MODULE-1

Topics

Introduction: Unix Components/Architecture. Features of Unix. The UNIX Environment and UNIX Structure, Posix and Single Unix specification. General features of Unix commands/ command structure. Command arguments and options. Basic Unix commands such as echo, printf, ls, who, date, passwd, cal, Combining commands. Meaning of Internal and external commands. The type command: knowing the type of a command and locating it. The root login. Becoming the super user: su command.

Unix files: Naming files. Basic file types/categories. Organization of files. Hidden files. Standard directories. Parent child relationship. The home directory and the HOME variable. Reaching required files- the PATH variable, manipulating the PATH, Relative and absolute pathnames. Directory commands – pwd, cd, mkdir, rmdir commands. The dot (.) and double dots (..) notations to represent present and parent directories and their usage in relative path names. File related commands – cat, mv, rm, cp, wc and od commands.

CHAPTER 1 INTRODUCTION

1.1 WHAT IS AN OPERATING SYSTEM (OS)?

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

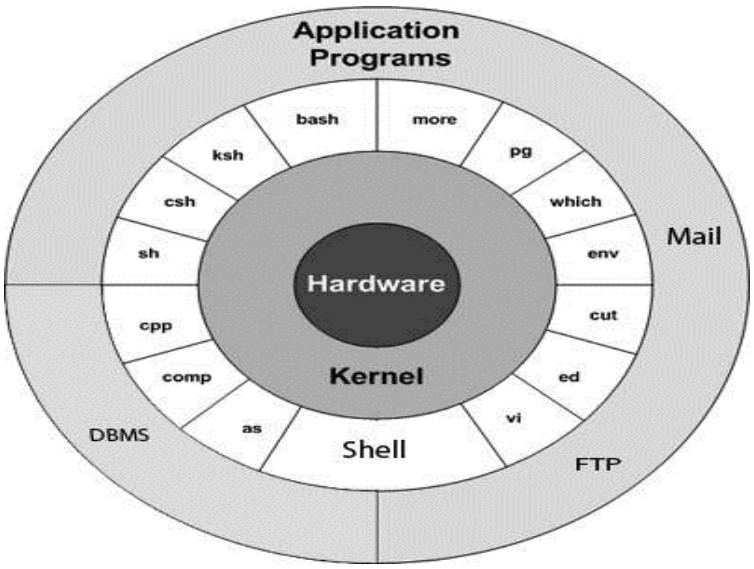
For example, an OS ensures safe access to a printer by allowing only one application program to send data directly to the printer at any one time. An OS encourages efficient use of the CPU by suspending programs that are waiting for I/O operations to complete to make way for programs that can use the CPU more productively. An OS also provides convenient abstractions (such as files rather than disk locations) which isolate application programmers and users from the details of the underlying hardware. UNIX Operating system allows complex tasks to be performed with a few keystrokes. It doesn't tell or warn the user about the consequences of the command.

1.2 BRIEF HISTORY

- In the late 1960s, researchers from General Electric, MIT and Bell Labs launched a joint project to develop an ambitious multi-user, multi-tasking OS for mainframe computers known as MULTICS (Multiplexed Information and Computing System). MULTICS failed, but it did inspire Ken Thompson, who was a researcher at Bell Labs, to have a go at writing a simpler operating system himself.
- He wrote a simpler version of MULTICS on a PDP7 in assembler and called his attempt UNICS (Uniplexed Information and Computing System). Because memory and CPU power were at a premium in those days, UNICS (eventually shortened to UNIX) used short commands to minimize the space needed to store them and the time needed to decode them - hence the tradition of short.
- The limitation of UNICS was not portable. In order to overcome the limitation, Ken Thompson started to work on the development of system using higher level language called B Language. As B language did not yield expected results, Dennis Ritchie developed higher level language called C.

- Ken Thompson then teamed up with Dennis Ritchie, the author of the first C compiler in 1973.
- They rewrote the UNIX kernel in C - this was a big step forwards in terms of the system's portability - and released the fifth Edition of UNIX to universities in 1974.
- The Seventh Edition, released in 1978, marked a split in UNIX development into two main branches: SYSV (System 5) and BSD (Berkeley Software Distribution).
- SYSV was developed by AT&T and other commercial companies. UNIX flavors based on SYSV have traditionally been more conservative, but better supported than BSD-based flavors.
- Until recently, UNIX standards were nearly as numerous as its variants. In early days, AT&T published a document called System V Interface Definition (SVID). X/OPEN (now The Open Group), a consortium of vendors and users, had one too, in the X/Open Portability Guide (XPG).
- In the US, yet another set of standards, named Portable Operating System Interface for Computer Environments (POSIX), were developed at the Institution of Electrical and Electronics Engineers (IEEE).
- In 1998, X/OPEN and IEEE undertook an ambitious program of unifying the two standards. In 2001, this joint initiative resulted in a single specification called the Single UNIX Specification, Version 3 (SUSV3), that is also known as IEEE 1003.1:2001 (POSIX.1). In 2002, the International Organization for Standardization (ISO) approved SUSV3 and IEEE 1003.1:2001**.(**POSIX 2)

1.3 UNIX COMPONENTS / ARCHITECTURE



- **Kernel:** is the core of operating system. A collection of routines mostly written in C. It is loaded into memory when the system is booted and communicates directly with the hardware. User programs that need to access the hardware use the services of the kernel which performs the job on the user's behalf. These programs access the kernel through a set of functions called system calls. The kernel manages system memory, processes, decides priorities.
- **Shell**: interface between Kernel and User. It functions as **command interpreter** i,e it receives and interprets the command from user and interacts with the hardware. There is only one

kernel running on the system, there could be several shells in action - one for each user who is logged in. When user enters a command, shell thoroughly examines the keyboard input and simplifies to a command line, and communicates with kernel to see that the command is executed.

Eg. $echo Sun Solaris

Sun Solaris //ignores all spaces in the above command line.

- **Files and Process**: File is an array of bytes and it contain virtually anything. Unix considers even the directories and devices as members of file system. The dominant file type is text and behavior of system is mainly controlled by text files.
- The second entity is the process, which is the name given to a file when it is executed as a program. Process is simply a "time image" of an executable file.
- **System Calls**: Though there are thousands of commands in the Unix system, they all use a handful of functions called system calls. User programs that need to access the hardware use the services of the kernel, which performs the job on user's behalf. These programs access the kernel through a set of functions called system calls.
- Ex: open()-- system call to access both file and device. Write()—system call to write a file.

1.4 FEATURES OF UNIX

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

- **Portable:** UNIX can be installed on many hardware platforms. Its widespread use can be traced to the decision to develop it using the C language. Because C programs are easily moved from one hardware environment to another, it is relatively simple to port it to different environments.

- **A Multiuser & Multitasking system:** It is a multiprogramming system. It permits multiple programs to run & compete for the attention of the CPU. This can happen in two ways:
  - **Multiuser system**: Multiple users can run separate job. In unix, the resources are actually shared between all users. The computer breaks up a unit n terminate them time expires, the previous job is kept in waiting, & the next user's job is.
  - **Multitasking system**: A single user can run multiple job. It's usual for a user to edit a file, print another file, send email etc. without leaving any applications. The kernel is designed to handle a user's multiple needs. In this environment, a user sees one job running in the foreground, & the rest running at the background. We can switch jobs between background & foreground, suspend.

- **The Building-block approach:** Unix comes with hundreds of simple commands which perform one simple job. It's through pipes & filters unix implement small-is-beautiful philosophy. Using this feature, the output of one tool can be used as input of another tool.
  Ex: $ls | wc. Here the output of ls command is given as input to another command wc.

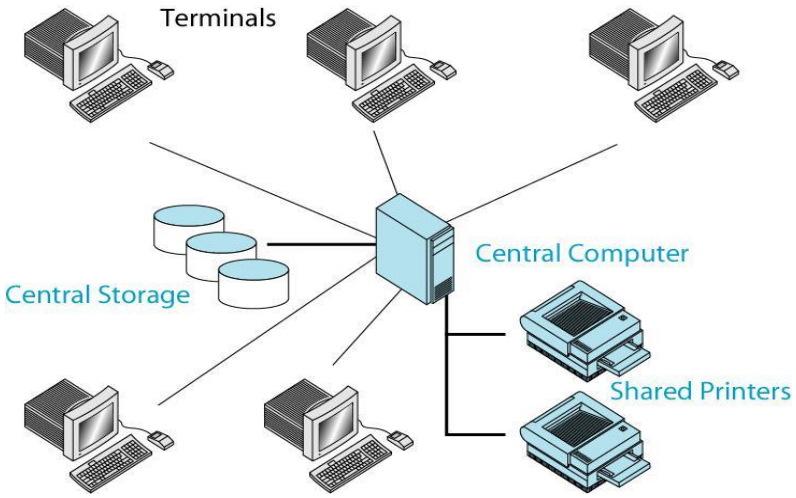
- **Unix toolkit:** Unix supplied with a set of applications such as compilers, interpreters, text-manipulation utilities, general purpose tools, & system administration tools. This set is constantly changing with every Unix release. The shell & utilities form part of POSIX specification. We must be able to download these tools & configure them to run on our machine.

- **Pattern Matching:** Unix has very strong pattern matching features. The * (known as metacharacter) is a special character used by the system to indicate that it can match a number of filenames. Ex: $ls *.c There are many more such special symbols in unix system. Some of the most advanced and useful tools use a special expression called regular expression that is framed with characters from this set.
- **Programming facility:** The Unix shell is a programming language. It has all the necessary features of a language like control structures, loops & variables. This makes it a powerful programming language. These features are used to design shell scripts, the programs that can also invoke the unix commands.
- **Documentation:** The principal online help facility is available is the man command in unix, which remains the most important reference for commands and their configuration files.

1.5 THE UNIX ENVIRONMENTS AND UNIX STRUCTURE

There 3 different environments in UNIX

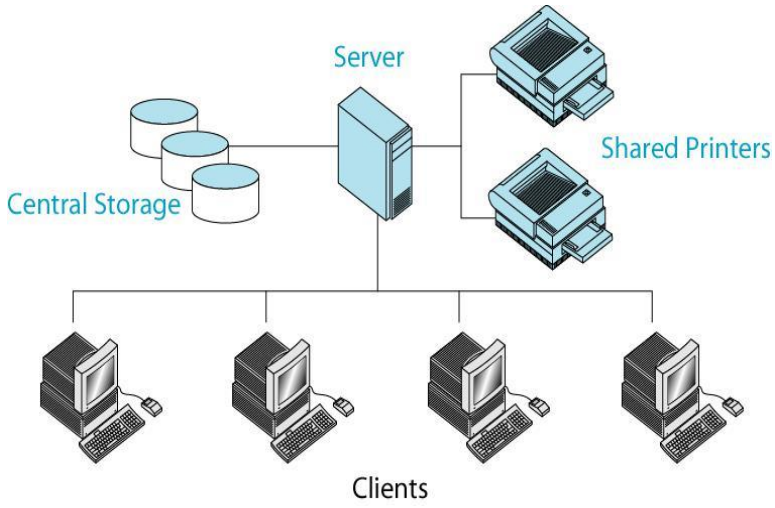
- Personal environment
- Timesharing environment: Many users connected to one computer
- Client/server environment Computing split between a central computer (server) and users' computers (clients)



- **Personal environment:** Originally Unix designed as a multiuser environment, many users are installed UNIX on their personal computers this tends to personal Unix system environment.
- **Timesharing environment:** In time-sharing environment, many users are connected to one or more computers. The output devices (such as printers) and auxiliary storage devices (such as disks) are shared by all users. In this environment all of the computing must be done by the central computer. The central computer has several responsibilities
  - It must control the shared resources.
  - It must manage the shared data
  - It must manage the printing data and resource.
  - It must handle the computing all task

All of this work ten ds to keep the central computer busy, so user has to wait more time for get done their work so, it is non-productive because of slow response.

