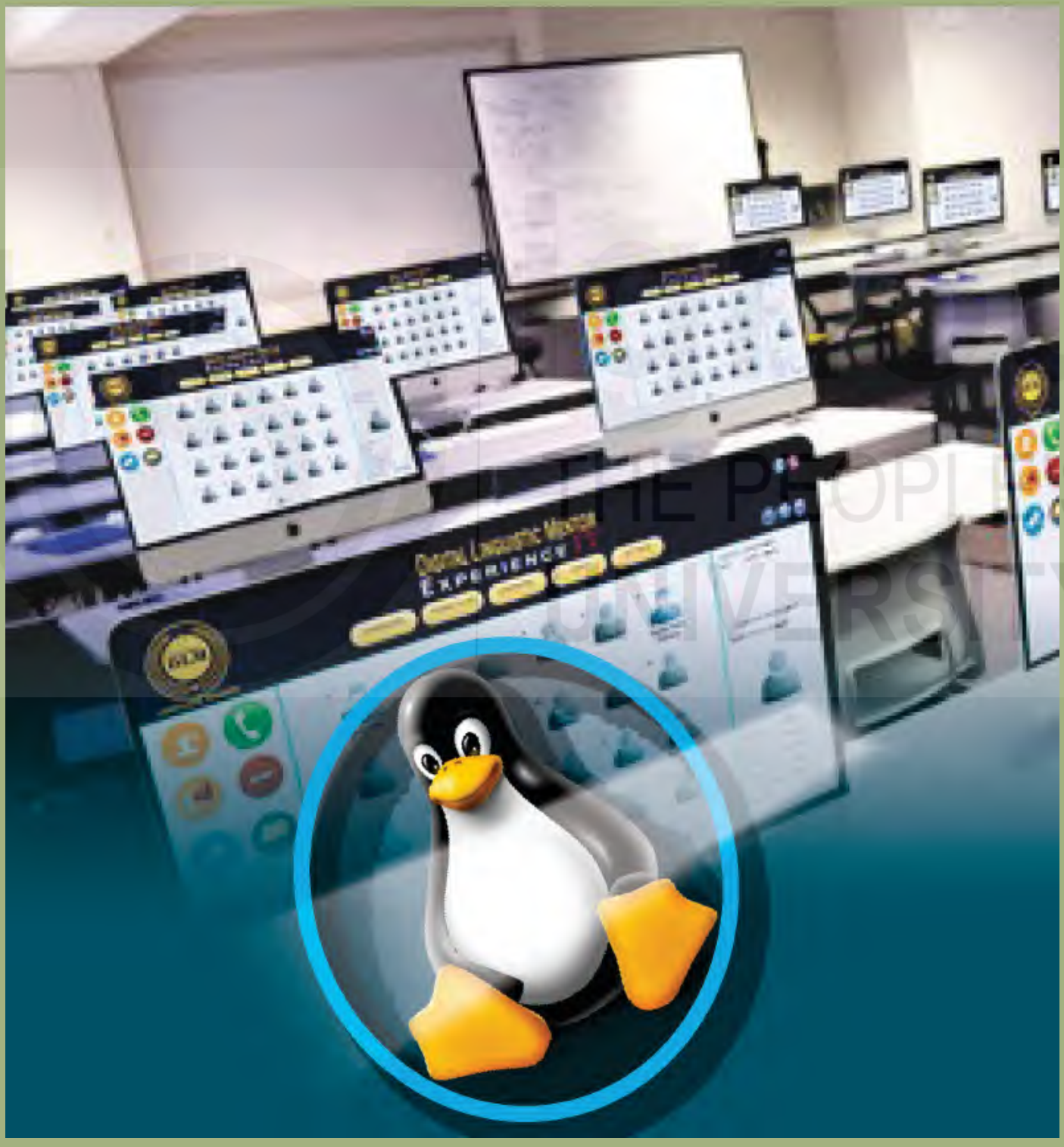
ignou
THE PEOPLE'S
UNIVERSITY

Indira Gandhi National Open University
School of Computer and Information
Sciences (SOCIS)

MCSL-204
WINDOWS AND LINUX LAB



Indira Gandhi National Open University
School of Computer & Information Sciences

MCSL-204
WINDOWS AND LINUX LAB

LAB MANUAL

PROGRAMME/COURSE DESIGN COMMITTEE

Prof. (Retd.) S.K. Gupta
IIT, Delhi

Prof. T.V. Vijay Kumar
Dean,
School of Computer & System Sciences,
JNU, New Delhi

Prof. Ela Kumar,
Dean, Computer Science & Engg
IGDTUW, Delhi

Prof. Gayatri Dhingra
GVMITM, Sonipat, Haryana

Mr. Milind Mahajani
Vice President
Impressico Business Solutions
Noida UP

Prof. V.V. Subrahmanyam
Director
SOCIS, IGNOU, New Delhi

Prof P. Venkata Suresh
SOCIS, IGNOU, New Delhi

Dr. Shashi Bhushan
Associate Professor
SOCIS, IGNOU, New Delhi

Shri Akshay Kumar
Associate Professor
SOCIS, IGNOU, New Delhi

Shri M. P. Mishra
Associate Professor
SOCIS, IGNOU, New Delhi

Dr. Sudhansh Sharma
Asst. Professor
SOCIS, IGNOU, New Delhi

BLOCK PREPARATION TEAM

Prof. D. P. Vidyarthi
(Content Editor)
New Delhi

Prof. V.V. Subrahmanyam
SOCIS, IGNOU

Prof. K. Swathi
(Unit Writer)
NRI Institute of Technology
Vijayawada, Andhra Pradesh

Prof (Retd) Anju Sahgal Gupta
(Language Editor)
SOH, IGNOU

Course Coordinator: Prof. V.V. Subrahmanyam

PRINT PRODUCTION

Mr. Tilak Raj
Assistant Registrar
MPDD, IGNOU, New Delhi

June, 2021

ISBN : 978-93-91229-73-3

Further information, about the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110 068.

Printed and published on behalf of the Indira Gandhi National Open University by Registrar, MPDD, IGNOU, New Delhi

Laser Composed by Tessa Media & Computers, C-206, Shaheen Bagh, Jamia Nagar, New Delhi

Printed at : Dee Kay Printers, 5/37 A, Kirti Nagar Indl. Area, New Delhi - 110 015

COURSE INTRODUCTION

This laboratory course is in continuation with MCS-203 *"Operating Systems"*. *This block is on practical problems of WINDOWS 10 and LINUX.* This lab course is an attempt to upgrade and enhance your theoretical skills obtained from theory to a practical environment. This lab course consists of **two credits**.

An operating system (OS) is system software of a computer system that is responsible for the management and coordination of activities and the sharing of the resources of the computer. The OS acts as a host for application programs that are run on the machine. As a host, one of the purposes of an OS is to handle the details of the operation of the hardware. This relieves application programs from having to manage these details and makes it easier to write applications. Almost all computers use an OS of some type. An operating system performs all the basic functions like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Also OS offers a number of services to application programs and users. Applications access these services through application programming interfaces (APIs) or system calls. By using these interfaces, the application can request a service from the OS, pass parameters, and receive the results of the operation. Users may also interact with the OS by typing commands or using a graphical user interface (GUI).

This lab course consists of 2 sections and is organized as follows:

Section – 1 WINDOWS 10 In Block-4 of MCS-203, as the part of the course you were introduced to four case studies among which WINDOWS and LINUX are very important. This section will help to explore and brush-up your practical skills on WINDOWS OS. Attempt all the problems given session-wise for 10 sessions.

Section – 2 LINUX This section will help to explore practical skills for LINUX OS. The list of commands and provided as ready reference. Practice them and also attempt all the tasks given session-wise for 10 sessions.

I wish you an eventful and interesting journey to the first semester laboratory world!

ignou
THE PEOPLE'S
UNIVERSITY

SECTION 1 WINDOWS 10

Structure **Page No.**

1.0 INTRODUCTION

This is the lab course, wherein you will have the hands on experience. You have studied the course material (MCS-203 Operating Systems). A list of tasks to be performed (sessionwise) on WINDOWS 10 is given. Please go through the general guidelines and the program documentation guidelines carefully.

1.1 OBJECTIVES

After completing this lab course you will be able to:

- Apply the concepts that have been covered in the theory course;
- Understand the features WINDOWS;
- Gain experience in overall interface;
- Make use of different functions/operations of WINDOWS;
- Work with the utilities in WINDOWS;
- Feel more confident of downloading and configuring APPS, applications etc..
- Write simple C programs and simulate the functionalities for the given tasks and;
- Know the alternative ways of providing solution to a given problem.

1.2 SALIENT FEATURES OF WINDOWS 10

Windows 10 is a Microsoft operating system for personal computers, tablets, embedded devices and internet of things devices. Microsoft released Windows 10 in July 2015 as a follow-up to Windows 8.

With Windows 10, Microsoft is trying to keep some of the touch and tablet features it created for Windows 8, combine them with the familiar Start menu and Desktop, and run it all on top of an improved operating system with more security, a new browser, the Cortana assistant, its own version of Office for on-the-go editing and plenty of new features intended to make life simpler.

On top of that, Windows 10 is more than just a PC operating system; it's also what will run on Windows phones – and on small tablets as well, because a 6-inch phone and a 7-inch tablet aren't such very different devices. Microsoft is expecting people to put Windows 10 on a billion devices.

Microsoft has said it will update Windows 10 continuously, rather than release a new, full-fledged operating system as a successor. The screenshot of the Windows 10 Desktop is given in fig 1.

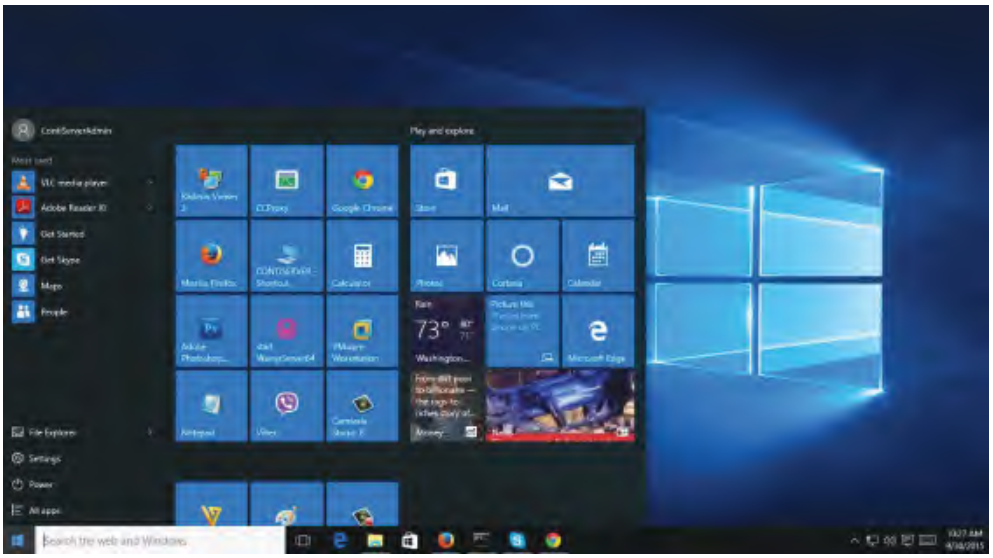


Fig 1: Windows 10 Desktop

The following are the salient features of Windows 10:

The Start Menu

The full-screen Start screen of Windows 8 is back to being a Start menu in Windows 10 that tries to combine the best of both options. You get a scrolling Start menu that's restricted to a single column, with jump lists and fly-out menus for extra options, divided into frequently used and recently installed programs, with the option to switch to a scrolling view of all your applications, sorted alphabetically. You also get an extra pane where you can pin Windows 8-style tiles, complete with 'rotating 3D cube' animations of live tiles. You can drag the Start menu to be a larger size or even set it to be full screen.

