MU | SEM 3 | S. Y. B.Sc. Information Technology

Core Subject (USIT304)

OPERATING SYSTEMS

New Syllabus 2023-24

with Lab Manual

Er. Manoj S. Kavedia

Mrs. Bhakti P. Patil

Mrs. Tamanna Agrawal

Ms. Cynthia N. Shinde

Important Feature of the Book

- Chapterwise most Important Questions as per New Syllabus.

TECH-NEO PUBLICATIONS
Where Authors Inspire Innovation
A Sachin Shah Venture
www.techneobooks.in
info@techneobooks.in

LAB Manual

Experiment No 1A.

Title : Installation and Configuration of virtual machine

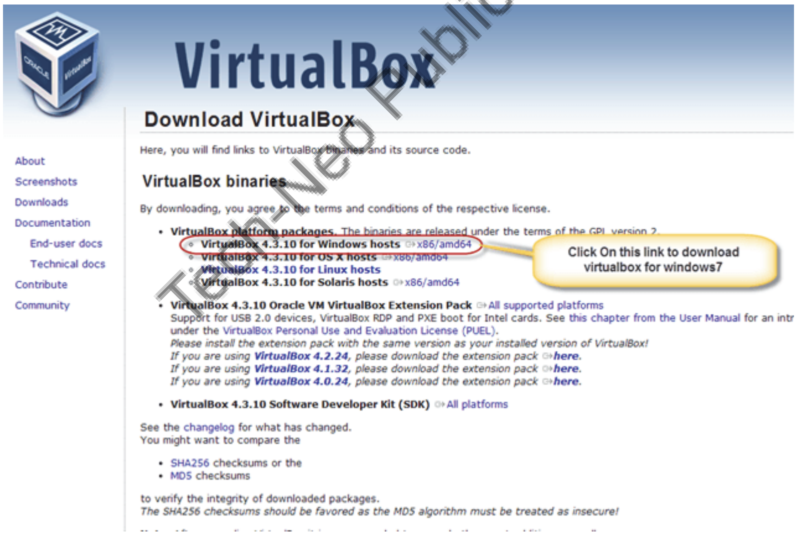
Aim: To Install Virtual Machine Software

☞ Procedure

Download Virtual box using this

https://www.virtualbox.org/wiki/Downloads

Depending on your processor and OS, select the appropriate package. In our case, we have selected Windows with AMD



Download VirtualBox

Here, you will find links to VirtualBox binaries and its source code.

VirtualBox binaries

By downloading, you agree to the terms and conditions of the respective license.

- VirtualBox platform packages. The binaries are released under the terms of the GPL version 2.
  - VirtualBox 4.3.10 for Windows hosts ⇒x86/amd64
  - VirtualBox 4.3.10 for OS X hosts ⇒x86/amd64
  - VirtualBox 4.3.10 for Linux hosts
  - VirtualBox 4.3.10 for Solaris hosts ⇒x86/amd64
- VirtualBox 4.3.10 Oracle VM VirtualBox Extension Pack ⇒All supported platforms
  Support for USB 2.0 devices, VirtualBox RDP and PXE boot for Intel cards. See this chapter from the User Manual for an intr
  under the VirtualBox Personal Use and Evaluation License (PUEL).
  Please install the extension pack with the same version as your installed version of VirtualBox!
  *If you are using **VirtualBox 4.2.24**, please download the extension pack ⇒**here**.*
  *If you are using **VirtualBox 4.1.32**, please download the extension pack ⇒**here**.*
  *If you are using **VirtualBox 4.0.24**, please download the extension pack ⇒**here**.*
- VirtualBox 4.3.10 Software Developer Kit (SDK) ⇒All platforms

See the changelog for what has changed.
You might want to compare the

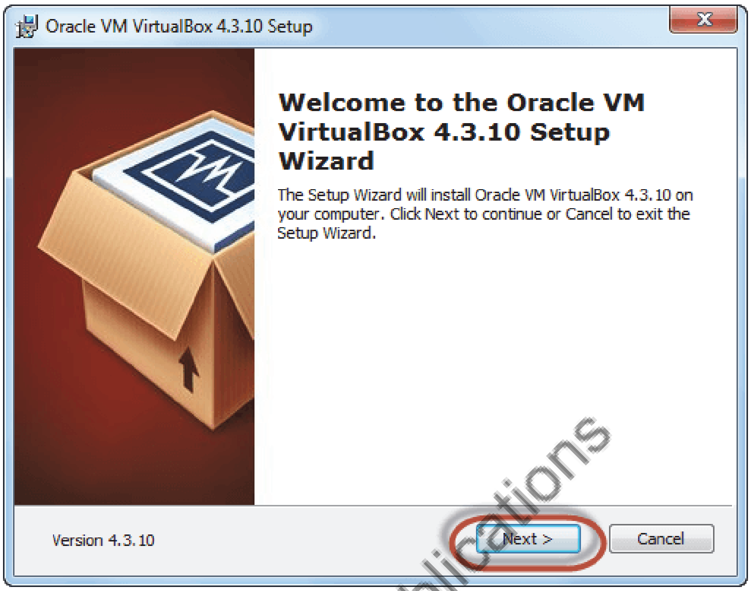
- SHA256 checksums or the
- MD5 checksums

to verify the integrity of downloaded packages.
The SHA256 checksums should be favored as the MD5 algorithm must be treated as insecure!

Click On this link to download virtualbox for windows7

Once the download is complete, Open setup file and follow the steps below:

Step 1 : Click On next



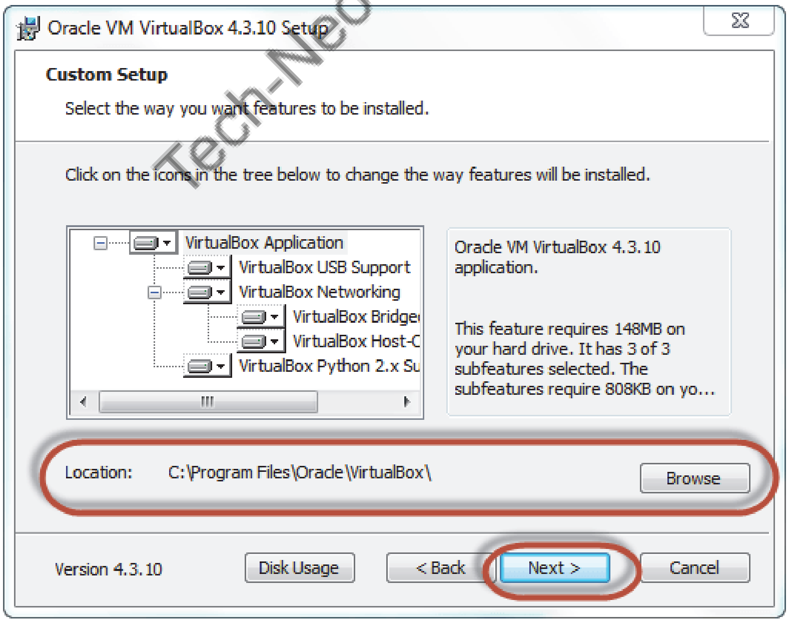
Oracle VM VirtualBox 4.3.10 Setup

Welcome to the Oracle VM VirtualBox 4.3.10 Setup Wizard

The Setup Wizard will install Oracle VM VirtualBox 4.3.10 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Version 4.3.10

Step 2 : Select you're the directory to install VirtualBox and click on next



Oracle VM VirtualBox 4.3.10 Setup

Custom Setup

Select the way you want features to be installed.

Click on the icons in the tree below to change the way features will be installed.

```
VirtualBox Application
    VirtualBox USB Support
    VirtualBox Networking
        VirtualBox Bridge
        VirtualBox Host-C
    VirtualBox Python 2.x Su
```

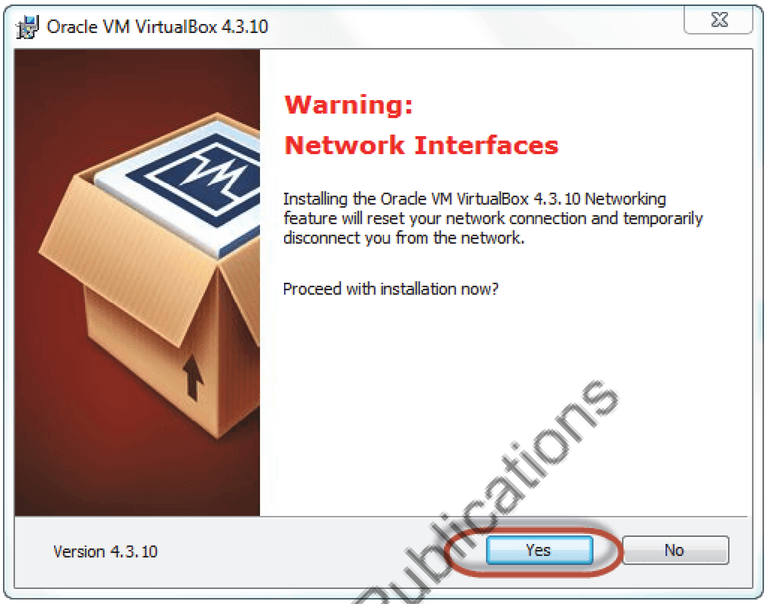
Oracle VM VirtualBox 4.3.10 application.

This feature requires 148MB on your hard drive. It has 3 of 3 subfeatures selected. The subfeatures require 808KB on yo...

Location: C:\Program Files\Oracle\VirtualBox\

Version 4.3.10

➧ Step 3 : Select Desktop icon and click on next, now click on yes



Oracle VM VirtualBox 4.3.10

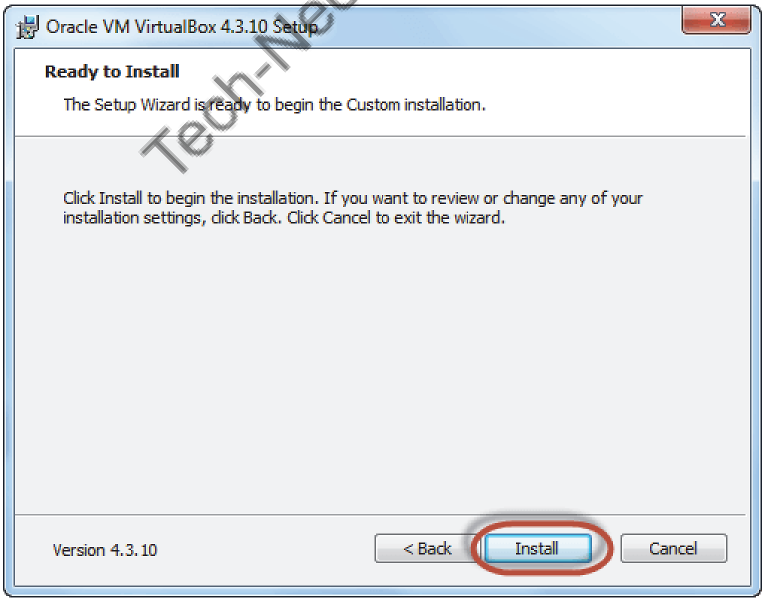
Warning:
Network Interfaces

Installing the Oracle VM VirtualBox 4.3.10 Networking feature will reset your network connection and temporarily disconnect you from the network.

Proceed with installation now?

Version 4.3.10

➧ Step 4 : Click On install to install Linux on Windows.



Oracle VM VirtualBox 4.3.10 Setup

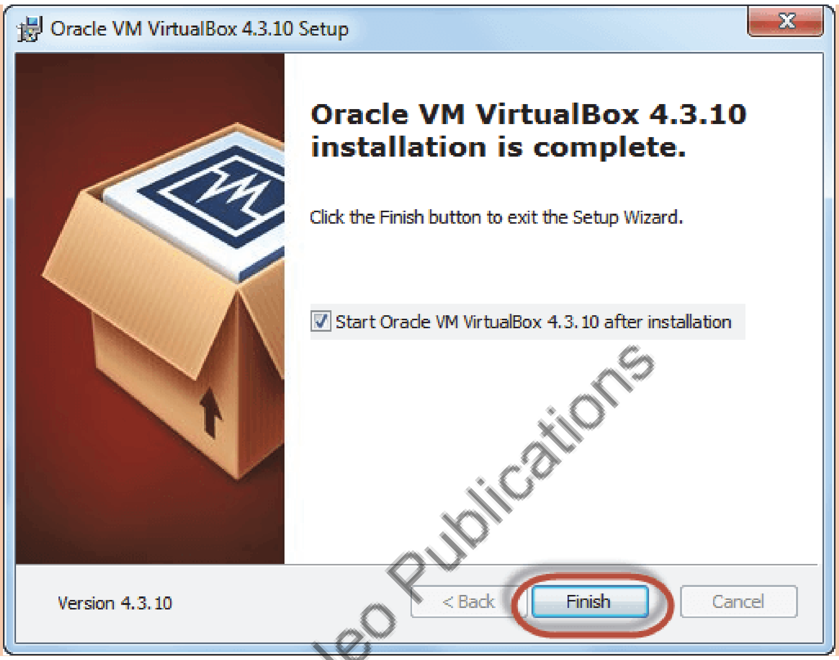
Ready to Install

The Setup Wizard is ready to begin the Custom installation.

Click Install to begin the installation. If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

Version 4.3.10

- **Step 5 : Now installation of the virtual box will start. Once complete, click on Finish Button to start Virtual Box**



Oracle VM VirtualBox 4.3.10 Setup

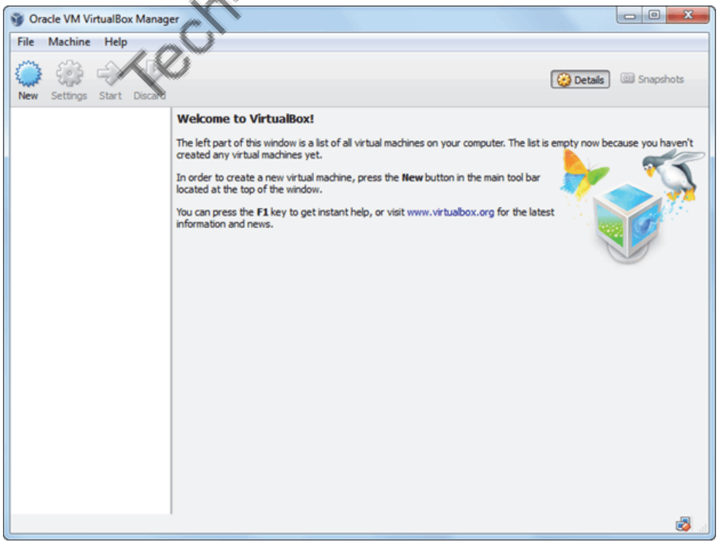
Oracle VM VirtualBox 4.3.10 installation is complete.

Click the Finish button to exit the Setup Wizard.

☑ Start Oracle VM VirtualBox 4.3.10 after installation

Version 4.3.10

The virtual box dashboard looks like this-



Welcome to VirtualBox!

The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for the latest information and news.

Experiment No 1B

Title : Installation and Configuration of virtual machine

Aim: To Install Windows Operating System

Procedure

There are several ways to install Windows 10 on your computer system. Any one can be out of them can be used. For example:

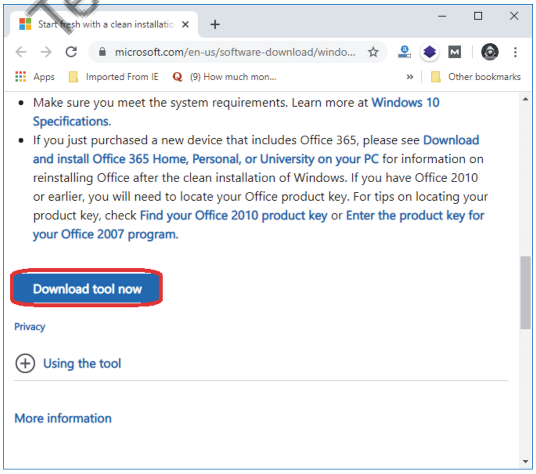
- Upgrade your existing Windows 7 or Windows 8 operating system to Windows 10.
- Install a fresh new Windows 10 from scratch.
- Reinstall a fresh version of Windows 10 if you already have Windows 10 installed in your computer system.

Fresh New windows Installation

This option is used when you don't want to upgrade from existing Windows and install a clean Windows 10.

Following are the steps given below:

- To perform a clean installation of Windows 10, download the free official Windows 10 installation media from Microsoft. Download it from here: https://www.microsoft.com/en-us/software-download/windows10startfresh


Start fresh with a clean installatio
microsoft.com/en-us/software-download/windo...
Apps
Imported From IE
(9) How much mon...
Other bookmarks

- Make sure you meet the system requirements. Learn more at Windows 10 Specifications.
- If you just purchased a new device that includes Office 365, please see Download and install Office 365 Home, Personal, or University on your PC for information on reinstalling Office after the clean installation of Windows. If you have Office 2010 or earlier, you will need to locate your Office product key. For tips on locating your product key, check Find your Office 2010 product key or Enter the product key for your Office 2007 program.

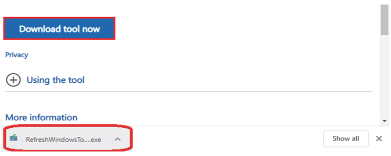
Download tool now

Privacy

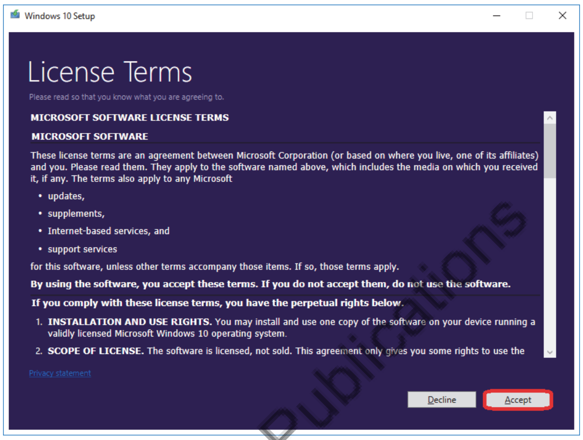
⊕ Using the tool

More information

- Click on the **"Download Tool Now"** button.



- Run the downloaded file.



License Terms

Please read so that you know what you are agreeing to.

MICROSOFT SOFTWARE LICENSE TERMS

MICROSOFT SOFTWARE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to the software named above, which includes the media on which you received it, if any. The terms also apply to any Microsoft

- updates,
- supplements,
- Internet-based services, and
- support services

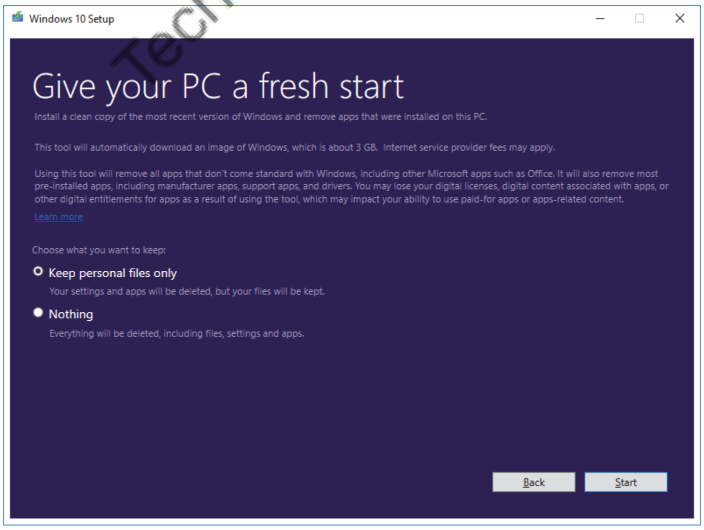
for this software, unless other terms accompany those items. If so, those terms apply.

By using the software, you accept these terms. If you do not accept them, do not use the software.

If you comply with these license terms, you have the perpetual rights below.

1. **INSTALLATION AND USE RIGHTS.** You may install and use one copy of the software on your device running a validly licensed Microsoft Windows 10 operating system.
2. **SCOPE OF LICENSE.** The software is licensed, not sold. This agreement only gives you some rights to use the

- Accept the license term. Now, you will be asked to **"Give your PC a fresh start."** Choose any one of the options given below. Select what you want to keep.



Give your PC a fresh start

Install a clean copy of the most recent version of Windows and remove apps that were installed on this PC.

This tool will automatically download an image of Windows, which is about 3 GB. Internet service provider fees may apply.

Using this tool will remove all apps that don't come standard with Windows, including other Microsoft apps such as Office. It will also remove most pre-installed apps, including manufacturer apps, support apps, and drivers. You may lose your digital licenses, digital content associated with apps, or other digital entitlements for apps as a result of using the tool, which may impact your ability to use paid-for apps or apps-related content.

Learn more

Choose what you want to keep:

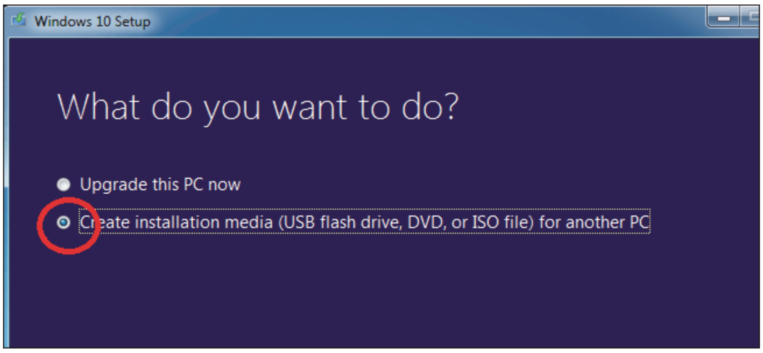
● Keep personal files only

Your settings and apps will be deleted, but your files will be kept.

○ Nothing

Everything will be deleted, including files, settings and apps.

- Now, select "Create installation media for another PC" option. As shown in fig. below

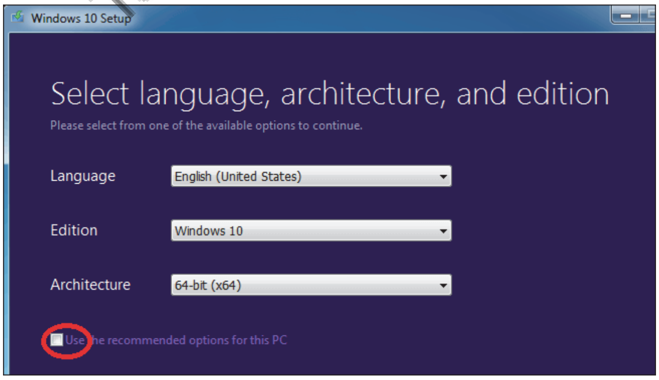


What do you want to do?

○ Upgrade this PC now

● Create installation media (USB flash drive, DVD, or ISO file) for another PC

- Select the language, architecture, and edition according to need. For example,
  - if you have a **laptop or PC with a 64-bit CPU**, then you have to install the 64-bit version.
  - If you have a **computer or PC with a 32-bit CPU**, then you have to install the 32-bit version.
  - If you want to install Windows 10 on your current using laptop or PC, check the box **"Use the recommended options for this PC,"** and it will automatically download the best compatible version for your current using PC;
  - otherwise, **uncheck** this box. As shown in fig. below:



Select language, architecture, and edition

Please select from one of the available options to continue.

Language: English (United States)

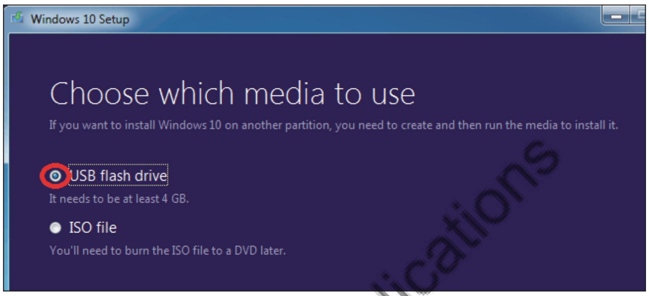
Edition: Windows 10

Architecture: 64-bit (x64)

□ Use the recommended options for this PC

- Now, you can copy the Windows 10 installation files to a USB drive or burn them to a DVD.

- The Windows 10 installation files occupy the space greater than 3 GB so, your USB drive must be 4 GB or larger in size. The USB drive must be empty because all files in the USB drive will be erased in this process.
- You can select the **"ISO file"** option if you want to install Windows 10 in a virtual machine.
- First, download the ISO file and then boot the downloaded ISO in a virtual machine to install Windows 10 inside that.



Choose which media to use

If you want to install Windows 10 on another partition, you need to create and then run the media to install it.

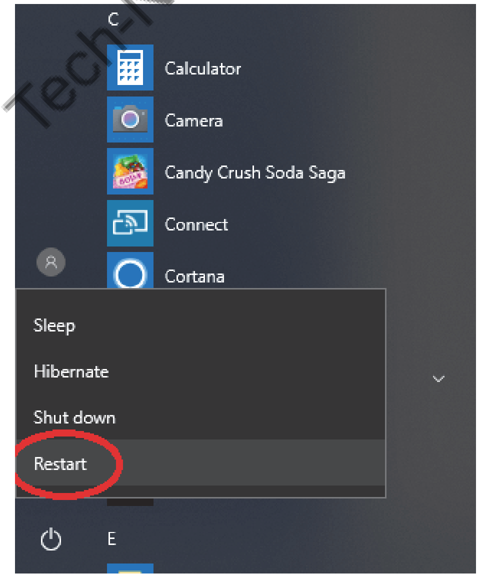
● USB flash drive

It needs to be at least 4 GB.

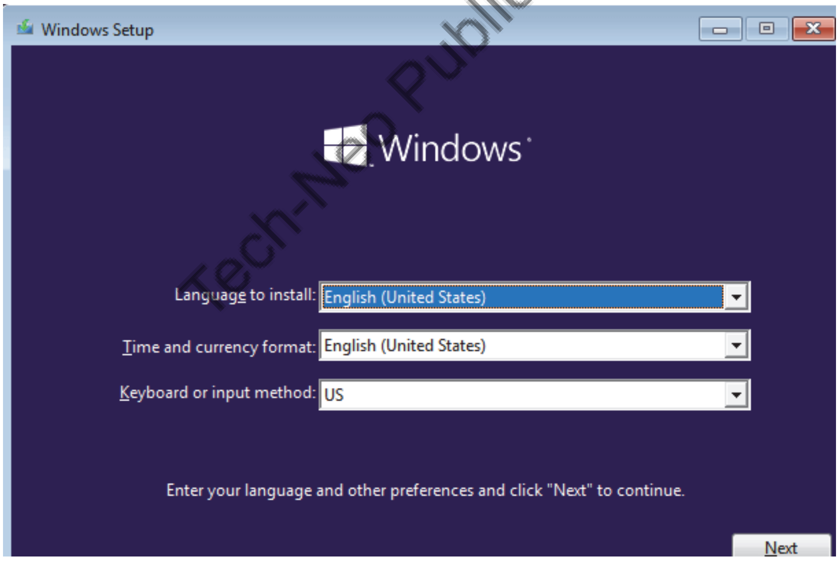
○ ISO file

You'll need to burn the ISO file to a DVD later.

- Now, the installation media is created. Please insert it into your laptop or PC and install Windows 10. You have to boot to the Windows 10 installer. For the booting process, Click on the Start menu button and restart your laptop.



- Now, press and hold Del or F2 button to enter setup. This key may be different in some computers. Generally, it is displayed as a message on startup that says, **"Press [key] to enter setup."**
- Now, go to the **"Boot"** option on the above menu bar and select a device from which you have to boot. You will see the following two option there:
  1. For a USB flash drive, select the **Removable Devices** option.
  2. For a disc installation, select the **CD-ROM Drive** option.
- Now, save your setting and press the Enter button to confirm the changes. Now, wait for your PC to restart.
- Now, the installation is started. Select your language, time, and currency format on the Windows setup screen and then click **"Next"** to continue.



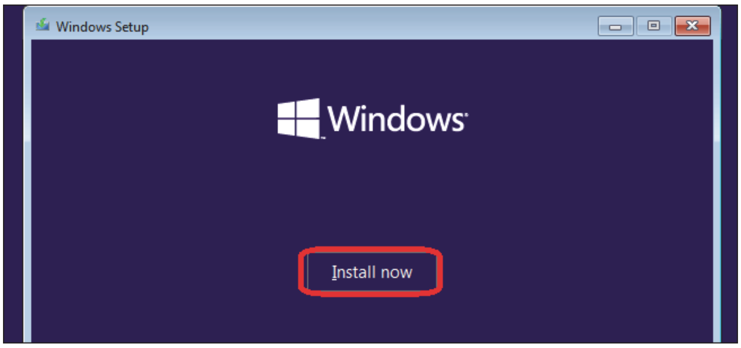
Language to install: English (United States)

Time and currency format: English (United States)

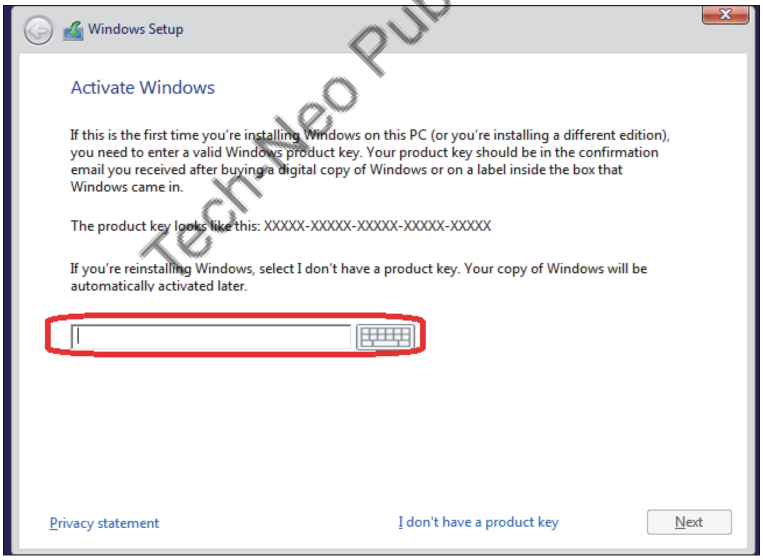
Keyboard or input method: US

Enter your language and other preferences and click "Next" to continue.

