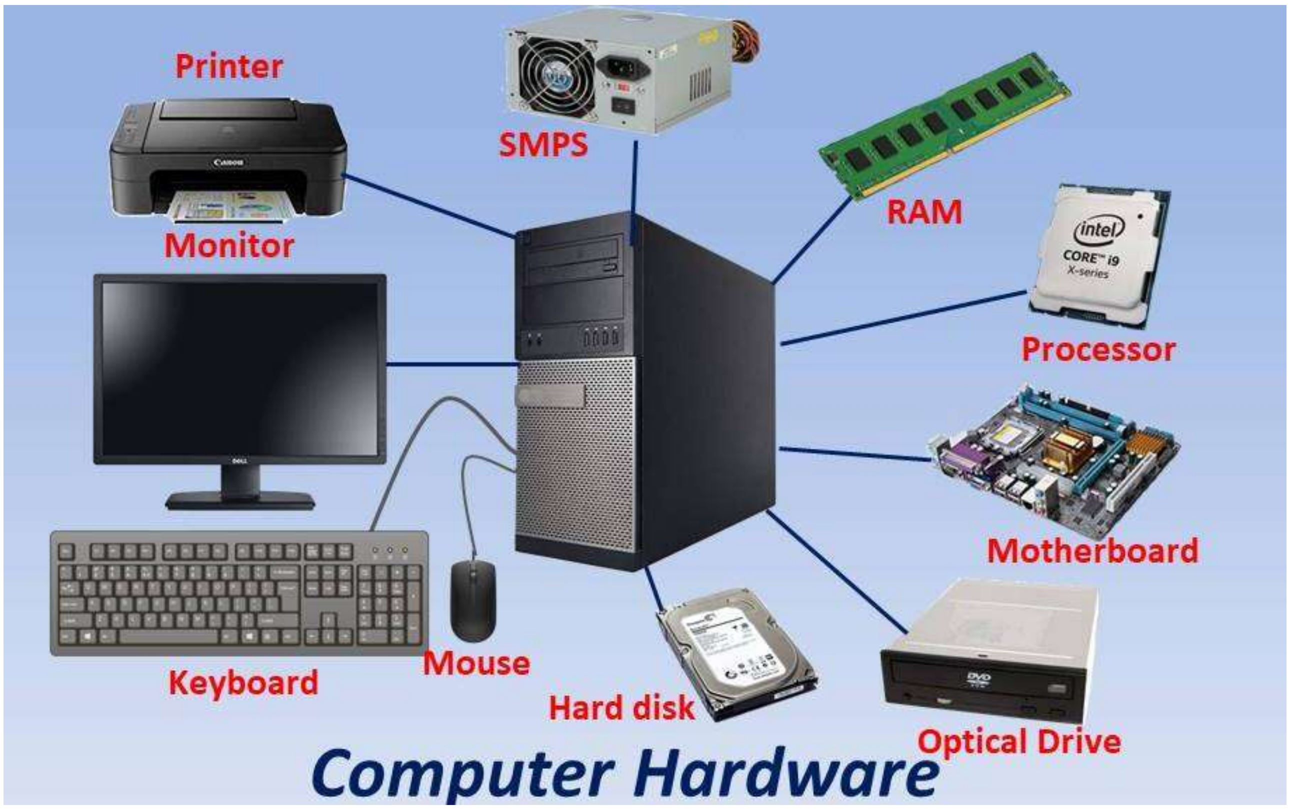
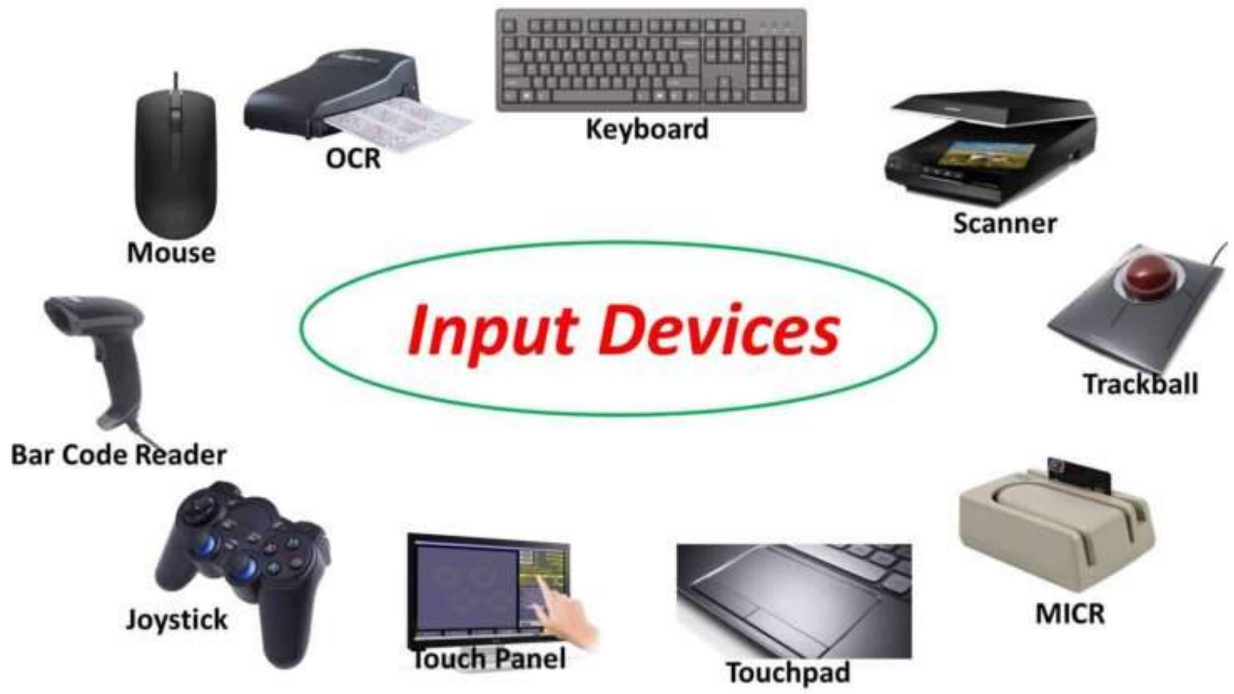
Date: 07/02/24

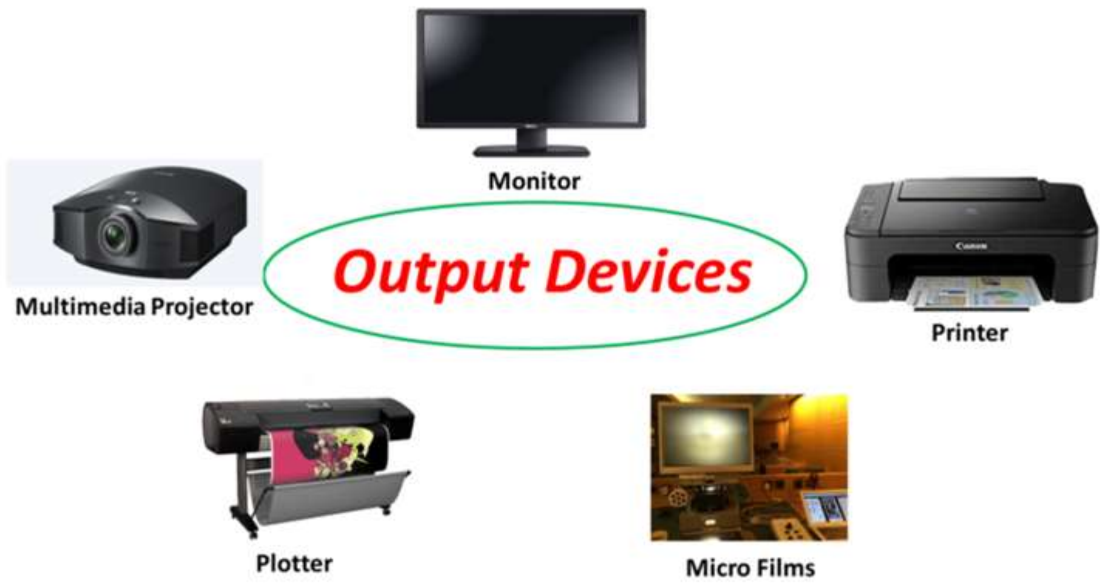
Experiment 1: **Introduction To Computer Hardware**



Input Devices:



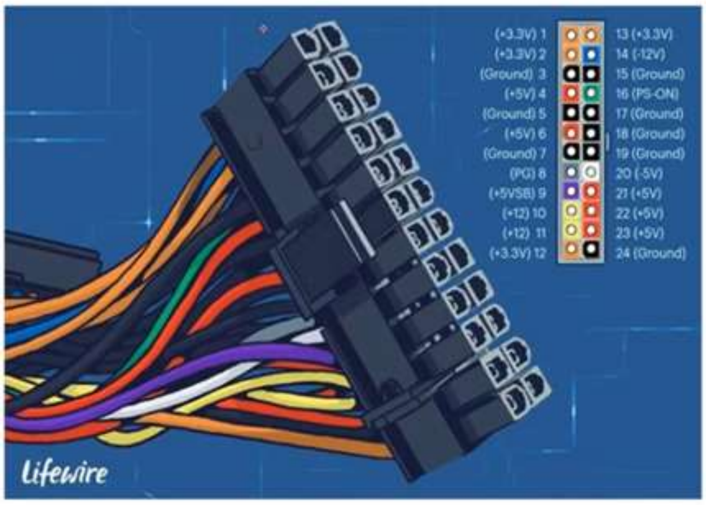
Output Devices:

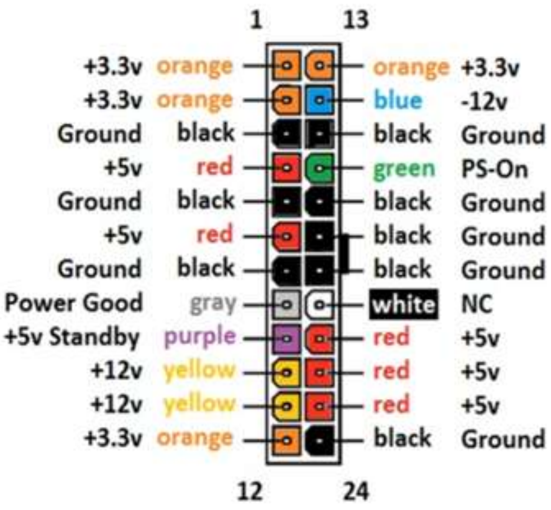


The Role of Software:

System software performs the basic functions to start and operate the hardware components of the computer system. They instruct the computer on how to operate its peripherals. In other words, system software deals with running the computer system or making it work.

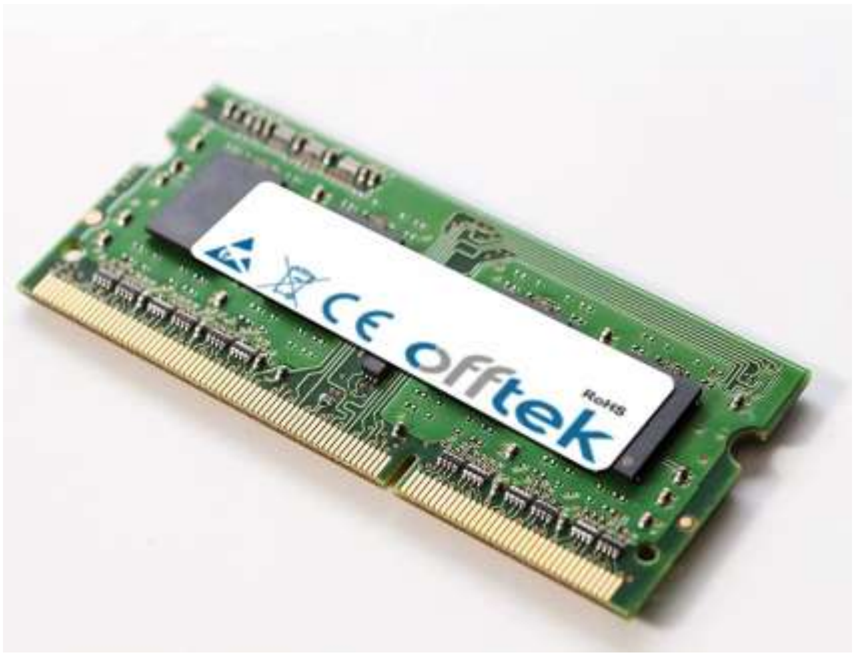
SMPS (Switched-Mode Power Supply):





What is RAM?

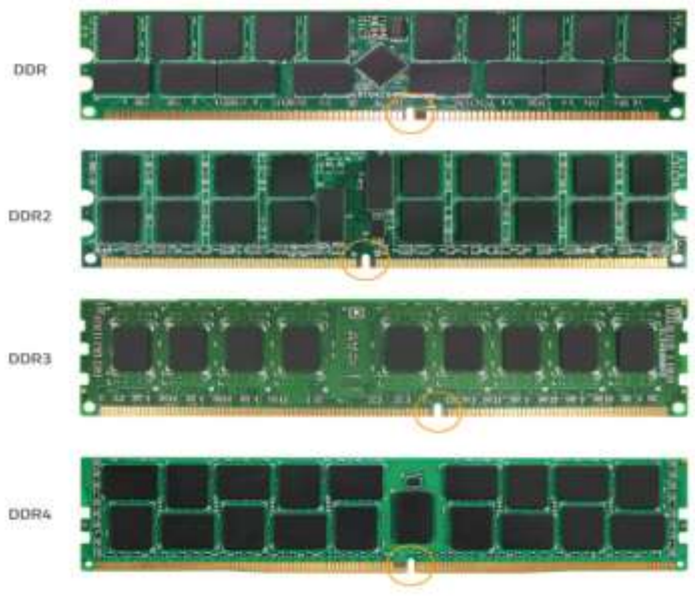
The full form of RAM is Random Access Memory. The information stored in this type of memory is lost when the power supply to the PC or laptop is switched off. The information stored in RAM can be checked with the help of BIOS. It is generally known as the main memory or temporary memory or cache memory or volatile memory of the computer system.



History of RAM:

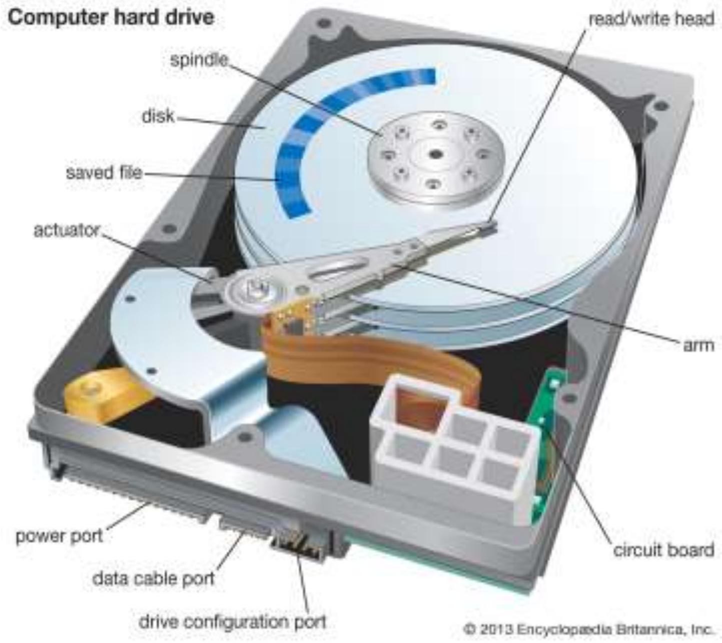
Here, are important landmarks from the history of RAM.

