

IGDTUW
OPERATING SYSTEM
LAB FILE

Submitted To:

Ms. Nidhi Arora

Submitted By:

Suhani Jain

I.T.-1 (Group-H)

Roll No.-06001032021

INDEX

S.No.	Topic	Date	Signature
1.	Name, version, vendor and target market place of OS	02.01.2023	
2.	Case study of LINUX,UNIX, Windows and their comparative analysis	09.01.2023	
3.	Important points regarding UNIX	16.01.2023	
4.	Important points regarding LINUX	20.01.2023	
5.	LINUX commands set 1	23.01.2023	
6.	LINUX commands set 2	30.01.2023	
7.	LINUX commands set 3	06.02.2023	
8.	LINUX command set 4 and more about LINUX	06.03.2023	
9.	Shell scripts	13.03.2023	

LAB EXERCISE-1

What is an Operating System?

An operating system is a piece of system software that controls computer hardware, software resources, and offers standard services to programmes on computers. It acts as an intermediary between a user of a computer and the computer hardware. OS is a **resource allocator** and **control program**.

S.No.	Name of the Operating System	Versions/Flavours/Distributions	Vendor	Target MarketPlace
1.	Microsoft Windows	Windows 1.0x, Windows 2.X, Windows 3.X, Windows NT, Windows NT 4.0, Windows 9x, Windows Me, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, Windows Server 2016, Windows Server 2019, Windows 11	Microsoft	PCs, Workstations

2.	Android	Tiramisu (OS 13), Snow Cone (OS 12), Red Velvet Cake (OS 11), Quince Tart (OS 10), Pie (9.0), Oreo (8.0), Nougat (7.0), Marshmallow (6.0), Lollipop (5.0), KitKat (4.4), Jellybean (4.3, 4.2 and 4.1), Ice Cream Sandwich (4.0), Honeycomb (3.0), Gingerbread (2.3), Froyo (2.2), Éclair (2.1), Donut (1.6)	Google	Touchscreen devices, cell phones, tablets
3.	Apple DOS	DOS 3.1, DOS 3.2, and DOS 3.3	Apple	Apple II series of microcomputers
4.	Linux	Fedora (Red Hat), openSUSE (SUSE) and Ubuntu (Canonical Ltd.), Debian, Slackware, Gentoo and Arch Linux, KaOS, Manjaro Linux, Kaisei Linux, Linux Lite, Alpine Linux, Alpine Linux 3.16	Unix Based	Server OS for web servers, database servers, file servers, email servers and other type of shared servers
5.	UNIX	SunOS, Solaris, SCO Unix, AIX, HP/UX, ULTRIX, NetBSD, and FreeBSD	Unix Based	Web servers, mainframes and supercomputers
6.	iOS	iPhone OS 1, iOS 6, iOS 9, iOS 11, iOS12, iOS 13, iOS15, iOS 15.7, iOS 16	Apple	iPhones, iPad Tablets
7.	macOS	Mac OS X 10.0(Cheetah), Mac OS X 10.1(Puma),	Apple	Mac desktops and laptops

		Mac OS X 10.2(Jaguar), Mac OS X 10.3(Panther), Mac OS X 10.4(Tiger), Mac OS X 10.5(Leopard), Mac OS X 10.6 (Snow Leopard), Mac OS X 10.7(Lion), OS X 10.8(Mountain Lion), OS X10.9(Mavericks), OS X 10.10(Yosemite), Mac OS X 10.11(EI Capitan), macOS(10.12, 10.13, 10.14, 10.15, 11, 12, 13)		
8.	Xenix	Version 7 UNIX, Xenix 2.0, Sperry, Xenix 3.0, Xenix 5.0	Microsoft	Various microcomputer platforms
9.	Chromium OS	109.0.5414.94, Stable, Long-term support (LTS), LTS candidate (LTC), Beta, and Dev	Google	Designed for running web applications and browsing the World Wide Web
10.	iRMX	iRMX I, II and III, iRMX-86, iRMX-286, DOS-RMX, iRMX for Windows, INtime.	Intel	Designed for use with the Intel 8080 and 8086 family of processors
11.	DOS	MS-DOS 1.x., MS-DOS 2. x., MS-DOS 3. x., MS-DOS 4.0, MS-DOS 4. x, MS-DOS 5. x., MS-DOS 6. x., MS-DOS 7/8 (as part of Windows 9x)	IBM	Provides file system for organising, reading, and writing files on storage disk
12.	MS-DOS	MS-DOS 1.x;	Microsoft	IBM PC

		MS-DOS 2.x; MS-DOS 3.x; MS-DOS 4.0 / MS-DOS 4.x; MS-DOS 5.x; MS-DOS 6.x		compatible personal computers
13.	Bada	Bada 2.0(2011), Bada 1.2(2010), Bada 1.1(2010), Bada 1.0(2010)	Samsung	Smartphones and tablet computers
14.	LiteOS	LiteOS 2.0, LiteOS 1.0,LiteOSV200R001 C50B039,LiteOSV20 0R001C50B038,Lite OSV200R001C50B03 5,LiteOSV200R001C 50B037,LiteOSV200 R001C50B036,LiteO SV200R001C50B033, LiteOSV200R001C50 B032,LiteOSV200R0 01C50B031,LiteOSV 200R001C50B030	Huawei	Used in IOT domains such as smart homes, wearables, Internet of Vehicles(IOV), intelligent manufacturing

LAB EXERCISE-2

CASE STUDY OF UNIX:

INTRODUCTION



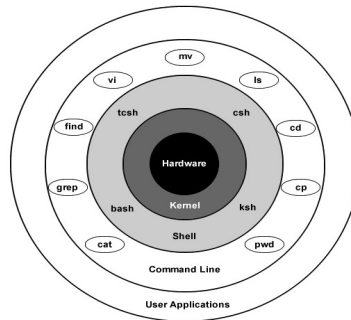
UNIX is an operating system that was created for the first time in the 1960s and has since undergone continuous development. When we refer to an operating system, we mean the collection of applications that run the computer. For servers, desktop computers, and laptops, it is a reliable, multi-user, multi-tasking system.

Additionally, UNIX systems include a graphical user interface (GUI) that is comparable to Microsoft Windows and offers a user-friendly environment. For tasks that cannot be

performed using a graphical software or when a windows interface is not accessible, such as during a telnet session, UNIX knowledge is necessary.

Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, and Joe Ossanna at Bell Labs are its developers.

ARCHITECTURE



The kernel, the shell, and user commands and applications are the three primary components of Unix. The operating system's kernel and shell are its heart and soul. The **kernel** processes user input through the shell and makes hardware accesses to carry out tasks like allocating memory and storing files. Monolithic, microkernel, and hybrid kernels are some of the several varieties. **The command line input** is interpreted by the **shell**, which then invokes the appropriate programmes to do the required tasks. The programmes you enter as commands will run when you submit them, returning the command line to a prompt where you can enter new commands. There are numerous distinct shells, and each one has a unique syntax and set of shortcuts. The "csh" shell, for instance, is known as "C shell," and it features syntax that is comparable to that of the C programming language. All shells have the same fundamental capabilities.

FEATURES

Various features of UNIX includes:

1. Multiuser System: Unix enables numerous programmes to run simultaneously and vie for the CPU's attention. Two things cause this:

- several users performing numerous tasks
- one user handling numerous tasks

Resources are truly shared by all users in UNIX, which is why the system is referred to as multi-user. To do this, the computer gives each user a time slice, which divides a unit of time into numerous segments. As a result, only one user is being served at any given time, despite the switching happening so quickly that it appears that all users are being served at once.

2. Multitask System: One user may carry out several tasks at once. An example would be simultaneously editing one file, printing it on the printer, emailing someone, and using the internet. The Kernel is made to cater to the various needs of users. The fact that just one job can be seen operating in the foreground and the others appear to be running in the

background is crucial in this situation. Users have the option to move between them and end or suspend any job.

3. The building-block method: When creating Unix, the developers considered keeping simple commands for each type of task. Unix offers several commands, each of which merely accomplishes a single straightforward task. Using the pipe symbol ('|'), you can execute two commands. Instance: `$ ls | wc` In this case, | (pipe) joins two instructions to form a pipeline. This command determines how many files are present in the directory. Filters are these interconnected commands that have the ability to filter or change data in different ways. Many UNIX tools today are created so that the output of one can be used as the input for another. By linking several tools, we can make a wide variety of combinations.

4. The UNIX Toolkit: Unix has a kernel, but the kernel by itself isn't very user-friendly. Thus, we must utilise the wide range of UNIX system-specific programmes. Numerous different uses are available. Tools for system management, networked programmes, compilers and interpreters, text manipulation utilities (referred to as filters), and general-purpose tools are all included. New tools are added and existing ones are changed or eliminated with each new UNIX release.

5. Pattern Matching: The pattern matching features offered by Unix are extremely complex. The system uses the meta-char '*' as a special character to match a variety of file names. In UNIX, there are numerous additional meta-chars. The matching process is not limited to filenames only. The characters from this set are used to frame a regular expression, which is used by more sophisticated tools.

6. Programming Facility: Unix offers shell, a programming language that is likewise intended for programmers rather than common consumers. It contains all the necessary control structures, loops, and variables for programming. The shell scripts are created using these characteristics (programmes that can invoke the UNIX commands). These shell scripts can manage and control a wide range of system operations.

7. Documentation: It features a command called "man" that stands for the manual, the most crucial source of information for all commands and the configuration files that go with them. There are a tonne of resources on the Internet in addition to the offline documentation.

HISTORY

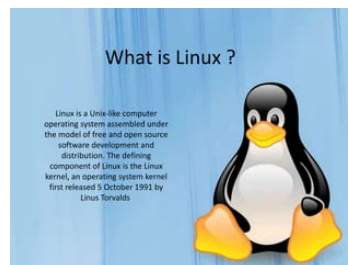
The Massachusetts Institute of Technology, AT&T Bell Labs, and General Electric collaborated to create Multics, an experimental time-sharing operating system, for the GE-645 mainframe in the middle of the 1960s. Multics made a lot of advances but also had a lot of issues. Bell Labs gradually withdrew from the project after becoming dissatisfied by the scope and complexity of Multics but not by its goals. Ken Thompson, Dennis Ritchie, Doug McIlroy, and Joe Ossanna were among their final researchers to quit Multics. They made the decision to redo the work, albeit on a much smaller scale. In 1979, Ritchie described the group's vision for UNIX.

DISTRIBUTION OF UNITS

Commercial and community-supported Unix distributions have a long history. Major commercial distributions frequently have a close relationship with hardware suppliers as they evolve. The BSD family of Unix distributions offers a broad ecosystem of community-developed, open-source operating systems that has a connection to the for-profit choices. Until recently, there were essentially just two varieties of Unix: AT&T's System V (five) and Berkeley Software Distribution (BSD). Other distributions of UNIX are SunOS, Solaris, SCO Unix, AIX, HP/UX, ULTRIX, NetBSD, and FreeBSD.

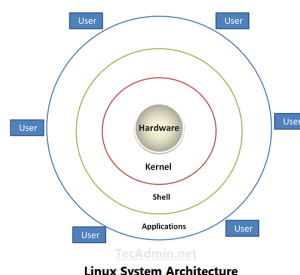
CASE STUDY OF LINUX:

INTRODUCTION



An open source operating system is Linux (OS). A system's hardware and resources, such as the CPU, memory, and storage, are directly managed by an operating system, which is a piece of software. The OS establishes links between all of your software and the working physical resources by sitting in between apps and hardware. An operating system or kernel that is made available under an open-source licence is called LINUX. Its feature set is very similar to UNIX. The Linux operating system's kernel is a piece of software that handles basic tasks including enabling hardware and software communication.

ARCHITECTURE



The Kernel, System Library, Hardware layer, System, and Shell utility make up the majority of the Linux operating system's architecture.

Kernel:- One of the fundamental components of an operating system is the kernel. It is in charge of all of the Linux OS's major operations. The kernel enables the necessary abstraction to shield the system from the specifics of low-level hardware or application programmes. The following list includes some of the significant kernel types:

- Monolithic Kernel

- Micro kernels
- Exo kernels
- Hybrid kernels

Libraries for systems: You can provide these libraries as some unique functions. These are used to implement operating system functionality and don't require kernel module code access rights.

System utility programmes: These are in charge of performing individual and specialised level tasks.

Hardware layer: The Linux operating system is equipped with a hardware layer that is made up of various peripherals, including a CPU, HDD, and RAM.

Shell:- It serves as a user-kernel interface. It is able to pay for kernel's services. It can accept commands from the user and executes kernel operations. There are various kinds of OSes that support the shell. These operating systems can be divided into two groups: command-line shells and graphical shells.

While command line shells support the command line interface, graphical line shells provide the graphical user interface. As a result, these two shells carry out operations. However, compared to command-line interface shells, graphical user interface shells operate more slowly.

There are a few types of these shells which are categorized as follows:

- Korn shell
- Bourne shell
- C shell
- POSIX shell

FEATURES

1.Free and Open-Source Software-

Costs are never a barrier to using Linux as an operating system because it is entirely free.

Linux is open source software. This means that anyone in the world is free to modify, analyse, redistribute, or sell copies of upgraded software as long as they do so in accordance with the same licence, which is also freely available.

2.Extremely Flexible-

Embedded items like watches, smart devices, and supercomputer servers all use extremely flexible Linux. Installing a full Linux outfit has no prerequisites. It enables a user to just install the components that are necessary for them.

3.Lightweight Infrastructure-

Linux requires between 4GB and 8GB of disc space for installation, but it uses less storage space overall. The memory footprint of the software, or the amount of memory (RAM) consumed while it is running, is also minimal, and it is compatible with all types of file formats, including text, audio, video, and graphic formats.

4.Graphical User Interface (GUI)-

Linux uses a command-line interface by default, but it can be modified to utilise a graphical user interface like Windows. Installing packages is the main method for doing this. Logging

into an Ubuntu server and installing its desktop environment is the most popular method for getting a GUI in a Linux system.

5.End-to-end encryption-

Linux supports data access with end-to-end encryption, saving public keys on the server. All information is password-protected and allows users to authenticate. Additionally, it offers a secure shell, file permissions, and many other security features.

6.Portable Environment-

Linux can run in any environment and is independent of how high- or low-end the hardware is. It can be used by many people at once on many different devices at any time. It can operate on any type of hardware. Linux also supports numerous distributions or businesses. To install the necessary programmes, one can use Linux's own software repository.

7.Shell/Command Line Interface-

The Shell command-line interpreter on Linux acts as a conduit between the user and the kernel, which in turn runs what are known as commands. As a result, Linux employs the command-line interface to carry out operations since it is quicker and more efficient to do so. Additionally, less memory space is required.

8.Hierarchical File System-

User files are organised in a clear directory structure in Linux, which has a well-defined file system. Binary directories, configuration directories, Data directories, memory directories, Usr (Unix System Resources), var (variable directory), and non standard directories are the different categories of folders based on the type of files they contain.

