e "raj" -e "paw" dept
```

"raj" → search pattern1; "paw" → search pattern2; dept → filename

OR

```
$ egrep "(raj|paw)" dept
```

"raj" → search pattern1; "paw" → search pattern2; dept → filename

Output:

```
rajgupta
rajkrishnan
pawan
urvashi
raj
rajj
rajjj
```

If the portion of patterns at either end is common.

```
$ egrep "(dip|raj) gupta" dept
dipgupta
rajgupta
$
```

```
$ egrep "raj(Kishor|Kamal)" dept
rajkishor
rajKamal
$
```

Search for lines which contains exactly two consecutive rr characters.

```
$ egrep "r{2} *" dept
   or
$ grep "rr *" dept
   OR
$ grep -E "r{2} *" dept
```

let us get an output of egrep command with all lines which end with wood.

It contain word like lockwood and Harwood.

$ egrep (lock|har) wood file.txt

Here wood is common to both words only the prefix vary which are matched using "|" pipe symbol.

Fgrep → Fgrep searches for fixed character strings in a file or files

Fgrep is useful when you need to search for strings which contain lots of regular expression metacharacters, such as "$", "^", etc.

If your string contains newlines, each line will be considered an individual fixed-character string to be matched in the search.

Running fgrep is same as running grep with the -F option.

"$" and "^" has special meaning in grep. But these are considered as word or fixed character in fgrep command.

fgrep is same as grep -F .

$ grep '^' dept.txt

pawan
^kabhi
^tum
urvashi
rajender
raj$
$
$$

$ fgrep '^' dept.txt

^kabhi
^tum

$ grep -F '^' dept.txt

^kabhi
^tum

$ grep '$' dept.txt

pawan
^kabhi
^tum
urvashi
rajender
raj$
$
$$

```
$ fgrep '$' dept.txt
raj$
$
$$
```
output

```
$ grep -F '$' dept.txt
raj$
$
$$
```
output

```
$ fgrep '$$' dept.txt
$$
```
output

```
$
```

SED Command in UNIX/LINUX → Sed is a stream editor used for modifying the files in Unix (& Linux). Whenever you want to make changes to the file automatically, sed is helpful in that. Apart from replacing text, you can do many many things with sed.

Features of SED with example:-

Consider the below text file as an input.

```
> cat file.txt
unix is great os. unix is opensource. Unix is free os.
learn operating system.
unixlinux which one you choose.
```

① Replacing or substituting string → It is mostly used to replace the text in a file.

```
$ sed 's/unix/linux/' file.txt
```

```
linux is great os. unix is opensource. unix is free os.
learn operating system.
linuxlinux which one you choose.
```

Here 's' specify substitution operation.

By default sed command replaces the first occurrence of pattern of each line and it would not replace the second, third ..... occurrence in the line.

② Replacing the nth occurrence of a pattern in a line →

```
$ sed 's/unix/linux/2' file.txt
```

```
unix is great os. linux is opensource. unix is free os.
learn operating system.
unixlinux which one you choose.
```

③ Replacing all occurrence of a pattern in a line →

```
$ sed 's/unix/linux/g' file.txt
```

```
linux is great os. linux is opensource. linux is free os.
learn operating system.
linuxlinux which one you choose.
```


④ Replacing from nth occurrence to all occurrences in a line →

```
$ sed 's/unix/linux/3g' file.txt
```

```
unix is great os. unix is opensource. linux is free os.
learn operating system.
unixlinux which one you choose.
```

(5) Using '&' as the matched string →

```
$ sed 's/unix/{&}/' file.txt
```

{unix} is great os. unix is opensource. unix is free os.
learn operating system.
{unix} linux which one you choose.

```
$ sed 's/unix/{&&}/' file.txt
```

{unixunix} is great os. unix is opensource. unix is free os.
learn operating system.
{unixunix} linux which one you choose.

(6) Switch words →

If you want to switch the words "unixlinux" as "linuxunix" the sed command is:

```
$ sed 's/\(unix\)\(linux\)/\2\1/' file.txt
```

unix is great os. unix is opensource. unix is free os.
learn operating system.
linuxunix which one you choose.

(7) Switching first 03 characters in a line:

```
$ sed 's/^\(.\)\(.\)\(.\)/\3\2\1/' file.txt
```

inux is great os. unix is opensource. unix is free os.
aelon operating system.
linuxlinux which one you choose.

(8) Duplicating the replaced line →

Each line is displayed 02 times with /p tag.

```
$ sed 's/unix/linux/p' file.txt
```

```
linux is great os. unix is opensource. unix is free os.
linux is great os. unix is opensource. unix is free os.
learn operating system.
linuxlinux which one you choose.
linuxlinux which one you choose.
```

(9) Printing only replaced line →

```
$ sed -n 's/unix/linux/p' file.txt
```

```
linux is great os. unix is opensource. unix is free os.
linuxlinux which one you choose.
```

(10) Running multiple sed commands →

```
$ sed 's/unix/linux/' file.txt | sed 's/os/system/'
```

```
linux is great system. unix is opensource. unix is free os.
learn operating system.
linuxlinux which one you chosystemse.
```

(11) Multiple sed commands with -e option →

```
$ sed -e 's/unix/linux/' -e 's/os/system/' file.txt
```

```
linux is great system. unix is opensource. unix is free os.
learn operating system.
linuxlinux which one you chosysteme.
```

(12) Replacing string on a specific line no.

```
$ sed '3 s/unix/linux/' file.txt
```

```
unix is great os. unix is opensource. unix is free os.
learn operating system.
linux linux which one you choose.
```

Replace with range of 1 to 3 line

```
$ sed '1,3 s/unix/linux/' file.txt
```

```
linux is great os. unix is opensource. unix is free os.
learn operating system.
linuxlinux which one you choose.
```

(13) Delete line from file →

```
$ sed '2 d' file.txt
```

```
unix is great os. unix is opensource. unix is free os.
unixlinux which one you choose.
```

```
$ sed '5, $ d' file.txt
```

(14) Duplicating lines →

```
$ sed 'p' file.txt
```

```
unix is great os. unix is opensource. unix is free os.
unix is great os. unix is opensource. unix is free os.
learn operating system.
learn operating system.
unixlinux which one you choose.
unixlinux which one you choose.
```

(15) Sed as grep command →

```
$ sed -n '/unix/p' file.txt
```

```
unix is great os. unix is opensource. unix is free os.
unixlinux which one you choose.
```

OR

```
$ grep 'unix' file.txt
```

```
unix is great os. unix is opensource. unix is free os.
unixlinux which one you choose.
```

(16) Add a line after a Match →

```
$ sed '/unix/ a "Add a new line" ' file.txt
```

```
unix is great os.
"Add a new line"
```

(17) Add a line before a Match →

```
$ sed '/unix/ i "Add a new line" ' file.txt
```

```
"Add a new line"
unix is great os.
```

(18) change a line with "c" option →

```
$ sed '/unix/ c "change line" ' file.txt
```

```
"change line"
learn operating system.
```

→ "change line"

Programming with AWK → AWK is a filter Program that was originally developed in 1977 as a pattern scanning language. AWK was named after its developers

A → Aho
W → weinberger
K → Kernighan.

It is a Programming language with Block control structures, functions and variables. Like sed, awk combines features of several filters and was the most powerful utility for text manipulation till the development of perl. It also accepts extended regular expressions for pattern matching.

① Syntax of an AWK statement →

$ awk options "pattern {action}" filename/s

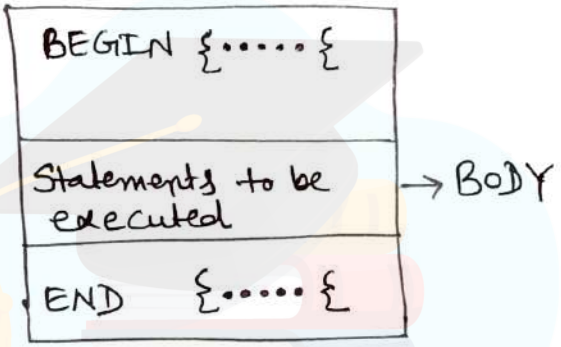
where

1. use of option is optional.
2. filenames will have zero or more files.
3. pattern specify the basis for line or record selection and manipulation.
4. Action statement is surrounded by curly brackets { }.

AWK employs only 02 options namely -F and -f.
The -F options specifies the input field separator & -f option specifies that the program operates on a separate file.

(2) Structure of an AWK script →

An AWK script is generally made up of 03 sections namely: BEGIN, BODY & END sections.



BEGIN Section:→ The BEGIN section contains statements that need to be executed before the awk actully starts executing the program statements from the body. The statements in this section are used for initializing variables, setting field separator which must be executed before the actual program execution starts. For example, BEGIN section contains

BEGIN { FS = "|" }

FS means field separator and it is initialized to Pipe symbol.

BODY section: The BODY section contains the actual program. The Program contains either a single statement or set of statements which need to be processed. These program statements are applied sequentially to each record or line to match the pattern portion of program.

END section → The END section contains statements that need to be executed after the last statement in the body section has been processed. The END section displays statement which contains the values of variables, total no. of lines processed and many more. For example:

```
END { print NR}
```

where NR stands for number of records processed.

Variables →

1. user defined
2. Built-in
3. FS → Field Separator
4. NF → Number of Field
5. NR → Number of Records
6. OFS → output field separator

Variable Name	value
$0	Entire line
$1	First Column
$2	Second Column
⋮	
$n	nth column

operator →

① Arithmetic → +, −, X, /, %

② Logical → ||
&&
!

③ Relational → >
>=
<
<=
==
!=

④ Assignment operator
- ~ Match
- !~ No Match

AWK Control Structures →

```
If (condition)
 {
    statements 1
 }
 else
     {
      statements 2
      }
```

while loop →

```
while (condition)
 {
  statements:
 }
```

```
do
   {
     statements :
   }
while (condition)
```

For loop →

```
For ( initialization ; Condition ; expression)
  {
    statement ;
  }
```

Break and Continue →

Break is used to terminate a loop. and executes all statements after the loop body.

employee.txt

S.No	name	designation	department	salary
1	a	AP	CSE	$20000
2	b	AP	CSE	$25000
3	c	AP	MECH	$26000
4	d	AP	MECH	$15000
5	e	AP	CIVIL	$11000

① Show complete File:

```
$ awk '{print}' employee.txt
```

This command will show the complete records of employee.txt

② Show specific column of File:

```
$ awk '{print $2, $5}' employee.txt
```

name	salary
a	$20000
b	$25000
c	$26000
d	$15000
e	$11000

③ Use of Inbuilt function (NF) →

NF → Total no. of field

$ awk '{print $2, $NF}' employee.txt

name	salary
a	$20000
b	$25000
c	$26000
d	$15000
e	$11000

④ Specific line records :→

$ awk '$1 >3' employee.txt

S.no	name	designation	department	salary
4	d	AP	MECH	$ 15000
5	e	AP	CIVIL	$ 11000

⑤ Print complete line with column match word →

$ awk '$4~/CIVIL/' employee.txt

[5 e AP CIVIL $11000]

This command will print the complete line if there is a match of column 4 = CIVIL, otherwise we will get no records.

(6) Print no. of employee from CSE →

```
$ awk ' BEGIN { count = 0;}
> $4 ~ /CSE/ { count++;}
> END { print " Number of employee from CSE
> are = " count;} employee.txt
```

output

Number of employee from CSE are = 2

(7) Use of tab and Newline Character →

```
$ awk 'BEGIN { print "Name\tdesignation";}
> { print $2"\t", $3}
> END { print " Report generated \n-----";}'
> employee.txt
```

Name	designation
a	AP
b	AP
c	AP
d	AP
e	AP

Report generated

---

output

(8) Use of NR inbuilt function →

NR → Number of records or count lines in a file