- You may have to click on several **"next"** buttons, and finally, you will see the installer screen. As shown in fig. below:



- Click on the install now button and follow the given instructions to install Windows 10 on your system successfully.
- Windows 10 is installed successfully. Now, you will see the **Activate Windows** screen.
- You can enter a key or skip it. If installing Windows 10 automatically detects a key associated with your PC's hardware, you will not see it on your screen.



Activate Windows

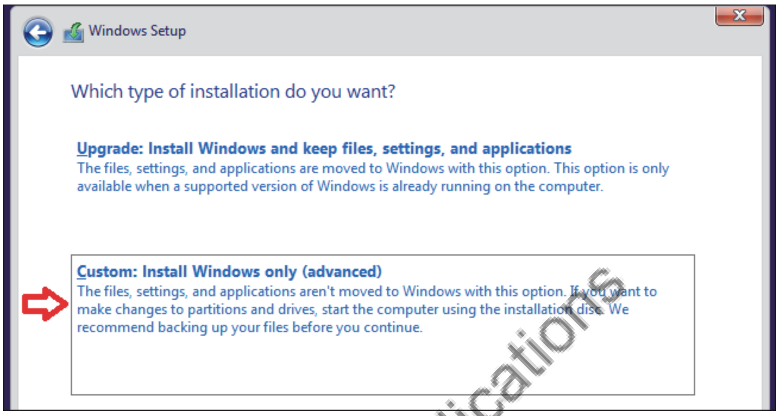
If this is the first time you're installing Windows on this PC (or you're installing a different edition), you need to enter a valid Windows product key. Your product key should be in the confirmation email you received after buying a digital copy of Windows or on a label inside the box that Windows came in.

The product key looks like this: XXXXX-XXXXX-XXXXX-XXXXX-XXXXX

If you're reinstalling Windows, select I don't have a product key. Your copy of Windows will be automatically activated later.

- If you have a valid product key for Windows 10, you can paste it here. You can **also paste a valid Windows 7, 8, or 8.1** key here. You will get the advantage of the free Windows 10 upgrade offer on your PC.

- After that, you will see a new Windows setup screen **"Which type of installation do you want?"** Click "Custom" if you want to perform a clean installation. It will remove everything on your PC. If you want to upgrade your existing installation, click the "Upgrade" option.



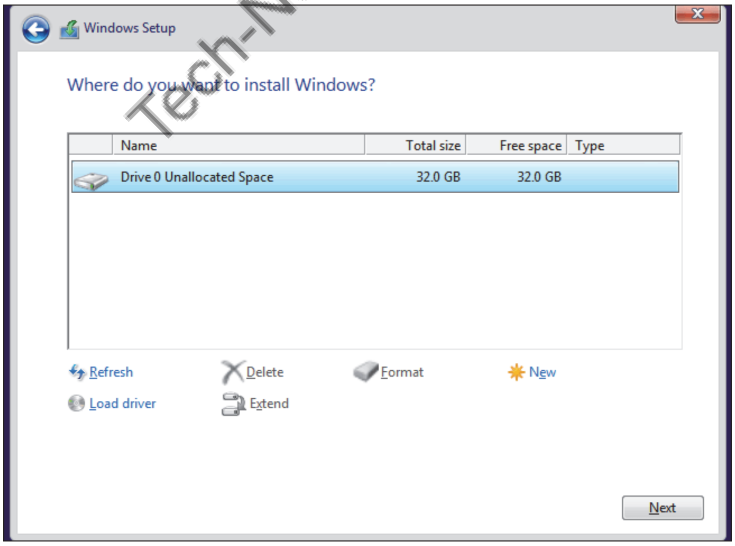
Windows Setup

Which type of installation do you want?

Upgrade: Install Windows and keep files, settings, and applications
The files, settings, and applications are moved to Windows with this option. This option is only available when a supported version of Windows is already running on the computer.

Custom: Install Windows only (advanced)
The files, settings, and applications aren't moved to Windows with this option. If you want to make changes to partitions and drives, start the computer using the installation disc. We recommend backing up your files before you continue.

- Select the hard drive partition where you want to install Windows. It will delete the data you have in that partition. Make sure that you have backups of any important files before doing this.

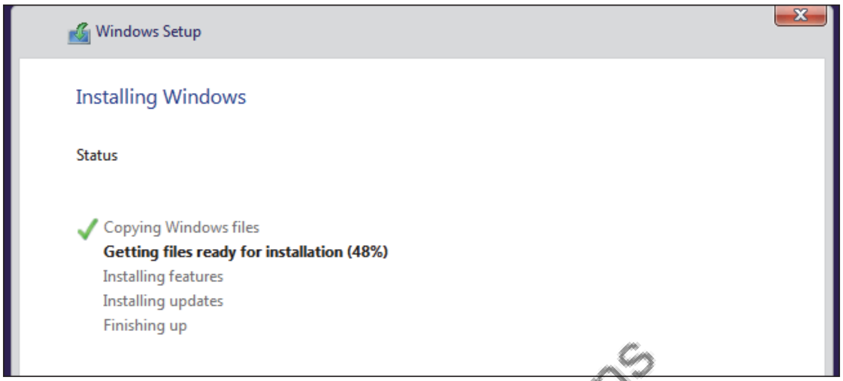


Windows Setup

Where do you want to install Windows?

Name	Total size	Free space	Type
Drive 0 Unallocated Space	32.0 GB	32.0 GB	

- In this installation process, your system may restart several times. At the end of the process, you will get a new Windows 10 as your operating system.



Installing Windows

Status

✓ Copying Windows files

Getting files ready for installation (48%)

Installing features

Installing updates

Finishing up

- Restart your PC after complete installation. It is now ready to work.

Experiment No 1C.

Title : Installation and Configuration of virtual machine

Aim: To Install Linux Operation System

☞ Procedure

Installation of Ubuntu can be done in different ways those are

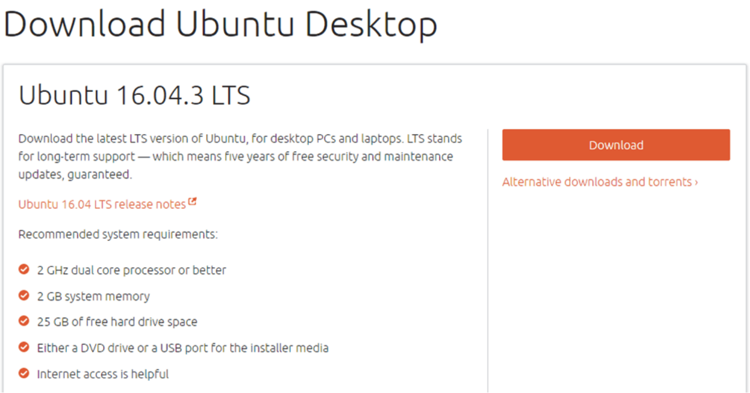
1. Installing Linux using USB stick
2. Installing Linux using CD-ROM
3. Create a Machine in Virtual Box

Part-1 : Installing Linux using USB stick

This is one of the easiest methods of installing Ubuntu or any distribution on your computer. Follow the steps to install Ubuntu from USB.

Step 1: Download required files.

Download the .iso or the OS files on your computer from this (https://ubuntu.com/download/desktop).


Download Ubuntu Desktop

Ubuntu 16.04.3 LTS

Download the latest LTS version of Ubuntu, for desktop PCs and laptops. LTS stands for long-term support — which means five years of free security and maintenance updates, guaranteed.

Ubuntu 16.04 LTS release notes

Recommended system requirements:

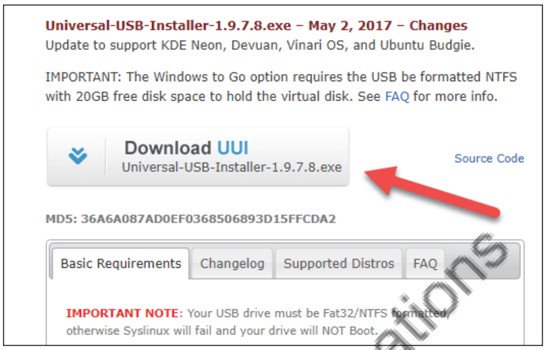
- 2 GHz dual core processor or better
- 2 GB system memory
- 25 GB of free hard drive space
- Either a DVD drive or a USB port for the installer media
- Internet access is helpful

Step 2: Download Universal USB Installer.

Download free software like Universal USB installer to make a bootable USB stick.

Link

https://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3/



Universal-USB-Installer-1.9.7.8.exe – May 2, 2017 – Changes

Update to support KDE Neon, Devuan, Vinari OS, and Ubuntu Budgie.

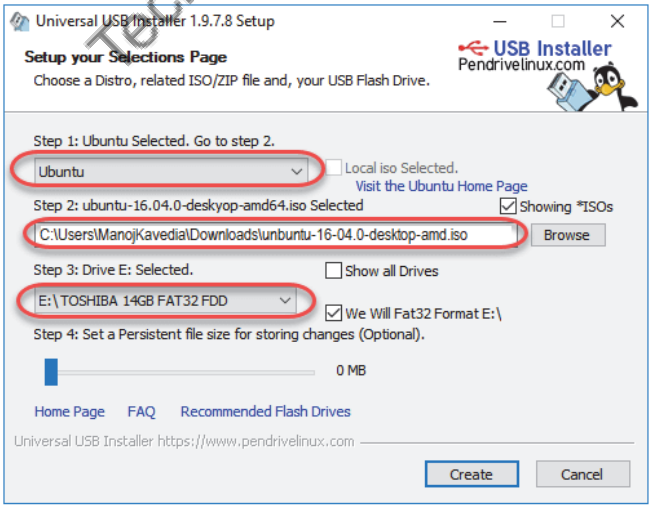
IMPORTANT: The Windows to Go option requires the USB be formatted NTFS with 20GB free disk space to hold the virtual disk. See FAQ for more info.

MD5: 36A6A087AD0EF0368506893D15FFCDA2

IMPORTANT NOTE: Your USB drive must be Fat32/NTFS formatted, otherwise Syslinux will fail and your drive will NOT Boot.

Step 3: Select Distribution.

- Select an Ubuntu Distribution form the dropdown to put on your USB
- Select your Ubuntu iso file download in step 1.
- Select the drive letter of USB to install Ubuntu and Press create button.



Universal USB Installer 1.9.7.8 Setup

Setup your Selections Page

Choose a Distro, related ISO/ZIP file and, your USB Flash Drive.

Step 1: Ubuntu Selected. Go to step 2.

Ubuntu

□ Local iso Selected.

Visit the Ubuntu Home Page

Step 2: ubuntu-16.04.0-deskyop-amd64.iso Selected

☑ Showing *ISOs

C:\Users\ManojKavedia\Downloads\unbuntu-16-04.0-desktop-amd.iso

Step 3: Drive E: Selected.

□ Show all Drives

E:\ TOSHIBA 14GB FAT32 FDD

☑ We Will Fat32 Format E:\

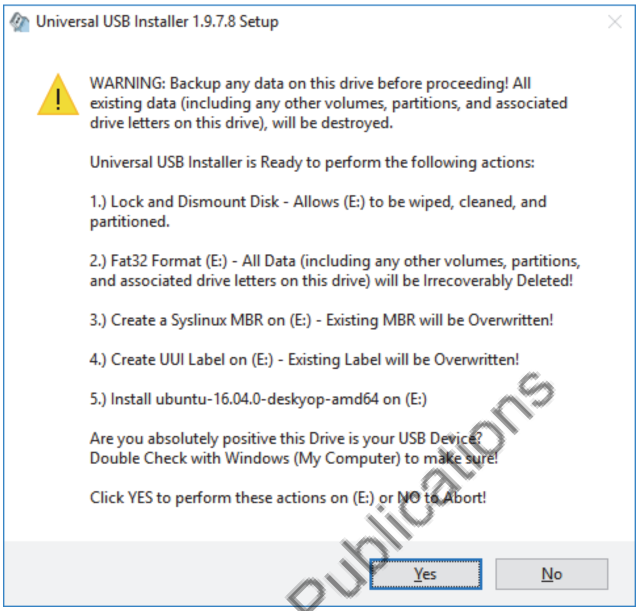
Step 4: Set a Persistent file size for storing changes (Optional).

0 MB

Universal USB Installer https://www.pendrivelinux.com

Step 4: Install Ubuntu.

Click YES to Install Ubuntu in USB.



Universal USB Installer 1.9.7.8 Setup

WARNING: Backup any data on this drive before proceeding! All existing data (including any other volumes, partitions, and associated drive letters on this drive), will be destroyed.

Universal USB Installer is Ready to perform the following actions:

1.) Lock and Dismount Disk - Allows (E:) to be wiped, cleaned, and partitioned.

2.) Fat32 Format (E:) - All Data (including any other volumes, partitions, and associated drive letters on this drive) will be Irrecoverably Deleted!

3.) Create a Syslinux MBR on (E:) - Existing MBR will be Overwritten!

4.) Create UUI Label on (E:) - Existing Label will be Overwritten!

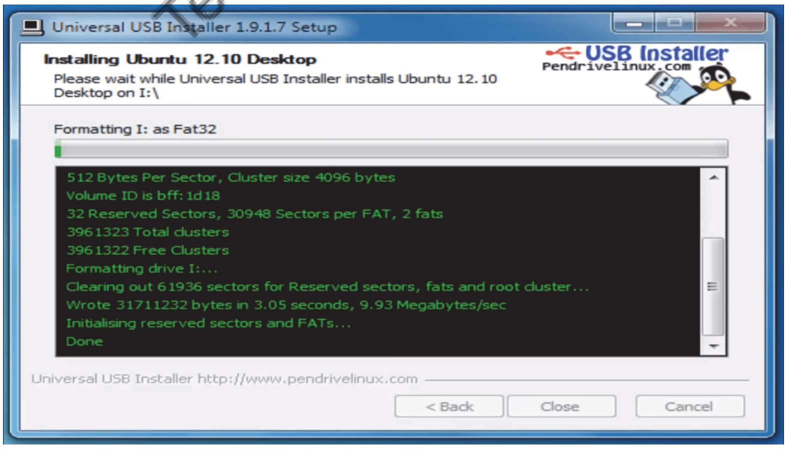
5.) Install ubuntu-16.04.0-deskyop-amd64 on (E:)

Are you absolutely positive this Drive is your USB Device? Double Check with Windows (My Computer) to make sure!

Click YES to perform these actions on (E:) or NO to Abort!

Step 5: Check your window.

After everything has been installed and configured, a small window will appear Congratulations! You now have Ubuntu on a USB stick, bootable and ready to go.



Universal USB Installer 1.9.1.7 Setup

Installing Ubuntu 12.10 Desktop

Please wait while Universal USB Installer installs Ubuntu 12.10 Desktop on I:\

Formatting I: as Fat32

```
512 Bytes Per Sector, Cluster size 4096 bytes
Volume ID is bff:1d18
32 Reserved Sectors, 30948 Sectors per FAT, 2 fats
3961323 Total clusters
3961322 Free Clusters
Formatting drive I:...
Clearing out 61936 sectors for Reserved sectors, fats and root cluster...
Wrote 31711232 bytes in 3.05 seconds, 9.93 Megabytes/sec
Initialising reserved sectors and FATs...
Done
```

Universal USB Installer http://www.pendrivelinux.com

Part-2 : Installing Linux using CD-ROM

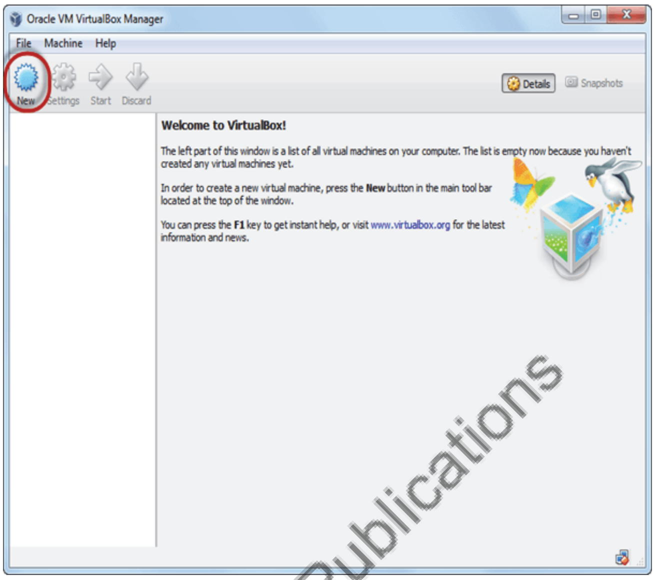
Those who like the way a CD runs should try using this method.

- **Step 1:** **Download the .iso or the OS files onto your computer from this link http://www.ubuntu.com/download/desktop.**
- **Step 2:** **Burn the files to a CD.**

- **Step 3:** **Boot your computer from the optical drive and follow the instructions as they come.**

PART 3 : Create a Machine in Virtual Box

Step-1: Open Virtual box and click on new button



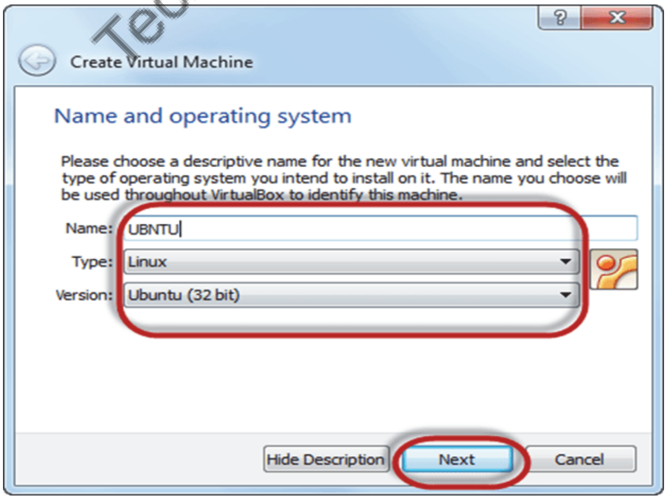
Welcome to VirtualBox!

The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for the latest information and news.

Step-2: In next window, give the name of your OS which you are installing in virtual box. And select OS like Linux and version as Ubuntu 32 bit. And click on next



Name and operating system

Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

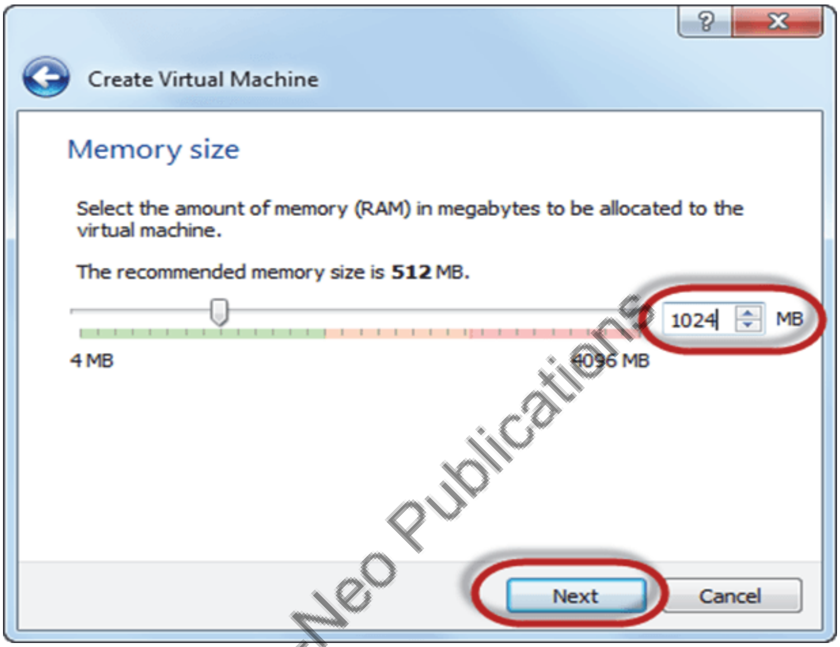
Name: UBNTU

Type: Linux

Version: Ubuntu (32 bit)

Note : For 64bit Ubuntu select Version as Ubuntu(64 bit)

- **Step-3: Now Allocate Ram Size To your Virtual OS. I recommended keeping 1024mb (1 GB) ram to run Ubuntu better. And click on next.**



Create Virtual Machine

Memory size

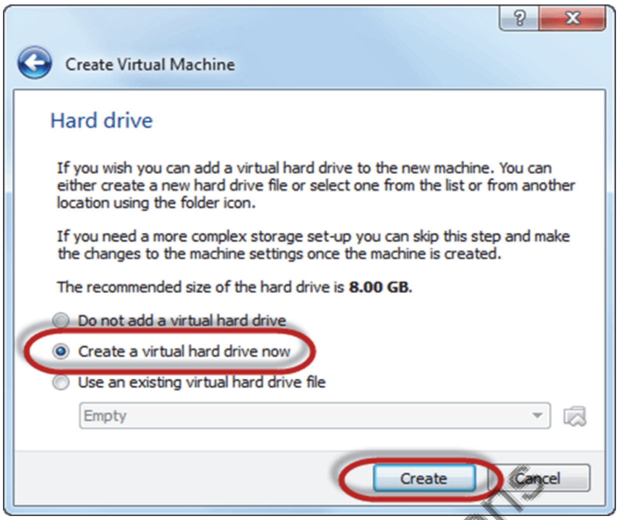
Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **512** MB.

1024 MB

- **Step-4: Now To run OS in virtual box we have to create virtual hard disk, click on create a virtual hard drive now and click on create button.**

The virtual hard disk is where the OS installation files and data/applications you create/install in this Ubuntu machine will reside



Create Virtual Machine

Hard drive

If you wish you can add a virtual hard drive to the new machine. You can either create a new hard drive file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard drive is **8.00 GB**.

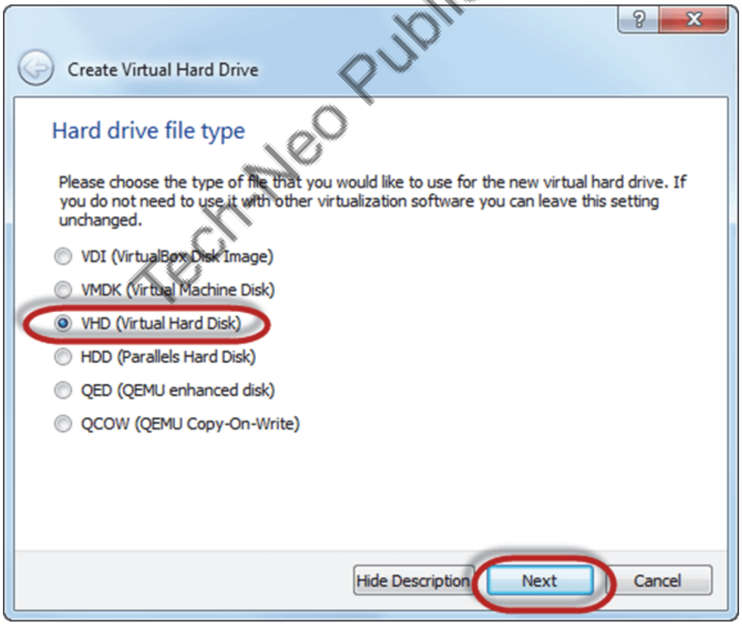
○ Do not add a virtual hard drive

● Create a virtual hard drive now

○ Use an existing virtual hard drive file

Empty

Step-5: Select VHD (virtual hard disk) option and click on next.



Create Virtual Hard Drive

Hard drive file type

Please choose the type of file that you would like to use for the new virtual hard drive. If you do not need to use it with other virtualization software you can leave this setting unchanged.

○ VDI (VirtualBox Disk Image)

○ VMDK (Virtual Machine Disk)

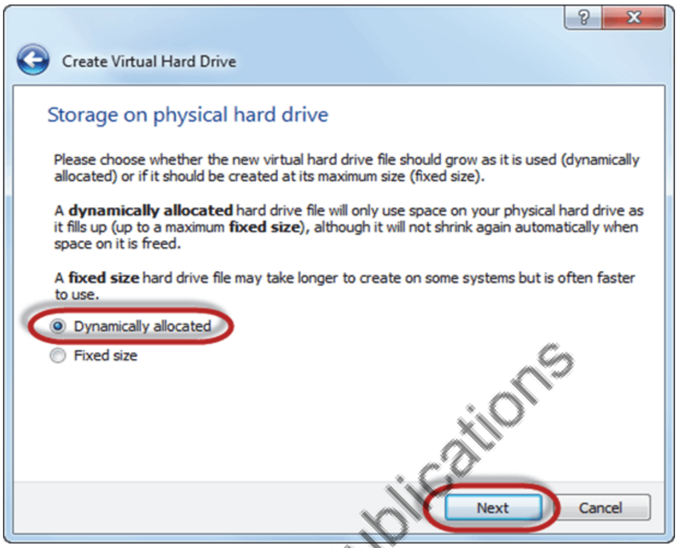
● VHD (Virtual Hard Disk)

○ HDD (Parallels Hard Disk)

○ QED (QEMU enhanced disk)

○ QCOW (QEMU Copy-On-Write)

- **Step-6: Click on dynamic allocated and click on next. This means that the size of the disk will increase dynamically as per requirement.**



Storage on physical hard drive

Please choose whether the new virtual hard drive file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

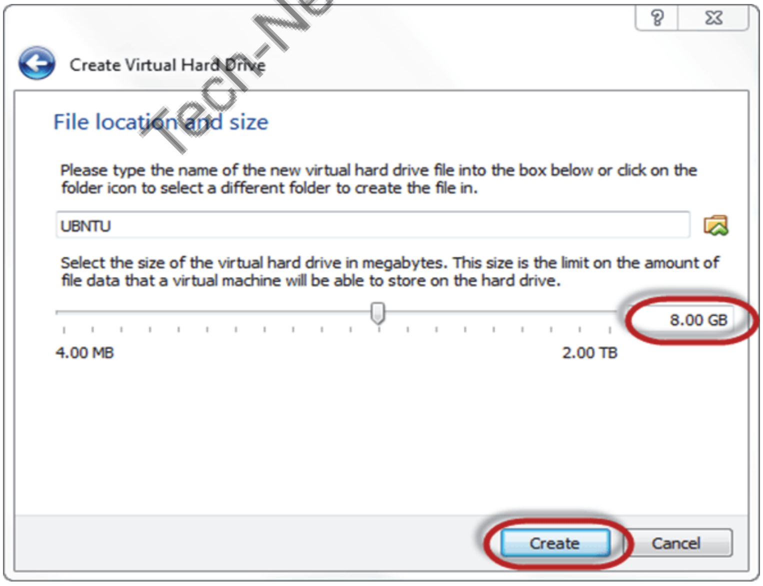
A **dynamically allocated** hard drive file will only use space on your physical hard drive as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard drive file may take longer to create on some systems but is often faster to use.

● Dynamically allocated

○ Fixed size

- **Step-7: Allocate memory to your virtual hard drive .8GB recommended. Click on create button.**



File location and size

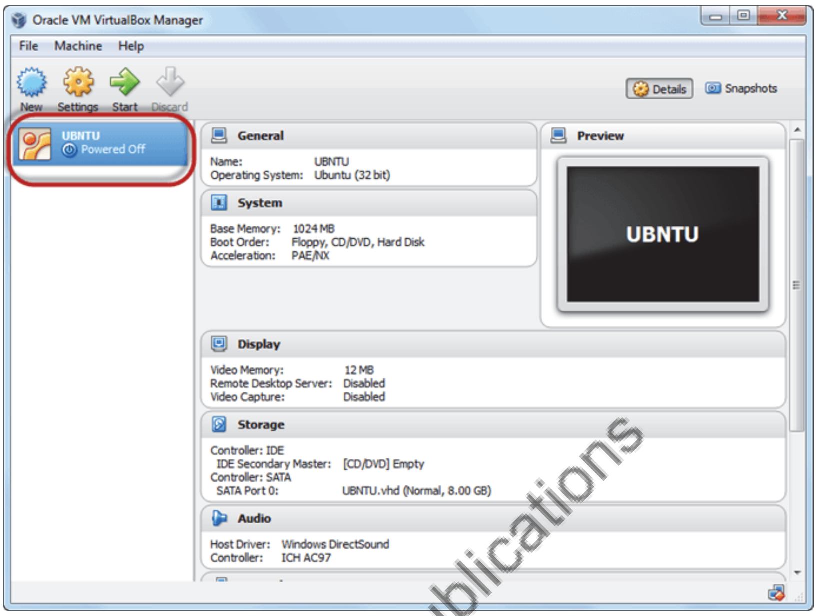
Please type the name of the new virtual hard drive file into the box below or click on the folder icon to select a different folder to create the file in.

UBNTU

Select the size of the virtual hard drive in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard drive.

8.00 GB

Step-8: Now you can see the machine name in left panel



Name: UBNTU
Operating System: Ubuntu (32 bit)

Base Memory: 1024 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: PAE/NX

Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

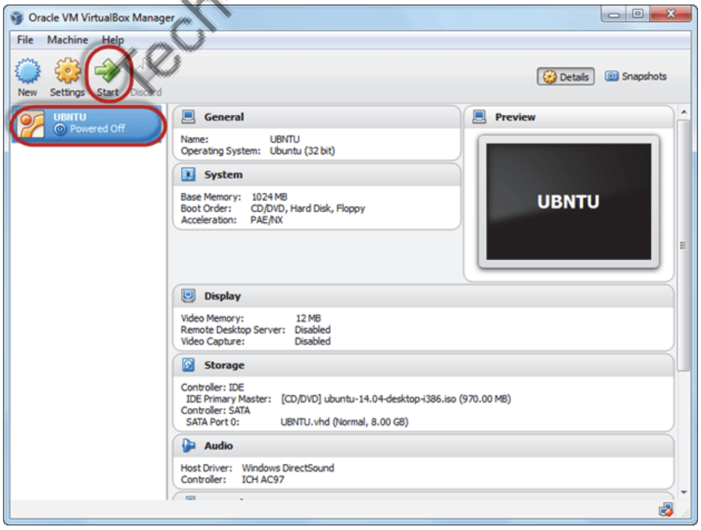
Controller: IDE
IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
SATA Port 0: UBNTU.vhd (Normal, 8.00 GB)

Host Driver: Windows DirectSound
Controller: ICH AC97

So a Machine (PC) with 8GB Hardisk, 1GB RAM is ready.