9.Multi-user and Multi-programming-

Linux enables simultaneous usage of numerous users and multiple applications, as well as simultaneous access to system resources.

HISTORY

Linus Torvalds, a student from Finland, started Linux in 1991 as a side project to develop a new kernel for a free operating system. The resulting Linux kernel has experienced continuous expansion over the course of its existence. Since its source code was first made available in 1991, it has expanded from a few C files distributed under a licence that forbade commercial distribution to the 4.15 version released in 2018 with more than 23.3 million lines of code (excluding comments) distributed under the GNU General Public License v2.

DISTRIBUTION OF UNITS

An operating system built on top of the Linux kernel is known as a Linux distribution, or distro for short. Distributions typically come with a sizable selection of free and open-source applications. Currently, there are more than 600 Linux distributions. Every distro offers common linux tools. Check out our links to popular distributions and library resources for information on how to install and run them. LINUX for desktop includes Ubuntu, Debian, Fedora, Linux Mint, Scientific Linux, Open Suse. Most popular Linux server oriented distributions are Ubuntu Server, Red Hat Enterprise Linux Server, SUSE Enterprise Linux Server, Oracle Linux, CentOS, Debian. Kali Linux and Backbox Linux are specialised distros and Arch Linux is advanced distro.

CASE STUDY OF WINDOWS:

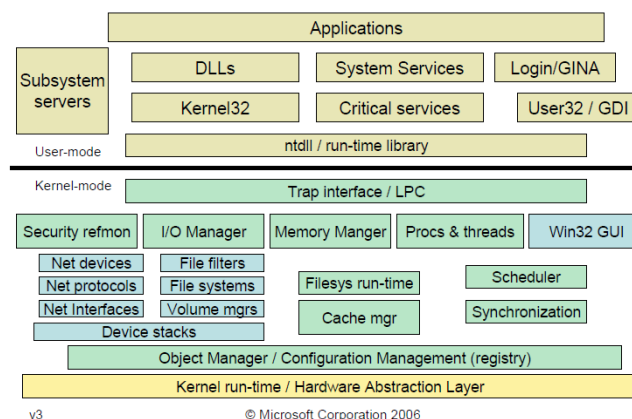
INTRODUCTION



Microsoft created the operating system called Windows. Windows is the most widely used operating system in the world since it comes preloaded on the majority of new personal computers (PCs). We can carry out all kinds of commonplace operations on your computer thanks to Windows. We can use Windows, for instance, to surf the web, check your email, edit digital images, play games, listen to music, and much more. As it provides access to productivity tools like calendars, word processors, and spreadsheets, Windows is also widely used in offices.

ARCHITECTURE

Windows Architecture



Windows NT, a range of operating systems created and offered by Microsoft, has a layered architecture made up of two key parts:

Kernel model and User model.

It is a preemptive, multitasking operating system that was created to operate on computers with symmetric multiprocessor (SMP) and uniprocessor architectures. They employ packet-driven I/O, which makes use of asynchronous I/O and I/O request packets (IRPs), to execute input/output (I/O) requests.

The system resources that programmes and subsystems in user mode can access are constrained, whereas kernel mode has full access to the system memory and external devices. In Windows NT, kernel mode has complete access to the computer's hardware and system resources. The hardware abstraction layer (HAL), drivers, and a variety of services (together referred to as Executive) are all present in kernel mode in the Windows NT kernel, which is a hybrid kernel.

In Windows NT, user mode is made up of subsystems that can use the I/O manager to send I/O requests to the proper kernel mode device drivers. The "Environment subsystems" of Windows NT's user mode layer run programmes created for a variety of operating systems, and the "Integral subsystem" manages system-specific operations on behalf of the environment subsystems. User mode services and apps cannot access crucial parts of the operating system without permission of the kernel.

FEATURES

The capabilities of the Windows operating system enable it to carry out tasks including accessing software, displaying commands on the screen, examining data, printing documents, and many other things. A graphical operating system called Windows was invented by Microsoft. Users can view and store files, run programmes, play games, watch videos, and connect to the internet using this device. It was made accessible for both private and business use. Any average person may simply utilise it thanks to its simple and intuitive user interface. In addition to being simple to access, it also offers excellent security. Some of the most important components of the Windows operating system are highlighted below:

1.Control Panel: The Windows operating system's Control panel is a feature that includes a number of tools for setting and managing the computer's resources.

2.Cortana: It is utilised to display the computer's files and folders. It also goes by the name Windows Explorer. It enables users to browse information on hard drives, SSDs, and other attached removable media. It enables the user to handle the material in accordance with their preferences, including the ability to search for and transfer data as well as remove or rename files.

3.Internet Browser: Accessing the internet and the web is one of the main uses of computers. In order to look for anything, an internet browser is therefore absolutely necessary. The Windows operating system comes with an internet browser pre-installed. The Edge internet browser has replaced Internet Explorer as the default browser on Windows 10 and later versions of the operating system.

4.Disk Cleanup: The Windows operating system's Disk Cleanup tool is used to clear up disc space by eliminating unused files or temporary files that are no longer needed. It improves system efficiency and gives users more storage space for downloading files and apps.

5.Speed: Microsoft put a lot of time and effort into making the Start Menu response in Windows 7 feel more brisk and snappy. Microsoft has also acknowledged the need for improved desktop responsiveness, which gives the impression that the user is in control and that the system is responding to them.

6.Hardware Requirements: Older systems will run Windows 7 without any issues, making the transition from Windows XP simpler.

7.Search and Organization: The improved search capability in Windows 7 is one of its best features. For instance, typing "mouse" will reveal the mouse option in the control panel, however typing a word will show it and neatly segment it into files, folders, and programmes. Additionally, the idea of Libraries, which builds on the idea of "My Documents," is introduced. You may keep everything in one place by using the various libraries, such as Documents and Pictures, which will keep an eye on multiple locations that you can add yourself.

8.Compatibility: On Windows XP, several commonly used consumer and commercial software packages required upgrades in order to function properly; however, with Windows 7, virtually all Vista-compatible software should continue to function.

9.Taskbar: Instead of having objects labelled with awkward wording, objects are grouped together. If you have a lot of Word documents or Windows Explorer windows open, a stack will show up on the taskbar. Each Window will show up as a thumbnail when your cursor is over the programme. Each thumbnail will become visible when your cursor is hovered over it, and all other open windows will disappear save for their outlines. From the thumbnail, each document or window can be closed instantly or brought to the front simply clicking on it. In the Start menu, a tiny arrow to the right of applications like Word now opens a list of recently seen documents that may be pinned to keep a particular document there indefinitely. The taskbar is a crucial component of the Windows operating system because of the aforementioned reasons.

10.Safety: The new security features in Windows include support for the "Secure Boot" functionality on UEFI systems to prevent malware from infecting the boot process, two new authentication techniques designed specifically for touchscreens, the addition of antivirus capabilities to Windows Defender Smart Screen filtering, and more.

11.Interface: Windows' user interface has seen significant upgrades. The new user interface is based on Microsoft's Metro design language and includes a Start screen for opening apps and desktop programmes, some of which can display continuously updated information and content via "live tiles." This Start screen is similar to that of Windows Phone. As a form of multitasking, apps can be snapped to the side of a screen. In addition to the traditional Control Panel, a new programme called "PC Preferences" is used for basic configuration and user settings. It is simplified and touch-optimised.

HISTORY

In response to the rising demand for graphical user interfaces, the first version of Windows was published on November 20, 1985, as a graphical operating system shell for MS-DOS (GUIs). According to StatCounter, Windows had a 75% market share for desktop operating systems worldwide as of April 2022.

DISTRIBUTION OF UNITS

Distribution Windows refers to all types of conventional distribution channels, such as network and syndicated television, pay television, home video, and theatre. Various distributions of Windows are Windows 1.0x, Windows 2.X, Windows 3.X, Windows NT, Windows NT 4.0, Windows 9x, Windows Me, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, Windows Server 2016, Windows Server 2019, Windows 11.

COMPARISON BETWEEN LINUX, UNIX AND WINDOWS:

PARAMETERS	UNIX	LINUX	WINDOWS
1.BASIC	It is a command based Operating system.	It is a command based Operating System.	It is a menu based Operating system.
2.LICENSING	It is an open source system which can be used to under General Public Licence.	The Linux kernel (and the GNU utilities and libraries which accompany it) in most distributions are entirely free and open source. Companies offer paid support for their distros, but the underlying software is still free to download and install.	It is a proprietary software owned by Microsoft.
3.USER INTERFACE	It has text based interface ,making it harder to gasp for newcomers.	Linux uses KDE and Gnome. Other GUI supported are LXDE, Xfce, Unity, Mate.	It has graphical user interface making it simple to use .
4.PROCESSING	It supports multiprocessing.	It supports multiprocessing.	It supports multithreading.
5.FILE SYSTEM	It uses unix file system UFS that comprises STD.ERR and STD.IO file systems.	It has more file system than UNIX system. In Linux, everything is treated like a file. Directories are files, files are files, and	It uses file allocation system (FAT32) and new technology file (NTFS).

		externally connected devices (such as Printer, mouse, keyboard) are also files.	
6.SECURITY	It is more secure as all changes to the system require explicit user permission. It has about 85-120 viruses listed till date.	It is also more secure as all changes to the system require explicit user permission. Linux has about 60-100 viruses listed till date.	It is less secure compared to unix and linux.
7.DATA BACKUP AND RECOVERY	It is tedious to create a backup and recovery.	It contains less redundant data and recovery is easy but the time needed for creating backup is high.	It has an integrated backup and recovery.
8.HARDWARE	Hardware support is limited in UNIX system.	Hardware support is limited in LINUX system. the Linux user base has grown exponentially. Today, most hardware manufacturers give Linux support the same priority as Microsoft Windows.	Drivers are available for almost all the hardwares.
9.RELIABILITY	UNIX and its distributions are well known for being very stable to run.	Linux is notoriously reliable and secure. It has a strong focus on process management, system security, and uptime. Users usually experience less issues in Linux.	Although windows has been stable in recent years, it is still to match the stability provided by UNIX system.
10.CASE SENSITIVITY	It is fully case sensitive and files can be considered separate files.	Linux file system treats file and directory names as case-sensitive. FOO.txt and foo.txt will be treated as	It has case sensitivity as an option.

		distinct files.	
--	--	-----------------	--

LAB EXERCISE-3

INSTALLATION PROCESS OF UNIX:

It is necessary to have a computer with a blank hard drive and a CD or DVD drive in order to install a Unix operating system. A copy of the operating system that you want to install is also required. Following these instructions, you can start the installation process once you have all the required materials:

1. Place the Unix operating system's CD or DVD in the computer's optical drive.
2. Use the optical drive to start the computer. You may accomplish this on the majority of PCs by pressing a key during bootup, such as F12, which will display a boot menu. Pick the CD or DVD to boot from option.
3. The installer for the Unix operating system will now load when the computer boots. To choose your language, keyboard layout, and other choices, follow the on-screen instructions.
4. The installation's destination will then be your next choice to make. Make a decision regarding the hard drive that you want to install the operating system on.
5. Now that the necessary files have been copied to the hard drive, the installer will install the operating system. When the installation is finished, the computer will need to be restarted.
6. The login screen will appear after the computer has restarted. To log in and utilise your new Unix operating system, enter the username and password you created during the installation procedure.

In only a few easy steps, Unix may be installed on **Windows**. The Unix installer should first be downloaded from the internet. Double-click the installer after that, then adhere to the instructions. After a final reboot, you should be able to utilise Unix on your Windows PC.

BOOTING PROCESS OF UNIX:

The process of booting is how a computer powers on. It can be started by software or hardware, like pressing a button. The process of initialising and testing hardware as well as loading and launching an operating system constitutes booting. For Unix-like systems, the boot process can be broken down into a number of stages, each of which completes a particular task.

The written memory format known as the BIOS (Basic Input Output System) makes it simple for you to access. To access the BIOS from the first hard drive's sector, a Master Boot Record (MBR) is necessary. The MBR includes instructions for loading a pre-configured operating system with a GRUB boot-loader, also referred to as a LILO boot-loader. If no entries are entered, it displays a splash screen, waits a short while, and then loads the default kernel image. Both boot methods are supported by GRUB, so you may use it almost anywhere.

When it first came out, the Linux loader LILO didn't comprehend the filesystem. While the Linux operating system is booting, a root file system is mounted using Initrd.

All background processes are started by the init software by looking for directories that have the default runlevel provided in /etc/inittab. The default run levels are as follows. # 0 – halt (this is NOT the default value in init) (this is NOT the default setting in init). In the beginning, you will be operating in single-user mode. Without networking, the second multiuser mode is available. A networking-based multiuser mode is the third.

LOGIN AND SHUTDOWN PROCESS OF UNIX:

Login process:

The process that takes place each time a user logs into a UNIX computer system is detailed in the steps that follow:

1. Usernames are entered by users.
2. User inputs password.
3. Your username and password are verified by the operating system.
4. A "shell" is created for you based on your entry in the "/etc/passwd" file (in small businesses, this is usually a Bourne Shell).
5. Your "home" directory is "put" with you.
6. The "/etc/profile" file is read for startup information. The system login file is the name of this file. Each user reads the data in this file when they log in.
7. More data is retrieved from the ".profile" file, which is kept in your "home" directory. The name of this file is "personal login file." The "menu" programme is typically found in this file.

Shutdown process:

You will occasionally need to turn the system off. This is required for carrying out planned maintenance, starting diagnostics, adding or changing hardware, and other administrative duties. In Unix, the shutdown command is available. Shutdown typically notifies all logged-on users via a series of timed messages that the system is shutting down; after sending the final of these messages, it logs all users off and switches the system to single-user mode.

The shutdown command comes in two major forms. Both Solaris and HP-UX use the System V version, with the latter somewhat deviating from the standard, while AIX, FreeBSD, Linux, Solaris (in /usr/ucb), and Tru64 use the BSD version.

The System V shutdown Command

The standard System V shutdown command has the following form:

```
# shutdown [-y] [-g  
    grace] [-i  
    new-level] message
```

where new-level defines the new run level to place the system in (the default is single-user mode), grace specifies the amount of seconds to wait before commencing the procedure (the

default is 60), and message is a text message sent to all users. On systems running Solaris, this form is employed.

The BSD-Style shutdown Command

BSD defines the shutdown command with the following syntax:

```
# shutdown [options] time message
```

where time can have three forms:

+m

Shut down in m minutes.

h:m

Shut down at the specified time (24-hour clock).

now Begin the shutdown at once.

now should be used with discretion on multiuser systems.

Alternative choices offer more variations on the system shutdown procedure:

shutdown -r instructs the system to reboot right away once it shuts down. The reboot command accomplishes the same task.

Instead of switching to single-user mode, shutdown -h instructs the computer to halt the processor. When this procedure is finished, the electricity can be shut off without risk. Once single-user mode has been reached, the CPU can also be explicitly stopped using the halt command.