```
$ awk '{print NR}' employee.txt
1
2
3
4
5
6
```

```
$ awk 'END {print NR}' employee.txt
6
```

] display only total lines in a file.

(9) use of OFS Inbuilt variable →

OFS → output field separator

```
$ awk 'BEGIN {OFS="::"} {print $1,$5}'
> employee.txt
```

```
S.no :: Salary
1 :: $20000
2 :: $25000
3 :: $26000
4 :: $15000
5 :: $11000
```

output

(10) Program "square of a number" →

```
$ awk 'BEGIN {
> for (i=1; i<=5; i++)
> print "square of" i, "is", i*i }'
```

> Output

```
square of 1 is 1
square of 2 is 4
square of 3 is 9
square of 4 is 16
square of 5 is 25
```

(11) use of Pipe | symbol :→

```
$ awk '$4 ~ /CIVIL|MECH/' employee.txt
```

```
3  c  AP  MECH  $26000
4  d  AP  MECH  $15000
5  e  AP  CIVIL $11000
```

(8) Use of NR inbuilt function →

NR → Number of records or count lines in a file

```
$ awk '{print NR}' employee.txt
1
2
3
4
5
6
```

```
$ awk 'END {print NR}' employee.txt
6
```

] display only total lines in a file.

(9) use of OFS Inbuilt variable →

OFS → output field separator

```
$ awk 'BEGIN {OFS="::"} {print $1,$5}'
> employee.txt
```

```
S.no :: Salary
1 :: $20000
2 :: $25000
3 :: $26000
4 :: $15000
5 :: $11000.
```

output

(10) Program "square of a number" →

```
$ awk 'BEGIN {
> for (i=1; i<=5; i++)
> print "square of" i, "is", i*i }'
```

> Output

```
square of 1 is 1
square of 2 is 4
square of 3 is 9
square of 4 is 16
square of 5 is 25
```


(11) use of Pipe | symbol :→

```
$ awk '$4 ~ /CIVIL|MECH/' employee.txt
```

```
3  c  AP  MECH  $26000
4  d  AP  MECH  $15000
5  e  AP  CIVIL $11000
```

Q1. what is the difference between print and print $0 ? Is print statement necessary for printing a line?

Ans. There is no difference between print and print $0 both print entire line. No print statement is not necessary if the selection criteria or pattern is specified.

$ awk 'NR <= 5' file.txt

↳ Select Five lines of file.txt

$ awk 'NR == 5, NR == 10' file.txt

↳ Select lines 5 to 10 of file.txt

$ awk 'NR >= 20' file.txt

↳ Select all lines after 20th of file.txt

Programming with Perl

Perl stands for practical extraction and report language. It was developed in 1986 by Larry Wall as an interpreter on Unix. Initially perl had an emphasis on system management & text handling but with subsequent revisions perl also became capable of handling regular expressions, signals & network sockets too. So its programs bear a similar resemblance to that of C programs.

• First Perl Program :-

A perl program consist of an ordinary text file containing a series of perl statements. Statements are generally written in a combination of C, unix shell scripts & english. Perl code is quite free following. The syntactic rules governing where statement starts & ends are :-

1. Leading spaces on a line are ignored. We can start a perl statement anywhere we want.

2. Statements are terminated with a semi-colon
3. Spaces, tools & blank lines outside of strings are irrelevant. We can split statements over several lines for clarity.
4. Anything after hash sign (#) is ignored except in strings e.g:-

```
print ("Hello world \n");
```

Perl runs above code & print Hello world.
\n means perl goes to next line

for eg:-

```
print ("My name is Raj \n");
print ("I am teaching Linux \n");
```

To write a complete program, we need to add the invocation line at the top & a few comments

```
# !. /user / local /bin / perl.
print ("Hello world " \n");
```

Numeric and String Literals :-

Literal is also known as constant data. following data types are :-

→ Numeric Data: Most basic datatype

→ String :- Series of characters that are handled as one unit.

→ Array :- It is series of numbers & strings handled as a unit. Array can be taken as a list.

→ Association array :- It is special array called hash, a list in which array value has an assoated look up term.

Variables :-

Perl like other languages, uses variables to keep track of usage of computer memory. We need to store a new piece of information, we assign it to a variable perl uses three types of variables.

Variable Type	Description
• Scalars :-	Holds one member or a string value at a time.. Scalar variable name begin with a $.
• Arrays :-	Holds a list of values. Values can be member string or other arrays. Array variable name begin with a @

- Associative Arrays :- Uses any values as an index into an array. Associative array variable name always begin with a %.

Operators:-

operators in computer language tell the computer what actions to perform. Perl has more operators than most languages. for eg:- we have already used the assignment operator. Operators are instructions we give to the computer so that it can perform operation. All operations to be performed are operands for example, the following statement 3 + 5 applies operator + on operands 3 & 5.

Operator types:-

Perl supports many types of operators.

1. Arithmetic → It includes addition, subtraction, multiplication & mathematical statements.
2. Assignment → It assign a value to variable for eg:- x = 10
3. Binding → Used during string comparisons

4. Bitwise :- These operators affect the bits that make up a value. For eg :- value 3 is also 11 in binary notation. Each character represents a bit in binary notation.

5. Comma :- It serves to separate array or list fn & also serves to separate expressions.

6. File Test :- This operator is used to test various operators associated with files. We can test for file existence, file type & access rights.

7. List :- List operators resemble fn calls in other languages.

8. Logical :- These operators implement true / false logic. eg :- AND, OR, NOT

9. Numeric Relational :- It allows to test the relationship of one numeric variable to another

10. Range :- It is used to create a range of elements in arrays. also used in a scalar context.

11. String :- Used to join two strings. ie String Concatenation.

12. String relational :- This operator allows to test the relationship of one string

variable to another.

13. Ternary :- This operator is used to choose b/w two choices based on a given condition.

Control Statements :-

3 types of control statements Decision statements, loop statements & jump statements

- Decision statements :-

The syntax of this statement is.

```
if (condition)
{
    statement 1;
}
    else
{
    statement 2;
}
```

- Loop statements :-

(i) While loop :-

The while loop has syntax :-

```
while (CONDITION)
{
    STATEMENTS1;
}
continue {
    STATEMENTS2;
}
```

(11) Do-while loop :-

```
do {
    statements;
} while (condition);
```

How to write perl program →

You can write any perl program by starting any text processor like VI.

it is used for comment

$var → variable

Print → to print

\n → new line → \n

\t tab → \t

Run Perl Program →

First give any name to program and add extension .pl.

and then run this program as

$ perl largest.pl

- Program to use chomp operator :-

Filename :- chomp_operator.

⇒ Without using chomp operator →

```
$ string 01 = " string1 \n " ;
$ string 02 = " string2 \n " ;
$ string 03 = " string3 \n " ;

print " $ string 01 $ string 02 $ string 03 " ;
```

Output :-

```
$ perl chomp_operator.pl
string 1
string 2
string 3
```

⇒ With chomp operator →

It ends the tailing new line character.

```
$ string 01 = " string1 \n " ;
$ string 02 = " string2 \n " ;
$ string 03 = " string3 \n " ;

    chomp $ string01 ;
    chomp $ string02 ;
    chomp $ string03 ;

print " $ string 01 $ string 02 $ string 03 " ;
```

OUTPUT :-

- **Use of If Statement :-**

```
if (40>30)
{
  print ("This is true");
}
```

OUTPUT :-

This is true

- **Use of Else If statement :-**

filename :- if-else.pl

```
if ("A" gt "a")
{
print ("this is true \n\n");
}
else if ("B ge "A")
{
 print (" this is true again. \n\n");
}
else
{
 print ("Default .\n\n");
}
```

OUTPUT :-

$ perl if-else.pl.

→ this is true again

- Program to find out largest of three numbers

FILENAME :- largest.pl

```
print " Enter three numbers \t";
$n1 = <STDIN>;
$n2 = <STDIN>;
$n3 = <STDIN>;
if ($n1 > $n2 && $n1 > $n3)
{
  print "$n1 is largest";
}
else if ($n2 > $n1 && $n2 > $n3)
{
  print "$n2 is largest ";
}
else
{
  print "$ n3 is largest ";
}
```

OUTPUT :-

$ perl largest.pl.

```
Enter three numbers 23
12
3
23 is largest
```

- **Program:-** Program of string concatenation and string length.

FILENAME :- string_cat.pl

```
print "Enter first string \t";
chomp (my $str1 = <STDIN>);
print "length of first string $str1 is \t",
        length ($str1), "\n";

print "Enter second string \t";
chomp (my $str2 = <STDIN>);
print "length of second string $str2 is \t",
        length ($str2), "\n";

my $final_str = $str1.$str2;
print "final string is : \t $final_str \n
    and length of final string after
    concatenation is \t", length ($final_str),
                "\n";
```

OUTPUT:-

```
$ perl string_cat.pl.

Enter first string abc
length of first string abc is : 3
Enter second string dog
length of second string dog is : 3
final string is : abcdog.
And length of final string after
concatenation is 6
```

File Compression Techniques in Linux :→ Compression is a way of reducing the size of a file on disk using different algorithms and mathematical calculations. Files are formatted in certain ways that makes their general structures somewhat predictable, even if their contents varies.

① Tar file Compression → The word tar is extracted from tape archive, this is most popular compression and decompression tool for linux. It is used to archive in multiple file formats like gzip, bzip2 etc.

Install tar in Debian/Ubuntu

```
# apt-get install tar
```

② gzip file Compression → This is most popular and fast file compression utility in Linux. It keep original file name the extension of compressed file .gz.