Type of RAM	Year Invented
FPM-(Fast page mode RAM)-	1990
EDO RAM (Extended data operations read-only memory)	1994
SDRAM (Single dynamic RAM)	1996
RDRAM (Rambus RAM)	1998
DDR (Double Data Rate)	2000
DDR2	2003
DDR3	2007
DDR4	2012



Processor:

Hard Disks:

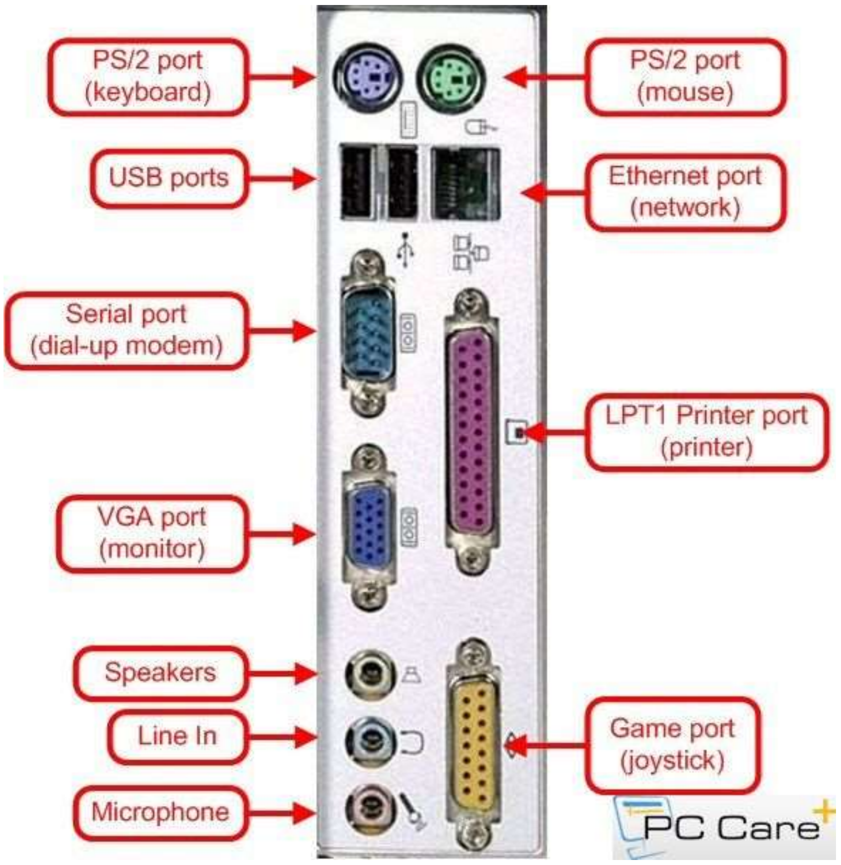


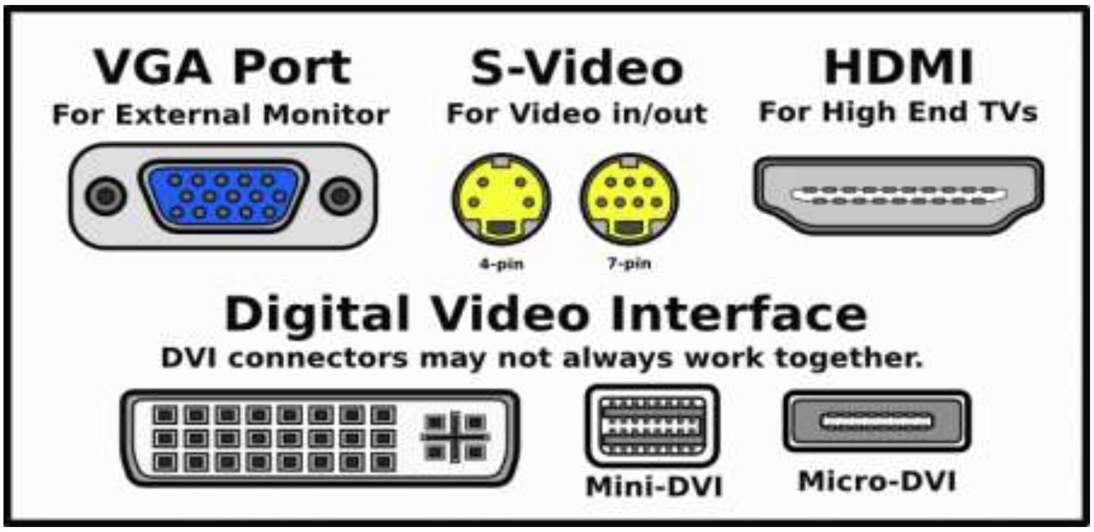
SSD (Solid State Drive):

Graphics Card:

GPUs can be used for video editing, 3D graphics rendering, and much more. With a high processing throughput, GPUs can process more data than their Central Processing Unit (CPU) counterparts, making them uniquely suited for highly demanding tasks such as machine learning and cryptocurrency mining.

Ports:



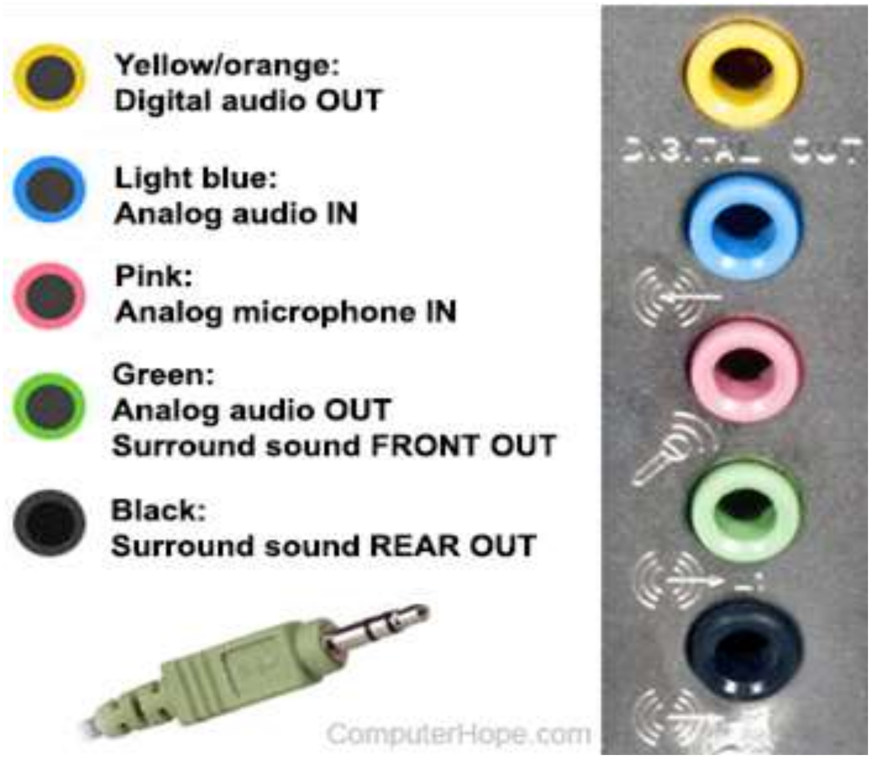


PS/2 cable

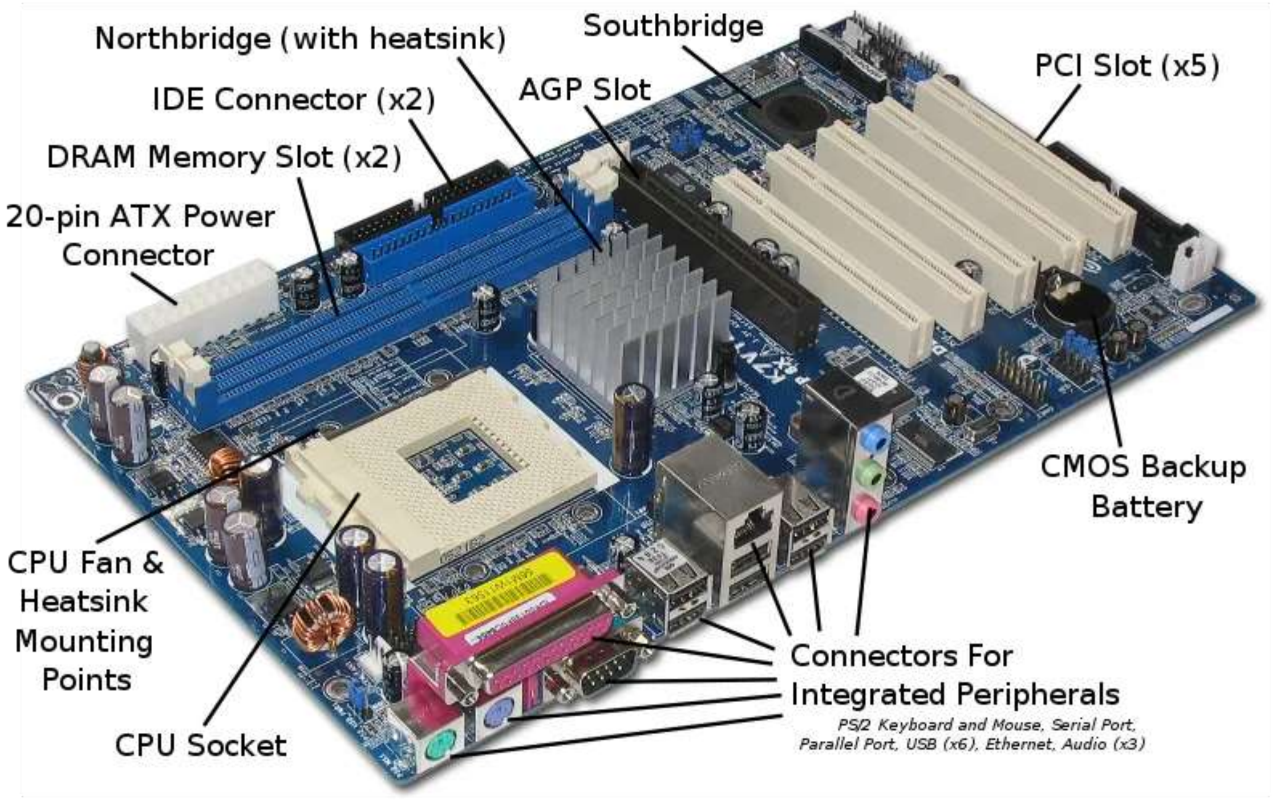
USB cable

Ethernet cable

Common Audio Connectors:



Major Motherboard Components and Their Functions:

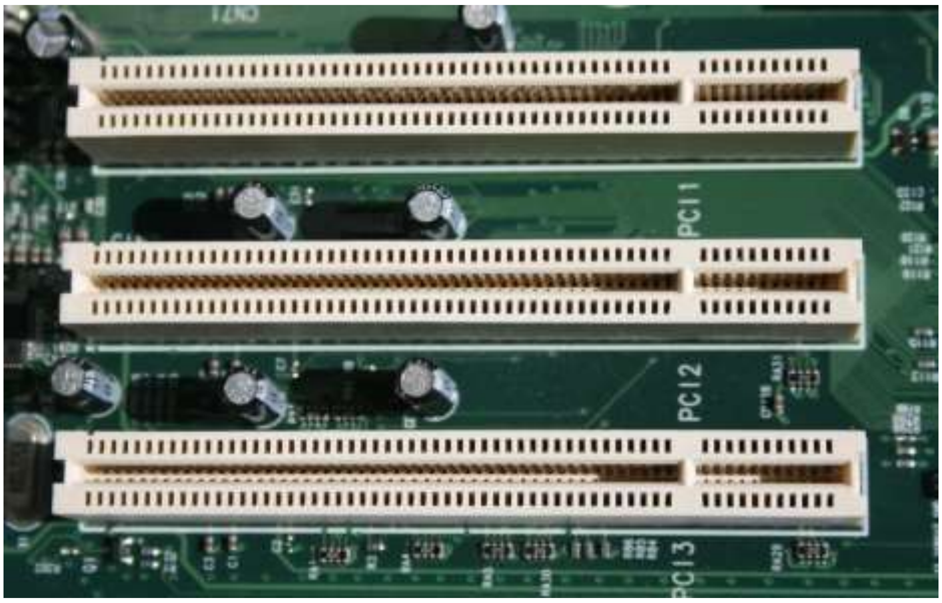

Northbridge (with heatsink)
Southbridge
IDE Connector (x2)
AGP Slot
PCI Slot (x5)
DRAM Memory Slot (x2)
20-pin ATX Power Connector
CMOS Backup Battery
CPU Fan & Heatsink Mounting Points
CPU Socket
Connectors For Integrated Peripherals
PS/2 Keyboard and Mouse, Serial Port, Parallel Port, USB (x6), Ethernet, Audio (x3)

CMOS Battery:

The CMOS RAM is used to store basic information about the PC's configuration for instance:-

- Floppy disk and hard disk drive types
- Information about CPU
- RAM size
- Date and time
- Serial and parallel port information
- Plug and Play information
- Power Saving settings

Expansion Bus (Network Cards, Audio Cards)



IDE & SATA:

IDE ribbon cable

SATA cable

North Bridge:

North bridge is one of the two chips located in the direction towards North in the motherboard. The main function of North bridge is to manage the communications between the Central Processing Unit and parts of motherboard. North bridge is directly towards Front Side Bus (FSB). Other names for North bridge are host bridge and Memory Controller Hub (MCH).

South Bridge:

South bridge is another chip of the logical chipset architecture. It is located to the South of Peripheral Component Interconnect (PCI) bus in the motherboard. The main function of South bridge is to control the IO functioning.

The North bridge is the medium that connects South bridge and Central Processing Unit. IO Controller Hub is the other name given to South bridge for its functionality.

Experiment 2: **Installation of Virtual Box**

VirtualBox is a powerful, open-source virtualization software developed by Oracle. It allows users to run multiple operating systems simultaneously on a single physical machine. When developing software, testing, and executing applications that might not be compatible with the host operating system, this feature is crucial. VirtualBox supports a wide range of guest operating systems, including various versions of Windows, Linux distributions, macOS, and others.

VirtualBox provides a user-friendly interface for creating and managing virtual machines (VMs), making it accessible to both novice and experienced users. Users can configure VM settings such as CPU, memory, storage, and network adapters to customize their virtual environment to suit their needs. VirtualBox also supports snapshots, allowing users to save the current state of a VM and revert back to it later if needed, making it ideal for testing and experimentation.

Operating Systems Supported by VirtualBox

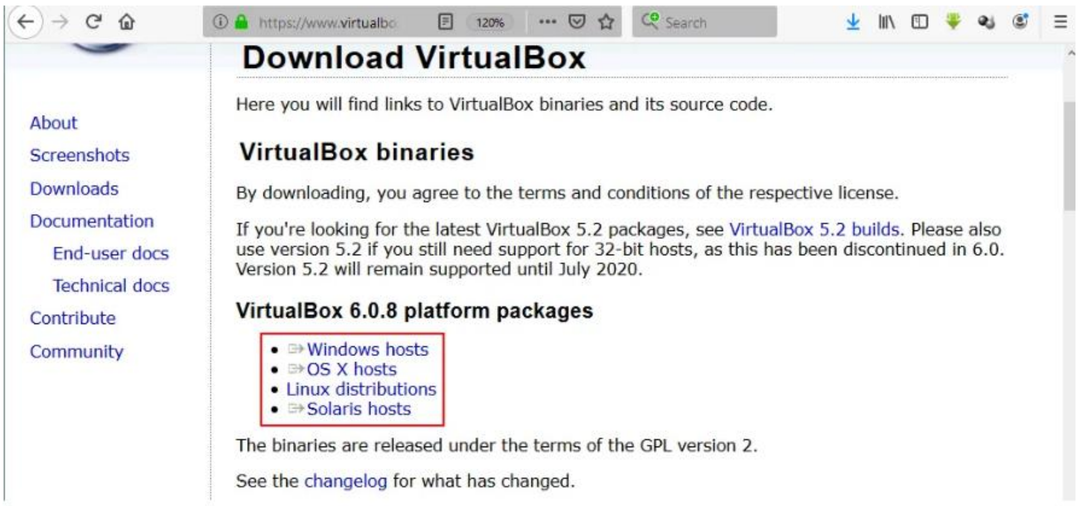
VirtualBox supports a long list of host and guest operating systems. A host OS is the operating system installed on a physical machine, on which VirtualBox is installed. A guest OS is an operating system installed on a virtual machine running inside VirtualBox. VirtualBox can be installed on Windows, Linux, macOS, Solaris, and FreeBSD. On VirtualBox you can run VMs with Windows, Linux, macOS, Solaris, FreeBSD, Novell Netware, and other operating systems.