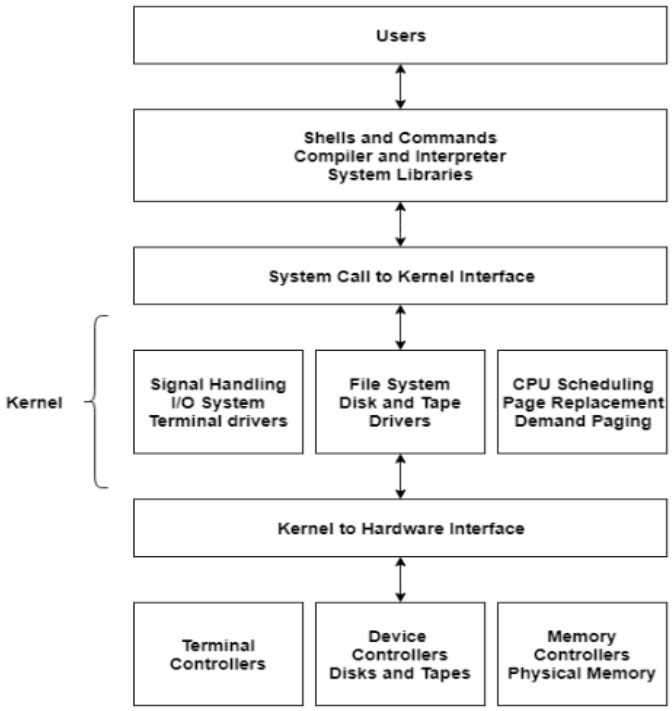
- **The Client/Server Environment**



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

UNIX STRUCTURE

Unix consists of four major components: the kernel, the shell, a standard set of utilities and application programs. The below fig shows the Unix Structure.



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

- **Shell**: is an interface between Kernel and User. It functions as **command interpreter** i,e it receives and interprets the command from user and interacts with the hardware. There is only one kernel running on the system, there could be several shells in action- one for each user who is logged in. Shell has two major parts.
  a. Interpreter: reads user commands and works with the kernel to execute them.
  b. Shell Programming: is a programming capability that allows user to write a shell script.

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

There are standard shells used in UNIX today.

1. The Bourne Shell

The Bourne shell (sh), written by Steve Bourne at AT&T Bell Labs, is the original UNIX shell. It is the preferred shell for shell programming because of its compactness and speed. A Bourne shell drawback is that it lacks features for interactive use, such as the ability to recall previous commands (history). The Bourne shell also lacks built-in arithmetic and logical expression handling.

The Bourne shell is the Solaris OS default shell. It is the standard shell for Solaris system administration scripts. For the Bourne shell the:

- Command full-path name is /bin/sh and /sbin/sh.
- Non-root user default prompt is $.
- Root user default prompt is #.

2. The C Shell

- Is a UNIX enhancement written by Bill Joy at the University of California at Berkeley.
- Incorporated features for interactive use, such as aliases and command history.
- Includes convenient programming features, such as built-in arithmetic and a C-like expression syntax.

For the C shell the:

- Command full-path name is /bin/csh.
- Non-root user default prompt is hostname %.
- Root user default prompt is hostname #.

3. The Korn Shell(ksh)

- Was written by David Korn at AT&T Bell Labs
- Is a superset of the Bourne shell.
- Supports everything in the Bourne shell.
- Has interactive features comparable to those in the C shell.
- Includes convenient programming features like built-in arithmetic and C-like arrays, functions, and string-manipulation facilities.
- Is faster than the C shell.
- Runs scripts written for the Bourne shell.

For the Korn shell the:

- Command full-path name is /bin/ksh.
- Non-root user default prompt is $.
- Root user default prompt is #.

4. The GNU Bourne-Again Shell(bash)

- Is compatible to the Bourne shell.
- Incorporates useful features from the Korn and C shells.
- Has arrow keys that are automatically mapped for command recall and editing.

For the GNU Bourne-Again shell the:

- Command full-path name is /bin/bash.
- Default prompt for a non-root user is bash-x.xx$. (Where x.xx indicates the shell version number. For example, bash-3.50$)
- Root user default prompt is bash-x.xx#. (Where x.xx indicates the shell version number. For example, bash-3.50$#)

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

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

1.6 POSIX and The Single Unix Specification

- Unix fragmentation & the absence of a single standard adversely affected the development of portable applications. A group of standards , POSIX(Portable Operating System Interface for Computer Environments), were developed at IEEE. POSIX refers to operating systems in general, but was based on UNIX. Two of the most cited standards from the POSIX family are known as POSIX.1 & POSIX.2.
- POSIX.1 specifies the C application program interface – the system calls. POSIX.2 deals with the shell & utilities.
- In 2001, a joint initiative of X/Open & IEEE resulted in the Combining of two standards. This is the Single Unix specification, version 3(SUSV3). The "write once, adopt everywhere" approach to this development means that once software has been developed on any POSIX-compliant Unix system, it can be ported to another POSIX-compliant Unix machine with minimum modifications.

Some of the commercial UNIX based on system V are:

- IBM's AIX
- Hewlett-Packard's HPUX
- SCO's Open Server Release 5
- DEC's Digital UNIX
- Sun Microsystems' Solaris 2

Some of the commercial UNIX based on BSD are:

- SunOS 4.1.X (now Solaris)
- DEC's Ultris
- BSD/OS, 4.4BSD

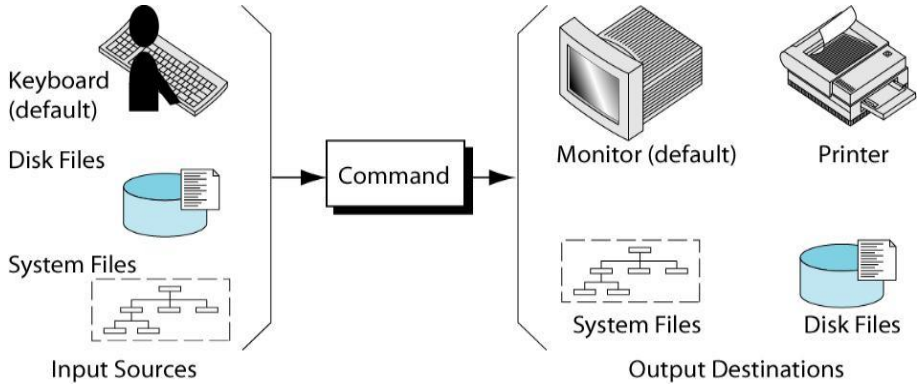
Some Free UNIX are:

- Linux, written by Linus Torvalds at University of Helsinki in Finland.
- sFreeBSD and NetBSD, a derivative of 4.4BSD

1.7 GENERAL FEATURES OF UNIX COMMANDS/ COMMAND STRUCTURE

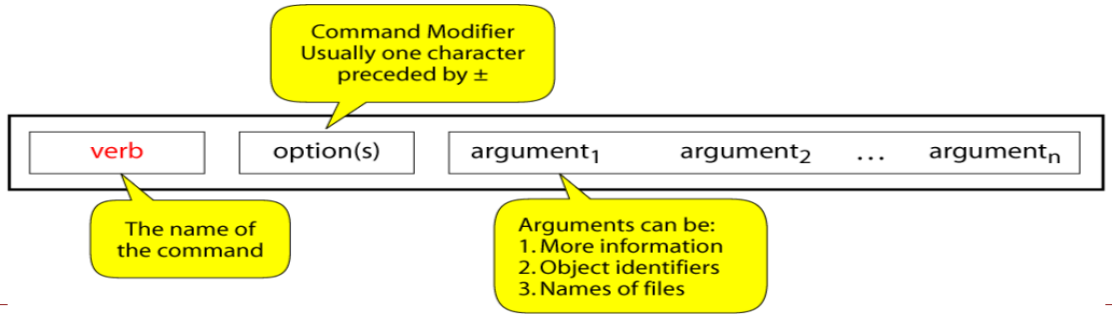
Commands are entered at shell prompt. The components of the command line are:

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



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

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



- **Verb**: is the command name. The command indicates what action is to be taken. This action concept gives us the name verb for action.
- **Option:** modifies how the action is applied.
- **Argument**: provides additional information to the command.

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

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

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

Options

An option is preceded by a minus sign (-) to distinguish it from filenames.

Example: **$ ls –l**

There must not be any whitespaces between – and l. Options are also arguments, but given a special name because they are predetermined. Multiple options can be given with only one – sign. i.e., instead of using

$ ls –l –a –t

we can as well use,

$ ls –lat

Because UNIX was developed by people who had their own ideas as to what options should look like, there will be variations in the options. Some commands use + as an option prefix instead of -.

Filename arguments:

- Many unix commands use a filename as arguments so the command can take input from the file. If a command uses a filename as argument at all, it will generally be its last argument and after all options.
- Its also quite common to see many commands working with multiple filenames as arguments. ls -lat chap01, chap02, chap03
- The command with its arguments and options is known as the command line.
- This line can be considered complete only after the user hit[Enter]. The complete line is then fed to the shell as its input for interpretation and execution.

1.8 BASIC UNIX COMMANDS SUCH AS echo, printf, ls, who, date, passwd, cal, combining commands.

➢ **echo: Displaying a message**

- The **echo** command used to display a message.
- To display the diagnostic messages on the terminal or to issue prompts for taking user input (like echo Sun Solaris).
- To evaluate shell variable (like echo $SHELL)
  General Syntax: $**echo [Short-Option]... [String]...**
- Orginally, echo was an external command, but nowadays all shells have echo built-in.
- Most of the echo's behavior differences relates to the way echo's interprets certain strings known as escape sequences.
- An escape sequence is generally a two character – string beginning with \(backslash).

Options

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

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

\\A	Literal backslash character ("\")
\a	An alert (The BELL character)
\b	Backspace

\c	Produce no further output after this
\e	The escape character; equivalent to pressing the escape key
\f	A form feed
\n	A newline
\r	A carriage return
\t	A horizontal tab
\v	A vertical tab

Execution of commands

```
chandana@chandana-VirtualBox: ~
chandana@chandana-VirtualBox:~$ echo $SHELL
/bin/bash
chandana@chandana-VirtualBox:~$ echo "Welcome to RNSIT"
Welcome to RNSIT
chandana@chandana-VirtualBox:~$ echo "Welcome \bto \bRNSIT"
Welcome \bto \bRNSIT
chandana@chandana-VirtualBox:~$ echo -e "Welcome \bto \bRNSIT"
WelcometoRNSIT
chandana@chandana-VirtualBox:~$ echo -e "Welcome \cto \cRNSIT"
Welcome chandana@chandana-VirtualBox:~$ echo -e "Welcome \nto \nRNSIT"
Welcome
to
RNSIT
chandana@chandana-VirtualBox:~$ echo -e "Welcome \tto \tRNSIT"
Welcome         to      RNSIT
chandana@chandana-VirtualBox:~$ echo -e "Welcome \rto RNSIT"
to RNSIT
chandana@chandana-VirtualBox:~$ echo -e "Welcome \vto \vRNSIT"
Welcome
        to
           RNSIT
chandana@chandana-VirtualBox:~$ echo -e "\aWelcome to RNSIT"
Welcome to RNSIT
chandana@chandana-VirtualBox:~$ echo -n "Welcome to RNSIT"
Welcome to RNSITchandana@chandana-VirtualBox:~$ echo *
combine.txt Desktop Documents Downloads Music myfolder out.txt Pictures Public snap Templates UNIX1 Videos
chandana@chandana-VirtualBox:~$
```

- **Printf: An alternative to echo command**
  - Like echo, it exists as an external command, its only the Bash shell that has printf built-in.
  - The command can be used as $printf "No file entered"
    Output: No file entered
  - Printf also accepts all escape sequences used by echo, but it doesn't automatically insert a newline unless the \n is used explicitly.

- While printf can do everything that echo does, some of its format strings can convert data from one form to another.