The shutdown messages are sent out regularly, but no actual shutdown takes place when the system is started with shutdown -k. Theoretically, you can scare users away from the system in this method, however some users can be fairly tenacious and would sooner be murdered by shutdown than log out.

EXTERNAL AND INTERNAL COMMANDS OF UNIX:

UNIX commands are classified into two types-

- **Internal Commands:** The shell's internal commands. All built-in commands in the shell can be executed quickly because they don't need to be found in the PATH variable by the shell, nor do they require the spawning of a new process. Examples are source, cd, and fg.
- **External Commands:** Commands which aren't built into the shell. When an external command has to be executed, the shell looks for its path given in the PATH variable, and also a new process has to be spawned and the command gets executed. They are usually located in /bin or /usr/bin. For example, when you execute the "cat" command, which usually is at /usr/bin, the executable /usr/bin/cat gets executed. Examples: ls, cat etc.

SHELLS IN UNIX:

You have access to the UNIX system through the shell. You provide input, and it uses that input to run programmes. The output of a programme is shown once it has completed running.

We may execute our commands, applications, and shell scripts in a shell environment. Similar to how there are various operating systems, there are various shells. Each type of shell has a unique set of widely used commands and features.

Shell Prompt: The shell issues the command prompt, or prompt, known as \$. You can type a command while the prompt is visible. After you press Enter, the shell reads what you entered. By examining the first word of your input, it determines which command you wish to perform. A word is a continuous string of letters. Words are divided by spaces and tabs. Following is a simple example of date command which displays current date and time: `$date`
Shell Types: In UNIX there are two major types of shells:

The Bourne shell- The default prompt when using a Bourne-type shell is the \$ symbol. Some categories are Bourne shell (sh), Korn shell (ksh), Bourne Again shell (bash) and POSIX shell (sh)

The C shell- The default prompt when using a C-type shell is the % symbol. Some categories are C shell (csh) and TENEX/TOPS C shell (tcsh)

Stephen R. Bourne created the first UNIX shell in the middle of the 1970s while working at AT&T Bell Labs in New Jersey. The Bourne shell is referred to as "the shell" because it was the first shell to debut on UNIX systems.

On most UNIX systems, the Bourne shell is often installed as /bin/sh. This makes it the preferred shell for creating scripts that can run on several UNIX systems.

FILE SYSTEMS AND DIRECTORY STRUCTURE OF UNIX:

Applications and utilities are all saved as files. It is a rational way to arrange and keep track of a lot of data. Users of UNIX can store and retrieve data using files in the file system, which is a hierarchical structure of files and directories.

In the UNIX system, every file is connected to every other file.

The UNIX File system's Directory structure is described below:

Root (/): Root is the node at the top of the directory hierarchy. It is also known as the parent directory since it houses all of the UNIX file system's subdirectories. In the UNIX operating system, it is denoted by the slash symbol.

/lib: This directory includes all the details on how the system libraries work as well as a few crucial items like kernel modules or device drivers. Additionally, it includes whatever system calls the compiler is permitted to include in a programme.

/bin: The binary files for the system as well as certain essential utilities are located in this directory. It serves as the directory for administrative-level commands like ls and cp. This directory is always displayed in the list by the path (var).

EDITORS IN UNIX:

The initial UNIX text editor is called d.

Start it by entering ed. An interactive session is now underway. By typing a on a single line and pressing enter, you can enter write mode. Enter after writing simply a dot (.) on a line after you have finished typing everything you wish to.

To save the buffer to a file, type `w` and then the filename. The amount of bytes that were written to the file will be returned.

Then, you can press `Q` to stop.

By calling `ed` with the file name `ed`, you can edit a file. You add content to the bottom of the file when you use the "add" key.

To print during an `ed` session, type `p`.

vi / vim

A well-liked file editor is `vim`. Modern operating systems just use `vi` as an alias for `vim`, hence `vi` has advanced.

To edit a particular file, you can give a filename at the time of execution:

`test.txt` in `vi`

You should be aware that `Vim` has two primary modes:

command mode or standard mode and place mode. You're in command mode when you open the editor. Text cannot be entered the way you would expect it to in a GUI-based editor. You must switch to insert mode. In order to achieve this, simply press the `I` key. When you do, the word — INSERT — will show up at the editor's bottom: The arrow keys and the `h- j- k- l` keys can be used to navigate the file. `j-k` for down-up, `h-l` for left-right.

Press the `esc` key to leave insert mode and return to command mode once you're done editing.

You can navigate the file at this point but cannot add content to it. You might wish to start by saving the file right now. In order to do so, hit (colon), then `w`.

By pressing `w` and `q`, you may save and stop.

By pressing `q` and `!::q`, you can exit without saving.

By entering command mode and pressing `u`, you can edit and undo. By hitting `CTRL-R`, you can undo something (cancel an undo).

emacs

The preferred editor for UNIX systems is `emacs`. By simply calling `emacs`, a new `emacs` session can be started:

`Emacs filename>` can be used to modify an existing file as well.

You can begin editing and then end it by pressing `CTRL-X` and then `CTRL-W`. `Emacs` asks you if it should overwrite the file after you confirm the folder exists. You receive a success confirmation when you respond with `y`. Pressing `CTRL-X` and then `CTRL-C` will close `Emacs`. Or use `Ctrl-X`, then `C` (keep `ctrl` pressed).

nano

`Nano` is a more user-friendly editor for newcomers.

Use `nano` to run `filename>`.

You don't need to bother about modes if you just type characters directly into the file.

`Ctrl-X` is used to exit without making any changes. If you made changes to the file buffer, the editor will prompt you for approval before letting you decide whether to keep the changes or not.

LAB EXERCISE-4

INSTALLATION PROCESS OF LINUX

Installing LINUX using USB stick

1) Download the necessary files.

Install the OS files or the.iso file on your machine.

2)Download the Universal USB Installer in step two. To create a bootable USB stick, download free software like Universal USB installer.

3)Select Distribution.

To put Ubuntu on your USB, choose a distribution from the dropdown menu.

In step 1, choose the Ubuntu ISO file that you downloaded.

To install Ubuntu, choose the USB drive letter and click the "Create" button.

4)Set up Ubuntu.

To install Ubuntu on a USB, select YES.

5)Examine the window.

A tiny window will show up following installation and configuration of everything.

Installing LINUX on top of Windows

1)Open the Windows Features window. To do this, use the taskbar to look for and then choose "Turn Windows features on or off."

2)From the features list, select "Windows Subsystem for Linux." Be sure to check that box.

3)Select OK. This will update your system with the new feature changes.

4)When prompted, restart your computer. You must restart your computer in order to enable the new hypervisor. Windows must adjust a few settings in order for WSL to function on your computer.

5)Refresh WSL. You should update WSL to the most recent version before using. Run `wsl —update` in a PowerShell or Command Prompt window that is elevated to accomplish this.

Search for "Command Prompt," "PowerShell," or "Windows Terminal," select "Run as administrator," and acknowledge the elevation prompt to launch an elevated terminal. A security shield will appear in Windows Terminal following a successful elevation.

6)Turn off WSL. You must end the process in order for the updated WSL to operate. The command `wsl —shutdown` can be used to accomplish this. You can start your Linux distribution and get the most recent features once WSL has been terminated.

7)Install a Linux operating system. By using the command `wsl —install`, you can easily install the default (Ubuntu). Use the `wsl —install distribution>` command to install a different Linux distribution.

You can also choose a distribution of your choice by going to the Microsoft Store.

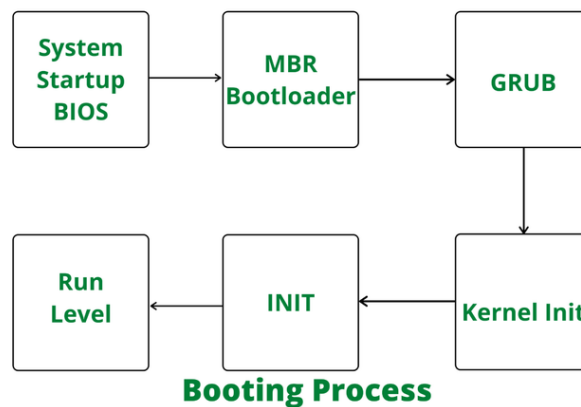
8)Make a user account on that distribution. The account will only be used with that one specific distribution. A separate distribution will need its own credentials if you install it. When prompted, enter a username and choose a password. Following that, you are ready to use that specific distribution.

BOOTING PROCESS OF LINUX

When we push the system's power button, numerous processes are active in the background. To comprehend how any operating system functions, it is crucial to grasp the booting process.

Stages of LINUX boot process:

- 1)The boot loader is controlled by the machine's BIOS or boot microcode.
- 2)In order to start the system, the boot loader locates the kernel image on the disc and loads it into memory.
- 3)The devices' drivers are initialised by the kernel.
- 4)The kernel mounts the filesystem used as a base.
- 5)The remaining system processes are started by the kernel by launching the init programme, which has method ID zero.
- 6)The remaining system processes are started by init.
- 7)For whatever reason, init launches a procedure that enables you to log in, usually at the very beginning or very close to the beginning of the boot sequence.



Startup Message

Upon boot, traditional UNIX operating systems generate a number of diagnostic messages that provide information on the boot process. Initial messages are returned by the kernel, then by processes and low-level formatting operations that init starts. These signals, meanwhile, aren't particularly attractive or reliable, and in some situations, they're not even all that instructive. Additionally, hardware improvements have made the kernel start much quicker than previously. Because of this, it may be difficult to keep track of what's happening as the messages flash past so quickly. As a result, the majority of modern Linux distributions try to conceal the boot medical speciality with splash screens and other types of filler to occupy your attention while the system boots.

Kernel Initialization and Boot Options:

1. CPU examination
2. Memory examination
3. Device bus discovery
4. Device discovery
5. Auxiliary kernel system setup
6. Root filesystem mount
7. Userspace begin

Kernel Parameters: The Linux kernel receives a set of text-based kernel parameters containing additional system information when it starts. The parameters define a wide range of possible behaviours, including the quantity of diagnostic outputs the kernel should provide and driver-specific options. When the user area starts, the Ro parameter informs the kernel to mount the basic filesystem in read-only mode. Before doing anything significant, fsck can securely verify the basis filesystem thanks to its standard read-only mode. The bootup method remounts the basic filesystem in read-write mode during the inspection.

Tasks of the boot loader:

- Pick from a variety of kernels.
- Change between different kernel parameter sets.
- Support the booting of several operating systems.

One astounding change is how much some wants have decreased. For instance, you rarely have to worry about manually entering kernel parameters or switching to single-user mode because you can perform a recovery or emergency boot from a USB drive. Modern bootloaders offer more power than ever, which might be especially useful if you're creating custom kernels or just want to change some settings.

Boot loader Overview

- GRUB, which comes in BIOS/MBR and UEFI variants, is almost often the default on Linux computers.
- One of the main bootloaders for Linux is called LILO. A UEFI version of ELILO might exist.
- SYSLINUX - It can be set up to run from a variety of different filesystem designs.
- A kernel is booted from DOS by LOADLIN.
- Simple UEFI boot manager for system boot.
- Coreboot is a BIOS replacement for computers that is superior and includes a kernel.
- Kernel for Linux EFISTUB- a kernel plug-in that enables direct kernel loading from a connected EFI/UEFI System Partition.
- A UEFI boot loader designed to serve as a template and point of reference for other UEFI boot loaders is called EFI Linux.

Grub introduction and work: Grand Unified Boot Loader is referred to as GRUB. The ability to navigate filesystems, which enables simple kernel image and configuration selection, is one of GRUB's most important features.

- The BIOS scans for the boot code and then runs it. This is frequently where GRUB starts.
- The GRUB core hundreds.
- The core starts up. GRUB will currently access filesystems and drives.
- The boot partition that GRUB uses and the several configurations there are identified.
- The user has the option to change the configuration thanks to GRUB.
- After a timeout or user action, GRUB runs the configuration.
- During configuration execution, GRUB may load additional code from the boot partition. Additionally, several of these modules are preloaded.
- Boot commands are executed by GRUB in order to load and run the kernel.

LOGIN AND SHUTDOWN PROCESS OF LINUX

Login process:

The steps are:

- Init starts getty process
- getty process initiates login prompt on terminal
- login command check user credentials from /etc/passwd
- getty starts user shell process
- shell reads the system wide files /etc/profile, /etc/bashrc
- Shell reads user specific files .profile, .login
- Now it reads shell specific configuration file .bashrc
- Shell displays the default prompt

Init starts getty: The boot process for a Linux machine involves several stages. In the final stage, it starts init, which consults a file called inittab in the /etc directory to determine the appropriate run level for execution. Getty will be launched by the init process once run-level operations and commands in /etc/rc.local have been completed. The procedure that will handle the entire login process is Getty.

Init starts getty: The boot process for a Linux machine involves several stages. In the final stage, it starts init, which consults a file called inittab in the /etc directory to determine the appropriate run level for execution. Getty will be launched by the init process once run-level operations and commands in /etc/rc.local have been completed. The procedure that will handle the entire login process is Getty.

Getty shows login prompt: Get Terminal is abbreviated as "getty." The login command is started by a getty programme, which also opens the terminal device, sets it up, prints login:, and waits for the user name to be entered.

Getty starts /etc/login: User password will be requested when user inputs login name into getty begins /etc/login. The user-entered password will not be visible on the screen and will be concealed.

#paraphrase

Getty verifies the credential and starts users shell: In next stage getty checks the user credentials by verifying it with /etc/passwd and /etc/shadow file, if password matches then it initiates user properties gathering and starts users shell. If the password doesn't match then getty terminates the login process and re-initiates again with a new login: prompt.

In the next stage the getty process reads the user properties (username, UID, GID, home directory, user shell etc.) from /etc/passwd file. After that it reads /etc/motd file for displaying content banner messages.

Shell reads system wide default files and specific default files: For getting shell related properties, user aliases and variables getty process reads appropriate system wide default files /etc/profile or /etc/bashrc . After the system wide default files are read the shell reads user specific login files .profile or .login .

Shell specific file read: At last stage it reads shell specific configuration files (.bashrc, .bash_profile etc. for bash shell) of that user which it gets on the users home directory.

Shell prompt: When all startup files are read the shell displays the default prompt, normally PS1 prompt for user to execute their commands, here user can type any command to execute.

Shutdown process:

The Linux shutdown Command:

The shutdown command present on the majority of Linux systems also contains a -t argument that can be used to define how long should pass between the time the kernel sends the TERM signal to all running processes and the KILL signal. 30 seconds is the standard. The system is shut down more quickly with the following command, which leaves just 5 seconds between the two signals:

```
# shutdown -h -t 5 now
```

The shutdown command also has a minimal security mechanism provided by the -a option in the command version. The command checks to see if any of the users specified in the /etc/shutdown.allow file are currently logged into the terminal when it is called with this option (or any virtual console attached to it). The shutdown command fails if it does not.