Enable CPU virtualization features

To run 64-bit guest operating systems in some virtualization software, hardware virtualization features like Intel VT-X or AMD-V on the Central Processing Unit (CPU) must be enabled. If these features are disabled in the UEFI/BIOS settings, an error like "**VT-x is not available**" might occur. On Windows machines with Hyper-V installed, uninstall Hyper-V before installing VirtualBox. Hyper-V can block hardware virtualization extensions needed by VirtualBox to run virtual machines (VMs). Fortunately, most modern processors support hardware virtualization.

Download the VirtualBox installer

From the official website (*https://www.virtualbox.org/wiki/Downloads*) download the VirtualBox installer.



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.

If you're looking for the latest VirtualBox 5.2 packages, see VirtualBox 5.2 builds. Please also use version 5.2 if you still need support for 32-bit hosts, as this has been discontinued in 6.0. Version 5.2 will remain supported until July 2020.

VirtualBox 6.0.8 platform packages

- Windows hosts
- OS X hosts
- Linux distributions
- Solaris hosts

The binaries are released under the terms of the GPL version 2.

See the changelog for what has changed.

Run the installer and define the installation options

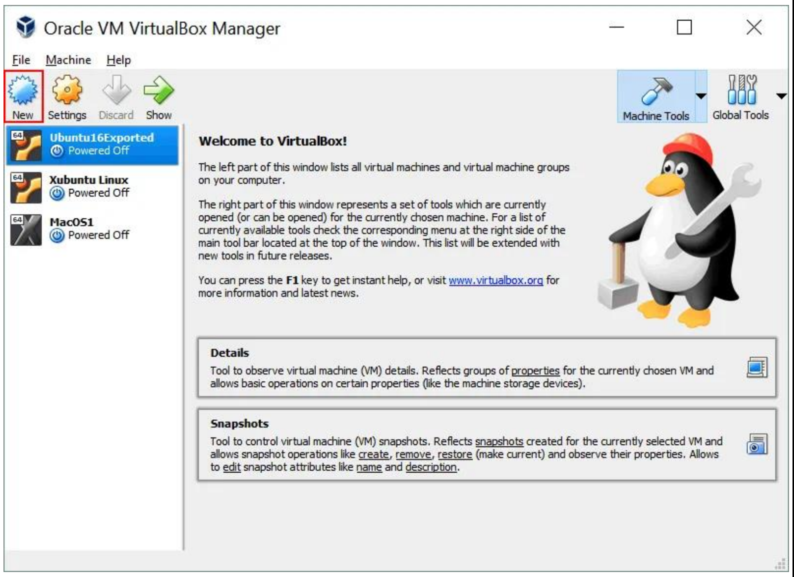
1. **Run the VirtualBox installer**. The installation wizard with a graphical user interface (GUI) will appear.
2. **Select installation options**. Leave the default values for the installation directory and installed components. Optionally, choose desired shortcut options and file associations by ticking the checkboxes.
3. **Confirm Network Interface Installation**. Click "Yes" to confirm installation of VirtualBox network interfaces.
4. **Start Installation**. On the "Ready to Install" screen, click "Install" to begin the process.
5. **Optional: Launch VirtualBox**. After installation finishes, a checkbox will be available to launch VirtualBox immediately. Tick the checkbox if desired.

Deploying a New VM

VirtualBox offers a unified graphical user interface (GUI) for all supported host operating systems.

Creating a Virtual Machine

Click **Machine** > **New** or hit the icon with the blue star to create a new virtual machine in VirtualBox GUI.



Welcome to VirtualBox!

The left part of this window lists all virtual machines and virtual machine groups on your computer.

The right part of this window represents a set of tools which are currently opened (or can be opened) for the currently chosen machine. For a list of currently available tools check the corresponding menu at the right side of the main tool bar located at the top of the window. This list will be extended with new tools in future releases.

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

Details

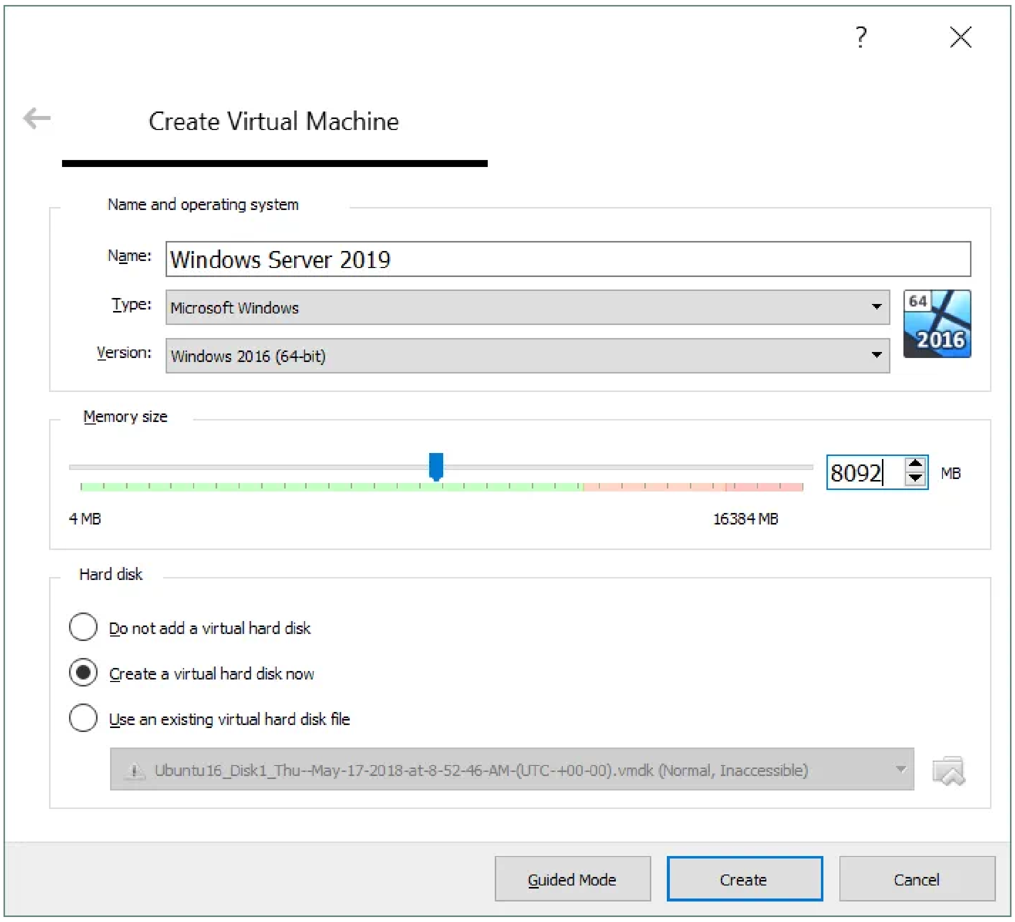
Tool to observe virtual machine (VM) details. Reflects groups of properties for the currently chosen VM and allows basic operations on certain properties (like the machine storage devices).

Snapshots

Tool to control virtual machine (VM) snapshots. Reflects snapshots created for the currently selected VM and allows snapshot operations like create, remove, restore (make current) and observe their properties. Allows to edit snapshot attributes like name and description.

Define the new VM options.

- ***Name***: WinServer2019.
- ***Type***: Microsoft Windows.
- ***Version***: Windows 2016 (64-bit). This parameter defines the reasonable default amount of virtual memory, virtual disk size; a set of emulated hardware (devices that are supported by the selected OS version, drivers for which are included); as well as a set of system features such as EFI, PAE (physical address extension), I/O APIC (input/output advanced programmable interrupt controllers). If you are using the latest version of VirtualBox, Windows 2019 is available in the list of OS versions.
- ***Memory size***: Set memory for the VM. 8 GB of RAM should be enough for Windows Server 2019 for the beginning. You can add more memory later, after installing the guest OS (a VM must be powered off to change the amount of memory).
- ***Hard Disk***: Create a virtual hard disk now.
- Click the **Create** button.



Create Virtual Machine

Name and operating system

Name: Windows Server 2019

Type: Microsoft Windows

Version: Windows 2016 (64-bit)

Memory size

8092 MB

4 MB 16384 MB

Hard disk

Do not add a virtual hard disk

[✓ Create a virtual hard disk now]

Use an existing virtual hard disk file

Ubuntu16_Disk1_Thu--May-17-2018-at-8-52-46-AM-(UTC-+00-00).vmdk (Normal, Inaccessible)

Creating a Virtual Hard Disk

Set the following parameters:

Name and file location for the virtual disk. Try not to use a system partition for storing virtual disks if possible.

The file size of the virtual disk. Select 50 GB for Windows Server 2019.

Hard disk file type. VirtualBox supports a lot of virtual disk formats:

- VDI (VirtualBox Disk Image) is a native VirtualBox format. Select this virtual disk type if you don't plan to migrate a VM to other platforms such as VMware of Hyper-V.
- VHD (Virtual Hard Disk) is a Hyper-V format.
- VMDK (Virtual Machine Disk) is the VMware virtual disk format.
- HDD is the Parallels Hard Disk.
- QCOW (QEMU Copy-On-Write).
- QED (QEMU enhanced disk).

Storage on physical hard disk: VirtualBox offers two options for virtual disk storage allocation: dynamically allocated and fixed size. These are analogous to thin and thick provisioning in VMware.

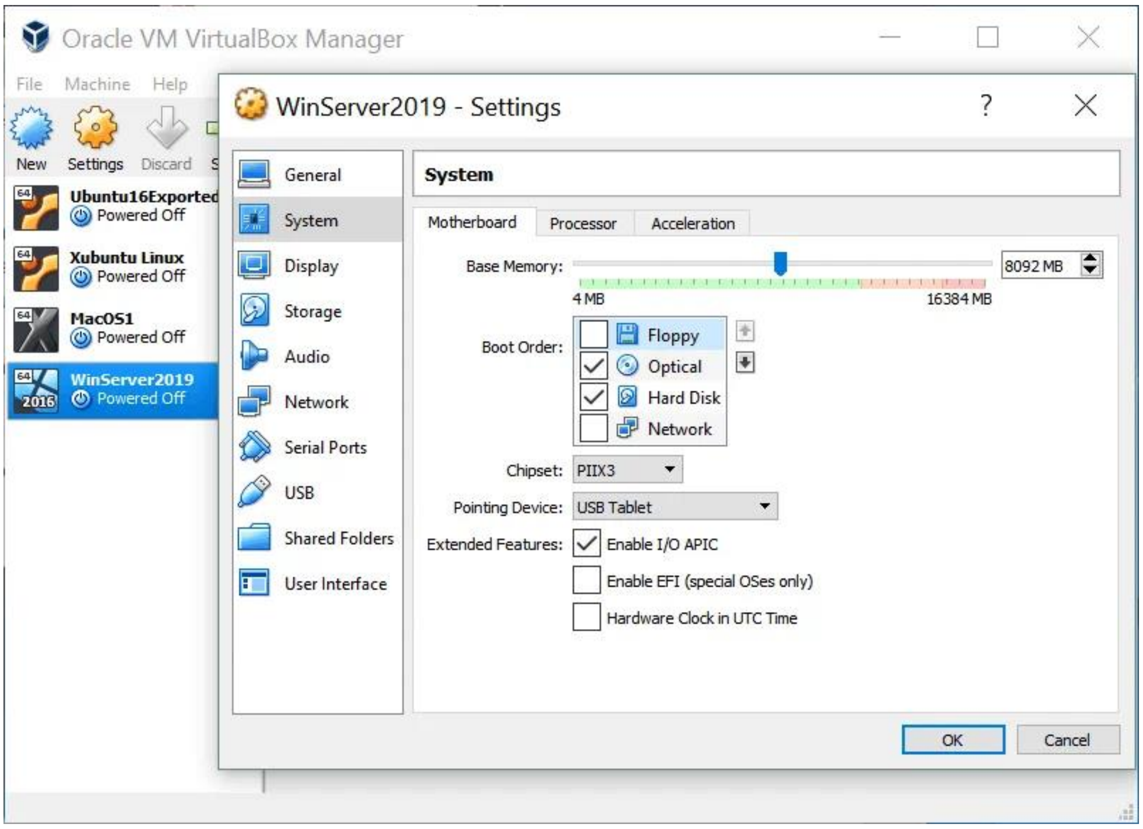
- **Dynamically Allocated:** This option saves disk space on the physical hard drive. Initially, the virtual disk occupies minimal space and expands as data is written to it, up to the maximum allocated size. This is a good choice when the initial virtual disk size is unknown or you want to optimize storage usage.
- **Fixed Size:** This option pre-allocates the entire maximum size of the virtual disk on the physical hard drive. This method offers potentially better performance but requires more upfront storage space.

Click Create to finish VM creation.

Virtual machine Tuning: Tune your virtual machine by going to *Machine* > *Settings*. The following sections are in this window.

General: VirtualBox allows editing VM name, enabling shared clipboard/drag & drop, writing descriptions, and encrypting virtual disks.

System: Virtual floppy drives can be disabled in the Motherboard tab, while the System tab lets you add processors, configure acceleration, and choose the emulated chipset for your virtual machine.



WinServer2019 - Settings

System

Base Memory: 8092 MB (4 MB – 16384 MB)

Boot Order: [ ] Floppy, [✓] Optical, [✓] Hard Disk, [ ] Network

Chipset: PIIX3

Pointing Device: USB Tablet

Extended Features: [✓] Enable I/O APIC; [ ] Enable EFI (special OSes only); [ ] Hardware Clock in UTC Time

Display: The Screen tab allows setting video memory, monitor count, scale factor, and enabling 2D/3D acceleration. The Remote Display tab facilitates managing the guest OS by connecting remotely to the VirtualBox VM desktop via RDP (Remote Desktop Protocol). Video capturing is configured in the Video Capture tab.

Storage: Virtual hard disks, virtual DVD drives, and disk controllers can be added or removed as needed. To install the operating system, select the empty DVD drive and insert the virtual ISO by clicking the disc icon and choosing "**Choose virtual optical disc file.**" Browse and open the WinServer2019.iso file. This will add the ISO to the list of storage devices.

Audio: Audio can be enabled or disabled; the host audio driver, audio controller and extended features can be selected.

Network: Virtual network adapters are configured in this section. The maximum number of virtual network adapters per VM is four. A virtual network adapter can use a variety of different network modes: Not attached, NAT, NAT Network, Bridged Adapter, Internal

Network, Host-only Adapter, Generic Driver, among which the NAT network mode is used by default.