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

Execution of commands

```
chandana@chandana-VirtualBox: ~
chandana@chandana-VirtualBox:~$ printf "%s\n" "Welcomt to RNSIT\n"
Welcomt to RNSIT\n
chandana@chandana-VirtualBox:~$ printf "%b\n" "Welcomt to RNSIT\n" "From ISE\n"
Welcomt to RNSIT

From ISE

chandana@chandana-VirtualBox:~$ printf "%30b\n" "Welcomt to RNSIT\n" "From ISE\n"
             Welcomt to RNSIT

                    From ISE

chandana@chandana-VirtualBox:~$ printf "%d\n" "220" "420"
220
420
chandana@chandana-VirtualBox:~$ printf "%f\n" "220" "420.25"
220.000000
420.250000
chandana@chandana-VirtualBox:~$ printf "The value of 255 is %o in octal and %x in hexadecimal\n" 255 255
The value of 255 is 377 in octal and ff in hexadecimal
chandana@chandana-VirtualBox:~$ printf "%08x\n" "1024"
00000400
chandana@chandana-VirtualBox:~$
```

In the first line printf command \n will print, no special meaning is there for \n when it is used with %s.

➢ **ls: Listing Files**

- The files are organized in separate folders called directories. We can list the names of the files available in this directory with the ls command. Following is the ls command without any option:

  $ ls
  Chap01 Chap02 Pgm.c F1.doc

- Unix has a special symbol * to access the files with same pattern $ ls Chap*

  Chap01 Chap02

Execution of commands

```
chandana@chandana-VirtualBox:~$ ls
combine.txt  Desktop  Documents  Downloads  Music  myfolder  out.txt  Pictures  Public  snap  Templates  UNIX1  Videos
chandana@chandana-VirtualBox:~$ ls -l
total 52
-rw-rw-r-- 1 chandana chandana   85 Sep 10 10:05 combine.txt
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Desktop
drwxr-xr-x 3 chandana chandana 4096 Sep  4 10:18 Documents
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Downloads
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Music
drwxrwxr-x 3 chandana chandana 4096 Sep 10 10:11 myfolder
-rw-rw-r-- 1 chandana chandana  274 Sep 10 10:08 out.txt
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Pictures
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Public
drwxr-xr-x 3 chandana chandana 4096 Sep 14 10:31 snap
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Templates
drwxrwxr-x 2 chandana chandana 4096 Sep  9 19:12 UNIX1
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Videos
```

$ls – lists directory contents of files and directories.
$ls -l – Display all information about files/directories contents.
Here is the information about all the listed columns:

1. First Column: represents file type and permission given on the file. Above is the description of all type of files.
2. Second Column: represents the number of memory blocks taken by the file or directory.
3. Third Column: represents owner of the file. This is the Unix user who created this file.
4. Fourth Column: represents group of the owner. Every Unix user would have an associated group.
5. Fifth Column: represents file size in bytes.
6. Sixth Column: represents date and time when this file was created.
7. Seventh Column: represents filename.

In the ls -l listing example, every file line began with a d, -, or l. These characters indicate the type of file that's listed.

```
chandana@chandana-VirtualBox:~$ ls -lh
total 52K
-rw-rw-r-- 1 chandana chandana   85 Sep 10 10:05 combine.txt
drwxr-xr-x 2 chandana chandana 4.0K Sep  2 19:58 Desktop
drwxr-xr-x 3 chandana chandana 4.0K Sep  4 10:18 Documents
drwxr-xr-x 2 chandana chandana 4.0K Sep  2 19:58 Downloads
drwxr-xr-x 2 chandana chandana 4.0K Sep  2 19:58 Music
drwxrwxr-x 3 chandana chandana 4.0K Sep 10 10:11 myfolder
-rw-rw-r-- 1 chandana chandana  274 Sep 10 10:08 out.txt
drwxr-xr-x 2 chandana chandana 4.0K Sep  2 19:58 Pictures
drwxr-xr-x 2 chandana chandana 4.0K Sep  2 19:58 Public
drwxr-xr-x 3 chandana chandana 4.0K Sep 14 10:31 snap
drwxr-xr-x 2 chandana chandana 4.0K Sep  2 19:58 Templates
drwxrwxr-x 2 chandana chandana 4.0K Sep  9 19:12 UNIX1
drwxr-xr-x 2 chandana chandana 4.0K Sep  2 19:58 Videos
chandana@chandana-VirtualBox:~$ ls -ld
drwxr-xr-x 19 chandana chandana 4096 Sep 14 14:20 .
```

$ls -lh – Displays file size in easy to read format. h represents human readable form.

```
chandana@chandana-VirtualBox:~$ ls -a
.               .cache       Downloads  myfolder  snap                      .vboxclient-clipboard.pid
..              combine.txt  .gnupg     out.txt   .ssh                      .vboxclient-display-svga-x11.pid
.bash_history   .config      .local     Pictures  .sudo_as_admin_successful  .vboxclient-draganddrop.pid
.bash_logout    Desktop      .mozilla   .profile  Templates                 .vboxclient-seamless.pid
.bashrc         Documents    Music      Public    UNIX1                     Videos
chandana@chandana-VirtualBox:~$ ls -A
.bash_history  combine.txt  Downloads  Music     .profile  .sudo_as_admin_successful  .vboxclient-display-svga-x11.pid
.bash_logout   .config      .gnupg     myfolder  Public    Templates                  .vboxclient-draganddrop.pid
.bashrc        Desktop      .local     out.txt   snap      UNIX1                      .vboxclient-seamless.pid
.cache         Documents    .mozilla   Pictures  .ssh      .vboxclient-clipboard.pid  Videos
chandana@chandana-VirtualBox:~$ ls -1
combine.txt
Desktop
Documents
Downloads
Music
myfolder
out.txt
Pictures
Public
snap
Templates
UNIX1
Videos
```

$ls -a – Displays all the hidden files including current and parent directory.
$ls -A – Displays all the hidden files excluding current and parent directory.
$ls -1 – Displays one file per line.

```
chandana@chandana-VirtualBox:~$ ls -lt
total 52
drwxr-xr-x 3 chandana chandana 4096 Sep 14 10:31 snap
drwxrwxr-x 3 chandana chandana 4096 Sep 10 10:11 myfolder
-rw-rw-r-- 1 chandana chandana  274 Sep 10 10:08 out.txt
-rw-rw-r-- 1 chandana chandana   85 Sep 10 10:05 combine.txt
drwxrwxr-x 2 chandana chandana 4096 Sep  9 19:12 UNIX1
drwxr-xr-x 3 chandana chandana 4096 Sep  4 10:18 Documents
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Desktop
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Downloads
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Music
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Pictures
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Public
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Templates
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Videos
chandana@chandana-VirtualBox:~$ ls -ltr
total 52
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Videos
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Templates
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Public
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Pictures
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Music
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Downloads
drwxr-xr-x 2 chandana chandana 4096 Sep  2 19:58 Desktop
drwxr-xr-x 3 chandana chandana 4096 Sep  4 10:18 Documents
drwxrwxr-x 2 chandana chandana 4096 Sep  9 19:12 UNIX1
-rw-rw-r-- 1 chandana chandana   85 Sep 10 10:05 combine.txt
-rw-rw-r-- 1 chandana chandana  274 Sep 10 10:08 out.txt
drwxrwxr-x 3 chandana chandana 4096 Sep 10 10:11 myfolder
```

$ls -lt – Displays all information about file/directories based on last modification time.
$ls -ltr – Displays all information about file/directories in the reverse order of last modification time.

```
chandana@chandana-VirtualBox:~$ ls -F
combine.txt  Documents/  Music/     out.txt    Public/  Templates/  Videos/
Desktop/     Downloads/  myfolder/  Pictures/  snap/    UNIX1/
chandana@chandana-VirtualBox:~$ ls --version
ls (GNU coreutils) 8.30
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Richard M. Stallman and David MacKenzie.
```

$ls -F – will add the '/' Character at the end each directory.
$ls –version – checks version of ls command.

- **Who: Displays the users who are currently logged into the computer.**
  - General Syntax: $ who [options] [filename]
  - who command is used to find out the following information:
    1. Time of last system boot
    2. Current run level of the system
    3. List of logged in users and more.

Options:

1. The who command displays the following information for each user currently logged in to the system if no option is provided:
   - Login name of the users
   - Terminal line numbers
   - Login time of the users in to system
   - Remote host name of the user
2. To display host name and user associated with standard input such as keyboard user needs to use who -m -H command.
3. To show all active processes which are spawned by INIT process who -p -H command will be used.
4. To show status of the users message as +, – or ?, who -T -H command need to be used.
5. To show list of users logged in to system, who -u is used.
6. To show time of the system when it booted last time, who -b -H command will be used.
7. To show details of all dead processes, who -d -H
8. To show system login process details, who -l -H
9. To display user identification information, $id is the command to be used.
10. To count number of users logged on to system, who -q -H to be used.
11. To display current run level of the system, who -r is the command.
12. To display all details of current logged in user. Who -a command is used.
13. To display system's username, whoami command is used.
14. To display list of users and their activities $w is used.

Execution of commands

```
chandana@chandana-VirtualBox:~$ who
chandana :0           2020-09-14 14:19 (:0)
chandana@chandana-VirtualBox:~$ who -m -H
NAME     LINE         TIME             COMMENT
chandana@chandana-VirtualBox:~$ who -p -H
NAME     LINE         TIME                   PID COMMENT
chandana@chandana-VirtualBox:~$ who -T -H
NAME       LINE         TIME             COMMENT
chandana ? :0          2020-09-14 14:19 (:0)
chandana@chandana-VirtualBox:~$ who -u
chandana :0           2020-09-14 14:19   ?          1379 (:0)
chandana@chandana-VirtualBox:~$ who -b -H
NAME     LINE         TIME                   PID COMMENT
         system boot  2020-09-14 14:18
chandana@chandana-VirtualBox:~$ who -d -H
NAME     LINE         TIME             IDLE          PID COMMENT  EXIT
chandana@chandana-VirtualBox:~$ who -l -H
NAME     LINE         TIME             IDLE          PID COMMENT
chandana@chandana-VirtualBox:~$ id
uid=1000(chandana) gid=1000(chandana) groups=1000(chandana),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131
(lxd),132(sambashare)
chandana@chandana-VirtualBox:~$ who -q -H
chandana
# users=1
chandana@chandana-VirtualBox:~$ who -r
         run-level 5  2020-09-14 14:19
```

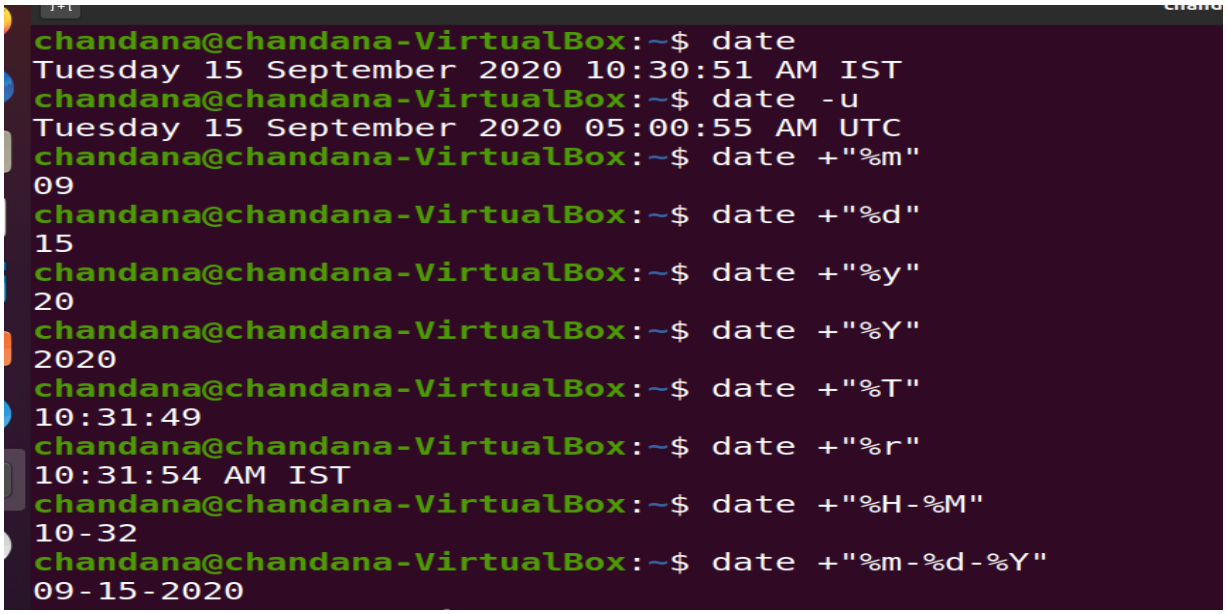
```
chandana@chandana-VirtualBox:~$ who -a
          system boot  2020-09-14 14:18
          run-level 5  2020-09-14 14:19
chandana ? :0          2020-09-14 14:19   ?          1379 (:0)
chandana@chandana-VirtualBox:~$ whoami
chandana
chandana@chandana-VirtualBox:~$ w
 15:35:28 up  1:17,  1 user,  load average: 0.27, 0.17, 0.11
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
chandana :0       :0               14:19   ?xdm?   3:33   0.01s /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_
chandana@chandana-VirtualBox:~$
```