This option is there to stop casual users from accidentally rebooting the system by pressing Ctrl-Alt-Delete on the console. As a result, it appears most frequently in the inittab entry related to this event.

EXTERNAL AND INTERNAL COMMANDS OF LINUX

LINUX commands are classified into two types-

- **Internal Commands:** Internal commands that are part of the shell are known as built-in commands. The shell is used to invoke built-in commands, which are then run within the shell. With the aid of the 'help' and 'compgen -b' commands, you can view a list of all built-in commands. Examples of built-in commands include "pwd," "help," "type," "set," and "unset."
- **External Commands:** Built-in commands are different from external commands. These commands are executable programmes that are stored in the filesystem as their own binary. These are the system-provided commands that are completely independent of the shell. These commands are typically found in /bin, /sbin, and /usr/sbin.

SHELLS IN LINUX

An operating system user interface is provided by a programme called SHELL. OS launches a user shell as soon as the user logs in. The kernel allocates resources amongst processes, restricts access to hardware, organises all running utilities, and controls all fundamental computer activities. Only the user has access to the operating system's utilities through the kernel. Types of Shell:

The C Shell –

The C Shell is represented by the symbol `csh`. It was developed by Bill Joy at the University of California, Berkeley. It included functions like command history and aliases. It has helpful programming capabilities like built-in math and expression syntax that is similar to C.

In the C shell, the default prompt for non-root users is `hostname%`, and the default prompt for root users is `hostname#`. The command full-path name is `/bin/csh`.

The Bourne Shell –

The Bourne Shell is abbreviated `sh`. Steve Bourne at AT&T Bell Labs wrote it. This shell is the first for UNIX. It is quicker and more popular. It lacks interactive features like the capacity to remember prior commands. Additionally, it lacks built-in expression support for logic and math. It is the Solaris OS's default shell. The Bourne shell's command full-path names are `/bin/sh` and `/sbin/sh`, and the default prompt for non-root users is `$` whereas the default prompt for root users is `#`.

The Korn Shell

It's written as `ksh`. At AT&T Bell Labs, David Korn wrote it. The Bourne shell is a superset of this. As a result, it supports all of Bourne shell. It includes interactive elements. It has functions, arrays, and string manipulation tools that are similar to those found in C, as well as built-in arithmetic. It outperforms C shell in speed. It works with C shell script that has been written. Regarding the Korn shell,

The name of the command is `/bin/ksh`, the default prompt for non-root users is `$`, and the default prompt for root users is `#`.

GNU Bourne-Again Shell –

Denoted as `bash`. It works with the Bourne shell. It has elements borrowed from Bourne shell and Korn. In relation to the GNU Bourne-Again shell, The entire path name of the command is `/bin/bash`, and the default prompt for users who are not root is `bash-g.gg$` (`g.gg` denotes the shell version number, such as `bash-3.50$`).

FILE SYSTEMS AND DIRECTORY STRUCTURE OF LINUX:

A Linux operating system's built-in layer known as the file system is typically utilised to manage the storage's data. Arranging the file on the disc storage is helpful. It controls a file's name, size, creation date, and a great deal more information.

The following divisions make up the Linux file system:

- `(/)` is the root directory.
- a particular format for storing data (EXT3, EXT4, BTRFS, XFS and so on)
- a logical volume or partition with a certain file system.

Similar to a tree, the Linux directory structure. The root represents the starting point of the Linux file system hierarchy. Although everything begins at the root, directories branch off of

it. The forward slash (/) is used in Linux as a directory separator. Forward slash is shortened to "slash" while discussing directories and speaking directory paths.

EDITORS IN LINUX:

Linux text editors are useful for a variety of tasks, including editing text files, creating code, updating user manuals, and more. Multiple text editors are supported on a Linux system. In Linux, there are two different categories of text editors, as follows:

- **Command-line text editors** such as Vi, nano, pico, and more.
- **GUI text editors** such as gedit (for Gnome), Kwrite, and more.

Using a text editor is crucial when programming. Therefore, choosing the best text editor is crucial. Not only should a text editor be easy to use, but it should also be practical and enjoyable to use. One of the best text editors is one that has IDE functionality.

Here are a few examples of text editors:

Atom editor, Brackets editor, Pico editor, Bluefish, Kate/Kwrite, Notepad ++, Eclipse, gVIM editor, Jed editor, Geany editor, Leaf Pad, Light Table, Medit text editor, CodeLite, Nano editor, Sublime text editor, GNU emacs.

LAB EXERCISE-5

LINUX COMMANDS SET-1

1.who

Usage/Function : who [OPTION]... [FILE | ARG1 ARG2] Print information about users who are currently logged in.

1. Time of last system boot
2. Current run level of the system
3. List of logged in users and more.

Syntax : \$who [options] [filename]

Examples : 1. The who command displays the following information for each user currently logged in to the system if no option is provided : 1. Login name of the users 2. Terminal line numbers 3. Login time of the users in to system 4. Remote host name of the user

```
hduser@mahesh-Inspiron-3543:~$ who hduser tty7 2018-03-18 19:08 (:0)
```

```
hduser@mahesh-Inspiron-3543:~$
```

2. To display host name and user associated with standard input such as keyboard

```
hduser@mahesh-Inspiron-3543:~$ who -m -H NAME LINE TIME COMMENT
```

3. To show all active processes which are spawned by INIT process

```
hduser@mahesh-Inspiron-3543:~$ who -p -H NAME LINE TIME PID COMMENT
```

4. To show status of the users message as +, - or ? `hduser@mahesh-Inspiron-3543:~$ who -T -H NAME LINE TIME COMMENT` `hduser + tty7 2018-03-18 19:08 (:0)`
 5. To show list of users logged in to system `hduser@mahesh-Inspiron-3543:~$ who -u` `hduser tty7 2018-03-18 19:08 01:16 3357 (:0)`
 6. To show time of the system when it booted last time `hduser@mahesh-Inspiron-3543:~$ who -b -H NAME LINE TIME PID COMMENT` `system boot 2018-03-18 19:07`
 7. To show details of all dead processes `hduser@mahesh-Inspiron-3543:~$ who -d -H` (NO dead process in this case) `NAME LINE TIME IDLE PID COMMENT EXIT NAME LINE TIME IDLE PID COMMENT EXIT`
 8. To show system login process details `hduser@mahesh-Inspiron-3543:~$ who -l -H` `NAME LINE TIME IDLE PID COMMENT LOGIN` `tty1 2018-03-18 19:07 3073 id=tty1`
 9. To count number of users logged on to system `hduser@mahesh-Inspiron-3543:~$ who -q -H` `hduser # users=1`
 10. To display current run level of the system `hduser@mahesh-Inspiron-3543:~$ who -r` `run-level 5 2018-03-18 19:07`
 11. To display all details of current logged in user `hduser@mahesh-Inspiron-3543:~$ who -a` `system boot 2018-03-18 19:07 LOGIN` `tty1 2018-03-18 19:07 3073 id=tty1 run-level 5 2018-03-18 19:07` `hduser + tty7 2018-03-18 19:08 01:13 3357 (:0)`
 12. To display system's username `hduser@mahesh-Inspiron-3543:~$ whoami` `hduser`
 13. To display list of users and their activities `hduser@mahesh-Inspiron-3543:~$ w` `20:39:20 up 1:32, 1 user, load average: 0.09, 0.06, 0.07` `USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT` `hduser tty7 :0 19:08 1:32m 38.95s 0.19s /sbin/upstart -`
 14. To display user identification information `hduser@mahesh-Inspiron-3543:~$ id` `uid=1001(hduser) gid=1001(hadoop) groups=1001(hadoop), 27(sudo)`
- Flags/Options :** -a, --all same as -b -d --login -p -r -t -T -u -b, --boot time of last system boot -d, --dead print dead processes -H, --heading print line of column headings -l, --login print system login processes --lookup attempt to canonicalize hostnames via DNS -m only hostname and user associated with stdin -p, --process print active processes spawned by init-q, --count all login names and number of users logged on -r, --runlevel print current runlevel -s, --short print only name, line, and time (default) -t, --time print last system clock change -T, -w, --mesg add user's message status as +, - or ? -u, --users list users logged in --message same as -T --writable same as -T --help display this help and exit --version output version information and exit
- Arguments :** 1. Command to display the hostname and user associated with the input/output devices like a keyboard
- Syntax 1. Who -m -H
 2. To display all details of currently logged in users With this command's help, one sees all the details of every user logged in to the current system. The syntax of this command is the same except the additional option "-a", as we can see in the given syntax: Syntax 1. who -a Or You can use "who -all" instead of "who -a" as it displays all information. Syntax 1. Who -all
 3. To display information about all active processes that are spawned by the NIT process This command will help you to display essential information as well as each and every active process. Syntax 1. who -p -h
 4. To display the status of the user's message as -, + or? This command will help us to display the status of the user's message. Syntax of this command given below: Syntax 1. who -T -H

5. To display the whole list of logged-in users This command will help us to display the whole list of the logged-in users. Syntax of this command given below: Syntax 1. `who -u`
6. To display the whole list of dead processes One can use this command to see the complete list of all dead processes. The syntax of the command given below: Syntax 1. `who -d -H`
7. To display system login process details One can use this command to see the login process. The syntax of the command is given below: Syntax: 1. `who -l -H`
8. To count the numbers of all logged-in users We can use this command to see how many users logged in in the form of numbers. The syntax of the command is given below: Syntax: 1. `who -q -H`
9. To display the current run level of the system The syntax of the command is given below: 1. `Who -r`
10. To display the system's username This command is generally used to know about the actual system's username. The syntax of the command is given below: Syntax 1. `Whoami`

Final Output :

```
$ who
webmaster pts/14      Jan 23 05:42
webmaster pts/1380    Jan 23 06:07
webmaster pts/871     Jan 23 06:13
```

2.cal

Function: If a user wants a quick view of the calendar in the Linux terminal, `cal` is the command for you. By default, the `cal` command shows the current month calendar as output. `cal` command is a calendar command in Linux which is used to see the calendar of a specific month or a whole year. Usage: `cal [options] [[[day] month] year]` `cal [options]` Display a calendar, or some part of it. Without any arguments, display the current month.

Syntax: `cal [[month] year]` The rectangular bracket means it is optional, so if used without an option, it will display a calendar of the current month and year.

Arguments: `cal -y` : Shows the calendar of the complete current year with the current date highlighted.

```
ubuntu@sanju6890:~$ cal -y
2021
January February March
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
 3 14 15 16 7 1 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

April May June
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
 4 15 16 17 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
18 19 20 21 22 23 24 25 26 27 28 29 30 31

July August September
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
 4 15 16 17 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
18 19 20 21 22 23 24 25 26 27 28 29 30 31

October November December
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
 3 14 15 16 17 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
```

`cal 08 2000` : Shows calendar of selected month and year.

```
dharam@dharam-H110MHC: ~
dharam@dharam-H110MHC:~$ cal 08 2000
August 2000
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
```

cal 2018 : Shows the whole calendar of the year.

cal 2018 | more : But year may not be visible in the same screen use more with cal use spacebar to scroll down.

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

cal -j : Shows the calendar of the current month in the Julian calendar format not in the default Gregorian calendar format. In Julian calendar format, the date does not reset to 1 after every month's end i.e. after 31st Jan, Feb will start as 32nd Feb, not as 1st Feb. But in the Gregorian calendar format, the date is reset to 1 after every month's end i.e after 31st Jan, Feb will start as of 1st Feb

Flags/Options: Options: -1, --one show only a single month (default) -3, --three show three months spanning the date -n, --months show num months starting with date's month -S, --span span the date when displaying multiple months -s, --sunday Sunday as first day of week -m, --monday Monday as first day of week -j, --julian use day-of-year for all calendars --reform Gregorian reform date (1752|gregorian|iso|julian) --iso alias for --reform=iso -y, --year show the whole year -Y, --twelve show the next twelve months -w, --week[=] show US or ISO-8601 week numbers --color[=] colorize messages (auto, always or never) colors are enabled by default -h, --help display this help -V, --version display version

Final Output :

```
$ cal
January 2023
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
```

3.date

Function: date command is used to display the system date and time. date command is also used to set date and time of the system. By default the date command displays the date in the time zone on which unix/linux operating system is configured.You must be the superuser (root) to change the date and time.

Syntax: date [OPTION]... [+FORMAT]

date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

Flags/Options:

1: **date (no option)** : With no options, the date command displays the current date and time, including the abbreviated day name, abbreviated month name, day of the month, the time separated by colons, the time zone name, and the year.

Command:

```
$date
```

Output:

```
Tue Oct 10 22:55:01 PDT 2017
```

2: **-u Option:** Displays the time in GMT(Greenwich Mean Time)/UTC(Coordinated Universal Time)time zone.

Command:

```
$date -u
```

Output :

```
Wed Oct 11 06:11:31 UTC 2017
```

3: **–date or -d Option:** Displays the given date string in the format of date. But this will not affect the system's actual date and time value. Rather it uses the date and time given in the form of string.

Syntax:

```
$date --date=" string "
```

Command:

```
$date --date="2/02/2010"
```

```
$date --date="Feb 2 2010"
```

Output:

```
Tue Feb 2 00:00:00 PST 2010
```

```
Tue Feb 2 00:00:00 PST 2010
```

4: Using –date option for displaying past dates:

- Date and time of 2 years ago.

Command: \$date --date="2 year ago" **Output:** Sat Oct 10 23:42:15 PDT 2015

- Date and time of 5 seconds ago.

Command: \$date --date="5 sec ago" **Output:** Tue Oct 10 23:45:02 PDT 2017

- Date and time of previous day.

Command: \$date --date="yesterday" **Output:** Mon Oct 9 23:48:00 PDT 2017

- Date and time of 2 months ago.

Command: `$date --date="2 month ago"`**Output:** Thu Aug 10 23:54:51 PDT 2017

- Date and time of 10 days ago.

Command: `$date --date="10 day ago"`**Output:** Sat Sep 30 23:56:55 PDT 2017

5:Using `--date` option for displaying future date:

Date and time of upcoming particular week day.

Command: `$date --date="next tue"`

- Date and time after two days.

Command: `$date --date="2 day"`

- Date and time of next day.

Command: `$date --date="tomorrow"`

- Date and time after 1 year on the current day.

Command: `$date --date="1 year"`

6:-**s or `--set` Option:** To set the system date and time `-s` or `--set` option is used.

Syntax: `$date --set="date to be set"`

7:-**file or `-f` Option:** This is used to display the date string present at each line of file in the date and time format.This option is similar to `--date` option but the only difference is that in `--date` we can only give one date string but in a file we can give multiple date strings at each line.

Syntax: `$date --file=file.txt`

```
$cat >> datefile
```

```
Sep 23 2018
```

```
Nov 03 2019
```

8:-**r Option:** This is used to display the last modified timestamp of a datefile .