How to Install Ubuntu

Step 1: Select the Machine and Click on Start



Name: UBNTU
Operating System: Ubuntu (32 bit)

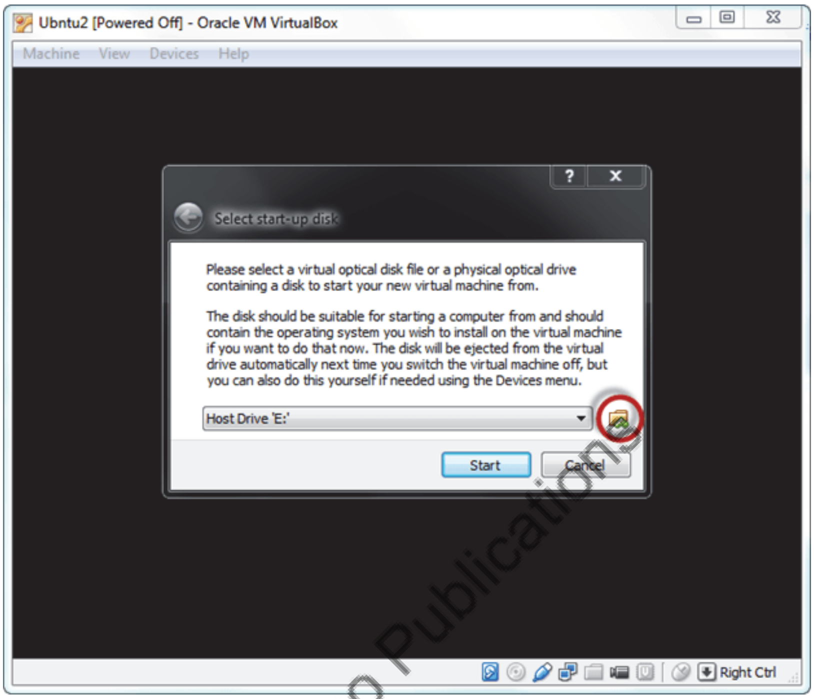
Base Memory: 1024 MB
Boot Order: CD/DVD, Hard Disk, Floppy
Acceleration: PAE/NX

Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Controller: IDE
IDE Primary Master: [CD/DVD] ubuntu-14.04-desktop-i386.iso (970.00 MB)
Controller: SATA
SATA Port 0: UBNTU.vhd (Normal, 8.00 GB)

Host Driver: Windows DirectSound
Controller: ICH AC97

Step 2: Select the Folder Option


Ubntu2 [Powered Off] - Oracle VM VirtualBox
Machine View Devices Help
Select start-up disk
Right Ctrl

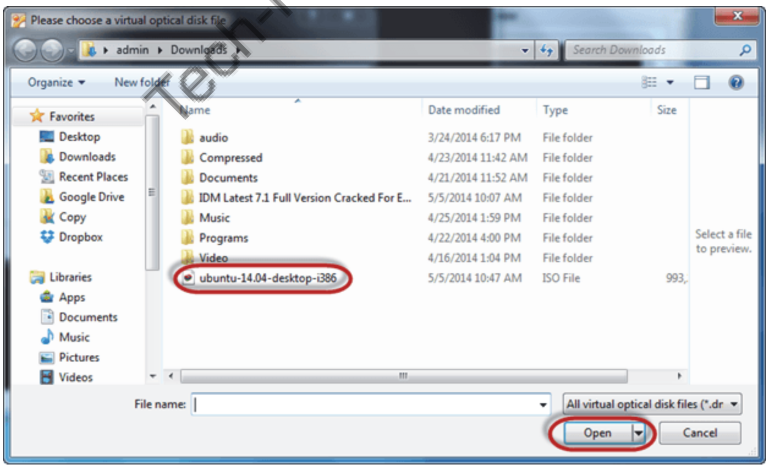
Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

Host Drive 'E:'

Start Cancel

Step 3: Select the Ubuntu iso file



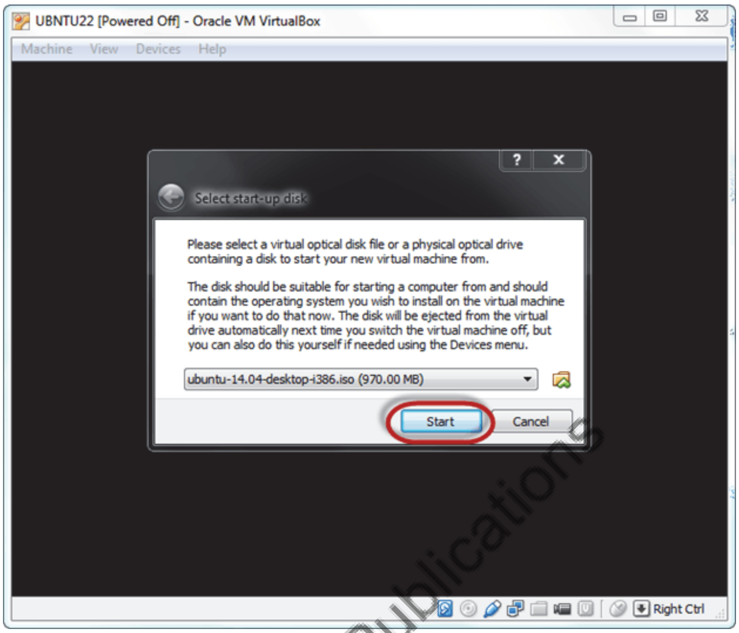
Name	Date modified	Type	Size
audio	3/24/2014 6:17 PM	File folder	
Compressed	4/23/2014 11:42 AM	File folder	
Documents	4/21/2014 11:52 AM	File folder	
IDM Latest 7.1 Full Version Cracked For E...	5/5/2014 10:07 AM	File folder	
Music	4/25/2014 1:59 PM	File folder	
Programs	4/22/2014 4:00 PM	File folder	
Video	4/16/2014 1:04 PM	File folder	
ubuntu-14.04-desktop-i386	5/5/2014 10:47 AM	ISO File	993,

File name:

All virtual optical disk files (*.dr

Open Cancel

Step 4: Click Start



UBNTU22 [Powered Off] - Oracle VM VirtualBox

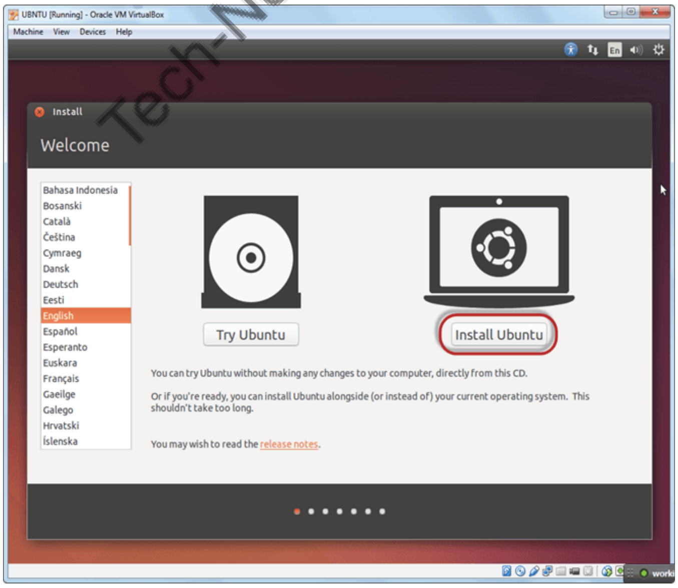
Select start-up disk

Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

ubuntu-14.04-desktop-i386.iso (970.00 MB)

Step-5: You have an option to Run Ubuntu WITHOUT installing. In this tutorial will install Ubuntu



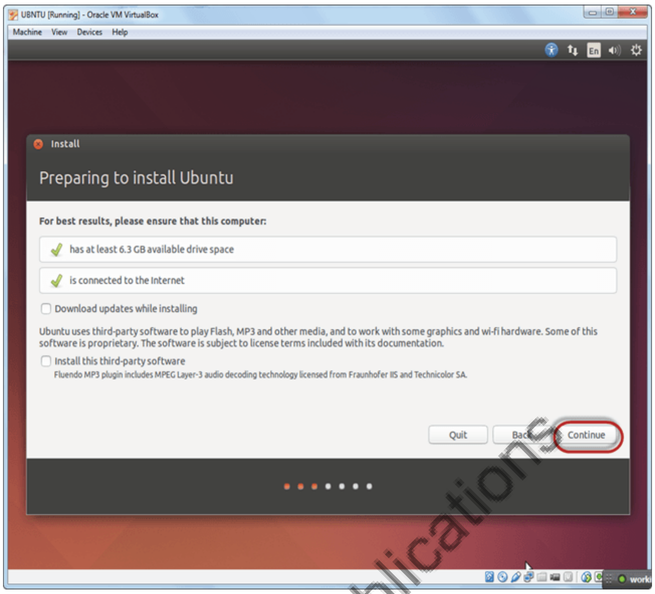
UBNTU [Running] - Oracle VM VirtualBox

You can try Ubuntu without making any changes to your computer, directly from this CD.

Or if you're ready, you can install Ubuntu alongside (or instead of) your current operating system. This shouldn't take too long.

You may wish to read the release notes.

Step-6: Click continue.



UBNTU [Running] - Oracle VM VirtualBox
Machine View Devices Help
Install

Preparing to install Ubuntu

For best results, please ensure that this computer:

✓ has at least 6.3 GB available drive space

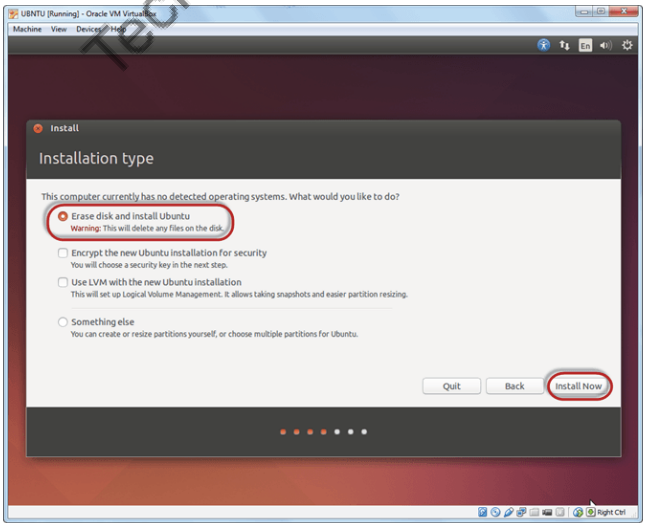
✓ is connected to the Internet

□ Download updates while installing

Ubuntu uses third-party software to play Flash, MP3 and other media, and to work with some graphics and wi-fi hardware. Some of this software is proprietary. The software is subject to license terms included with its documentation.

□ Install this third-party software
Fluendo MP3 plugin includes MPEG Layer-3 audio decoding technology licensed from Fraunhofer IIS and Technicolor SA.

Step-7: Select option to erase the disk and install Ubuntu and click on install now. This option installs Ubuntu into our virtual hard drive which is we made earlier. It will not harm your PC or Windows installation



UBNTU [Running] - Oracle VM VirtualBox
Machine View Devices Help
Install

Installation type

This computer currently has no detected operating systems. What would you like to do?

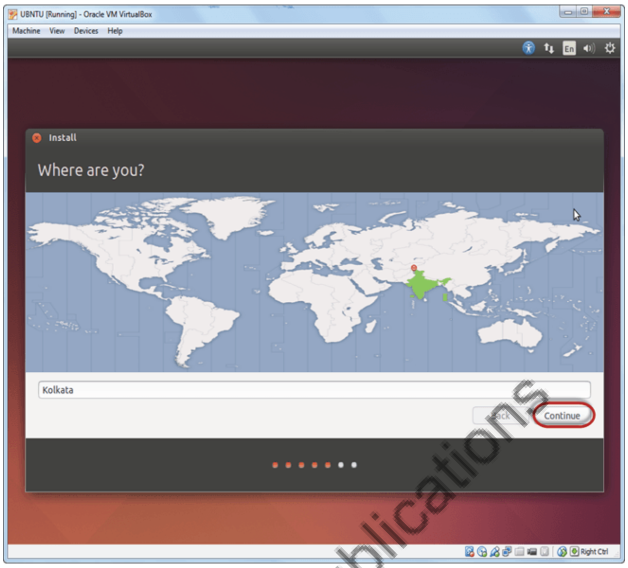
● Erase disk and install Ubuntu
Warning: This will delete any files on the disk.

□ Encrypt the new Ubuntu installation for security
You will choose a security key in the next step.

□ Use LVM with the new Ubuntu installation
This will set up Logical Volume Management. It allows taking snapshots and easier partition resizing.

○ Something else
You can create or resize partitions yourself, or choose multiple partitions for Ubuntu.

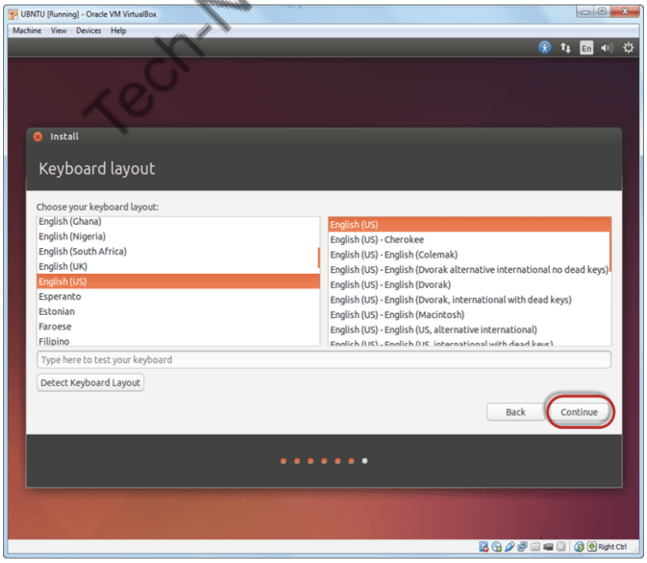
- **Step-8: Select your location for setting up time zone, and click on continue**


UBNTU [Running] - Oracle VM VirtualBox
Machine View Devices Help
Install
Back
Continue

Where are you?

Kolkata

- **Step-9: Select your keyboard layout, by default English (US) is selected but if you want to change then, you can select in the list. And click on continue**


UBNTU [Running] - Oracle VM VirtualBox
Machine View Devices Help
Install
Detect Keyboard Layout
Back
Continue

Keyboard layout

Choose your keyboard layout:

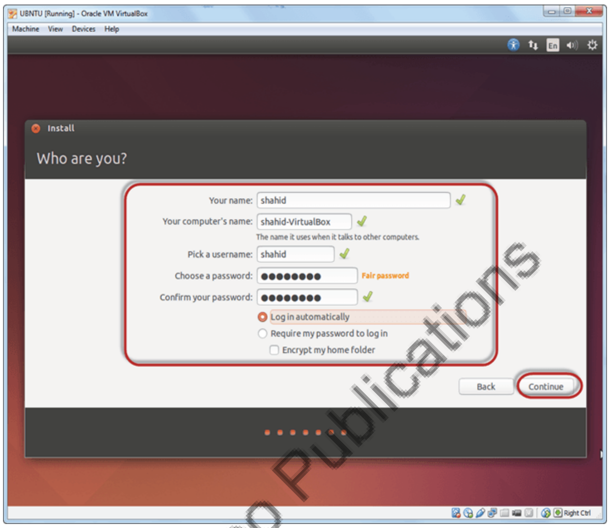
English (Ghana)
English (Nigeria)
English (South Africa)
English (UK)
English (US)
Esperanto
Estonian
Faroese
Filipino

English (US)
English (US) - Cherokee
English (US) - English (Colemak)
English (US) - English (Dvorak alternative international no dead keys)
English (US) - English (Dvorak)
English (US) - English (Dvorak, international with dead keys)
English (US) - English (Macintosh)
English (US) - English (US, alternative international)
English (US) - English (US, international with dead keys)

Type here to test your keyboard

➧ Step-10: Select your username and password for your Ubuntu admin account.

This information has been needed for installing any software package into Ubuntu and also for login to your OS. Fill up your details and tick on login automatically to ignore login attempt and click on continue



Who are you?

Your name: shahid ✓

Your computer's name: shahid-VirtualBox ✓

The name it uses when it talks to other computers.

Pick a username: shahid ✓

Choose a password: ●●●●●●●● Fair password

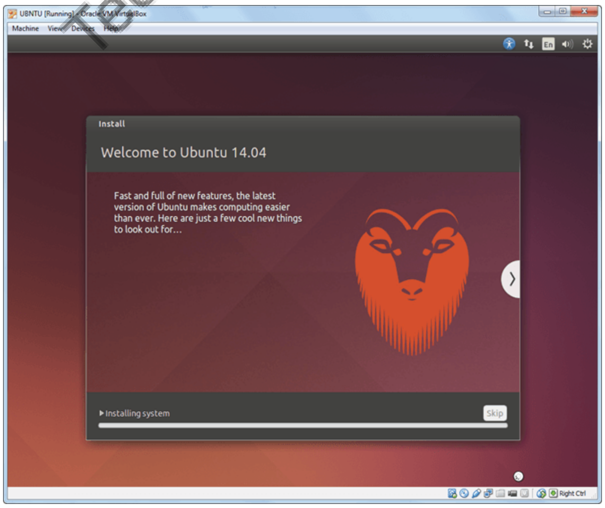
Confirm your password: ●●●●●●●● ✓

● Log in automatically

○ Require my password to log in

□ Encrypt my home folder

➧ Step-11: Installation process starts. May take up to 30 minutes. Please wait until installation process completes.

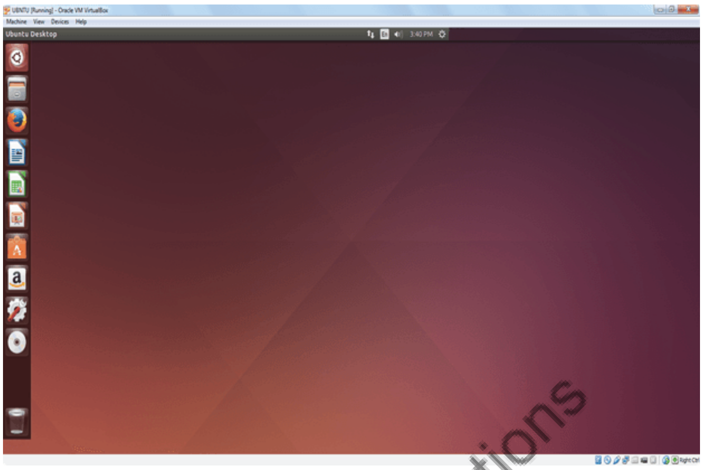


Welcome to Ubuntu 14.04

Fast and full of new features, the latest version of Ubuntu makes computing easier than ever. Here are just a few cool new things to look out for…

▸ Installing system

Step-12: After finishing the installation, you will see Ubuntu Desktop.


UBUNTU [Running] - Oracle VM VirtualBox
Machine View Devices Help
Ubuntu Desktop
3:40 PM
Right Ctrl

▸ Experiment No 2A

Title : Windows Dos Command

Aim: To study following Windows Dos Commands – Date, Time, prompt, md, cd, rd, path

☞ Procedure

1. Date(Displays or sets the date)

The Date command displays the current system date and allows you to change it.

☞ Syntax

```
DATE [/T | date]
```

- Type **DATE** without parameters to display the current date setting and a prompt for a new one. Press ENTER to keep the same date.
- If Command Extensions are enabled the DATE command supports the **/T** switch which tells the command to just output the current date, without prompting for a new date.
- Example: To Display Todays Date

```
C:\> date
The current date is: Wed 08/08/2023
Enter the new date: (mm-dd-yy)
```

- If Enter is pressed then C:\> will be displayed
- Example: To Display Todays Date

```
C:\> date
The current date is: Wed 08/08/2023
Enter the new date: 08/10/2023
```

- When enter is pressed will change the date to Thursday, August 10, 2023.

2. Time(Displays or sets the system time)

The Time command displays the current system time and allows you to change it.

Syntax

```
TIME [/T | time]
```

- Type **TIME** with no parameters to display the current time setting and a prompt for a new one. Press ENTER to keep the same time.
- If Command Extensions are enabled the TIME command supports the **/T** switch which tells the command to just output the current time, without prompting for a new time.
- Example: To display time

```
C:\> time
The current time is:  8:38:50.85
Enter the new time:
```

- If enter is pressed then C:\> will be displayed
- Example: To change time

```
C:\> time
The current time is:  8:38:50.85
Enter the new time: 8:50:00
```

- If enter is pressed then C:\> will be displayed but new time is set

3. Prompt (Changes the command prompt text)

Changes the cmd.exe command prompt.

Synax

```
PROMPT [text]
```

- text Specifies a new command prompt.

  Prompt can be made up of normal characters and the following special codes:

```
 $A   & (Ampersand)
 $B   | (pipe)
 $C   ( (Left parenthesis)
 $D   Current date
 $E   Escape code (ASCII code 27)
 $F   ) (Right parenthesis)
 $G   > (greater-than sign)
```

```
  $H   Backspace (erases previous character)
  $L   < (less-than sign)
  $N   Current drive
  $P   Current drive and path
  $Q   = (equal sign)
  $S     (space)
  $T   Current time
  $V   Windows version number
  $_   Carriage return and linefeed
  $$   $ (dollar sign)
If Command Extensions are enabled the PROMPT command supports the
following additional formatting characters:
  $+   zero or more plus sign (+) characters depending upon the
       depth of the PUSHD directory stack, one character for each
       level pushed.
  $M   Displays the remote name associated with the current drive
       letter or the empty string if current drive is not a network
       drive.
```

Examples

```
C:\Users\Manoj>prompt $A
&
C:\Users\Manoj>prompt $B
|
C:\Users\Manoj>prompt $C
(
C:\Users\Manoj>prompt $Q
=
C:\Users\Manoj>prompt $$
$
```

4. md / mkdir(Creates a directory)

The md (or mkdir) command creates a new directory.

☞ Syntax

```
MKDIR [drive:]path
MD [drive:]path
```

If Command Extensions are enabled MKDIR changes as follows:

- MKDIR creates any intermediate directories in the path, if needed.
- For example, assume \a does not exist then:

```
mkdir \a\b\c\d is the same as:
mkdir \a
chdir \a
mkdir b
chdir b
mkdir c
chdir c
mkdir d
```

which is what you would have to type if extensions were disabled.

- Example: (md or mkdir both are same)

```
C:\> md ManojKavedia
```

- This will create a new directory called " ManojKavedia" in the current directory.

```
C:\> md manoj\kavedia
```

- This will create a new directory called " manoj" and subdirectory "kavedia" in the "manoj" directory.

5. cd (Change the directory)

The cd command changes the current directory.

☞ Syntax

```
CHDIR [..]
CD [/D] [drive:][path]
CD [..]
```

- Specifies that you want to change to the parent directory.
- Type CD drive: to display the current directory in the specified drive.
- Type CD without parameters to display the current drive and directory.
- Use the /D switch to change current drive in addition to changing current directory for a drive.

- CHDIR command does not treat spaces as delimiters, so it is possible to CD into a subdirectory name that contains a space without surrounding the name with quotes. For example:

```
cd \winnt\profiles\username\programs\start menu
```

is the same as:

```
cd "\winnt\profiles\username\programs\start menu"
```

which is what you would have to type if extensions were disabled.

☞ Syntax

```
cd [directory name]
```

Example:

```
C:\> cd ManojKavedia
```

- This will change the current directory to "ManojKavedia".

```
C:\Users\Manoj>cd manoj\kavedia
C:\Users\Manoj\manoj\kavedia>
```

- This will change the directory so sub directory kavedia inside manoj directory

```
C:\Users\Manoj\manoj\kavedia>cd..
```

- This will change current directory to its parent directory

```
C:\Users\Manoj\manoj>cd..
```

- This will change current directory to its parent directory

```
C:\Users\Manoj>
```

6. Rd / Rmdir(Removes (deletes) a directory)

The Rd (or Rmdir) command removes an empty directory.

☞ Syntax

```
RMDIR [/S] [/Q] [drive:]path
RD [/S] [/Q] [drive:]path
/S  Removes all directories and files in the specified directory in addition to the
        directory itself.  Used to remove a directory tree.
  /Q     Quiet mode, do not ask if ok to remove a directory tree with /S
```

Example

```
C:\> rd ManojKavedia
```

- This will remove the directory called " ManojKavedia" if it is empty.

```
C:\Users\Manoj>rd manoj/s
manoj, Are you sure (Y/N)? y
```

- This will delete subdirectories inside main directory

```
C:\Users\Manoj>cd manoj
```

- The system cannot find the path specified.

```
C:\Users\Manoj>rd manoj/s/q
C:\Users\Manoj>
```

- This will delete subdirectories inside main directory, wont ask for confirmation

```
C:\Users\Manoj>cd manoj
```

- The system cannot find the path specified.

7. Path(Displays or sets a search path for executable files.)

The Path command sets the search path for executable files.

☞ Syntax

```
PATH [[drive:]path[;...][;%PATH%]
PATH ;
```

- Type PATH ; to clear all search-path settings and direct cmd.exe to search only in the current directory.
- Type PATH without parameters to display the current path.
- Including %PATH% in the new path setting causes the old path to be appended to the new setting.
- Example:

```
C:\> path c:\myprograms;c:\windows\system32
C:\Users\Manoj>path c:\myprograms;c:\windows\system32
C:\Users\Manoj>path
PATH=c:\myprograms;c:\windows\system32
%path% will append the existing path
C:\Users\Manoj>path =%path%;c:\javaprogram;c:\python
C:\Users\Manoj>path
PATH=c:\myprograms;c:\windows\system32;c:\javaprogram;c:\python
```

▸ Experiment No 2B.

Title : Windows Dos Command

Aim: To study following Windows Dos Commands – Chkdsk, copy, xcopy, format, fidsk, cls, defrag, del, move.

☞ Procedure

1. Chkdsk (Checks a disk and displays a status report.)

chkdsk stands for "Check Disk" and is a command-line utility in Windows that is used to check and repair file system issues and bad sectors on a disk. It can be used to diagnose and fix errors on your hard drive or other storage devices.

☞ Syntax

```
CHKDSK [volume[[path]filename]]] [/F] [/V] [/R] [/X] [/I] [/C] [/L[:size]] [/B]
[/scan] [/spotfix]
```

volume	Specifies the drive letter (followed by a colon), mount point, or volume name
filename	FAT/FAT32 only: Specifies the files to check for fragmentation.
/F	Fixes errors on the disk.
/V	On FAT/FAT32: Displays the full path and name of every file on the disk. On NTFS: Displays cleanup messages if any.
/R	Locates bad sectors and recovers readable information(implies /F, when /scan not specified).
/L:size	NTFS only: Changes the log file size to the specified
	number of kilobytes. If size is not specified, displays current size.
/X	Forces the volume to dismount first if necessary.
	All opened handles to the volume would then be invalid (implies /F).

/I	NTFS only: Performs a less vigorous check of index entries.
/C	NTFS only: Skips checking of cycles within the folder structure.
/B	NTFS only: Re-evaluates bad clusters on the volume (implies /R)

The /I or /C switch reduces the amount of time required to run Chkdsk by skipping certain checks of the volume.

Example

```
C:\> chkdsk /f/r
```

In this example, the /f flag fixes errors, and the /r flag locates bad sectors and recovers readable information. If necessary, the /x flag could be used to force dismount the volume before checking.

2. Copy(Copies one or more files to another location)

☞ Syntax

```
COPY [/D] [/V] [/N] [/Y | /-Y] [/Z] [/L] [/A | /B ] source [/A | /B]
     [+ source [/A | /B] [+ ...]] [destination [/A | /B]]
```

source	Specifies the file or files to be copied.
/A	Indicates an ASCII text file.
/B	Indicates a binary file.
/D	Allow the destination file to be created decrypted destination Specifies the directory and/or filename for the new file(s).
/V	Verifies that new files are written correctly.
/N	Uses short filename, if available, when copying a file with a non-8dot3 name.
/Y	Suppresses prompting to confirm you want to overwrite an existing destination file.

/-Y	Causes prompting to confirm you want to overwrite an existing destination file.
/Z	Copies networked files in restartable mode.
/L	If the source is a symbolic link, copy the link to the target instead of the actual file the source link points to.

Examples

1. Suppose you want to copy a file named "file.txt" from the "C:\Documents" folder to the "D:\Backup" folder:

```
C:\> copy C:\Documents\file.txt D:\Backup
```

2. You can also use wildcards to copy multiple files. For example, to copy all text files from a folder to another folder:

```
C:\> copy C:\SourceFolder\*.txt D:\DestinationFolder
```

3. If you want to combine the contents of multiple text files into a single file, you can do the following:

Content of file1.txt

```
C:\Users\Manoj>copy file1.txt con
My self ManojKavedia
Assistant professor in Thadomal Shahani Engineering College, Bandra
        1 file(s) copied.
```

Content of file2.txt

```
C:\Users\Manoj>copy file2.txt con
YouTube Channel : IotWala
Mobile no : 8329988738
watapp no : 9324258878
        1 file(s) copied.
```

Combine Multiple Files

```
C:\> copy C:\Files\file1.txt + C:\Files\file2.txt combinedFile.txt
```

In this example, the contents of "file1.txt" and "file2.txt" will be combined into a new file named "CombinedFile.txt."

```
C:\Users\Manoj>copy file.txt+file2.txt combinedfile.txt
```

```
file.txt
file2.txt
        1 file(s) copied.
```

Content of Combined file

```
C:\Users\Manoj>copy combinedfile.txt con
My self ManojKavedia
Assistant professor in Thadomal Shahani Engineering College, Bandra
YouTube Channel : IotWala
Mobile no : 8329988738
watapp no : 9324258878
```

```
        1 file(s) copied.
```

a. To create a file

```
C:\Users\Manoj>copy con manoj.txt
My self ManojKavedia
Assistant professor in Thadomal Shahani Engineering College, Bandra
YouTube Channel : IotWala
Mobile no : 8329988738
watapp no : 9324258878
^Z
```

```
        1 file(s) copied.
```

b. To display File

```
C:\Users\Manoj>copy manoj.txt con
My self ManojKavedia
Assistant professor in Thadomal Shahani Engineering College, Bandra
YouTube Channel : IotWala
Mobile no : 8329988738
watapp no : 9324258878
```

```
        1 file(s) copied.
```

Note : con keyword at source will help creating a file and at destination will help in displaying file

2. xcopy (command-line utility)

The xcopy command is a powerful command-line utility in Windows used to copy files and directories with more advanced options compared to the basic copy command. It allows you to copy multiple files, directories, and their subdirectories while providing options for preserving attributes, copying only new or changed files, and more.