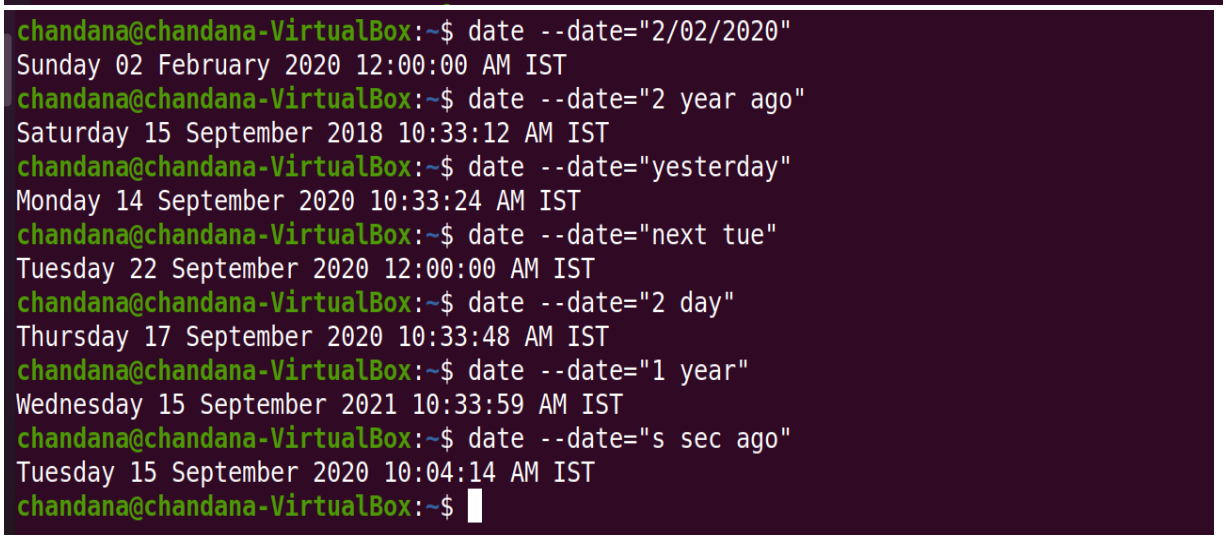
- **date :** It is a command that instructs the machine to display the current date & time. This command does not allow the user to change the date or the time. Only administrator can do that.

date %a	Displays Weekday name in short (like Mon, Tue, Wed)
date +%A	Displays Weekday name in full short (like Monday, Tuesday)
date +%b	Displays Month name in short (like Jan, Feb, Mar)
date +%B	Displays Month name in full short (like January, February)
date +%d	Displays Day of month (e.g., 01)
date +%D	Displays Current Date; shown in MM/DD/YY
date +%F	Displays Date; shown in YYYY-MM-DD
date +%H	Displays hour in (00..23) format
date +%m	Displays month (01..12)

date +%M	Displays minute (00..59)
date +%S	Displays second (00..60)
date +%Y	Displays full year i.e. YYYY
date +%T	Displays time; shown as HH:MM:SS Note: Hours in 24 Format

Execution of commands





- **passwd: It is used to change the password**

  $ passwd

  old password: ****

  new password: ******

  re-enter new password: ******

Now the new password is encrypted & registered by the system. The encrypted password is stored in the file /etc/shadow.

Rules for giving passwords

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

- **cal:** It displays the calendar of any specific month, or a complete year.

  If a user wants a quick view of calendar in Unix terminal, cal is the command. By default, cal command shows current month calendar as output.

  General Syntax: cal [[month] year]

  cal : Shows current month calendar on the terminal.

  cal 01 1990 : Shows calendar of selected month and year.

  cal 2020 : Shows the whole calendar of the year.

  cal -3 : Shows calendar of previous, current and next month

```
chandana@chandana-VirtualBox:~$ cal
    September 2020
Su Mo Tu We Th Fr Sa
          1   2   3   4   5
 6   7   8   9  10  11  12
13  14  15  16  17  18  19
20  21  22  23  24  25  26
27  28  29  30
```

```
chandana@chandana-VirtualBox:~$ cal 01 1990
    January 1990
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

chandana@chandana-VirtualBox:~$ cal 09 2020
   September 2020
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

chandana@chandana-VirtualBox:~$ cal 2020
                              2020
      January               February                March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                     1   1  2  3  4  5  6  7
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   8  9 10 11 12 13 14
12 13 14 15 16 17 18   9 10 11 12 13 14 15  15 16 17 18 19 20 21
19 20 21 22 23 24 25  16 17 18 19 20 21 22  22 23 24 25 26 27 28
26 27 28 29 30 31     23 24 25 26 27 28 29  29 30 31
```

```
chandana@chandana-VirtualBox:~$ cal -3
    August 2020          September 2020         October 2020
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                   1            1  2  3  4  5                1  2  3
 2  3  4  5  6  7  8   6  7  8  9 10 11 12   4  5  6  7  8  9 10
 9 10 11 12 13 14 15  13 14 15 16 17 18 19  11 12 13 14 15 16 17
16 17 18 19 20 21 22  20 21 22 23 24 25 26  18 19 20 21 22 23 24
23 24 25 26 27 28 29  27 28 29 30           25 26 27 28 29 30 31
30 31
```

1.9 COMBINING COMMANDS

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

Example:

$wc note; ls -l note

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

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

Example :

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

The combined output of the two commands is now sent to the file newlist. Whitespace is provided here only for better readability. You might reduce a few keystrokes like this $wc note;ls -l note)>newlist

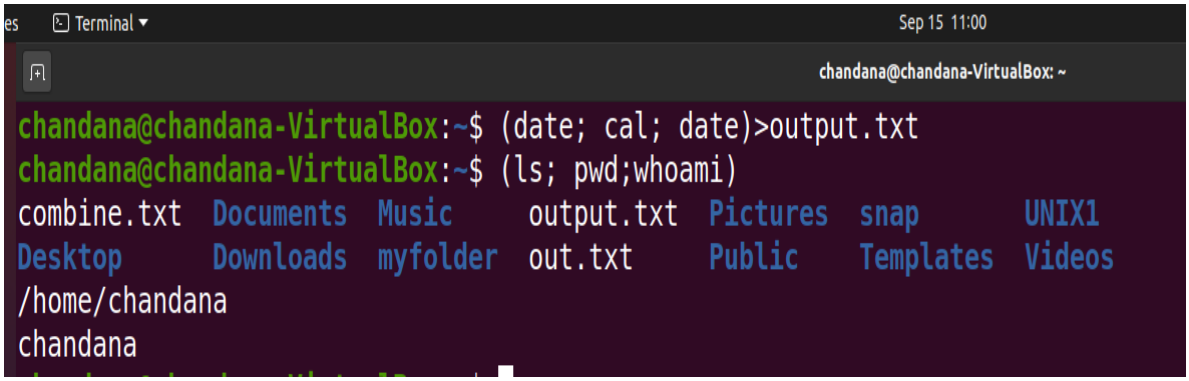
When a command line contains a semicolon, the shell understands that the command on each side of it needs to be processed separately. The ; here is known as a metacharacter, and you'll come across several metacharaters that have special meaning to the shell.

A command line can overflow or be split into multiple lines

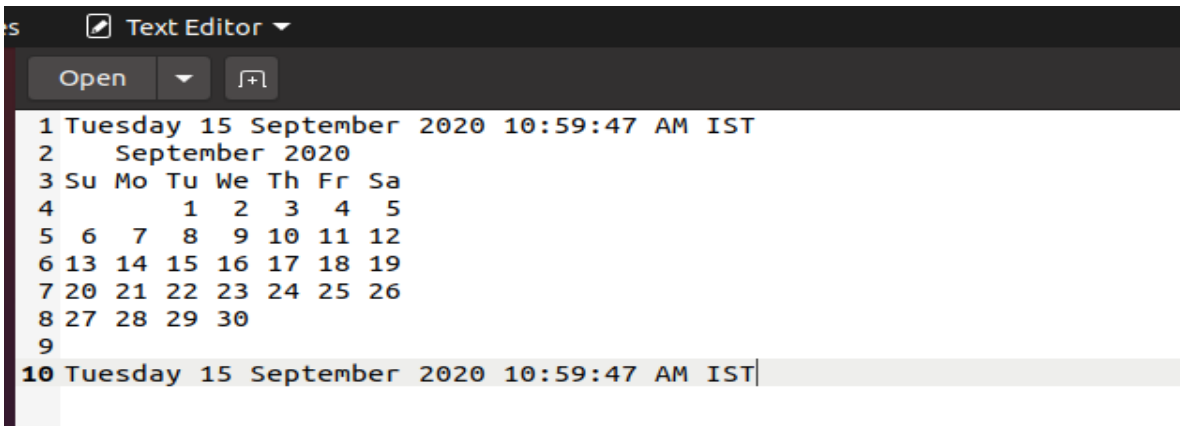
A command is often keyed in. though the terminal width is restricted to 80 characters, that doesn't prevent you from entering a command, or a sequence of them, in one line even though the total width may exceed 80 characters. The command simply overflows to the next line though it is stil in a single logical line.

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

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



For the first command output.txt text file has created, since redirecting the output of (date; cal; date) > output.txt file as shown below.



1.10 MEANING OF INTERNAL AND EXTERNAL COMMANDS

UNIX commands are classified into two types

Internal Commands - Ex: echo

External Commands - Ex: ls, cat

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

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

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

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

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

***type* command:**

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

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

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

type - Display information about command type.

The **type** command is a shell built-in that displays the kind of command the shell will execute, given a particular command name. It works like this: type *command*

where “command” is the name of the command you want to examine. Here are some examples:

$type type

Output: type is a shell built-in

$type ls

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

$type cp
Output: cp is /bin/cp

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

1.12 THE ROOT LOGIN

Root: the system administrator's login. The unix system provides a special login name for the exclusive use of the administrator, it is called **root**. This account doesn't need to be separately created but comes with every system. Its password is generally set at the time of installation of the system and has to be used on logging in

Login: **root**

Password: ******

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

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

1.13 BECOMING THE SUPER USER: SU COMMAND

Any user can acquire super user status with the **su** command if user knows the root password. Example, the user RNSIT becomes a super user in this way

$su
Password:****
#pwd
/home/RNSIT

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

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

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

$su – RNSIT

This sequences executes RNSIT's .profile and temporarily creates RNSIT's environment.

Su runs a separate sub shell, so this mode is terminated by hitting [Ctrl-d] or using **exit.**

CHAPTER 2 UNIX FILES

UNIX system has thousands of files. If user write a program, user add one more file to the system. When user compile it, user add some more. Files grow rapidly, and if they are not organized properly, user will find it difficult to locate them. So UNIX has a file system (UFS) to manage or organizes its own files in directory.