Syntax: `$date -r file.txt`

We can modify the timestamp of a datefile by using touch command.

```
$touch datefile
```

```
$date -r datefile
```

```
Wed Oct 11 15:54:18 PDT 2017
```

```
//this is the current date and time
```

```
$touch datefile
```

```
//The timestamp of datefile is changed using touch command.
```

This was done few seconds after the above date command's output.

```
$date -r datefile
```

```
Wed Oct 11 15:56:23 PDT 2017
```

```
//display last modified time of datefile
```

9: List of Format specifiers used with date command:

%D: Display date as mm/dd/yy.

%d: Display the day of the month (01 to 31).

%a: Displays the abbreviated name for weekdays (Sun to Sat).

%A: Displays full weekdays (Sunday to Saturday).

%h: Displays abbreviated month name (Jan to Dec).

%b: Displays abbreviated month name (Jan to Dec).

%B: Displays full month name(January to December).

%m: Displays the month of year (01 to 12).

%y: Displays last two digits of the year(00 to 99).

%Y: Display four-digit year.

%T: Display the time in 24 hour format as HH:MM:SS.

%H: Display the hour.

%M: Display the minute.

%S: Display the seconds.

Syntax:

\$date +%[format-option]

Examples:

Command:

```
$date "+%D"
```

Output:

```
10/11/17
```

Command:

```
$date "+%D %T"
```

Output:

```
10/11/17 16:13:27
```

Command:

```
$date "+%Y-%m-%d"
```

Output:

```
2017-10-11
```

Command:

```
$date "+%Y/%m/%d"
```

Output:

```
2017/10/11
```

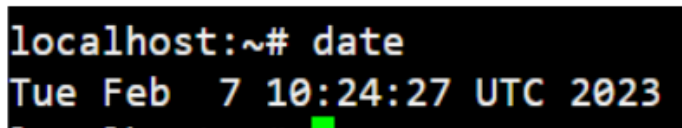
Command:

```
$date "+%A %B %d %T %y"
```

Output:

```
Thursday October 07:54:29 12 17
```

Final Output:



```
localhost:~# date
Tue Feb 7 10:24:27 UTC 2023
```

4.bc

Function:

The Linux bc command (short for basic calculator) is a command-line utility that acts as a scientific calculator. The command interprets the bc language and performs arbitrary precision arithmetic with interactive statement execution. Use the bc command as an interactive mathematical shell that takes standard input or as a mathematical scripting language.

Syntax: bc [-hlwsqv] [long-options] [file ...]

Examples:

Input : `$ echo "12+5" | bc`

Output : 17

Input : `$ echo "10^2" | bc`

Output : 100

Flags/Options:

- h**, {**- -help** } : Print the usage and exit
- i**, {**- -interactive** } : Force interactive mode
- l**, {**- -mathlib** } : Define the standard math library
- w**, {**- -warn** } : Give warnings for extensions to POSIX bc
- s**, {**- -standard** } : Process exactly the POSIX bc language
- q**, {**- -quiet** } : Do not print the normal GNU bc welcome
- v**, {**- -version** } : Print the version number and copyright and quit

Final Output:

```
localhost:~# bc
bc 1.31.1
Adapted from https://github.com/gavinhoward/bc
```

5.passwd

Function:

The passwd command changes passwords for user accounts. A normal user may only change the password for their own account, while the superuser may change the password for any account. passwd also changes the account or associated password validity period.

Syntax: passwd [options] [username]

Examples: Command: passwd

Command [root]: passwd user1

Flags/Options:

- **-d, --delete:** This option deletes the user password and makes the account password-less.
- **-e, --expire:** This option immediately expires the account password and forces the user to change password on their next login.
- **-h, --help:** Display help related to the passwd command.
- **-i, --inactive INACTIVE_DAYS:** This option is followed by an integer, INACTIVE_DAYS, which is the number of days after the password expires that the account will be deactivated.
example: passwd -i 3 user1
- **-k, --keep-tokens:** This option is used when you only want to change the password if it is expired. It keeps the authentication tokens for the authentication if the password is not yet expired, even if you requested to change it. Note that if the expiry period for a user is set to 99999, then this option will not keep tokens and the password will be changed.

- **-l, --lock:** Lock the password of user. This appends the encrypted password of the user with a character '!', and thus making it unable to match with any of input password combinations. This does not disable the account but prevents the user from logging in using a password. Though other authentication methods like ssh keys can be used to login to the account.
- **-n, --mindays MIN_DAYS:** Change the minimum number of days between password changes to MIN_DAYS so that the user can't change the password for MIN_DAYS.
- **-q, --quiet:** This option is used for quiet mode. While using this option to change a password, the message "Changing password for \$user.", which usually gets printed before changing a password, does not get echoed.
- **-r, --repository REPO:** This option is used to change password for repository named "REPO".
- **-R, --root CHROOT_DIR:** Apply changes in the CHROOT_DIR directory and use the configuration files from the CHROOT_DIR directory. This basically changes the root directory for the passwd process for once, and since CHROOT_DIR is a sub-directory of the root, it can not access the configuration files outside the CHROOT_DIR.
- **-S, --status:** Shows the password status (7 fields) of user in the following format:
user1 P 12/22/2018 0 99999 7 3

Final Output:

```
localhost:~# passwd
Changing password for root
New password:
```

6. uname

Function:

The uname command writes to standard output the name of the operating system that you are using.

Syntax: uname [OPTION]

Examples:

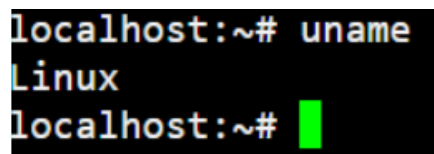
\$uname -o

\$uname -p

Flags/Options:

1. -a option: It prints all the system information in the following order: Kernel name, network node hostname, kernel release date, kernel version, machine hardware name, hardware platform, operating system
2. -s option: It prints the kernel name.
3. -n option: It prints the hostname of the network node(current computer).
4. -r option: It prints the kernel release date.
5. -v option: It prints the version of the current kernel.
6. -m option: It prints the machine hardware name.
7. -p option: It prints the type of the processor.
8. -i option: It prints the platform of the hardware.
9. -o option: It prints the name of the operating system.

Final Output:



```
localhost:~# uname
Linux
localhost:~#
```

7.help

Function:

The help command which as its name says help you to learn about any built-in command. help command as told before just displays information about shell built-in commands.

Syntax: \$help [-dms] [pattern ...]

Examples:

\$help -d help

help - Display information about builtin commands.

\$help -s help

help: help [-dms] [pattern ...]

\$help -m help

NAME

help - Display information about builtin commands.

SYNOPSIS

help [-dms] [pattern ...]

DESCRIPTION

Display information about builtin commands.

Displays brief summaries of builtin commands. If PATTERN IS specified, gives detailed help on all commands matching PATTERN, otherwise the list of help topics is printed.

Options:

- d output short description for each topic
- m display usage in pseudo-manpage format

-s output only a short usage synopsis for each topic matching
PATTERN

Arguments:

PATTERN Pattern specifying a help topic

Exit Status:

Returns success unless PATTERN is not found or an invalid option is given.

SEE ALSO

bash(1)

IMPLEMENTATION

GNU bash, version 4.3.11(1)-release (i686-pc-linux-gnu)

Copyright (C) 2013 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later

Flags/Options:

- -d option : It is used when you just want to get an overview about any shell built-in command *i.e* it only gives short description.
- -m option : It displays usage in pseudo-manpage format.
- -s option : It just displays only a short usage synopsis for each topic matching.

Final Output:

```
localhost:~# help
Built-in commands:
-----
. : [ [[ alias bg break cd chdir command continue echo eval exec
exit export false fg getopt hash help history jobs kill let
local printf pwd read readonly return set shift source test times
trap true type ulimit umask unalias unset wait
```

8.man

Function:

man command in Linux is used to display the user manual of any command that we can run on the terminal. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS.

Syntax: \$man [OPTION]... [COMMAND NAME]..

Examples:

\$ man printf

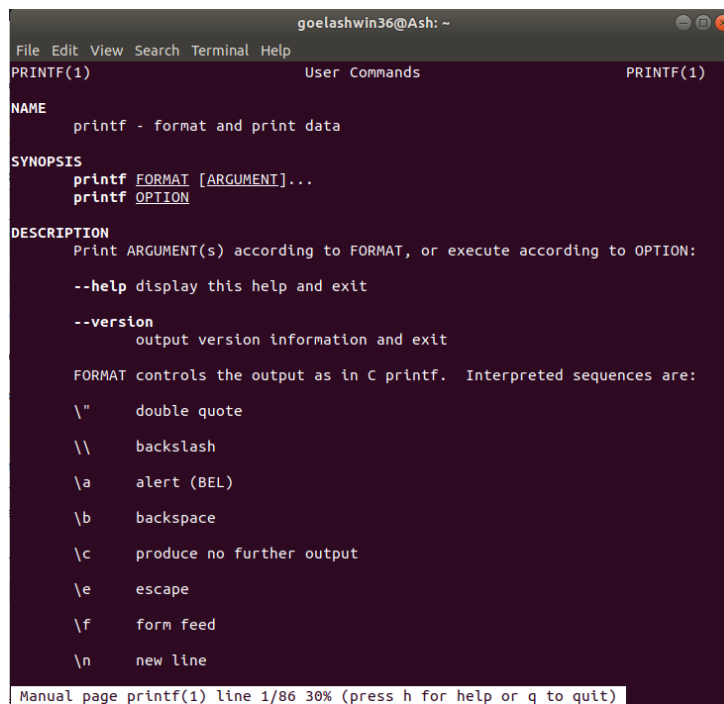
\$ man 2 intro

\$ man -f [COMMAND NAME]

Flags/Options:

1. No Option: It displays the whole manual of the command.
2. Section-num: Since a manual is divided into multiple sections so this option is used to display only a specific section of a manual.
3. -f option: One may not be able to remember the sections in which a command is present. So this option gives the section in which the given command is present.
4. -a option: This option helps us to display all the available intro manual pages in succession.
5. -k option: This option searches the given command as a regular expression in all the manuals and it returns the manual pages with the section number in which it is found.
6. -w option: This option returns the location in which the manual page of a given command is present.
7. -I option: It considers the command as case sensitive.

Final Output:



```
goelashwin36@Ash: ~
File Edit View Search Terminal Help
printf(1) User Commands printf(1)

NAME
    printf - format and print data

SYNOPSIS
    printf FORMAT [ARGUMENT]...
    printf OPTION

DESCRIPTION
    Print ARGUMENT(s) according to FORMAT, or execute according to OPTION:

    --help display this help and exit

    --version
        output version information and exit

    FORMAT controls the output as in C printf.  Interpreted sequences are:

    \"    double quote
    \\    backslash
    \a    alert (BEL)
    \b    backspace
    \c    produce no further output
    \e    escape
    \f    form feed
    \n    new line

Manual page printf(1) line 1/86 30% (press h for help or q to quit)
```

LAB EXERCISE-6

LINUX COMMANDS SET-2

1.pwd

Function:

pwd stands for Print Working Directory. It prints the path of the working directory, starting from the root. pwd is shell built-in command(pwd) or an actual binary(/bin/pwd).

\$PWD is an environment variable which stores the path of the current directory.

Syntax: pwd [-options]

Examples: We can use the below command in the terminal window to print our current working directory: `$ /bin/pwd`

1.To create a folder's symbolic link, move to the created directory and print the current working directory without symbolic links and with symbolic links.

2.Print the current working directory through the environment, even if it includes symlinks:

`$ /bin/pwd -L`

Print the original physical working directory by compelling every symbolic links:

`$ /bin/pwd -P`

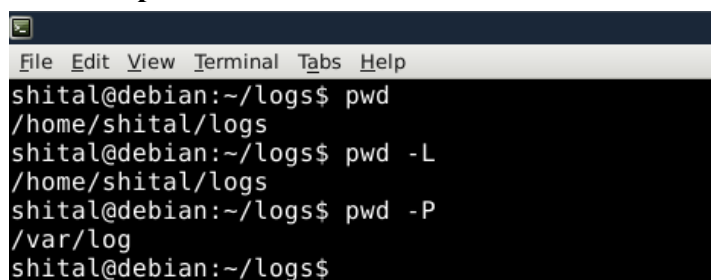
Flags/Options:

This command has two flags.

`pwd -L`: Prints the symbolic path.

`pwd -P`: Prints the actual path.

Final Output:

A terminal window screenshot with a dark background and light text. The window has a title bar with 'File Edit View Terminal Tabs Help'. The prompt is 'shital@debian:~/logs\$'. The first command is 'pwd', which outputs '/home/shital/logs'. The second command is 'pwd -L', which also outputs '/home/shital/logs'. The third command is 'pwd -P', which outputs '/var/log'. The prompt returns to 'shital@debian:~/logs\$'.

2.cd

Function: Linux `cd` command is used to change the current working directory (i.e., in which the current user is working). The "`cd`" stands for 'change directory.' It is one of the most frequently used commands in the Linux terminal.

Syntax: `cd <dirname>`

Examples:

1.Change from the current directory to a new directory: `cd < current directory> <specified directory>`

2. Change directory using an absolute path: To change the directory by using an absolute path, we have to mention the whole path starting from the root.

3. Change directory using a relative path: `cd certs`

4. Change to the home directory: `cd ~`

5. Change to the previous directory: `cd-`

Flags/Options:

- `-L`: Force following symbolic links. This option is enabled by default.
- `-P`: Do not follow symbolic links. This option resolves the symbolic link to determine the parent directory before moving to the user-requested directory.
- `-e`: Exit with a non-zero status if using the `-P` option and the command cannot resolve the symbolic link.
- `-@`: Present a file with extended attributes as a directory containing the file attributes.

Final Output:

```
raghvendra@raghvendra-Inspiron-15-3567: ~/My songs
File Edit View Search Terminal Help
raghvendra@raghvendra-Inspiron-15-3567:~$ ls
Desktop  git_hand      Music      Public      Videos
Documents  git_repos    'My songs'  snap
Downloads  java2python-0.5.1  Pictures    Templates
raghvendra@raghvendra-Inspiron-15-3567:~$ cd My\ songs
raghvendra@raghvendra-Inspiron-15-3567:~/My songs$ pwd
/home/raghvendra/My songs
raghvendra@raghvendra-Inspiron-15-3567:~/My songs$
```

3.mkdir

Function:

mkdir command in Linux allows the user to create directories (also referred to as folders in some operating systems). This command can create multiple directories at once as well as set the permissions for the directories.

Syntax: mkdir [options...] [directories ...]

Examples:

–version: It displays the version number, some information regarding the license and exits.

Syntax: mkdir --version

–help: It displays the help related information and exits. Syntax: mkdir --help

–v or –verbose: It displays a message for every directory created.

Syntax: mkdir -v [directories]

Flags/Options:

Options	Description
mkdir -p, -parents	Add directory including its sub directory.
mkdir -v, -verbose	Print a message for each created directory.
mkdir -m -mode=MODE	Set access privilege.