☞ Syntax

The basic syntax of the xcopy command is as follows:

xcopy source destination [/options]

source

Specifies the source file(s) or directory to be copied. You can use wildcards (* and ?) to specify multiple files.

destination

Specifies the destination location where the files/directories will be copied.

/options

Specifies various options for the copying process. Some commonly used options are:

- /s: Copies directories and subdirectories except empty ones.
- /e: Copies directories and subdirectories, including empty ones.
- /i: If the destination does not exist and you are copying multiple files, it assumes the destination is a directory and prompts for confirmation.
- /y: Suppresses the confirmation prompt when overwriting existing files.
- /d: Copies only files that are newer than those already in the destination.
- /h: Copies hidden and system files.
- /r: Overwrites read-only files.
- /k: Copies attributes.

Example

1. Suppose you want to copy all text files from the "C:\Source" directory and its subdirectories to the "D:\Destination" directory, including empty subdirectories:

```
C:\> xcopy C:\Source\*.txt D:\Destination /s
```

2. To copy an entire directory and its contents while preserving attributes, including hidden and system files:

```
C:\> xcopy C:\Source\Directory D:\Destination\Directory /e /h /k
```

If you want to copy only new or changed files from one directory to another:

```
C:\> xcopy C:\Source\*.txt D:\Destination /d
```

Note : The xcopy command provides various other options and combinations to suit your specific copying needs. It's a versatile tool for managing file copying tasks in the Windows Command Prompt.

For more details c:\> xcopy /? Can be used

1. format (create a new file system on a specified storage device)

The format command is used to create a new file system on a specified storage device, such as a hard drive, USB flash drive, or memory card. This command initializes the storage device and prepares it for storing files by creating a new file system structure. Be cautious when using this command, as it will erase all existing data on the device.

☞ Syntax

The basic syntax of the format command is as follows:

```
format <drive_letter:> [/options]
<drive_letter>:  Specifies the drive letter of the storage device you want to
format.
```

/options:	Specifies various formatting options. Some commonly used options are:
/fs:file_system:	Specifies the file system to be used (e.g., NTFS, FAT32).

/q:	Performs a quick format (does not scan for bad sectors).
/x:	Forces the volume to dismount before formatting.
/p:<passes>:	Specifies the number of formatting passes (default is 1).
/v:<label>:	Specifies a volume label for the formatted drive.
/a:size:	Specifies the allocation unit size (cluster size).

Example

Suppose you want to format a USB flash drive with the drive letter E: using the FAT32 file system, a volume label of "MyUSB," and a quick format:

```
C:\> format E: /fs:FAT32 /q /v:MyUSB
```

If you want to perform a full format with a specific allocation unit size of 4096 bytes (default is usually 512 bytes):

```
C:\> format D: /fs:NTFS /p:1 /a:4096
```

Please be extremely cautious when using the format command, as it will erase all data on the specified drive. Make sure you have backed up any important data before proceeding with the formatting.

For more details try c:\>format /? Help command

2. fdisk (partitioning tools)

The fdisk command is not a native Windows DOS command. **fdisk** is typically associated with partitioning tools used in some other operating systems, such as MS-DOS, Windows 9x, and early versions of Windows NT.

In modern Windows operating systems, including Windows XP, Vista, 7, 8, and 10, as well as Windows Server editions, the command-line utility for managing disks, partitions, and volumes is **diskpart**.

The fdisk command is a DOS command that is used to create, delete, and modify partitions on a hard drive. It is a powerful command that can be used to change the layout of your hard drive, but it should be used with caution as it can also damage your data.

Description

The fdisk command allows you to do the following:

- Create primary and extended partitions
- Create logical drives within extended partitions
- Delete partitions
- Change the size of partitions
- Set the active partition
- Format partitions

Syntax

The basic syntax of the fdisk command is as follows:

```
fdisk [drive letter]
```

For example, to use fdisk to manage the C drive, you would type the following command:

```
C:\> fdisk c:
```

The fdisk command will then display a menu of options that you can use to manage the partitions on the C drive.

Examples

To create a new primary partition on the C drive, you would type the following command:

```
C:\> fdisk c: /p
```

This will prompt you to enter the size of the new partition. Once you have entered the size, fdisk will create the new partition.

To delete a partition on the C drive, you would type the following command:

```
C:\>fdisk c: /d [partition number]
```

Replace [partition number] with the number of the partition that you want to delete.

To change the size of a partition on the C drive, you would type the following command:

```
C:\>fdisk c: /s [partition number] [new size]
```

Replace [partition number] with the number of the partition that you want to change the size of, and replace [new size] with the new size of the partition in megabytes.

Cautions

- The fdisk command is a powerful command that can be used to damage your data if used incorrectly. It is important to back up your data before using fdisk to make any changes to your hard drive partitions.
- It is also important to note that fdisk can only be used to manage partitions on physical hard drives. It cannot be used to manage partitions on virtual hard drives.

3. cls (Clear Screen)

The cls command is used in the Windows Command Prompt to clear the screen and provide a clean slate for displaying new output. It does not delete any command history; it simply clears the visible content from the screen.

Syntax

The syntax of the cls command is very simple:

```
cls
```

This command is entered directly into the Command Prompt, and no additional parameters are required.

Example:

When you have executed several commands in the Command Prompt, and the output has filled up the screen. cls command can be used to clear the screen and make it ready for new input:

```
C:\> dir
Volume in drive C is OS
Volume Serial Number is ABCD-EFGH
Directory of C:\
08/16/2023 02:30 PM <DIR> .
```

```
08/16/2023 02:30 PM <DIR> ..
08/16/2023 02:30 PM <DIR> Documents
08/16/2023 02:30 PM <DIR> Downloads
0 File(s) 0 bytes 4 Dir(s) 100,000,000,000 bytes free
C:\> cls
C:\>
```

The cls command has cleared the screen, and you're left with a clean Command Prompt ready for new input.

4. defrag (defragment a specified disk volume)

The defrag command is used in the Windows Command Prompt to defragment a specified disk volume. Defragmentation is the process of reorganizing the files stored on a disk so that they are stored in contiguous blocks. This can improve the performance and efficiency of reading and writing files, as it reduces the time the system takes to access data.

☞ Syntax

The syntax of the defrag command is as follows:

```
defrag <volume> [/options]
```

<volume>:	Specifies the drive letter or mount point of the volume you want to defragment.
/options:	Specifies various defragmentation options. Some commonly used options are:
/A:	Analyzes the volume for fragmentation but does not defragment.
/U:	Defragments the volume (consolidates free space and optimizes file placement).
/V:	Displays detailed progress information during defragmentation.

Example

To defragment the C: drive:

```
C:\> defrag C:
```

To analyze the fragmentation level without actually performing defragmentation:

```
C:\> defrag C: /A
```

To perform a full defragmentation:

```
C:\> defrag C: /U
```

Note : Keep in mind that the defragmentation process can take some time, especially on larger drives with a high level of fragmentation. It's a good practice to regularly defragment your hard drives to maintain optimal performance, but modern versions of Windows often perform automatic background optimization, making manual defragmentation less necessary for most users.

5. del (delete one or more files)

The del command (short for "delete") is used in the Windows Command Prompt to delete one or more files from a specified location. It permanently removes the specified files from the storage device.

☞ Syntax

The syntax of the del command is as follows:

```
del <filename(s)> [/options]
```

<filename(s)>:	specify multiple files.
/options:	Specifies various options for deleting files. Some commonly used options are:
/p:	Prompts for confirmation before deleting each file.
/f:	Forces deletion of read-only files.
/s:	Deletes specified files from all subdirectories.

Example:

To delete a file named "file.txt" from the current directory:

```
C:\> del file.txt
```

To delete all text files with the .txt extension from the current directory:

```
C:\> del *.txt
```

If you want to delete all text files from the current directory and its subdirectories:

```
C:\> del /s *.txt
```

To delete a file and prompt for confirmation before each deletion:

```
C:\> del /p file.txt
```

Note : Please exercise caution when using the del command, as it permanently removes files from the storage device, and the deleted data cannot be easily recovered. Make sure you are confident about the files you are deleting, and consider making backups of important data before using the del command.

Note : For more details c:\> del /? Can be used

5. move(move files or directories)

The move command is used in the Windows Command Prompt to move one or more files or directories from one location to another. It allows you to relocate files and directories while preserving their attributes.

☞ Syntax

The syntax of the move command is as follows:

```
move <source> <destination>
```

<source>:	Specifies the file or directory to be moved.
<destination>:	Specifies the location to which the file or directory should be moved.

Example

1. Suppose you want to move a file named "file.txt" from the "C:\Source" directory to the "D:\Destination" directory:

```
C:\> move C:\Source\file.txt D:\Destination
```

2. To move a directory named "Folder" and its contents from the current directory to the "D:\Backup" directory:

```
C:\> move Folder D:\Backup
```

3. If you want to rename a file while moving it, you can specify a new filename in the destination path:

```
C:\> move C:\Source\oldfile.txt D:\Destination\newfile.txt
```

4. To move multiple files from one directory to another:

```
C:\> move C:\Source\*.txt D:\Destination
```

Note : Please be cautious when using the move command, especially if moving or renaming files and directories across drives or locations. Ensure that the destination directory exists and that you have the necessary permissions to perform the move operation. The move command can result in permanent changes to your file system, so it's recommended to make backups of important data before performing such operations

- Experiment No 2C

Title : Windows Dos Command

Aim: To study following Windows Dos Commands – Diskcomp, diskcopy, diskpart, doskey, echo

☞ Procedure

1. Diskcomp(compare the contents of two floppy disks or disk images)

The diskcomp command is used to compare the contents of two floppy disks or disk images. This command is used to determine whether two floppy disks are identical or if there are any differences between them. It is not commonly used in modern Windows operating systems and may not be available in all versions of Windows. The diskcomp command has been largely replaced by more advanced file comparison tools and is not supported for all types of storage media.

☞ Syntax

The syntax of the diskcomp command is as follows:

```
diskcomp [drive1:][path1] [drive2:][path2]
```

[drive1:][path1]:	Specifies the drive letter and path of the first floppy disk or disk image.
[drive2:][path2]:	Specifies the drive letter and path of the second floppy disk or disk image.

Example:

To Compare two floppy disks, one in drive A: and the other in drive B:, and to compare their contents:

```
C:\> diskcomp A: B:
```

The command would compare the contents of the floppy disks in drives A: and B: and display a message indicating whether the disks are identical or if there are differences between them.

2. diskcopy (copy the entire contents of one floppy disk to another floppy disk)

The diskcopy command is used to copy the entire contents of one floppy disk to another floppy disk. This command is used to create a duplicate copy of a floppy disk. Similar to the diskcomp command, diskcopy is also not commonly used in modern Windows operating systems and may not be available in all versions of Windows. It's worth noting that this command is specifically designed for floppy disks and may not work with other types of storage media.

☞ Syntax

The syntax of the diskcopy command is as follows:

```
diskcopy [drive1: [drive2:]]
```

[drive1:]:	Specifies the source drive (the drive containing the source floppy disk).
[drive2:]:	Specifies the target drive (the drive where the copy of the floppy disk will be created). If this is not provided, the command will prompt you to insert the target disk.

Example:

A source floppy disk in drive A: and want to create a duplicate copy on another floppy disk in drive B:. The following command can be used:

```
C:\> diskcopy A: B:
```

The command would prompt you to insert the target floppy disk into drive B:. Once you insert the target disk, the command will copy the entire contents of the source floppy disk to the target floppy disk.

Note : The diskcopy command is tailored for floppy disks and may not be suitable for other types of storage media, such as USB drives or hard disks.

3. diskpart (command-line disk partitioning utility)

The diskpart command is a command-line disk partitioning utility available in Windows. It allows you to manage disks, partitions, and volumes on your system. With diskpart, one can create, delete, format, and manage partitions and volumes, as well as assign drive letters and perform other disk-related operations.

☞ Syntax

To start the diskpart utility, open a Command Prompt and type:

```
diskpart
```

This will launch the diskpart command-line tool. Once inside the diskpart prompt, One can enter various commands to manage your disks and partitions. Some commonly used commands include:

list disk:	Lists all available disks on the system.
select disk <disk_number>:	Selects a specific disk for further operations.
list partition:	Lists all partitions on the selected disk.
select partition <partition_number>:	Selects a specific partition for further operations.
create partition primary size=<size>:	Creates a primary partition of the specified size.
format fs=<file_system> quick:	Formats the selected partition with the specified file system (e.g., NTFS) quickly.
assign letter=<drive_letter>:	Assigns a drive letter to the selected volume.
exit:	Exits the diskpart utility.

Example

Use of diskpart to create a new primary partition, format it, and assign a drive letter:

1. Open Command Prompt.
2. Type diskpart and press Enter.
3. In the diskpart prompt:
   - Type list disk to see a list of available disks.
   - Type select disk <disk_number> to select the desired disk.
   - Type create partition primary size=102400 to create a new primary partition of 100 MB.
   - Type format fs=NTFS quick to format the partition as NTFS quickly.
   - Type assign letter=E to assign drive letter E: to the partition.
   - Type exit to exit diskpart.

Note : When using diskpart, as it directly affects disk partitions and data. Ensure that you have backups of important data before making any changes. Additionally, be aware that diskpart commands can vary slightly based on the version of Windows you are using, so consult the documentation or help resources for your specific version if needed.

4. doskey

The doskey command is used to manage and recall previously entered commands in the Windows Command Prompt. It allows you to create, edit, and recall macros for commonly used commands, making it easier to reuse and modify command lines without retyping them entirely.

☞ Syntax

The syntax of the doskey command is as follows:

```
doskey [/reinstall] [/listsize=<Size>] [/macrofile=<FileName>]
[macroname=[definition]]
```

/reinstall:	Installs a new copy of Doskey.
/listsize=<Size>:	Sets the number of commands that Doskey stores in its history buffer.
/macrofile=<FileName>:	Specifies a text file containing a list of macros.
macroname:	Specifies the name for a new macro or an existing macro to edit.
definition:	Specifies the command-line text to be associated with the macro.

Example:

Few examples of how to use the doskey command:

Creating a Macro:

If you frequently using a long command, and then you can create a macro named ls for it.

```
C:\> doskey ls=dir /w $*
```

Now, whenever ls is typed and press Enter, it will execute the dir /w command.

- **Recalling Macros:** After creating the ls macro, you can simply type ls and press Enter to run the associated dir /w command.
- **Listing Macros:** To see a list of currently defined macros, you can use:

```
C:\> doskey /macros
```

Editing Macros

To edit an existing macro, you can use:

```
C:\> doskey ls=dir /p $*
```

This will modify the ls macro to use dir /p instead of dir /w.

Using Macros with Arguments:

Macros can accept arguments. For example, if you create a macro called goto to navigate to a specific directory, you can use:

```
C:\> doskey goto=cd /d $*
```

Now, you can use goto C:\MyFolder to quickly navigate to the

```
"C:\MyFolder" directory.
```

Note : doskey macros are only available within the current Command Prompt session. They are not permanent and will be lost when you close the Command Prompt window.

5. echo

The echo command is used in the Windows Command Prompt to display text or enable/disable the display of commands in a batch script. When used without any arguments, the echo command simply toggles the display of commands on or off in the script execution. When used with text arguments, it displays the specified text on the screen.

☞ Syntax

The syntax of the echo command is as follows:

```
echo [on | off] [message]
```

- **on:** Enables the display of commands in a batch script. This is the default behavior.
- **off:** Disables the display of commands in a batch script.
- **message:** Specifies the text message to be displayed on the screen.

Example How to use echo command

Displaying Text

To display a message on the screen, simply provide the message after the echo command. For example:

```
C:\> echo Hello, World!
Hello, World!
```

Toggling Command Display

In a batch script, you can use @echo off to turn off the display of commands and @echo on to turn it back on. For example:

@echo off

echo This text will not be displayed.

@echo on

echo This text will be displayed.

Batch Script with Echoed Commands

A simple batch script that echoes some commands and their output:

```
@echo off echo Welcome to My Batch Script
echo.
echo Current directory: %CD%
echo List of files in this directory: dir /b
echo.
echo End of script.
```

When you run this script, it will display the echoed text along with the output of the dir /b command.

Redirecting Output

The output of the echo command to a text file. For example:

C:\> echo This is a test > output.txt

This will create a file named "output.txt" containing the text "This is a test."

The echo command is a fundamental tool for displaying messages and controlling the display of commands in batch scripts and the Command Prompt.

▶ Experiment No 2D.

Title : Windows Dos Command

Aim: To study following Windows Dos Commands – Edit, fc, find, rename, set, type, ver

☞ Procedure

1. Edit (launch the MS-DOS Editor)

The edit command is used to launch the MS-DOS Editor, a simple text editor that was included with early versions of MS-DOS and some Windows versions. It allowed users to create and edit text files from the command prompt. However, please note that the edit command is not available in modern versions of Windows, starting from Windows 95.

☞ Syntax

The syntax of the edit command is as follows:

```
edit [/b] [/h] [/s] [filename]
```

- **/b:** Opens a file in binary mode.
- **/h:** Displays the online Help for the editor.
- **/s:** Starts the editor without displaying the startup banner.
- **filename:** Specifies the name of the file to be opened or created.

Example:

To create a new text file named "mytext.txt" using the MS-DOS Editor:

```
C:\> edit mytext.txt
```

This would open the MS-DOS Editor, allowing you to create and edit the contents of the "mytext.txt" file.

Note : Since the edit command is not available in modern Windows versions, you can use alternative text editors like Notepad, Notepad++, Visual Studio Code, or other preferred text editors to create and edit text files.

2. Fc(File Compare)

The fc (File Compare) command is used in the Windows Command Prompt to compare the contents of two text files and display the differences between them. It's a useful tool for checking if two files are identical or identifying the specific lines that differ between them.

Syntax

The syntax of the fc command is as follows:

```
fc [/A] [/C] [/L] [/LBn] [/N] [/OFF[LINE]] [/T] [/U] [/W] [/nnnn]
[drive1:][path1]filename1 [drive2:][path2]filename2
```

- **/A:** Displays only first and last lines for each set of differences.
- **/C:** Performs a case-insensitive comparison.
- **/L:** Compares files as ASCII text.
- **/LBn:** Sets the maximum consecutive mismatches before stopping.
- **/N:** Displays the line numbers in the output.
- **/OFF[LINE]:** Do not skip files with offline attribute set.
- **/T:** Does not expand tabs to spaces.
- **/U:** Compares files as Unicode text.
- **/W:** Compresses white space (tabs and spaces) for comparison.
- **/nnnn:** Specifies the number of consecutive lines that must match after a mismatch.

Example:

1. There are two text files, "file1.txt" and "file2.txt," and to compare their contents using the fc command:

```
C:\> fc file1.txt file2.txt
```

2. This command would compare the contents of "file1.txt" and "file2.txt" and display the differences between them, including the lines that differ.

   If you want to perform a case-insensitive comparison and display the line numbers:

```
C:\> fc /C /N file1.txt file2.txt
```

Note : The fc command is specifically designed for comparing text files. It may not work as expected for non-text files or binary files.

2. find (search for a specific string of text)

The find command is used in the Windows Command Prompt to search for a specific string of text in one or more files. It scans the contents of text files and displays the lines that contain the specified search string.

Syntax

The syntax of the find command is as follows:

```
find "string" [filename]
```

- **"string":** Specifies the text string you want to search for. Enclose the string in double quotation marks.
- **[filename]:** Specifies the name of the file or files you want to search within. If omitted, find reads from the standard input (usually the keyboard).

Example

Content of file "sample.txt"

This is a sample text file. Hello, world! Welcome to the world of programming. Programming is fun.

1. To search for the word "world" in the file.

```
C:\> find "world" sample.txt
```

The output would be:

```
Hello, world! Welcome to the world of programming.
```

2. The find command can be used in combination with other commands, such as dir, to search for files containing a specific string. For example, to search for files containing the word "error" in the current directory:

```
C:\> dir /b | find "error"
```

This command uses the dir /b command to list the names of all files in the current directory, and then pipes (|) the output to the find command to filter out the files containing the word "error."

3. The find command is case-sensitive by default. If you want to perform a case-insensitive search, you can use the /i option:

```
C:\> find /i "world" sample.txt
```

Note : the find command is a basic way to search for text in files, there are more advanced text search and manipulation tools available, such as grep for Windows or search features in modern text editors.

3. rename (rename a file or group of files)

The rename command is used in the Windows Command Prompt to rename a file or group of files. It allows you to change the name of one or more files without changing their contents.

☞ Syntax

The syntax of the rename command is as follows:

```
rename <old_filename> <new_filename>
```

- **<old_filename>:** Specifies the current name of the file you want to rename.
- **<new_filename>:** Specifies the new name you want to assign to the file.

Example:

1. There is file named "oldfile.txt" in the current directory, and rename it to "newfile.txt."

```
C:\> rename oldfile.txt newfile.txt
```

This command would change the name of the file from "oldfile.txt" to "newfile.txt."

2. Wildcards character can be used to rename multiple files that match a specific pattern. For example, if to rename all .txt files in the current directory to have a .bak extension

```
C:\> rename *.txt *.bak
```

This command renames all .txt files to have a .bak extension.

4. set (to define or modify environment variables)

The set command is used in the Windows Command Prompt to define or modify environment variables. Environment variables are dynamic values that can affect the behavior of processes and applications. They are commonly used for storing configuration settings, paths, and other information that can be accessed by various programs.

Syntax:

The syntax of the set command is as follows:

```
set [variable=value]
```

- **variable:** Specifies the name of the environment variable you want to define or modify.
- **value:** Specifies the value you want to assign to the environment variable.

Example:

How to use the set command:

1. Defining an Environment Variable:

To define a new environment variable named MY_VARIABLE with the value Hello, World!, you would use:

```
C:\> set MY_VARIABLE=Hello, World!
```

2. Displaying Environment Variables:

To display a list of all defined environment variables, you can use the set command without any arguments:

```
C:\> set
```

This will display a list of all environment variables along with their values.

3. Using Environment Variables in Commands:

Environment variables can be used in commands and scripts. For example, if you've defined an environment variable MY_PATH with a specific path:

```
C:\> set MY_PATH=C:\MyFolder
```

command can be used like this:

```
C:\> dir %MY_PATH%
```

This would list the contents of the "C:\MyFolder" directory.

4. Modifying an Existing Environment Variable

In order to modify the value of an existing environment variable by reassigning it using the set command. For example, to change the value of the MY_VARIABLE variable:

```
C:\> set MY_VARIABLE=New Value
```

This updates the value of MY_VARIABLE to "New Value."

Note : Environment variables set using the set command are specific to the current Command Prompt session and any processes launched from that session. They are not permanent and will be lost when you close the Command Prompt window. To define environment variables that persist across sessions, you need to set them in the system or user environment variables settings.

4. Type(display the contents of a text file)

The type command is used in the Windows Command Prompt to display the contents of a text file directly in the console. It allows you to view the contents of a file without having to open it in a separate text editor.

Syntax:

The syntax of the type command is as follows:

```
type [drive:][path]filename
```

- **[drive:][path]filename:** Specifies the path to the text file you want to display.

Example:

1. Consider text file named "sample.txt" in the "C:\Documents" directory, and to view its contents using the type command can be used

```
C:\> type C:\Documents\sample.txt
```

This command would display the contents of the "sample.txt" file directly in the console window.

2. If the file is located in the current directory, you can omit the path:

```
C:\> type sample.txt
```

3. Wildcards character can be used to display the contents of multiple files that match a specific pattern. For example, to display the contents of all .txt files in the current directory,use following command

```
C:\> type *.txt
```

Note : For more help on type command use c:\>type /? command

5. Ver(display the version number of the operating system)

The ver command is used in the Windows Command Prompt to display the version number of the operating system.

Syntax:

```
ver
```

Example:

When you enter the ver command in the Command Prompt, it will display the version number of the operating system you are using.

```
Microsoft Windows [Version 10.0.19045.3208]
In this example, the version number of the operating system is
"10.0.19045.3208."
```

Experiment No 3A.

Title : Linux Command

Aim: To study following Linux Commands – pwd, cd, absolute and relative path, ls, mkdir, rmdir

☞ Procedure

1. pwd(Print Working Directory)

The pwd command in Linux stands for **"Print Working Directory."** It is used to display the current working directory, which is the directory you are currently in within the file system hierarchy. This command is helpful for quickly determining your location within the directory structure.

☞ Syntax

```
pwd [OPTION]
```

Options:

- **-L:** Displays the logical current working directory (default behavior).
- **-P:** Displays the physical current working directory, which may include symbolic links.

Example

1. Display the current working directory:

```
pwd
```

Output: /home/user/documents

2.Display the physical current working directory:

```
pwd -P
```

Output: /mnt/data

3.Display the logical current working directory (default behavior):

```
pwd -L
```

Output: /home/user/documents

> **Note:** The actual output of the pwd command will depend on your system's directory structure and your current location within it.

2. cd (change directory)

The cd command in Linux is used to change the current working directory. It allows you to navigate to different directories within the file system hierarchy.

☞ Syntax

```
cd [DIRECTORY]
```

Examples

1. Change to a specific directory:

```
$cd /home/user/documents
```

2. Change to the user's home directory:

```
$cd
```

3. Change to the parent directory:

```
$cd ..
```

4. Change to a subdirectory:

```
$cd myfolder
```

5. Change to the previous directory (using the - symbol):

```
$cd -
```

6. Change to a directory with spaces in its name (use quotes):

```
$cd "my folder"
```

7. Change to the directory using an absolute path:

```
$cd /usr/share
```

8. Change to the directory using a relative path:

```
$cd ../backup
```

3. absolute and relative path

In Linux, both absolute and relative paths are used to specify the location of files and directories within the file system. Understanding the difference between them is essential for efficient navigation and management of files and directories.

- **Absolute Path:** An absolute path specifies the complete path from the root directory (/) to the target file or directory. It provides the full and unambiguous location of the file or directory in the file system hierarchy.

☞ **Syntax**

/absolute/path/to/target

Example

```
/home/user/documents/file.txt
```

In this example, / represents the root directory, home is a subdirectory of the root, user is a subdirectory of home, and so on, until we reach the target file file.txt.

- **Relative Path:** A relative path specifies the location of a file or directory relative to the current working directory. It doesn't start from the root directory (/) but rather from the current location.

☞ **Syntax**

```
relative/path/to/target
```

Example

Suppose the current working directory is /home/user, and you want to reference file.txt located in the documents subdirectory:

```
documents/file.txt
```

In this example, the path is relative to the current working directory (/home/user), and it only specifies the necessary steps to reach the target from there.

☞ Usage

- Absolute paths are useful when you want to specify an exact location and avoid ambiguity, especially when working across different directories or scripts.
- Relative paths are convenient for navigating within a specific directory or its subdirectories.

> **Note:** represents the current directory, and **..** represents the parent directory. When using relative paths, the current working directory is assumed as the starting point. If the current working directory changes, the relative path will point to a different location.

Examples

```
Absolute Path:
/usr/share/applications/firefox.desktop
Relative Path (assuming the current working directory is /home/user):
documents/report.pdf
Relative Path (assuming the current working directory is /var/www):
../../html/index.html
```

4. ls (list the contents of a directory)

The ls command in Linux is used to list the contents of a directory. It provides information about files and directories within the specified location, including file names, permissions, ownership, size, and timestamps.

☞ Syntax

```
ls [OPTION] [DIRECTORY]
```

Options

- -l: Long listing format, displaying detailed information about each file or directory.
- -a: Display hidden files (files that start with a dot .).
- -h: Human-readable file sizes (e.g., 1K, 2M, 3G).
- -R: Recursively list subdirectories and their contents.
- -t: Sort files by modification time, with the most recently modified first.
- -S: Sort files by size, with the largest first.
- -r: Reverse the order of listing.
- --color: Colorize output for better readability.
- -i: Display inode number for each file.

Examples

1. List files and directories in the current directory:

```
$ls
```

2. List files and directories with detailed information:

```
ls -l
```

3. List all files and directories, including hidden ones:

```
ls -a
```

4. List files and directories with human-readable sizes:

```
$ls -lh
```

5. List files and directories in a specific directory:

```
$ls /path/to/directory
```

6. List files and directories recursively:

```
$ls -R
```

7. List files and directories sorted by modification time:

```
$ls -lt
```

8. List files and directories sorted by size:

```
$ls -lS
```

9. List files and directories in reverse order:

```
$ls -r
```

10. List files and directories with colorized output:

```
$ls --color
```

Note: The output of the ls command can vary based on the files and directories present in the specified location and the options used. By combining different options, you can tailor the ls command to your specific needs for listing and exploring the contents of directories.

5. mkdir (create new directories (folders))

The mkdir command in Linux is used to create new directories (folders) within the file system. It allows you to quickly and easily create directories to organize your files and other directories.

☞ Syntax

```
mkdir [OPTION] DIRECTORY
```

Options

- -p: Create parent directories if they don't exist. This option allows you to create nested directories in a single command.

Example

1. Create a new directory named "myfolder":

```
$mkdir myfolder
```

2. Create multiple directories "folder1", "folder2", and "folder3":

```
$mkdir folder1 folder2 folder3
```

3. Create a nested directory structure "parent/child/grandchild":

```
$mkdir -p parent/child/grandchild
```

4. Create a directory with a space in its name:

```
$mkdir "my folder"
```

5. Create a directory with specific permissions (using chmod):

```
$mkdir -m 755 mydirectory
```

Note: The mkdir command is used for creating directories, and you can specify one or more directory names as arguments. You can also use the -p option to create nested directories, ensuring that parent directories are created if they don't already exist.

6. rmdir(remove (delete) empty directories)

The rmdir command in Linux is used to remove (delete) empty directories from the file system. It is used to clean up directories that no longer contain any files or subdirectories.

Syntax

```
rmdir [OPTION] DIRECTORY
```

Options

- -p: Remove parent directories as well if they become empty after removing the specified directory.