Cortana

Cortana, the Windows Phone assistant, shows up in Windows 10 as a search pane on the taskbar, which you can also trigger by saying “Hey Cortana” – and when you start searching the Start menu. Cortana gets you apps you have installed, documents you have access to, apps you could install from the Store, search results from the web and a range of other information – including from apps and services that integrate with Cortana. You can set reminders for different times and places that appear on other Cortana devices, so you can get your Microsoft Band to remind you to take the rubbish out as you walk up to your front door.

Task Switcher

Most Windows users don't know the Alt-Tab keyboard combination to see and switch between all running apps, so as well as having a redesigned task switcher with bigger thumbnails, Windows 10 also puts a task view icon in the taskbar to help them find it.

Taskbar

The Cortana search bar and task switcher button take up a large chunk of the taskbar, which is also rather more subtle about showing you which icons are for open programs, with just an underlined bar rather than a full highlight. The standard tools in the system tray all get updates to the new Windows 10 look, with a new menu showing available Wi-Fi, a new volume slider and a power monitor that also lets you change screen brightness.

Snap Assist

Because all your apps and programs run in windows on the desktop, instead of modern apps from the Store being in their own space, you can no longer drag across the left edge of the screen to bring another app on screen and get a split view. Instead, you drag windows into the corners of the screen to get the familiar Snap view. You can now use all four corners of your screen if you want each window to take up a quarter of the screen instead of half, and the space that isn't filled by the window you just dragged shows thumbnails of your other windows to make it easier to snap the next one into place.

Command Prompt

Learner who use the command prompt have been stuck with pretty much the same experience since the 1990s, but in Windows 10 you can finally resize the command prompt window and use familiar keyboard shortcuts to copy and paste at the command prompt.

Universal apps - including Office

Windows 10 gets a new Windows Store, where you can download desktop programs as well as modern Windows apps. Many of those apps will be universal apps that are the same code on a PC, a Windows phone, an Xbox One and even on HoloLens, with the interface changing to suit the different screen sizes. The Office for Windows apps like Word and Excel are universal apps, as are the Outlook Mail and Calendar apps.

File Explorer / Windows Explorer

File Explorer is also known as Windows Explorer. It has a core part of the operating system. You can use the shortcut “Windows + e” on your keyboard. When file Explore opens, we can access the frequently used folders and recently use files. The new “Home” view in Explorer shows you a Quick Access list of useful locations and folders you visit frequently, with a list of recently opened files underneath it, which is faster than having to go to the Recent Places link in older versions of Windows. The Share tab on the ribbon gets a makeover too.

The New Edge browser

To catch up with fast-moving browsers like Chrome and Firefox, Microsoft took its browser back to basics, ripping out years of code that didn't fit with web standards and making a lean, fast browser.

Improved Multitasking

A new Multiple Desktops feature lets you run another set of windows as if on another screen, but without the physical monitor. Instead of having multiple windows open on top of each other on one desktop, you can set up a whole other virtual desktop for those programs to reside in. Set up one specifically for home and leave your apps such as Netflix and Amazon open, and create another desktop for work on which you keep Word, Excel and Internet Explorer open. With the new desktops comes a new way to keep track of your open apps on Windows 10. On the new operating system, you can either hit the new Task View button on the task bar or swipe in from the left edge of the screen to pull up a one-page view of all your open apps and files. It's not much different from using the Alt-Tab combination shortcut on your keyboard, but this presents a convenient way for touch-oriented users to get an overview of what's running.

Calculator and Maps

Microsoft has developed a standard calculator in window 10. As we know that desktop Calculator in window 7 and window 8/8.1 sports a simple interface but it's very powerful. The new standard style Calculator in Window 10 is well-looking.

Action Center

If you've used Windows Phone 8.1 (or Android and/or iOS), you're used to a notification centre you can drag down from the top of the screen. Windows 10 puts that on the right of the screen, where the charms bar was in Windows 8, with notifications from various apps at the top and your choice of various settings buttons at the bottom for quick access.

Xbox and Streaming

Microsoft developed the Xbox app for game streaming to Windows 10 back in January; we can meet up with friends online, see what your friends are playing. The Xbox app dashboard has been updated to support the new feature; the Xbox one app for Windows 10 is not quite ready yet. This feature allows you to leave your living room and play your favorite Xbox One games anywhere with access to your home network.

Desktop and Security Improvements

Microsoft gives the features of upgraded task manager, it's so easier to what's requirements resources of your system and even manages a startup program without third party software. Windows 10 includes Windows Defender by default, Window Defender is just changed the version of Microsoft Security Essentials. Window Defender has antivirus protection to safe your system.

Continuum - on phones as well as PCs

You can change the look of Windows 10 on a touch-screen PC by turning on tablet mode – either as a setting or by removing or folding away the keyboard on a two-in-one or convertible PC. That takes away the normal taskbar, giving

the user one with just a Windows button (which opens a full-screen Start menu that shows the tiles and hides the scrolling list of programs), a back button, Cortana and the task switcher button.

Settings and Control Panel

The Windows 8 Settings app has taken over many more of the settings that used to be in Control Panel, and it has a Control Panel-style interface with icons to navigate with. But the old Control Panel interface is still there, for settings that aren't in the new Settings app (or if you're just used to finding things there).

DirectX 12

Windows 10 includes the latest version of Microsoft's graphics API – which has major performance improvements but will also work with many existing graphics cards. That's not just good news for gamers; it will speed up any Direct3D apps that are written to DirectX 12, which will include CAD software and other demanding graphics tools.

Phone Companion

Windows 10 includes a new app to help you get your phone set up to work with your PC and with any Microsoft services you use – like Cortana, Skype, Office and OneDrive. So you can plug in an iPhone and set it up to back up photos to OneDrive or get your Xbox Music tracks on an Android phone.

Other Features

Device Guard: A feature that helps protect a system by locking a device so that it can only run trusted applications

DirectAccess: An advanced VPN technology that allows remote users to securely access internal network file shares while connected to the internet

Encrypting File System (EFS): A feature that provides transparent file-level encryption

Enterprise Mode Internet Explorer (EMIE): A compatibility mode that runs Internet Explorer 11 or higher and lets websites render using a modified browser configuration that's designed to emulate either Windows Internet Explorer 7 or 8, avoiding the common compatibility problems associated with web apps written and tested on older versions of Internet Explorer

Group Policy management: An infrastructure that allows you to centrally manage computer settings and configuration

Joining to a domain: A feature that allows you to join an Active Directory domain

Long-Term Servicing Branch: An option for organizations that only want to receive features updates every two to three years, so that the current systems

can be stable • Private catalog: A feature that provides a list of applications that users within the organization can download apps from

Remote Desktop: A program or feature that allows you to connect to a remote computer and access the desktop and applications as if you were accessing the machine directly

RemoteApp: A feature that enables you to run a program remotely through Remote Desktop Services, although the application appears to be running on your local machine

User Experience control and lockdown: A feature that allows you to customize and lock down the Windows 10 user interface

Virtual desktops: A feature that allows you to run and switch between multiple desktops

Windows Hello: A credential technology that provides multi-factor authentication, including the recognition of a personal identification number (PIN) or biometrics (face, iris, or fingerprint recognition)

Windows Spotlight: An option that displays a new image on the lock screen each day

Windows To Go: A feature that allows you to boot and run Windows from USB mass storage devices such as USB flash drives and external hard drives

Windows Update for Business: A free service for Windows 10 Pro, Enterprise, and Education editions that can provide updates to users based on distribution rings

Windows 10 System Requirements

The minimum Windows 10 hardware requirements for a PC or 2-in-1 device are:

Processor	:	1 gigahertz (GHz) or faster processor or system-on-a-chip (SoC)
RAM	:	1 gigabyte (GB) for 32-bit or 2 GB for 64-bit
Hard disk space	:	16 GB for 32-bit OS 20 GB for 64-bit OS
Graphics card	:	DirectX 9 or later with Windows Display Driver Model 1.0
Display	:	800x600

1.3 GENERAL GUIDELINES

Following are some of the general guidelines:

- Observation book and Lab record are compulsory.
- You should attempt all problems/assignments given in the list session wise.
- For the tasks related to the working with the WINDOWS utilities, describe the procedure and also present screenshots wherever applicable.
- You may seek assistance in doing the lab exercises from the concerned lab instructor. Since the assignments have credits, the lab instructor is obviously not expected to tell you how to solve these, but you may ask questions concerning the Windows Operating system.

- The comment block above the main function should describe the purpose of the program. Proper comments are to be provide where and when necessary in the programming.

- The program should be interactive, general and properly documented with real Input/ Output data.

- If two or more submissions from different students appear to be of the same origin (i.e. are variants of essentially the same program), none of them will be counted. You are strongly advised not to copy somebody else's work.

- It is your responsibility to create a separate directory to store all the programs, so that nobody else can read or copy.

- The list of the programs(list of programs given at the end, session-wise) is available to you in this lab manual. For each session, you must come prepare with the algorithms and the programs written in the Observation Book. You should utilize the lab hours for executing the programs, testing for various desired outputs and enhancements of the programs.

- As soon as you have finished a lab exercise, contact one of the lab instructor / incharge in order to get the exercise evaluated and also get the signature from him/her on the Observation book.

- Completed lab assignments should be submitted in the form of a Lab Record in which you have to write the algorithm, program code along with comments and output for various inputs given.

- The total no. of lab sessions (3 hours each) are 10 and the list of assignments is provided session-wise. It is important to observe the deadline given for each assignment.

In the next section, the lists of lab assignments to be performed sessionwise are given.