Selecting the right network mode for a virtual machine (VM) depends on its intended use. NAT mode connects the VM to a virtual router, allowing access to the host machine, the host's network, and external networks reachable by the host. This is ideal for standalone VMs that don't require direct visibility on the physical network. Conversely, bridged mode makes the VM appear as a regular machine on the physical network. This mode is suitable for VMs needing full network integration and visibility within the physical network environment.

Serial Ports: Enable Serial ports if for some particular reason COM ports need to be enabled on a VM.

USB: USB options for a VM are configured in this section.

Shared Folders: Shared folders are used for file exchange between host OS and guest OS.

User interface: Customize the elements of GUI if needed.

Click **OK** to apply the edited VM configuration.

Installing a guest OS

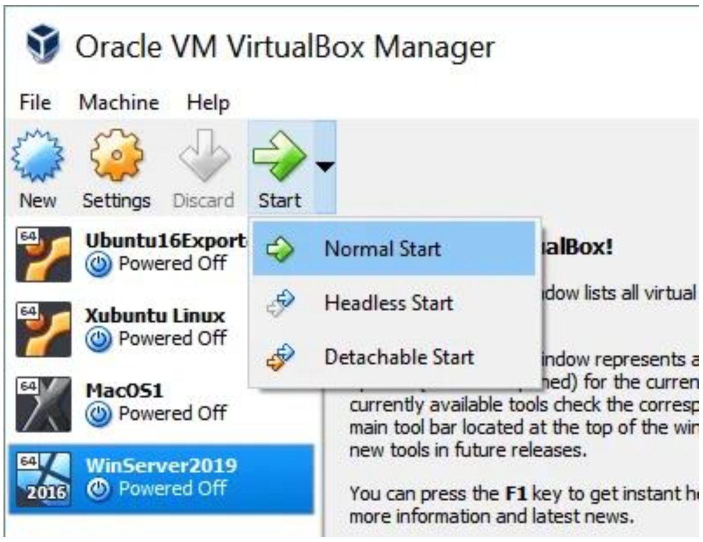
Now to start the VM. Hit ***Machine > Start > Normal Start.***

Normal Start: Opens a VM window and displays the video output of the VM in that window similarly as the output of the physical machine is displayed on a monitor. Closing the window prompts VirtualBox with these options:

- **Save the machine state**: The VM is hibernated (a VM is on pause). Start the VM to continue VM operation from the saved state.
- **Send the shutdown signal:** The VM shuts down gracefully, similar to shutting down a physical machine from its operating system.
- **Power Off the machine:** This option is the equivalent of unplugging the power cable from the physical computer.

Headless Start**:** In some cases, a virtual machine (VM) might start without displaying its video output window. However, the VM itself is still running. For management purposes, the VM can be accessed remotely using protocols like VRDP (VirtualBox Remote Display Protocol, backward compatible with Microsoft RDP) or SSH.

Detachable start: Combines normal and headless start modes. Closing the VirtualBox VM window offers an additional option: "Continue running in the background." This allows closing the window without interrupting the VM's operation.

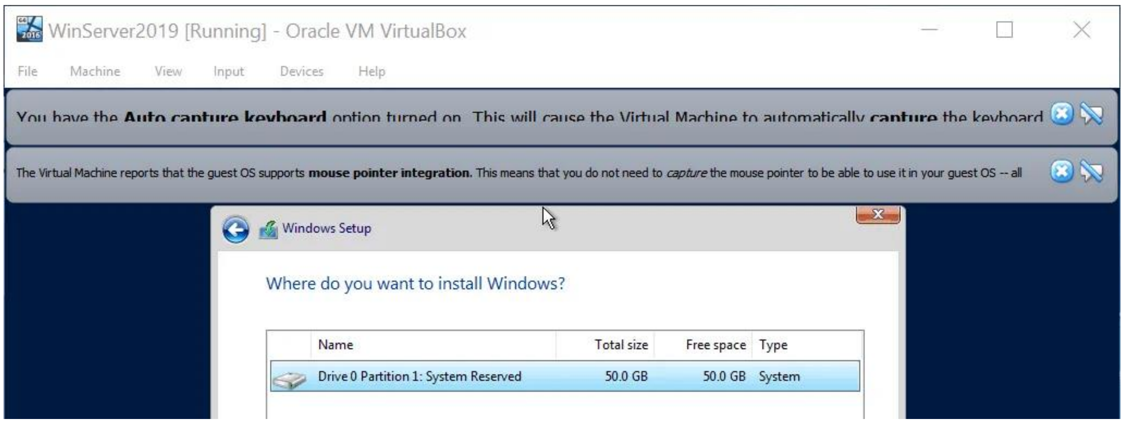


The OS installer is now booting from the ISO image inserted to a virtual DVD drive. This process is displayed in the new VirtualBox VM window.

Select suitable options for the Windows installation wizard:

- Windows Server 2019 Datacenter Evaluation (Desktop experience).
- Custom: Install Windows only (advanced).

Create a new partition (or partitions) on the 50-GB virtual hard disk.



You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard

The Virtual Machine reports that the guest OS supports **mouse pointer integration**. This means that you do not need to *capture* the mouse pointer to be able to use it in your guest OS -- all

Where do you want to install Windows?

	Name	Total size	Free space	Type
	Drive 0 Partition 1: System Reserved	50.0 GB	50.0 GB	System

The VM automatically restarts a few times during Windows Server 2019 installation.

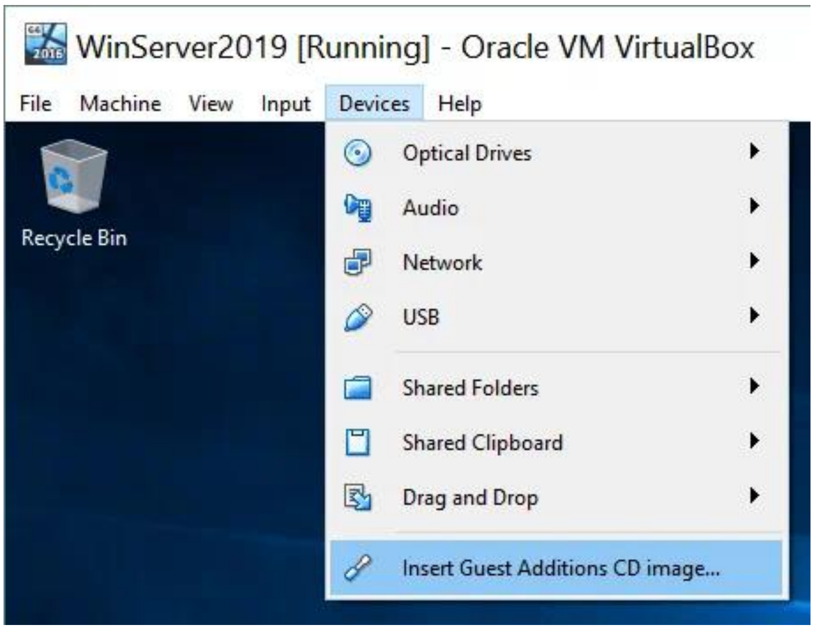
Set the Windows administrator password to finish Windows Server 2019 setup on VirtualBox.

After loading, Windows asks to press **Ctrl+Alt+Delete** to unlock.

Click **Input > Keyboard > Insert Ctrl+Alt+Del** in the VirtualBox VM window.

VirtualBox Guest Additions, a set of drivers and utilities, optimize performance and usability for supported guest operating systems. They also improve interaction between the host and guest environments. Features like shared folders, clipboard, time synchronization, and enhanced video modes become available after installing Guest Additions on the guest OS. The installation process occurs within the guest system after logging in. The Guest Additions ISO file resides in the VirtualBox installation directory.

In the VirtualBox VM window, click **Devices > Insert Guest Additions CD image**. The virtual ISO disc is now inserted to the virtual DVD drive of your Windows VM.



Open the contents of the disc and run the ***VBoxWindowsAdditions-amd64.exe*** file. After the installation wizard opens, follow the tips of the wizard recommendations, clicking *Next* on each step to continue. In the end of installation, reboot the virtual machine.

Date: 14/02/24

Experiment 3: **Basic Linux Commands**

1) **man:**

The **man** command, short for manual, is a powerful tool in the Linux operating system that allows users to access detailed information about various commands, utilities, and system calls. The "man" command provides comprehensive documentation, helping users understand how to use and configure different elements of the Linux environment.

Syntax:

```
man [option] [command]
```

- **option** refers to additional flags that modify the behavior of the "man" command.

Options	Description
-f	Display a concise one-line description of the command.
-k, –apropos	Search for commands related to a given keyword.
-a, –all	Display all matching manual pages for the specified command.

- **command** is the Linux command or utility for which you want to access the manual.

Sections in Manual Pages:

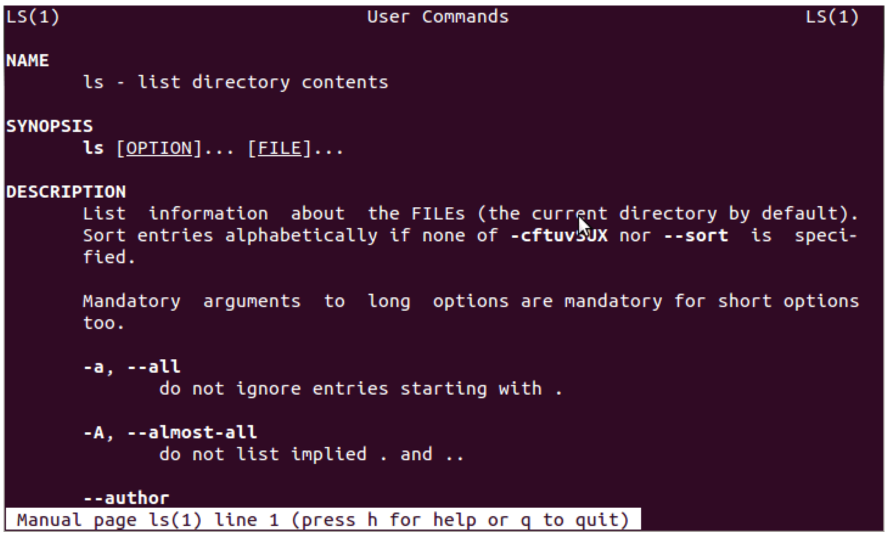
Manual pages are organized into different sections, each serving a specific purpose. The primary sections include:

- **NAME:** Provides the name and a brief description of the command.
- **SYNOPSIS:** Describes the syntax of the command.
- **DESCRIPTION:** Offers a detailed explanation of the command's functionality.
- **OPTIONS:** Lists the available command-line options and their descriptions.
- **EXAMPLES:** Provides practical examples demonstrating command usage.

- **SEE ALSO:** Suggests related commands or resources.

Example:

```
man ls
```



```
LS(1)                          User Commands                         LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speci-
       fied.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
 Manual page ls(1) line 1 (press h for help or q to quit)
```

2) <u>ls:</u>

The **ls** command line utility lists all the files and directories under a specified directory. It provides valuable information about files, directories, and their attributes. By default, ls uses the current directory and lists files and directories in alphabetical order by name.

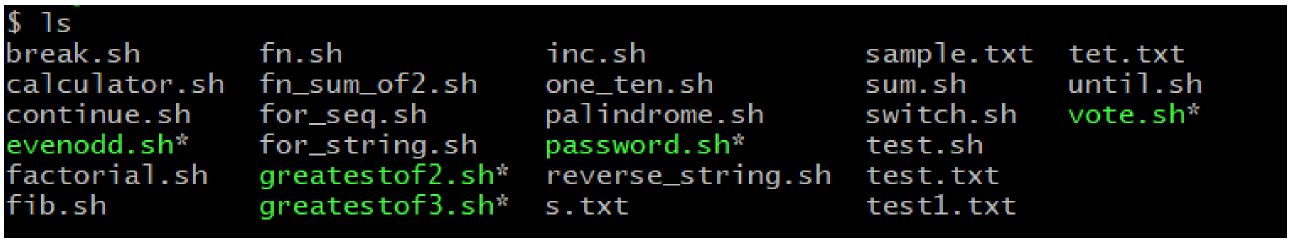
Syntax:

```
ls [option] [file/directory]
```

Option	Description
-l	known as a long format that displays detailed information about files and directories.
-a	Represent all files Include hidden files and directories in the listing.
-t	Sort files and directories by their last modification time, displaying the most recently modified ones first.
-r	known as reverse order which is used to reverse the default order of listing.
-S	Sort files and directories by their sizes, listing the largest ones first.