```
# apt-get install gzip   For ubuntu.
```

③ lzma → It is like zip or tar, but it perform quick in comparison to bzip, comes as builtin for all Linux distributions.

④ **xz file compression** → xz is successor of the lzma utility, it can only compress single file but can not compress multiple file in a single command. It was introduced in 2009 so there is a possibility that it can not be supported with all of the older linux versions.

For compression → # xz examplefile

To decompress a file →

xz -d examplefile.xz

⑤ **bzip2 file compression** → It perform more faster then gzip, it compress files and folders more compactly. It required more RAM during compressing files, to reduce memory consumption, pass -s flag in option.

Example:→

bzip2 examplefile

Pass -s flag

bzip2 -s examplefile

To decompress .bz2 compressed file.

bzip2 -d examplefile.bz2

Data Redundancy elimination using fingerprint generation deduplication →

Data Redundancy → It is a condition created within a database or data storage technology in which the same piece of data is held in two separate places. This can mean two different fields within a single database, or two different spots in multiple software environments or plateforms. Whenever data is repeated, this basically constitutes data Redundancy.

Data Deduplication → Data deduplication is the new data compaction technology which eliminates redundant copies of data. The deduplication process consists of data chunking, creating smaller unique identifiers called fingerprints for these data chunks, comparing these fingerprints for duplicates and storing only unique chunks.

Data deduplication partitions input data stream into smaller units, called chunks and

represents these chunks by their fingerprints.

After chunk fingerprint index lookup, it replaces the duplicate chunks with their fingerprints and only transfer or stores the unique chunks.

Efficiency of any data deduplication is measured by the:

- Dedup ratio (size of actual data) / (size of data after deduplication) and
- Throughput (Megabyte of data deduplication per sec.).

Data deduplication can be either client (Source) based deduplication and server (target) based deduplication, depending on where the processing occurs.

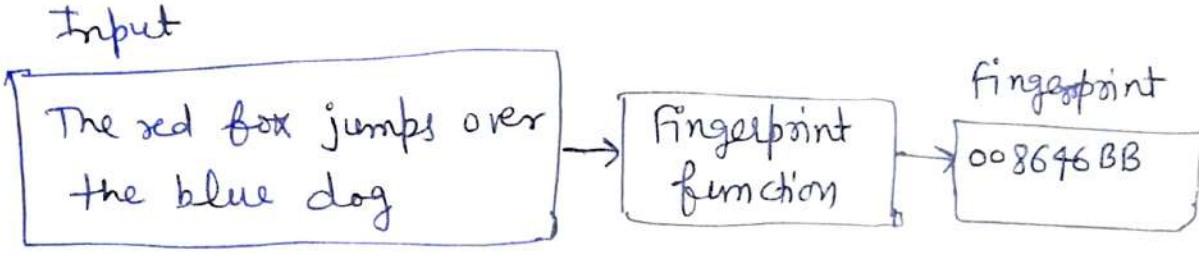
Source deduplication eliminates redundant data on the client side and target deduplication applies at server side.

Data deduplication can be categorised into Inline and Post Process depending on when the data is eliminated. If it is Inline,

then processes data before it is written on disk. An advantage of Inline is that duplicate chunks are never written to the disk.

Post Process refers to handling processing after the data has written to the disk.

Fingerprinting generation deduplication → In computer science, a fingerprinting algorithm is a procedure that maps an arbitrarily large data item to a much shorter bit string, its fingerprint, that uniquely identifies the original data, just as human fingerprints identify people. This fingerprint may be used for data deduplication. Fingerprints are used to avoid the comparison and transmission of bulky data. For instance, a web browser or proxy server can efficiently check whether a remote file has been modified, by fetching only its fingerprints and comparing it with that of the previously fetched copy.



Fingerprint algorithms:→

Rabin algorithm:→ Rabin's fingerprinting algorithm is the prototype of the class. It is fast and easy to implement and comes with a mathematical analysis of the probability of collision.

Namely, the probability of 2 strings r and s yielding the same w-bit fingprint does not exceed $\max(|r|, |s|) / 2^{w-1}$, where $|r|$ denotes the length of r in bits.

Rabin's method is not secure against malicious attacks.

Cryptographic hash functions → It serve high quality fingerprint function but hash algorithms such as MD5 (Message digest) and SHA (Secure Hash Algo) take longer to execute than Rabin's fingerprint algorithm.

Data Similarities removal using Delta technique for data reduction storage :→

Delta Technique is a way of storing or transmitting data in the form of differences (delta) between sequential data rather than complete files; more generally this is known as delta differencing. Delta encoding is sometimes called Delta compression.

The differences are recorded in discrete files called "deltas" or "diffs".

Simplest example is storing values of byte as differences (deltas) between sequential values, rather than the values themselves.

So, instead of 2, 4, 6, 9, 7, we would store 2, 2, 2, 3, -2. This reduce the variance (range) of the values when neighbor samples are correlated, enabling a lower bit usage of same delta.

Delta → Symmetric
→ Directed

A delta can be defined in 02 ways, symmetric and directed delta. A symmetric delta can be

expressed as:

$$\Delta(V_1, V_2) = (V_1 \setminus V_2) \cup (V_2 \setminus V_1)$$

where V_1 and V_2 represent two versions.

A directed delta, also called a change, is a sequence of change operations which, when applied to one version V_1, yields another version V_2.

Delta Technique is best when data has small or constant variation; for an unsorted data set, these may be little to no compression possible with this method.

(1) In data similarities removal the diff command is used as

```
$ diff file1 file2
1 c 1
< 0 top of file one
---
> 0 top of file 2
3 c 3
< 2
```

using < and > character to represent first and second of two files and report these differences in a format that could be used by the Patch command.

You can compare the two files, save the diffs to a file, and then use that file to force the second file to be the same as the first.

Create the difference file as

$ diff file1 file2 > diffs

Use the differences file to make the second file just like the first:

$ patch -i diffs file2

At this point you have two identical files. Your file2 would be just like your file1. You could then use the diffs file on any number of systems to update the targeted file.

(2) The cmp command tells you that files are different and where the first difference appears.

$ cmp file1 file2

file1 file2 differ: byte 15, line 1

(3) The Comm command will display the differences in text files in a different format.

$ comm file1 file2

So for data similarities we can apply these commands and than use this in patching Process.

After this we get the diffs and Patch that can we used for the removal of data similarities. on this we apply the Delta technique to find the most suitable result for data reduction storage.

Parallel compression with Xdelta Utility :

Parallel compression apply on different parts of an input file and on multiple cores and processes simultaneously. Its primary goal is to utilize all resources to speed up compression time and minimal possible influence on compression ratio.

Xdelta is a command line program for delta encoding, which generates the difference between two files. It was first released in 1997 by Joshua MacDonald.

This is similar to diff and Patch, but it is targeted for binary files and does not generate human readable output. It was written in C and C++ with Apache License 2.0.

The Patch file (also called a Patch) is a text file that consists of a list of differences and is produced by running the related diff program with the original and updated file as arguments. Updating files with Patch is often referred to as applying the Patch or simply Patching the files.

Xdelta is very useful to do the parallel compression. It is a patching format, similar to but different from IPS, UPS or Ninja. IPS is unable to handle file sizes bigger than 2MB. So Xdelta is popular alternative when patching CD images. Rather than applying changes directly to the file, it uses the source file plus a series of instructions to generate an entirely new file. This makes it possible to create very small patches in instances where data is moved within a file.

Unit 3 – ULP

THE C ENVIRONMENT →

The GNU C Compiler was originally used by Richard Stallman, the founder of GNU project. The GNU C Compiler is also abbreviated as GCC. The GNU project was started in 1984 with the purpose of creating a complete Unix-like OS as free software. Every Unix like OS needs a C compiler. The first GNU C Compiler was released in 1987. This was a free software.

Major features of GCC →

① Portable → GCC is a portable compiler. It runs on most platforms available today.

② Cross Compilation → GCC is not only a native compiler, it can also cross-compile any program, producing executable file for a different system than the one used by GCC itself. For eg. A GCC Compiler running on a Linux system can also produce executable files for a windows system.

③ Multiple Front-ends → GCC has multiple language front-ends for parsing different languages. Front-end means the language which we are using to write our code.

④ Modularity → GCC supports a modular design allowing support for new languages and architecture to be added.

⑤ Free of Cost → GCC is a free software distributed under the GNU general Public license abbreviated as GNU GPL. This means everyone has the freedom to use and modify GCC as with all GNU software.

The C Compiler → Compilation refers to the process of converting a program written in some programming language such as C or C++ into machine code which is a sequence of 0's and 1's. This machine code is then stored in a file known as executable file which is referred to as binary file.

Make a C Program →

We can make a C program by 3 options.

① Right click on Desktop → Create document → Empty file.
open the file and write C program and then save it with .c extension.

② write a C program using cat command as:

$ cat > largest.c

③ write a program using VI Editor as:

$ vi largest.c

Run a C Program →

We can run a C program using gcc compiler.

gcc → GNU compiler collection

GNU → GNU means 'GNU' Not Unix/Linux OS.

Program 1: hello.c

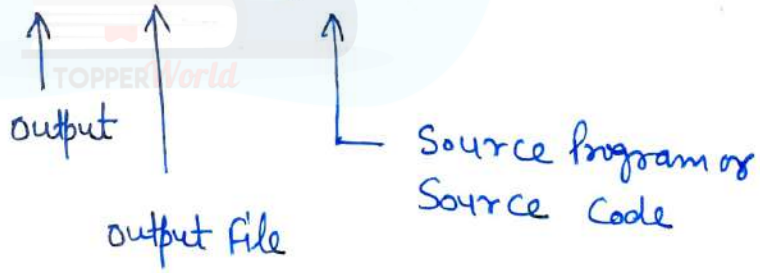
```
#include <stdio.h>
void main()
{
  printf("Welcome");
}
```

① $ gcc hello.c This command compile the hello.c program. we have default output file of each c program i.e a.out. So if we want to run this program then we have to use command as:

$ ./a.out

② If we want to give every c programs different output file name then use the option for compile a c program as:

$ gcc -o lar largest.c



Run this program as:

↓

$ ./lar

VI editor → The vi editor is the most popular editor in Linux. The current version is really "vim", but to invoke it simply type "vi".

Syntax of VI →

$ vi name_of_the_file

once the file is open, you can type any text by pressing a or i key from the keyboard.

Modes →

① Command Mode → moving cursor or the keys h, j, k, l. we can not write text in this mode.

② Insert Mode → write text in this mode and then save file.

$ vi viexample.txt

Getting out of vi (Command mode) →

First press Esc key

quite → :q!
write → :w!
reload → e!
write and quite → shift ZZ

Moving the cursor →

Key	Cursor Movement
h →	left one space.
j →	down one line.
k →	up one line.
l →	right one space.

Editing the text →

i →	insert before character.
a →	insert after character.
x →	delete one character.
dd →	delete current line.

Managing Projects:→

When the size of programs increases especially in case of projects, the process of building these projects becomes complex and time consuming because large amount of CPU time consumed and when errors then debug time also increases.

Thus it becomes necessary to manage these projects. The Linux environment provides a tool to manage these projects called GNU make. Consider the example of assembly line for production of cars. Every parts must be linked together for final product. If anything goes wrong, car may not work properly. For everything working properly, both human and computers control is necessary to regulate the flow of materials from one place to another.

This build process involves executing thousands of commands, and if these commands extracted

manually then the process becomes tedius and it contains dozens of errors.

So there is an autonomous system to build process for code. we define these rules to define how the code is built and then system applies these rules to build the entire project.

The file where these rules are defined are called make file. These program invokes compiler, linkers, assemblers and all other programs that are needed to build the final executable.

when we type "make" on command line, the system automatically examines the rules and files.

After the changes if we run "make" on command line then system automatically detects these changes and performs necessary actions needed to update the program with these changes.

Use of Make Files →

A make file is a collection of rules. Each rule in a makefile defines 03 things namely:

① First when the file is built and rule is processed.

② Second as the process we must go through to make files into final product. For example, for executable C program, first we compile and then link the objects files.

③ Third, is list of dependencies for each file. These dependencies need to be calculate before we can process a file.
For example in C program functions like strlen(), math() require <string.h> and <math.h>.

① A Simple Makefile →

Consider the code which consists of 03 source files and 01 header file. Source files are init.c, calculate.c, output.c and header file stdio.h.

The code for 03 files are as:

① For init.c, the source is

```
#include <stdio.h>
int x=10;
main (void)
{
  print ();
   return 0;
 {
```

② For calculate.c, the source is

```
extern int x;
 compute (void)
 {
  return x* x;
 {
```

③ For output.c, the source is

```
#include<stdio.h>
int print(void)
 {
    printf ("The value is %d", compute ());
    {
```

④ For header file program.h, the source is

```
int  compute (void);
int  print (void);
```

Now, the code for makefile is,

```
all : program
```

Here, all the name of default target program contains all 03 objects files link together, as

```
program : output.o init.o calculate.o
```

Now each of these files have the corrosponding c files as their dependencies as given below,

```
calculate.o : calculate.c
gcc - wall -o calculate.o compute.c
init.o : init.c program.h
gcc - wall -o init.o init.c
output.o : output.c program.h
gcc -wall -o output.o output.c
```

Now, we can compile the entire program with a single command at the prompt.

Make build our program

```
$ make
gcc - wall - o calculate.o calculate.c
gcc - wall -o init.o init.c
gcc - wall -o output.o output.c
gcc - wall -o program calculate. o init.o output.o
```

Building & Using Static Libraries → A library is a collection of precompiled object files which can be linked into programs. The most common use of libraries is to produce system function such as the square root function sqrt found in Math library. Libraries are special files with the extension 'a' referred as static libraries. They are created from the object files with a special tool, the GNU archiver or used by linker to resolve references to functions at compile time. The standard system libraries are usually found in the libraries /usr/lib. For example, the math library is typically stored in the file

/usr/lib/libm.a on Unix like system.

Consider the example which make call to the external function sqrt in the math library 'libm.a'.

```
#include <math.h>
#include <stdio.h>
main (void)
{   double x = sqrt (4.0);
    printf ("The square root of 4, 0 is %.f, x);
    return 0;
{
```

The compiler also gives an error from this source file at the link stage.