1.4 LIST OF LAB ASSIGNMENTS – SESSIONWISE

Session 1: Accessing Windows 10

a) Get Started with Windows

b) Navigate the Windows 10 Desktop

c) Use the Start Menu and use the Apps

d) Understanding User Accounts (Administrator, Standard User and Guest)

e) Configuring and Optimizing User Account Control (UAC)

f) Working with various categories of Windows Settings (System, Devices, Network and Internet, Personalization, Accounts, Time and language, Ease of Access, Privacy and Update & Security)

g) Using Control Panel

Session 2: Working with Administrative Tools and other Settings

a) Work with Common Administrative Tools (Component Services, Computer Management, Event Viewer, Print Management, Task Scheduler etc..)

b) Changing date and time

c) Configuring the Desktop, Taskbar Settings, Start Menu, Display settings, Power settings, accessibility settings,

d) Creating and managing shortcuts
e) Personalizing your Windows Desktop
f) Choose your own desktop pictures, window colors, and sound scheme

Session 3: Using Windows Universal Apps, Desktop Applications and Accessory Programs

a) Use Desktop Applications
b) Use Windows Universal Apps
c) Multitask with Open Apps
d) Install Apps from Windows Store
e) Configuring Internet Explorer / Edge
f) Managing Cookies and Privacy settings
g) Creating and managing Checkpoints
h) Managing Favourites
i) Managing LAN Settings
j) Working with Calculator, Notepad, Snipping Tool
k) Working with Windows Media Player, WordPad, Paint, Math Input Panel
l) Working with Command Prompt, Run, Sticky notes, Sync Center

Session 4: Working with Files and Folders

a) Manage Files and Folders with File Explorer
b) Store and Share Files with OneDrive
c) Using Cortana, Edge and Cortana
d) Get to Know Cortana
e) Use Cortana as a Personal Assistant
f) Browse the Web with Edge
g) Get to know OneDrive (storage medium)

Session 5: Configuring Remote Desktop

a) Configuring Remote Desktop
b) Configuring Remote Assistance
c) Configuring Remote Management Settings
d) Using the Microsoft Management Console to Manage systems remotely
e) Using Remote Windows PowerShell

Session 6: Managing Applications, Services and Disks

a) Configuring Desktop Apps
b) Uninstall an App / Change a Program
c) Configuring Startup Options
d) Configuring the Windows Store
e) Using MSConfig
f) Understanding Storage Devices / External storage devices and accessing them
g) Disk management (to configure disks, volumes and file systems)
h) Understanding BitLocker

Session 7: Installing, Removing and Managing Devices

a) Manage Peripheral Devices
b) Manage Printers
c) Understanding Device Drivers
d) Update a device driver

e) Using Device Manager (Device by type, Devices by Connection, Resources by type, Resources by Connection)

Session 8: Using Built-in Maintenance Tools

a) Using Disk Fragmentor
b) Using Disk Cleanup
c) Using Task Scheduler
d) Using Action Center
e) Maintaining the Windows Registry
f) Configuring and Managing Updates
g) Managing update history and Rolling back updates

Session 9: Using Windows 10 Security Features

a) Manage Passwords and Privacy Levels
b) Using Windows Defender
c) Defending system from Malicious Software
d) Understanding Windows Firewall
e) Managing Client Security Using Windows Defender

Session 10: Using Backup and Recovery

a) Understanding Local, Network and Automated Backup Methods
b) Restoring Previous Versions of Files and Folders
c) Scheduling a Backup to include system image
d) Configuring system recovery / file recovery

1.5 FURTHER READINGS

1) Mike Mcgrath, Windows 10 in Easy Steps, BPB, 2017.
2) Lambert Joan, Lambert Steve, Windows 10 Step by Step, PHI, 2016.
3) James Jordan, Windows 10 for Beginners, 2020
4) Geoff Adams, Windows 10 – A Complete Guide and User Manual for Beginners, 2019

SECTION 2 LINUX

Structure **Page Nos.**

2.0 INTRODUCTION

This is the lab course, wherein you will have the hands on experience. You have studied the course material (MCS-203 Operating Systems). A list of tasks to be performed (sessionwise) on LINUX is given along with some programming problems towards the end. Please go through the general guidelines and the program documentation guidelines carefully.

2.1 OBJECTIVES

After completing this lab course you will be able to:

- Apply the concepts that have been covered in the theory course;
- Understand the features LINUX;
- Gain experience in overall command line interface as well as GUI;
- Make use of different functions/operations of LINUX;
- Work with the utilities in LINUX;
- Write simple shell programs and simulate the functionalities for the given tasks;
- To understand and make effective use Shell scripting language Bash to solve Problems.
- To implement in C some standard Linux utilities such as ls, mv, cp etc. using system calls.
- To develop the skills necessary for systems programming including file system programming, process and signal management, and interprocess communication and;
- Know the alternative ways of providing solution to a given problem.

2.2 HISTORY OF UNIX AND LINUX

The tabular form given below shows the developments of UNIX and LINUX year wise.

Year	Developments
1965	Multics project begun as joint venture of AT&T, MIT, and GE to create a new operating system for the GE computer.
1969	AT&T Bell Labs researchers Ken Thompson, Dennis Ritchie, J. F. Ossanna, and R. H. Canaday create a prototype file management system as an alternative to MULTICS. Commercial systems at the time were written entirely in assembly language. One of the goals of UNIX is to have a small kernel written in assembler, and the rest in any high-level language. Unix also has a hierarchical file system and a collection of utility programs.
1970	Brian Kernighan coins the name UNICS (UNiplexed Information and Computing System). Unix development increases with the acquisition of a DEC (Digital Equipment Corporation) PDP-11, a state-of-the-art $65,000 computer with 24 kilobytes of RAM and 512 kilobytes of disk space. Thompson develops B as an alternative to FORTRAN and BCPL.
1971	1st Unix version, V1, used only within Bell Labs. Needing to justify the cost of development, UNIX is used (with the assembly-language-coded *troff*) in the Bell Labs patent department as a one of the first word-processing programs. B is improved upon and its successor is named C.
1972	M. D. McIlroy introduces the novel idea of 'pipes'. June – Version 2, 10 Unix installations.
1973	Version 3, 16 Unix installations. November – Version 4 is rewritten in C, easing the portability of Unix.
1974	S. R. Bourne develops the Bourne Shell (/bin/sh, indicated with a '$') June – Version 5 Estimated 50 Unix installations.
1975	AT&T leases Version 6 to universities at low cost, making UNIX use widespread. Thompson spends year at UC Berkeley, leads development of a BSD variant of Unix. UC Berkeley graduate student Bill Joy (who later starts Sun Microsystems) develops the C-shell (/bin/csh, indicated with a '%') and the vi text editor. TENEX-style C-shell developed (/bin/tcsh). David Korn from AT&T develops the Korn shell (/bin/ksh).
1976	Emacs originally written by Richard Stallman.
1977	1BSD released. Tom Duff and Byron Rakitzis develop the rc shell.
1978	Students at UC Berkeley, known as "Berkeley Software Distribution", develop their own variant of UNIX, called BSD. 2BSD released, 75 copies distributed. 600 Unix installations worldwide.
1979	Private companies begin porting commercial versions of Unix. BSD releases 3BSD. AT&T releases the 40KB-kernel Version 7.
1980	Microsoft releases Xenix, which is the first attempt to bring Unix to desktop computers. October - BSD releases 4.0 BSD
1982	AT&T releases its first commercial version of Unix, System III. *Ksh* was delivered working in 1982 to AT&T Bell labs.
1983	Computer Research Group (CRG), UNIX System Group (USG), and

	Programmer's WorkBench (PWB) merge to become UNIX System Development Lab. AT&T releases System V, incorporating Xenix and other variants. BSD releases 4.2BSD which includes complete implementation of TCP/IP networking protocols, including telnet and ftp. SVID, the System 5 Interface Definition, is released in an effort to standardize the UNIX flavors as much as possible.
1984	Estimated 100,000 UNIX installations worldwide. U.S. government charges AT&T with monopolistic practices and AT&T is forced to divest its interests. AT&T releases SVR2, incorporating many features from 4.2BSD X/Open consortium of vendors founded, eventually known as The Open Group, gets UNIX trademark. Richard Stallman develops GNU (GNU's Not UNIX) as a free UNIX clone.
1985	February - AT&T releases Version 8 Paul Falstad develops *zsh*.
1986	September - AT&T releases Version 9 DEC, which had been supporting VAX/VMS, is forced to acknowledge and support UNIX as an inexpensive alternative. *Rc* shell upgraded to *es*.
1987	Estimated 100,000 Unix installations worldwide.
1988	Unix International (UI) and Open Software Foundation (OSF) are formed. SVR4 releases as a combo of System V, BSD, and SunOS.
1989	October - AT&T releases V10, the final version. Wanting a free alternative to *ksh*, GNU advocates develop *bash* (Bourne-again shell).
1991	Unix Systems Laboratory (USL) spun off as a separate company, majority-owned by AT&T. OSF releases OSF/1. **Linus Torvalds releases Linux kernel (Linus's Minix, pronounced 'lin-ux')**
1992	July 14 - William and Lynne Jolitz release 386BSD as open source, eventually evolving into NetBSD, FreeBSD, and OpenBSD.
1993	4.4BSD released as final Berkely release. June 16 - Novell buys USL from AT&T.
1994	Torvalds and many others relase version 1.0 of the Linux kernel. Used with Stallman's GNU command-set, users around the world have access to a free UNIX variant known as GNU/Linux, or just Linux.

Let us study the features of LINUX operating system in the following section.

2.3 FEATURES OF LINUX

Following are some of the important features of Linux Operating System:

- **Portable –** Portability means software can works on different types of hardware in same way. Linux kernel and application programs support their installation on any kind of hardware platform.
- **Open Source –** Linux source code is freely available and it is community based development project. Multiple team's works in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User –** Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.

- **Multiprogramming –** Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System –** Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell –** Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs etc.
- **Graphical user interface (X Window System):** People think that Linux is a command line OS, somewhere its true also but not necessarily, Linux have packages which can be installed to make the whole OS graphics based as Windows.
- **Security –** Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Advantages of Open Source

There are various advantages of an open-source which includes:

1. It helps in distributing the software carrying its source codes freely to the users.
2. It supports the users to add new features, debug, and correct errors in the source code.
3. It helps in redistributing the new, improved source code back again free of cost to other users.

Linux distributions

Some popular LINUX distributions are:

- RedHat Enterprise Linux (RHEL)
- Fedora
- Ubuntu
- Debian
- SuSE Linux Enterprise Server (SLES)
- OpenSuSE
- Linux Mint

2.4 KERNEL AND THE SHELL

The main control program in a UNIX operating system is called the **kernel**. However, the kernel does not allow the user to give its commands directly; instead when the user types commands on the keyboard they are read by another program in the Operating System called a **shell** which parses, checks, translates and then passes them to the kernel for execution.

There are a number of different shells available, with names such as ***bash, zsh, Korn, Tcsh, Fish*** each with different rules of syntax; these are partly though not completely responsible for the diversity of LINUX. Later in our discussion we will see what are the criteria to select a shell. Once the command has been interpreted and executed, the kernel sends its reply, which may simply be a **prompt** for the next command to be entered, either directly to the display monitor. This is a program responsible for deciding where and in what form the output will appear on the display monitor. If for any reason the kernel cannot perform the command requested (wrong syntax), for example, the reply will be an error message; the user must then re-enter the corrected command.

2.4.1 Commands and Processes

The kernel and the shell programs running in the CPU are examples of **processes**; these are self-contained programs that may take over complete control of the CPU. Although there can only be one kernel process running in a particular CPU, there may be any number of shell and other processes, subject of course to memory limitations.

Some commands to the shell are **internal** (or **built-in**), that is, they only involve the shell and the kernel. Others are **external** and may be supplied with the OS, or may be user-written. An external command is the name of a file which contains either a single executable program or a **script**. The latter is a text file, the first line of which contains the name of a file containing an executable program, usually but not necessarily a shell, followed by a sequence of commands for that program. A script may also invoke other scripts – including itself. Its purpose is simply to avoid having to re-type all the command it contains.

A command may also be an **alias** for an internal or external command (*e.g.,* the user may not like the LINUX name "***rm***" for the command which deletes files, and may prefer to alias it to "delete").