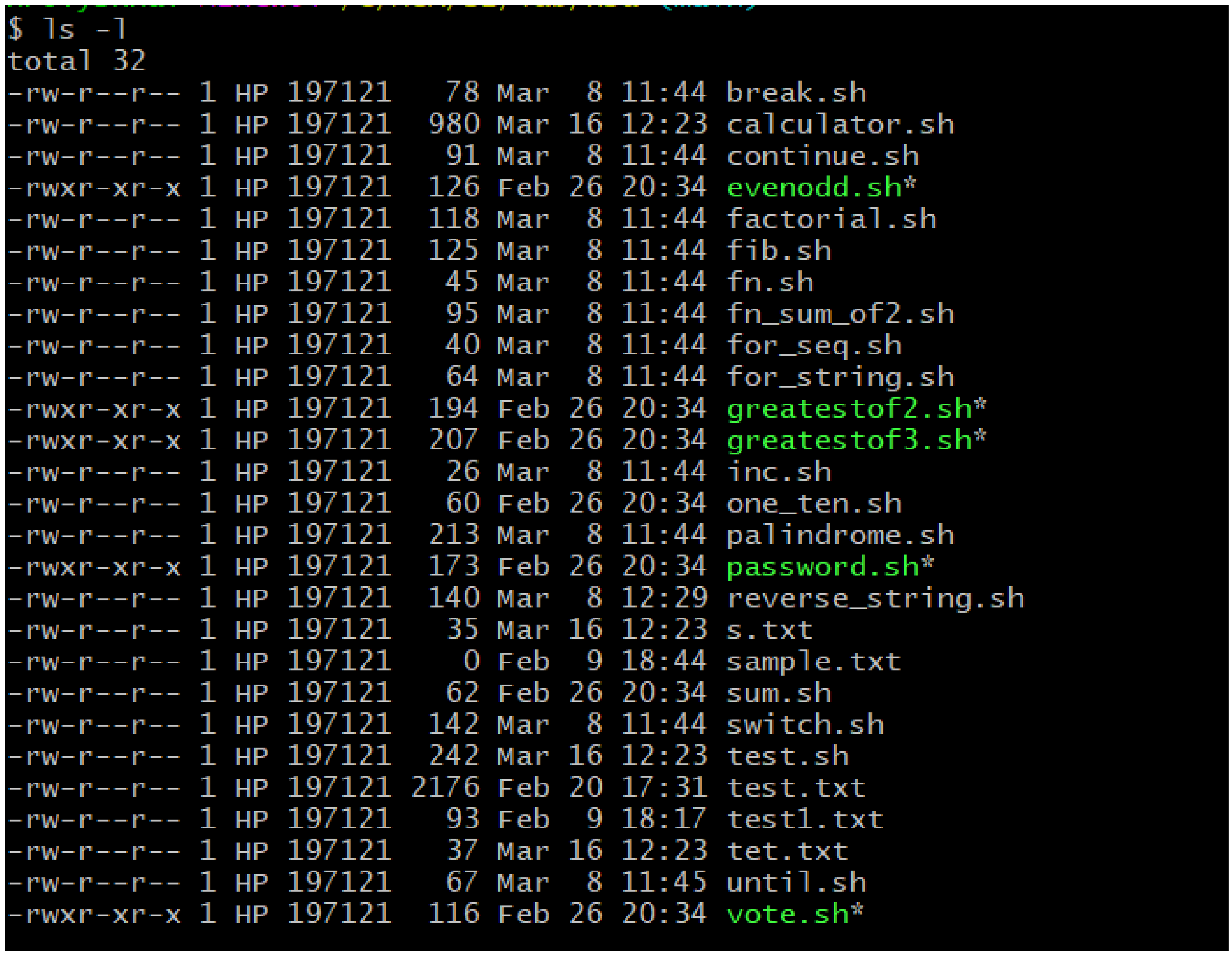
Example:

```
ls
```



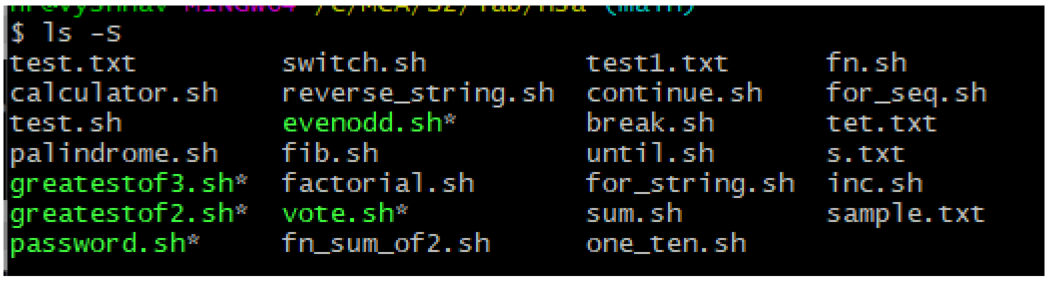
```
$ ls
break.sh        fn.sh            inc.sh             sample.txt  tet.txt
calculator.sh   fn_sum_of2.sh    one_ten.sh         sum.sh      until.sh
continue.sh     for_seq.sh       palindrome.sh      switch.sh   vote.sh*
evenodd.sh*     for_string.sh    password.sh*       test.sh
factorial.sh    greatestof2.sh*  reverse_string.sh  test.txt
fib.sh          greatestof3.sh*  s.txt              test1.txt
```

```
ls -l
```



```
$ ls -l
total 32
-rw-r--r-- 1 HP 197121   78 Mar  8 11:44 break.sh
-rw-r--r-- 1 HP 197121  980 Mar 16 12:23 calculator.sh
-rw-r--r-- 1 HP 197121   91 Mar  8 11:44 continue.sh
-rwxr-xr-x 1 HP 197121  126 Feb 26 20:34 evenodd.sh*
-rw-r--r-- 1 HP 197121  118 Mar  8 11:44 factorial.sh
-rw-r--r-- 1 HP 197121  125 Mar  8 11:44 fib.sh
-rw-r--r-- 1 HP 197121   45 Mar  8 11:44 fn.sh
-rw-r--r-- 1 HP 197121   95 Mar  8 11:44 fn_sum_of2.sh
-rw-r--r-- 1 HP 197121   40 Mar  8 11:44 for_seq.sh
-rw-r--r-- 1 HP 197121   64 Mar  8 11:44 for_string.sh
-rwxr-xr-x 1 HP 197121  194 Feb 26 20:34 greatestof2.sh*
-rwxr-xr-x 1 HP 197121  207 Feb 26 20:34 greatestof3.sh*
-rw-r--r-- 1 HP 197121   26 Mar  8 11:44 inc.sh
-rw-r--r-- 1 HP 197121   60 Feb 26 20:34 one_ten.sh
-rw-r--r-- 1 HP 197121  213 Mar  8 11:44 palindrome.sh
-rwxr-xr-x 1 HP 197121  173 Feb 26 20:34 password.sh*
-rw-r--r-- 1 HP 197121  140 Mar  8 12:29 reverse_string.sh
-rw-r--r-- 1 HP 197121   35 Mar 16 12:23 s.txt
-rw-r--r-- 1 HP 197121    0 Feb  9 18:44 sample.txt
-rw-r--r-- 1 HP 197121   62 Feb 26 20:34 sum.sh
-rw-r--r-- 1 HP 197121  142 Mar  8 11:44 switch.sh
-rw-r--r-- 1 HP 197121  242 Mar 16 12:23 test.sh
-rw-r--r-- 1 HP 197121 2176 Feb 20 17:31 test.txt
-rw-r--r-- 1 HP 197121   93 Feb  9 18:17 test1.txt
-rw-r--r-- 1 HP 197121   37 Mar 16 12:23 tet.txt
-rw-r--r-- 1 HP 197121   67 Mar  8 11:45 until.sh
-rwxr-xr-x 1 HP 197121  116 Feb 26 20:34 vote.sh*
```

```
ls -S
```



```
$ ls -S
test.txt         switch.sh          test1.txt       fn.sh
calculator.sh    reverse_string.sh  continue.sh     for_seq.sh
test.sh          evenodd.sh*        break.sh        tet.txt
palindrome.sh    fib.sh             until.sh        s.txt
greatestof3.sh*  factorial.sh       for_string.sh   inc.sh
greatestof2.sh*  vote.sh*           sum.sh          sample.txt
password.sh*     fn_sum_of2.sh      one_ten.sh
```

3) echo:

The **echo** command in Linux is a built-in command that allows users to display lines of text or strings that are passed as arguments. It is commonly used in shell scripts and batch files to output status text to the screen or a file.

Syntax:

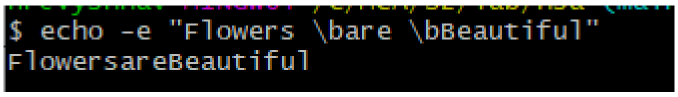
```
echo [option] [string]
```

[string] => It is the string that we want to display.

Option	Description
-n	Do not output the trailing newline
-e	Enable interpretation of backslash escapes.
-E	Sort files and directories by their last modification time, displaying the most recently modified ones first.

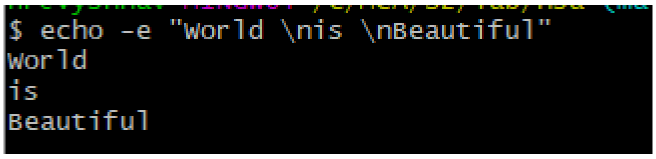
Example:

```
echo -e "Flowers \bare \bBeautiful"
```



```
$ echo -e "Flowers \bare \bBeautiful"
FlowersareBeautiful
```

```
echo -e "World \nis \nBeautiful"
```



```
$ echo -e "World \nis \nBeautiful"
World
is
Beautiful
```

Redirecting echo to a file instead of displaying it.

```
echo "Welcome" > output.txt
```

This will write the output of the echo command to the file name `output.txt`. File will be overwritten if it already exists.

4) read:

The **read** command in Linux is an in-built command-line utility that is primarily used for accepting user input or reading a line of input from a standard input and storing it into a variable.

Syntax:

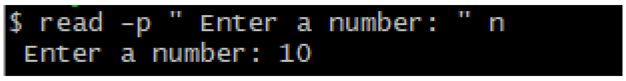
```
read [options] [name...]
```

[name]: The variables where the input from the read command will be stored. If more than one variable name is given, the line of input is split into words and each word is assigned to a variable.

Option	Description
-p	Displays a prompt to the user before reading the input.
-t	Sets a timeout for the read command. If no input is provided within the specified number of seconds, the command ends.
-a	Reads the input into an array instead of a single variable.
-s	Used for **security purpose**. It is used to read the sensitive data. By using this option, the entered text won't appear in the terminal.
-n	Limits the length of the character in the entered text.

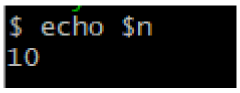
Example:

```
read -p " Enter a number: "  n
```



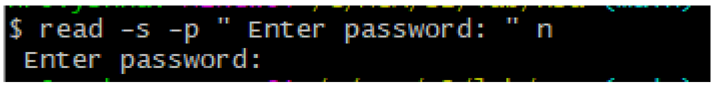
```
$ read -p " Enter a number: " n
 Enter a number: 10
```

```
echo $n
```



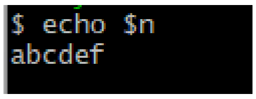
```
$ echo $n
10
```

```
read -s -p " Enter password: " n
```



```
$ read -s -p " Enter password: " n
 Enter password:
```

```
echo $n
```



```
$ echo $n
abcdef
```

5) cat:

Cat is short for concatenate. This command displays the contents of one or more files without having to open the file for editing.

Syntax:

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

[FILE] : the name of the file(s) to be processed.

Option	Description
-n	To enable line numbering.
-e	Highlight the end of each line and spaces between lines with **$.**
-t	Displaying the file content along with the tab space within the text.
-s	To omit blank lines from the output of cat

Examples:

- **Create a new file**

  ```
  cat >test1.txt
  ```

  The cursor moves to a new line where you can add the wanted text.

- **Display contents of a single file**

  ```
  cat test1.txt
  ```

- **Display contents of a multiple file**

  ```
  cat test1.txt test2.txt
  ```

- **Redirect contents of a single file**

  ```
  cat test1.txt > test3.txt
  ```

  If the destination filename doesn’t exist, it will be created.
  If a file is exported that already exists, this will **overwrite the contents of the file.**

- **Append file contents of another file**

  ```
  cat test1.txt >> test3.txt
  ```

  Adds the contents of a file to the end of another file.

- **Combining Operations**

```
cat test1.txt test2.txt > test4.txt
```

To combine the output of two files, and store the result in a new file.

```
cat test1.txt test2.txt >> test4.txt
```

To combine the output of two files, and append to the end of an existing file.

6) cd:

The **cd** stands for change directory. It is used to change the current working directory. Cd can be used to modify into a subdirectory, return to the parent directory, move every way behind the root directory, or move to a given directory.

Syntax:

```
cd [directory_name]
```

Examples:

- **Change directory using a relative path**

```
cd Desktop
```

A relative path is a location that is relative to the current directory. Change our current working directory to Desktop.

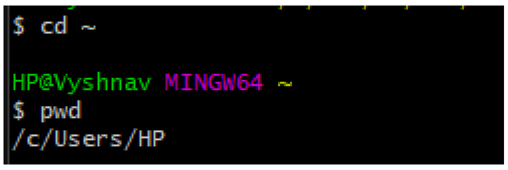
- **Change directory using an absolute path**

```
cd C:/Users/HP/Desktop
```

To change the directory by using an absolute path, we have to mention the whole path starting from the root. Change our current working directory to Desktop.

- **Change to home directory**

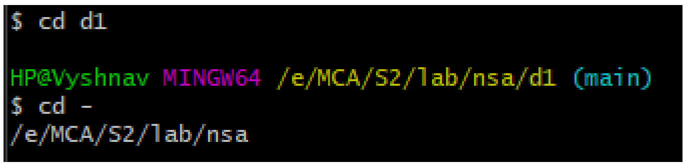
```
cd ~
```



```
$ cd ~

HP@Vyshnav MINGW64 ~
$ pwd
/c/Users/HP
```

- **Change to previous directory**

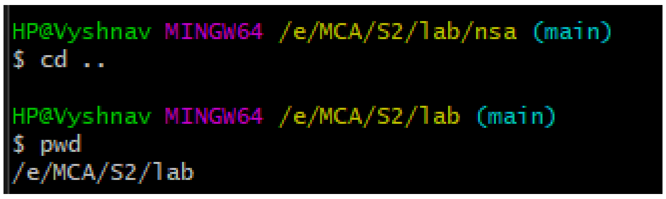
```
cd -
```



```
$ cd d1

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa/d1 (main)
$ cd -
/e/MCA/S2/lab/nsa
```

- **Change to parent directory**

```
cd ..
```



```
HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa (main)
$ cd ..

HP@Vyshnav MINGW64 /e/MCA/S2/lab (main)
$ pwd
/e/MCA/S2/lab
```

- **Change to another user's home directory**

```
cd ~username
```

7) mkdir:

The **mkdir** command in Linux is a command-line utility that allows users to create new directories. **mkdir** stands for **make directory**. With **mkdir**, you can also set permissions, create multiple directories at once, and much more.

Syntax:

```
mkdir [options...] [directory_name]
```

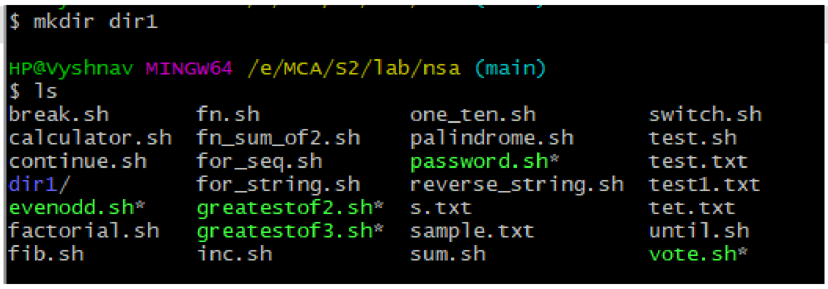
[dir_name]=> is the name or names of the directories you want to create. You can specify multiple directory names separated by spaces.

Option	Description
-p	Creates parent directories if they don't exist.
-m a=[rwx]	Sets the file modes, i.e., permissions, for the new directory.
-v	Displays a message for each created directory.
-z	It sets the SELinux security context of all created directories to their default type.