2.1 NAMING FILES

On a UNIX system, a filename can consist of up to 255 characters. Files may or may not have extensions and can consist of practically any ASCII character except the / and the Null character. Users are permitted to use control characters or other nonprintable characters in a filename. However, user should avoid using these characters while naming a file. It is recommended that only the following characters be used in filenames:

- Alphabets and numerals.
- The period (.), hyphen (-) and underscore (_).

UNIX imposes no restrictions on the extension. In all cases, it is the application that imposes that restriction. Eg. A C-Compiler expects C program filenames to end with .c, Oracle requires SQL scripts to have .sql extension.

A file can have as many dots embedded in its name. A filename can also begin with or end with a dot. ex: .sample, .one

The file name may consist of ordinary characters, digits and special tokens like the underscore, except the forward slash (/).

It is permitted to use special tokens like the ampersand (&) or spaces in a filename.

UNIX is case sensitive; chap01, Chap01 and CHAP01 are three different filenames that can coexist in the same directory.

2.2 BASIC FILE TYPES/CATEGORIES

A simple description of the UNIX system is this:

"On a UNIX system, everything is a file; if something is not a file, it is a process."

A UNIX system makes no difference between a file and a directory, since a directory is just a file containing names of other files. Programs, services, texts, images, and so forth, are all files. Input and output devices, and generally all devices, are considered to be files, according to the system.

Ordinary (Regular) File

This is the most common file type. An ordinary file can be either a text file or a binary file. A text file contains only printable characters and user can view and edit them. All C and Java program sources, shell scripts are text files. Every line of a text file is terminated with the newline character.

A binary file, on the other hand, contains both printable and nonprintable characters that cover the entire ASCII range. The object code and executables that you produce by compiling C programs are binary files. Sound and video files are also binary files.

Directory File

A directory contains no data, but keeps details of the files and subdirectories that it contains. A directory file contains one entry for every file and subdirectory that it houses. Each

entry has two components namely, the filename and a unique identification number of the file or directory (called the inode number). ls -i is the command to be used to know inode number.

When user create or remove a file, the kernel automatically updates its corresponding directory by adding or removing the entry (filename and inode number) associated with the file.

Device File

All the operations on the devices are performed by reading or writing the file representing the device. It is advantageous to treat devices as files as some of the commands used to access an ordinary file can be used with device files as well.

Device filenames are found in a single directory structure, /dev. A device file is not really a stream of characters. It is the attributes of the file that entirely govern the operation of the device. The kernel identifies a device from its attributes and uses them to operate the device.

2.3 ORGANIZATION OF FILES

All of the files in the UNIX file system are organized into a multi-leveled hierarchy called a directory tree. A family tree is an example of a hierarchical structure that represents how the UNIX file system is organized. The UNIX file system might also be envisioned as an inverted tree or the root system of plant. At the very top of the file system is single directory called "root" which is represented by a / (slash). All other files are "descendents" of root. The number of levels is largely arbitrary, although most UNIX systems share some organizational similarities. There is no single standard unix file structure. Most follow a general convention for filesystem organisation at the highest level.

- /: The slash / character alone denotes the root of the filesystem tree.
- /bin: Stands for “binaries” and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users.
- /boot: Contains all the files that are required for successful booting process.
- /dev: Stands for “devices”. Contains file representations of peripheral devices and pseudo-devices.
- /etc: Contains system-wide configuration files and system databases. Originally also contained “dangerous maintenance utilities” such as init, but these have typically been moved to /sbin or elsewhere.
- /home: Contains the home directories for the users.
- /lib: Contains system libraries, and some critical files such as kernel modules or device drivers.
- /root: The home directory for the superuser “root” – that is, the system administrator. This account’s home directory is usually on the initial filesystem, and hence not in /home (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during which other filesystems are not available. Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted.
- /tmp: A place for temporary files. Many systems clear this directory upon startup; it might have tmpfs mounted atop it, in which case its contents do not survive a reboot, or it might be explicitly cleared by a startup script at boot time.
- /usr: Originally the directory holding user home directories,its use has changed. It now holds executables, libraries, and shared resources that are not system critical, like the X Window

System, KDE, Perl, etc. However, on some Unix systems, some user accounts may still have a home directory that is a direct subdirectory of /usr, such as the default as in Minix. (on modern systems, these user accounts are often related to server or system use, and not directly used by a person).

- /usr/bin: This directory stores all binary programs distributed with the operating system not residing in /bin, /sbin or (rarely) /etc.
- /usr/include: Stores the development headers used throughout the system. Header files are mostly used by the #include directive in C/C++ programming language.
- /usr/lib: Stores the required libraries and data files for programs stored within /usr or elsewhere.
- /var: A short for "variable." A place for files that may change often – especially in size, for example e-mail sent to users on the system, or process-ID lock files.
- /var/log: Contains system log files.
- /var/mail: The place where all the incoming mails are stored. Users (other than root) can access their own mail only. Often, this directory is a symbolic link to /var/spool/mail.
- /var/spool: Spool directory. Contains print jobs, mail spools and other queued tasks.
- /var/tmp: A place for temporary files which should be preserved between system reboots.

2.4 HIDDEN FILES

An invisible file is one whose first character is the dot or period character (.)
UNIX programs (including the shell) use most of these files to store configuration information.
Some common examples of hidden files include the files:

.profile: the Bourne shell (sh) initialization script
.kshrc: the Korn shell (ksh) initialization script
.cshrc: the C shell (csh) initialization script
.rhosts: the remote shell configuration file

To list invisible files, specify the -a option to ls:

```
chandana@chandana-VirtualBox:~$ ls -a
.               .local      .ssh
..              .mozilla    .sudo_as_admin_successful
.bash_history   Music       Templates
.bash_logout    p1.c        .vboxclient-clipboard.pid
.bashrc         p2.c        .vboxclient-display-svga-x11.pid
.cache          p3.c        .vboxclient-draganddrop.pid
.config         p4.c        .vboxclient-hostversion.pid
Desktop         Pictures    .vboxclient-seamless.pid
Documents       .profile    Videos
Downloads       Public
.gnupg          snap
```

Single dot . : This represents current directory.
Double dot .. : This represents parent directory.

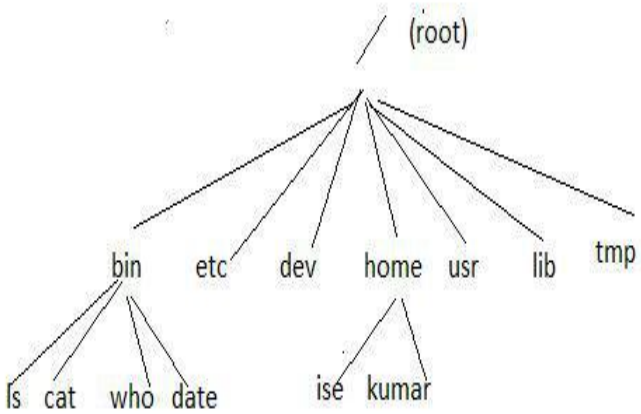
2.5 STANDARD DIRECTORIES

Although there may be a few exceptions, you will find a similar set of system directories on all UNIX and Linux operating systems. These system directories are located directly below the root directory, and are essential to the startup and continuous operation of the system. The following is a list of these key directories and their contents:

- /bin - The /bin directory contains programs needed for using and managing the system. Some of the frequently used commands in this directory are date (displays today's date), ls (lists the contents of a directory), and cp (makes a copy of a file). The name bin is used for this directory because executable programs in UNIX and Linux are called binary files.
- /dev - This directory contains system device files. A device file is a special object in the file system that provides an interface to a particular device. Examples of devices having device files in /dev are disk drives, tape drives, or CDROM drives.
- /etc - System specific configuration files, and files essential for system startup are located in the /etc directory. In the past, administrative executable programs were also stored in this directory but have been moved to the /sbin directory.
- /home - The /home directory is where the home directories for all users of the system are stored. For example, if John Doe's username is jdoe, the path to his home directory would be /home/jdoe. John would store his personal files and programs in this directory.
- /mnt - This directory is where temporary file systems are mounted. It may contain subdirectories like cdrom, floppy, and disk. Once a cdrom has been mounted on the system (made available for reading its contents), the path to the files located on the cdrom would be /mnt/cdrom.
- /opt - The /opt directory contains software files that are not installed when the operating system is installed. This directory usually contains products provided by third-party software vendors.
- /sbin - Programs for administering a system are located in the /sbin directory. Some examples are fdisk (used to partition a disk), fsck (used to check the integrity of a file system), and shutdown (used for stopping a system). An easy way to remember the contents of this directory is to equate sbin with "system binaries."
- /tmp - As the name implies, this directory is used for holding temporary files. This directory is commonly referred to as a scratch directory, and can be used by all system users. You should not store any important files in this directory because its contents may be deleted at any time. Your important files should be kept in your home directory.
- /usr - The /usr directory contains programs and files related to the users of a system. The data in /usr is typically read-only, and may be shared with other computer systems on a network. You will notice that the directory structure under /usr is somewhat similar to the directory structure under the root directory (/).
- /var - Files with varying content are stored in the /var directory. This includes system log files, mail system files, and print spooling system files.

2.6 THE PARENT-CHILD RELATIONSHIP

- A file is a set of data that has a name. The information can be an ordinary text, a user-written computer program, results of a computation, a picture, and so on.
- Unix organizes files in a tree-like hierarchical structure, with the root directory, indicated by a forward slash (/), at the top of the tree.
- The directories have specific purposes and generally hold the same types of information for easily locating files.
- Following are the directories that exist on the major versions of Unix−



- The root directory / has a number of subdirectories under it. These subdirectories in turn have more subdirectories and other files under them.
- Every file, apart from root, must have a parent, & it should be possible to trace the ultimate parent of a file to root. Thus, home directory is the parent of ise & kumar subdirectories & / is the parent of home & it is the grandparent of ise & kumar. If we create a file called sample.c under ise, then ise is the parent of sample.c

2.7 THE HOME VARIABLE: THE HOME DIRECTORY

- When user logs on to the system, UNIX automatically places user in a directory called the **home directory.** It is created by the system when user account is opened.
- If you log in using the login name chandana, you will land up in a directory that could have the pathname **/home/chandana.**
- The shell variable HOME known's yours home directory **$echo $HOME /home/chandana**. //First / represents root directory
- The above is an absolute pathname, which is simply a sequence of directory names separated by slashes.
- An absolute pathname shows a file's location with reference to the top i.e root. These slashes act as delimiters to file and directory names except that the first slash is a synonym
- for root. The directory Chandana is placed two levels below root.
- It's often convenient to refer a file foo located in user home directory as $HOME/foo and it is same as ~/foo in these shells.
- The ~ symbol is a tricky to use because it can refer to any user's home directory and not just our own. If user **yuktha** has the file foo in her home directory, then Chandana can access it as ~yuktha/foo.
- The principle is this: A tilde followed by / (like ~/foo) refers to one's own home directory, but when followed by a string(~yuktha) refers to the home directory of that user represented by the string.

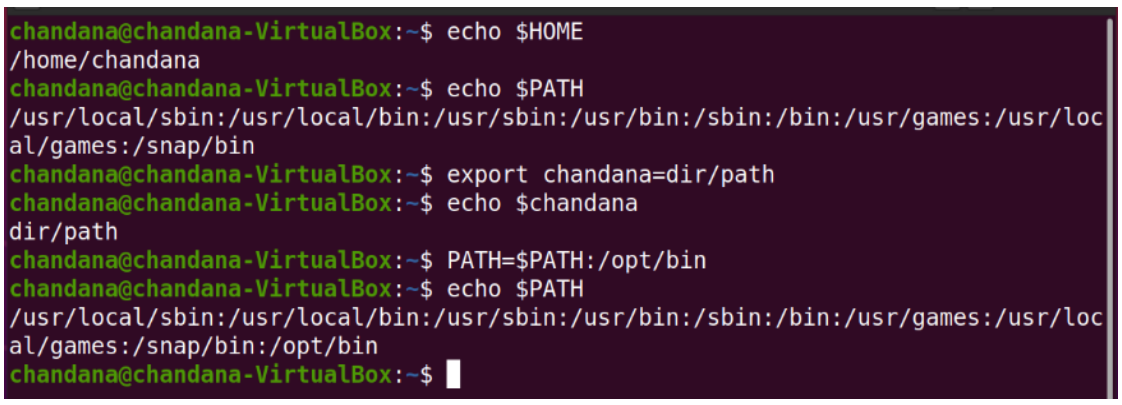
2.8 REACHING REQUIRED FILES - THE PATH VARIABLE

- A command runs in UNIX by executing a disk file. When user specify a command like date, the system will locate the associated file from a list of directories specified in the PATH variable and then executes it. The PATH variable normally includes the current directory also.