The external command may optionally cause execution of the shell process to be temporarily suspended, and then run another program, which may then take over input from the keyboard and mouse and send output for display. The shell may or may not wait for the program to finish, before it wakes up again and cause its prompt to be displayed. It is very important that the user be continuously aware of which process is currently reading keyboard input: the shell or another program, because they usually speak completely different languages.

The above is an example of a **parent** process – the shell, and a **child** process – the external program. In fact the child could just as well have been, and often is, another invocation of the same shell, or of a different shell, and the child process can be the parent of other child and so on (almost) *ad infinitum*. Consequently a typical LINUX system has many processes either waiting or running.

2.4.2 LINUX Directory Structure

Like other Operating system's, LINUX organises information into **files**, and related files may be conveniently organised in **directories**. Files may contain text, data, executable programs, scripts (which are actually just data for a scripting program such as a shell), and may also be links to other files, or to physical devices or communications channels.

The directory structure is **hierarchical** with the **root directory**, indicated by a forward slash (/), at the base of the tree. This may contain files as well as other directories such as /bin, /etc, /lib, /tmp, /usr, and /d. Actually in this example the root directory will contain directory files called bin, etc, lib, tmp, usr and d, each of which contains a list of files and their locations on the disk for each of the corresponding directories. Each directory may contain further ordinary files and directory files, and so on. The / character is used to delimit the components of the name. For example, the /d directory may contain directories such as /d/user1 and /d/user2 and these may contain the user's **home directories.**

The entire directory hierarchy may reside on a single physical disk, or it may be spread across several disks; the boundaries between physical disks cannot be seen merely by looking at the directory hierarchy. Your system administrator will decide where your home directory is to be both physically and logically located.

Every file in the hierarchy is identified by a **pathname**, this is merely a description of the path you have to traverse to get from the root directory to the file. Strictly, a **filename** is distinct from a pathname; a filename is just one of the components of the pathname delimited by /'s.

Relative pathnames which have a start point anywhere but / in the hierarchy may be used instead, and are often more convenient.

A standard **Linux** distribution follows the directory structure as provided in the Table 1 given below:

Table 1: Directories in LINUX and their Content

Directory	Contents / Purpose
/root	This is the home directory of root user and should never be confused with '/'
/bin	All the executable binary programs (file) required during booting, repairing, files required to run into single-user-mode, and other important, basic commands **viz.**, cat, du, df, tar, rpm, wc, history, etc.
/boot	Holds important files during boot-up process, including Linux Kernel.
/dev	Contains device files for all the hardware devices on the machine e.g., cdrom, cpu, etc
/etc	Contains Application's configuration files, startup, shutdown, start, stop script for every individual program.
/home	Home directory of the users. Every time a new user is created, a directory in the name of user is created within home directory which contains other directories like Desktop, Downloads, Documents, etc.
/lib	The Lib directory contains kernel modules and shared library images required to boot the system and run commands in root file system.
/lib : /lost+found	This Directory is installed during installation of Linux, useful for recovering files which may be broken due to unexpected shut-down.
/media	Temporary mount directory is created for removable devices viz., media/cdrom.
/opt	Optional is abbreviated as opt. Contains third party application software. Viz., Java, etc.
/mnt	Temporary mount directory for mounting file system .
/proc	A virtual and pseudo file-system which contains information about running process with a particular Process-id aka pid.
/run	This directory is the only clean solution for early-runtime-dir problem.
/sbin	Contains binary executable programs, required by System Administrator, for Maintenance. Viz., iptables, fdisk, ifconfig, swapon, reboot, etc.
/srv	Service is abbreviated as 'srv'. This directory contains server specific and service related files.
/sys	Modern Linux distributions include a /sys directory as a virtual filesystem, which stores and allows modification of the devices connected to the system.
/tmp	System's Temporary Directory, Accessible by users and root. Stores temporary files for user and system, till next boot.
/usr	Contains executable binaries, documentation, source code, libraries for second level program.
/var	Stands for variable. The contents of this file is expected to grow. This directory contains log, lock, spool, mail and temp files

Among the directories given in the table, lets us see the some important files, their location and usability in the following section.

Important Files, their location and Usability

Linux is a complex system which requires a more complex and efficient way to start, stop, maintain and reboot a system unlike Windows. There is a well defined configuration files, binaries, man pages, info files, etc. for every process in Linux. Some to the important files, their location and respective usability is compiled in the Table 2 given below:

Table 2: Some Important Files, location and Utility

Files	Usability
/boot/vmlinuz	The Linux Kernel file.
/dev/hda	Device file for the first IDE HDD (Hard Disk Drive).
/dev/hdc	Device file for the IDE Cdrom, commonly.
/dev/null	A pseudo device, that don't exist. Sometime garbage output is redirected to /dev/null, so that it gets lost, forever.
/etc/bashrc	Contains system defaults and aliases used by bash shell.
/etc/crontab	A shell script to run specified commands on a predefined time Interval.
/etc/exports	Information of the file system available on network
/etc/fstab	Information of Disk Drive and their mount point.
/etc/group	Information of Security Group.
/etc/grub.conf	grub bootloader configuration file.
/etc/init.d	Service startup Script.
/etc/lilo.conf	lilo bootloader configuration file.
/etc/hosts	Information of Ip addresses and corresponding host names
/etc/hosts.allow	List of hosts allowed to access services on the local machine.
/etc/host.deny	List of hosts denied to access services on the local machine.
/etc/inittab	INIT process and their interaction at various run level.
/etc/modules.conf	Configuration files for system modules.
/etc/passwd	Contains password of system users in a shadow file, a security implementation.
/etc/printcap	Printer Information.
/etc/profile	Bash shell defaults.
/etc/resolv.conf	Domain Name Servers (DNS) being used by System.
/etc/securetty	Terminal List, where root login is possible.

/usr/bin	Normal user executable commands.
/usr/include	Contains include files used by ‘c‘ program.
/usr/sbin	Commands for Super User, for System Administration.
/proc/cpuinfo	CPU **Information.**
/usr/lib	Library files which are required during program compilation.
/usr/share	Shared directories of man files, info files, etc..
/proc/filesystems :	File-system Information being used currently.
/proc/mount	Mounted File-system Information.
/version	Linux Version Information.
/var/log/wtmp	list login time and duration of each user on the system currently.
/var/log/lastlog	log of last boot process.

2.4.3 Wild Card Characters

Another shorthand character is the use of “wildcard” character in filenames, technically called filename **globbing**. The ***** character in a filename represents any string of characters, including no characters; the **?** character represents any single character. These wildcard characters may be used more than once, and may appear in combination in a pathname.

Always note that LINUX is always fussy about the case of letters in commands, usernames, passwords and filenames; so Vvs.data is not the same file as vvs.data.

2.4.4 Syntax of LINUX Commands

The general form of a LINUX command is:

command [option(s)] [argument(s)] (the [ ] here indicate that the items they contain are optional; they are not part of the syntax).

If a typing error is made the line may be changed using the left/right arrow keys to move the cursor and the *Backspace* key to delete the character to the left of the cursor; new characters are inserted before the cursor. After keying-in the desired command, press the *Enter* key to execute the command. The command may be cancelled before execution by using *Ctrl*-u (hold down the *Ctrl* key and press the u key). If an incorrectly spelled command is entered and spelling correction is enabled in the shell, the shell will attempt to correct the mistake and ask for verification.

If the shell has **filename completion** enabled, use of the *Tab* key after part of a command or filename has been typed will cause the shell to attempt completion of the name, up to the character where the result is unique. Use of the *Ctrl*-d key will cause all names that match what has been typed to be listed.

The options and arguments if present must be separated from the preceding item by at least one space. However, multiple options may or may not need to be separated from each other by at least 1 space; multiple arguments are always separated from each

other by at least 1 space. Options usually start with -, but occasionally it is a +, and sometimes the – or + may be omitted.

A line may contain several commands (each possibly followed by options and/or arguments) separated by semicolons (;). The commands are executed in sequence, just as if they had been typed on separate lines. If it is necessary to continue a command onto the next line, end the line with a backslash (\), press *Enter* and continue typing on the next line.

In a script anything after # on a line is treated as a ***comment***, *i.e.,* it is ignored.

To background a command simply add an ampersand (&) on the end. Commands may be piped using a vertical bar | to separate them. As an alternative to piping the output may be **redirected** to write to a file using > filename or appended to the file using >> filename. This only redirects the **standard output** stream; redirection of error messages (**standard error** stream) requires a different syntax which is shell-dependent. The **standard input** stream may be redirected to read from a file using <filename.

2.4.5 Getting Help

LINUX is an operating system, with hundreds of commands that can be combined to execute thousands of possible actions. It provides complete information about each and every command which is stored in the LINUX *man* pages (*man* stands for manual). We can go through them by giving the *man* command at the prompt as shown below:

$ man cp

$ man ls

$ man grep

2.5 LINUX COMMANDS

The following is a list of **commonly used commands** LINUX / UNIX which are organised under different categories for understanding and ease of use. Keys proceeded by a ^ character are CONTROL key combinations.

Terminal Control Characters

^h	backspace erase previously typed character
^u	erase entire line of input so far typed
^d	end-of-input for programs reading from terminal
^s	stop printing on terminal
^q	continue printing on terminal
^z	currently running job; restart with bg or fg
DEL, ^c	kill currently running program and allow clean-up before exiting
^\	emergency kill of currently running program with no chance of cleanup

Login and Authentication

login	access computer; start interactive session
logout	disconnect terminal session
passwd	change local login password; you MUST set a non-trivial password

System Information

uname -a	to display Linux system information
uname –r	to display kernet release information
uptime	to show how long the system has been running
hostname	to show system host name
last reboot	to show system reboot history
w	to display who is online

Information

date	show date and time
history	list of previously executed commands
pine	send or receive mail messages
msgs	display system messages
man	show on-line documentation by program name
info	on-line documentation for GNU programs
who	who is on the system and what are they doing
who am i	who is logged onto this terminal
top	show system status and top CPU-using processes
uptime	show one line summary of system status
finger	find out info about a user@system

File Management

cat	combine files
cp	copy files
ls	list files in a directory and their attributes
mv	change file name or directory location
rm	remove files
ln	create another link (name) to a file
chmod	set file permissions
des	encrypt a data file with a private key
find	find files that match specified criteria

Display Contents of Files

cat	copy file to display device
vi	screen editor for modifying text files
more	show text file on display terminal with paging control
head	show first few lines of a file(s)
tail	show last few lines of a file; or reverse line order
grep	display lines that match a pattern
lpr	send file to line printer
pr	format file with page headers, multiple columns etc.
diff	compare two files and show differences
cmp	compare two binary files and report if different
od	display binary file as equivalent octal/hex codes
file	examine file(s) and tell you whether text, data, etc.
wc	count characters, words, and lines in a file

Directories

cd	change to new directory
mkdir	create new directory
rmdir	remove empty directory (remove files first)
mv	change name of directory
pwd	show current directory

Devices

df	summarize free space on disk device

du	show disk space used by files or directories
fdisk –l	to display disks partition sizes and types