```
gcc -wall roof.c - o roof
/tmp/ccb.o: In function 'main':
/tmp/ccbr.o: Undefined referred to 'sqrt'
```

The problem is that reference to sqrt function can not be resolved without the external math library 'libm.a'. The function sqrt is not default library 'libm.a'. The compiler does not include 'libm.a' unless it is explicitly selected. To enable the compiler to link the 'sqrt' function in the main program we need to supply the library 'libm.a' on command line:

```
gcc -Wall root -c /usr/lib/libm.a -o root.
```

The library 'libm.a' contains object files for all mathematical functions such as sin, cos, log and sqrt.

Building and Using dynamic Libraries →

Even a program has been sucessfully compiled and linked, a step is needed before being able to load and execute directly, the following error will occurs:

```
$ ./a.out
./a.out: error while loading shared libraries
libgdbm.so.3: can not open shared object file.
```

This is because the GDBM package provides a shared library. Shared library is also known as dynamic library.

So → is shared object.
GDBM is GNU database manager.

Before the executable file starts running, the machine code for external functions is copied

into memory from the shared library file on the disk by the operating system by the process of dynamic libraries linking. Dynamic linking makes executable files smaller and shrinks disk space, because one copy of library can be shared by multiple programs.

Most OS also provides a virtual directory mechanism which allows one copy of a shared library in physical memory to be used by all running programs, saving memory as well as disk space.

Dynamic Loader → The dynamic loader also known as dynamic linker is automatically invoked when the program is invoked. Its job is to ensure that all the essential libraries that the program needs are present in memory with their proper versions. It is named either as ld.so or ld-linker.so depending upon linux C library version.

In Linux DL libraries are standard object files or standard shared libraries that are not automatically loaded at program link time, there is an API (Application programming Interface) for opening a library, looking up symbols, handling errors and closing the library.

C users will include the header file <dlfcn.h> to use this API.

$ ldd /bin/ls

This command will give the information of all the shared objects.

$ ldd /bin/ls

linux-gate.so.1 ⇒ (0x00ccb00)
libselinux.so.1 ⇒ /lib/libselinux.so.1 (0x00994000)
librt.so.1 ⇒ lib/librt.so.1 (0x00989000)
$

Dynamic Loader Functions →

→ dlopen() → This function opens a library and prepare it for use.

→ dlerror() → To report error.

→ dlsym() → Look up the value of a symbol in a opened library.

→ dlclose() → It closes a DL library.

Debugging with GDB →

GDB is GNU debugger.

Normally an executable file does not contain any reference to the original program source code such as variable names, line numbers. This is insufficient for debugging since there is no easy way to find out the cause of error if the program crashes.

GCC provides the '-g' option to store the additional debugging information allows errors to be traced back.

Using the debugger we can also examined the values of variables used in the program.

The Command 'ulimit -c' controls the maximum size of core files. If the size is 0, no core files is produced.

The current size of file can be checked by typing the command as

```
$ ulimit -c
0
$
```

If the result is 0, the size can be increased with the following command to allow core of any size to be written.

```
$ ulimit -c unlimited
$ ulimit -c
unlimited
$
```

A Null Pointer cause a problem at run time and a message " core dumped" is displayed. The OS produce a file called "core file" in the current directory and this file contain the complete copy of the pages of memory used by the program at the time it was terminated. we have the "segmentation fault" to tried the restricted memory segment.

So both executable file and core file are required for debugging. It is not possible to debug a core file without corrasponding executable file.

We can load and execute the core file and executable file with the following command:

$ gdb a.out core

The debugger immediately begins printing diagnosis functions and shows listing of lines where the program crashed.

$ gdb larger

use gdb with larger program.

(gdb) info functions

↳ This will give information about all the functions used in the C program.

(gdb) b → Breakpoint

(gdb) info b → Information about all breakpoints.

(gdb) b main

↳ Breakpoint 1 at 0x 80484 1d: file largest.c, line 5.

(gdb) r

↳ to run c program.

(gdb) c

↳ To continue with the program.

(gdb) list

↳ For source code of the program.

(gdb) p

↳ gt will print the value of variables.

(gdb) q

↳ To quit, we use q option.

(gdb) disass

↳ For getting assembly code, we use disass option with gdb.

CMake : CMake is extensible, open-source system that manages the build process in an operating system and in a compiler-independent manner. Unlike many cross-platform systems, CMake is designed to be used in conjunction with the native build environment.

CMake also supports static and dynamic libraries builds.

How to Create CMake file? →

1. open the top-level CMakeLists.txt file;
2. open the CMakeCache.txt file;
3. Locate a directory with generated CMake files and open it as a project.

Use of Cmake :→

CMake is software for managing the build process of software using compiler independent method. It is designed to support directory hierarchies and applications

that depends on multiple libraries. It is used in conjunction with native build environments such as

Make,
Apple's Xcode, and
Microsoft visual Studio.

It is part of a family of tools designed to build, test and package software.

(1) CMake copyright is an open-source, BSD license.

(2) It allows unrestricted use, including use of CMake in commercial products.

(3) copyright (c) 2002 Kitware.

(4) Available Build Environments

Unix | Linux → Make files
Windows → VS Projects | Workspaces
Mac → Xcode

Dependency Calculations → we have calculate and determined the names of all source and object files in our project automatically. Now if all the dependency calculate automatically then this will be great benefit for the Programmer.

When dependencies are calculated manually, the programmer must update manually the makefile every time it includes any other file or removes an existing one.

Calculating dependencies automatically have the advantage that makefile never needs to be updated even when new files are added to the code having dependences on other header files.

The dependency files are generated by using the -M option with gcc. This option asks the compiler to suppress the normal output and examine the source file and output

an actual make rule specifying the dependencies C file on the right.

If a given header file is modified then the dependencies files are automatically updated.

The use of -M option is shown below as:

```
$ make
gcc -M io.c > io.d
gcc -M init.c > init.d
```

Here io.c and init.c are source files. io.d and init.d are corrosponding dependence files.

First the dependency files are created using -M option which are stored in the corrosponding source files. After that source files are compiled one by one and finally linked for the final executable code.

Memory Management :→ It is a fundamental concern to the Programmers in C. In C, we have to explicitly manage memory allocation and its removal. The language does not do this for us. There are number of options available for managing memory. Some of them are easy and others are complex.

Memory allocated in C is Primarily done in two ways

1. Statically Allocated Memory
2. Dynamically Allocated Memory

1. Statically Allocated Memory → As the name indicate it is static in nature. This type of memory is allocated by the system implicitly. The advantage of this type of memory is that it is always there whenever we are in the relevant area.

An example of data structure using static memory allocation is array.

For example consider the declaration:

```
int array [50];
```

The above declaration creates an array of integer type having 50 items. This memory allocations however two disadvantages:

① Since the amount of memory to be allocated is done at compile time. If only 10 items are used then remaining memory will be wasted.

② If at run-time we came to know that we need memory for 100 items then result would be in memory overflow.

② **Dynamically Allocated Memory:→** Dynamic memory stands for memory which we can allocate on request. This is also known as run-time memory allocation.

For dynamically allocation and deallocation, C language provides:

① **malloc ( ) →** The malloc function allocates a block of memory in bytes. It is used

to allocate a single block of memory of specific data type.

```
ptr = (type*) malloc(size);
```

where ptr = pointer variable
type = data type to be stored in memory.
size = size of memory to be allocated.

② Calloc() → It is used to allocate multiple blocks of same size during the program execution.