Example:

1. Remove an empty directory named "myfolder":

```
$rmdir myfolder
```

2. Remove multiple empty directories "folder1", "folder2", and "folder3":

```
$rmdir folder1 folder2 folder3
```

3. Remove a nested empty directory structure "parent/child/grandchild":

```
$rmdir -p parent/child/grandchild
```

Note: The rmdir command is used only for removing empty directories. If a directory contains files or other subdirectories, the command will not work. To remove directories with content, you can use the rm command with appropriate options. Always exercise caution when deleting directories or files to avoid data loss.

Experiment No 3B.

Title : Linux Command

Aim: To study following Linux Commands – file, touch, rm, cp, rename, head, tail, cat, tac, more, less, , strings, chmod

Procedure

1. File(type of a file by analyzing its content)

The file command in Linux is used to determine the type of a file by analyzing its content, rather than relying solely on its extension. It uses a database of file signatures (magic numbers) to identify the file type. This makes it a powerful tool for identifying files, especially when the extension or file name is not reliable or when dealing with unknown files.

Syntax

```
file [options] [filename]
```

Options

- -b: Brief mode. Displays only the file type.
- -i: MIME type mode. Displays the MIME type of the file.
- -z: Compressed files mode. Identifies compressed files.
- -L: Follow symbolic links. Reports information about the target file for symbolic links.
- -k: Keep going mode. Continues processing even if an error occurs.

Examples

1. Display the file type of a single file:

```
$file myfile.txt
```

2. Display the MIME type of a single file:

```
$file -i myfile.png
```

3. Identify a compressed file:

```
$file -z archive.tar.gz
```

4. Follow symbolic links and display information about the target file:

```
$file -L mylink
```

5. Display the MIME type of all files in a directory:

```
$file -i *
```

6. Display brief information about all files in a directory:

```
$file -b *
```

7. Determine the file type of a remote file via URL:

```
$file -i https://example.com/document.pdf
```

8. Use with find command to identify files of a specific type:

```
$find /path/to/search -type f -exec file {} \; | grep "PDF document"
```

9. Determine the file type of a raw disk image:

```
$file -s /dev/sdb
```

10. Display information about a specific file using a custom magic file:

```
$file -m /path/to/custom.magic myfile
```

Note : for more details $man file can be used

2. Touch(creates new empty files)

The touch command creates new empty files if they don't exist and updates the timestamps of existing files. By default, it updates both the access (atime) and modification (mtime) timestamps to the current time. It's a convenient way to manipulate file timestamps without changing their content.

☞ Syntax

```
touch [OPTION]... FILE...
```

Options

- -a or --time=atime: Update only the access timestamp (atime) of the file.
- -m or --time=mtime: Update only the modification timestamp (mtime) of the file.
- -c or --no-create: Do not create the file if it doesn't exist.
- -d, --date=STRING: Parse STRING and use it instead of the current time.
- -r, --reference=FILE: Use the timestamps of FILE instead of the current time.
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Create an empty file:

```
$touch myfile.txt
```

2. Create multiple empty files at once:

```
$touch file1.txt file2.txt file3.txt
```

3. Update the modification timestamp of a file:

```
$touch -m myfile.txt
```

4. Update the access timestamp of a file:

```
$touch -a myfile.txt
```

5. Update both access and modification timestamps of a file:

```
$touch myfile.txt
```

6. Create a file with a specific timestamp using the -d option:

```
$touch -d "2023-08-17 15:30:00" myfile.txt
```

7. Update a file's timestamps to match another file's timestamps using the -r option:

```
$touch -r reference-file.txt myfile.txt
```

8. Prevent creating a file if it doesn't exist using the -c option:

```
$touch -c non_existent_file.txt
```

9. Create a file with a specific timestamp using a reference file's timestamp:

```
$touch -r reference-file.txt new-file.txt
```

10. Use touch with wildcard to update timestamps of multiple files:

```
$touch -m *.txt
```

Note : for more details $man touch can be used

3. rm(to delete files and directories)

The rm command is used to delete files and directories. When used without any options, it permanently removes files and directories from the file system. Be cautious when using this command, as deleted files cannot typically be easily recovered.

☞ Syntax

```
rm [OPTION]... FILE...
```

Options

- -i or --interactive: Prompt before every removal.

- -f or --force: Ignore non-existent files and do not prompt.
- -r or --recursive: Recursively remove directories and their contents.
- -v or --verbose: Explain what is being done.
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Remove a file:

```
$rm myfile.txt
```

2. Remove multiple files:

```
$rm file1.txt file2.txt file3.txt
```

3. Remove a file without prompting (forceful):

```
$rm -f myfile.txt
```

4. Prompt before removing each file:

```
$rm -i myfile.txt
```

6. Remove a directory and its contents:

```
$rm -r mydir
```

7. Remove a directory forcefully (with contents):

```
$rm -rf mydir
```

8. Remove all files with a specific extension:

```
$rm *.log
```

9. Remove files using a wildcard and prompt before each removal:

```
$rm -i *.txt
```

10. Remove a directory and its contents using verbose mode:

```
$rm -rv mydir
```

11. Remove a directory using a reference file (timestamp) and its contents:

```
$rm -r --reference=reference-file mydir
```

4. Cp(to create a copy of one or more files or directories)

The cp command is used to create a copy of one or more files or directories. It can be used to duplicate individual files or entire directory structures. It's a versatile command commonly used for backup, data migration, and general file manipulation.

☞ Syntax

```
cp [OPTION]... SOURCE... DESTINATION
```

Options

- -i or --interactive: Prompt before overwriting files.
- -r or --recursive: Copy directories recursively.
- -u or --update: Copy only when the SOURCE file is newer than the destination file or when the destination file is missing.
- -v or --verbose: Explain what is being done.
- -a or --archive: Preserve permissions, ownership, timestamps, and other attributes while copying (equivalent to -dR --preserve=all).
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Copy a file to a new location:

```
$cp myfile.txt /path/to/destination/
```

2. Copy multiple files to a directory:

```
$cp file1.txt file2.txt /path/to/destination/
```

3. Copy a file and prompt before overwriting:

```
$cp -i sourcefile.txt /path/to/destination/
```

4. Copy a directory and its contents recursively:

```
$cp -r sourcedir /path/to/destination/
```

5. Copy a directory with all attributes preserved (archive mode):

```
$cp -a sourcedir /path/to/destination/
```

6. Copy a file only if it's newer or the destination is missing:

```
$cp -u sourcefile.txt /path/to/destination/
```

7. Copy a file while displaying verbose information:

```
$cp -v sourcefile.txt /path/to/destination/
```

8. Copy a directory and its contents, preserving timestamps and ownership:

```
$cp -rp sourcedir /path/to/destination/
```

9. Copy a file with a new name in the same directory:

```
$cp oldfile.txt newfile.txt
```

10. Copy multiple files with a specific extension using a wildcard:

```
$cp *.log /path/to/destination/
```

5. Rename(rename files)

The rename command takes a Perl expression as an argument and uses it to rename files that match a specified pattern. It provides a powerful way to perform batch renaming of files using regular expressions.

☞ Syntax

```
rename [options] 's/old-pattern/new-pattern/' files
```

Options

- -n or --no-act: Dry-run mode. Show what would be renamed, but don't actually rename the files.
- -v or --verbose: Be verbose. Display detailed information about the renaming process.
- -f or --force: Force the renaming of files without prompting for confirmation.
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Rename files with a specific pattern:

```
$rename 's/old/new/' file1.txt file2.txt
```

2. Rename files with a specific extension in a directory:

```
$rename 's/.txt/.md/' *.txt
```

3. Dry-run to preview renaming without actually changing file names:

```
$rename -n 's/old/new/' file1.txt file2.txt
```

4. Use a regular expression to replace a portion of the file name:

```
$rename 's/(\d+)/file_$1/' 1.txt 2.txt 3.txt
```

5. Rename files in a directory and its subdirectories recursively:

```
$rename -v 's/old/new/' dir/**/*.txt
```

6. Force renaming of files without confirmation:

```
$rename -f 's/old/new/' file1.txt file2.txt
```

7. Replace spaces with underscores in filenames:

```
$rename 's/ /_/g' *.txt
```

8. Append a prefix to all files in a directory:

```
$rename 's/^/prefix_/' *.txt
```

9. Remove a specific prefix from filenames:

```
$rename 's/^prefix_//' prefix_*.txt
```

10. Replace characters using a more complex regular expression:

```
$rename 's/\.(jpg|jpeg)/.png/' *.jpg *.jpeg
```

6. Head

The head command is useful for quickly previewing the contents of a text file, especially when dealing with large files. It allows you to see the initial part of the file without having to load the entire content into memory.

☞ Syntax

```
head [OPTION]... [FILE]...
```

Options

Here are some commonly used options with the head command:

- -n N: Display the first N lines of the file. For example, -n 20 will display the first 20 lines.
- -c N: Display the first N bytes of the file, rather than lines.
- -q: Quiet mode. Display headers only when processing multiple

files.

- -v: Verbose mode. Display headers always when processing multiple files.
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Display the first 10 lines of a file:

```
$head myfile.txt
```

2. Display a specific number of lines (e.g., 5) from a file:

```
$head -n 5 myfile.txt
```

3. Display the first 20 bytes of a file:

```
$head -c 20 myfile.txt
```

4. Display the first 5 lines of multiple files:

```
$head -n 5 file1.txt file2.txt file3.txt
```

5. Display the first 10 lines of a file, but omit headers if processing multiple files:

```
$head -q myfile.txt otherfile.txt
```

6. Display the first 10 lines of a file, but always show headers if processing multiple files:

```
$head -v myfile.txt otherfile.txt
```

7. Display the first few lines of a file while preserving colors and formatting (using ANSI escape codes):

```
$head -n 10 -- myfile-with-colors.txt
```

8. Display the first 1000 lines of a log file:

```
$head -n 1000 mylog.log
```

9. Display the first few lines of a binary file (using -c option):

```
$head -c 128 binaryfile.bin
```

10. Display the first lines of multiple files and omit headers (using -q option):

```
$head -n 5 -q file1.txt file2.txt file3.txt
```

7. Tail

The tail command is useful for quickly checking the most recent content of a text file, especially when dealing with log files that are continuously updated. It allows you to see the end part of the file without having to load the entire content into memory.

☞ Syntax

```
tail [OPTION]... [FILE]...
```

Options

- -n N: Display the last N lines of the file. For example, -n 20 will display the last 20 lines.
- -c N: Display the last N bytes of the file, rather than lines.
- -f: Output appended data as the file grows (useful for monitoring log files).
- -q: Quiet mode. Suppress headers when displaying multiple files.
- -v: Verbose mode. Always display headers when displaying multiple files.
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Display the last 10 lines of a file:

```
$tail myfile.txt
```

2. Display a specific number of lines (e.g., 5) from the end of a file:

```
$tail -n 5 myfile.txt
```

3. Display the last 20 bytes of a file:

```
$tail -c 20 myfile.txt
```

4. Display the last 5 lines of multiple files:

```
$tail -n 5 file1.txt file2.txt file3.txt
```

5. Display the last 10 lines of a file and continuously monitor for new lines (log tailing):

```
$tail -f mylogfile.log
```

6. Display the last 100 bytes of a binary file:

```
$tail -c 100 binaryfile.bin
```

7. Display the last lines of a file while preserving colors and formatting (using ANSI escape codes):

```
$tail -n 10 -- myfile-with-colors.txt
```

8. Display the last lines of a log file and suppress headers for multiple files:

```
$tail -n 20 -q file1.log file2.log
```

9. Display the last lines of a file and always show headers for multiple files:

```
$tail -n 10 -v file1.txt file2.txt
```

10. Display the last few lines of multiple files (using -q option to suppress headers):

```
$tail -n 5 -q file1.txt file2.txt file3.txt
```

8. Cat(display the contents of a file directly in the terminal)

The cat command is often used to quickly display the contents of a file directly in the terminal. It can also be used to combine multiple files and display their combined content

☞ Syntax

```
cat [OPTION]... [FILE]...
```

Options

- -n or --number: Number all output lines, starting from 1.
- -b or --number-nonblank: Number nonempty output lines, starting from 1.
- -s or --squeeze-blank: Squeeze multiple adjacent blank lines into one.
- -A, --show-all: Display non-printing characters, such as tabs and line endings, using control characters (^I for tab, $ for line end).

- -T, --show-tabs: Display tabs using ^I (caret-I) and display line endings using $.
- -v, --show-nonprinting: Display non-printing characters as-is (octal or hexadecimal notation).
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Display the contents of a file:

```
$cat myfile.txt
```

2. Display the contents of multiple files:

```
$cat file1.txt file2.txt file3.txt
```

3. Display the contents of a file with line numbers:

```
$cat -n myfile.txt
```

4. Display the contents of a file, showing non-printing characters:

```
$cat -A myfile.txt
```

5. Combine multiple files and display their content:

```
$cat file1.txt file2.txt > combined.txt
```

6. Display the contents of a file with tabs and line endings visible:

```
$cat -T myfile.txt
```

7. Display the contents of a file, showing non-printing characters as hexadecimal:

```
$cat -v myfile.txt
```

8. Combine files and display them with line numbers:

```
$cat -n file1.txt file2.txt > combined_with_numbers.txt
```

9. Display the contents of a file, squeezing multiple blank lines into one:

```
$cat -s myfile.txt
```

10. Display the contents of multiple files, showing all non-printing characters:

```
$cat -A file1.txt file2.txt
```

9. Tac

The tac command is often used to quickly view the contents of a file in reverse order, especially when dealing with log files or other files where the most recent entries are at the bottom.

☞ Syntax

```
tac [OPTION]... [FILE]...
```

Options

- -b, --before: Output is separated with newline (default).
- -s SEP, --separator=SEP: Use SEP as the separator between lines (instead of newline).
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Display the contents of a file in reverse order:

```
$tac myfile.txt
```

2. Display the contents of multiple files in reverse order:

```
$tac file1.txt file2.txt file3.txt
```

3. Display the contents of a file in reverse order with a custom separator:

```
$tac -s " | " myfile.txt
```

4. Combine tac with other commands, like grep, to search for patterns in reverse:

```
$tac mylog.txt | grep "error"
```

5. Display the contents of a file with a different separator between lines:

```
$tac -s ", " myfile.txt
```

6. Display the contents of a file in reverse order and save it to a new file:

```
$tac original.txt > reversed.txt
```

7. Display the contents of a file in reverse order and show line numbers:

```
$tac myfile.txt | nl
```

8. Combine tac with head to display the last N lines of a file in their original order:

```
$tac myfile.txt | head -n N
```

10. More

The more command is commonly used to read and navigate through the contents of text files that are too large to be displayed in a single screen.

☞ Syntax

```
more [OPTIONS] [FILE]
```

Options

- -d: Display help information at the bottom of the screen.
- -f: Count logical rather than screen lines (useful for files with long lines).
- -l: Fold long lines instead of wrapping.
- -c: Do not scroll, clear the screen, or display line numbers when scrolling.
- -p: Preserve the file's original formating and use screen width to determine line breaks.
- -s: Squeeze multiple blank lines into one.
- -u: Disable underlining (useful for terminals that do not support underlining).
- -n N: Specify the number of lines to display per screen.
- +N: Start displaying from line N.
- -v: Display version information and exit.
- --help: Display help message and exit.

Examples

1. Display the contents of a file using more:

```
$more myfile.txt
```

2. Display the contents of a file and start from a specific line:

```
$more +20 myfile.txt
```

3. Display a long file with line numbers and help information at the bottom:

```
$more -d -n 20 longfile.txt
```

4. Display a file without scrolling, clearing, or line numbers:

```
$more -c myfile.txt
```

5. Display a file with long lines folded instead of wrapping:

```
$more -l myfile.txt
```

6. Display a file with multiple blank lines squeezed into one:

```
$more -s myfile.txt
```

7. Display a file while preserving its original formatting:

```
$more -p formatted.txt
```

8. Display a file without underlining:

```
$more -u plain.txt
```

9. Display a file with a specific number of lines per screen:

```
$more -n 30 largefile.txt
```

10. Display help information about the more command:

```
$more --help
```

11. Less

The less command is a text viewer that displays the contents of a file one screenful (page) at a time. It allows you to scroll forward and backward, search for text, and navigate through large files without loading the entire file into memory. less also supports syntax highlighting for certain file types, making it useful for viewing source code and configuration files.

Syntax

```
less [OPTIONS] [FILE]
```

Options

- -N, --LINE-NUMBERS: Display line numbers.
- -i, --IGNORE-CASE: Ignore case when searching.
- -S, --chop-long-lines: Truncate long lines instead of wrapping.

- -F, --quit-if-one-screen: Quit immediately if the file fits on one screen.
- -r, --RAW-CONTROL-CHARS: Display control characters.
- -X, --no-init: Do not clear the screen on exit.
- -h, --help: Display help message and exit.
- --version: Display version information and exit.

☞ Usage

Once inside less, you can use various keyboard shortcuts to navigate and interact with the text:

- Up Arrow, Down Arrow: Scroll one line up or down.
- Page Up, Page Down: Scroll one screenful up or down.
- Spacebar: Scroll one screenful down.
- B, b: Scroll one screenful backward.
- Q: Quit less.
- /text: Search for "text" forward in the file.
- ?text: Search for "text" backward in the file.
- n: Repeat the last search in the same direction.
- N: Repeat the last search in the opposite direction.
- G: Go to the end of the file.
- 1G or gg: Go to the beginning of the file.
- NUMG: Go to line number NUM.
- q: Quit less.

Examples

1. View the contents of a file named example.txt:

```
$less example.txt
```

2. View the contents of a file with line numbers:

```
$less -N example.txt
```

3. Search for a specific text in the file (use / followed by the text):

```
$/search_text
```

4. Scroll one screenful backward: Press b or B.

Quit less: Press q.

12. chmod(tool for managing access to files and directories)

The chmod command is a fundamental tool for managing access to files and directories in a Unix-like operating system. It enables you to specify the permissions using a symbolic representation (e.g., u+rwx, g-x) or an octal notation (e.g., 755, 644). Properly managing permissions is crucial for maintaining security and control over your files and ensuring that users have appropriate access levels.

Syntax

```
chmod [OPTION]... MODE FILE...
```

Options

- -c, --changes: Like verbose, but report only when a change is made.
- -f, --quiet, --silent: Suppress most error messages.
- -v, --verbose: Output a diagnostic for every file processed.
- --help: Display help message and exit.
- --version: Display version information and exit.

Modes

The MODE argument specifies the new permissions you want to apply. It can be specified in either symbolic notation or octal notation.

Symbolic Notation

Uses letters (u for user, g for group, o for others, a for all) and operators (+ to add permissions, - to remove permissions, = to set exact permissions).

Example

```
chmod u+rwx, g+rw, o-rx file.txt
```

☞ Octal Notation

Uses a 3-digit octal number to represent the permissions.

- The first digit represents the owner's permissions,
- the second digit represents the group's permissions, and
- the third digit represents others' permissions.
- Each digit is a sum of values: 4 for read, 2 for write, and 1 for execute.

Example

```
chmod 755 script.sh
```

Examples

1. Give read, write, and execute permissions to the owner, and only read and execute permissions to the group and others:

```
$chmod u+rwx,g+rx,o+rx myfile.txt
```

2. Set read and write permissions for the owner, and only read permissions for the group and others:

```
$chmod 644 data.txt
```

3. Remove execute permission from all users (owner, group, others):

```
$chmod a-x script.sh
```

4. Make a script executable by the owner and others:

```
$chmod u+x,o+x myscript.sh
```

5. Change permissions recursively for a directory and its contents:

```
$chmod -R u+rwX,g+rX,o-rwx mydir/
```

6. Change permissions on multiple files at once:

```
$chmod go-rx file1.txt file2.txt
```

7. Grant execute permission to everyone for a specific script:

```
$chmod a+x script.sh
```

8. Show verbose output while changing permissions:

```
chmod -v u+x,g+x,o+x myscript.sh
```

Experiment No 3C.

Title : Linux Command

Aim: To study following Linux Commands – ps, top, kill, pkill, bg, fg

☞ Procedure

1. ps(powerful tool for monitoring and managing processes)

The ps command is a powerful tool for monitoring and managing processes on a Linux system. It allows you to see a list of running processes and provides valuable information about their resource usage and status. The output of the ps command can be customized using various options to suit different monitoring and troubleshooting needs.

☞ Syntax

```
ps [options]
```

Options

- -A, --all: Show information about all processes except those associated with terminals.
- -a, --deselect: Show processes other than those running with the controlling terminal.
- -u, --user: Display processes owned by a specific user.
- -x: Include processes without a controlling terminal (typically background processes).
- -e: Display information about all processes.
- -f: Display full-format listing.
- -l: Long format listing.
- -o: Define the output format using custom keywords.
- -p: Display information about specific process IDs.
- -h: Suppress the display of column headers.
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Display information about all processes currently running:

```
$ps
```

2. Display detailed information about all processes in long format:

```
$ps -l
```

3. Show information about all processes, including those without a controlling terminal:

```
$ps -x
```

4. Display processes owned by a specific user:

```
ps -u username
```

5. Show detailed information about a specific process using its PID:

```
$ps -p 1234
```

6. Display a customized output format, showing only specific columns:

```
$ps -o pid,ppid,user,%cpu,command
```

6. Display information about all processes, including those without a controlling terminal, in full format:

```
$ps -ef
```

7. Show information about all processes except those associated with terminals:

```
$ps -A
```

8. Display a list of all processes owned by the current user:

```
$ps -u $USER
```

9. Display information about a specific process and its children (tree view):

```
$ps -o pid,ppid,cmd --forest
```

2. Top(real-time display of system activity)

The top command is a powerful utility that provides an ongoing, real-time display of system activity. It updates its display periodically, allowing you to see how system resources are being utilized and which processes are consuming the most CPU and memory. The top command is particularly useful for diagnosing performance issues and identifying processes that might be causing system slowdowns.

Syntax

```
top [options]
```

Options

- -d <delay>: Specifies the delay between updates (in seconds).
- -n <iterations>: Limits the number of updates to the specified number of iterations.
- -p <pid1,pid2,...>: Monitor specific process IDs.
- -u <username>: Display processes owned by a specific user.
- -C <command>: Monitor processes with a specific command name.
- -i: Toggle the display of idle processes.
- -H: Toggle the display of individual threads.
- -b: Run in batch mode (useful for scripting).
- -c: Show the full command path instead of just the command name.
- -M: Display memory usage in megabytes.
- -k: Toggle display of kilobytes instead of 4K pages.
- q: Quit top.

Interactive Commands

When running top, you can use various interactive commands to perform actions or change the display. Some common commands include:

- h: Display help (show a list of available interactive commands).
- k: Send a signal (terminate) to a selected process.
- u: Filter processes by a specific user.
- r: Renice (change the priority) of a selected process.

- 1: Toggle the display of CPU usage per core.
- s: Change the update interval (in seconds).

Examples:

1. Launch top and view real-time system statistics:

```
$top
```

2. Monitor processes owned by a specific user:

```
$top -u username
```

3. Display memory usage in megabytes:

```
$top -M
```

4. Monitor a specific process by its process ID:

```
$top -p 1234
```

5. Display only a specific number of iterations:

```
$top -n 5
```

6. Display processes using a specific command name:

```
$top -C nginx
```

7. Run top in batch mode (useful for scripting):

```
$top -b -n 1
```

8. Filter for threads (individual tasks within processes):

```
$top -H
```

9. Display a list of interactive commands:

```
$top -h
```

3. Kill(tool for managing processes)

The kill command is an essential tool for managing processes on a Linux system. It allows you to interact with processes by sending various signals to them. The most common signal sent with kill is the default signal, which terminates a process. However, other signals can be used to request different actions, such as restarting a process or instructing it to gracefully shut down.

☞ Syntax

```
kill [OPTIONS] <PID>...
```

Options

- -s <signal>: Specify the signal to send (e.g., -s SIGTERM).
- -l, --list: List available signals.
- -a, --all: All processes except session leaders and processes not associated with a terminal.
- -p, --pid: Specify process IDs directly.
- -n, --newest: Send the signal to the newest process only.
- -o, --older: Send the signal to the oldest process only.
- --help: Display help message and exit.
- --version: Display version information and exit.

Common Signals

- SIGTERM (15): Terminate the process (default signal if none is specified).
- SIGKILL (9): Forcefully terminate the process (cannot be ignored or caught).
- SIGINT (2): Interrupt signal (usually sent by pressing Ctrl+C).
- SIGHUP (1): Hangup signal (often used to restart or reload processes).
- SIGSTOP (19): Stop the process (pauses execution).
- SIGCONT (18): Resume a stopped process.

Examples

1. Terminate a process with a specific PID using the default signal (SIGTERM):

```
$kill 1234
```

2. Terminate multiple processes by specifying their PIDs:

```
$kill 1234 5678 9012
```

3. Terminate a process with a specific PID using a specific signal (e.g., SIGKILL):

```
$kill -s SIGKILL 1234
```

4. Use a signal number instead of its name (e.g., SIGTERM is signal 15):

```
$kill -s 15 1234
```

5. Send an interrupt signal to a process (e.g., simulate pressing Ctrl+C):

```
$kill -s SIGINT 1234
```

6. Send a hangup signal to a process (often used to reload configuration):

```
$kill -s SIGHUP 1234
```

7. Display a list of available signals:

```
$kill -l
```

8. Terminate a group of processes by specifying the process group ID (PGID):

```
$kill -- -1234
```

9. Terminate all processes with a specific command name (e.g., "myprocess"):

```
$pkill myprocess
```

10. Terminate all processes owned by a specific user (e.g., "username"):

```
$pkill -u username
```

4. pkill(tool for terminating or signaling processes based on their names or attributes)

The pkill command is a useful tool for terminating or signaling processes based on their names or attributes. It provides a convenient way to manage processes without needing to know their exact process IDs (PIDs). It is often used to automate the termination of processes or to perform other actions on multiple processes that match a certain criteria.

☞ Syntax

```
pkill [OPTIONS] <pattern>
```

Options

- -u <username>: Match processes owned by a specific user.

- -n: Match the newest (most recently started) process.
- -o: Match the oldest (least recently started) process.
- -g: Match processes in the specified process group ID (PGID).
- -P: Match processes whose parent process ID (PPID) matches the specified PID.
- -t: Match processes associated with the specified terminal.
- -f: Match against the entire command line, not just the process name.
- -x: Match whole words (exact match) of the process name.
- -s <signal>: Send a specific signal (default is SIGTERM).
- --list: List the PIDs of processes that match the criteria, without sending signals.
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Terminate all processes with a specific name (e.g., "myprocess"):

```
pkill myprocess
```

2. Terminate all processes with a specific name using a specific signal (e.g., SIGKILL):

```
pkill -9 myprocess
```

3. Terminate processes owned by a specific user (e.g., "username"):

```
pkill -u username
```

4. Terminate the newest (most recently started) process with a specific name:

```
pkill -n myprocess
```

5. Terminate processes associated with a specific terminal:

```
pkill -t pts/0
```

6. Terminate processes with a specific command line (full process name):

```
pkill -f "myprocess --option"
```

7. List the PIDs of processes that match the criteria without sending signals:

```
pkill --list myprocess
```

8. Terminate processes in a specific process group (PGID):

```
pkill -g 12345
```

9. Terminate processes whose parent process ID (PPID) matches a specific PID:

```
pkill -P 6789
```

10. Terminate processes with an exact (whole word) match of the process name:

```
pkill -x myprocess
```

5. bg(suspended process to the background)

When a process is running in the foreground (i.e., it's actively using the terminal), you can suspend it by pressing Ctrl+Z. This will pause the process and return control of the terminal to you. The bg command then allows you to move the suspended process to the background, freeing up the terminal for other commands while still allowing the process to continue running.

☞ Syntax

```
bg [JOB_SPEC]
```

Options

- JOB_SPEC: Specifies the job or process to be moved to the background. It can be specified using its job number (preceded by %) or its process ID (PID).

Example:

1. Suspend a process by pressing Ctrl+Z (e.g., a long-running command):

```
$long_running_command
^Z
```

2. Move the suspended process to the background using bg:

```
$bg
```

3. Suspend a process and then move it to the background using a job number:

```
$long_running_command
^Z
bg %1
```

4. Suspend a process, move it to the background, and then resume it in the background using a job number:

```
$long_running_command
^Z
bg %1
disown %1
```

5. Move a specific suspended process to the background using its PID:

```
$bg 1234
```

6. List the jobs in the current session to see their statuses:

```
$jobs
```

6. fg(bring a suspended process back to the foreground)

When a process is running in the foreground (i.e., it's actively using the terminal), you can suspend it by pressing Ctrl+Z. This will pause the process and return control of the terminal to you. The fg command allows you to bring a suspended process back to the foreground, where you can interact with it directly.

☞ Syntax

```
fg [JOB_SPEC]
```

Options

- JOB_SPEC: Specifies the job or process to be brought to the foreground. It can be specified using its job number (preceded by %) or its process ID (PID).

Example

1. Suspend a process by pressing Ctrl+Z (e.g., a long-running command):

```
long_running_command
^Z
```

2. Bring the suspended process back to the foreground using fg:

```
fg
```

3. Suspend a process and then bring it back to the foreground using a job number:

```
long_running_command
^Z
fg %1
```

4. Suspend a process, bring it back to the foreground, and then resume it in the foreground using a job number:

```
long_running_command
^Z
fg %1
```

5. Bring a specific suspended process back to the foreground using its PID:

```
fg 1234
```

6. List the jobs in the current session to see their statuses:

```
jobs
```

▶ Experiment No 3D.

Title : Linux Command

Aim: To study following Linux Commands – grep, locate, find

☞ Procedure

1. grep(utility for searching and filtering text in files or output streams)

The grep command is an essential utility for searching and filtering text in files or output streams. It uses regular expressions to match patterns and then displays the lines containing the matched pattern. grep is widely used for tasks like searching log files, extracting specific information from files, and performing text processing.