Hardware Information

dmesg	to display messages in kernel ring buffer
free –h	to display free and used memory
lspci –tv	to display PCI devices
lsusb –tv	to display USB devices
dmidecode	to display DMI/SMBIOS(h/w info) from the BIOS
hdparm – i/dev/sda	to show information about the disk sda
badblocks –s/dev/sda	to test for unreadable blocks on disk sda

Special Character Handling for C-shell

*	match any characters in a file name
~user	shorthand for home directory of "user"
$name	substitute value of variable "name"
\	turn off special meaning of character that follows
‘	In pairs, quote string w/ special chars, except !
“	In pairs, quote string w/ special chars, except !, $
`	In pairs, substitute output from enclosed command

Process Management

&	run job in background
DEL, ^c	kill job in foreground
^z	suspend job in foreground
fg	restart suspended job in foreground
bg	run suspended job in background
;	delimit commands on same line
( )	group commands on same line
!	re-run earlier command from history list
ps	print process status
ps –ef	to display all the currently running processes on the system
top	display and manage the top processes
kill	kill background job or previous process
nice	run program at lower priority
at	run program at a later time
crontab	run program at specified intervals
limit	see or set resource limits for programs
alias	create alias name for program (in .login)
sh, csh	execute command file

Controlling Program Input/Output for C-shell

|	pipe output to input
>	redirect output to a storage file
<	redirect input from a storage file
>>	append redirected output to storage file
tee	copy input to both file and next program in pipe
script	make file record of all terminal activity

E-mail and communication

pine	process mail with full-screen menu interface or read USENET news groups
msgs	read system bulletin board messages
mail	send e-mail; can be run by other programs to send existing files via e-mail

uuencode	uudecode encode/decode a binary file for transmission via mail
finger	translate real name to account name for e-mail
talk	interactive communication in real-time
rn	read USENET news groups

Editors and Formatting Utilities

sed	stream text editor
vi	screen editor
emacs	GNU emacs editor for character terminals
xemacs	GNU emacs editor for X-Windows terminals
pico	very simple editor, same as used in "pine"
fmt	fill and break lines to make all same length
fold	break long lines to specified length

Printing

lpr	send file to print queue
lpq	examine status of files in print queue
lprm	remove a file from print queue
enscript	convert text files to PostScript format for printing

Interpreted Languages and Data Manipulation Utilities

sed	stream text editor
perl	Practical Extraction and Report Language
awk	pattern scanning and processing language; 1985 vers.
sort	sort or merge lines in a file(s) by specified fields
tr	translate characters
cut	cut out columns from a file
paste	paste columns into a file
dd	copy data between devices; reblock; convert EBCDIC

Networking/Communications

telnet	remote network login to other computer
ftp	network file transfer program
rlogin	remote login to “trusted” computer
rsh	execute single command on remote “trusted” computer
rcp	remote file copy to/from “trusted” computer
host	find IP address for given host name, or vice versa
lynx	Web browser for character based (text-only) terminals
gzip, gunzip	compress/decompress a file
tar	combine multiple files/dirs into single archive
uuencode, uudecode	encode/decode a binary file for transmission via mail
ifconfig –a	to display all network interfaces and IP address
ping *host*	to send ICMP echo request to *host*
dig *domain*	to display DNS information for *domain*
host *domain*	to display DNS IP address for *domain*
netstat –nutlp	to display listening top and UDP ports and corresponding programs

Compilers, Interpreters and Programming Tools

csh	command language interpreter (shell scripts)
f77	DEC Fortran 77 compiler
f2c	convert fortran source code to C source code
cc, c89	DEC ANSI 89 standard C compiler
gcc	GNU C compiler
g++	GNU C++ compiler
pc	DEC Pascal compiler
dbx	symbolic debugger for compiled C or Fortran

make	recompile programs from modified source
gmake	GNU version of make utility
awk	interpreter for awk language
error	analyze and disperse compiler error messages.

Performance and Monitoring and Statistics

top	to display and manage the top processes
htop	interactive process viewer
mpstat 1	to display processor related statistics
vmstat 1	to display virtual memory statistics
iostat 1	to display I/O statistics
watch df –h	to execute df –h showing periodic updates

Installing Packages

yum –i *packagae.rpm*	to install package from local file named package.rpm
yum install *package*	install *package*

2.6 DESCRIPTION OF COMMONLY USED LINUX COMMANDS

The description for the most commonly used LINUX commands is given below in an alphabetic order.

cat

cat allows you to read multiple files and then print them out. You can combine files by using the > operator and append files by using >>.

Syntax: ***cat [argument] [specific file]***

Example:

cat abc.txt

If you want to append three files (abc.txt, def.txt, xyz.txt), give the command as,
cat abc.txt def.txt xyz.txt > all

cd, chdir

cd (or chdir) stands for "change directory". This command is the key command to move around your file structure.

Syntax: ***cd [name of directory you want to move to]***

When changing directories, start with / and then type the complete file path, like
cd /vvs/abc/xyz

chmod

chmod (which stands for "change mode") changes who can access a particular file. A "mode" is created by combining the various options from who, opcode, and permission.

Syntax: ***chmod [option] mode file***

If you look at a list of files using the long list command *ls –l*, you'll see the permissions, owner, file size, modification time, and filename. The first column of the list shows who can read, write, and execute the files or directories, in other words, the permissions. It basically shows who has permission to do what to a given file or

directory. ***r*** stands for "read" and means that you're allowed to read the file or directory. ***w*** stands for "write" and gives permission to edit or change the file as well as create, move, rename, or remove a directory. ***x*** stands for "execute" which gives permission to run a file or search a directory. Every file or directory has four sets of ***rwx*** permissions. The first set represents the user (u), the second set represents the group (g), the third set represents other (o), and the fourth set represents all (a). The column will look like this:

rwxrwxrwx

Each set of ***rwx*** represents user, group, and other respectively. Only the owner of a file or a privileged user may change the permissions on a file. There are two ways to change permissions on a file or directory, either numerically or by using lettered commands. Both ways use the command ***chmod***. To add permissions to a file, you use +, to remove permissions you use-.

For example, take a file:

-rw-r--r-- 1 yash mony 476 Apr 14 17:13 vvs.txt

To allow a group (mony, in this case) "write" access, you would type:

chmod g+w vvs.txt

If you wanted to remove "read" ability from "other" you would type:

chmod o-r vvs.txt

It is also possible to specify permissions using a three-digit sequence. This is a more efficient way to change permissions (or at least it requires less typing), so use this method if it doesn't confuse you. Each type of permission is given an octal value. Read is given the value of 4, write is given the value of 2, and execute is given the value of 1. These values are added together for each user category. The permissions are changed by using a three-digit sequence with the first digit representing owner permission, the second digit representing group permission, and the third digit representing other permission. For example, if you wanted to make vvs.txt readable, writable, and executable for the user, readable and writable for the group, and readable for other, you would type:

chmod 764 vvs.txt

The first digit means readable and writable for the user (4+2+1), the second digit means readable and writable for the group (4+2+0), and the third digit means readable for other (4+0+0).

If you want to change the permissions on a directory tree use the -R option. *chmod -R* will recursively change the permissions of directories and their contents.

chown

chown changes who owns a particular file or set of files. New owner files refer to a user ID number or login name that is usually located in the /etc/password directory. The owner of a file or directory can be seen by using the command.

Syntax: ***chown [option] newowner files***

Only the owner of a file or a privileged user can change the permissions on a file or directory. The following example changes the owner of vvs.txt to sridhar

chown sridhar vvs.txt

cp

The ***cp*** command copies files or directories from one place to another. You can copy a set of files to another file, or copy one or more files under the same name in a directory. If the destination of the file you want to copy is an existing file, then the existing file is overwritten. If the destination is an existing directory, then the file is copied into that directory.

Syntax: ***cp [options] file1 file2***

If you want to copy the file ***favourites.html*** into the directory called ***laksh***, you give the command as:

cp favourites.html /vvs/laksh/

A handy option to use with ***cp*** is ***-r***. This recursively copies a particular directory and all of its contents to the specified directory, so you won't have to copy one file at a time.

date

The ***date*** command can be used to display the date or to set a date.

Syntax: ***date [option] [+format]***
date [options] [string]

The first structure shows how date can be used to display the current date. A certain format can be specified in which the date should be displayed. Check the LINUX manual for specific formats and options. The second structure allows you to set the date by supplying a numeric string. Only privileged users will be able to use this second command structure.

diff

diff displays the lines that differ between two given files.

Syntax: ***diff [options] [directory options] file1 file2***

diff can be an extremely valuable tool for both checking errors and building new pages. If you run a ***diff*** between two files, you'll be shown what differences the files have line by line. The lines referring to file1 are marked with the < symbol. The lines referring to file2 are marked by the > symbol. If the file is a directory, diff will list the file in the directory that has the same name as file2. If both of the files are directories, diff will list all the lines differing between all files that have the same name.

If you have a file that is not working properly, it can be a great help to check it against a similar file that is working. It will often quickly alert you to a line of code that's missing.

A handy option to use if you want to generally compare two files without noting the complex differences between them is the -h option (h stands for half-hearted). Using -i

as an option will ignore differences in uppercase and lowercase characters between files, and -b will ignore repeating blanks and line breaks.

exit

The ***exit*** command allows you to terminate a process that is currently occurring.

For example, if you wanted to leave a remote host that you were logged onto (see rlogin also), you should type exit. This would return you to your home host.

find

find searches through directory trees beginning with each pathname and finds the files that match the specified condition(s). You must specify at least one pathname and one condition.

Syntax: ***find pathname(s) condition(s)***

There are several handy conditions you can use to find exactly what you want. The ***-name*** condition will find files whose names match a specified pattern. The structure for the ***name*** condition is:

find pathname -name pattern

The condition ***-print*** will print the matching files to the pathname specified. ***-print*** can also be used in conjunction with other conditions to print the output.

If you wanted to find all the files named favorites.html in the directory ***Ram***, then you'd do this:

find /Ram -name favorites.html –print

This looks through the directory ***Ram*** and finds all the files in that directory that contain favorites.html, then prints them to the screen. Your output would look like this:

/Ram/sixteen_candles/favorites.html
/Ram/favorites.html
/Ram/breakfast_club/favorites.html

All meta-characters (!, *, ., etc.) used with ***-name*** should be escaped (place a \ before the character) or quoted. Meta-characters come in handy when you are searching for a pattern and only know part of the pattern or need to find several similar patterns. For example, if you are searching for a file that contains the word "favorite", then use the meta-character * to represent matching zero or more of the preceding characters. This will show you all files which contain favorite.

find /Ram -name '*favorite*' -print

This looks through the directory ***Ram*** and finds all the files in that directory that contain the word "favorite". The output would look like this:

/Ram/sixteen_candles/favorites.html
/Ram/favorites.html
/Ram/least_favorites.html
/Ram/breakfast_club/favorites.html
/Ram/favorite_line.html

The ***-user*** condition finds files belonging to a particular user ID or name.

finger

finger displays information about various users as well as information listed in the .plan and .project files in a user's home directory. You can obtain the information on a particular user by using login or last names. If you use the latter, the info on all users with that last name will be printed. Environments that are hooked up to a network recognize arguments (users) in the form of user@host or @ host.