- When user enter any UNIX command, user is actually specifying the name of an executable file located somewhere on the system. The system goes through the following steps in order to determine which program to execute:
  - Built in commands (such as cd and echo) are executed within the shell.
  - If an absolute path name (such as /bin/ls) or a relative path name (such as ./myprog) is specified, then the system executes the program from the specified directory.
  - Otherwise the PATH variable is used.
- The PATH variable is a colon-delimited list of directories that the shell searches through when user enter a command. Program files (executables) are kept in many different places on the Unix system. User path tells the Unix shell where to look on the system when user request a particular program. To find out what user path is, at the Unix shell prompt echo $PATH, user path will look something like the following.
  /usr2/username/bin:/usr/local/bin:/usr/bin:.
- User can see their username in place of username. Using the above example path, if user enter the ls command, shell will look for the appropriate executable file in the following order: first, it would look through the directory /usr2/username/bin, then /usr/local/bin, then /usr/bin, and finally the local directory, indicated by the . (a period).

2.9 MANIPULATING THE PATH

- The PATH variable contains the search path for executing commands and scripts.
  To see your PATH, enter:
  $ echo $PATH
  /home/khess/.local/bin:/home/khess/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
- Temporarily change your PATH by entering the following command to add /opt/bin:
  $** PATH=$PATH:/opt/bin **$ echo $PATH
  /home/khess/.local/bin:/home/khess/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/bin
- The change is temporary for the current session. It isn't permanent because it's not entered into the .bashrc file.
- To make the change permanent, enter the command PATH=$PATH:/opt/bin into your home directory's .bashrc file.
- When you do this, you're creating a new PATH variable by appending a directory to the current PATH variable, $PATH. A colon (:) separates PATH entries.
- Few shells also use export command to manipulate the path.



2.10 RELATIVE AND ABSOLUTE PATHNAMES

Relative pathname: Pathnames that don't begin with / specify locations relative to the current working directory. It uses either the current or parent directory as reference and specifies path relative to it. A relative pathname uses one of these cryptic symbols.

cd progs
cat login.sql

Here both are presumed to exist in the current directory. Now, if progs contain a directory scripts under it, user won't need an absolute pathname to change to that directory. Just user can use, cd progs/scripts.

Here we have a pathname that has a /, but it is not absolute because it doesn't begin with a /.

Absolute: If the first character of a pathname is / the files location must be determined with respect to root(/) . Such a pathname is called absolute pathname.

cat /home/chandana

When user have more than one / in a pathname for such / have to descend one level in the file system. Thus chandana is one level below home and two levels below root.

When user specify a file by using frontslashes to demarcate the various levels, have a mechanism of identifying a file uniquely. No two files in a UNIX system can have identical absolute pathnames. User can have two files with the same name, but in different directories, their pathnames will also be different. Thus, the file /home/chandana/progs/p1.c can coexist with the file /home/chandana/scripts/p1.c.

When user specify the date command, the system has to locate the file date from a list of directories specified in the PATH variable and then execute it. However if user know the location of a command in prior, for example date is usually located in /bin or /usr/bin .

Use absolute pathname i,e precede its name with complete path $/bin/date. For example if you need to execute program **less** residing in /usr/local/bin you need to enter the absolute pathname $/usr/local/bin/less

2.11 DIRECTORY COMMANDS – PWD, CD, MKDIR, RMDIR COMMANDS

- **pwd: Print Working Directory (checking your current directory)**
  - Unix encourages us to believe that, like a file, a user is placed in a specific directory of the file system on logging in.
  - We can move around from one directory to another, but at any point of time, we are located in only one directory. This directory is known as current directory.
  - At any time, we can able to know what is our directory. The pwd command tells us that

    $pwd
    /home/csr
  - This command is built in shell command and is available on most of the shell – bash, Bourne shell, ksh,zsh, etc.
  - Like HOME, pwd displays the absolute pathname. As we navigate the file system with the cd command, we'll be using pwd to know our current directory.

Basic syntax

$pwd [option]

Options	Description
-L (logical)	Use PWD from environment, even if it contains symbolic links
-P (physical)	Avoid all symbolic links

If both '**-L**' and ' **-P**' options are used, option **'L**' is taken into priority. If no option is specified at the prompt, pwd will avoid all symlinks, i.e., take option '**-P**' into account.

```
csr@csr-VirtualBox:~$ pwd
/home/csr
csr@csr-VirtualBox:~$ pwd -L
/home/csr
csr@csr-VirtualBox:~$ pwd -P
/home/csr
csr@csr-VirtualBox:~$ pwd --help
pwd: pwd [-LP]
    Print the name of the current working directory.

    Options:
      -L        print the value of $PWD if it names the current working
                directory
      -P        print the physical directory, without any symbolic links

    By default, `pwd' behaves as if `-L' were specified.

    Exit Status:
    Returns 0 unless an invalid option is given or the current directory
    cannot be read.
```

- **cd: Changing The Current Directory**
  - The **cd** command, which stands for "change directory", changes the shell's current working directory.
  - The **cd** command is one of the commands you will use the most at the command line in UNIX. It allows you to change your working directory. You use it to move around within the hierarchy of your file system.

  Ex 1: When used with an argument it changes the current directory to the directory specified as argument. For example, **unix** is a directory under user directory **csr**. To change from csr directory to unix directory, issue the command as follows

  $pwd
  /home/csr
  $cd unix
  $pwd
  /home/csr/unix

```
csr@csr-VirtualBox:~$ pwd
/home/csr
csr@csr-VirtualBox:~$ cd unix
bash: cd: unix: No such file or directory
csr@csr-VirtualBox:~$ mkdir unix
csr@csr-VirtualBox:~$ cd unix
csr@csr-VirtualBox:~/unix$ pwd
/home/csr/unix
```

Ex 2: When cd used without arguments: cd when used without arguments reverts to home directory

$pwd
/home/csr/unix
$cd
cd without argument will change directory from unix to its home directory csr
$pwd
/home/csr

Ex 3: If your present working directory is /home/csr/unix and you need to switch to /bin directory directly, use absolute pathname i.e /bin with cd command

$pwd
/home/csr/unix
$cd /bin
$pwd
/bin

```
csr@csr-VirtualBox:~/unix$ pwd
/home/csr/unix
csr@csr-VirtualBox:~/unix$ cd /bin
csr@csr-VirtualBox:/bin$ pwd
/bin
csr@csr-VirtualBox:/bin$
```

- **mkdir: "making directory". mkdir** is used to create directories on a file system.

If the specified directory does not already exist, **mkdir** creates it. More than one directory may be specified when calling **mkdir**.

mkdir syntax

mkdir [OPTION ...] DIRECTORY ...

Ex 1: To create a directory named rnsit, issue the following command**.**

$mkdir rnsit

rnsit directory is created under present working directory.

Assume that pwd is /home/csr, then rnsit directory is created under csr directory.

Ex 2: To create three directories at a time, named ise cse mca pass directory names as arguments.

$mkdir ise cse mca

Ex 3: To create a directory tree:

To create a directory named Faculty and create two subdirectories named Teaching and NonTeaching under Faculty, issue the command. Faculty is a parent directory.

$mkdir	parent directory	sub-directories	
$mkdir	**Faculty**	**Faculty/Teaching**	**Faculty/NonTeaching**

Ex 4: Error while creating a directory tree

$mkdir Faculty/Teaching Faculty/NonTeaching

mkdir: Failed to make a directory “Faculty/Teaching”; no such file or directory

mkdir: Failed to make a directory “Faculty/NonTeaching”; no such file or directory

Error is due to the fact that the parent directory named Faculty is not created before creating sub directories Teaching and NonTeaching**.**

Ex 5: **$mkdir test**

mkdir: Failed to make directory "test"; Permission denied.

This can happen due to:

a. The directory named test may already exist

b. There may be an ordinary file by the same name in the current directory.

c. The permissions set for the current directory do not permit the creation of files and directories by the user.

- **rmdir: Removing Directories**

The **rmdir** command removes the directory entry specified by each directory argument, provided the directory is empty.

Ex 1: **$rmdir rnsit**

removes the directory named rnsit

Arguments are processed in the order given. To remove both a parent directory and a subdirectory of that parent, the subdirectory must be specified first, so the parent directory is empty when **rmdir** tries to remove it.

The reverse logic of mkdir is applied.

$rmdir	**subdirectories**	**parent directory**
$rmdir	**Faculty/NonTeaching Faculty/Teaching**	**Faculty**

You cant delete a directory with rmdir unless it is empty. In this example Faculty directory cannot be removed until the sub directories Faculty/NonTeaching and Faculty/Teaching are removed.

You cant remove a sub directory unless you are place in a directory which is hierarchically above the one you have chosen to remove.

2.12 THE DOT(.) AND DOUBLE DOTS(..) NOTATIONS TO REPRESENT PRESENT AND PARENT DIRECTORIES AND THEIR USAGE IN RELATIVE PATH NAMES.

- User can move from working directory /home/csr/unix/module1 to home directory /home/unix using cd command like

  $pwd

  /home/csr/unix/module1

  $cd /home/csr

  $pwd

  /home/csr

- Navigation becomes easy by using common ancestor.

. (a single dot) - This represents the current directory

.. (two dots) - This represents the parent directory

- Assume user is currently placed in /home/csr/unix/module1

  $pwd

  /home/csr/unix/module1

  $cd ..

  $pwd

  /home/csr/unix

- This method is compact and easy when ascending the directory hierarchy. The command

cd .. Translates to this ―change your current directory to parent of current directory.

- The relative paths can also be used as:

$pwd
/home/csr/unix
$cd ../..
$pwd
/home

2.13 FILE RELATED COMMANDS – CAT, MV, RM, CP, WC, AND OD COMMANDS.

- **cat: Create, view, concatenate files**

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

```
$ cat cprogram.c
#include<stdio.h>
void main()
{
        printf("hello");
}
```

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

```
$ cat a.txt b.txt
It contains the contents of a.txt
It contains the contents of b.txt
```

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

cat Options

- To view contents of a file preceding with line numbers **-n** is the numbering option helps programmer in debugging programs.
- To create a file **$cat >newfile** This is a new file which contains some text, just to add some contents to the file new [ctrl-d] $_
  When the command line is terminated with [Enter], the prompt vanishes. Cat now waits to take input from the user. Enter few lines; press [ctrl-d] to signify the end of input to the system To display the file contents of new use file name with cat command. **$ cat new** This is a new file which contains some text, just to Add some contents to the file new.
- is the command and can input the content of newfile also.
- To copy the contents of one file to another file **$cat [filename whose contents is to be copied] > [destination filename]** is the command.
- Cat command can append the contents of one file to the end of another file by using the command **$cat file1 >> file2.**
- Cat command can display content in reverse order using tac command. **$tac filename**
- Cat command can highlight the end of line **$cat -E "filename"**
- Cat command to merge the contents of multiple file **$cat "file1" "file2" "file3" > "newfile"**

```
csr@csr-VirtualBox:~/Desktop$ cat a.txt
Hi Hello
csr@csr-VirtualBox:~/Desktop$ cat a.txt b.txt
Hi Hello
RNSIT, Bengaluru
csr@csr-VirtualBox:~/Desktop$ cat -n a.txt
     1  Hi Hello
csr@csr-VirtualBox:~/Desktop$ cat >newfile
Welcome to unix programming
^C
csr@csr-VirtualBox:~/Desktop$ cat newfile
Welcome to unix programming
csr@csr-VirtualBox:~/Desktop$ cat a.txt > b.txt
csr@csr-VirtualBox:~/Desktop$ cat a.txt
Hi Hello
csr@csr-VirtualBox:~/Desktop$ cat b.txt
Hi Hello
csr@csr-VirtualBox:~/Desktop$ cat a.txt >> b.txt
csr@csr-VirtualBox:~/Desktop$ cat b.txt
Hi Hello
Hi Hello
```

```
csr@csr-VirtualBox:~/Desktop$ cat c.txt
Welcome to RNSIT
ISE Department
5th sem A
csr@csr-VirtualBox:~/Desktop$ tac c.txt
5th sem A
ISE Department
Welcome to RNSIT
csr@csr-VirtualBox:~/Desktop$ cat -E "c.txt"
Welcome to RNSIT$
ISE Department $
5th sem A$
```

```
csr@csr-VirtualBox:~/Desktop$ cat *.txt
Hi Hello
Hi Hello
Hi Hello
Welcome to RNSIT
ISE Department
5th sem A
Hi, Hello
csr@csr-VirtualBox:~/Desktop$ cat "a.txt" "b.txt" "c.txt" > "d.txt"
csr@csr-VirtualBox:~/Desktop$ cat d.txt
Hi Hello
Hi Hello
Hi Hello
Welcome to RNSIT
ISE Department
5th sem A
```

- **mv: Renaming Files**

The mv command renames (moves) files. The main two functions are:

1. It renames a file(or directory)
2. It moves a group of files to different directory

It doesn't create a copy of the file; it merely renames it. No additional space is consumed on disk during renaming.