```
ptr = (type*) calloc(i, j)
```

where = ptr is pointer variable
type = data type to be stored in memory
i = no. of blocks to be allocated
j = no. of byte in each block.

It has two parameters whereas in malloc only a single parameter is exist.

③ Realloc(): → This function is used to reallocate or modify the space which was previously allocated. This facilitates to increase or reduce the allocated space at a later stage in program. The first byte

of the newly allocated space is assigned to the pointer. It has the following form.

ptr = realloc (ptr, size)

where ptr is pointer variable

size = no. of bytes to be reallocated newly

④ Free () → This function is used to free the space which is allocated using malloc() and calloc().

It has the following form;

free (ptr)

where ptr is pointer variable to which space is allocated in memory.

Fixed-size and Variable-size block of data files chunks divisor chunking techniques :

In the deduplication technology, data are broken down into multiple pieces called "chunks" and every chunks is identified with a unique hash identifier. These identifiers are used to compare the chunks with previously stored chunks and verified for duplication. Since the chunking algorithm is the first step involved in getting efficient data deduplication ratio and throughput, it is very important in the deduplication scenario.

The key objective of chunking algorithm is to divide the data object into small fragments. The data may be a file, a data stream, or some other form of data.

Chunking Methods :→ Efficient chunking is one of the key elements that decide the overall deduplication performance. There are no. of methodologies to detect duplicate chunk of data using fixed-size chunking and

fixed-level chunking using rolling checksums.

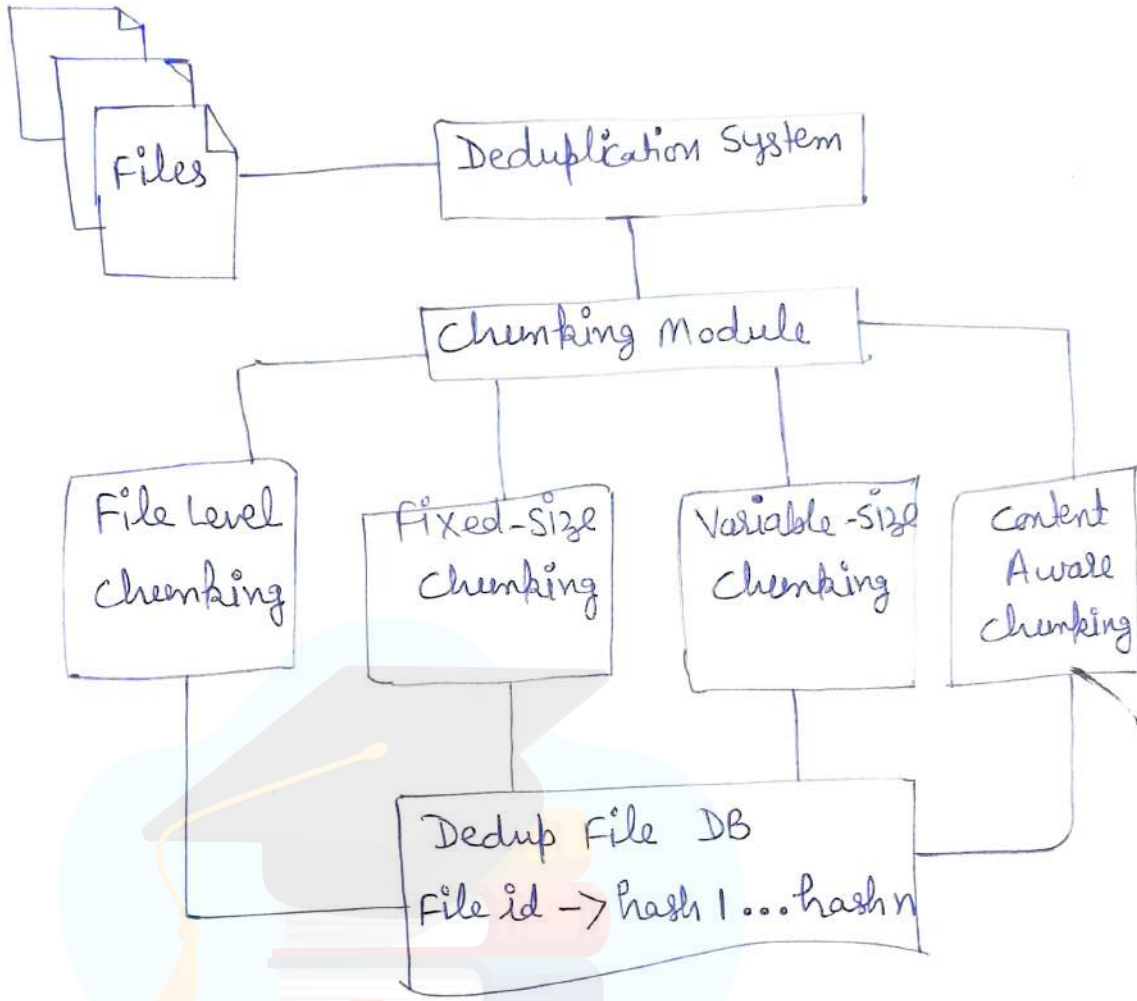


Figure: Different chunking module structure

① File-Level chunking :→ File level chunking or whole file chunking considers as an entire file as a chunks, rather than breaking files into multiple chunks. In this method, only one index is created for the complete file and same is compared with the already

Stored whole file indexes. As it creates one index for whole file, this approach stores less no. of index values, which in turn save space and helps store more index values compared to other approaches. It avoids maximum metadata lookup overhead and CPU usage.

It reduces the index lookup process as well as the I/O operation for each chunk.

Disadvantages: This method fails when a small portion of the file is changed. Instead of computing the index for the changed part, it calculates the index for the entire file.

(2) Block-Level chunking →

(2.1) Fixed-Size chunking
(2.2) Variable-Size chunking

(2.1) Fixed-Size chunking → Fixed-Size chunking

method splits files into equally sized chunks. The chunk boundaries are based on offsets

like 4, 8, 16KB etc. This method effectively solve issues of the file-level chunking method. If a huge file is altered in only a few bytes, only the changed chunks must be reindexed and moved to the backup location.

However, this method creates more chunks for larger file which requires extra space to store the metadata.

→ As a file splits into fixed-size, byte shifting problem occurs for the altered file.

Disadvantages :→ ① If the bytes are inserted or deleted on the file, it change all the subsequent chunk position which results in duplicate index values.

② Hash collision is likely to happen on chunking method by creating same hash value for different chunks. This can be eliminated by using bit-by-bit comparison which is more accurate, but requires more time to compare the files.

ULP - Unit 3

(2.2) **Variable-Size chunking:→** The Files can be broken into multiple chunks of variable-size by breaking them up based on the content rather than on the fixed-size of the file. This method resolves the fixed-size chunk size issue. In case of fixed-size chunking algo, fixed boundaries are defined on data based on chunk size which do not alter even when the data are changed. However, in case of variable-size algorithm different boundaries are defined, which are based on the multiple parameters that can shift when the content is changed or deleted.

* Widely used variable-size algorithm is Rabin's algorithm to create the chunk boundaries.
* Delta Encoding is another method to find the boundaries of the file.
* Compared to variable-size and fixed-size chunking, Delta encoding gives a good deduplication ratio in desktop applications such as Word, Excel, Zip, Photos through the use of fingerprints.

Basic Sliding Window → The other approach is the basic sliding window (BSW) approach. It applies break condition logic and marks the boundaries of file. Chunk boundary is computed based on fingerprint algorithm. File boundary is marked based on break condition. The problem with this approach is the chunk-size. The size of the chunk can not be predicted with this approach.

Two Threshold Two Divisor (TTTD) :→

This is also variable-size length chunking algorithm. With this method chunks smaller than a particular size are not produced. However, it has a drawback that the chunks produced might escape duplicate detection as larger chunks are more likely to be related than smaller ones, To achive this, the method ignores the chunk boundaries after a minimum size is reached. TTTD applies two techniques where two divisors (D, the regular divisor and D0, the backup divisor) are used.

By applying these divisors, TTTD guarantees a minimum and maximum chunk size. A minimum and a maximum size limit is used for splitting a file into chunks for searching for duplicates.

TTTD-S algorithm → This algorithm overcomes the drawback of the TTTD algo by introducing TTTD-S algorithm. It computes the average threshold as a new parameter of the main and backup divisors and uses it as a benchmark for the chunking algorithm.

Content or Application Aware-Based Chunking :→

The earlier chunking method do not consider file format of the file of the underlying data, such as file characteristics. If the chunking method understand the data stream of the file (format of file), the deduplication method can provide the best redundancy detection ratio compared

with fixed and variable size chunking methods.

The fixed-size and variable-size chunking methods set the chunk boundaries based on the parameter or some predefined condition, whereas the file-type aware chunking understands the structure of original file data and defines the boundaries based on the file type such as audio, video or document.

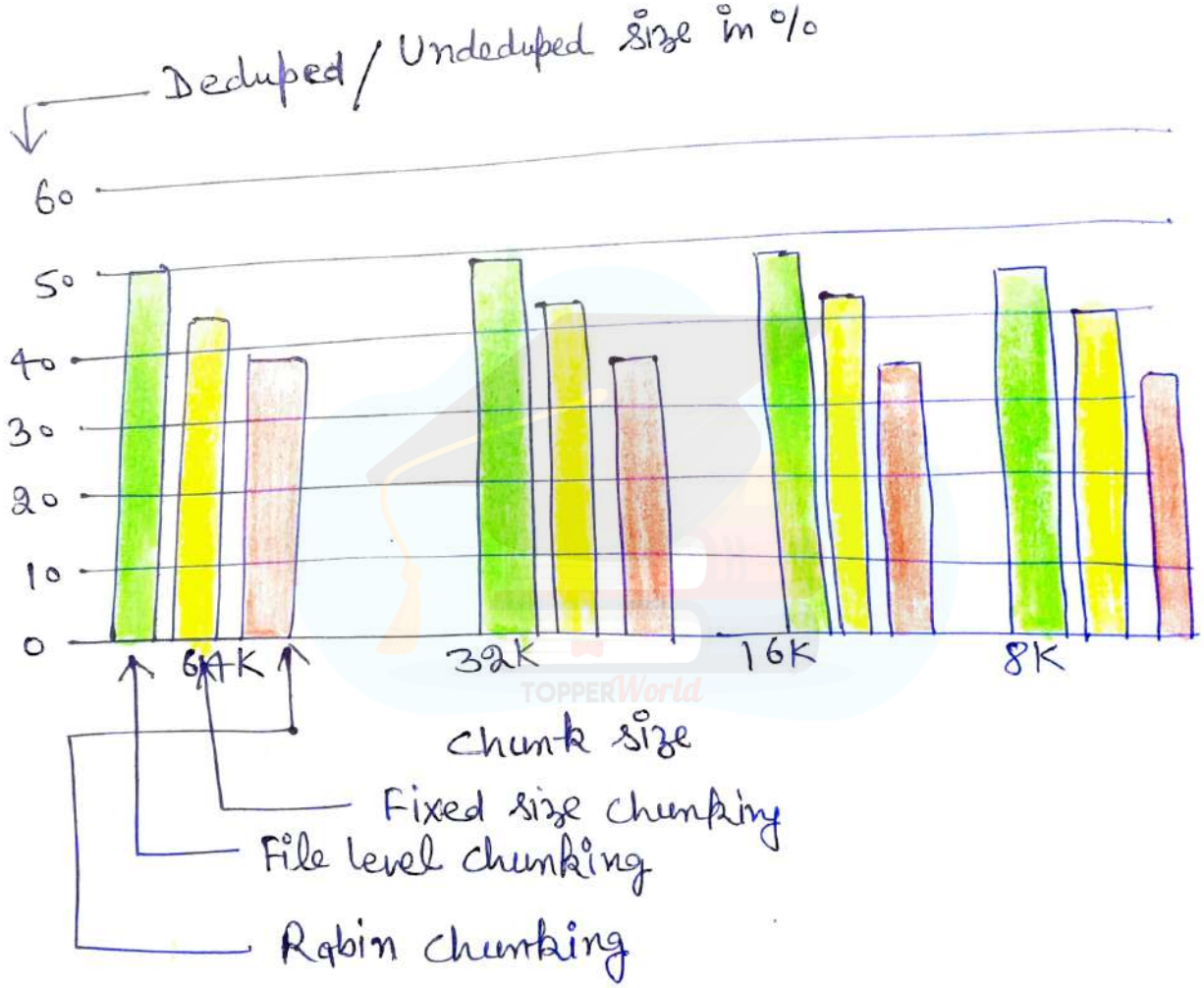
Advantages: → ① The advantage of content-aware or content based chunking method is to save more space because this algorithm is more aware of file format and sets the boundaries more natural than other algorithm methods.

② If the file format is known, it also provides the option to change the chunk-size depending on the section of the file.

③ By this method it can assume how often a file or section is going to change in the future and size of the chunk accordingly.

ULP - Unit 3

→ Comparison of different deduplication versus various chunk-size.



① Figure depicts that File-level and Fixed-size chunking occupying more spaces compare than Rabin chunking algorithm.

② Rabin chunking algorithm provides good

UNIT-4 ULP

Introduction to Processes → A Process is a concept by which we can understand and control the execution of a program in an operating system. A process is also defined as a running instance of a program.

Each process in Linux system is identified by a unique process identification number, also called as pid which is allocated to it by the kernel.

A process is represented by the type.

process = (process id, code, Data, register value, PC value)

→ **PID →** Every process has unique process ID that distinguishes it from other processes in the system.

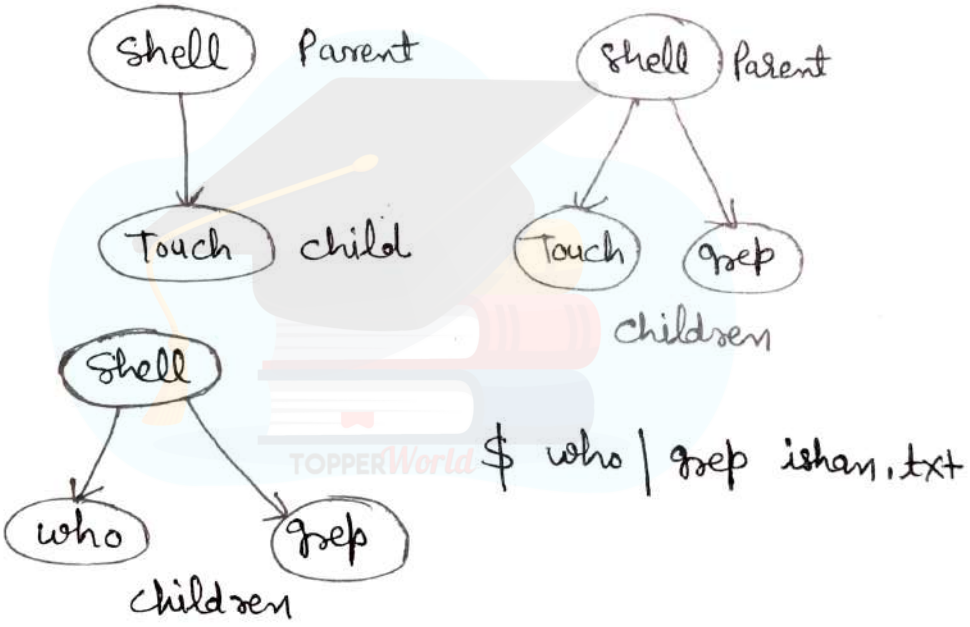
→ **Code →** Code is the program code that is under execution.

→ **Data →** Data is the data used by the program code during execution. Data may be variables, files and so on.

→ **Register values →** Register values are the values shared in CPU registers when the calculation work is performed during program execution.

→ PC value → address of the program counter from where the execution of the program starts or continues. Maximum value of PID that can be assigned is 32767.

⑪ **Parent and child Processes** → A Process is responsible for generating other processes. A process that generates a new process is known as Parent Process and new process is known as child Process.



Parent and child Processes

For example, consider the following command.

$ who | grep ishan.txt

when this command is given to shell, two processes, for who and other for grep command are created simultaneously. Shell is the Parent of both of these processes.

(2) Types of Processes →

(1) Interactive Processes and (2) Non-Interactive (3) Daemons

(1) Interactive Processes → All the processes which are created by the users with the shell act upon the direction of users and normally attached to terminal are called Interactive Processes. These processes are also called as foreground processes.

(2) Non-Interactive Processes → Certain processes can be made to run independent of terminals. These processes that run without any attachment are called as non-interactive processes or background processes.