Final Output:

```
Terminal
File Edit View Search Terminal Help
rossoskull@RossoSkull ~/GFG $ mkdir -v one two three
mkdir: created directory 'one'
mkdir: created directory 'two'
mkdir: created directory 'three'
rossoskull@RossoSkull ~/GFG $ ls
one three two
rossoskull@RossoSkull ~/GFG $
```

4.cp

Function:

cp stands for **copy**. This command is used to copy files or group of files or directory. It creates an exact image of a file on a disk with different file name. *cp* command require at least two filenames in its arguments.

Syntax: cp [OPTION] Source Destination

cp [OPTION] Source Directory

cp [OPTION] Source-1 Source-2 Source-3 Source-n Directory

First and second syntax is used to copy Source file to Destination file or Directory.

Third syntax is used to copy multiple Sources(files) to Directory.

Examples: 1. Copy file in current directory itself.

```
$ cp viewers_list.txt users_list.txt
```

2. Copy a file in 'backup' directory

```
$ cp viewers_list.txt backup/
```

3. Copy in 'backup' directory with different name

```
$ cp viewers_list.txt backup/viewers_list.bak
```

4. Use **-i** option of **cp** command for interactive mode to prompt before overwriting an existing file.

```
$ cp -i viewers_list.txt users_list.txt
```

```
cp: overwrite 'users_list.txt'? y
```

```
$ cp --interactive viewers_list.txt users_list.txt
```

```
cp: overwrite 'users_list.txt'? Y
```

5. Copy multiple files to a specified directory, in this case 'news'\$ cp current_news.txt headlines.txt cover_story.txt news/

6. Copy multiple files using wild card. It copies all files with extension .txt to 'newsportal' directory.\$ cp *.txt newsportal/

Arguments: cp command works on three principal modes of operation and these operations depend upon number and type of arguments passed in cp command :

1. Two file names : If the command contains two file names, then it copy 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.
cp Src_file Dest_file
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, so be careful !!.

```
cp Src_file1 Src_file2 Src_file3 Dest_directory
```

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 -R Src_directory Dest_directory`

Flags/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.

```
$ cp -i a.txt b.txt
```

```
cp: overwrite 'b.txt'? y
```

```
$ cat b.txt
```

GFG

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.

```
$ ls
```

```
a.txt b.txt
```

```
$ cp -b a.txt b.txt
```

```
$ ls
```

```
a.txt b.txt b.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.

```
$ ls -l b.txt
```

```
-r-xr-xr-x+ 1 User User 3 Nov 24 08:45 b.txt
```

User, group and others doesn't have writing permission.

Without `-f` option, command not executed

```
$ cp a.txt b.txt
```

```
cp: cannot create regular file 'b.txt': Permission denied
```

With `-f` option, command executed successfully

```
$ cp -f a.txt b.txt
```

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 `geeksforgeeks` directory containing many files, directories into `gfg` directory(not exist).

```
$ ls geeksforgeeks/
```

```
a.txt b.txt b.txt~ Folder1 Folder2
```

Without `-r` option, error

```
$ cp geeksforgeeks gfg
```

```
cp: -r not specified; omitting directory 'geeksforgeeks'
```

With `-r`, execute successfully

```
$ cp -r geeksforgeeks gfg
```

```
$ ls gfg/
```

```
a.txt b.txt b.txt~ Folder1 Folder2
```

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.

Note: For the preservation of characteristics you must be the root user of the system, otherwise characteristics changes.

```
$ ls -l a.txt
```

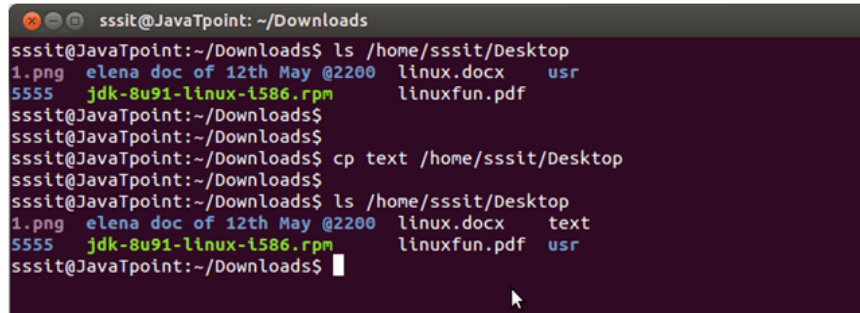
```
-rwxr-xr-x+ 1 User User 3 Nov 24 08:13 a.txt
```

```
$ cp -p a.txt c.txt
```

```
$ ls -l c.txt
```

```
-rwxr-xr-x+ 1 User User 3 Nov 24 08:13 c.txt
```

Final Output:



```
ssstt@JavaTpoint: ~/Downloads
ssstt@JavaTpoint:~/Downloads$ ls /home/ssstt/Desktop
1.png  elena doc of 12th May @2200  linux.docx  usr
5555   jdk-8u91-linux-i586.rpm      linuxfun.pdf
ssstt@JavaTpoint:~/Downloads$
ssstt@JavaTpoint:~/Downloads$
ssstt@JavaTpoint:~/Downloads$ cp text /home/ssstt/Desktop
ssstt@JavaTpoint:~/Downloads$
ssstt@JavaTpoint:~/Downloads$ ls /home/ssstt/Desktop
1.png  elena doc of 12th May @2200  linux.docx  text
5555   jdk-8u91-linux-i586.rpm      linuxfun.pdf  usr
ssstt@JavaTpoint:~/Downloads$
```

5.mv

Function: mv stands for move. mv is used to move one or more files or directories from one place to another in a file system like UNIX. It has two distinct functions:

(i) It renames a file or folder.

(ii) It moves a group of files to a different directory.

No additional space is consumed on a disk during renaming. This command normally works silently means no prompt for confirmation.

Syntax: mv [Option] source destination

Examples: Move *main.c* *def.h* files to */home/usr/rapid/* directory:

```
$ mv main.c def.h /home/usr/rapid/
```

Move all C files in current directory to subdirectory *bak* :

```
$ mv *.c bak
```

Move all files in subdirectory *bak* to current directory :

```
$ mv bak/* .
```

Rename file *main.c* to *main.bak*:

```
$ mv main.c main.bak
```

Rename directory *bak* to *bak2*:

```
$ mv bak bak2
```

Update - move when *main.c* is newer:

```
$ mv -u main.c bak
```

```
$
```

Move *main.c* and prompt before overwrite *bak/main.c*:

```
$ mv -v main.c bak
```

```
'bak/main.c' -> 'bak/main.c'
```

```
$
```

Flags/Options:

option	description
mv -f	force move by overwriting destination file without prompt
mv -i	interactive prompt before overwrite
mv -u	update - move when source is newer than destination
mv -v	verbose - print source and destination files
man mv	help manual

Final Output:

```
sssit@JavaTpoint: ~/Desktop
sssit@JavaTpoint:~/Downloads$ ls
docc file imp temp
sssit@JavaTpoint:~/Downloads$
sssit@JavaTpoint:~/Downloads$ mv * /home/sssit/Desktop
sssit@JavaTpoint:~/Downloads$ ls
sssit@JavaTpoint:~/Downloads$
sssit@JavaTpoint:~/Downloads$ cd /home/sssit/Desktop
sssit@JavaTpoint:~/Desktop$
sssit@JavaTpoint:~/Desktop$ ls docc file imp temp
docc file

imp:

temp:
sssit@JavaTpoint:~/Desktop$
```

6.rmdir

Function: The rmdir command removes the directory, specified by the *Directory* parameter, from the system. The directory must be empty before you can remove it, and you must have write permission in its parent directory.

Syntax: rmdir [-p] *Directory* ...

Examples: To empty and remove a directory, type:

```
rm mydir/* mydir/.*
```

```
rmdir mydir
```

To remove the /home, /home/demo, and /home/demo/mydir directories, type:

```
rmdir -p /home/demo/mydir
```

Flags/Options:

Item	Description
-p <i>Directory</i>	Removes all directories along the path name specified by the <i>Directory</i> parameter. Parent directories must be empty and the user must have write permission in the parent directories before they can be removed.

Final Output:

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~/created$ ls  
file1 file2  
sssit@JavaTpoint:~/created$ rmdir file1  
sssit@JavaTpoint:~/created$ ls  
file2  
sssit@JavaTpoint:~/created$ cd ..  
sssit@JavaTpoint:~$ rmdir created  
rmdir: failed to remove `created': Directory not empty  
sssit@JavaTpoint:~$  
sssit@JavaTpoint:~$ rmdir created/file2  
sssit@JavaTpoint:~$ rmdir created  
sssit@JavaTpoint:~$
```

7.rm

Function: The rm command removes the entries for a specified file, group of files, or certain select files from a list within a directory. User confirmation, read permission, and write permission are not required before a file is removed when you use the rm command.

Syntax: rm -dfiPRr file ...

Examples: The following are examples of how to use the rm command:

- To delete the file named myfile, type the following:

```
rm myfile
```
- To delete all the files in the mydir directory, one by one, type the following:

```
rm -i mydir/*
```

Flags/Options:

Option	Description
rm *extension	Used to delete files having same extension.
rm -r or R	To delete a directory recursively.
rm -i	Remove a file interactively.
rm -rf	Remove a directory forcefully.

Final Output:

```
gfg@geekforgeeks ~/GFG $ mkdir A  
gfg@geekforgeeks ~/GFG $ touch B.txt  
gfg@geekforgeeks ~/GFG $ ls  
A B.txt  
gfg@geekforgeeks ~/GFG $ rm B.txt  
gfg@geekforgeeks ~/GFG $ rm A  
rm: cannot remove 'A': Is a directory  
gfg@geekforgeeks ~/GFG $
```

LAB EXERCISE-7

LINUX COMMANDS SET-3

1.ls

Function: In computing, ls is a command to list computer files and directories in Unix and Unix-like operating systems. It is specified by POSIX and the Single UNIX Specification. It lists the information about the files (the current directory by default).

Syntax: ls [options] [file... | directory ...]

Examples:

```
# ls

0001.pcap      Desktop      Downloads
index.html     install.log syslog  Pictures
Templates      anaconda-ks.cfg Documents
fbcmd_update.php install.log   Music
Public         Videos
```

List all files including hidden files starting with ‘.’.

```
# ls -a

.          .bashrc      Documents    .gconfd
install.log .nautilus    .pulse-cookie
..         .cache       Downloads    .gnome2
install.log .netstat.swp .recently-used.xbel
0001.pcap  .config      .elinks      .gnome2_private
.kde       .opera       .spice-vdagent
anaconda-ks.cfg .cshrc      .esd_auth    .gtk-bookmarks
.libreoffice Pictures     .tcshrc
.bash_history .dbus       .fbcmd       .gvfs
.local      .pki        Templates
.bash_logout Desktop     fbcmd_update.php .ICEauthority
.mozilla    Public      Videos
.bash_profile .digrc     .gconf       index.html
Music       .pulse     .wireshark
```

Flags/Options:

<code>ls -g or ls -lg</code>	With this you can exclude column of group information and owner.
<code>ls -n</code>	It is used to print group ID and owner ID instead of their names.
<code>ls --color=[VALUE]</code>	This command is used to print list as colored or discolored.
<code>ls -li</code>	This command prints the index number if file is in the first column.
<code>ls -p</code>	It is used to identify the directory easily by marking the directories with a slash (/) line sign.
<code>ls -r</code>	It is used to print the list in reverse order.
<code>ls -R</code>	It will display the content of the sub-directories also.
<code>ls -lX</code>	It will group the files with same extensions together in the list.
<code>ls -lt</code>	It will sort the list by displaying recently modified files at top.
<code>ls ~</code>	It gives the contents of home directory.
<code>ls ../</code>	It gives the contents of parent directory.

Final Output:

```

/home/sssit
sssit@JavaTpoint:~$ ls
Desktop    Downloads      Music    Public    Videos
Documents  examples.desktop  Pictures  Templates
sssit@JavaTpoint:~$

```

2.tty

Function: The tty command writes the name of your terminal to standard output. If your standard input is not a terminal and you do not specify the -s flag, you get the message Standard input is not a tty.

Syntax: `tty [OPTION]...`

Flags/Options:

- `-s, --silent, --quiet` : Prints nothing, only returns an exit status.
- `--help` : It will display the help message and exit.
- `--version` : Prints the version information and exits.

Final Output:

```

File Edit View Search Terminal Help
master@hellboy:~$ sudo tty
[sudo] password for master:
/dev/pts/0
master@hellboy:~$

```

3.tput clear

Function: The tput command uses the terminfo database to make terminal-dependent information available to the shell. The tput command outputs a string if the attribute CapabilityName is of type string. The output string is an integer if the attribute is of type integer. If the attribute is of type Boolean, the tput command sets the exit value (0 for TRUE, 1 for FALSE), and produces no other output. tput clear displays the clear screen sequence

Syntax: For Outputting Terminal Information

tput [-T *Type*] [*CapabilityName* {*clear*, *init*, *longname*, *reset*} [*Parameters*...]]

For Using stdin to Process Multiple Capabilities

tput [-S]

Examples:

To clear the screen for the current terminal, enter:

```
tput clear
```

To display the number of columns for the current terminals, enter:

```
tput cols
```

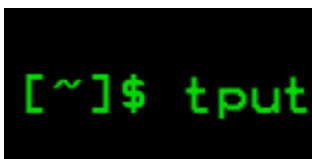
To display the number of columns for the aixterm terminal, enter:

```
tput -T aixterm cols
```

Flags/Options:

Item	Description
clear	Displays the clear screen sequence (this is also a capability name).
init	Displays the sequence that initializes the user's terminal in an implementation-dependent manner.
reset	Displays the sequence that will reset the user's terminal in an implementation-dependent manner.
longname	Displays the long name and the specified terminal (or current terminal if none specified).
-S	Uses stdin. This allow the tput to process multiple capabilities. When using the -S option, the capabilities cannot be entered on the command line. Enter ^D token finished.
-T<i>Type</i>	Indicates the type of terminal. If -T is not specified, the TERM environment variable is used for the terminal.

Final Output:

A terminal window with a black background and green text. The prompt is [~]\$ and the command tput has been entered.

4.cat

Function: Concatenate, or cat, is one of the most frequently used Linux commands. It lists, combines, and writes file content to the standard output. To run the cat command, type cat followed by the file name and its extension.

Syntax: cat <filename>

Examples:

cat filename.txt.

Here are other ways to use the cat command:

- `cat > filename.txt` creates a new file.
- `cat filename1.txt filename2.txt > filename3.txt` merges filename1.txt and filename2.txt and stores the output in filename3.txt.
- `tac filename.txt` displays content in reverse order.

Flags/Options:

Option	Function
<code>cat > [fileName]</code>	To create a file.
<code>cat [oldfile] > [newfile]</code>	To copy content from older to new file.
<code>cat [file1 file2 and so on] > [new file name]</code>	To concatenate contents of multiple files into one.
<code>cat -n/cat -b [fileName]</code>	To display line numbers.
<code>cat -e [fileName]</code>	To display \$ character at the end of each line.
<code>cat [fileName] <<EOF</code>	Used as page end marker.