Examples:

- **Create new directory**

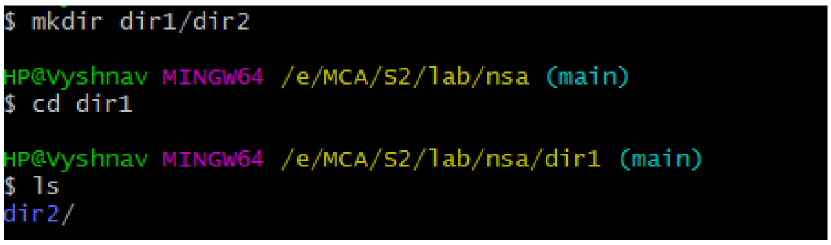
```
mkdir dir1
```



```
$ mkdir dir1

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa (main)
$ ls
break.sh        fn.sh           one_ten.sh          switch.sh
calculator.sh   fn_sum_of2.sh   palindrome.sh       test.sh
continue.sh     for_seq.sh      password.sh*        test.txt
dir1/           for_string.sh   reverse_string.sh   test1.txt
evenodd.sh*     greatestof2.sh* s.txt               tet.txt
factorial.sh    greatestof3.sh* sample.txt          until.sh
fib.sh          inc.sh          sum.sh              vote.sh*
```

- **Create directory in specific Location**

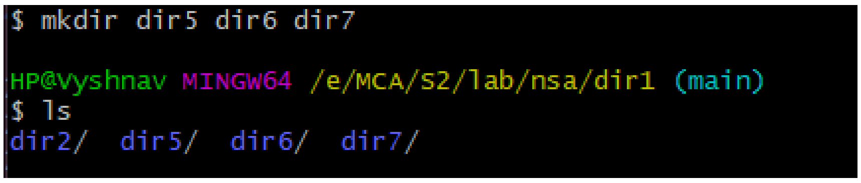
```
mkdir /dir1/dir2
```



```
$ mkdir dir1/dir2

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa (main)
$ cd dir1

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa/dir1 (main)
$ ls
dir2/
```

- **Create multiple directories**

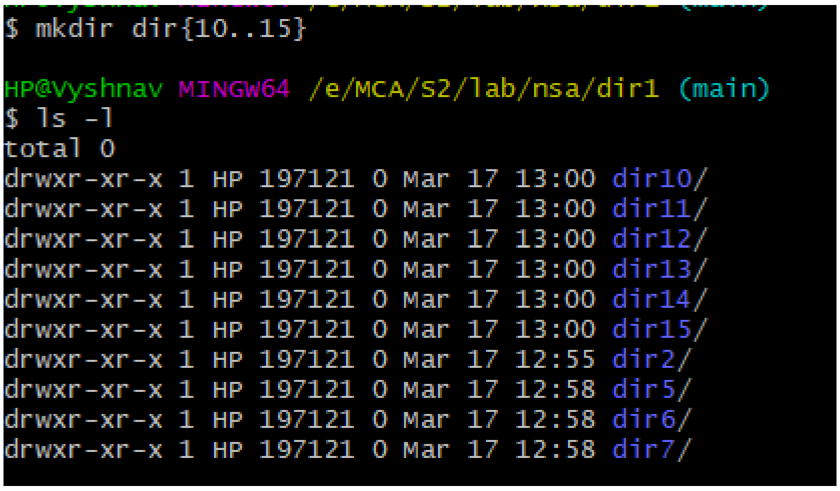
```
mkdir dir5 dir6 dir7
```



```
$ mkdir dir5 dir6 dir7

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa/dir1 (main)
$ ls
dir2/  dir5/  dir6/  dir7/
```

```
mkdir dir{10..15}
```

The command above creates 5 directories, from dir10 to dir15.



```
$ mkdir dir{10..15}

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa/dir1 (main)
$ ls -l
total 0
drwxr-xr-x 1 HP 197121 0 Mar 17 13:00 dir10/
drwxr-xr-x 1 HP 197121 0 Mar 17 13:00 dir11/
drwxr-xr-x 1 HP 197121 0 Mar 17 13:00 dir12/
drwxr-xr-x 1 HP 197121 0 Mar 17 13:00 dir13/
drwxr-xr-x 1 HP 197121 0 Mar 17 13:00 dir14/
drwxr-xr-x 1 HP 197121 0 Mar 17 13:00 dir15/
drwxr-xr-x 1 HP 197121 0 Mar 17 12:55 dir2/
drwxr-xr-x 1 HP 197121 0 Mar 17 12:58 dir5/
drwxr-xr-x 1 HP 197121 0 Mar 17 12:58 dir6/
drwxr-xr-x 1 HP 197121 0 Mar 17 12:58 dir7/
```

8) pwd:

pwd stands for Print Working Directory. It writes the complete path name of the working directory to standard output.

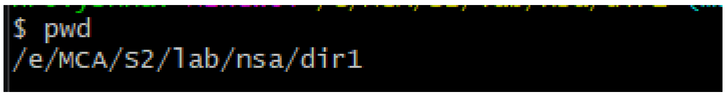
Syntax:

```
pwd [OPTIONS]
```

Option	Description
-L	Prints the symbolic path.
-P	Prints the actual path.

Example:

```
pwd
```



```
$ pwd
/e/MCA/S2/lab/nsa/dir1
```

9) mv:

Linux **mv** command is used to move existing file or directory from one location to another. It is also used to rename a file or directory.

Syntax:

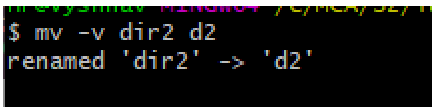
```
mv [options] [source-file] [destination-file]
```

Option	Description
-b	Create a backup of files that will be overwritten or removed.
-f	Overwrite destination files without prompting the user.
-i	Prompt the user to confirm if the **mv** action should overwrite a file.
-S	Provide a suffix for a backup file. The default suffix is ~.
-u	Perform the **mv** action only if the source file is newer or the destination file does not exist.
-v	Show an output describing the action taken.

Examples:

- **Rename file**

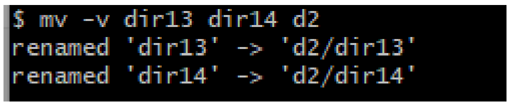
```
mv -v dir2 d2
```



```
$ mv -v dir2 d2
renamed 'dir2' -> 'd2'
```

- **Move multiple file**

```
mv -v dir3 dir4 ~/d2
```



```
$ mv -v dir13 dir14 d2
renamed 'dir13' -> 'd2/dir13'
renamed 'dir14' -> 'd2/dir14'
```

10) cp:

The **cp** command is used for creating copies of files and directories across a filesystem.

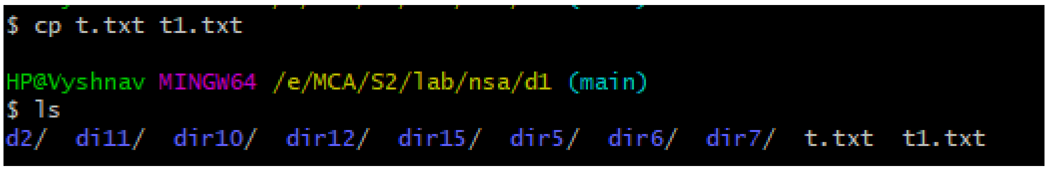
Syntax:

```
cp source_file destination
```

Option	Description
-a	Preserve the source's metadata, such as creation date, permissions, and extended attributes.
-b	Create backups for destination files. The backup file has the (~) suffix unless **--suffix** is used.
-f	If the destination file/directory already exists, replace it with the source.
-i	Ask before overwriting the destination with the source.
-n	Do not overwrite the destination file if it exists.
-R	Copy the source directory recursively.
-S	Provide a custom suffix for the backup file.
-t	Specify a directory for the source files.
-u	Replace files only if they satisfy the update condition.
-v	Output information about the copying process.

Example:

```
cp t.txt t1.txt
```



```
$ cp t.txt t1.txt

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa/d1 (main)
$ ls
d2/  di11/  dir10/  dir12/  dir15/  dir5/  dir6/  dir7/  t.txt  t1.txt
```

11) rm:

rm command is used to remove objects such as files, directories, symbolic links and so on from the file system.

Syntax:

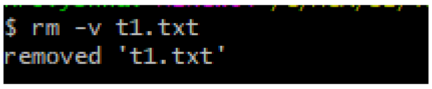
```
rm [options] [file or directory name]
```

Option	Description
-f	Forces the removal of all files or directories.
-i	Prompts for confirmation before removing.
-r	Removes directories and their content recursively.
-d	Removes empty directories.
-v	Provides a verbose output.

Examples:

- **Removing one file**

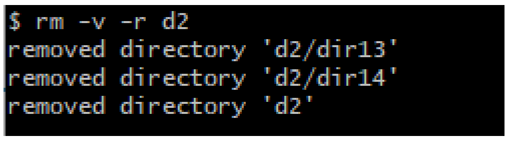
```
rm -v t1.txt
```



```
$ rm -v t1.txt
removed 't1.txt'
```

- **Removing entire directory**

```
rm -v -r d2
```



```
$ rm -v -r d2
removed directory 'd2/dir13'
removed directory 'd2/dir14'
removed directory 'd2'
```

12) touch:

touch command is a way to create empty. With touch command we can also update the modification and access time of each file.

Syntax:

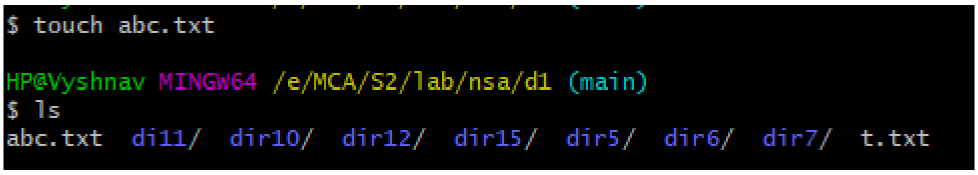
```
touch [options] file_name
```

Option	Description
-a	Changes the access time.
-c	Avoids creating a new file.
-d=<string>	Changes a timestamp using a date string.
-h	Changes a symbolic link's timestamp.
-m	Changes the modification time.
-r=<file>	Changes a timestamp to the referenced file's timestamp.
-t <stamp>	Modifies a timestamp, where the stamp is the date/time format.

Examples:

- **Create a file**

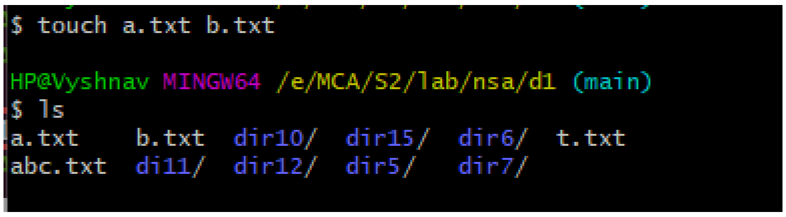
```
touch abc.txt
```



```
$ touch abc.txt

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa/d1 (main)
$ ls
abc.txt  di11/  dir10/  dir12/  dir15/  dir5/  dir6/  dir7/  t.txt
```

- **Create multiple file**

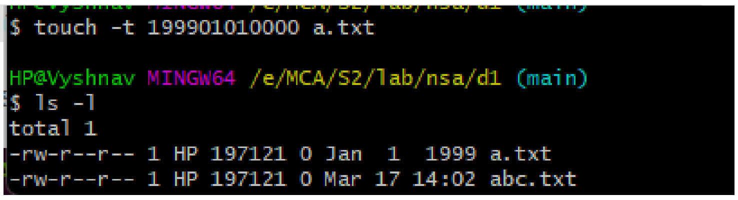
```
touch abc.txt
```



```
$ touch a.txt b.txt

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa/d1 (main)
$ ls
a.txt    b.txt  dir10/  dir15/  dir6/  t.txt
abc.txt  di11/  dir12/  dir5/   dir7/
```

- **Change timestamp of a file**

```
touch -t 199901010000 a.txt
```



```
$ touch -t 199901010000 a.txt

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa/d1 (main)
$ ls -l
total 1
-rw-r--r-- 1 HP 197121 0 Jan  1  1999 a.txt
-rw-r--r-- 1 HP 197121 0 Mar 17 14:02 abc.txt
```

13) chmod:

Linux **chmod** command is used to change the access permissions of files and directories. It stands for change mode. It cannot change the permission of symbolic links.

Syntax:

```
chmod [options] [permissions] [file name]
```

Option	Description
-c	Similar to the verbose option, but the difference is that it is reported if a change has been made.
-f	It is used to suppress the error messages.
-v	It is used to display a diagnostic for every processed file.
-reference=RFILE	It is used to specify the RFILE's mode alternatively MODE values.
-R	It is used to change files and directories recursively.

The permission statement is represented in indicators such as u+x, u-x. Where 'u' stands for 'user,' '+' stands for add, '-' stands for remove, 'x' stands for executable.

The user value can be:

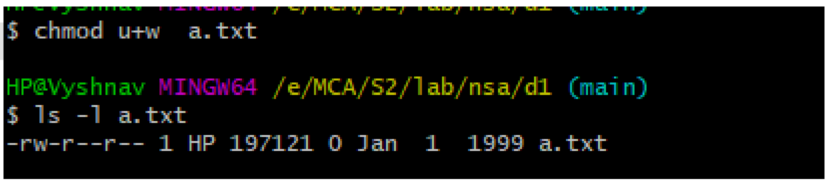
- u: the owner of the file
- g: group member
- o: others
- a: all

The permission types can be r, w, and x.

Examples:

- **Adding permission**

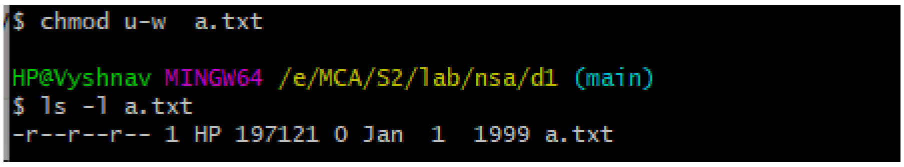
```
chmod u+w a.txt
```



```
$ chmod u+w  a.txt

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa/d1 (main)
$ ls -l a.txt
-rw-r--r-- 1 HP 197121 0 Jan  1  1999 a.txt
```

- **Removing permission**

```
chmod u-w a.txt
```



```
$ chmod u-w  a.txt

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa/d1 (main)
$ ls -l a.txt
-r--r--r-- 1 HP 197121 0 Jan  1  1999 a.txt
```

14) <u>chown:</u>

Linux chown command is used to change a file's ownership, directory, or symbolic link for a user or group. The chown stands for change owner. In Linux, each file is associated with a corresponding owner or group.

Users can be listed in different groups. The group allows us to set permission on the group level instead of setting permission on an individual level.

The file's ownership in the system may be only altered or edited by a super-user. The users can't give away a file's ownership even if the user owns it.

Syntax:

```
chown [options] new_owner[:new_group] file(s)
```

Option	Description
-c	Similar to the verbose option, but the difference is that it is reported if a change has been made.
-f	It is used to suppress the error messages.
-v	It is used to display a diagnostic for every processed file.
-reference=RFILE	It is used to specify the RFILE's mode alternatively MODE values.
-R	It is used to change files and directories recursively.

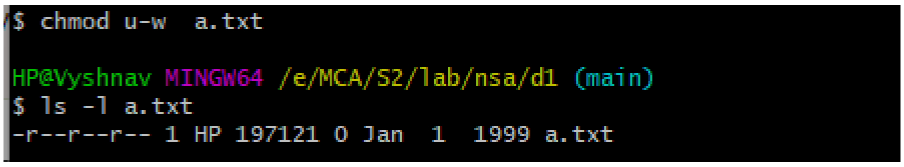
Example:

```
chown HP a.txt
```

The above command will set the **HP** as the owner of the file a.txt'.

- **Removing permission**

```
chmod u-w a.txt
```



```
$ chmod u-w  a.txt

HP@Vyshnav MINGW64 /e/MCA/S2/lab/nsa/d1 (main)
$ ls -l a.txt
-r--r--r-- 1 HP 197121 0 Jan  1  1999 a.txt
```

14) chown:

Linux chown command is used to change a file's ownership, directory, or symbolic link for a user or group. The chown stands for change owner. In Linux, each file is associated with a corresponding owner or group.

Users can be listed in different groups. The group allows us to set permission on the group level instead of setting permission on an individual level.

The file's ownership in the system may be only altered or edited by a super-user. The users can't give away a file's ownership even if the user owns it.

Syntax:

```
chown [options] new_owner[:new_group] file(s)
```

Option	Description
-c	Similar to the verbose option, but the difference is that it is reported if a change has been made.
-f	It is used to suppress the error messages.
-v	It is used to display a diagnostic for every processed file.
-reference=RFILE	It is used to specify the RFILE's mode alternatively MODE values.
-R	It is used to change files and directories recursively.

Example:

```
chown HP a.txt
```

The above command will set the **HP** as the owner of the file a.txt'.

15) more:

more command is used to view the contents of a file one page at a time. The more command is often used when dealing with large files that cannot be viewed in their entirety on a single screen.

Syntax:

```
more [options] [-num] [+/pattern] [+linenum] [file_name]
```

[-num]=> no:of lines to be displayed on the screen.

[+/pattern]=> any string pattern to find and display from file

[+/linenum]=> from which line to start displaying

Option	Description
-n	Displays the line number
-p	Display % of file that has been viewed
-s	Removes empty lines
-u	Underlines the output
-f	Doesn’t wrap long text

While viewing the contents:

- **Enter key**: to scroll down
- **Space bar**: to go to next page
- **b**: to got to previous page.

Example:

```
more t1.txt
```

16) less

The **less** command is a powerful tool for viewing and navigation through file content. It is useful when dealing with large files where we need to quickly move to different sections.