☞ Syntax

```
grep [OPTIONS] PATTERN [FILE...]
```

Options

- -i: Ignore case (perform case-insensitive matching).
- -v: Invert the match, displaying lines that do not match the pattern.
- -c: Count the number of matching lines instead of displaying them.
- -l: Display only the names of files containing the pattern (not the matched lines).
- -r or -R: Recursively search directories for files matching the pattern.
- -n: Show line numbers for matched lines.
- -w: Match whole words only (exact match).
- -A N: Display N lines after each matching line.
- -B N: Display N lines before each matching line.
- -C N: Display N lines before and after each matching line.
- --color: Highlight matching text (enabled by default on most systems).
- --exclude: Exclude specific files or patterns from the search.
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Search for a specific pattern in a file:

```
$grep "search_term" file.txt
```

2. Search for a pattern in multiple files:

```
$grep "pattern" file1.txt file2.txt
```

3. Perform a case-insensitive search:

```
$grep -i "case_insensitive" file.txt
```

4. Display only the filenames containing the pattern (not the lines):

```
$grep -l "pattern" *.txt
```

5. Search recursively in directories for files containing a pattern:

```
$grep -r "pattern" /path/to/directory
```

6. Show line numbers for matching lines:

```
$grep -n "pattern" file.txt
```

7. Display lines containing "word" as a whole word:

```
$grep -w "word" file.txt
```

8. Display lines before and after each matching line:

```
$grep -C 2 "pattern" file.txt
```

9. Exclude files matching a certain pattern from the search:

```
$grep "pattern" . --exclude="*.log"
```

10. Search for multiple patterns using alternation (|):

```
$grep "pattern1\|pattern2" file.txt
```

2. locate (pre-generated database (index) of file and directory names on your system)

The locate command uses a pre-generated database (index) of file and directory names on your system. This database allows locate to perform searches very quickly, but it might not include recently added or modified files until the database is updated.

☞ Syntax

```
locate [OPTIONS] PATTERN
```

Options

- -i: Perform a case-insensitive search.
- -c: Display the count of matching entries.
- -l: Show file names without path information.
- -e: Print only entries that actually exist on the filesystem.
- -r: Treat the pattern as a regular expression.
- -q: Quiet mode (no output for nonexistent files).
- -n: Limit the number of results displayed.
- --version: Display version information and exit.
- --help: Display help message and exit.

Examples

1. Search for a file by name:

```
$locate myfile.txt
```

2. Perform a case-insensitive search:

```
$locate -i MyFile.TXT
```

3. Display the count of matching entries:

```
$locate -c pattern
```

4. Show file names without path information:

```
$locate -l myfile
```

5. Use a regular expression for the search pattern:

```
$locate -r "^myfile.*\.txt$"
```

6. Limit the number of results displayed:

```
$locate -n 5 file
```

7. Only show entries that actually exist on the filesystem:

```
$locate -e myfile
```

8. Perform a quiet search (no output for nonexistent files):

```
$locate -q missingfile
```

3. find(essential tool for searching and locating files and directories within a specified directory and its subdirectories)

The find command is an essential tool for searching and locating files and directories within a specified directory and its subdirectories. It can search based on a wide range of criteria, making it highly flexible for various use cases, including cleaning up files, organizing data, and managing system resources.

☞ Syntax

```
find [path...] [expression]
```

Options

- -name <pattern>: Search for files matching the specified name pattern (supports wildcards).

- -type <type>: Search for files of the specified type (f for regular files, d for directories, l for symbolic links, etc.).
- -size <size>: Search for files with a specific size (e.g., -size +1M for files larger than 1 megabyte).
- -user <username>: Search for files owned by a specific user.
- -group <groupname>: Search for files belonging to a specific group.
- -mtime <days>: Search for files modified within a specific number of days.
- -atime <days>: Search for files accessed within a specific number of days.
- -ctime <days>: Search for files created within a specific number of days.
- -exec <command>: Execute a command on each found file.
- -print: Display the path of each found file (default action).
- -delete: Delete found files and directories (use with caution).
- -maxdepth <level>: Set the maximum depth of subdirectories to search.

-mindepth <level>: Set the minimum depth of subdirectories to search.

- -iname <pattern>: Perform a case-insensitive search for file names.
- --help: Display help message and exit.
- --version: Display version information and exit.

Expression

The expression consists of various options, tests, and actions that determine the search criteria and what to do with the found files. It is a combination of predicates (tests) and actions, and it is usually placed after the path.

Examples

1. Search for a file named "myfile.txt" in the current directory and its subdirectories:

```
$find . -name "myfile.txt"
```

2. Search for directories larger than 100 megabytes:

```
$find /path/to/search -type d -size +100M
```

3. Search for files owned by a specific user:

```
$find /path/to/search -user username
```

4. Search for files modified within the last 7 days:

```
$find /path/to/search -mtime -7
```

5. Delete all files with the ".tmp" extension in the current directory and its subdirectories:

```
$find . -name "*.tmp" -delete
```

6. Execute a command on each found file (replace "echo" with your desired command):

```
$find /path/to/search -type f -exec echo {} \;
```

7. Search for files with names containing "document" (case-insensitive):

```
$find /path/to/search -iname "*document*"
```

8. Search for symbolic links in the user's home directory:

```
$find ~ -type l
```

9. Search for empty directories and delete them:

```
$find /path/to/search -type d -empty -delete
```

10. Search for files modified within the last 30 days and copy them to another directory:

```
$find /path/to/search -type f -mtime -30 -exec cp {} /destination/path \;
```

▸ Experiment No 3E.

Title : Linux Command

Aim: To study following Linux Commands – date, cal, uptime, who, whoami, finger, uname, df, du, free, whereis, which

☞ Procedure

1. date (interact with the system's date and time settings)

The date command allows you to interact with the system's date and time settings. It can display the current date and time, format date and time values, calculate time intervals, and set the system's date and time if you have appropriate permissions.

☞ Syntax

```
date [OPTION]... [+FORMAT]
```

Options

- -d <STRING>: Display date and time from a specified string.
- -u, --utc, --universal: Display or set in Coordinated Universal Time (UTC).
- -R, --rfc-2822: Display date and time in RFC 2822 format.
- -I, --iso-8601: Display date and time in ISO 8601 format.
- +%FORMAT: Specify a custom output format using format codes.
- --help: Display help message and exit.
- --version: Display version information and exit.

Format Codes

When using the date command, you can use various format codes to customize the output. Some common format codes include:

- %Y: Year (4-digit).
- %m: Month (01-12).
- %d: Day of the month (01-31).
- %H: Hour (00-23).
- %M: Minute (00-59).
- %S: Second (00-59).
- %a: Abbreviated day of the week (Sun, Mon, ...).
- %A: Full day of the week (Sunday, Monday, ...).
- %b: Abbreviated month name (Jan, Feb, ...).
- %B: Full month name (January, February, ...).
- %Z: Timezone name.

Examples

1. Display the current date and time:

```
$date
```

2. Display the current date in a specific format:

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

3. Display the current time in 12-hour format:

```
$date +"%I:%M %p"
```

4. Display the date and time in UTC (Coordinated Universal Time):

```
$date -u
```

5. Display the date and time in RFC 2822 format:

```
$date -R
```

6. Display the date and time in ISO 8601 format:

```
$date -I
```

7. Calculate the date and time 7 days from now:

```
$date -d "+7 days"
```

8. Calculate the date and time 1 hour ago:

```
$date -d "1 hour ago"
```

9. Set the system date and time (requires root privileges):

```
$sudo date -s "2023-08-17 12:00:00"
```

10. Display the day of the week and timezone:

```
$date +"%A, %Z"
```

3. cal(calendar on the terminal)

The cal command displays a calendar on the terminal, showing the days of the week (Sunday, Monday, etc.) and the dates for a specific month and year. By default, it displays the current month. The cal command is useful for quickly referencing dates, checking days of the week, and planning events.

☞ Syntax

```
cal [options] [month] [year]
```

Options

- -1: Display the calendar as a single month.
- -3: Display the calendar for the current month and the previous and next months.
- -y: Display a calendar for the entire year.
- -m: Display Monday as the first day of the week (default is Sunday).
- --help: Display help message and exit.
- --version: Display version information and exit.

Examples

1. Display the calendar for the current month (default):

```
$cal
```

2. Display the calendar for a specific month and year:

```
$cal 8 2023
```

3. Display a calendar for the entire year:

```
$cal -y 2023
```

4. Display the calendar for a specific year with Monday as the first day of the week:

```
$cal -y -m 2023
```

5. Display the calendar for the current month and the previous and next months:

```
$cal -3
```

6. Display a calendar with Monday as the first day of the week and show the previous, current, and next months:

```
$cal -3 -m
```

4. uptime(provides information about how long the system has been running)

The uptime command provides information about how long the system has been running since it was last booted, as well as the average load on the system over the past 1, 5, and 15 minutes. The load average indicates the number of processes that are either in a runnable or uninterruptable state. The uptime command is often used to quickly assess the current state of the system and determine if the system is under heavy load.

☞ Syntax

```
uptime [OPTIONS]
```

Options

- -p: Display the uptime in a more human-readable format.
- --help: Display help message and exit.
- --version: Display version information and exit.

Example

1. Display the current system uptime and load averages:

```
$uptime
```

2. Display the system uptime in a more human-readable format:

```
$uptime -p
```

Output Example

The output of the uptime command typically looks like this:

```
14:36:51 up 2 days, 6:12, 3 users, load average: 0.25, 0.30, 0.35
```

In this example, the system has been running for 2 days and 6 hours and 12 minutes. There are currently 3 users logged in, and the load averages over the past 1, 5, and 15 minutes are 0.25, 0.30, and 0.35, respectively.

5. who(a list of users who are currently logged)

The who command displays a list of users who are currently logged in to the system. It provides information about each user session, including the username, terminal, login time, and remote IP address (if the user is connected remotely). The command is useful for system administrators to monitor user activity and manage active sessions.

☞ Syntax

```
who [OPTIONS]
```

Options

- -a: Same as who -b (last system reboot time).
- -b: Display the time when the system was last rebooted.
- -H: Show column headers.
- -q: Display only the count of logged-in users.
- -r: Display the current runlevel.
- -s: Display information about the system shutdown time.
- --help: Display help message and exit.
- --version: Display version information and exit.

Example

1. Display a list of currently logged-in users:

```
$who
```

2. Display a list of logged-in users along with their terminal and login times:

```
$who -H
```

3. Display only the count of logged-in users:

```
$who -q
```

4. Display the time when the system was last rebooted:

```
$who -b
```

5. Display the current runlevel (system state):

```
$who -r
```

6. Display information about the system shutdown time:

```
$who -s
```

Output Example

The output of the who command typically looks like this:

```
username1 pts/0 2023-08-17 09:00 (:0)
username2 pts/1 2023-08-17 10:30 (:1)
```

In this example, two users (username1 and username2) are logged in. They are connected via pseudo-terminals (pts/0 and pts/1) and have login times at 09:00 and 10:30, respectively.

6. whoami (returns user ID (EUID) of the user who is executing the command)

The whoami command returns the effective user ID (EUID) of the user who is executing the command. It is often used in shell scripts or command-line operations to obtain the current username without needing to explicitly specify it. This can be helpful for various tasks, such as conditional operations or generating user-specific file paths.

☞ Syntax

```
whoami [OPTIONS]
```

Options

The whoami command does not have any commonly used options. It typically provides the current username without additional options.

Example

1. Display the current username:

```
$whoami
```

2. Use whoami in a shell script to perform a user-specific operation:

```
#!/bin/bash
username=$(whoami)
echo "Hello, $username!"
```

3. Check if the current user is the root user (superuser):

```
if [ "$(whoami)" = "root" ];
then
echo "You are the superuser."
else
echo "You are not the superuser."
fi
```

4. Use whoami to generate a user-specific file path:

```
filename="$(whoami)_file.txt"
echo "This is a file for user $filename."
```

Output Example

The output of the whoami command is simply the current username, such as:

```
ManojKavedia
```

In this example, the whoami command returns the username "ManojKavedia" of the user who is executing the command.

- finger (command retrieves and displays information from the user information database)

The finger command retrieves and displays information from the user information database (usually stored in the /etc/passwd file) about one or more users. It provides details about the specified users, including their real name, login shell, home directory, terminal, and login time. The finger command is often used by system administrators to monitor user activity and obtain information about users on the system.

☞ Syntax

```
finger [options] [username] [...]
```

Options

- -l: Display a long format with additional information.
- -s: Display a short format (default).
- -m: Display machine-readable output.
- -p: Display the user's plan file (usually stored in the user's home directory as .plan).
- -h <hostname>: Specify a remote host to query for user information.
- -q <query>: Search for a user using a keyword.
- -F <file>: Read usernames from a file.
- --help: Display help message and exit.
- --version: Display version information and exit.

Example

1. Display information about a specific user:

```
$finger manojkavedia
```

2. Display information about multiple users:

```
$finger vimla manoj kaushal rishabh asha
```

3. Display detailed information about a user's plan (usually stored in .plan file):

```
$finger -p manoj
```

4. Display user information in a long format:

```
$finger -l manoj
```

5. Display user information from a remote host:

```
$finger -h remote-host asha
```

6. Search for users using a keyword (e.g., "admin"):

```
$finger -q admin
```

7. Read usernames from a file and display their information:

```
$finger -F users.txt
```

Output Example

The output of the finger command typically looks like this:

```
Login: manoj              Name: manoj kavedia
Directory: /home/manoj        Shell: /bin/bash
On since Mon Aug 16 09:30 (UTC) on pts/0 from 192.168.1.10
1 day 2 hours idle
No mail.
No Plan.
```

In this example, the finger command displays detailed information about the user "manoj," including their full name, home directory, login shell, terminal, login time, and idle time.

7. uname (retrieves and displays information about the system and the kernel)

The uname command retrieves and displays information about the system and the kernel. It can provide details about the operating system name, kernel version, machine hardware name, processor type, and more. The command is useful for identifying the system's characteristics and for scripting purposes, where certain actions may need to be taken based on the system's configuration.

☞ Syntax

```
uname [OPTIONS]
```

Options

- -a or --all: Display all available information (default if no options are specified).
- -s or --kernel-name: Display the kernel name.
- -n or --nodename: Display the network node (hostname).
- -r or --kernel-release: Display the kernel release.
- -v or --kernel-version: Display the kernel version.
- -m or --machine: Display the machine hardware name.
- -p or --processor: Display the processor type (hardware platform).
- -i or --hardware-platform: Display the hardware platform.
- -o or --operating-system: Display the operating system name.
- --help: Display help message and exit.
- --version: Display version information and exit.

Example

1. Display basic system information (default behavior):

```
$uname
```

2. Display all available information:

```
$uname -a
```

3. Display the kernel name:

```
$uname -s
```

4. Display the hostname (network node name):

```
$uname -n
```

5. Display the kernel release:

```
$uname -r
```

6. Display the kernel version:

```
$uname -v
```

7. Display the machine hardware name (system architecture):

```
$uname -m
```

8. Display the processor type (hardware platform):

```
$uname -p
```

9. Display the operating system name:

```
$uname -o
```

Output Example

The output of the uname command typically looks like this:

```
Linux myhost 5.13.0-27-generic #29~20.04.1-Ubuntu SMP Fri Jan 14
00:32:30 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
```

In this example, the uname command displays various system information, including the kernel version, hostname, machine hardware name, processor type, and operating system name.

8. df (disk free)

The df command stands for "disk free" and is used to provide an overview of the storage capacity and usage of mounted filesystems on the system. It helps users and administrators understand how much space is available on each filesystem and how much space is currently being used. This information is crucial for managing and optimizing disk usage, especially on systems with limited storage resources.

Syntax

```
df [OPTIONS] [FILESYSTEM...]
```

Options

Here

- -h, --human-readable: Display sizes in human-readable format (e.g., KB, MB, GB).
- -H, --si: Like -h, but use powers of 1000 instead of 1024.
- -T, --print-type: Print the filesystem type along with disk space information.

- -t <TYPE>, --type=<TYPE>: Limit the display to filesystems of the specified type.
- -a, --all: Include all filesystems, including those with zero blocks used.
- --help: Display help message and exit.
- --version: Display version information and exit.

Example

1. Display disk usage information for all mounted filesystems:

$df

2. Display disk usage information in a human-readable format:

```
$df -h
```

3. Display disk usage information in powers of 1000 (SI units):

```
$df -H
```

4. Display disk usage information with filesystem type:

```
$df -T
```

5. Display disk usage information for a specific filesystem (e.g., root filesystem):

```
$df /
```

6. Display disk usage information for specific filesystem types (e.g., ext4):

```
$df -t ext4
```

7. Display disk usage information for all filesystems, including those with zero blocks used:

```
$df -a
```

Output Example

The output of the df command typically looks like this:

```
Filesystem          1K-blocks  Used          Available    Use%       Mounted on
/dev/sda1           10321208        3512956     6284500     36%      / tmpfs
1034544 4 1034540 1% /dev/shm /dev/sdb1       20642420 7654320
11912760      40%          /data
```

In this example, the df command displays information about three filesystems (/dev/sda1, tmpfs, and /dev/sdb1). It shows the total block size, used space, available space, usage percentage, and the mount point for each filesystem.

9. du (disk usage)

The du command stands for "disk usage" and is used to calculate and display the size of files and directories in terms of disk blocks. By default, it shows the total space used by each specified file or directory and their subdirectories. The du command is particularly useful for finding out how much space specific files or directories consume and for identifying areas of high disk usage.

☞ **Syntax**

```
du [OPTIONS] [FILE...]
```

Options

- -h, --human-readable: Display sizes in a human-readable format (e.g., KB, MB, GB).
- -H, --si: Like -h, but use powers of 1000 instead of 1024.
- -c, --total: Display a total summary of disk usage at the end.
- -s, --summarize: Display only the total disk usage for each specified file or directory.
- -d <N>, --max-depth=<N>: Limit the depth of directory traversal to <N> levels.
- -x, --one-file-system: Exclude directories on different filesystems.
- --exclude=<PATTERN>: Exclude files or directories that match the specified pattern.
- --help: Display help message and exit.
- --version: Display version information and exit.

Example

1. Display disk usage for a specific directory (default is current directory):

```
$du /path/to/directory
```

2. Display disk usage in a human-readable format:

```
$du -h /path/to/directory
```

3. Display disk usage of a specific file or directory and its subdirectories:

```
$du -h /path/to/file_or_directory
```

4. Display total disk usage of multiple files or directories and show a summary at the end:

```
$du -ch /path/to/file1 /path/to/file2
```

5. Display total disk usage for each specified file or directory without showing subdirectory details:

```
$du -s /path/to/file_or_directory
```

6. Limit the depth of directory traversal to 2 levels:

```
$du -h -d 2 /path/to/directory
```

7. Exclude specific files or directories using a pattern:

```
$du -h --exclude=*.log /path/to/directory
```

8. Display disk usage for a specific filesystem, excluding other filesystems:

```
$du -h -x /
```

Output Example

The output of the du command typically looks like this:

- 4.0K /path/to/directory/file1.txt
- 8.0K /path/to/directory/subdirectory
- 12K /path/to/directory

In this example, the du command displays the disk usage for each file and directory within the specified directory. The -h option is used to show sizes in a human-readable format.

10. free (provides an overview of the system's memory usage and availability)

The free command provides an overview of the system's memory usage and availability. It displays information about the physical RAM and swap space, including how much memory is currently in use, how much is free, and how much is used for caching and buffering. The free command helps users and administrators assess the memory status of the system and identify if there is a need for memory optimization.

☞ Syntax

```
free [OPTIONS]
```

Options

- -b, --bytes: Display memory sizes in bytes.
- -k, --kilo: Display memory sizes in kilobytes (default).
- -m, --mega: Display memory sizes in megabytes.
- -g, --giga: Display memory sizes in gigabytes.
- -h, --human: Display memory sizes in a human-readable format (e.g., KB, MB, GB).
- -t, --total: Display a total summary at the end.
- -s <DELAY>, --seconds=<DELAY>: Update the display every <DELAY> seconds.
- -c <COUNT>, --count=<COUNT>: Display <COUNT> iterations before exiting.
- --help: Display help message and exit.
- --version: Display version information and exit.

Example

1. Display memory usage information in kilobytes:

```
$free -k
```

2. Display memory usage information in megabytes:

```
$free -m
```

3. Display memory usage information in a human-readable format:

```
$free -h
```

4. Display memory usage information with a total summary at the end:

```
$free -t
```

5. Display memory usage information every 2 seconds (updating):

```
$free -s 2
```

6. Display memory usage information for a specific number of iterations (e.g., 5):

```
$free -c 5
```

Output Example

The output of the free command typically looks like this:

```
Total          used           free         shared    buff/cache available
Mem:     2049608       563124      932124  4196        519360     1304432
Swap:    2097148       2468        2094680
```

In this example, the free command displays information about both physical memory (RAM) and swap space usage. It shows the total, used, free, shared, buffered/cache, and available memory amounts for both memory types.

10. whereis (locate important files related to a specific command or program)

The whereis command helps you locate important files related to a specific command or program. It searches for the binary executable, source code files, and manual pages associated with the specified command. The whereis command is particularly useful when you want to know where a particular command is located in the system's directory structure.

Syntax

```
whereis [OPTIONS] COMMAND
```

Options

- -b: Search only for the binary executable.
- -s: Search only for the source files.
- -m: Search only for the manual pages.
- -u: Search only for the location of files that are not up-to-date.
- -B <DIRECTORIES>: Specify custom directories to search for binaries.
- -M <DIRECTORIES>: Specify custom directories to search for manual pages.
- -S <DIRECTORIES>: Specify custom directories to search for

source files.

- --help: Display help message and exit.
- --version: Display version information and exit.

Example

1. Locate the binary executable of a command (e.g., ls):

```
$whereis ls
```

2. Locate the manual pages associated with a command (e.g., grep):

```
$whereis -m grep
```

3. Locate the source files associated with a command (e.g., gcc):

```
$whereis -s gcc
```

4. Search for the binary executable, source files, and manual pages of a command (e.g., vim):

```
$whereis vim
```

5. Specify custom directories to search for binaries, manual pages, and source files:

```
$whereis -B /usr/local/bin:/opt/bin -M /usr/local/man:/opt/man -S
/usr/src:/opt/src gcc
```

Output Example

The output of the whereis command typically looks like this:

```
ls: /bin/ls /usr/share/man/man1/ls.1.gz
```

In this example, the whereis command displays the locations of the binary executable (/bin/ls) and the manual page (/usr/share/man/man1/ls.1.gz) for the ls command.

11. which(find the location of the executable file for a given command)

The which command helps you find the location of the executable file for a given command. It searches for the command in the directories listed in the system's PATH environment variable and displays the full path to the executable file that will be executed when you run the command. This can be helpful for troubleshooting, verifying that the correct executable is being used, and understanding how commands are resolved in the system.

☞ Syntax

```
which [OPTIONS] COMMAND
```

Options

- -a, --all: Display all matching executable paths (if there are multiple).
- -s, --skip-dot: Skip directories that start with a dot (hidden directories).
- --help: Display help message and exit.
- --version: Display version information and exit.

Example

1. Find the location of the executable for a command (e.g., ls):

```
$which ls
```

2. Find the location of the executable for multiple commands (e.g., ls and grep):

```
$which ls grep
```

3. Display all matching executable paths for a command (if there are multiple):

```
$which -a python
```

4. Skip hidden directories when searching for executables:

```
$which -s ls
```

Output Example

The output of the which command typically looks like this:

```
/usr/bin/ls
```

In this example, the which command displays the full path to the executable file for the ls command, which is /usr/bin/ls.

▶ Experiment No 3F.

Title : Linux Command

Aim: To study following Linux Compression Commands –tar, gzip

Procedure

Compression Commands

1. tar (create compressed or uncompressed archive files)

tar is used to create compressed or uncompressed archive files (tarballs) containing one or more files and directories. Although tar itself does not compress files, it is often combined with other compression tools, such as gzip or bzip2, to create compressed archive files.

Syntax

```
tar [OPTIONS] [ARCHIVE_FILENAME] [FILES_OR_DIRECTORIES]
```

Example

```
tar -czvf mydata.tar.gz mydata/
```

2. bzip2

bzip2 is another compression utility that uses the Burrows-Wheeler block sorting text compression algorithm. It typically produces smaller compressed files compared to gzip but may be slower.

Syntax

```
bzip2 [OPTIONS] [FILE]
```

Example

Compress a file named example.txt using bzip2:

```
bzip2 example.txt
```

3. bunzip2

bunzip2 is used to decompress files compressed with bzip2. It restores the original file by decompressing the contents of a ".bz2" file.

Syntax

```
bunzip2 [OPTIONS] [FILE.bz2]
```

Example

Decompress a file named example.txt.bz2 using bunzip2:

```
bunzip2 example.txt.bz2
```

4. gzip

gzip is a widely used compression utility in Linux that uses the DEFLATE compression algorithm. It compresses a single file and replaces the original file with the compressed version, appending a ".gz" extension to the filename.

☞ Syntax

```
gzip [OPTIONS] [FILE]
```

Example

Compress a file named example.txt using gzip:

```
gzip example.txt
```

5. gunzip

gunzip is used to decompress files compressed with gzip. It restores the original file by decompressing the contents of a ".gz" file.

☞ Syntax

```
gunzip [OPTIONS] [FILE.gz]
```

Example

Decompress a file named example.txt.gz using gunzip:

```
gunzip example.txt.gz
```

6. tar(create and manipulate archive files)

The tar command is a versatile utility that allows you to create and manipulate archive files. It does not perform compression on its own but is often used in combination with compression tools like gzip, bzip2, or xz to create compressed tarballs. The name "tar" stands for "tape archive," as it was originally designed for tape backup operations.

☞ Syntax

```
tar [OPTIONS] [ARCHIVE_FILENAME] [FILES_OR_DIRECTORIES]
```

Options

- -c, --create: Create a new archive.
- -x, --extract, --get: Extract files from an archive.
- -t, --list: List the contents of an archive.
- -f <FILENAME>, --file=<FILENAME>: Specify the archive filename.
- -z, --gzip: Compress the archive using gzip.
- -j, --bzip2: Compress the archive using bzip2.
- -J, --xz: Compress the archive using xz.
- -v, --verbose: Display verbose output.
- -C <DIRECTORY>, --directory=<DIRECTORY>: Change to a specified directory before performing the operation.
- -p, --preserve-permissions: Preserve file permissions.
- --help: Display help message and exit.
- --version: Display version information and exit.

Example

1. Create a tarball from a directory named mydata:

```
tar -cvf mydata.tar mydata/
```

2. Extract files from a tarball:

```
tar -xvf mydata.tar
```

3. List the contents of a tarball:

```
tar -tvf mydata.tar
```

4. Create a compressed tarball using gzip:

```
tar -czvf mydata.tar.gz mydata/
```

5. Extract files from a compressed tarball using gzip:

```
tar -xzvf mydata.tar.gz
```

6. Create a compressed tarball using bzip2:

```
tar -cjvf mydata.tar.bz2 mydata/
```

7. Extract files from a compressed tarball using bzip2:

```
tar -xjvf mydata.tar.bz2
```

Output Example

The output of the tar command typically provides feedback on the operations being performed, such as listing files or directories being processed and their permissions. If you use the -v (verbose) option, you'll see more detailed information about the operations.

Experiment No 4.

Title : Working with Linux Desktop and Utilities

Aim: To study following Linux Desktop and Utilities

a. Vi Editor

b. Graphics user interface

c. Working with Terminal

d. Adjusting Display Resolution

e. Using the Browser

f. Configuration Simple Networking

g. Creating user and Shares

a. Vi Editor

- Vi is a terminal-based text editor that comes pre-installed on most Unix-like systems, including Linux.
- It's known for its modal editing, which means it operates in different modes: Normal, Insert, Visual, and Command-Line mode.
- Vi is highly configurable and has a wide range of features that make it a favorite among programmers, system administrators, and power users.

Modes of Operation

1. **Normal Mode (Command Mode):** This is the default mode when you open Vi. In this mode, you can navigate, delete, copy, and manipulate text. You issue commands to perform various tasks. To enter Normal mode, press the Esc key.

2. **Insert Mode:** In this mode, you can directly input and edit text. To enter Insert mode:
   - Press ‘I’ to insert text before the cursor.
   - Press ‘a’ to insert text after the cursor.
   - Press ‘I’ to insert text at the beginning of the line.
   - Press ‘A’ to insert text at the end of the line.
3. **Visual Mode:** Visual mode allows you to select and manipulate blocks of text. To enter Visual mode, press ‘v’. You can then use navigation keys to select text.
4. **Command-Line Mode:** This mode is used to execute commands and save changes. To enter Command-Line mode, press ‘:’ in Normal mode.

Common Vi Commands (Normal Mode)

- i: Enter Insert mode before the cursor.
- a: Enter Insert mode after the cursor.
- o: Open a new line below the cursor and enter Insert mode.
- O: Open a new line above the cursor and enter Insert mode.
- h, j, k, l: Navigate left, down, up, and right.
- x: Delete the character under the cursor.
- dd: Delete the current line.
- yy: Copy the current line.
- p: Paste the copied or deleted text.
- u: Undo the last change.
- Ctrl + r: Redo the undone change.
- :w: Save the file.
- :q: Quit Vi.
- :q!: Quit without saving changes.
- :wq or ZZ: Save and quit.

Usage Examples

1. Open a file in Vi

```
vi filename.txt
```

2. Navigate and edit text

- Press i to enter Insert mode.
- Edit the text.
- Press Esc to exit Insert mode.

3. Delete a line

- Move the cursor to the line.
- Type dd to delete the line.

4. Save and quit

- Press Esc to ensure you're in Normal mode.
- Type :wq and press Enter.

Point to be noted

- Vi supports advanced features like regular expressions, macros, and global search and replace.
- Vim is an improved version of Vi and is often recommended for beginners due to additional features and better usability.

b. Graphics user interface

- The graphical user interface (GUI) is a visual representation of the operating system and applications.
- It allows users to interact with the computer using a combination of graphical elements such as windows, buttons, menus, and icons.
- The Linux GUI provides a user-friendly environment for performing tasks, managing files, running applications, and customizing settings.

Modes of Operation

The Linux GUI operates in the following modes:

1. **Desktop Mode:** This is the primary mode where you interact with the desktop environment. It includes the desktop background, icons, panels, and other visual elements. You can

launch applications, open files, and perform various tasks from the desktop.

2. **Application Mode:** When you open an application, it runs in its own window. Each application window can have its own set of menus, buttons, and controls for interacting with the application's features.