```
$ ls -l | grep main > largest.c &
   2447
$
```

To make a command been in background, we need to terminate the command line with the ampersand (&) character. After this command, the shell returns a process Id no. of currently submitted background job.

③ Daemons → There are certain processes which keep executing in the background whenever the system is started eg. init and swapper.

③ Viewing Active Processes → The "ps" command is used to display the processes that are running on the system.

ps is process status.

```
$ ps
  PID   TTY      TIME     CMD
 2183   pts/0    00:00    bash
 2745   pts/0    00:00    ps
 $
```

Two processes, one is bash and other is ps itself

```
$ ps -f                                   Full listing of processes
UID        PID  PPID  C    STIME TTY      TIME CMD
rajender  2183  2181  0    14:20 pts/0    00:00 /bin/bash -l
rajender  2783  2183  0    16:02 pts/0    00:00 ps -f
$
```

other options →

-a → Processes not including system processes.
-u → Processes corresponding to particular user.
-e → Processes including system processes.
-o → output as comma-separated list.

UNIT-4 ULP

Starting and Stopping Process →

A Process is just a program that is running. Each process contains information that are needed by the CPU to run the program. Each process runs in its corrosponding address space which comprises of 03 segments:

① **Text Segment →** This segment contains the code that needs to be executed. Since several users may be using this area segment, this area remains fixed.

② **Data Segment →** All the data that the program uses like array and variables are held here.

③ **User Segment →** This segment contains all the attributes related to the process namely userid's, groupid's, and more. This information is used by Kernel to manage all processes.

① **Using "System" Function: →** The "system" function in c library provides an easy way to execute a command from within a program just like as if the command had been typed into the shell.

Infact, the "system" command create a sub process running the standard Bash shell.

For example the following program invokes the "ls" command to find the contents of the root directory as if the command "ls -l" were typed into the shell.

```
#include<stdlib.h>
int main()
{
  int returnvalue;
return value = system ("ls -l");
 return returnvalue;
```

The "system" function returns the exits states of the shell command if the shell itself can not be run, system return 127 and if some often error occurs, system returns -1.

(2) **FORK()** → The DOS and Windows API contains the family of functions to start processes and to create processes. Linux does not contain a single function that does this all in one step. Instead, Linux provides one function, "fork" that makes a child process that is an exact copy of its parent process.

UNIT-4 ULP

To create a new process, first we need to use "fork" to make a copy of current process. then, we need to use "exec" to transform one of these processes into an instances of the program that you want to spawn.

→ A Shell creates child processes.

like

```
$ date
Sat Nov 5 20:55:38 IST 2016
$
```

Here shell is the Parent process and date is the child process.

→ - A shell can also give birth to another shell process.

* creating process is also called spawning a process.

```
$ sh
sh-4.1 $ date
        Sat Nov 5 20:57:18 IST 2016
        Sh-4.1 $
```

Here date is child Process.

sh (Shell) is Parent Process of date (child) Process.

Bash shell is Grand father of date, Father of Sh.

```
sh-4.1 $ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
rajender  2220  2218  0 20:32 pts/0    /bin/bash -l
rajender  2303  2220  0 20:56 pts/0    sh
rajender  2311  2303  0 20:59 pts/0    ps -f
sh-4.1 $ exit
```

→ we can also get complete information of all the processes with the command "ps -ef"

```
$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 20:28 ?        00:00 /sbin/init
root         2     0  0 20:28 ?        00:00 [kthreadd]
root         3     2  0 20:28 ?        00:00 [migration/0]
root         4     2  0 20:28 ?        00:00 [ksoftirqd/0]
root         5     2  0 20:28 ?        00:00 [migration/0]
             .
             .
             .
and so on
$
```

If a process contains n fork() calls, it will creates

$2^n - 1$ child processes.

UNIT-4 ULP

Parent Process

```
main()
{
printf("hello");
}
```

output → 01 time hello Print

```
main
{
fork();
printf("hello");
}
```

Parent Process

Child Process

```
main()
{
printf("hello");
}
```

2 Process
2 times "hello" prints

Parent Process

```
main()
{
fork();
fork();
printf("hello");
}
```

child 1

```
main()
{
fork();
printf("hello");
}
```

child 2

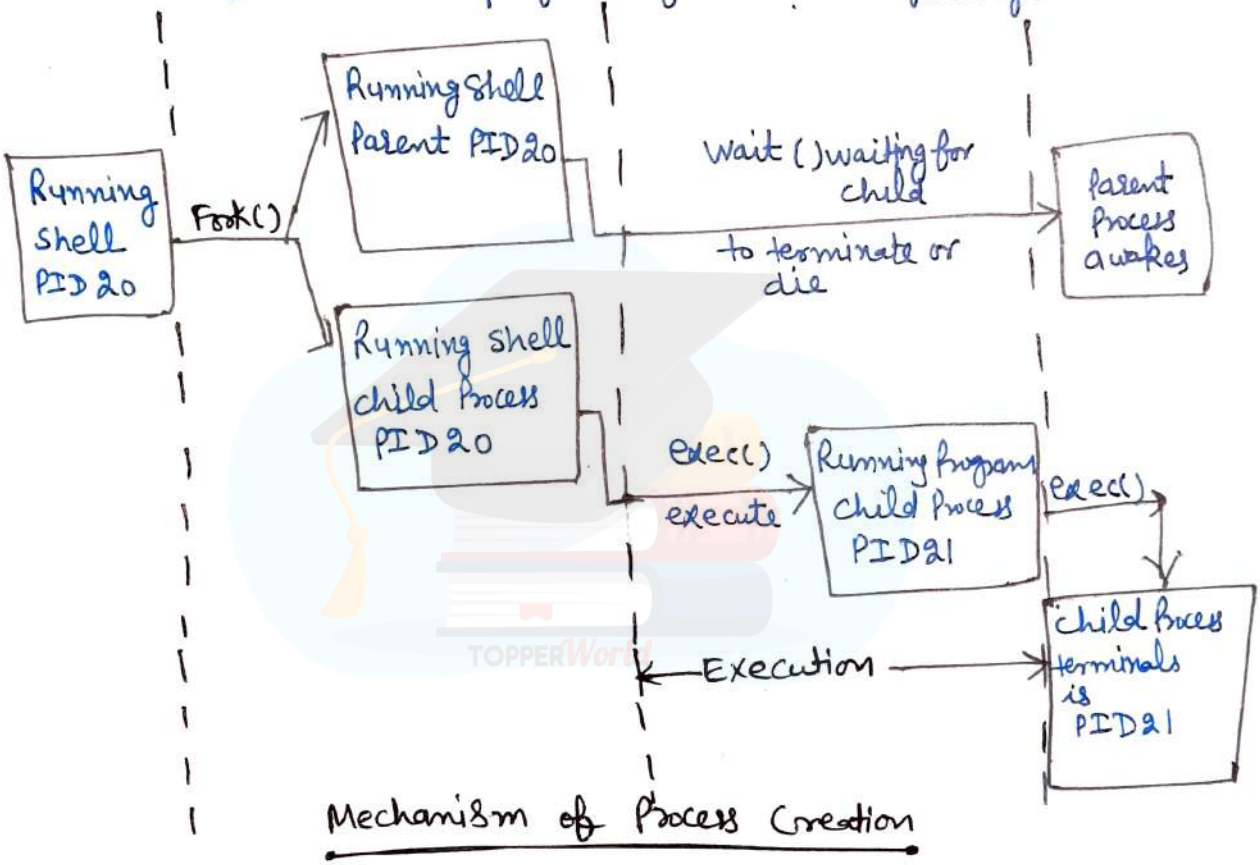
```
main()
{
printf("hello");
}
```

child 3

```
main()
{
printf("hello");
}
```

output: → 3 child Process
4 times Hello Print

③ Exec Family → The "exec" functions replaces the program running in a process with another program. when a program calls an "exec" function, the process immediately stops executing that program and begins executing a new program from the begining.



Mechanism of Process Creation

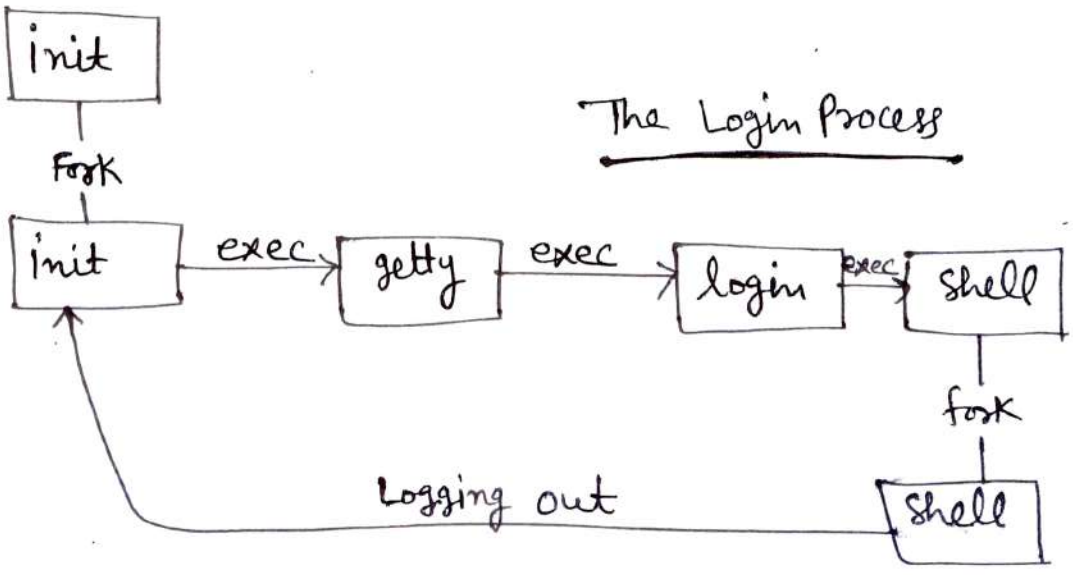
④ Wait() signal → This is the final step in creation of process. Immediately after the forking, the Parent Process makes a system calls to one of the wait() functions. By doing so, Parent Process is waiting for child process to complete its task. It awakens only when it receive a complete signal from child after which Parent is free to continue with its other functions.

UNIT-4 ULP

Initialization Processes → when a Linux system is booted, a process starts which initialize all the processes of the system. This process is called init process and it is generally termed as the Parent of all the processes. Init is very important process and all other processes in system are child of this Process.

It has a PID of 1.

All other processes has PPID (Parent PID) 1, which means these are all child of init processes (PID = 1).



The Login Process

whenever a system starts and moves to multi-user mode, the init process spawns a process called getty for every terminal. Each one of these getty prints the login terminal prompt on the respective terminal.

when a user attempts to enter the Unix environment, the login program is executed in order to verify the username and password.

On sucessful login, the login program runs a shell corrosponding to that user. The shell then prints a prompt usually $ or # and waits for further input.

After logout, control is returns to the init Daemon.

Init also consists the run level and decides which process to run for each run level. Init also ensures the proper running of all daemons.

There are 06 different run levels or states in which the system call be present. Each run level is normally a single digit between 0 and 6. A different set of processes are scheduled

Run level	Process Executed
0	System Shutdown
1	Single user mode
2	Multi user mode
3	Full Multi user mode
6	Shutdown or reboot mode

① **init files** → The behaviour of init is controlled by the file /etc/inittab. Init reads all the instruction from this file only once. Its fields determines the process that should be spawned for each of the init states and program to run at various ports.

For example the inittab file:

```
fs : :sys init   : /sbin/rc5 sysinit
ls : 3           : initdefault :

s0 : 0 : wait : /sbin/rc0
s1 : 1 : respawn : /sbin/rc/1
```

level — run level — Action — command

There are number of actions that init understands. These are:

- ① sysinit → It is used for initializing the system. It also asks input from administrator.

② Respawns → This action makes sure that the process restarts on termination. This is always required for getty process.

③ Boot → This action is executed when the inittab file is read for the first time. All run levels are ignored here.

④ off → Kill the running process.

⑤ control Alt Delete → Executes the shutdown command.

② rc Files → init and /etc/inittab completely control the way system is booted and powered down. whenever the system change a run level, init looks the inittab file to identify which should or which should not running for that level. It first kills the process that should not be running and respawns those that should be running. Every line in inittab file specifies the execution of some rc scripts specified in /etc/ or /sbin.