One of the advantage is that it doesn't load the entire file at once. This makes it much faster and more efficient when dealing with large file compared to other command like **cat.**

Syntax:

```
less [options] file_name
```

Option	Description
-g	Highlight current match
-i	Case-insensitive search
-m	Show detailed prompt
-N	Show line numbers
-S	Disable line wrap

Navigation keys:

- **ctrl+f:** forward one window
- **ctrl+d:** forward half window
- **ctrl+b:** backward one window
- **ctrl+u:** backward half window
- **j:** forward by one line
- **k:** backward by one line
- **G:** go to end of file
- **g:** go to start of file
- **10j:** 10 lines forward
- **10k:** 10 lines backward

/string is used to search for a specific string in the content of the file.

While viewing multiple file, to navigate between files:

- **n:** go to next file.
- **p:** go to previous file.

17) find:

The **find** command helps us to find a particular file within a directory. It is used to find the list of files for the various conditions like permission, user ownership, modification, date/time, size.

Syntax:

```
find [path] [options] [expression]
```

[path]=> starting directory for search

[option]=> additional setting or conditions for the search

[expression]=> criteria for filtering and locating files

Examples:

- **Find files by name**

  ```
  find . -name “*.txt”
  ```
  Displays all the files ending with the extension .txt

- **Directory name based on files by name**

  ```
  find . -type d -name “*.txt”
  ```
  List all directories having files ending with the extension .txt

- **Find newer files**

  ```
  find . -newer abc.txt
  ```
  List of all files created after abc.txt.

- **Find and delete a file**

  ```
  find . -name abc.txt -delete
  ```

- **Find file by modification time**

  ```
  find . -mtime n abc.txt
  ```
  No:of days can be positive or negative. -1 represents last day. +1 represents more than 1 day…

- **Find files by permission**

  ```
  find . -prem 777
  ```
  List of all files everyone could read, write and execute.

18) wc:

wc stands for word count. It is mainly used for counting purpose.

By default it displays four-columnar output

i. No:of lines present (starting from 0)
ii. No: of word
iii. No:of character
iv. File name

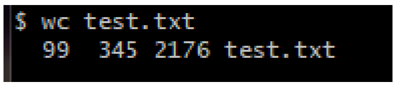
Syntax:

```
wc [options] file_name
```

Option	Description
-c	Print byte count
-m	Print character count
-l	Print no:of newline characters
-w	Print word count

Example:

```
wc test.txt
```



```
$ wc test.txt
  99  345 2176 test.txt
```

19) cut:

Linux **cut** command is useful for selecting a specific column of a file. It is used to cut a specific section by byte position, character and field and displays them.

To **cut** a specific section it is necessary to specify the delimiter.

Syntax:

```
cut [options] file_name
```

Option	Description
-b=LIST	Selects only the bytes specified in LIST
-c=LIST	Selects only characters specified in the LIST
-d=DELIM	Use DELIM as delimiter character
-f=LIST	Selects only the field after splitting with delimiter.

Example:

Consider a file **abc.txt** with the following content

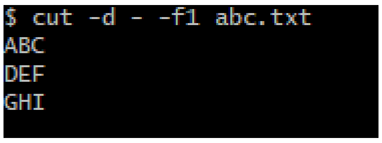
ABC-10

DEF-20

GHI-30

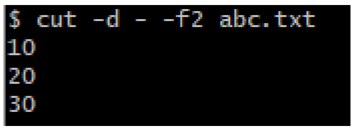
To cut using ‘**-**’as delimiter

```
cut -d - -f1 abc.txt
```



```
$ cut -d - -f1 abc.txt
ABC
DEF
GHI
```

```
cut -d - -f2 abc.txt
```



```
$ cut -d - -f2 abc.txt
10
20
30
```

20) paste:

Paste command is used to join files horizontally (parallel merging) by outputting lines consisting of lines from each file separated by tab as default delimiter.

Syntax:

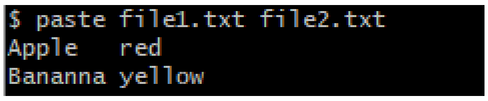
```
paste [options] [file 1] [file 2]
```

Option	Description
-d=LIST	Specify a list of delimiters to use when merging.
-s	Merge file in series rather than parallel
u	Removes any duplicate lines when merging.

Example:

Consider a file **file1.txt** with the following contents Apple Banana	Consider a file **file2.txt** with the following contents red yellow

```
paste file1.txt file2.txt
```



```
$ paste file1.txt file2.txt
Apple   red
Bananna yellow
```

21) head:

The **head** command prints top N number of data pf the given input. By default it prints 10 lines of specified file.

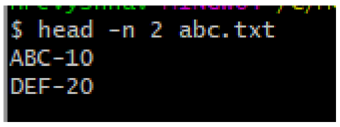
Syntax:

```
head [options] [file]
```

Option	Description
-n	Specify number of lines.
-c	Prints first ‘n’ bytes
-q	Used when there is multiple files and the file name is not displayed.

Example:

```
head -n 2 abc.txt
```



```
$ head -n 2 abc.txt
ABC-10
DEF-20
```

22) tail:

It is the complementary of head command. The **tail** command prints last N number of data. BY default last 10 lines are displayed

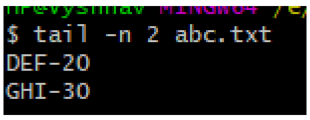
Syntax:

```
tail [options] [file]
```

Option	Description
-n	Specify number of lines.
-c	Prints first ‘n’ bytes
-q	Used when there is multiple files and the file name is not displayed.

Example:

```
tail -n 2 abc.txt
```



```
$ tail -n 2 abc.txt
DEF-20
GHI-30
```

23) useradd:

In Linux the **useradd** command is a low-level utility used for adding or creating user account.

When **useradd** command is executed the terminal performs few tasks

- It edits **/etc/passwd, /etc/shadow, etc/group, etc/gshadow** files for the newly created account.
- Creates and populates a home directory for the new user.
- Sets permission and ownership to the home directory.

Syntax:

```
useradd [options] username
```

Option	Description
-m	Creates a home directory for the new user in the default location
-d	Specify a custom home directory location for the user.
-g	Assign the user to a specific initial login group using either the group name or group ID.
-s	Set the login shell for the user. By default, /bin/bash is used.
-u	Assign a specific user ID (UID) to the new user.
-p	Set the password for the user account.

Examples:

- **Add user by specific home directory**

  ```
  useradd -d /home/dir1 user1
  ```

- **Add user with specific user ID**

  ```
  useradd -u 1234 user2
  ```

24) userdel:

This command is used to delete a user account and related files. The command modifies the system account files, deleting all the entries which refer to the username.

Syntax:

```
userdel [options] username
```

Option	Description
-f	Fore removal of user account, including home directory and mail spool even if the user is logged in.
-r	Remove users home directory along with the account.
-R	Applies the changes in the specified CHROOT-DIr
-z	Remove SELinux user mapping for the user’s login.

25) usermod:

The usermod command is one of the several Linux commands system administrators have at their disposal for user management. It is used to modify existing user account details, such as username, password, home directory location, default shell, and more.

Syntax:

```
usermod [options] username
```

Option	Description
-d,--home	Changes the user's home directory.
-s, --shell	Changes the user's login shell.
-a	Used with -G and adds the user to the supplementary group(s).

Examples:

- **Changing user’s home directory**

```
usermod -d /new/home/directory username
```

- **Adding user to additional groups**

```
usermod -a -G group1,group2 username
```

26) passwd:

Securing user accounts is a fundamental aspect of maintaining a robust and secure Linux system. One essential task is changing user passwords regularly to prevent unauthorized access. The passwd command in Linux provides a straightforward and effective way to modify user passwords.

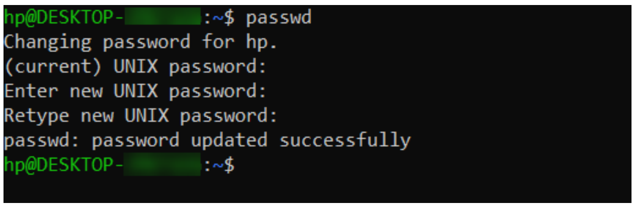
Syntax:

```
passwd [options] username
```

Option	Description
-d,--delete	Deletes the user password, making the account password-less.
-e, --expire	Immediately expires the account password, prompting the user to change it on the next login.
-i, --inactive	Sets the number of days after password expiration before the account is deactivated.
-k	Changes the password only if it is expired, keeping authentication tokens if not expired.
-r	Changes the password for a specified repository.

Examples:

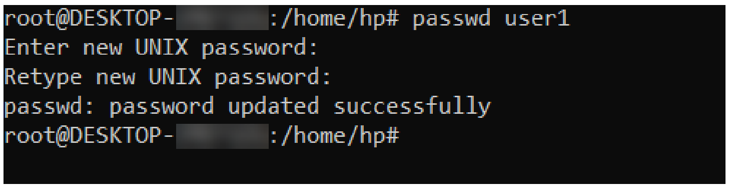
- **Changing user's password**



```
hp@DESKTOP-[redacted]:~$ passwd
Changing password for hp.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
hp@DESKTOP-[redacted]:~$
```

- **Change another user's password**

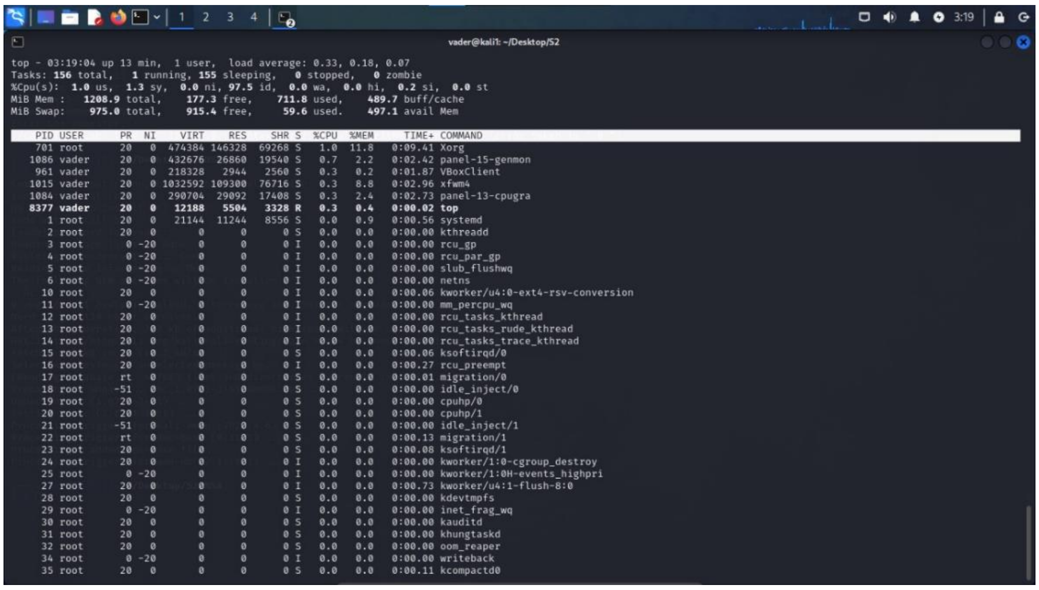
```
passwd user1
```



```
root@DESKTOP-[redacted]:/home/hp# passwd user1
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@DESKTOP-[redacted]:/home/hp#
```

27) top:

Top provides a dynamic real-time view of the running system. Usually, this command shows the summary information of the system and the list of processes or threads which are currently managed by the Linux Kernel.

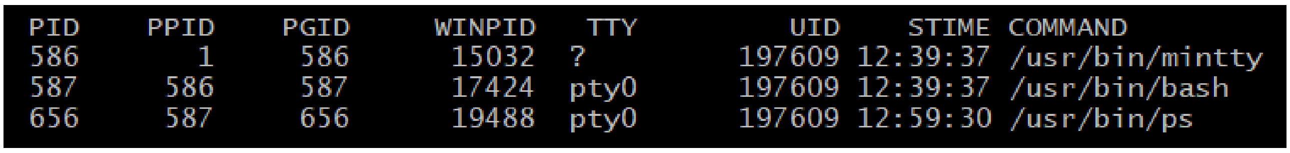


```
vader@kali1: ~/Desktop/S2

top - 03:19:04 up 13 min,  1 user,  load average: 0.33, 0.18, 0.07
Tasks: 156 total,   1 running, 155 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.0 us,  1.3 sy,  0.0 ni, 97.5 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
MiB Mem :   1208.9 total,    177.3 free,    711.8 used,    489.7 buff/cache
MiB Swap:    975.0 total,    915.4 free,     59.6 used.    497.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    701 root      20   0  474384 146328  69268 S   1.0  11.8   0:09.41 Xorg
   1086 vader     20   0  432676  26860  19540 S   0.7   2.2   0:02.42 panel-15-genmon
    961 vader     20   0  218328   2944   2560 S   0.3   0.2   0:01.87 VBoxClient
   1015 vader     20   0 1032592 109300  76716 S   0.3   8.8   0:02.96 xfwm4
   1084 vader     20   0  290704  29092  17408 S   0.3   2.4   0:02.73 panel-13-cpugra
   8377 vader     20   0   12188   5504   3328 R   0.3   0.4   0:00.02 top
      1 root      20   0   21144  11244   8556 S   0.0   0.9   0:00.56 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_flushwq
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
     10 root      20   0       0      0      0 I   0.0   0.0   0:00.06 kworker/u4:0-ext4-rsv-conversion
     11 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     12 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_kthread
     13 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_rude_kthread
     14 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_trace_kthread
     15 root      20   0       0      0      0 S   0.0   0.0   0:00.06 ksoftirqd/0
     16 root      20   0       0      0      0 I   0.0   0.0   0:00.27 rcu_preempt
     17 root      rt   0       0      0      0 S   0.0   0.0   0:00.01 migration/0
     18 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
     19 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     20 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
     21 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/1
     22 root      rt   0       0      0      0 S   0.0   0.0   0:00.13 migration/1
     23 root      20   0       0      0      0 S   0.0   0.0   0:00.08 ksoftirqd/1
     24 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0-cgroup_destroy
     25 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0H-events_highpri
     27 root      20   0       0      0      0 I   0.0   0.0   0:00.73 kworker/u4:1-flush-8:0
     28 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kdevtmpfs
     29 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 inet_frag_wq
     30 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kauditd
     31 root      20   0       0      0      0 S   0.0   0.0   0:00.00 khungtaskd
     32 root      20   0       0      0      0 S   0.0   0.0   0:00.00 oom_reaper
     34 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 writeback
     35 root      20   0       0      0      0 S   0.0   0.0   0:00.11 kcompactd0
```

- **PID:** Shows task's unique process id.
- **PR:** The process's priority. The lower the number, the higher the priority.
- **VIRT:** Total virtual memory used by the task.
- **USER:** User name of owner of task.
- **%CPU:** Represents the CPU usage.
- **TIME+:** CPU Time, the same as 'TIME', but reflecting more granularity through hundredths of a second.
- **SHR:** Represents the Shared Memory size (kb) used by a task.
- **NI:** Represents a Nice Value of task. A Negative nice value implies higher priority, and positive Nice value means lower priority.
- **%MEM:** Shows the Memory usage of task.
- **RES:** How much physical RAM the process is using, measured in kilobytes.
- **COMMAND:** The name of the command that started the process.

28) ps:

The ps command is used to view currently running processes on the system. It helps us to determine which process is doing what in our system, how much memory it is using, how much CPU space it occupies, user ID, command name, etc .



```
PID     PPID     PGID      WINPID   TTY         UID     STIME COMMAND
586        1      586       15032  ?          197609 12:39:37 /usr/bin/mintty
587      586      587       17424  pty0       197609 12:39:37 /usr/bin/bash
656      587      656       19488  pty0       197609 12:59:30 /usr/bin/ps
```

- **PID:** It shows the number of a Process ID.
- **PPID:** It shows the number of the parent processes of the process.
- **TTY or TT**: It shows the terminal corresponding to the process.
- **USER or UID**: It shows the username of the owner of the process.