Common GUI Concepts and Commands

While the Linux GUI is largely point-and-click, there are several common concepts and commands that are useful:

1. **Menus and Toolbars:** Applications have menus at the top of the window that provide access to various functions. Common menu items include "File," "Edit," "View," "Help," etc. Toolbars offer quick access to frequently used commands.
2. **Icons and Desktop Shortcuts:** Icons represent applications, files, and folders. You can click on icons to launch applications or open files.
3. **Window Management:**
   - **Minimize:** Click the minimize button (usually represented by an underscore) to hide a window.
   - **Maximize/Restore:** Click the maximize button (usually represented by a square) to expand the window to full screen. Click again to restore the original size.
   - **Close:** Click the close button (usually represented by an "X") to close the window.

4. **File Management**
   - **Nautilus (Files):** The file manager in GNOME, providing a graphical way to navigate, manage, and organize files and directories.
   - **Dolphin:** The file manager in KDE, offering similar functionality.
   - **Application Launching:** You can launch applications from the application menu, desktop shortcuts, or the launcher.

5. **Taskbar/Panel:** The taskbar or panel at the bottom (or top) of the screen displays the currently open applications, allowing you to switch between them.
6. **Keyboard Shortcuts:** Many GUI actions can also be performed using keyboard shortcuts. For example, Ctrl + C to copy, Ctrl + V to paste, etc.

Usage Examples

1. Launch an application

- Click on the application's icon on the desktop or in the application menu.

2. Open a file

- Double-click on the file's icon on the desktop or in a file manager.

3. Manage windows

- Click the maximize button to expand a window to full screen.
- Click the close button to close a window.

4. Organize files

- Click and drag files to move or copy them within the file manager.
- Right-click on a file to access context menus for various actions.

Note the following points

- The Linux GUI experience can vary depending on the desktop environment you are using (e.g., GNOME, KDE, XFCE, etc.).
- Customizing the appearance and behavior of the GUI is often possible through system settings.

C. Working with Terminal

The terminal, also known as the command-line interface (CLI), allows users to interact with the Linux operating system by typing commands instead of using a graphical user interface (GUI). The terminal provides a powerful and efficient way to perform tasks, manage files, run programs, and configure system settings.

Modes of Operation

The terminal operates in various modes, which include:

- **Command Mode:** In this mode, you can enter commands to perform tasks, manage files, and execute programs. The terminal processes the entered commands and provides output as text.
- **Text Input Mode:** When you run certain commands, the terminal may require you to provide additional input, such as text for configuration or editing purposes.
- **Common Terminal Commands:** Here are some commonly used terminal commands and their descriptions:

1. Navigation and File Management

- ls: List files and directories.
- cd: Change directory.
- pwd: Print current working directory.
- mkdir: Create a new directory.
- cp: Copy files or directories.
- mv: Move or rename files or directories.
- rm: Remove files or directories.
- find: Search for files and directories.

2. Text Editing and Viewing

- nano: A simple terminal-based text editor.
- vi or vim: A powerful terminal-based text editor.
- cat: Display the contents of a file.
- less: View text files one screenful at a time.

3. System Information and Management

- top: Display system processes and resource usage.

- ps: List active processes.
- df: Display disk space usage.
- free: Display memory usage.
- uname: Display system information.
- shutdown: Shutdown or reboot the system.

4. Networking and Connectivity

- ping: Test network connectivity.
- ifconfig or ip: Display network interface information.
- ssh: Securely connect to remote servers.

5. Archiving and Compression

- tar: Create and manage archive files.
- gzip, bzip2, xz: Compress and decompress files.

Usage Examples

1. List files and directories in the current directory:

```
$ls
```

2. Change to a different directory:

```
$cd /path/to/directory
```

3. Display disk space usage:

```
$df -h
```

4. View the contents of a text file using less:

```
$less filename.txt
```

5. Edit a text file using nano:

```
$nano filename.txt
```

Point to be noted

- Use the Tab key for auto-completion of commands and file paths.
- Use Ctrl + C to interrupt a running command.
- Use Ctrl + D to log out of the terminal.

d. Adusting Display Resolution

- Display resolution refers to the number of pixels that make up the content displayed on your monitor.
- Higher resolutions provide more detail and allow you to fit more content on the screen, but the trade-off can be smaller text and icons.
- Lower resolutions may result in larger text and icons, but less content can fit on the screen. Adjusting the display resolution can improve visual clarity and optimize your work environment.

Commands

Adjusting display resolution can often be done using GUI tools provided by your desktop environment. However, some commands can be used to manage display settings as well.

- **xrandr:** The xrandr command-line tool is commonly used to configure display settings. It can be used to list available resolutions, change resolutions, and configure multiple monitors.
- **Mode of Operation:**

  How to use the xrandr command to adjust display resolution:

1. List Available Resolutions

- To list available resolutions for your display, open a terminal and run:

```
xrandr
```

- The output will show a list of resolutions that your display supports.

2. Change Resolution

- To change the resolution, use the --mode option followed by the desired resolution. For example, to change the resolution to 1920x1080, run:

```
xrandr --output <display_name> --mode 1920x1080
```

- Replace <display_name> with the actual name of your display (e.g., eDP-1, HDMI-1, etc.).

☞ Usage Examples

1. List available resolutions

```
xrandr
```

2. Change resolution to 1920x1080

```
xrandr --output eDP-1 --mode 1920x1080
```

Points to be noted

- GUI tools provided by your desktop environment (e.g., System Settings, Display Settings) often offer a user-friendly way to adjust display resolution. Look for these settings in your system's menus.
- Changing display resolution may require adjustments to other settings, such as font scaling, to maintain readability.

e. Uisng the Browser

- A web browser is a software application that allows you to access and navigate the World Wide Web.
- It enables you to view websites, download files, watch videos, and interact with online content.
- Web browsers provide a graphical interface for navigating the internet and offer various features such as bookmarks, tabs, extensions, and privacy settings.

Commands

Interacting with a web browser primarily involves using the graphical user interface (GUI) rather than terminal commands. However, you can use the terminal to open a browser from the command line.

1. Opening a Browser

You can open a specific web browser from the terminal using its command. For example:

- Open Firefox: firefox
- Open Google Chrome: google-chrome
- Open Chromium: chromium-browser

Note the actual commands may vary based on your system and installed browsers.

Mode of Operation

1. Launch the Browser

Open your preferred web browser from the application menu, desktop icon, or terminal command.

2. Enter URLs

In the browser's address bar, type the URL of the website you want to visit and press Enter.

3. Navigate Web Pages

- Use the mouse to click links and buttons.
- Scroll using the mouse wheel or touchpad.
- Use keyboard shortcuts like Ctrl + T to open a new tab and Ctrl + W to close a tab.

4. Tabs

- Open multiple tabs to view multiple websites simultaneously.
- Click on a tab to switch between open web pages.

5. Bookmarks

- Save frequently visited websites as bookmarks for quick access.
- Manage bookmarks through the browser's menu.

6. Search

- Use the search bar in the browser's GUI to perform web searches.
- Search engines like Google, Bing, or DuckDuckGo are commonly used.

7. Download Files

- Click on links to download files.
- Manage downloaded files in the browser's download manager.

8. Extensions/Add-ons

- Customize the browser with extensions or add-ons for added functionality.
- Extensions can block ads, enhance security, and more.

9. Settings

- Access the browser's settings to configure preferences, privacy, and other options.

☞ Usage Examples

1. **Open Firefox from the terminal :** firefox
2. **Visit a website**
   - Launch the browser.
   - Type the URL (e.g., "https://www.example.com") in the address bar and press Enter.
3. **Open a new tab and perform a search**
   - Press Ctrl + T to open a new tab.
   - Type your search query in the address bar and press Enter.

☞ Points to be noted

- Familiarize yourself with keyboard shortcuts for faster navigation (e.g., Ctrl + T for a new tab, Ctrl + L to focus on the address bar).
- Explore browser extensions/add-ons to enhance your browsing experience.

f. Configuration Simple Networking

- Simple networking configuration in Linux involves setting up the basic network connectivity options required for your system to communicate with other devices on a local network or the internet.

- This includes configuring IP addresses, DNS settings, and network interfaces.

Commands

To configure networking, you'll use commands and utilities that interact with the network settings. While the primary focus is on GUI tools for simplicity, you can also use terminal commands for more advanced configuration.

1. **NetworkManager**: A commonly used GUI tool for managing network connections and settings in Linux.
2. **Mode of Operation**: To use GUI tools to configure simple networking in Linux:

a. NetworkManager

- Open your system settings and look for "Network," "Network Connections," or a similar option.
- Use the GUI tool to configure wired, wireless, or other network connections.
- Enter the SSID and password for wireless connections.
- Configure IP addressing (Automatic DHCP or Manual).
- Set DNS servers and other advanced settings if necessary.
- Save the settings and activate the connection.

Usage Examples

- Open the "Network" settings from the system menu.
- Click on "Wi-Fi" to configure wireless settings.
- Enter the SSID and password for your Wi-Fi network.
- Choose the IP addressing method (DHCP or Manual).
- Configure DNS servers if needed.
- Save the settings and activate the Wi-Fi connection.

☞ Points to be noted

For more advanced networking configurations, commands like **ifconfig, ip, iwconfig**, and edit configuration files manually can be used.

Ifconfig(used to configure and display information)

- The ifconfig command is used to configure and display information about network interfaces on Linux systems.
- It allows you to view and manipulate network settings such as IP addresses, netmasks, broadcast addresses, and more.
- While ifconfig is widely used, newer Linux distributions may recommend using the ip command for more advanced networking tasks.

Syntax

```
ifconfig [INTERFACE] [OPTIONS]
```

Options

- up: Activate the specified interface.
- down: Deactivate the specified interface.
- inet: Configure an IPv4 address.
- netmask: Set the subnet mask for the interface.
- broadcast: Set the broadcast address for the interface.
- promisc: Enable or disable promiscuous mode.
- mtu: Set the Maximum Transmission Unit (packet size) for the interface.

Example

1. Display information about all active network interfaces:

```
$ifconfig
```

2. Activate a network interface:

```
$sudo ifconfig eth0 up
```

3. Deactivate a network interface:

```
$sudo ifconfig eth0 down
```

4. Configure an IPv4 address, subnet mask, and broadcast address for an interface:

```
$sudo ifconfig eth0 192.168.1.100 netmask 255.255.255.0 broadcast
192.168.1.255
```

5. Enable promiscuous mode for an interface:

```
$sudo ifconfig eth0 promisc
```

6. Set the Maximum Transmission Unit (MTU) for an interface:

```
$sudo ifconfig eth0 mtu 1500
```

> **Note:** The ifconfig command requires superuser privileges (sudo) to make changes to network interfaces.

Ip address

- The ip address command is used to view and manage IP addresses assigned to network interfaces on a Linux system.
- It provides detailed information about IP addresses, subnet masks, broadcast addresses, and other related networking parameters.
- Additionally, it allows you to add, delete, or modify IP addresses and configure other networking properties.

Syntax

```
ip address [OPTIONS] [OBJECT]
```

Options

- show: Display IP address information (default action if no option is provided).
- add: Add an IP address to an interface.
- delete: Remove an IP address from an interface.
- label LABEL: Specify a label for the interface.
- dev INTERFACE: Specify the network interface.
- up: Activate the specified interface.
- down: Deactivate the specified interface.

Example

1. Display information about all active network interfaces and their IP addresses:

```
$ip address show
```

2. Display information about a specific network interface (e.g., eth0):

```
$ip address show dev eth0
```

3. Add an IP address to an interface (assuming eth0 is the interface):

```
$sudo ip address add 192.168.1.100/24 dev eth0
```

4. Delete an IP address from an interface:

```
$sudo ip address delete 192.168.1.100/24 dev eth0
```

5. Activate a network interface:

```
$sudo ip link set dev eth0 up
```

6. Deactivate a network interface:

```
$sudo ip link set dev eth0 down
```

Note: The ip address command requires superuser privileges (sudo) to make changes to network interfaces.

g. Creating user and Shares

- Creating users involves adding new user accounts to the Linux system. Each user has a unique username and can have their own home directory where they can store their files and settings.
- Shares, on the other hand, refer to shared directories or folders that multiple users can access and collaborate on.
- These shares can be set up using various methods, such as Network File System (NFS) or Samba.

Commands

To create users and set up shares, you'll use a combination of user management commands and file-sharing utilities. Below are some commonly used commands and utilities for creating users and shares:

- useradd / adduser: Commands for adding new user accounts.
- passwd: Command for setting or changing user passwords.
- usermod: Command for modifying user account properties.
- userdel: Command for deleting user accounts.
- groupadd: Command for creating user groups.
- chown / chgrp: Commands for changing file ownership and group ownership.
- chmod: Command for changing file permissions.
- NFS (Network File System): A protocol for sharing files between UNIX-like systems.
- Samba: A suite of programs for file and printer sharing between Linux and Windows systems.

☞ Mode of Operation

1. Creating Users

- Use the useradd or adduser command to create a new user account:

```
sudo useradd username
or
sudo adduser username
```

- Set the user's password using the passwd command:

```
sudo passwd username
```

2. Creating User Groups

- Use the groupadd command to create a new user group:

```
sudo groupadd groupname
```

3. Setting Up Shares (NFS)

- Install the NFS server package if not already installed.
- Edit the NFS configuration file (/etc/exports) to define shared directories and access permissions.
- Export the shared directories using the exportfs command:

```
sudo exportfs -a
```

- Start the NFS server service.

4. Setting Up Shares (Samba)

- Install the Samba package if not already installed.
- Configure the Samba shares by editing the Samba configuration file (/etc/samba/smb.conf).
- Create Samba user accounts using the smbpasswd command:

```
sudo smbpasswd -a username
```

- Start the Samba server service.

☞ Usage Examples

1. Create a new user account named "manoj":

```
$sudo useradd manoj
```

2. Set a password for the user "manoj":

```
$sudo passwd manoj
```

3. Create a new user group named "developers":

```
$sudo groupadd developers
```

4. Configure an NFS share in the /etc/exports file:

```
$/shared_directory *(rw,sync,no_root_squash)
```

5. Export the NFS share:

```
$sudo exportfs -a
```

6. Configure a Samba share in the /etc/samba/smb.conf file:

```
$[Shared] path = /shared_directory writable = yes guest ok = no
```

7. Create a Samba user and set a password:

```
$sudo smbpasswd -a john
```

Points to be Noted

- Ensure that you have administrative privileges (sudo) when creating users, groups, and setting up shares.
- Customize permissions and access controls based on your security and sharing requirements.

Experiment No 5.

Title : Installing utility software on Linux and Windows

Aim: To study installation utility software on Linux and Windows

☞ Procedure

- Installing utility software on both Linux and Windows systems involves downloading, installing, and configuring various tools that enhance the functionality and usability of your computer.
- Utility software includes a wide range of applications designed to perform specific tasks or provide additional features.

Installation on Linux

Linux distributions use package managers to install and manage software. The specific package manager depends on the distribution. Some common package managers are:

1. APT (Advanced Package Tool) - Debian/Ubuntu:

- To install a utility using APT, open a terminal and use the following command:

```
$sudo apt-get install utility-name
```

- Replace utility-name with the name of the utility you want to install.

2. dnf/yum - Fedora/RHEL/CentOS:

- To install a utility using dnf/yum, open a terminal and use the following command:

```
sudo dnf install utility-name
or
sudo yum install utility-name
```

3. Pacman - Arch Linux:

- To install a utility using Pacman, open a terminal and use the following command:

```
sudo pacman -S utility-name
```

4. Snap - Available on multiple distributions:

- To install a utility using Snap, open a terminal and use the following command:

```
sudo snap install utility-name
```

☞ Installation on Windows

On Windows, utility software is typically installed using executable installer files or Windows Package Manager (winget). Here's how you can install utility software on Windows:

1. Executable Installer:

- Download the executable installer file (usually in .exe format) from the utility's official website.
- Double-click the installer file to run it.
- Follow the on-screen instructions to complete the installation.

2. Windows Package Manager (winget):

- Open PowerShell with administrative privileges.
- To search for a utility, use the following command:

```
winget search utility-name
```

- To install a utility, use the following command:

```
winget install utility-name
```

☞ Common Utility Software

Utility software varies widely in its purpose and functionality. Here are a few examples of utility software you might install on both Linux and Windows:

1. Text Editor

- Linux: Vim, Nano, Gedit
- Windows: Notepad++, Visual Studio Code

2. File Compression/Extraction

- Linux: Tar, Gzip, Unzip

- Windows: 7-Zip, WinRAR

3. System Maintenance

- Linux: BleachBit, Stacer
- Windows: CCleaner

4. Backup and Sync

- Linux/Windows: Dropbox, Google Drive

5. Screen Capture

- Linux: Shutter, Flameshot
- Windows: Snipping Tool, Greenshot

6. Remote Desktop

- Linux/Windows: TeamViewer, AnyDesk

Points to be noted

- Always download utility software from official and trusted sources to ensure security and reliability.
- Be cautious while granting administrative permissions during installation.
- Read user reviews and documentation before installing new utility software.

▶ Experiment No 6.

Title : Running C/C++/Python programs in Linux

Aim: To study how edit, run and execute c/cpp/python program in Linux

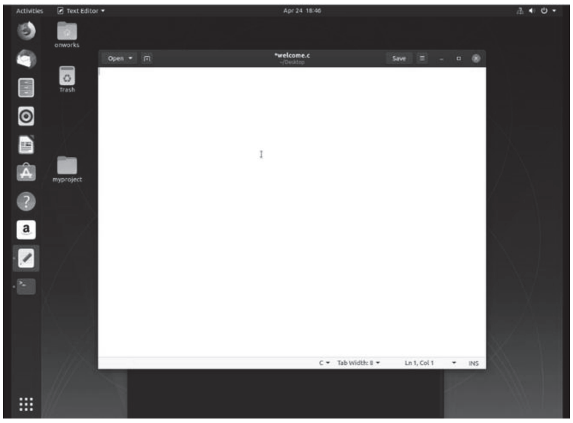
☞ Procedure

Part-1 : To Edit , Compile and execute C program

☞ Method 1: Using CC Compiler

In this method, we will be compiling and executing the C program code using CC Compiler.

Step 1: First, open the text editor and terminal for writing code and executing it through the terminal.

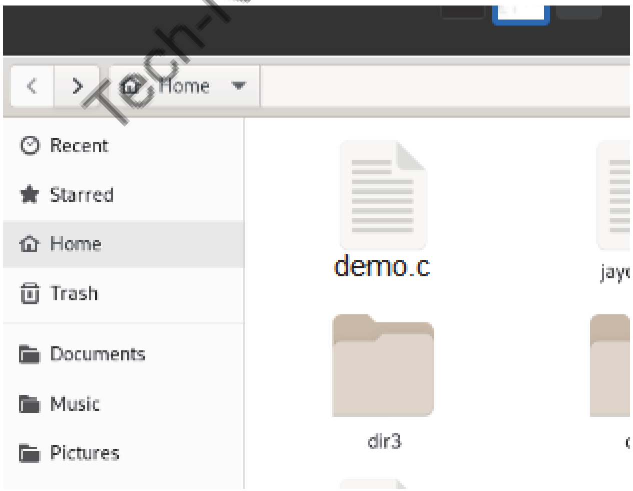


Note : any edit like vi, vim,nano,sublime,gedit etc

Step 2: Write C code in text editor

```
#include <stdio.h>
 int main() {
    printf("Operating System by Er.Manoj S. Kavedia");
    return 0;
}
```

Step 3: Save the file with the .c extension.In this example demo.c is name of file



Step 4: Now compile and run the C code in the terminal using the commands below.

After compiling c program a.out executable file is generated

```
$ cc demo.c
```

To execute the executable file type following

```
$./a.out
```

Output on Linux terminal

```
$ls
dir1 dir2 dir3  dummy.txt manoj.txt demo.c abc.txt hero.img
$ cc demo.c
$ ./a.out
Operating System by Er.Manoj S. Kavedia
```

☞ Method 2: Using GCC Compiler

To compile and execute the script using the GCC compiler can also be used.

C program

```
#include <stdio.h>
 int main()
{
  int a=10,b=20,c;          //declare variable
  c=a+b;                     //perform addition and store result in c variable
  printf("addition = %d\n",c);   // display the result
  return 0;
}
```

Step 1: Navigate to the directory where the file is been saved. Use the cd command.

In this example we are in current directory

Step 2: Execute the command below for compilation and execution.

```
$cc -o add add.c
$./add

Linux Terminal Output
[manojkavedia@localhost]~$ls
```

```
dir1 dir2 dir3  dummy.txt manoj.txt add.c demo.c abc.txt hero.img
[manojkavedia@localhost]~$ gcc add add.c
[manojkavedia@localhost]~$ ./add
addition  = 30
manojkavedia@localhost]~$
```

Part-2 : To Edit , Compile and execute C++ program

In this method, we will be compiling and executing the C++ program code using G++ Compiler.

- **Step 1:** Write the C++ program code in a text file using a text editor and save the file with the .cpp extension.

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Subscribe , Like and Share Youtube channel : IoTwala";
   return 0;
}
```

- **Step 2: Navigate to the directory where the file is saved.using cd command**

We are already in the current directory.

- **Step 3: Execute the command below for compilation and execution.**

```
Compile cpp code and it will generate a.out file
$ g++ cplus.cpp
To execute a.out file
$./a.out

Linux terminal Output
[manojkavedia@localhost]~$ls
dir1 dir2 dir3  dummy.txt manoj.txt demo.c cplus.cpp abc.txt hero.img
[manojkavedia@localhost]~$ g++ cplus.cpp
[manojkavedia@localhost]~$ ./a.out
Subscribe , Like and Share Youtube channel : IoTwala
manojkavedia@localhost]~$
```

Part-3 : To run python code

Open a Terminal: Launch the terminal on your Linux system.

- **Navigate to the Directory:** Use the cd command to navigate to the directory where your Python program is located.
- **Write python program :** using any editor(thorny python, idle etc) write python code and save with extension .py. Here code is saved with name demo.py

```
Print(“Welcome to world of Python”)
```

- **Run the Python Program:** To run a Python program, simply use the python command followed by the name of your Python script. For example, to run a Python program named "myscript.py":

```
$python myscript.py
Welcome to world of Python
```

- If you have both Python 2 and Python 3 installed, then use python3 instead:

```
$ python3 myscript.py
Welcome to world of Python
```

Note: if python is not installed then To install python use

$sudo apt-get install python

To install idle use

$sudo apt-get install idle

▸ Experiment No 7.

Title : Linux Shell Script

Aim : To study and execute shells script in Linux

a. Basic Operator

b. Decison

c. Looping

d. Regular Expression

e. Special Variables and Command Line Argument

☞ Procedure

Scripting

Shell Scripting is an open-source computer program designed to be run by the Unix/Linux shell which could be any one of the following :

- The Bourne Shell
- The C Shell
- The Korn Shell
- The GNU Bourne-Again Shell

A shell is a command-line interpreter and typical operations performed by shell scripts include file manipulation, program execution, and printing text.

Shell Scripting is a program to write a series of commands for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script that can be stored and executed anytime which, reduces programming efforts.

Example of shell script

1. Write a script which says "Hello World".

```
first.sh
#!/bin/sh
# This is a comment!
echo Hello World      # This is a comment, too!
echo $shell            # Give default shell
```

Explanation

The first line tells Unix that the file is to be executed by /bin/sh.

The second line begins with a special symbol: **#**. This marks the line as a comment, and it is ignored completely by the shell.

The third line runs a command:

echo, with two parameters, or arguments - the first is **"Hello"**; the second is **"World"**. Note that **echo** will automatically put a single space between its parameters. The **#** symbol still marks a comment; the # and anything following it is ignored by the shell. Will tell what is your default interpreter to execute command

The fourth line echo $shell

Execution of Bash Script

run **chmod 755 first.sh** to make the text file executable, and run **./first.sh**.

```
$ chmod755 first.sh
$ ./first.sh
HelloWorld
/bin/bash
$
```

2. Write Bash script to display name and a greeting message with some information

```
Greeting.sh
#!/bin/bash
greeting="Welcome"
user=$(whoami)
day=$(date + %A)

echo"$greeting back $user! Today is $day, is the best day of the entire week!"
echo"Bash shell version is: $BASH_VERSION. Enjoy scripting!"
```

☞ Explanation

- First, a variable greeting is declared and assigned a string value Welcome to it.
- The next variable user contains a value of user name running a shell session.
- The output of the whoami command will be directly assigned to the user variable.
- The next variable day holds a name of today's day produced by date +%A command. The second part of the script utilises the echo command to print a message while substituting variable names now prefixed by $ sign with their relevant values.

- The last variable used $BASH_VERSION know that this is a so called internal variable defined as part of your shell.

Output

```
$./greeting.sh
Welcome back Er.ManojKavedia! Today is Monday, is the best day of the entire
week!
Bash shell version is:4.4.12(1) -release Enjoy scripting!
```

a. Basic Operator

There are **5** basic operators in bash/shell scripting:

1. Arithmetic Operators
2. Relational Operators
3. Boolean Operators
4. Bitwise Operators

5.File Test Operators

1. **Arithmetic Operators** : These operators are used to perform normal arithmetics/mathematical operations. There are 7 arithmetic operators:
   - **Addition (+)**: Binary operation used to add two operands.
   - **Subtraction (-)**: Binary operation used to subtract two operands.
   - **Multiplication (*)**: Binary operation used to multiply two operands.
   - **Division (/)**: Binary operation used to divide two operands.
   - **Modulus (%)**: Binary operation used to find remainder of two operands.
   - **Increment Operator (++)**: Unary operator used to increase the value of operand by one.
   - **Decrement Operator (- -)**: Unary operator used to decrease the value of a operand by one
2. **Relational Operators**: Relational operators are those operators which define the relation between two operands.

They give either true or false depending upon the relation. They are of 6 types:

- **'==' Operator**: Double equal to operator compares the two operands. Its returns true is they are equal otherwise returns false.
- **'!=' Operator**: Not Equal to operator return true if the two operands are not equal otherwise it returns false.
- **'<' Operator**: Less than operator returns true if first operand is less than second operand otherwise returns false.
- **'<=' Operator**: Less than or equal to operator returns true if first operand is less than or equal to second operand otherwise returns false
- **'>' Operator**: Greater than operator return true if the first operand is greater than the second operand otherwise return false.
- **'>=' Operator**: Greater than or equal to operator returns true if first operand is greater than or equal to second operand otherwise returns false

3. **Logical Operators** : They are also known as boolean operators. These are used to perform logical operations. They are of 3 types:
   - **Logical AND (&&)**: This is a binary operator, which returns true if both the operands are true otherwise returns false.
   - **Logical OR (||)**: This is a binary operator, which returns true is either of the operand is true or both the operands are true and return false if none of then is false.
   - **Not Equal to (!)**: This is a unary operator which returns true if the operand is false and returns false if the operand is true.
4. **Bitwise Operators**: A bitwise operator is an operator used to perform bitwise operations on bit patterns. They are of 6 types:
   - **Bitwise And (&)**: Bitwise & operator performs binary

AND operation bit by bit on the operands.

- **Bitwise OR (|)**: Bitwise | operator performs binary OR operation bit by bit on the operands.
- **Bitwise XOR (^)**: Bitwise ^ operator performs binary XOR operation bit by bit on the operands.
- **Bitwise complement (~)**: Bitwise ~ operator performs binary NOT operation bit by bit on the operand.
- **Left Shift (<<)**: This operator shifts the bits of the left operand to left by number of times specified by right operand.
- **Right Shift (>>)**: This operator shifts the bits of the left operand to right by number of times specified by right operand.

5. **File Test Operator**: These operators are used to test a particular property of a file.
   - **-b operator**: This operator check whether a file is a block special file or not. It returns true if the file is a block special file otherwise false.
   - **-c operator**: This operator checks whether a file is a character special file or not. It returns true if it is a character special file otherwise false.
   - **-d operator**: This operator checks if the given directory exists or not. If it exists then operators returns true otherwise false.
   - **-e operator**: This operator checks whether the given file exists or not. If it exits this operator returns true otherwise false.
   - **-r operator**: This operator checks whether the given file has read access or not. If it has read access then it returns true otherwise false.
   - **-w operator**: This operator check whether the given file has write access or not. If it has write then it returns true otherwise false.
   - **-x operator**: This operator check whether the given file

has execute access or not. If it has execute access then it returns true otherwise false.

- **-s operator**: This operator checks the size of the given file. If the size of given file is greater than 0 then it returns true otherwise it is false.

b. Decision Statement

There are total 5 conditional statements which can be used in bash programming

1. if statement
2. if-else statement
3. if..elif..else..fi statement (Else If ladder)
4. if..then..else..if..then..fi..fi..(Nested if)
5. switch statement

Their description with syntax is as follows:

1. if statement

This block will process if specified condition is true.

Syntax

```
if [ expression ]
then
  statement
fi
```

2. if-else statement

If specified condition is not true in if part then else part will be execute.

Syntax

```
if [ expression ]
then
  statement1
else
  statement2
fi
```

3. if..elif..else..fi statement (Else If ladder)

- To use multiple conditions in one if-else block, then elif keyword is used in shell.
- If expression1 is true then it executes statement 1 and 2, and this process continues. If none of the condition is true then it processes else part.

Syntax

```
if [ expression1 ]
then
  statement1
  statement2
  .
  .
elif [ expression2 ]
then
  statement3
  statement4
  .
  .
else
  statement5
fi
```

4. if..then..else..if..then..fi..fi..(Nested if)

- Nested if-else block can be used when, one condition is satisfies then it again checks another condition.
- In the syntax, if expression1 is false then it processes else part, and again expression2 will be check.

☞ Syntax

```
if [ expression1 ]
then
  statement1
  statement2
  .
else
  if [ expression2 ]
  then
    statement3
    .
  fi
fi
```

5. switch statement

- case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern
- When a match is found all of the associated statements until the double semicolon (;;) is executed.
- A case will be terminated when the last command is executed. If there is no match, the exit status of the case is zero.

☞ Syntax

```
case  in
  Pattern 1) Statement 1;;
  Pattern n) Statement n;;
esac
```

c. Looping Statement

There are total 3 looping statements which can be used in bash programming

1. while statement
2. for statement
3. until statement

To alter the flow of loop statements, two commands are used they are,

1. break
2. continue

Their descriptions and syntax are as follows:

1. while statement

Here command is evaluated and based on the result loop will executed, if command raise to false then loop will be terminated

☞ Syntax

```
while command
do
  Statement to be executed
      done
```

2. for statement

The for loop operate on lists of items. It repeats a set of commands for every item in a list. Here var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

☞ Syntax