Syntax: ***finger [options] users***

grep

The ***grep*** command searches a file or files for lines that match a provided regular expression ("grep" comes from a command meaning to **g**lobally search for a **r**egular **e**xpression and then **p**rint the found matches).

Syntax: ***grep [options] regular expression [files]***

To exit this command, type 0 if lines have matched, 1 if no lines match, and 2 for errors. This is very useful if you need to match things in several files. If you wanted to find out which files in our ***vvs*** directory contained the word "***mca***" you could use ***grep*** to search the directory and match those files with that word. All that you have to do is give the command as shown:

grep 'mca' /vvs/*

The * used in this example is called a meta-character, and it represents matching zero or more of the preceding characters. In this example, it is used to mean "all files and directories in this directory". So, ***grep*** will search all the files and directories in ***vvs*** and tell you which files contain "***mca"***.

head

head prints the first couple of lines of one or multiple files. ***-n*** is used to display the first ***n*** lines of a file(s). The default number of lines is 10.

Syntax: ***head [-n] [files]***

For example, the following command will display the first 15 lines of favourites.html.

head -15 favourites.html

kill

kill ends the execution of one or more process ID's. In order to do this you must own the process or be designated a privileged user. To find the process ID of a certain job give the command ***ps***.

Syntax: ***kill [options] PIDs***

There are different levels of intensity to the *kill* command, and these can be represented either numerically or symbolically. ***kill -1*** or HUP makes a request to the server to terminate the process, while ***kill -9*** or ***kill KILL*** forces a process to terminate absolutely. Most politely, LINUX users will attempt to kill a process using -1 first before forcing a process to die.

less

less is similar to ***more*** in that it displays the contents of files on your screen. Unlike ***more, less*** allows backward and forward movement within the file. It does not read the whole file before displaying its contents, so with large files less displays faster than more. Press ***h*** for assistance with other commands or ***q*** to quit.

Syntax: ***less [options] [files]***

lprm

lprm removes printer queue requests.

Syntax: ***lprm /usr/ucb/lprm [optons] [job#] [users]***

The ***lprm*** command will remove a job or jobs from a printer's queue. If ***lprm*** is used without any arguments, it will delete the active job if it is owned by the user. If the command is used with **-**, then all the jobs owned by the user will be removed. To remove a specific job, use the job number.

ls

ls will list all the files in the current directory. If one or more files are given, ***ls*** will display the files contained within "name" or list all the files with the same name as "name". The files can be displayed in a variety of formats using various options.

Syntax: ***ls [options] [names]***

ls is a command you'll end up using all the time. It simply stands for list. If you are in a directory and you want to know what files and directories are inside that directory, type ***ls***. Sometimes the list of files is very long and it flies past your screen so quickly you miss the file you want. To overcome this problem give the command as shown below:

ls | ***more***

The character | (called pipe) is typed by using shift and the \ key. | ***more*** will show as many files as will fit on your screen, and then display a highlighted ***"more"*** at the bottom. If you want to see the next screen, hit enter (for moving one line at a time) or the spacebar (to move a screen at a time). | ***more*** can be used anytime you wish to view the output of a command in this way.

A useful option to use with ***ls*** command is ***-l***. This will list the files and directories in a long format. This means it will display the permissions (see chmod), owners, group, size, date and time the file was last modified, and the filename.

```
 drwxrwxr-x  vvs   staff  512  Apr 5 09:34 sridhar.txt
 -rwx-rw-r-- vvs   staff  4233 Apr 1 10:20 resume.txt
-rwx-r--r-- vvs   staff  4122 Apr 1 12:01 favourites.html
```

There are several other options that can be used to modify the ls command, and many of these options can be combined. ***-a*** will list all files in a directory, including those files normally hidden. ***-F*** will flag filenames by putting / on directories, @ on symbolic links, and * on executable files.

man

The ***man*** command can be used to view information in the online LINUX manual.

Syntax: ***man [options] [[section] subjects***]

man searches for information about a file, command, or directory and then displays it on your screen. Each command is a subject in the manual. If no subject is specified, you must give either a keyword or a file. You can also search for commands that serve a similar purpose. For example, if you want more information about the ***chmod*** command, you should type:

man chmod

A screen will then appear with information about ***chmod***. Type ***q*** to quit.

mkdir

mkdir creates a new directory.

Syntax: ***mkdir [options] directory name***

For example, to create a directory called ***parkhyath*** in the present working directory, give the command as,

mkdir prakhyath

more

more displays the contents of files on your screen.

Syntax: ***more [options] [files]***

To have the next line displayed, hit the return key, otherwise press the spacebar to bring up the next screen. Press h for assistance with other commands, n to move to the next file, or q to quit.

mv

mv moves files and directories. It can also be used to ***rename files or directories***.

Syntax: ***mv [options] source target***

If you wanted to rename vvs.txt to vsv.txt, you should give the command as:

mv vvs.txt vsv.txt

After executing this command, vvs.txt would no longer exist, but a file with name vsv.txt would now exist with the same contents.

passwd

The ***passwd*** command creates or changes a user's password. Only the owner of the password or a privileged user can make these changes.

Syntax: ***passwd [options] files***

ps

The ***ps*** command prints information about active processes. This is especially useful if you need to end an active process using the ***kill*** command. Use ***ps*** to find out the process ID number, then use ***kill*** to end the process.

Syntax: ***ps [options]***

pwd

pwd prints the pathname of the current directory. If you wanted to know the path of the current directory you were in you give the command as ***pwd***. You will get the complete path.

rlogin

The ***rlogin*** command, which stands for remote login, lets you connect your local host to a remote host.

Syntax: ***rlogin [options] host***

If you wanted to connect to the remote host ***vsmanyam*** and you were on ***sree***, you would do this:

rlogin vsmanyam password:***

You would then be at ***vsmanyam***

rm

rm removes or deletes files from a directory.

Syntax: ***rm [options] files***

In order to remove a file, you must have write permission to the directory where the file is located. While removing a which does't have write permission on, a prompt will come up asking you whether or not you wish to override the write protection.

The ***-r*** option is very handy and very dangerous. ***-r*** can be used to remove a directory and all its contents. If you use the ***-i*** option, you can possibly catch some disastrous mistakes because it'll ask you to confirm whether you really want to remove a file before going ahead and doing it.

rmdir

rmdir allows you to remove or delete directories but not their contents. A directory must be empty in order to remove it using this command.

Syntax: ***rmdir [options] directories***

If you wish to remove a directory and all its contents, you should use rm -r.

su

su stands for superuser (a privileged user), and can be used to log in as another user. If no user is specified and you know the appropriate password, ***su*** can be used to log in as a superuser.

Syntax: ***su [option] [user] [shell_args]***

tail

The ***tail*** command will print the last ten lines of a file. ***tail*** is often used with the option ***-f***, which tells ***tail*** not to quit at the end of file and instead follow the file as it grows.

Syntax: ***tail [options] [file]***

Use ***ctrl-c*** to exit this command.

telnet

You can communicate with other computers by using the ***telnet*** protocol. The host must be a name or an Internet address. ***telnet*** has two modes: the command mode, which is indicated by the ***telnet > prompt***, and an input mode which is usually a session where you would log on to the host system. The default mode is command mode, so if no host is given it will automatically go into this mode. If you need help while in the command mode, type? or ***help***.

Syntax: ***telnet [host [port]]***

who

The ***who*** command prints out information about the most recent status of the system. If no options are listed, then all of the usernames currently logged onto the system are displayed.

Syntax: ***who [options] [file]***

The option ***am i*** will print the name of the current user. The ***-u*** option will display how long the terminal has been idle.

2.7 INTRODUCTION TO SHELL PROGRAMMING

The LINUX operating system was designed by programmers for programmers. The hardware resources demanded a compact, efficient kernel and flexible file-handling system. LINUX provides tool making tool for programmers with shell. Shells are really interpreted languages, any sequence of commands you wish to run can be placed in a file and run regularly LINUX provides, five shells Bourne Again Shell(bash), Z Shell, Tcsh, Korn Shell and Fish (friendly interactive shell). User is requested to go through the shell available in your system. This material is assumed to be on Bourne Shell.

2.7.1 LINUX Shells

In the beginning there was the Bourne shell /bin/sh (written by S. R. Bourne). It had (and still does) a very strong, powerful syntactical language built into it, with all the features that are commonly considered to produce structured programs; it has particularly strong provisions for controlling input and output and in its expression matching facilities. But no matter how strong its input language is, it had one major drawback; it made nearly no concessions to the interactive user (the only real concession being the use of shell functions and these were only added later) and so there was a gap for something better.

Bash

Bash, or the Bourne-Again Shell, is by far the most widely used choice and it comes installed as the default shell in the most popular Linux distributions. It was developed from the original UNIX Bourne shell (also known as sh) and was designed to be fully compatible with the old scripts, while adding multiple improved features.

Bash is a very solid shell option, since it has been used for a long time and there is ample documentation for it. In fact, most tutorials will assume that you're using bash. As a result, it is recommended for most users and works great for most common system administration tasks. However, if you need more powerful scripting options or other advanced tools, it is time to explore some of the newer shells available.

Zsh

Zsh or the Z-shell was designed from the onset to be interactive and incorporate some of the best features of older shells. It provides unique scripting features; it is highly customizable and is easy to use, with spelling correction, command completion or filename globbing.

Korn

KornShell (also known as ksh) is a very old bash alternative that has been developed in the 1980s. It is very similar to bash but doubles as a complete and powerful programming language, so it has a number of passionate fans among sysadmins. It is not widely used, so it's a bit more difficult to find online documentation or help.

Tcsh

Tcsh is a better version of the C shell (csh), which was developed in the UNIX era. It is favored by programmers because its syntax is very similar to the C programming language, so they can use its scripting features without having to learn bash. It is also the default shell in operating systems from the BSD family. It offers several other useful features, such as job control, a command-line editor or a configurable command-line completion tool. Tcsh is installed with yum from the standard repositories.

Fish

Fish, or the ***friendly interactive shell***, aims to be simpler to use and more user friendly than its competitors. It is a great choice for Linux beginners, because it uses colors to help the user. For example, commands with incorrect syntax are displayed in red, while correct ones are blue. In addition, fish provides very useful auto-complete suggestions and even parses the man pages of any new installed package and suggests command completions based on them.

2.7.2 Deciding on a Shell

Which of these shells is the best? That depends on your actual use case. *Bash* is a great all-rounder, with excellent documentation, while *Zsh* adds a few features on top of it to make it even better. *Fish* is amazing for newbies and helps them learn the command line. *Ksh* and *Tcsh* are better suited for advanced users, who need some of their more powerful scripting capabilities.

Let us see how to write the Bourne shell scripts in the following section.

2.8 BOURNE AGAIN SHELL PROGRAMMING

Bash is the GNU Project's shell—the Bourne Again SHell. This is an sh-compatible shell that incorporates useful features from the Korn shell (ksh) and the C shell (csh). It is intended to conform to the IEEE POSIX P1003.2/ISO 9945.2 Shell and Tools standard. It offers functional improvements over sh for both programming and interactive use. In addition, most sh scripts can be run by Bash without modification. The improvements offered by Bash include:

- command-line editing
- unlimited size command history
- job control
- shell functions and aliases
- indexed arrays of unlimited size
- integer arithmetic in any base from two to sixty-four.