Final Output:

```
sssit@JavaTpoint:~/Desktop$ cat jtp.txt
this is javatpoint
you are learning linux here
thankyou
thankyou
thankyou
a
b
c
d
e
f
g
h
i
j
k
l
MMMMMM
nnnnnn
```

5.more

Function: The **more** command sets the terminal to NOECHO mode so the output can be continuous. With the exception of the / and ! subcommands, commands that are typed do not normally show up on the terminal. If the standard output is not a terminal, the **more** command will act just like the **cat** command, except that a header will be printed before each file in a series.

Syntax: more [options].....

Examples:

more -d sample.txt

more -f sample.txt

Flags/Options:

-d : Use this command in order to help the user to navigate. It displays “[Press space to continue, ‘q’ to quit.]” and displays “[Press ‘h’ for instructions.]” when wrong key is pressed.

-f : This option does not wrap the long lines and displays them as such.

-p : This option clears the screen and then displays the text.

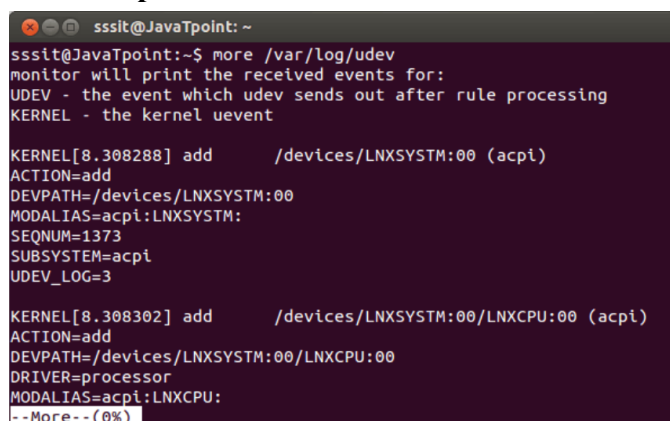
-c : This command is used to display the pages on the same area by overlapping the previously displayed text.

-s : This option squeezes multiple blank lines into one single blank line.

-u : This option omits the underlines

+/pattern : This option is used to search the string inside your text document. You can view all the instances by navigating through the result.

Final Output:



```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ more /var/log/udev  
monitor will print the received events for:  
UDEV - the event which udev sends out after rule processing  
KERNEL - the kernel uevent  
  
KERNEL[8.308288] add      /devices/LNXSYSTM:00 (acpi)  
ACTION=add  
DEVPATH=/devices/LNXSYSTM:00  
MODALIAS=acpi:LNXSYSTM:  
SEQNUM=1373  
SUBSYSTEM=acpi  
UDEV_LOG=3  
  
KERNEL[8.308302] add      /devices/LNXSYSTM:00/LNXCPU:00 (acpi)  
ACTION=add  
DEVPATH=/devices/LNXSYSTM:00/LNXCPU:00  
DRIVER=processor  
MODALIAS=acpi:LNXCPU:  
--More-- (0%)
```

6.wc

Function: Wc stands for word count. As the name implies, it is mainly used for counting purpose.

- It is used to count the number of lines, word count, byte and characters count in the files specified in the file arguments.
- By default it displays four-columnar output.
- First column shows number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column itself is the file name which is given as argument.

Syntax: wc [option].....[File]....

wc [OPTION]... --files0-from=F

Examples:

wc exm.txt

The above command will display the number of lines, number of words, number of bytes, and file name from the file 'exm.txt'.

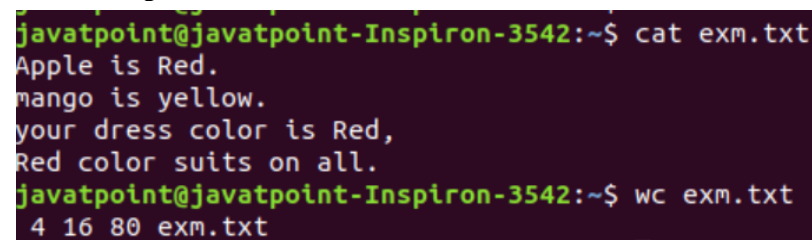
To display the complete count information of multiple files at once, specify the file names after space (' '). It is executed as follows:

`wc <file1> <file2>`

Flags/Options:

- c, --bytes: It is used to print the byte counts.
- m, --chars: It is used to print the character counts.
- l, --lines: It is used to print the newline counts.
- files0-from=F: It is used to read input from specified files.
- L, --max-line-length: It is used to print the maximum display width.
- w, --words: It is used to print the word counts.
- help: It is used to display the help manual.
- version: It is used to display the version information.

Final Output:



```
javatpoint@javatpoint-Inspiron-3542:~$ cat exm.txt
Apple is Red.
mango is yellow.
your dress color is Red,
Red color suits on all.
javatpoint@javatpoint-Inspiron-3542:~$ wc exm.txt
 4 16 80 exm.txt
```

7.ps

Function: The ps command is used to view currently running processes on the system. It helps us to determine which process is doing what in our system, how much memory it is using, how much CPU space it occupies, user ID, command name, etc. The ps command may display different results for different systems because it displays information about the currently running process of a system.

Syntax: `ps [options]`

Flags/Options:

Option	Function
<code>ps -ef / -aux</code>	List currently running process in full format
<code>ps -ax</code>	List currently running process
<code>ps -u <username></code>	List process for specific user
<code>ps -C <command></code>	List process for given command
<code>ps -p <PID></code>	List process with given PID
<code>ps -ppid <PPID></code>	List process with given ppid
<code>pstree</code>	Show process in hierarchy
<code>ps -L</code>	List all threads for a particular process
<code>ps --sort pmem</code>	Find memory leak
<code>ps -eo</code>	Show security information

Final Output:

```
[root@rhel7 ~]# ps -A
[root@rhel7 ~]# ps -e
```

8.file

Function: File command is used to determine the type of a file. .file type may be of human-readable(e.g. ‘ASCII text’) or MIME type(e.g. ‘text/plain; charset=us-ascii’). This command tests each argument in an attempt to categorize it.

It has three sets of tests as follows:

filesystem test: This test is based on the result which returns from a stat system call. The program verifies that if the file is empty, or if it’s some sort of special file. This test causes the file type to be printed.

magic test: These tests are used to check for files with data in particular fixed formats.

language test: This test search for particular strings which can appear anywhere in the first few blocks of a file.

Syntax: file <filename>

Flags/Options:

Option	Function
<code>file -s</code>	Used for special files.
<code>file *</code>	Used to list types of all the files.
<code>file /directory name/*</code>	Used to list types of all the files from mentioned directory.
<code>file [range]*</code>	It will list out all the files starting from the alphabet present within the given range.

Final Output:

```

sssit@JavaTpoint:~/Desktop$ file jdk-8u91-linux-i586.rpm
jdk-8u91-linux-i586.rpm: RPM v3.0 bin i386/x86_64
sssit@JavaTpoint:~/Desktop$ 
sssit@JavaTpoint:~/Desktop$ file 1.png
1.png: PNG image data, 724 x 463, 8-bit/color RGBA, non-interlaced
sssit@JavaTpoint:~/Desktop$ 
sssit@JavaTpoint:~/Desktop$ file linux.docx
linux.docx: Microsoft Word 2007+
sssit@JavaTpoint:~/Desktop$ 
sssit@JavaTpoint:~/Desktop$ file linuxfun.pdf
linuxfun.pdf: PDF document, version 1.4
sssit@JavaTpoint:~/Desktop$ file usr
usr: directory
sssit@JavaTpoint:~/Desktop$ 

```

LAB EXERCISE-8

LINUX COMMANDS SET-4

1.gzip

Function: Gzip (GNU zip) is a compressing tool, which is used to truncate the file size. By default original file will be replaced by the compressed file ending with extension (.gz).

Syntax: gzip <file1> <file2> <file3> . . .

Examples and Flags/Options:

1. -c or --stdout: Writes the compressed output to standard output instead of creating a new file.

```
gzip -c file.txt > file.txt.gz
```

2. -d or --decompress: Decompresses the specified files.

gzip -d file.txt.gz

3. -f or --force: Forces gzip to overwrite existing files.
4. -k or --keep: Preserves the original file and creates a new compressed file with the same name.
5. -r or --recursive: Compresses files recursively within a directory.
6. -t or --test: Tests the integrity of the compressed file.

Final Output:

```
gzip -c file.txt > file.txt.gz
```

Writes the compressed output to standard output instead of creating a new file.

2.gunzip

Function: gunzip command is used to compress or expand a file or a list of files in Linux. It accepts all the files having extension as .gz, .z, _z, -gz, -z , .Z, .taz or.tgz and replace the compressed file with the original file by default.

Syntax: gunzip [Option] [archive name/file name]

Examples and Flags/Options:

1. **-c or --stdout:** Writes the uncompressed output to standard output instead of creating a new file.

```
gunzip -c file.gz > output.txt
```

2. **-d or --decompress:** Decompresses the specified files.

```
gunzip -d file.gz
```

3. **-f or --force:** Forces gunzip to decompress files that would otherwise not be decompressed, such as files with multiple extensions.

```
gunzip -f file.tar.gz
```

4. **-k or --keep:** Preserves the original compressed file and creates a new uncompressed file with the same name.
5. **-l or --list:** Lists the contents of the compressed file without actually decompressing it.

Final Output:

```
gunzip -c file.gz > output.txt
```

3.tar

Function:

The `tar` command in Linux is used to create, manipulate, and extract archive files

Syntax: `tar [options] [archive-name] [file(s) or directory]`

Examples: For example, to create a new archive named `myfiles.tar.gz` containing the files `file1.txt` and `file2.txt`, you could use the following command: `tar -czvf myfiles.tar.gz file1.txt file2.txt`

To extract the files from the `myfiles.tar.gz` archive, you could use the following command:

```
tar -xzvf myfiles.tar.gz -C /home/user/mydir/
```

This will extract the files to the current directory. If you want to extract them to a specific directory, you can use the `-C` option followed by the directory path. For example:

```
tar -xzvf myfiles.tar.gz -C /home/user/mydir/
```

Flags/Options:

- **-c:** Create a new archive.

- **-x:** Extract files from an archive.
- **-v:** Verbose output. Displays the name of each file as it is processed.
- **-f:** Use the following filename as the archive.
- **-z:** Use gzip compression.
- **-j:** Use bzip2 compression.

Final Output:

```
tar -cvf archive.tar file1 file2 directory/
```

This creates a new archive.

4.zip

Function: The zip command is a commonly used utility in Linux for creating and managing compressed archives.

Syntax: `zip my_archive.zip *`

Examples and Flags/Options:

1. **-r:** Recursively compresses a directory and its contents.

`zip -r compressed.zip directory/`

2. **-m:** Moves the original files to the archive (useful for deleting files after compressing).

`zip -m compressed.zip file.txt`

3. **-q:** Quiet mode, which suppresses the output of the command.
4. **-j:** Compresses files, discarding any directory structure.
5. **-e:** Encrypts the archive with a password.
6. **-9:** Sets the compression level to the maximum (slow but produces the smallest file).
7. **-u:** Updates an existing archive by adding new files.
8. **-x:** Excludes specific files or directories from the archive.

Final Output:

```
adding: file1.txt (stored 0%)
adding: file2.txt (deflated 25%)
adding: file3.txt (deflated 50%)
adding: file4.txt (deflated 75%)
```

In this example, file1.txt was added to the archive without compression, while the other files were compressed to varying degrees. The percentage values indicate how much the file size was reduced by compression.

5.unzip

Function: The unzip command in Linux is used to extract files from a ZIP archive. When you run the unzip command in the terminal, it will output information about the files that are being extracted, including their names, sizes, and permissions.

Syntax: unzip [options] <zipfile> [-x <exclude_pattern>] [-d <destination_directory>]

Examples: 1. Extract all files from a ZIP archive:

unzip example.zip

This command will extract all files from the example.zip archive to the current working directory.

2. Extract specific files from a ZIP archive: **unzip example.zip file1.txt file2.txt**

This command will extract only the file1.txt and file2.txt files from the example.zip archive to the current working directory.

3. Extract all files from a ZIP archive, but exclude certain files:

unzip example.zip -x *.jpg

This command will extract all files from the example.zip archive to the current working directory, but it will exclude any files with the extension .jpg.

4. Extract all files from a ZIP archive to a specific directory:

unzip example.zip -d /path/to/directory/

This command will extract all files from the example.zip archive to the /path/to/directory/ directory.

5. List the contents of a ZIP archive:

unzip -l example.zip

This command will list the contents of the example.zip archive without extracting any files.

6. Extract encrypted ZIP archive with password:

unzip -P PASSWORD example.zip

This command will extract all files from the example.zip archive to the current working directory, but it will prompt for a password to decrypt the archive. Replace PASSWORD with the actual password required to open the ZIP archive.

Flags/Options:

- **-l:** Lists the contents of a ZIP archive without extracting any files.
- **-t:** Tests the integrity of a ZIP archive.
- **-d <destination_directory>:** Specifies the directory where the extracted files should be placed.
- **-x <exclude_pattern>:** Excludes files from being extracted based on a pattern.
- **-v:** Verbose mode - displays additional information during the extraction process.
- **-q:** Quiet mode - suppresses output during the extraction process.
- **-f:** Forces the overwriting of existing files during extraction.
- **-u:** Updates the contents of an existing file during extraction, only if the version in the archive is newer.
- **-n:** Never overwrite existing files during extraction.
- **-P <password>:** Specifies a password to decrypt an encrypted ZIP archive.
- **-U:** Use Unicode UTF-8 character encoding during extraction.
- **-j:** Junk paths - extracts files without creating any directories.
- **-o:** Overwrite existing files during extraction without prompting.

Arguments:

The unzip command in Linux takes two types of arguments:

1. Required arguments:

- **<zipfile>:** This is the name of the ZIP archive that you want to extract files from. It is a required argument, and you must provide the name of the ZIP archive to the unzip command.

2. Optional arguments:

- [options]: These are optional flags that you can use to customize the behavior of the unzip command. These are not required, but they can be used to change how unzip behaves when extracting files from a ZIP archive.
- [-x <exclude_pattern>]: This is an optional flag that you can use to exclude files from being extracted based on a pattern. This is also an optional argument and is used in combination with options.
- [-d <destination_directory>]: This is an optional flag that you can use to specify a directory where the extracted files should be placed. This is also an optional argument and is used in combination with options.