```
S0 : 0 : wait : /sbin/rc0
S2 : 23 : wait : /sbin/rc2
S3 : 3 : wait : /sbin/rc3
```

UNIT-4 ULP

JOB CONTROL → Linux is a Multitasking system and there will be many number of files or processes running simultaneously in the Linux environment.

There are lots of important job control commands:

① Jobs → This command list all the job associated with the current shell. It also give the job ID's.

```
$ jobs
[1] + Running.    ping www.hctmKaithal.com &
[2] + Running     ping www.google.com
$
```


[−] → Previous job
[+] → current job
& → background Processes

② fg → This command is used to move background processes to foreground. You will need to get the job ID of the Process with the job command. Then prefix the job ID with a % after fg command.

fg → foreground.

Example → $ fg %1

fg → foreground

%1 → Job ID of Process

③ bg →

bg → background

This command move foreground processes to background. To do this you must use (Ctrl + Z) to pause the Process. when you do this it will stop it and give you a job number.

Prefix this job no. with the % after the bg command.

Example → $ bg %1

[1]+ Ping www.hctmkaithal.com &

$

④ Batch →

Job submitted by using batch command are executed when the system is relatively free and system load is light. Time at which command will be executed is decided by the system.

```
$ batch
at> grep unix file.txt
ctrl+D
job 16.6 at 2016-11-08 21:16
```

UNIT-4 ULP

The extension .b attached to a job identification no. indicates that the job has been submitted using batch command.

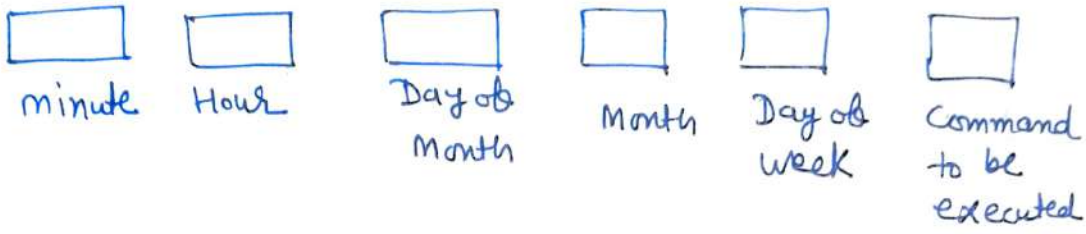
(5) TIME → This command is used to know the resources usage is, and how much time is needed to execute a command.

```
$ time
real    0m0.000s
user    0m0.000s
sys     0m0.000s
$
```

(6) Crontab → cron is a daemon that keep sleeping most of the time. It executes commands that need to be run regularly according to time schedule. The schedule and corresponding time is stored in the file /etc/crontab.

Cron daemon wakes up every minute and check its crontab file for any job that need to be executed during this minute.

Every entry in crontab file /etc/crontab contain six fields separated by space or tab in the following form:




```
$ crontab -l
no crontab for rajender
$
```

This file is submitted to crontab as:

```
$ crontab contentsfile ↵
```

$ crontab -l → list user's crontab.
-e → edit user's crontab.
-r → delete user's crontab.
-i → prompt before deleting user's crontab.
-S → selinux context.

⑦ **at command** → This command is capable of executing Linux command at a future date and time. The input to this command comes from the standard input, keyboard. Then the command typed.

once the job is submitted using at command, all the details regarding jobid no., date and time at which commands are executed to be displayed.

For example:

```
$ at 21:45 Nov 8 2016
at> cat > manisha.txt
  ctrl+d
job 18 at 2016-11-08 21:45
$
```

So manisha.txt file create at 21:45 on Nov 8, 2016. This is future time to execute a command.

⑧ **Kill** → Kill command can be used to terminate running processes and daemons.

It can eliminate lots of processes with the combination of ps command and these processes can not be stopped on unix system otherwise.

There are 64 different signals that can be sent. You may list them with

$ kill -l option.

$ kill -l

This command list total 64 kill signals.

for example:

1) SIGHUP
2) SIGINT
3) SIGQUIT
5) SIGTRAP
9) SIGKILL
19) SIGSTOP
30) SIGPWR

$ kill %1

This command will kill the job number 1 in background.

$ kill -9 1

To force a process to terminate use -9 option (SIGKILL).

$ kill -9 0

You can kill all processes including the shell by using 9 as option and 0 as argument to kill command.

UNIT-4 ULP

Network Files → These files contain information about various network parameters. Network files are also called as network configuration files.

① /etc/hosts → In a network every computer is known as host and every such host has a hostname. This file contains list of host names and corresponding IP addresses.

Sample of /etc/hosts are as:

127.0.0.1 localhost localhost.localdomain

② /etc/inetd.conf →

inetd is a daemon which is responsible for starting services like telnet, ftp.

inetd listens on multiple ports for any connection requests. When a new connection is detected, it defined that port.

```
ftp     stream  tcp  nowait  root  /usr/sbin/tcpd
telnet  stream  tcp  nowait  root  /usr/sbin/tcpd
```

③ /etc/services →

This file contains the port number used by various services on the system. This file related to inetd.conf way in its first field namely services name, this file contains two fields: services name and protocol used.

ftp	21/tcp
telnet	23/tcp
smtp	25/tcp

④ /etc/resolv.conf → To resolve hostname into

IP addresses, we need to use domain name servers. These servers map hostnames to the IP addresses. This mapping is necessary because internet recognise only IP form and we can not remember the IP address of each site.

The enteries of this file is:

```
# Generated by networkManager
domain localdomain
search localdomain
nameserver 192.168.12.2
```

UNIT-4 ULP

SECURITY :→

Much of the Power of a Linux system comes from its support for multiple user and for networking. Many user can use the system at once and they can connect to the system from remote locations. But there is a risk, especially for systems connected to the Internet. Under some circumstances, a remote hacker could read, modify or remove each other's files when they should not allowed to do so. When this happen, system security is said to have compromised.

The Linux kernel provides a variety of facilities to ensure that these events do not take place.

① Users and Groups →

Every Linux system is assigned a unique number called a User ID or UID. when we login, we use a user name rather than a user id. The system converts user name to a particular user ID. There can also be more than one username corresponding to a particular user id. But there is no way to give

one username more power than the other one if they both belong to same User ID.

Each group is assigned a unique number called a groupid or GID. Every group contains one or more user id's. A single user ID can be a member of lot of group but group can not contain other groups.

To know the userid and groupid of any user, we use "id" command.

```
$ id
uid = 500 (rajender) gid = 501 (rajender
$  groups = 501 (rajender)

$ echo $UID
 500
$
```

② **Real and Effective ID's** → Every process in Linux has two user ID's that is Real user ID and effective user ID. Also an effective group ID and real group ID. Most of the time, kernel checks only effective user ID.

For example, when a process tries to open a file the kernel checks the effective user ID and

UNIT-4 ULP

effective group ID. Corresponding "getuid" and "getid" functions return the real user ID and real group ID. If we want to change the effective user ID of an already running process, kernel looks the real as well as effective user ID.

(3) **Sticky bit** → In addition to read, write and execute permission, here is a magic bit called the sticky bit. This bit applies only to directories. A directory that has a sticky bit set allows us to delete a file only if we are the owner of the file. Normally we can delete a file only if we have write access to directory that contains it, even if we are not the owner of that file.

A few directories in Linux system have sticky bit set. For example /tmp directory in which user can place the temporary files has sticky bit set.

$ mkdir sticky
filename

```
$ echo "helloworld" > file1.txt &&
    mv file1.txt sticky/

$ ls -ld sticky
drwxrwxr-x. 2 rajender rajender 4096 Nov 9
                                21:55 sticky

$ su
Password:
# chmod +t sticky
# ls -ld sticky
drwxrwxr-t. 2 rajender rajender 4096 Nov 9
                                21:55 sticky
```

t → sticky

④ Restricted Shell → The standard shell in Unix allows the user to move around the file system and execute commands. To restrict the activities of a user, a special form of shell is present in almost all version of unix called restricted shell or rsh.

→ User cannot change any directory and forced to work under his home or HOME directory only.

→ This is for new user and privileges is cutting due to security reasons.

```
# bash -r
# pwd > file1.txt
    cannot redirect output

# cd
bash: cd: restricted
```

UNIT-4 ULP

Privileges of A System Administrator →

The system admin has terminator powers. There are several commands that are reserved for the exclusive use. There are several commands that behave differently.

For example ordinary user use date command for just for date of system but system admin uses date command to set system time and date.

The system admin also responsible for performing following tasks :→

① For maintaining existing user accounts.
② For installation and management of devices.
③ Start and Shutdown the system.
④ Change the System Configuration.

① Managing Users :→

Linux is Multiuser OS. The admin manages the account of each user. Each user uses this account to store files.

In addition to username and password, the system admin also maintains - User ID, group ID, Fullname, Password, Login shell and Home directory.

You can view the contents of this file by typing the command

```
$ cat /etc/passwd
```

- **Adding New Users:→** It is the responsibility of system admin to add new users in the users database and provide username and password to new users.

```
# adduser username
# adduser rajender
```

- **Setting Password :→** After creating the User, the next step is to set the password for the user by using passwd command. The syntax is

```
# passwd username
```

For example

```
# passwd rajender
Enter Password:
Confirm Password:
#
```

- **Deleting User →** If the users and their files are no longer required, then user can be removed by using the userdel command like

```
# userdel -r rajender
```

(rajender: username)

(2) Managing groups :→ A group is a logical collection of users. Users with similar needs or characteristics are usually placed into groups. A group is a collection of user account that can be given special permissions.

/etc/group

The /etc/group file maintains a list of the current groups for the system.

- The default group → Every user is member of atleast one group sometimes referred to as default group. The group ID specified in /etc/passwd file.

- The Other groups → A user can infact be a member of several groups. Any extra groups the user is a member are specified in the /etc/group file.

- **Creating a group** → To create a new group, use the groupadd command:

```
# groupadd group
```

where group specifies the name of the group to be added. Groups are stored in file /etc/group, which can be read by any user but modified by only by root.