A shell script is a plain-text file that contains shell commands. It can be executed by typing its name into a shell, or by placing its name in another shell script.

- To be executable, a shell script file must meet some conditions:
  - The file must have a special first line that names an appropriate command processor. For this tutorial, the following will work in most cases:

    #!/bin/bash

    If this example doesn't work, you will need to find out where your Bash shell executable is located and substitute that location in the above example. Here is one way to find out:

    $ whereis bash

  - The file must be made executable by changing its permission bits. An example:

    $ chmod +x *(shell script filename)*

- A shell script file may optionally have an identifying suffix, like ".sh". This only helps the user remember which files are which. The command processor responsible for executing the file uses the executable bit, plus the file's first line, to decide how to handle a shell script file.

- One normally executes a shell script this way:

  $ ./scriptname.sh

  This special entry is a way to tell the command processor that the desired script is located in the current directory. Always remember: if you cannot get

your shell script to run, remember this trick to provide its location as well as its name.

Hello World Shell Script

- This will get you past the details of writing and launching a simple script.
    1. Choose a text editor you want to use. It can be a command-line editor like emacs, pico or vi, or an X Windows editor if you have this option.
    2. Run your choice of editor and type the following lines:
       #!/bin/bash
       echo "Hello, world"

       NOTE: *Be sure to place a linefeed at the end of your script.* Forgetting a terminating linefeed is a common beginner's error.

    3. Save the file in the current working directory as "myscript.sh".
    4. Move from the text editor to a command shell.
    5. From the command shell, type this:

       $ chmod +x myscript.sh

    6. To execute the script, type this:

       $ ./myscript.sh
       Hello, world

Echo Command

You can use echo command with various options. Some useful options are mentioned in the following example. When you use ***'echo'*** command without any option then a newline is added by default. **'-n'** option is used to print any text without new line and **'-e'** option is used to remove backslash characters from the output.

Create a new bash file with a name, 'ec_example.sh' and add the following script.

```
#!/bin/bash
echo "Printing text with newline"
echo –n "Printing text without newline"
echo –e "\nRemoving \t backslash \t characters\n"
```

To execute the script, type this:
$ bash ec_example.sh

Comments / Multiline Comments

'#' symbol is used to add single line comment in bash script. Create a new file named 'com_example.sh' and add the following script with single line comment.

```
#!/bin/bash
#Add two numeric values
((sum=30+25))

#Print the result
Echo $sum
```

To execute the script, type this:
$ bash com_example.sh

You can use multi line comment in bash in various ways. **':'** and **" ' "** symbols are used to add multiline comment in bash script.

While Loop

Create a bash file with the name, 'wh_example.sh', to know the use of *while* loop. In the example, *while* loop will iterate for 5 times. The value of *count* variable will increment by 1 in each step. When the value of count variable will 5 then the while loop will terminate.

```
#!/bin/bash
valid=true
count=1
while [ $valid ]
do
echo $count
if [ $count -eq 5 ];
then
break
fi
((count++))
done
```

To execute the script, type this:
$ bash wh_example.sh

For Loop

The basic for loop declaration is shown in the following example. Create a file named *'for_example.sh'* and add the following script using *for* loop. Here, *for* loop will iterate for 10 times and print all values of the variable, counter in single line.

```
#!/bin/bash
for (( counter=10; counter>0; counter-- ))
do
echo -n "$counter "
done
printf "\n"
```

To execute the script, type this:
$ bash for_example.sh

User Input

'read' command is used to take input from user in bash. Create a file named 'user_input.sh' and add the following script for taking input from the user. Here, one string value will be taken from the user and display the value by combining other string value.

```
#!/bin/bash
echo "Enter Your Name"
read name
echo "Welcome $name to LinuxHint"
```

To execute the script, type this:
$ bash user_input.sh

If Statement

You can use if condition with single or multiple conditions. Starting and ending block of this statement is define by 'if' and 'fi'. Create a file named **'**simple_if.sh**'** with the following script to know the use *if* statement in bash. Here, 10 is assigned to the variable, n. if the value of $n is less than 10 then the output will be **"**It is a one digit number**"**, otherwise the output will be **"**It is a two digit number**".** For comparison, '-lt' is used here. For comparison, you can also use '-eq' for equality**,** '-ne' for not equality and '-gt' for greater than in bash script.

```
#!/bin/bash
n=10
if [ $n -lt 10 ];
then
echo "It is a one digit number"
else
echo "It is a two digit number"
fi
```

To execute the script, type this:
$ bash simple_if.sh

If statement with AND

Different types of logical conditions can be used in if statement with two or more conditions. How you can define multiple conditions in if statement using *AND* logic is shown in the following example. *'&&'* is used to apply *AND* logic of **if** statement. Create a file named 'if with AND.sh' to check the following code. Here, the value of *username* and *password* variables will be taken from the user and compared with 'admin' and 'secret'. If both values match then the output will be "valid user", otherwise the output will be "invalid user".

```
!/bin/bash

echo "Enter username"
read username
echo "Enter password"
read password

if [[ ( $username == "admin" && $password == "secret" ) ]]; then
echo "valid user"
else
echo "invalid user"
fi
```

To execute the script, type this:
$ bash if_with_AND.sh

If statement with OR

'||' is used to define *OR* logic in *if* condition. Create a file named 'if with OR.sh' with the following code to check the use of *OR* logic of *if* statement. Here, the value of **n** will be taken from the user. If the value is equal to 15 or 45 then the output will be "You won the game", otherwise the output will be "You lost the game".

```
#!/bin/bash

echo "Enter any number"
read n

if [[ ( $n -eq 15 || $n  -eq 45 ) ]]
then
echo "You won the game"
else
echo "You lost the game"
fi
```

To execute the script, type this:
$ bash if_with_OR.sh

To execute the script, type this:
$ bash if_with_OR.sh

Else if Statement

The use of else if condition is little different in bash than other programming language. 'elif' is used to define else if condition in bash. Create a file named, 'elif_example.sh' and add the following script to check how else if is defined in bash script.

```
#!/bin/bash

echo "Enter your lucky number"
read n

if [ $n -eq 101 ];
then
echo "You got 1st prize"
elif [ $n -eq 510 ];
then
echo "You got 2nd prize"
elif [ $n -eq 999 ];
then
echo "You got 3rd prize"
else
echo "Sorry, try for the next time"
fi
```

To execute the script, type this:
$ bash elif_with_OR.sh

Case Statement

Case statement is used as the alternative of if-elseif-else statement. The starting and ending block of this statement is defined by 'case' and 'esac'. Create a new file named, 'cs example.sh' and add the following script. The output of the following script will be same to the previous else if example.

```
#!/bin/bash

echo "Enter your lucky number"
read n
case $n in
```

```
101)
echo echo "You got 1st prize" ;;
510)
echo "You got 2nd prize" ;;
999)
echo "You got 3rd prize" ;;
*)
echo "Sorry, try for the next time" ;;
esac
```

To execute the script, type this:

```
$ bash cs_example.sh
```

Command Line Arguments

Bash script can read input from command line argument like other programming language. For example, $1 and $2 variable are used to read first and second command line arguments. Create a file named “com_line.sh” and add the following script. Two argument values read by the following script and prints the total number of arguments and the argument values as output.

```
#!/bin/bash
echo "Total arguments : $#"
echo "1st Argument = $1"
echo "2nd argument = $2"
```

To execute the script, type this:

```
$ bash com_line.sh PGDCA IGNOU
```

Command Line Arguments with Labels

How you can read command line arguments with names is shown in the following script. Create a file named, ‘cmd_line_names.sh’ and add the following code. Here, two arguments, *X* and *Y* are read by this script and print the sum of X and Y.

```
#!/bin/bash
for arg in "$@"
do
index=$(echo $arg | cut -f1 -d=)
val=$(echo $arg | cut -f2 -d=)
case $index in
X) x=$val;;

Y) y=$val;;

*)
esac
done
((result=x+y))
echo "X+Y=$result"
```

To execute the script, type this:

```
$ bash cmd_line_names X=15 Y=50
```

String Concatenation

You can easily combine string variables in bash. Create a file named "st_combine.sh" and add the following script to check how you can combine string variables in bash by placing variables together or using '+' operator.

```
#!/bin/bash

string1="IGNOU"
string2="PGDCA"
echo "$string1$string2"
string3=$string1+$string2
string3+=" is a good programme"
echo $string3
```

To execute the script, type this:
$ bash st_combine.sh

To Print Substring of a String

Like other programming language, bash has no built-in function to cut value from any string data. But you can do the task of substring in another way in bash that is shown in the following script. To test the script, create a file named 'subst_example.sh' with the following code. Here, the value, **6** indicates the starting point from where the substring will start and **5** indicates the length of the substring.

```
#!/bin/bash
Str="Learn Linux from IGNOU"
subStr=${Str:6:5}
echo $subStr
```

To execute the script, type this:
$ bash subst_example.sh

Email

You can send email by using 'mail' or 'sendmail' command. Before using these commands, you have to install all necessary packages. Create a file named, 'email_example.sh' and add the following code to send the email.

```
#!/bin/bash
Recipient="admin@example.com"
Subject="Greeting"
Message="Welcome to IGNOUs PGDCA Programme"
'mail -s $Subject $Recipient <<< $Message'
```

To execute the script, type this:
$ bash email_example.sh

2.9 GENERAL GUIDELINES

Following are some of the general guidelines:

- Observation book and Lab record are compulsory.
- You should attempt all problems/assignments given in the list session wise.

- For the tasks related to the working with the LINUX utilities / shell programs, describe the procedure and also present screenshots wherever applicable.

- You may seek assistance in doing the lab exercises from the concerned lab instructor. Since the assignments have credits, the lab instructor is obviously not expected to tell you how to solve these, but you may ask questions concerning the Operating system and C programs.

- For each C program you should add comments (i.e. text between /* ... */ delimiters) above each function in the code, including the main function. This should also include a description of the function written, the purpose of the function, meaning of the argument used in the function and the meaning of the return value (if any). These descriptions should be placed in the comment block immediately above the relevant function source code.

- The comment block above the main function should describe the purpose of the program. Proper comments are to be provide where and when necessary in the programming.

- The program written for the problem given should conform to the ANSI standard for the C language.

- The program should be interactive, general and properly documented with real Input/ Output data.

- If two or more submissions from different students appear to be of the same origin (i.e. are variants of essentially the same program), none of them will be counted. You are strongly advised not to copy somebody else's work.

- It is your responsibility to create a separate directory to store all the programs, so that nobody else can read or copy.

- The list of the programs(list of programs given at the end, session-wise) is available to you in this lab manual. For each session, you must come prepare with the algorithms and the programs written in the Observation Book. You should utilize the lab hours for executing the programs, testing for various desired outputs and enhancements of the programs.

- As soon as you have finished a lab exercise, contact one of the lab instructor / incharge in order to get the exercise evaluated and also get the signature from him/her on the Observation book.

- Completed lab assignments should be submitted in the form of a Lab Record in which you have to write the algorithm, program code along with comments and output for various inputs given.

- The total no. of lab sessions (3 hours each) are 10 and the list of assignments is provided session-wise. It is important to observe the deadline given for each assignment.

Get started with Windows 10 and explore the utilities sessionwise.