Final Output:

```
$ unzip example.zip
Archive:  example.zip
  creating: example/
  creating: example/dir1/
 inflating: example/dir1/file1.txt
 inflating: example/dir1/file2.txt
  creating: example/dir2/
 inflating: example/dir2/file3.txt
 inflating: example/dir2/file4.txt
```

In this example, the unzip command is extracting the contents of a ZIP archive called example.zip. The output shows that the example directory is being created, along with two subdirectories (dir1 and dir2). The files file1.txt, file2.txt, file3.txt, and file4.txt are also being extracted, with their paths relative to the example directory. The inflating keyword indicates that a file is being extracted, while the creating keyword indicates that a new directory is being created. Once the extraction is complete, the terminal prompt will return and the extracted files will be available in the specified directory.

6.cmp

Function: The “cmp” command is a simple yet powerful feature used to compare the contents of two files and determine if they have similar content. The “cmp” command is particularly useful for system administrators and developers who need to verify the integrity of files or compare the contents of different file versions.

The 'cmp' command lets you carry out a byte-by-byte comparison of two files. The utility provides several features in the form of command line options. In this tutorial, we will discuss some key options that'll give you (a beginner) a good idea of how the tool works.

Syntax: `cmp [file1-name] [file2-name]`

Examples:

1. Compare two files and display differences:

`cmp file1.txt file2.txt`

This command will compare file1.txt and file2.txt and display the differences between them.

2. Suppress output and only return the exit status:

`cmp -s file1.txt file2.txt`

This command will compare file1.txt and file2.txt but will not display any output. It will only return the exit status, which will be 0 if the files are the same and 1 if they are different.

3. Compare only a specific number of bytes:

`cmp -n 100 file1.txt file2.txt`

This command will compare the first 100 bytes of file1.txt and file2.txt and display the differences between them.

4. Compare two binary files:

`cmp -l file1.bin file2.bin`

Flags/Options:

- **-b or --print-bytes:** This flag prints the byte number and the differing byte in octal format for each difference found.
- **-i N or --ignore-initial=N:** This flag skips the first N bytes of both files before starting the comparison.
- **-l or --verbose:** This flag prints all differing bytes in the files, along with their byte number and the corresponding byte in octal format.
- **-s or --quiet or --silent:** This flag suppresses the output of differences and only reports whether the files differ or not.
- **-n N or --bytes=N:** This flag limits the comparison to the first N bytes of each file.

Final Output:

```
cmp file1.txt file2.txt
```

7.comm

Function:

comm compare two sorted files line by line and write to standard output; the lines that are common and the lines that are unique. Suppose you have two lists of people and you are asked to find out the names available in one and not in the other, or even those common to both. comm is the command that will help you to achieve this. It requires two sorted files which it compares line by line.

Syntax: \$comm [OPTION]... FILE1 FILE2

Examples: Let us suppose there are two sorted files file1.txt and file2.txt and now we will use comm command to compare these two.

// displaying contents of file1 //

```
$cat file1.txt
```

Apaar

Ayush Rajput

Deepak

Hemant

// displaying contents of file2 //

```
$cat file2.txt
```

Apaar

Hemant

Lucky

Pranjal Thakral

```
[root@localhost sample]# comm --output-delimiter=Test f1.txt f2.txt
1
2
TestTest3
TestTest4
Test7
Test8
[root@localhost sample]#
```

Flags/Options:

1. Display only the lines that are common to both files:

comm FILE1 FILE2

This will display only the lines that are common to both FILE1 and FILE2.

2. Display only the lines that are unique to the first file:

comm -23 FILE1 FILE2

This will display only the lines that are unique to FILE1.

3. Display only the lines that are unique to the second file:

comm -13 FILE1 FILE2

This will display only the lines that are unique to FILE2.

4. Display the lines that are common to both files as well as the lines that are unique to the first file:

comm -12 FILE1 FILE2

This will display both the lines that are common to both files and the lines that are unique to FILE1.

Final Output:

```
comm file1.txt file2.txt
```

8.chmod

Function:

Linux chmod command is used to change the access permissions of files and directories. It stands for change mode. It can not change the permission of symbolic links. Even, it ignores the symbolic links come across recursive directory traversal.

Syntax: chmod <options> <permissions> <file name>

Examples:

1. To set read, write and execute permissions for owner and group, and only read and execute permissions for others on a file named example.txt:

```
chmod [options] mode file/dir
```

2. To set read, write, and execute permissions for all users on a directory named mydir:
3. To remove write permissions for group and others on a file named myfile.txt:
4. To add execute permissions for others on a directory named docs:

Flags/Options:

- **u (user):** This flag modifies the permissions for the owner of the file or directory.
- **g (group):** This flag modifies the permissions for the group that the file or directory belongs to.
- **o (other):** This flag modifies the permissions for all other users who are not the owner or part of the group.

- **a (all):** This flag modifies the permissions for all users (equivalent to using the ugo flags together).
- **r (read):** This flag grants read permission.
- **w (write):** This flag grants write permission.
- **x (execute):** This flag grants execute permission.
- **+** (add): This flag adds permissions.
- **-** (remove): This flag removes permissions.
- **=** (set): This flag sets the permissions to the specified mode.

Final Output:

```
chmod u=rw,go=r myfile.txt
```

This command sets the owner's permissions to read and write, and the group and other users' permissions to read only.

9.od

Function:

The od command in Linux is used to display the contents of a file in different formats. It is typically used to examine binary files, but can also be used to view text files in different formats

Syntax: od [OPTIONS] [FILE]

Examples and Flags/Options:

1. Display the contents of a file in octal format:

od -b FILE

This will display the contents of FILE in octal format.

2. Display the contents of a file in hexadecimal format:

od -x FILE

This will display the contents of FILE in hexadecimal format.

3. Display the contents of a file in ASCII format:

od -c FILE

This will display the contents of FILE in ASCII format.

4. Display the contents of a file in decimal format:

od -d FILE

This will display the contents of FILE in decimal format.

5. Display the contents of a file in little-endian format:

od -A1 -t x1 FILE

This will display the contents of FILE in little-endian format.

Final Output:

```
anurag@HP:~$ cat input.txt
100
101
102
103
104
105

anurag@HP:~$ od -b input.txt
0000000 061 060 060 012 061 060 061 012 061 060 062 012 061 060 063 012
0000020 061 060 064 012 061 060 065 012 012
0000031
anurag@HP:~$
```

b)Explore different options of vi editor.

Vi is a powerful text editor that is available on most Unix-based systems, including Linux. Here are some of the most commonly used options for vi:

1. Navigation:

- h: move the cursor left
- j: move the cursor down
- k: move the cursor up
- l: move the cursor right
- 0: move to the beginning of the line
- \$: move to the end of the line
- G: move to the end of the file
- gg: move to the beginning of the file
- CTRL-f: move forward one page
- CTRL-b: move backward one page

2. Editing:

- i: insert text before the cursor
- a: append text after the cursor
- o: insert a new line below the current line
- O: insert a new line above the current line
- x: delete the character under the cursor
- dd: delete the current line
- yy: copy the current line
- p: paste the copied line after the current line

3. Saving and quitting:

- :w: save the changes to the file
- :q: quit the editor
- :wq: save the changes and quit the editor
- :q!: quit the editor without saving changes

4. Search and replace:

- /search-term: search forward for the specified term
- ?search-term: search backward for the specified term
- n: repeat the search in the same direction
- N: repeat the search in the opposite direction
- :%s/old-text/new-text/g: replace all occurrences of old-text with new-text in the entire file

These are just a few of the many options available in vi. The editor has a steep learning curve, but it's a powerful tool once you get the hang of it.

c)Execute the following commands: sed,awk,ping,ssh,ftp,telnet

1. sed:

- Replace all occurrences of "old" with "new" in file.txt and save the changes to a new file:

Syntax: `sed 's/old/new/g' file.txt > new_file.txt`

Example:

Input file.txt:

```
Hello old world
This is an old file
```

- Command:

```
sed 's/old/new/g' file.txt > new_file.txt
```

le..txt

- Output new_file.txt:

```
Hello new world
This is an new file
```

2.awk

Print the second field (separated by commas) of each line in file.txt:

```
awk -F, '{print $2}' file.txt
```

- Input file.txt:

```
John,Doe,25  
Jane,Smith,30
```

- Command:

```
awk -F, '{print $2}' file.txt
```

- Output:

```
Doe  
Smith
```

Smith

3. ping:

The ping command in Linux is used to test the connectivity between two network hosts by sending ICMP (Internet Control Message Protocol) echo request packets to the destination host and waiting for an ICMP echo reply response. This is useful for diagnosing network connectivity issues and testing network latency.

Here is an example of using the ping command in Linux:

- Input:

```
ping 8.8.8.8
```

This command pings the IP address 8.8.8.8, which is a well-known DNS server provided by Google.

- Output:

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.  
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=10.1 ms  
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=9.92 ms  
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=10.0 ms  
64 bytes from 8.8.8.8: icmp_seq=4 ttl=118 time=9.91 ms  
64 bytes from 8.8.8.8: icmp_seq=5 ttl=118 time=10.1 ms  
^C  
--- 8.8.8.8 ping statistics ---  
5 packets transmitted, 5 received, 0% packet loss, time 4004ms  
rtt min/avg/max/mdev = 9.910/10.025/10.104/0.080 ms
```

In this example, the ping command sent 5 ICMP echo request packets to 8.8.8.8, and received 5 ICMP echo reply responses back. The output shows the round-trip time (RTT) for each packet, as well as statistics about the packet transmission, including the packet loss rate and the minimum/average/maximum RTT.

Note that the ping command can also take various options, such as the number of packets to send, the time interval between packets, and the packet size. You can refer to the `man ping` command to learn more about these options.

4.ssh Command

The ssh command in Linux is used to establish a secure encrypted connection between two computers over an unsecured network, such as the internet. It allows a user to log into a remote machine and execute commands as if they were sitting at the console of that machine.

Here are some common use cases for the ssh command in Linux:

- Remote login: Using ssh, a user can log in to a remote machine and work as if they were physically present at that machine.
- File transfer: ssh can be used to securely transfer files between two machines, using the scp (secure copy) command.

- Remote execution: ssh can be used to execute a command or script on a remote machine without logging in, using the ssh command with the remote command specified as an argument.

Here is an example of **using the ssh command in Linux**:

Input: `ssh user@example.com`

This command connects to the remote machine with the hostname example.com as the user user. The user will be prompted to enter their password to authenticate.

Output:

```
The authenticity of host 'example.com (203.0.113.0)' can't be established.  
RSA key fingerprint is 1c:67:56:fe:18:27:68:3a:94:2d:bd:9c:67:86:dd:f0.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added 'example.com,203.0.113.0' (RSA) to the list of known  
user@example.com's password:  
Last login: Fri Mar 11 12:01:25 2023 from 203.0.113.1  
  
user@example:~$
```

In this example, the user has connected to the remote machine example.com as the user user and has been prompted to accept the host's RSA key fingerprint. The user then authenticated using their password and is now logged into the remote machine, with the remote machine's prompt displayed.

Note that the ssh command can also take various options, such as specifying the port number, using a private key for authentication, or forwarding ports. You can refer to the man ssh command to learn more about these options.

5.Ftp

The `ftp` command in Linux is a client program used to transfer files between two machines over the File Transfer Protocol (FTP). It allows a user to connect to an FTP server, navigate through directories on the server, and transfer files to and from the server.

Here are some common use cases for the `ftp` command in Linux:

- Uploading files: A user can upload files from their local machine to an FTP server using the `put` command.
- Downloading files: A user can download files from an FTP server to their local machine using the `get` command.
- Navigating directories: `ftp` allows a user to navigate through the directories on the server using the `cd` command.

Here is an example of using the `ftp` command in Linux:

Input:

```
ftp example.com
```

This command connects to the FTP server with the hostname `example.com`. The user will be prompted to enter their username and password to authenticate.

```
Connected to example.com.
220 (vsFTPd 3.0.3)
Name (example.com:user): user
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

- Output:

In this example, the user has connected to the FTP server `example.com` and authenticated using their username and password. The output shows that the connection is successful, and the remote system is identified as a UNIX system. The user is now at the `ftp>` prompt and can start navigating and transferring files.

Note that the `ftp` command can also take various options, such as specifying the transfer mode, using passive mode, or setting up a bookmark. You can refer to the `man ftp` command to learn more about these options.

6. The `telnet` command in Linux is a client program used to establish a command-line connection to a remote machine over the Telnet protocol. Telnet is an unencrypted protocol, and therefore, it is not recommended to use Telnet for any sensitive data or login credentials.

Here are some common use cases for the `telnet` command in Linux:

- Remote login: Using `telnet`, a user can remotely log in to a machine and work as if they were physically present at that machine.
- Remote execution: `telnet` can be used to execute a command or script on a remote machine without logging in, using the `telnet` command with the remote command specified as an argument.

Here is an example of using the `telnet` command in Linux:

- Input: `telnet example.com`

This command connects to the remote machine with the hostname `example.com` using the Telnet protocol. The user will be prompted to enter their username and password to authenticate.


```
Trying 203.0.113.0...
Connected to example.com.
Escape character is '^]'.
Ubuntu 18.04 LTS
example login: user
Password:
Welcome to example.com!

user@example:~$
```

- Output:

In this example, the user has connected to the remote machine example.com using Telnet and has been prompted to enter their username and password to authenticate. The user then authenticated using their password and is now logged into the remote machine, with the remote machine's prompt displayed.

Note that Telnet is an unencrypted protocol, and all the communication, including the authentication process, is sent in plain text. Therefore, it is not recommended to use Telnet for any sensitive data or login credentials. It is recommended to use the Secure Shell (SSH) protocol instead, which is an encrypted protocol and provides better security.

d)Explore the use of grep command to perform the following operations:search and find files, filter files, display the number of lines before and after a search string.

The grep command in Linux is a powerful tool that can be used to search for text patterns in files, filter files based on patterns, and display the number of lines before and after a search string.

1.Search and find files:

To search for a pattern in one or more files, you can use the following command:

```
grep 'pattern' file1.txt file2.txt file3.txt
```

This command will search for the string "pattern" in the files file1.txt, file2.txt, and file3.txt. If the pattern is found in any of the files, grep will display the matching lines.

To search for a pattern in all files within a directory and its subdirectories, you can use the `-r` flag: `grep -r 'pattern' /path/to/directory`

This command will search for the pattern in all files within the directory `/path/to/directory` and its subdirectories.

2. Filter files:

To filter files based on a pattern and save the output to a new file, you can use the following command:

```
grep 'pattern' file.txt > filtered_file.txt
```

This command will search for the pattern in the file `file.txt` and save the matching lines to a new file called `filtered_file.txt`.

To filter files based on a pattern and exclude the matching lines, you can use the `-v` flag:

```
grep -v 'pattern' file.txt > filtered_file.txt
```

This command will search for lines that do not match the pattern in the file `file.txt` and save them to a new file called `filtered_file.txt`.

3. Display the number of lines before and after a search string:

To display a certain number of lines before and after a search string, you can use the `-B` and `-A` flags respectively:

```
grep -B 3 -A 2 'pattern' file.txt
```

This command will search for the pattern in the file `file.txt` and display 3 lines before and 2 lines after each matching line.