```
# groupadd programmers
```

- **Deleting a group** → To delete a group, use the groupdel command as:

```
# groupdel group
```

For example

```
# groupdel programmers
```

- **Adding a member to a group** → To add a member to a group, you use a special form of the adduser command as: adduser user group

```
# adduser user group
```

For example:

```
# adduser manisha programmers
```

UNIT-4 ULP

Authentication :→ The 'Su' program lets us become root only when we know the root password. The program lets us prove that we are entitled to become root before going ahead with its actions. The process is called as authentication. The 'Su' program is checking those that if we are authentic. While administrating a very secure system, we donot want people to login first by typing an ordinary passwords. In really secure facility, retinal scam or other types of biometric testing are used. While writing a program, we should allow system administrator to use whatever means of authentication is appropriate for that installation. GNU / LINUX comes with a library that makes it very simple. This fouling is called Pluggable Authentication Module (PAM).

PAM makes it easy to write applic-ations that authenticate their users as the system administrater feels easy.

When a program is typed in the terminal, its password will not actually appeared as we typed in. If the hacker tries to use wrong password, then the PAM library will correctly indicate failure:

```
$ ./pam
password: wrongguess
Authentications failed
```

If valid password for current user is "password". Then we get

```
$ ./pam
password: password
Authentication OK.
```

Password Authentication: Passwords are nothing but the keys by using which a user can enter into the system. In Unix, password security is managed with help of two major/special files namely, /etc/password and /etc/shadow files.

1). **/etc/password file:** Complete information about user is obtained & stored in separate files called /etc/passwd file. Every user has a line pertaining to him in file. Every line is separated by : General format for each line is shown-

user: password : UID : GID : comment : home : shell

→ user: field contain login name of user

→ password: field used to store password.
'*' in this field indicates that user can't login with this user name.
'x' indicates that passwords now stored in separate file.

→ UID: field contains numeral userid.

→ GID: field contains default group ID of user. A user may be member of many groups.

→ comment: includes comments and also called GECOS field.

→ home: field contain absolute path name of user's home directory.

→ shell: This is the shell in which users enter as soon as user login.

Consider a sample line from the /etc/password file:

ishan: X : 116 : 360: ishan java : /home/ishan : /bin/sh

2). /etc/shadow file:- This field holds the user's password in an encrypted form. It hold additional information about the passwords like password aging & many more. Every line has 8 fields. Format is

user: coded password: last change : mindays : maxdays : wandays: disablein : expiredate

→ user: hold login name of user.

→ coded password: contain encrypted password.

→ last change: holds the date of last password change.

→ mindays: indicates the minimum number of days a new password must be kept before it can be changed again.

→ maxdays: indicate the maximum number of days after which a user must change password

→ warndays: contains information about the number of days before password expiration date that warning is given to user.

→ Disablein: holds the number of days of expiration that the agreement will be disabled.

→ Expire date: date on which account of user will be disabled.

Archiving :→ An achieve is a single file that contains a collection of other files and directories. Thus the process of combining a large number of files into a single file is called archiving. Archieves are usually used to make backup copy of collection of files and directories. Single file can be easily compressed by using commands such as gzip, compress and many more. The process of restoring the required files from archieve is called extracting files. Different tools used are tar tool, CPIO and dump and restore.

① Tar tool :→ Tar stands for 'tape archieve program'. This tool was designed especially for maintaining an archieve of files on magnetic tape. These are sometimes called "tar files", "tar archieves" because all the archieved files are rolled into a single tar ball. Two common options used are '-f' and '-v'. To specify name of archieve file we use '-f' followed by the filename, use of '-v' (verbose) option to have tar output names of files as they are processed.

(i) Creating archieve: To create an archieve with tar, we use 'C' (create) option and specify the name of archieve file to create with '-f' option. It is common to use name with a '.tar' extension as 'backup.tar'. For example, to create archieve called 'project.tar' from contents of 'project' directory, as we type:

```
$ tar -cvf project.tar project
```

(ii) Listing the contents of an archieve: To list the contents of tar archieve without extracting them, we use tar with '-t' option. For example, to list the contents of archieve called 'project.tar' we use:

```
$ tar -tvf project.tar
```

(iii) Extracting files from archieve: To extract the contents of tar archieve, we use tar with '-X' (extract) option. For example, to extract contents of an archieve called 'projection', we use:

```
$ tar -xvf project.tar
```

To extract combatants of compressed devices i.e. with '.tar.gz' or '.tgz' extension, we include '-Z' option. For

example,

$ tar -zxvf project.tar.gz

② CPIO tool :- CPIO stands for 'copy in and copy out'. This command can be used to copy files from the system to the back up device as well as copy files from a backup device back to the system. This command doesnot work with file names directory and is used with redirection and piping. It works with following three modes:

(a) Output mode (CPIO - O)

(b) Input mode (CPIO - I)

(c) Pass through mode (CPIO - P)

The output mode option used to take backup of required files. The process of taking backups in this way is known as creating an archieve. For example,

```
# ls | cpio -O /dev/rdsk/foq
    ishan.txt
    student.list
    calendar
    160 blocks
#
```

In this, all files in current directory have been piped to CPIO command using 'ls' command onto 1.44 MB floppy. Also, all files which needs to be backedup can be put into separate file by writing their

names as

UNIT-4 ULP

cpio -O > /dev/rdsh/foq < flist

We can also backup files by using the output of find command and piping it to CPIO command. First, the files that need to be backed up are found using find command & then are backed up as shown:

find -typef -mtime -1 -brw | CPIO -OV > /dev/rdish/feq

Wild cards can be used to select required files. For example, to extract all shell files, we have following commands:

cpio -iv "*.sh" </dev/rdsh/foq

For restoring all files except certain particular files using -f option as shown:

CPIO -ivf "*.C" </dev/rdsk/foq

This command restores all files except c files.

'-p' option is same as that of '-O' option. But with this option, no archieves are made. It is used to copy each file individually to another directory in

unix file system tree. This mode is used with two options: '-v' for verbose and '-d' for creating new directory. For example,

```
# ls | CPIO -pvd /tmp
```

This command copies all files in current directory to /tmp directory.

③ Dump and Restore tool :- This tool allow image backup as well as multiple volume backups. Dump is used to perform backups & restore is used to return information from backups. Command line format for dump is

dump [options] [arguments] filenames

Dump works on the concept of levels generally from 0 to 9. A dump level of 0 means that all files will be backed up. Dump level 1 ... 9 means that all files that last changed since the last dump of lower level will be backed up. Version arguments used are :-

Arguments	Purpose
u	update the dump received
v	after writing each volume rewind the tape and verify

Purpose of restore command is to extract files archived using dump command. It also provide ability to extract single individual files, directories & their contents & even file system. Syntax is:

Arguments	Purpose
i —	Intractive, directory information read from tape after which we can browse directory & select files to be extracted.
r —	restore the entire tape should only be used to restore entire file system.
t —	table of contents, if no filename provided, root directory listed
x —	extract named files. If directory is specified, it and its all subdirectories are extracted

Signals and Signals handlers :→

Signal are mechanisms for communicating with and manipulating process in Linux. It is a special message to a process. Signals are asynchronous in that when a process receives a signal, it processes the signal immedietely without finishing the current function or even the current line of code.

There are several different signals each with a different meaning. Each signal type is specified by a signal number but in program, we usually refer to signal by their names.

In Linux, signals are defined in file

/usr/ensued/bits/signum.h

we use the header file namely

<signal.h> in our program.

A Process may also send a signal to another process by sending it a SIGTERM or SIGKILL.

UNIT-4 ULP

The SIGNUP signal is used to wake up an idle program or to cause a program to reread its configuration files.

A Signal handler performs the minimum work necessary to respond to the signal and give returns control to the main program. This consists of recording of facts and signal occured.

Signals are identified by integers. A list of exit or interrupt signals is given below:

Signal Number	Name	Factor
1	SIGHUP	Hurry up, Close Process Comm^{n} links
2	SIGNT	Interrupt, tell Process to exit (Ctrl+C)
3	SIGQUIT	Quit, forces the Process to Quit.
9	SIGKILL	Signal Kill
15	SIGTERM	Software terminator
24	SIGSTOP	Stop (Ctrl+Z)

LINUX I/O SYSTEM → In Linux, there are 02 ways to handle Input/output of the System.

① First is the stream I/O which involves standard library function such as printf, fopen.

② Second is system call I/O. These calls include functions such as open(), read(), write().

① Reading and writing data Using System Call I/O →

With the linux low level I/O operations, we use a filehandle instead of a file pointer. A file descriptor is an integer value that refers to a particular instance of an open file in a single process. It can be open for reading, writing and for both reading and writing.

- **Opening a File →** To open a file and file descriptor that can access that file, uses the open call. It take Pathname of the file as an argument to open as a character string and flags specify

how to open it. we can use open to create a new file, pass a third argument that specifies the access permissions to be set for that file.

- **Closing a file →** we close a file with the close. Sometimes it is not necessary to close the file explicitly because Linux closes all file descriptors automatically where a processes terminates. Closing a file descriptor may cause Linux to take a particular action depends on the nature of file descriptor. For example, when we close a File descriptor for a Network socket, Linux closes the network connection between the two computers communicating through socket.

- **Writing Data →** To write data to a file descriptor, we use the 'write' call. This file descriptor is provided a pointer to a buffer of data and the number of bytes to write. The file descriptor must be opened for writing. The data written to the file need not be character strings.

- **Reading Data →** The corresponding call for reading is 'read'. Like 'write', it takes a file descriptor, a pointer to a buffer and a count. The count specifies how many bytes are read from the file descriptor into the buffer. The call to read returns the error or the number of bytes actually read;

(2) **Stream I/O →** Streams are a portable way of reading and writing data. They provide a flexible and efficient means of input/output. A stream is a file or physical device such as Printer or monitor which is manipulated with a pointer to the stream.

There exists an internal C data structure namely FILE which represents all streams and is declared in <stdio.h>. We just need to declare a pointer or variable of this type in our program.

There are 03 operations which must be performed while performing input/output on streams:

1. Open the stream
2. Access the stream
3. Close the stream

Networking Tools :→ These are network device managment program, that helps a network administator to keep track of moves, additions and changes (known as MACs) to the hardware infrastucture of a network.

① PING :→ The Ping Command (named after the sound of an active Sonar system) sends eacho requests to the host you specify on the command line.

PING (Packet INternet Gropher) Command is the best way to test connectivity between two host or two nodes. Whether it is LAN or WAN. Ping use ICMP (Internet Control Message Protocol) to communicate to other devices.

You can ping host name of ip address using below command.

```
$ ping google.com
PING google.com (216.58.198.78): 56 data bytes
64 bytes from 216.58.198.78: icmp_seq=0 ttl=46 time=6.108 ms
```

② Telnet :→

③ Route → The route command is the tool used to display or modify the routing table. To add a gateway as default you would type:

```
$ route -n
Kernel IP routing table
```

Destination	Gateway	Genmask	Flag	Metric	Ref	Use	Iface
0.0.0.0	172.31.16.1	0.0.0.0	UG	0	0		eth0
172.17.0.0	0.0.0.0	255.255.0.0	U	0	0		docker0
172.31.16.0	0.0.0.0	255.255.240.0	U	0	0		eth0

④ FTP → FTP command is used to connect to Ftp server, upand download files and create directories.

Step 1: Establishing and FTP connection

Examples: ftp domain.com
ftp 192.168.0.1
ftp user@ftpdomain.com

Step 2: Login with User and Password

Most FTP servers logins are password protected, so the server will ask us for a 'username' and 'password'.

Step 3: Working with Directories

The Command to list, move and create folder on an FTP server and almost the same as we would use the shell locally on our computer. ls stands for list, cd for change directories, mkdir to create directories

ftp > ls
The server will return

```
200 PORT Command successful.
Consider using PASV.
150 Here comes the directory listing
....
.....
```

ftp > Cd directory

The Server will return

```
250 Directory successfully changed
```

Step 4: Downloading files with FTP

Before downloading a file, we should set the local FTP file download directory by using 'lcd' Command.

```
lcd
/home/user/ yourdirectoryname
```

Now, we can use the command 'get' command to download a file as:

```
get file
```

Step 5: Uploading Files with FTP

we can upload files that are in the local directory where we made the FTP connection.

To upload files, we can use 'put' command as.

```
put file
  or
put /path/file        ← when the file not in local directory
```

To upload several files we can use the 'mput' command similar to 'mget' command

```
mput *.xls
```

Step 6: Closing the FTP connection

once we have done the FTP work, we should close the connection for security reasons. There are three commands that we can use to close the connection as:

```
bye

exit

quit
```

Any of them will disconnect our PC from the FTP server and will return:

```
221 Goodbye
```