2.10 PRACTICAL SESSIONS

Sessionwise practical problems are given as following:

Session 1

1) Explore all the LINUX commands given in this manual.

2) Create a directory with your *surname*.
3) Create a subdirectory in the directory created with your *name*.
4) Change your current directory to the subdirectory.
5) Display the calendar for the current month.
6) Get a directory listing of the parent directory.
7) How many users were logged onto your system?
8) Display your name in the form of a banner.
9) Display the name of device name of your terminal.
10) Move to the root directory.

Session 2

11) Change your directory to the directory ***exercises***. Create a file called ***example1*** using the cat command containing the following text:

water, water everywhere
and all the boards did shrink;
water, water everywhere,
No drop to drink.

12) Use the man command to obtain further information on the finger command.
13) List all the processes that are presently running.
14) List the text files in your current directory.
15) Make a copy of any text file.
16) Rename one of your text files in the current directory.
17) Delete an unneeded copy of a file.
18) Print out any file on paper.
19) Send a message to another user on your LINUX system, and get them to reply.
20) Create a small text file and send it to another user.

Session 3

21) When you receive a message, save it to a file other than your mailbox.
22) Send a message to a user on a different computer system.
23) Try to move to the home directory of someone else in your group. There are several ways to do this, and you may find that you are not permitted to enter certain directories. See what files they have, and what the file permissions are.
24) Try to copy a file from another user's directory to your own.
25) Set permissions on all of your files and directories to those that you want. You may want to give read permission on some of your files and directories to members of your group.
26) Create a number of hierarchically related directories and navigate through them using a combination of absolute pathnames (starting with "/") and relative pathnames.
27) Try using wildcards ("*" and possibly "?").
28) Put a listing of the files in your directory into a file called *filelist*. (Then delete it!)
29) Create a text file containing a short story, and then use the ***spell*** program to check the spelling of the words in the file.
30) Redirect the output of the ***spell*** program to a file called ***errors***.

Session 4

31) Type the command ***ls -l*** and examine the format of the output. Pipe the output of the command ***ls -l*** to the word count program wc to obtain a count of the number of files in your directory.

32) Use cut to strip away the reference material and leave just the text field.

33) Use tr to strip away any tags that are actually in the text (e.g., attached to the words), so that you are left with just the words.

34) Set a file to be read-only with the **chmod** (from *ch*ange *mod*e) command. Interpret the file permissions displayed by the **ls -l** command.

35) Delete one or more directories with the **rmdir** (from *rem*ove *dir*ectory) command. See what happens if the directory is not empty. Experiment (carefully!) with the **rm -r** command to delete a directory and its content.

36) Experiment with redirecting command output (e.g., **ls -l >file1**). Try ">> " instead of " >" with an existing text file as the output.

37) See whether upper-case versions of any of these commands work as well as the lower-case versions.

38) Use the who command to see users logged into the system.

39) Pipe the output of the ***who*** command to the **sort** command

40) Search for your login name in **whofile** using the **grep** command.

Session 5

41) Using ***sed*** (Stream Editor) perform the following:
    a) Write a sed command that print lines numbers of lines beginning with "A or a".
    b) Write a sed command that delete digits in the given input file.
    c) Write a sed command that delete lines that contain both START and END.
    d) Write a sed command that delete lines that contain START but not END**.**
    e) Write a sed command that deletes the first character in each line in a file.
    f) Write a sed command that deletes the last character in each line in a file.

42) Using ***awk utility*** perform the following:

a) Write an awk command to print the lines and line number in the given input file.
b) Write an awk command to print first field and second field only if third field value is >=25 in the given input file.
   (Hint: input field separator is ":" and output field separator is ",").
c) Consider the marks.txt is a file that contains one record per line(comma separate fields) of the student data in the form of SRollNo, SName, Hindi marks, English marks, Maths Marks, Science marks, Social Marks. Write an awk script to generate result for every students in the form of SRollNo, SName, Total Marks, Average and Grade.

Grade is PASS if marks is >=30 in Hindi and English respectively, and if marks >= 40 in other subjects. Result is fail otherwise.

d) Write an awk program to print the fields 1 and 4 of a file that is passed as command line argument. The file contains lines of information that is separated by "," as delimiter. The awk program must print at the end the total and average of all 3rd field data.
e) Write an awk program to demonstrate user defined functions and system command.
f) Write an awk script to count the number of lines in a file that do not contain vowels.
g) Write an awk script to find the number of characters, words and lines in a file.

Session 6

43) Use the ***grep*** command to search the file ***example1*** for occurrences of the string "water".

44) Write grep commands to do the following activities:
a) Write a grep command that selects the lines from the file1 that have exactly three characters
b) Write a grep command that selects the lines from the file1 that have at least three characters.
c) Write a grep command that selects the lines from the file1 that have three or fewer characters
d) Write a grep command that count the number blank lines in the file1
e) Write a grep command that count the number nonblank lines in the file1
f) Write a grep command that selects the lines from the file1 that have the string LINUX.
g) Write a grep command that selects the lines from the file1 that have only the string LINUN.
h) Write a grep command that copy the file to the monitor, but delete the blank lines.
i) Write a grep command that selects the lines from the file1 that have at least two digits without any other characters in between
j) Write a grep command that selects the lines from the file1 that do not start with A to G

45) Make a sorted wordlist from the file.

Session 7

46) Write shell script to perform integer arithmetic operations.
47) Write a shell script to perform floating point arithmetic operations.
48) Write a shell script to check the given file is writable or not.
49) Write a shell program to find out reverse string of the given string and check the given string is palindrome or not.
50) Write a shell program to find out factorial of the given number.
51) Write a shell script to find out whether the given number is prime number or not.
52) Write a shell script to accept two file names and check if both exists. If the second filename exists, then the contents of the first filename should be appended to it. If the second file name does not exist then create a new file with the contents of the first file.

53) Write a shell script that computes the gross salary of a employee according to the following:
- if basic salary is >=15000 then HRA 5% and DA =10% of the basic
- if basic salary is <15000 then HRA 10% of the basic and DA =20% of the basic

Note: The basic salary is entered interactively through the key board.

54) Write a shell script that accepts a file name, starting and ending line numbers as arguments and displays all the lines between the given line numbers.

55) Write a shell script that deletes all lines containing a specified word in one or more files supplied as arguments to it.

56) Write a shell script to encrypt any text file.

Session 8

57) Write a shell script to translate all the characters to lower case in a given text file.

58) Write a shell script to combine any three text files into a single file (append them in the order as they appear in the arguments) and display the word count.

59) Write a shell script that, given a file name as the argument will write the even numbered line to a file with name ***evenfile*** and odd numbered lines to a file called ***oddfile***.

60) Write a shell script which deletes all the even numbered lines in a text file.

61) Write a script that will count the number of files in each of your subdirectories.

62) Write a shell script like a more command. It asks the user name, the name of the file on command prompt and displays only the 15 lines of the file at a time on the screen. Further, next 15 lines will be displayed only when the user presses the enter key / any other key.

63) Write a shell script that counts English language articles (a, an, the) in a given text file.

64) Write the shell script which will replace each occurrence of character ***c*** with the characters ***chr*** in a string ***s***. It should also display the number of replacements.

Session 9

65) Write a shell script which reads the contents in a text file and removes all the blank spaces in them and redirects the output to a file.

66) Write a shell script that accepts a list of file names as its arguments, counts and reports the occurrence of each word that is present in the first argument file on other argument files.

67) Write a shell script to list all of the directory files in a directory.

68) Write a shell script to implement menu driven program to display list of users who are currently working in the system, copying files (cp command), rename a file, list of files in the directory and quit option.(Hint: use case structure)

69) Write a shell program to simulate ‘cat’ command

70) Write a C program to implement the Producer/Consumer problem using semaphores using UNIX/LINUX system calls.

71) Write C programs to simulate the following memory management techniques: (a) Paging (b) Segmentation

Session 10

72) Write C programs to simulate the following CPU Scheduling algorithms a) FCFS b) SJF c) Round Robin d) Priority
73) Write programs using the I/O system calls of UNIX/LINUX operating system (open, read, write, close, fcntl, seek, stat, opendir, readdir)
74) Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention.

2.11 SUMMARY

LINUX is a popular operating System, which is mostly using the C language, making it easy to port to different configurations. LINUX programming environment is unusually rich and productive. It provides features that allow complex programs to be built from simpler programs.

LINUX uses a hierarchical file system that allows easy maintenance and efficient implementation. It uses a consistent format for files, the byte stream, making application programs easier to write. It is a multi-user, multitasking system. Each user can execute several processes simultaneously. It hides the machine architecture from the user, making it easier to write programs that run on different hardware implementation. It is highly secured system.

2.12 FURTHER READINGS

1) Behrouz A. Forouzan, Richard F. Gilberg, *UNIX and Shell Programming*, Thomson
2) Brian W. Kernighan, Rob Pike, *The UNIX Programming Environment*, PHI.
3) K. Srirengan, *Understanding UNIX*, PHI, 2002
4) Sumitabha Das, *Your UNIX- The Ultimate Guide*, TMGH.
5) Sumitabha Das, *UNIX Concepts and Applications, Second Edition*, TMGH.
6) Linux System Programming, Robert Love, O'Reilly, 2014.
7) System Programming with C and Unix, A. Hoover, Pearson
8) Shell Scripting, S. Parker, Wiley India Pvt. Ltd.
9) Advanced Programming in the Unix Environment, 2nd edition, W. R. Stevens and S. A. Rago, Pearson Education.

2.13 WEBSITE REFERENCES

1) www.linux.org/
2) www.kernel.org/
3) www.linux.com/
4) www.unix.org/
5) unixhelp.ed.ac.uk
6) www.unix.org/resources.html
7) www.osnews.com/
8) www.levenez.com/unix/
9) cnc.k12.mi.us/websites/bsdtree.html/

2.14 Virtual Labs / IDEs / Online Lab Resources

LINUX

1) http://copy.sh/v86/?profile=linux26
2) https://www.webminal.org/
3) https://www.tutorialspoint.com/codingground.htm
4) https://www.tutorialspoint.com/execute_bash_online.php
5) https://www.masswerk.at/jsuix/index.html
6) http://cb.vu/

C Compilers / IDEs

1) https://www.onlinegdb.com/online_c_compiler
2) https://www.programiz.com/c-programming/online-compiler/
3) https://www.codechef.com/ide
4) http://codeblocks.org/
5) https://www.jdoodle.com/c-online-compiler/
6) http://codelite.org/
7) https://www.onworks.net/programs/ubuntu-emulator-online

Spoken Tutorials

1) https://spoken-tutorial.org/tutorial-search/?search_foss=Linux&search_language=English
2) https://spoken-tutorial.org/tutorial-search/?search_foss=Linux+AWK&search_language=English
3) https://spoken-tutorial.org/tutorial-search/?search_foss=BASH&search_language=English
4) https://spoken-tutorial.org/tutorial-search/?search_foss=Linux+for+Sys-Ads&search_language=English
5) https://spoken-tutorial.org/tutorial-search/?search_foss=Advance+C&search_language=English

NOTES

NOTES

NOTES

MPDD/IGNOU/P.O. 2K/June, 2021

ISBN : 978-93-91229-73-3