```
for var in word1 word2 ...wordn
do
  Statement to be executed
    done
```

3. until statement

The until loop is executed as many as times the condition/command evaluates to false. The loop terminates when the condition/command becomes true.

☞ Syntax

```
until command
do
  Statement to be executed until command is true
done
```

d. Regular Expression

A regular expression is a sequence of characters that defines a search pattern. It is a versatile tool for text processing tasks like searching, extracting, and replacing specific patterns within strings. Regular expressions are supported by many Linux commands, including grep, sed, awk, and others.

☞ Syntax

Regular expressions use a combination of characters, symbols, and metacharacters to define patterns. Here are some common metacharacters and their meanings:

- .: Matches any character.
- *: Matches zero or more occurrences of the previous character or group.
- +: Matches one or more occurrences of the previous character or group.
- ?: Matches zero or one occurrence of the previous character or group.
- []: Defines a character set; matches any character within the set.
- [^]: Negates a character set; matches any character not in the set.
- (): Groups characters or expressions together.
- \: Escapes a metacharacter to treat it as a literal character.

e. Special Variables and Command Line Argument

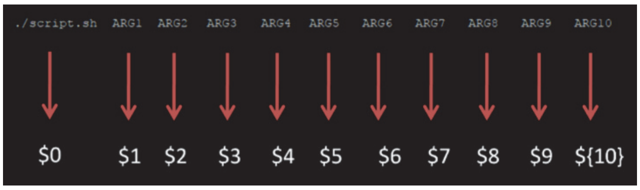
Command Line Arguments in Shell Script

- Command line arguments are also known as positional parameters.
- These arguments are specific with the shell script on terminal during the run time.
- Each variable passed to a shell script at command line are stored in corresponding shell variables including the shell script name.

☞ Syntax

```
./myshellscript.sh  ARG1 ARG2 ARG3 ARG4 ARG5 ARG6 ARG7 ARG8
ARG9 ARG10
```

Here ARG1, ARG2 to ARG10 are command line values, which is assigned to corresponding shell variables.



These are also known as special variables provided by the shell. There are some more special variables as given below.

Special Variable	Variable Details
$1 to $n	$1 is the first arguments, $2 is second argument till $n n'th arguments. From 10'th argument, you must need to inclose them in braces like ${10}, ${11} and so on
$0	The name of script itself
$$	Process id of current shell
$*	Values of all the arguments. All arguments are double quoted
$#	Total number of arguments passed to script
$@	Values of all the arguments
$?	Exit status id of last command
$!	Process id of last command

Common Environmental and Shell Variables

Some environmental and shell variables are very useful and are referenced fairly often. Here are some common environmental variables that you will come across:

- **SHELL:** This describes the shell that will be interpreting any commands you type in. In most cases, this will be bash by default, but other values can be set if you prefer other options.

- **TERM:** This specifies the type of terminal to emulate when running the shell. Different hardware terminals can be emulated for different operating requirements. You usually won't need to worry about this though.
- **USER:** The current logged in user.
- **PWD:** The current working directory.
- **MAIL:** The path to the current user's mailbox.
- **PATH:** A list of directories that the system will check when looking for commands. When a user types in a command, the system will check directories in this order for the executable.
- **LANG:** The current language and localization settings, including character encoding.
- **HOME:** The current user's home directory.
- **_:** The most recent previously executed command.
- **BASH_VERSION:** The version of bash being executed, in human-readable form.
- **BASH_VERSINFO:** The version of bash, in machine-readable output.
- **HOSTNAME:** The hostname of the computer at this time.
- **UID:** The UID of the current user.

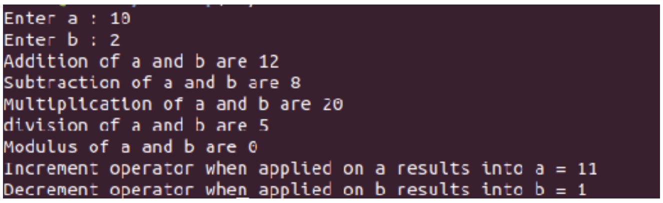
Example of Shell Script

1. Shell script to perform Arithmetic Operators

```
#!/bin/bash
#reading data from the user
read - p 'Enter a : ' a    // a = input("Enter a")
          read
   - p 'Enter b : ' b

         //add
  add = $((a + b))
       echo Addition of a and b are $add    // print
  sub = $((a - b))
    echo Subtraction of a and b are $sub

      //mul
  = $((a * b))
    echo Multiplication of a and b are $mul

      //div
  = $((a / b))
    echo division of a and b are $div

      // mod
  = $((a % b))
     echo Modulus of a and b are $mod

   ((++a))
     echo Increment
     operator when applied on "a" results into a = $a

   ((--b))
     echo Decrement
     operator when applied on "b" results into b = $b
```

Output

$ ./arithmetic.sh



```
Enter a : 10
Enter b : 2
Addition of a and b are 12
Subtraction of a and b are 8
Multiplication of a and b are 20
division of a and b are 5
Modulus of a and b are 0
Increment operator when applied on a results into a = 11
Decrement operator when applied on b results into b = 1
```

2. Shell script to perform Relational Operators

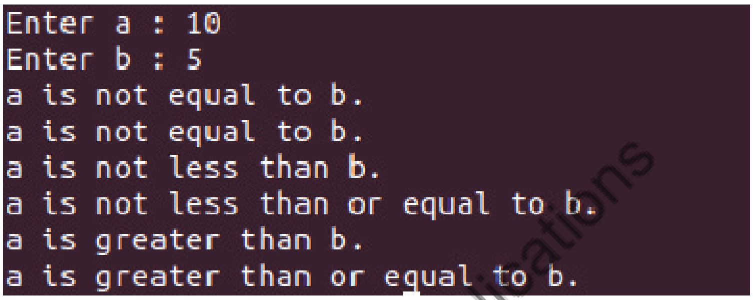
```
#!/bin/bash

#reading data from the user
```

```
read -p 'Enter a : ' a
read -p 'Enter b : ' b

if(( $a==$b ))
then
   echo a is equal to b.
else
   echo a is not equal to b.
fi

if(( $a!=$b ))
then
   echo a is not equal to b.
else
   echo a is equal to b.
fi

if(( $a<$b ))
then
   echo a is less than b.
else
   echo a is not less than b.
fi

if(( $a<=$b ))
then
   echo a is less than or equal to b.
else
   echo a is not less than or equal to b.
fi

if(( $a>$b ))
then
   echo a is greater than b.
else
   echo a is not greater than b.
fi
```

```
if(( $a>=$b ))
then
   echo a is greater than or equal to b.
else
   echo a is not greater than or equal to b.
fi
```

Output

$ ./relational.sh



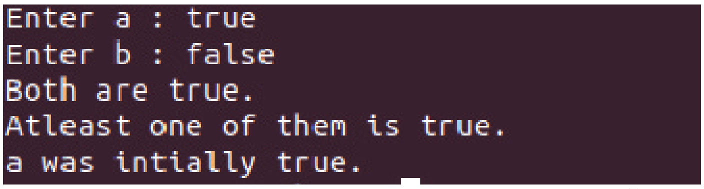
```
Enter a : 10
Enter b : 5
a is not equal to b.
a is not equal to b.
a is not less than b.
a is not less than or equal to b.
a is greater than b.
a is greater than or equal to b.
```

3. Shell script to perform Relational Operators

```
#!/bin/bash
#reading data from the user
read -p 'Enter a : ' a
read -p 'Enter b : ' b


if(($a == "true" & $b == "true" ))
then
   echo Both are true.
else
   echo Both are not true.
fi


if(($a == "true" || $b == "true" ))
then
   echo Atleast one of them is true.
else
   echo None of them is true.
fi
```

```
if(( ! $a == "true"  ))
then
   echo "a" was initially false.
else
    echo "a" was initially true.
 fi
```

Output



```
Enter a : true
Enter b : false
Both are true.
Atleast one of them is true.
a was intially true.
```

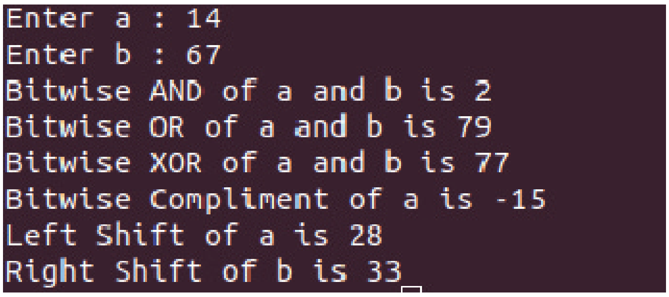
4. Shell script to perform Bitwise Operators

```
#!/bin/bash

#reading data from the user
read -p 'Enter a : ' a
read -p 'Enter b : ' b

bitwiseAND=$(( a&b ))
echo Bitwise AND of a and b is $bitwiseAND

bitwiseOR=$(( a|b ))
echo Bitwise OR of a and b is $bitwiseOR

bitwiseXOR=$(( a^b ))
echo Bitwise XOR of a and b is $bitwiseXOR

bitiwiseComplement=$(( ~a ))
echo Bitwise Compliment of a is $bitiwiseComplement

leftshift=$(( a<<1 ))
echo Left Shift of a is $leftshift

rightshift=$(( b>>1 ))
```

```
echo Right Shift of b is $rightshift
```

Output

$ ./bitwise.sh



```
Enter a : 14
Enter b : 67
Bitwise AND of a and b is 2
Bitwise OR of a and b is 79
Bitwise XOR of a and b is 77
Bitwise Compliment of a is -15
Left Shift of a is 28
Right Shift of b is 33_
```

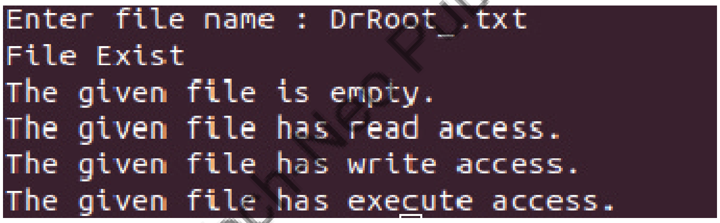
5. Shell script to perform File Test Operators

```
#!/bin/bash

#reading data from the user
read -p 'Enter file name : ' FileName

if [ -e $FileName ]
then
    echo File Exist
else
    echo File doesnot exist
fi

if [ -s $FileName ]
then
    echo The given file is not empty.
else
    echo The given file is empty.
fi

if [ -r $FileName ]
then
    echo The given file has read access.
else
    echo The given file does not has read access.
```

```
fi

if [ -w $FileName ]
then
    echo The given file has write access.
else
    echo The given file does not has write access.
fi

if [ -x $FileName ]
then
    echo The given file has execute access.
else
    echo The given file does not has execute access.
fi
```

Output

$ ./filetestoperator.sh



```
Enter file name : DrRoot_.txt
File Exist
The given file is empty.
The given file has read access.
The given file has write access.
The given file has execute access.
```

6. Shell script to demonstrate if-else statement

```
#!/bin/bash
# Get the user's age as input
read -p "Please enter your age: " age

# Check if the user is eligible to vote
if [ $age -ge 18 ]; then
    echo "You are eligible to vote. Make your voice count!"
else
    echo "Sorry, you are not eligible to vote yet. Wait a few more years."
fi
```

Output

Suppose you run the script and input an age of 20:

Please enter your age: 20 You are eligible to vote. Make your voice count!

If you input an age of 16:

Please enter your age: 16 Sorry, you are not eligible to vote yet. Wait a few more years.

7. Shell script to demonstrate if-elif-else-fi statement

```
#!/bin/sh
a=10
b=20
if [ $a == $b ]
then
   echo "a is equal to b"
elif [ $a -gt $b ]
then
   echo "a is greater than b"
elif [ $a -lt $b ]
then
   echo "a is less than b"
else
   echo "None of the condition met"
fi
```

Output

```
$ ./ifelifelseif.sh
a is less than b
```

8. Shell script to demonstrate Switch-case statement

```
CARS="nano"

#Pass the variable in string
case "$CARS" in
    #case 1
    "mercedes") echo "Headquarters - Germany" ;;

    #case 2
    "audi") echo "Headquarters - Australia" ;;
```

```
   #case 3
   "nano") echo "Headquarters - India" ;;
esac
```

Output

```
$bash -f Switch.sh  or ./Switch.sh
Headquarters - India.
```

9. Shell script to demonstrate for-loop

1. Implementing for loop with break statement

```
#Start of for loop
for a in 1 2 3 4 5 6 7 8 9 10
do
   # if a is equal to 5 break the loop
   if [ $a == 5 ]
   then
      break
   fi
   # Print the value
   echo "Iteration no $a"
done
```

Output

```
$bash -f main.sh
Iteration no 1
Iteration no 2
Iteration no 3
Iteration no 4
```

2. Implementing for loop with continue statement

```
for a in 1 2 3 4 5 6 7 8 9 10
do
   # if a = 5 then continue the loop and
   # don't move to line 8
   if [ $a == 5 ]
   then
      continue
   fi
   echo "Iteration no $a"
done
```

Output

```
$bash -f main.sh
Iteration no 1
Iteration no 2
Iteration no 3
Iteration no 4
Iteration no 6
Iteration no 7
Iteration no 8
Iteration no 9
Iteration no 10
```

10. Shell script to find factorial of Number using for -Loop

Forfactorial.sh

```
#!/bin/bash
echo "Enter a number"     #display message
read num                  # read number

fact=1                    #intialise variable fact to 1

for((i=2;i<=num;i++))     # repeat the process using for loop
{
  fact=$((fact * i))      #fact = fact * i
}

echo "Factorial of Number is"
echo $fact                #display factorial of a number
```

Output

```
Enter a number
3
Factorial of Number is
6
```

11. Shell script to demonstrate while-loop

```
a=0
# -lt is less than operator

#Iterate the loop until a less than 10
while [ $a -lt 10 ]
do
   # Print the values
   echo $a

   # increment the value
   a=`expr $a + 1`
done
```

Output

```
$bash -f main.sh
0
1
2
3
4
5
6
7
8
9
```

12. Shell script to find factorial of number using while-loop

whileFactorial.sh

```
#!/bin/bash
echo "Enter a number"    #display message
read num                  # read number
fact=1                   #intialise variable fact to 1

while [ $num -gt1 ]    # repeat while num > 1
do
fact=$((fact * num))  #fact = fact * num
  num=$((num - 1))   #num = num - 1
```

```
done                # end of while loop

echo “Factorial of Number is”
echo $fact           #display factorial of a number
```

13. Shell script to demonstrate until-loop

```
a=0
# -gt is greater than operator

#Iterate the loop until a is greater than 10
until [ $a -gt 5 ]
do
   # Print the values
   echo $a

   # increment the value
   a=`expr $a + 1`
done
```

Output

```
$bash -f main.sh
0
1
2
3
4
5
```

14. Shell script to find whether character is vowel or consonant

Vovelconsonant.sh

```
#!/bin/bash
clear
echo "Enter any character: "
read ch case $ch in                  # case...esac statement
        "a") echo "It is a vowel.";;
        "e") echo "It is a vowel.";;
        "i") echo "It is a vowel.";;
        "o") echo "It is a vowel.";;
        "u") echo "It is a vowel.";;
          *) echo "It is consonant."   #default condition
esac
```

Output

```
Enter any character
a
It is a vowel

Enter any character
b
It is  consonant
```

15. Shell script to demonstrate Shell variable and special command Line argument

Commandlineargument.sh

```
#!/bin/sh
echo "File Name: $0"
echo "First Parameter : $1"
echo "Second Parameter : $2"
echo "Quoted Values: $@"
echo "Quoted Values: $*"
echo "Total Number of Parameters : $#"
```

Output

```
$./test.sh Manoj Kavedia
File Name : ./test.sh
First Parameter : Manoj
Second Parameter : Kavedia
Quoted Values: Manoj Kavedia
Quoted Values: Manoj Kavedia
Total Number of Parameters : 2
Shellvariable.sh
#!/bin/bash
# Display the value of the HOME environmental variable
echo "Home Directory: $HOME"
# Display the value of the PATH environmental variable
echo "Path: $PATH"
# Display the value of the USER environmental variable
```

```
echo "Username: $USER"
# Display the value of the SHELL environmental variable
echo "Shell: $SHELL"
# Display the value of the LANG environmental variable
echo "Language: $LANG"
```

Output

```
$./shellvariables.sh
Home Directory: /home/username
Path:
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/g
ames:/snap/bin
Username: username
Shell: /bin/bash
Language: en_US.UTF-8
```

Experiment No 8.

Title : case Study of Server OS

Aim: To study windows Server 2022 Operating System – Architecture, Components, Services, and Configuration

Theory : Architecture

Windows Server 2022 is a powerful and versatile server operating system developed by Microsoft. It is built upon the Windows NT architecture and shares many architectural components with its predecessors. Here are some key architectural aspects of Windows Server 2022:

Case study of Server OS

Windows Server 2022 operating System - Architecture, Components, Services, Configuration

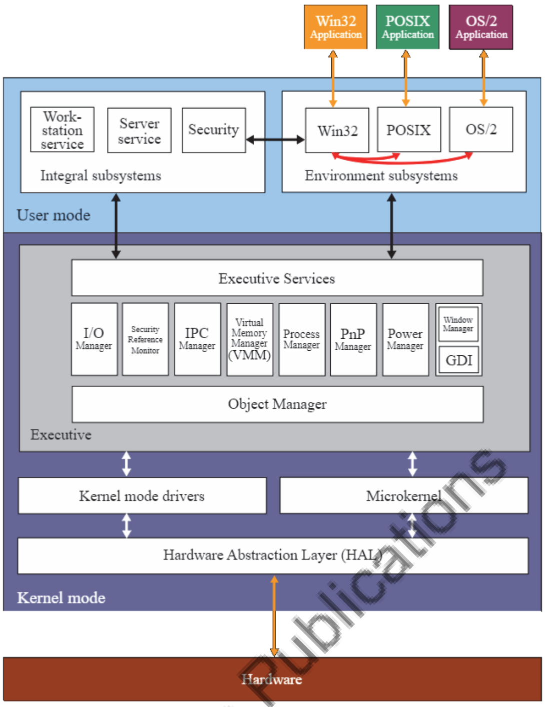


Fig. 1 : Architecture

- **Kernel:** The core of the operating system that manages memory, processes, and hardware interactions.
- **User Mode and Kernel Mode:** Windows Server 2022, like other Windows operating systems, employs a protected mode architecture where user mode and kernel mode processes are isolated for security and stability.
- **Hardware Abstraction Layer (HAL):** An abstraction layer that separates hardware-dependent functions from the rest of the operating system, enabling Windows Server to run on various hardware platforms.
- **File System:** Windows Server 2022 supports NTFS (New Technology File System) and ReFS (Resilient File System) for advanced file management and data protection.

Components

Windows Server 2022 is composed of various components that work together to provide a wide range of services and functionalities. Some key components include:

- **Active Directory:** A centralized directory service that manages and authenticates network resources, users, and devices.
- **Hyper-V:** A virtualization platform that allows you to create and manage virtual machines.
- **Networking:** Includes components for managing networking, such as TCP/IP stack, DNS, DHCP, and routing.
- **Windows Defender:** Provides built-in antivirus and antimalware protection for the server.
- **Internet Information Services (IIS):** A web server role that enables hosting websites and web applications.
- **Remote Desktop Services (RDS):** Allows users to access applications and desktops remotely.
- **Storage Spaces:** Enables the creation of flexible and redundant storage solutions.
- **Failover Clustering:** Provides high availability and failover capabilities for critical services.
- **Windows Admin Center:** A management tool that provides a modern web-based interface for server administration.

Services

Windows Server 2022 offers a wide range of services to support various enterprise needs. Some important services include:

- **Active Directory Domain Services (AD DS):** Manages user accounts, security, and permissions.
- **Domain Name System (DNS):** Resolves domain names to IP addresses.
- **Dynamic Host Configuration Protocol (DHCP):** Assigns IP addresses and network configuration to clients.
- **File and Storage Services:** Provides file sharing, storage management, and data protection.

- **Web Services:** Includes Internet Information Services (IIS) for hosting websites and web applications.
- **Remote Desktop Services (RDS):** Enables remote access to applications and desktops.
- **Hyper-V:** Virtualization platform for creating and managing virtual machines.
- **Print Services:** Manages network printing and print server roles.
- **Windows Update Services (WSUS):** Manages and deploys Windows updates within the network.

Configuration

Configuring Windows Server 2022 involves setting up roles, features, and services to meet specific organizational requirements. The process includes:

- **Server Manager:** Windows Server 2022 comes with the Server Manager tool, which allows administrators to manage roles, features, and overall server configuration.
- **Roles and Features:** Use the Server Manager to add or remove server roles and features based on the server's intended use (e.g., web server, domain controller, file server).
- **Networking Configuration:** Configure network interfaces, IP addresses, and network settings using the Network and Sharing Center or PowerShell commands.
- **Security Configuration:** Implement security policies, manage user accounts, and configure firewalls to ensure server security.
- **Remote Management:** Enable remote management through tools like Windows Admin Center, PowerShell Remoting, and Remote Desktop.

Conclusion

Windows Server 2022 is a robust and feature-rich server operating system designed to provide a wide range of services and capabilities for enterprise environments. Its architecture, components, services, and configuration options make it a versatile platform for hosting critical applications, managing network resources, and ensuring the efficient operation of IT infrastructure.

▸ Experiment No 9.

Title : Case Study of Android OS

Aim: To study Android Operating System – Architecture, Components, Services, and Configuration

Thoery : Architecture

Android is an open-source operating system developed by Google primarily for mobile devices. It is based on the Linux kernel and follows a layered architecture that enables a wide range of hardware and software compatibility. Here are the key architectural aspects of the Android operating system:

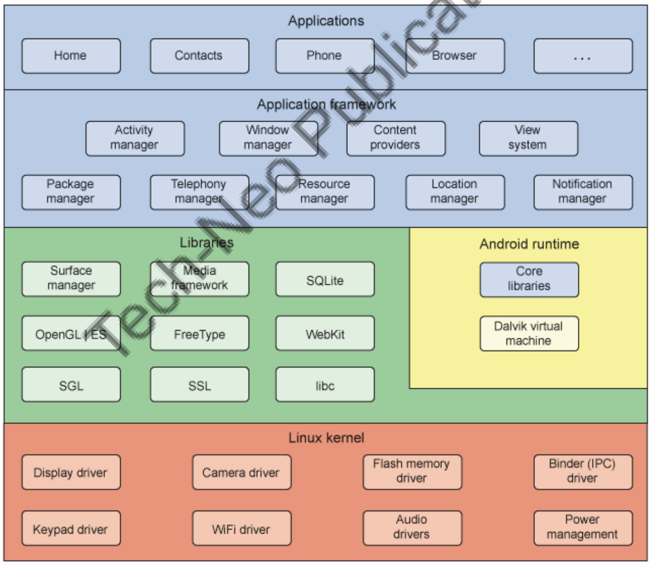


Fig. 2 : Architecture of Android OS

- **Linux Kernel:** The core of the Android OS, responsible for hardware abstraction, memory management, and process scheduling.
- **Android Runtime (ART):** The runtime environment that

executes Android applications. It compiles and executes Java-based code into native machine code.

- **Libraries:** Libraries provide essential functions and APIs for app development, including graphics rendering, database access, and more.
- **Application Framework:** A collection of Java classes and APIs that enable developers to build applications. It includes services for activities, content providers, broadcast receivers, and services.
- **Applications:** User-facing apps that run on the Android platform. These apps are developed using the Java programming language and the Android SDK.
- **Components:** Android is composed of various components that work together to provide a seamless user experience and app development environment. Some key components include:
- **Activities:** Represent user interfaces and screen interactions within an app.
- **Services:** Run in the background and perform long-running tasks independently of the user interface.
- **Content Providers:** Manage access to structured data, such as databases or files, and enable data sharing between apps.
- **Broadcast Receivers:** Respond to system-wide events or broadcasts, such as incoming calls or battery status changes.
- **Intents:** Facilitate communication between components and apps by triggering actions or launching activities.

Services

Android offers a variety of services that enhance the user experience and support app development:

- **Google Play Services:** Provides APIs for authentication, location services, maps, and more, allowing developers to integrate Google services into their apps.
- **Notification Service:** Manages notifications to keep users informed about events and activities.

- **Location Service:** Provides location data to apps based on GPS, Wi-Fi, or cellular network signals.
- **Media Services:** Supports audio and video playback, recording, and media streaming.
- **Bluetooth and NFC Services:** Enable communication between devices via Bluetooth and Near Field Communication (NFC).

Configuration

Configuring Android involves setting up the device, managing apps, and adjusting system settings. Key configuration tasks include:

- **Initial Setup:** Configure language, time zone, Wi-Fi networks, and Google account during the device setup process.
- **App Installation and Management:** Download apps from the Google Play Store, update or uninstall apps, and manage app permissions.
- **Network Settings:** Configure Wi-Fi, mobile data, and hotspot settings.
- **Security Settings:** Set up device security, including screen lock, fingerprint recognition, and device encryption.
- **Accessibility Settings:** Customize settings for users with disabilities, such as text-to-speech, screen magnification, and gesture controls.
- **System Updates:** Install software updates provided by device manufacturers or carriers to improve performance and security.

Conclusion

Android is a versatile and widely used operating system that powers a diverse range of mobile devices. Its architecture, components, services, and configuration options contribute to its popularity among developers and users alike. Android's open nature and extensive ecosystem enable a rich app ecosystem and provide users with a customizable and feature-rich mobile experience.

▸ Experiment No 10.

Title : Case Study of Cloud OS

Aim: To study Cloud Operating System – AWS, Azure, Google Cloud

Thoery

Cloud operating systems, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP), provide a comprehensive set of cloud computing services that enable organizations to build, deploy, and manage applications and services without the need for physical hardware. These cloud platforms offer a wide range of features, services, and tools that support various use cases, from web hosting to machine learning. In this case study, we will explore the architecture, components, services, and benefits of each of these leading cloud operating systems.

A. Amazon Web Services (AWS)

Architecture: AWS is designed as a collection of cloud services, each of which provides a specific function. It offers a global network of data centers, known as Availability Zones, distributed across multiple geographic regions. These regions consist of multiple data centers to ensure high availability and fault tolerance.

Components

- **Compute Services:** AWS offers virtual servers through Amazon Elastic Compute Cloud (EC2), allowing users to launch instances with various configurations to run applications.
- **Storage Services:** Amazon Simple Storage Service (S3) provides scalable object storage, while Amazon Elastic Block Store (EBS) offers persistent block storage for EC2 instances.
- **Networking Services:** Amazon Virtual Private Cloud (VPC)

allows users to create isolated networks within the AWS cloud, while AWS Direct Connect offers dedicated network connections.

- **Database Services:** AWS provides managed database services, including Amazon RDS (Relational Database Service) and Amazon DynamoDB for NoSQL databases.

Services

- **Amazon EC2:** Offers resizable compute capacity in the cloud, allowing users to create and manage virtual machines.
- **Amazon S3:** Provides object storage with high durability and scalability, suitable for storing and retrieving large amounts of data.
- **AWS Lambda:** A serverless computing service that allows users to run code without provisioning or managing servers.
- **Amazon RDS:** Managed relational database service that supports multiple database engines like MySQL, PostgreSQL, and SQL Server.

Benefits

- **Scalability:** AWS allows users to scale resources up or down based on demand.
- **Flexibility:** A wide range of services and configurations to suit various use cases.
- **Global Reach:** Availability in multiple regions ensures low-latency access worldwide.

B. Microsoft Azure

Architecture: Azure is a collection of integrated cloud services that provide both Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) offerings. It consists of global data center regions with high availability and data redundancy.

Components

- **Compute Services:** Azure Virtual Machines (VMs) enable users to run Windows or Linux-based applications.

- **Storage Services:** Azure Blob Storage offers scalable object storage, while Azure Managed Disks provide persistent storage for VMs.
- **Networking Services:** Azure Virtual Network allows users to create isolated network environments, and Azure ExpressRoute provides dedicated connections.
- **Database Services:** Azure SQL Database and Azure Cosmos DB offer managed database solutions.

Services

- **Azure Virtual Machines:** Offers a wide range of VM sizes and configurations to support various workloads.
- **Azure App Service:** A platform for building, deploying, and scaling web apps.
- **Azure Functions:** Serverless computing service similar to AWS Lambda, enabling event-driven code execution.
- **Azure SQL Database:** Managed relational database service with built-in intelligence and security.

Benefits

- **Integration with Microsoft products:** Seamless integration with Microsoft software and tools.
- **Hybrid Capabilities:** Azure supports hybrid cloud scenarios, enabling on-premises and cloud resources to work together.
- **PaaS Offerings:** Provides a variety of PaaS services for application development and deployment.

C. Google Cloud Platform (GCP)

Architecture: GCP offers a global network of data centers, called regions and zones, to ensure high availability and data redundancy. It follows a layered architecture with a focus on open-source technologies.

Components

- **Compute Services:** Google Compute Engine provides virtual machine instances, while Google Kubernetes Engine offers managed Kubernetes clusters.
- **Storage Services:** Google Cloud Storage provides object storage with strong consistency and global scalability.
- **Networking Services:** Google Virtual Private Cloud (VPC) allows users to isolate network resources, and Google Cloud Load Balancing distributes traffic.

- **Database Services:** Google Cloud SQL offers managed database services, and Firestore provides a NoSQL document database.

Services

- **Google Compute Engine:** Offers customizable VM instances with various hardware configurations.
- **Google App Engine:** Platform for building and deploying applications without managing infrastructure.
- **Google Cloud Functions:** Serverless compute service for event-driven code execution.
- **Google Cloud SQL:** Managed relational database service with automatic backups and patch management.

Benefits

- **Data Analytics:** GCP provides data analytics and machine learning services through BigQuery and TensorFlow.
- **Scalability:** Supports auto-scaling of resources to meet varying workloads.
- **Open Source:** Embraces open-source technologies, promoting interoperability and flexibility.

Conclusion

Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) are leaders in the cloud computing space, offering a comprehensive set of services and tools for organizations to build, deploy, and manage applications and services. Each cloud operating system has its unique architecture, components, services, and benefits, catering to diverse business needs. As businesses increasingly adopt cloud technology, these platforms play a critical role in enabling digital transformation and innovation on a global scale.

Lab Manual Ends

❑❑❑