Syntax: mv [option] source destination

Ex: To rename the file a.txt as movefile.txt we can use the following command

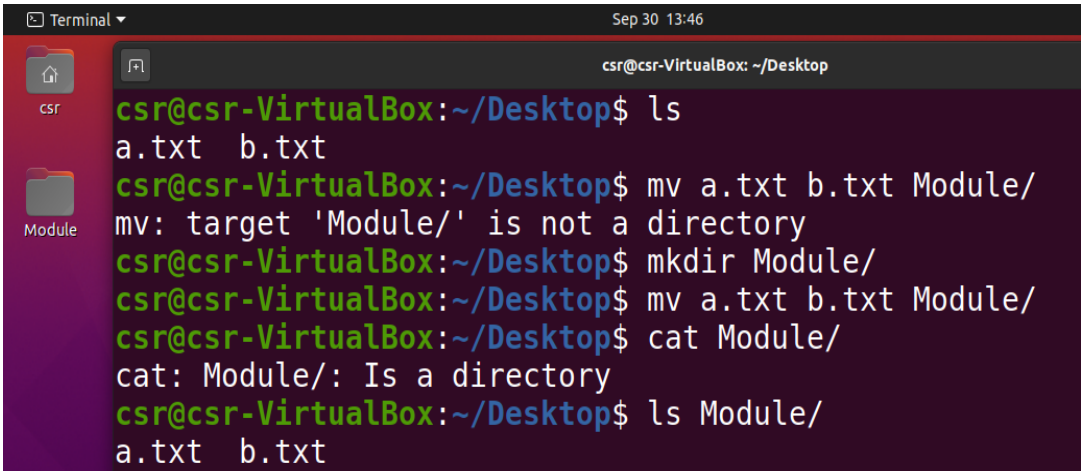
$ mv a.txt movefile.txt

- If the destination file doesn‘t exist in the current directory, it will be created. Or else it will just rename the specified file in mv command.

```
csr@csr-VirtualBox:~/Desktop$ ls
a.txt  b.txt
csr@csr-VirtualBox:~/Desktop$ cat a.txt
RNSIT, Bengaluru
csr@csr-VirtualBox:~/Desktop$ cat b.txt
Hi Hello
Welcome to RNSIT
ISE Department
csr@csr-VirtualBox:~/Desktop$ mv a.txt movefile.txt
csr@csr-VirtualBox:~/Desktop$ ls
b.txt  movefile.txt
csr@csr-VirtualBox:~/Desktop$ cat movefile.txt
RNSIT, Bengaluru
csr@csr-VirtualBox:~/Desktop$ mv movefile.txt b.txt
csr@csr-VirtualBox:~/Desktop$ ls
b.txt
```

- A group of files can be moved to a directory. Ex: Moves three files a.txt, b.txt, to the directory Module

$ mv a.txt b.txt Module



- Can also used to rename directory

$ mv nameofthedirectory newname

- mv replaces the filename in the existing directory entry with the new name. It doesn't create a copy of the file; it renames it.

Options:

1. -i (Interactive): Like in cp, the -i option makes the command ask the user for confirmation before moving a file that would overwrite an existing file, you have to press **y** for confirm moving, any other key leaves the file as it is. This option doesn't work if the file doesn't exist, it simply rename it or move it to new location.

```
csr@csr-VirtualBox:~/Desktop$ ls
a.txt  b.txt
csr@csr-VirtualBox:~/Desktop$ cat a.txt b.txt
RNSIT, Bengaluru
Hi Hello
Welcome to RNSIT
ISE Department
csr@csr-VirtualBox:~/Desktop$ mv -i a.txt b.txt
mv: overwrite 'b.txt'? y
csr@csr-VirtualBox:~/Desktop$ ls
b.txt
csr@csr-VirtualBox:~/Desktop$ cat b.txt
RNSIT, Bengaluru
```

2. -f (Force): mv prompts for confirmation overwriting the destination file if a file is write protected**.** The **-f** option overrides this minor protection and overwrite the destination file forcefully and delete the source file.

```
csr@csr-VirtualBox:~/Desktop$ cat b.txt
RNSIT, Bengaluru
csr@csr-VirtualBox:~/Desktop$ ls -l b.txt
-r--r--r-- 1 csr csr 17 Sep 30 12:16 b.txt
csr@csr-VirtualBox:~/Desktop$ mv a.txt b.txt
mv: replace 'b.txt', overriding mode 0444 (r--r--r--)? ls
csr@csr-VirtualBox:~/Desktop$ ls
a.txt  b.txt
csr@csr-VirtualBox:~/Desktop$ mv -f a.txt b.txt
csr@csr-VirtualBox:~/Desktop$ ls
b.txt
csr@csr-VirtualBox:~/Desktop$ cat b.txt
RNSIT, Bengaluru
```

3. -n (no-clobber): With **-n** option, **mv** prevent an existing file from being overwritten.

4. -b(backup): With this option it is easier to take a backup of an existing file that will be overwritten as a result of the **mv** command. This will create a backup file with the tilde character(~) appended to it.

5. **–version:** This option is used to display the version of **mv** which is currently running on your system.

```
csr@csr-VirtualBox:~/Desktop$ ls
a.txt  b.txt
csr@csr-VirtualBox:~/Desktop$ mv -b a.txt b.txt
csr@csr-VirtualBox:~/Desktop$ ls
b.txt  b.txt~
csr@csr-VirtualBox:~/Desktop$ mv --version
mv (GNU coreutils) 8.30
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.ht
ml>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Mike Parker, David MacKenzie, and Jim Meyering.
```

- **rm: Deleting Files**

rm stands for **remove** here. rm command is used to remove objects such as files, directories, symbolic links and so on from the file system like UNIX. To be more precise, rm removes references to objects from the filesystem, where those objects might have had multiple references (for example, a file with two different names). By default, it does not remove directories.

This command normally works silently and you should be very careful while running **rm** command because once you delete the files then you are not able to recover the contents of files and directories.

Syntax: rm [option] [filename]

Options:

1. -i (Interactive Deletion): Like in cp, the -i option makes the command ask the user for confirmation before removing each file, you have to press **y** for confirm deletion, any other key leaves the file un-deleted.

2. -f (Force Deletion): rm prompts for confirmation removal if a file is **write protected**. The **-f** option overrides this minor protection and removes the file forcefully. **-f** option of rm command will not work for write-protect directories.

3. -r (Recursive Deletion): With **-r(or -R)** option rm command performs a tree-walk and will delete all the files and sub-directories recursively of the parent directory. At each stage it deletes everything it finds. Normally, **rm** wouldn't delete the directories but when used with this option, it will delete.

4. –version: This option is used to display the version of **rm** which is currently running on your system.

```
csr@csr-VirtualBox:~/Desktop$ ls
a.txt  b.txt  c.txt  d.txt  e.txt
csr@csr-VirtualBox:~/Desktop$ rm a.txt
csr@csr-VirtualBox:~/Desktop$ ls
b.txt  c.txt  d.txt  e.txt
csr@csr-VirtualBox:~/Desktop$ rm b.txt c.txt
csr@csr-VirtualBox:~/Desktop$ ls
d.txt  e.txt
csr@csr-VirtualBox:~/Desktop$ rm -i d.txt
rm: remove regular file 'd.txt'? n
csr@csr-VirtualBox:~/Desktop$ ls
d.txt  e.txt
csr@csr-VirtualBox:~/Desktop$ ls -l
total 8
-rw-rw-r-- 1 csr csr 26 Sep 30 14:07 d.txt
-rw-rw-r-- 1 csr csr 16 Sep 30 14:08 e.txt
csr@csr-VirtualBox:~/Desktop$ chmod a-w *.txt
csr@csr-VirtualBox:~/Desktop$ ls -l
total 8
-r--r--r-- 1 csr csr 26 Sep 30 14:07 d.txt
-r--r--r-- 1 csr csr 16 Sep 30 14:08 e.txt
```

```
csr@csr-VirtualBox:~/Desktop$ rm d.txt
rm: remove write-protected regular file 'd.txt'? y
csr@csr-VirtualBox:~/Desktop$ ls
e.txt
csr@csr-VirtualBox:~/Desktop$ rm -f e.txt
csr@csr-VirtualBox:~/Desktop$ ls
csr@csr-VirtualBox:~/Desktop$ mkdir A A/B A/C
csr@csr-VirtualBox:~/Desktop$ ls
A
csr@csr-VirtualBox:~/Desktop$ cd A
csr@csr-VirtualBox:~/Desktop/A$ ls
B  C
csr@csr-VirtualBox:~/Desktop/A$ ls B
a.txt  b.txt
csr@csr-VirtualBox:~/Desktop/A$ ls C
c.txt  d.txt
csr@csr-VirtualBox:~/Desktop/A$ rm *
rm: cannot remove 'B': Is a directory
rm: cannot remove 'C': Is a directory
```

```
csr@csr-VirtualBox:~/Desktop/A$ rm -r *
csr@csr-VirtualBox:~/Desktop/A$ ls
csr@csr-VirtualBox:~/Desktop/A$ rm --version
rm (GNU coreutils) 8.30
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.ht
ml>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Paul Rubin, David MacKenzie, Richard M. Stallman,
and Jim Meyering.
```

- **cp: Copying A File**

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

Syntax:

cp [option] source destination

cp [option] source directory

cp [option] source-1 source-2 source-3 source-n directory

1. Two file names: If the command contains two file names, then it copies the contents of 1st file to the 2nd file. If the 2nd file doesn't exist, then first it creates one and content is copied to it. But if it existed then it is simply overwritten without any warning. So be careful when you choose destination file name.

2. One or more arguments: If the command has one or more arguments, specifying file names and following those arguments, an argument specifying directory name then this command copies each source file to the destination directory with the same name, created if not existed but if already existed then it will be overwritten.

Suppose there is a directory named **A** having a text file a.txt, b.txt and a directory name **B** in which we are going to copy all files.

```
csr@csr-VirtualBox:~/Desktop$ ls
A  a.txt  b.txt  c.txt  d.txt  e.txt
csr@csr-VirtualBox:~/Desktop$ cp a.txt new.txt
csr@csr-VirtualBox:~/Desktop$ ls
A  a.txt  b.txt  c.txt  d.txt  e.txt  new.txt
csr@csr-VirtualBox:~/Desktop$ ls A
csr@csr-VirtualBox:~/Desktop$ cp a.txt b.txt A
csr@csr-VirtualBox:~/Desktop$ ls A
a.txt  b.txt
```

3. Two directory names: If the command contains two directory names, cp copies all files of the source directory to the destination directory, creating any files or directories needed. This mode of operation requires an additional option, typically R, to indicate the recursive copying of directories.

cp behavior depends upon whether Dest_directory is exist or not. If the Dest_directory doesn't exist, cp creates it and copies content of Src_directory recursively as it is. But if Dest_directory exists then copy of Src_directory becomes sub-directory under Dest_directory.

Options:

1. -i(interactive): i stands for Interactive copying. With this option system first warns the user before overwriting the destination file. **cp** prompts for a response, if you press **y** then it overwrites the file and with any other option leave it uncopied.

2. -b(backup): With this option **cp** command creates the backup of the destination file in the same folder with the different name and in different format.

```
csr@csr-VirtualBox:~/Desktop$ ls
A  a.txt  b.txt  c.txt  d.txt  e.txt  new.txt
csr@csr-VirtualBox:~/Desktop$ cat a.txt
RNSIT, Bengaluru
csr@csr-VirtualBox:~/Desktop$ cat b.txt
Hi Hello
Welcome to RNSIT
ISE Department
csr@csr-VirtualBox:~/Desktop$ cp -i a.txt b.txt
cp: overwrite 'b.txt'? y
csr@csr-VirtualBox:~/Desktop$ cat a.txt
RNSIT, Bengaluru
csr@csr-VirtualBox:~/Desktop$ cat b.txt
RNSIT, Bengaluru
csr@csr-VirtualBox:~/Desktop$ cp -b a.txt b.txt
csr@csr-VirtualBox:~/Desktop$ ls
A  a.txt  b.txt  b.txt~  c.txt  d.txt  e.txt  new.txt
```

3. -f(force): If the system is unable to open destination file for writing operation because the user doesn't have writing permission for this file then by using **-f** option with **cp** command, destination file is deleted first and then copying of content is done from source to destination file.

4. -r or -R: Copying directory structure. With this option **cp** command shows its recursive behavior by copying the entire directory structure recursively. Suppose we want to copy **A** directory containing many files, directories into **unix** directory(not exist).

```
csr@csr-VirtualBox:~/Desktop$ ls -l a.txt
-r--r--r-- 1 csr csr 17 Sep 30 12:16 a.txt
csr@csr-VirtualBox:~/Desktop$ cp b.txt a.txt
cp: cannot create regular file 'a.txt': Permission denied
csr@csr-VirtualBox:~/Desktop$ cp -f b.txt a.txt
csr@csr-VirtualBox:~/Desktop$ ls A/
a.txt  b.txt
csr@csr-VirtualBox:~/Desktop$ cp A unix
cp: -r not specified; omitting directory 'A'
csr@csr-VirtualBox:~/Desktop$ cp -r A unix
csr@csr-VirtualBox:~/Desktop$ ls unix
a.txt  b.txt
```

5. -p(preserve): With **-p** option **cp** preserves the following characteristics of each source file in the corresponding destination file: the time of the last data modification and the time of the last access, the ownership (only if it has permissions to do this), and the file permission-bits.

```
csr@csr-VirtualBox:~/Desktop$ ls -l a.txt
-rw-rw-r-- 1 csr csr 17 Sep 30 14:42 a.txt
csr@csr-VirtualBox:~/Desktop$ cp a.txt b.txt
csr@csr-VirtualBox:~/Desktop$ ls -l b.txt
-rw-rw-r-- 1 csr csr 17 Sep 30 14:45 b.txt
csr@csr-VirtualBox:~/Desktop$ ls -l a.txt
-rw-rw-r-- 1 csr csr 17 Sep 30 14:42 a.txt
csr@csr-VirtualBox:~/Desktop$ ls -l b.txt
-rw-rw-r-- 1 csr csr 17 Sep 30 14:45 b.txt
csr@csr-VirtualBox:~/Desktop$ cp -p a.txt b.txt
csr@csr-VirtualBox:~/Desktop$ ls -l b.txt
-rw-rw-r-- 1 csr csr 17 Sep 30 14:42 b.txt
```

Copying using * wildcard: The star wildcard represents anything i.e. all files and directories. Suppose we have many text document in a directory and wants to copy it another directory, it takes lots of time if we copy files 1 by 1 or command becomes too long if specify all these file names as the argument, but by using * wildcard it becomes simple.

```
csr@csr-VirtualBox:~/Desktop$ ls A
csr@csr-VirtualBox:~/Desktop$ ls
A  a.txt  b.txt  b.txt~  c.txt  d.txt  e.txt  new.txt
csr@csr-VirtualBox:~/Desktop$ cp *.txt A
csr@csr-VirtualBox:~/Desktop$ ls A
a.txt  b.txt  c.txt  d.txt  e.txt  new.txt
```

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

    Syntax: $ wc filename
  - Line: Any group of characters not containing a newline
  - Word: group of characters not containing a space, tab or newline
  - Character: smallest unit of information, and includes a space, tab and newline

    wc offers 3 options to make a specific count. –l option counts only number of lines, - w and –c options count words and characters, respectively.

```
csr@csr-VirtualBox:~/Desktop$ ls
a.txt  b.txt  b.txt~  c.txt  d.txt  e.txt
csr@csr-VirtualBox:~/Desktop$ cat b.txt
Hi Hello
Welcome to RNSIT
ISE Department
csr@csr-VirtualBox:~/Desktop$ wc -l b.txt
3 b.txt
csr@csr-VirtualBox:~/Desktop$ wc -w b.txt
7 b.txt
csr@csr-VirtualBox:~/Desktop$ wc -c b.txt
41 b.txt
csr@csr-VirtualBox:~/Desktop$ wc -m b.txt
41 b.txt
csr@csr-VirtualBox:~/Desktop$ wc -L b.txt
16 b.txt
```

- Multiple filenames, wc produces a line for each file, as well as a total count.

```
csr@csr-VirtualBox:~/Desktop$ wc *.txt
  1   2  17 a.txt
  3   7  41 b.txt
  0   0   0 chap11.txt
  0   0   0 chap12.txt
  1   2  10 c.txt
  1   4  26 d.txt
  1   3  16 e.txt
  7  18 110 total
```

➢ **od: Displaying Data in Octal**

od command is used to convert the content of input in different formats with octal format as the default format. This command is especially useful when debugging Linux scripts for unwanted changes or characters. If more than one file is specified, od command concatenates them in the listed order to form the input. It can display output in a variety of other formats, including hexadecimal, decimal, and ASCII. It is useful for visualizing data that is not in a human-readable format, like the executable code of a program.

Syntax: od [Option] … [File]

Options:

1. -b: It displays the contents of input in octal format. The first column in the output of od represents the byte offset in file.

2. -c: It displays the contents of input in character format.

3. -An: It displays the contents of input in character format but with no offset information.

4. -A: It displays the contents of input in different format by concatenation some special character with -A.

- **-Ax for Hexadecimal format (we concatenate x with -A)**
- **-Ao for Hexadecimal format (we concatenate o with -A)**
- **-Ad for Hexadecimal format(we concatenate d with -A)**

5. –: Accept input from command line.

Ex 1: $ more odfile this file is an example for od command created manually with content below:

White space includes a
The ^G character rings a bell
The ^L character rings a page

```
csr@csr-VirtualBox:~/Desktop$ gedit odfile
csr@csr-VirtualBox:~/Desktop$ more odfile
White space includes a
The ^G character rings a bell
The ^L character skips a page
csr@csr-VirtualBox:~/Desktop$ od -b odfile
0000000 127 150 151 164 145 040 163 160 141 143 145 040 151 156 143 154
0000020 165 144 145 163 040 141 012 124 150 145 040 136 107 040 143 150
0000040 141 162 141 143 164 145 162 040 162 151 156 147 163 040 141 040
0000060 142 145 154 154 012 124 150 145 040 136 114 040 143 150 141 162
0000100 141 143 164 145 162 040 163 153 151 160 163 040 141 040 160 141
0000120 147 145 012
0000123
```

```
csr@csr-VirtualBox:~/Desktop$ od -bc odfile
0000000 127 150 151 164 145 040 163 160 141 143 145 040 151 156 143 154
          W   h   i   t   e       s   p   a   c   e       i   n   c   l
0000020 165 144 145 163 040 141 012 124 150 145 040 136 107 040 143 150
          u   d   e   s       a  \n   T   h   e       ^   G       c   h
0000040 141 162 141 143 164 145 162 040 162 151 156 147 163 040 141 040
          a   r   a   c   t   e   r       r   i   n   g   s       a
0000060 142 145 154 154 012 124 150 145 040 136 114 040 143 150 141 162
          b   e   l   l  \n   T   h   e       ^   L       c   h   a   r
0000100 141 143 164 145 162 040 163 153 151 160 163 040 141 040 160 141
          a   c   t   e   r       s   k   i   p   s       a       p   a
0000120 147 145 012
          g   e  \n
0000123
```

Ex 2: input.txt is the input file for od command which contains the digits from 101 to 110 as input.

```
csr@csr-VirtualBox:~/Desktop$ cat input.txt
101
102
103
104
105
106
107
108
109
110
csr@csr-VirtualBox:~/Desktop$ od -b input.txt
0000000 061 060 061 012 061 060 062 012 061 060 063 012 061 060 064 012
0000020 061 060 065 012 061 060 066 012 061 060 067 012 061 060 070 012
0000040 061 060 071 012 061 061 060 012
0000050
csr@csr-VirtualBox:~/Desktop$ od -c input.txt
0000000   1   0   1  \n   1   0   2  \n   1   0   3  \n   1   0   4  \n
0000020   1   0   5  \n   1   0   6  \n   1   0   7  \n   1   0   8  \n
0000040   1   0   9  \n   1   1   0  \n
0000050
```

```
csr@csr-VirtualBox:~/Desktop$ od -An -c input.txt
   1   0   1  \n   1   0   2  \n   1   0   3  \n   1   0   4  \n
   1   0   5  \n   1   0   6  \n   1   0   7  \n   1   0   8  \n
   1   0   9  \n   1   1   0  \n
csr@csr-VirtualBox:~/Desktop$ od -Ax -c input.txt
000000   1   0   1  \n   1   0   2  \n   1   0   3  \n   1   0   4  \n
000010   1   0   5  \n   1   0   6  \n   1   0   7  \n   1   0   8  \n
000020   1   0   9  \n   1   1   0  \n
000028
csr@csr-VirtualBox:~/Desktop$ od -Ao -c input.txt
0000000   1   0   1  \n   1   0   2  \n   1   0   3  \n   1   0   4  \n
0000020   1   0   5  \n   1   0   6  \n   1   0   7  \n   1   0   8  \n
0000040   1   0   9  \n   1   1   0  \n
0000050
csr@csr-VirtualBox:~/Desktop$ od -Ad -c input.txt
0000000   1   0   1  \n   1   0   2  \n   1   0   3  \n   1   0   4  \n
0000016   1   0   5  \n   1   0   6  \n   1   0   7  \n   1   0   8  \n
0000032   1   0   9  \n   1   1   0  \n
0000040
```

```
csr@csr-VirtualBox:~/Desktop$ od -c -
welcome to rnsit
0000000   w   e   l   c   o   m   e       t   o       r   n   s   i   t
0000020  \n
0000021
csr@csr-VirtualBox:~/Desktop$ od -bc input.txt
0000000 061 060 061 012 061 060 062 012 061 060 063 012 061 060 064 012
          1   0   1  \n   1   0   2  \n   1   0   3  \n   1   0   4  \n
0000020 061 060 065 012 061 060 066 012 061 060 067 012 061 060 070 012
          1   0   5  \n   1   0   6  \n   1   0   7  \n   1   0   8  \n
0000040 061 060 071 012 061 061 060 012
          1   0   9  \n   1   1   0  \n
0000050
csr@csr-VirtualBox:~/Desktop$
```

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