29) SSH (Secure Shell)

SSH functions by creating a secure encrypted connection to a remote Linux machine, enabling users to log in to remote servers and execute commands directly. This secure access also allows for managing files and directories on the remote machine, while additionally offering the ability to set up secure tunnels for other applications to function securely over a potentially unsecure network.

Syntax:

```
ssh username@remote_server_ip
```

Example:

```
ssh user123@192.168.1.100
```

This logs you in as user "user123" on the server with IP address "192.168.1.100". You'll be prompted for the password for user123.

30) SCP (Secure Copy)

The Secure Copy Protocol (SCP) leverages SSH connections to securely transfer files and directories between your local machine and remote machines. This functionality allows you to seamlessly move files in both directions, be it uploading data to a remote server or downloading essential files from it.

Syntax:

```
scp source_file/directory destination_path
```

Example:

```
scp important_file.txt user123@192.168.1.100:/home/user123/documents
```

This command securely copies the file "important_file.txt" to the remote server with IP address "192.168.1.100", placing it in the user "user123" documents directory.

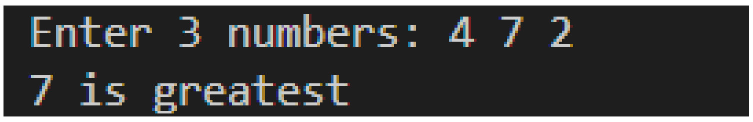
Experiment 4: **Largest Among 3 Numbers**

Aim: Write a Shell script to find the largest among 3 numbers.

Source Code:

```
#!/bin/bash
read -p"Enter 3 numbers: " a b c
if [ $a -gt $b ] && [ $a -gt $c ]
then
    echo $a "is greatest"
elif [ $b -gt $c ]
then
    echo $b "is greatest"
else
    echo $c "is greatest"
fi
```

Output:



```
Enter 3 numbers: 4 7 2
7 is greatest
```

Result: The script has executed successfully and required output is obtained.

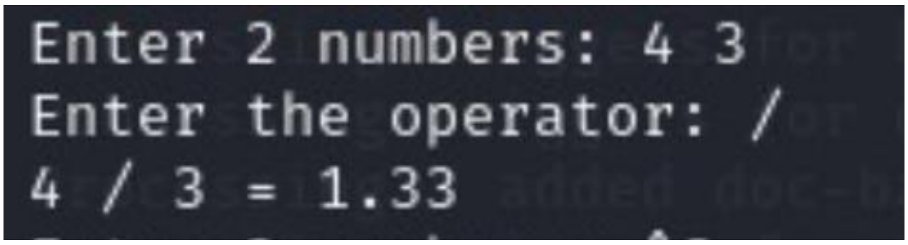
Experiment 5: **Simple Calculator**

Aim: Write a Shell script to perform operations of a simple calculator.

Source Code:

```
#!/bin/bash
while true
do
    read -p "Enter 2 numbers: " a b
    read -p "Enter the operator: " op
    case $op in
        "+")
            ans=$(echo "$a + $b" | bc);;
        "-")
            ans=$(echo "$a - $b" | bc);;
        "*")
            ans=$(echo "$a * $b" | bc);;
        "/")
            ans=$(echo "scale=2; $a / $b" | bc);;
        *)
            exit 1;;
    esac
    echo "$a $op $b = $ans"
done
```

Output:



```
Enter 2 numbers: 4 3
Enter the operator: /
4 / 3 = 1.33
```

Result: The script has executed successfully and required output is obtained.

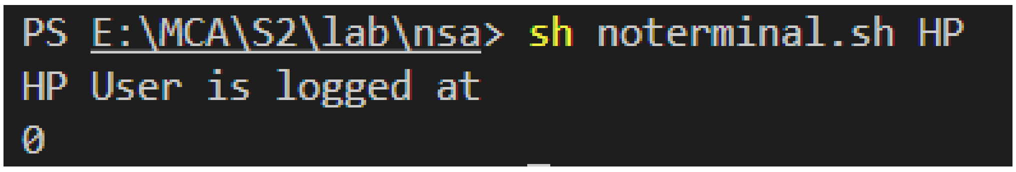
Experiment 6: **Number of Terminals Logged**

Aim: Write a Shell script to find the number of terminal user has logged in.

Source Code:

```
#!/bin/bash
if [ $# -eq 1 ]
then
    who > user.1st
    echo "$1 User is logged at"
    grep -c $1 user.1st
else
    echo "Pls enter User name"
fi
```

Output:



```
PS E:\MCA\S2\lab\nsa> sh noterminal.sh HP
HP User is logged at
0
```

Result: The script has executed successfully and required output is obtained.

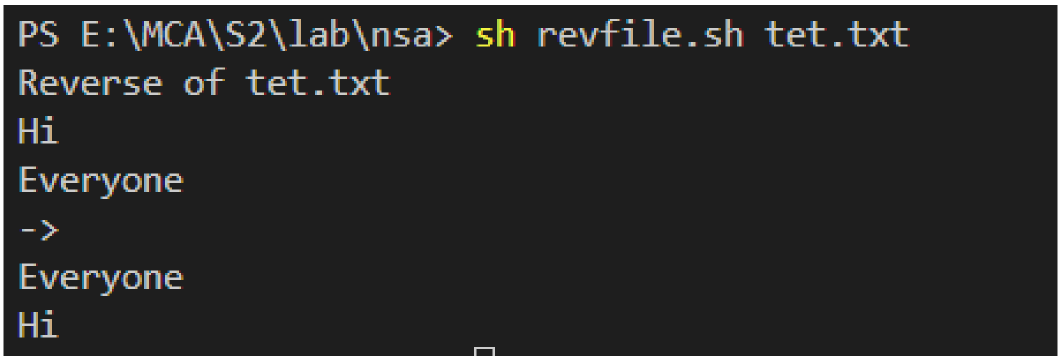
Experiment 7: **Reverse Content of File**

Aim: Write a Shell script to reverse the contents of a file.

Source Code:

```
#!/bin/bash
if [ $# -eq 1 ]
then
    if [ -f $1 ]
    then
        echo "Reverse of $1"
        cat $1
        echo "-> "
        tac $1
    else
        echo "File does not exist!!"
    fi
else
    echo "Enter file name or path"
fi
```

Output:



```
PS E:\MCA\S2\lab\nsa> sh revfile.sh tet.txt
Reverse of tet.txt
Hi
Everyone
->
Everyone
Hi
```

Result: The script has executed successfully and required output is obtained.

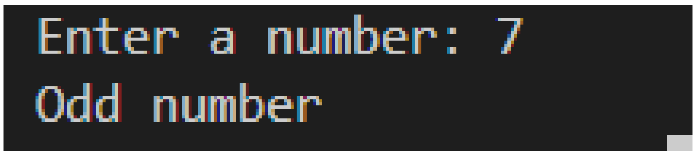
Experiment 8: **Even or Odd**

Aim: Write a Shell script to check whether a number is even or odd.

Source Code:

```
#!/bin/bash
read -p"Enter a number: " n
if [ $(( $n % 2 )) -eq 0 ]
then echo "Even number"
else
    echo "Odd number"
fi
```

Output:



```
Enter a number: 7
Odd number
```

Result: The script has executed successfully and required output is obtained.

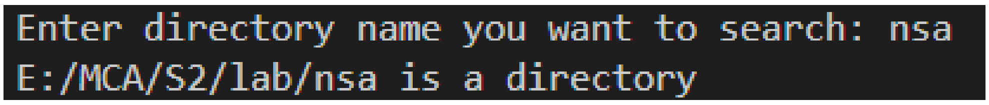
Experiment 9: **Check Directory**

Aim: Write a Shell script to check whether a file is a directory or not.

Source Code:

```
#!/bin/bash
read -p "Enter directory name you want to search: " dir
for filename in "E:/MCA/S2/lab/$dir"
do
    if [ -d "$filename" ]
    then
        echo "$filename is a directory"
    else
        echo "$filename is not directory"
    fi
done
```

Output:



```
Enter directory name you want to search: nsa
E:/MCA/S2/lab/nsa is a directory
```

Result: The script has executed successfully and required output is obtained.

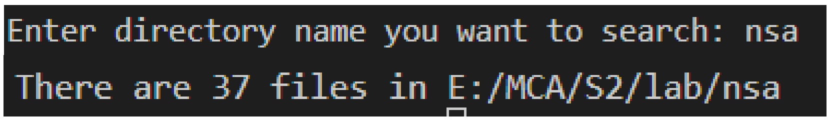
Experiment 10: **Count Number of Files**

Aim: Write a Shell script to count number of files in a directory.

Source Code:

```
#!/bin/bash
read -p "Enter directory name you want to search: " dir
direct="E:/MCA/S2/lab/$dir"
if [ -d $direct ]
  then
    num_files=$(find $direct -type f | ls -l $direct|
    wc -l)
    echo "There are $num_files files in $direct"
  else
    echo "$direct is not a directory"
  fi
```

Output:



```
Enter directory name you want to search: nsa
There are 37 files in E:/MCA/S2/lab/nsa
```

Result: The script has executed successfully and required output is obtained.

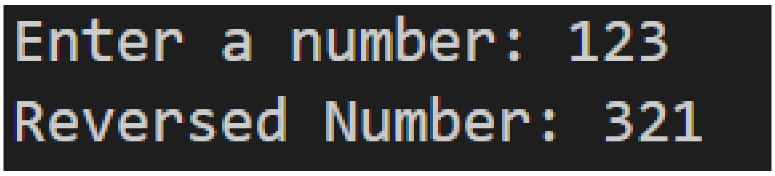
Experiment 11: **Reverse a Number**

Aim: Write a Shell script to reverse a number.

Source Code:

```
#!/bin/bash
read -p "Enter a number: " n
no=$n
p=0
while [ $n -gt 0 ]
do
    ld=$(($n % 10))
    p=$(($p * 10 + $ld))
    n=$(($n / 10))
done
echo "Reversed Number: $p"
```

Output:



```
Enter a number: 123
Reversed Number: 321
```

Result: The script has executed successfully and required output is obtained.

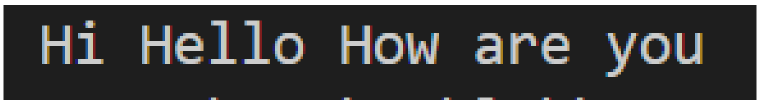
Experiment 12: **Combine Strings**

Aim: Write a Shell script to combine 2 or more string.

Source Code:

```
#!/bin/bash
s1="Hi"
s2="Hello"
s3="$s1 $s2"
s3+=" How are you"
echo $s3
```

Output:



```
Hi Hello How are you
```

Result: The script has executed successfully and required output is obtained.

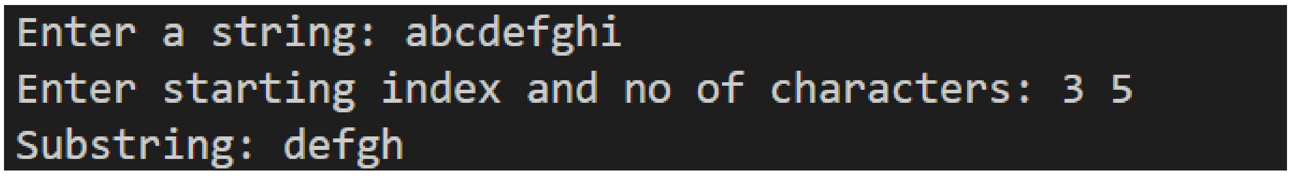
Experiment13: **Substring**

Aim: Write a Shell script to get the substring of a string.

Source Code:

```
#!/bin/bash
read -p "Enter a string: " str
read -p "Enter starting index and no of characters: " a b
substr=${str:$a:$b}
echo "Substring: $substr"
```

Output:



```
Enter a string: abcdefghi
Enter starting index and no of characters: 3 5
Substring: defgh
```

Result: The script has executed successfully and required output is obtained.

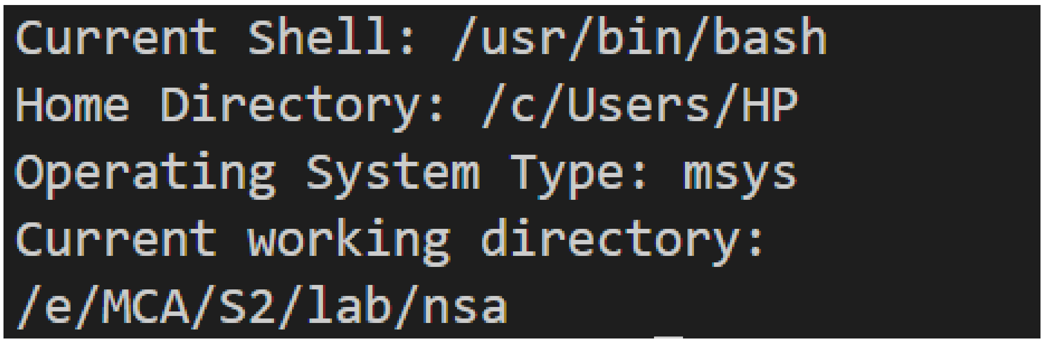
Experiment 14: **System Config**

Aim: Write a Shell script to show various system configuration.

Source Code:

```
#!/bin/bash
echo "Current Shell: $SHELL"
echo "Home Directory: $HOME"
echo "Operating System Type: $OSTYPE"
echo "Current working directory: "
pwd
```

Output:



```
Current Shell: /usr/bin/bash
Home Directory: /c/Users/HP
Operating System Type: msys
Current working directory:
/e/MCA/S2/lab/nsa
```

Result: The script has executed successfully and required output is obtained.

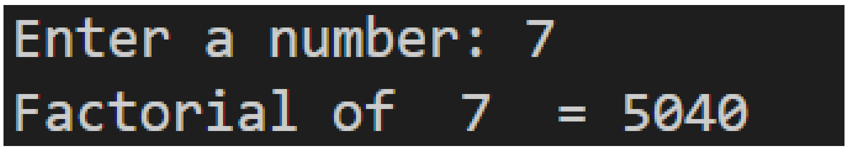
Experiment 15: **Factorial**

Aim: Write a Shell script to find factorial of a number.

Source Code:

```
read -p "Enter a number: " n
f=1
for i in $(seq 1 $n)
do
    f=$(($f * $i))
done
echo "Factorial of " $n " = "$f
```

Output:



```
Enter a number: 7
Factorial of  7  = 5040
```

Result: The script has executed successfully and required output is obtained.

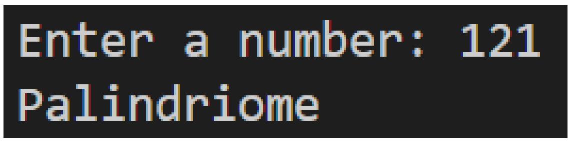
Experiment 16: **Palindrome**

Aim: Write a Shell script to check if a number is Palindrome or not.

Source Code:

```
read -p "Enter a number: " n
no=$n
p=0
while [ $n -gt 0 ]
do
    ld=$(($n % 10))
    p=$(($p * 10 + $ld))
    n=$(($n / 10))
done
if [ $p -eq $no ]
then echo "Palindriome"
else echo "Not palindrome"
fi
```

Output:



```
Enter a number: 121
Palindriome
```

Result: The script has executed successfully and required output is obtained.

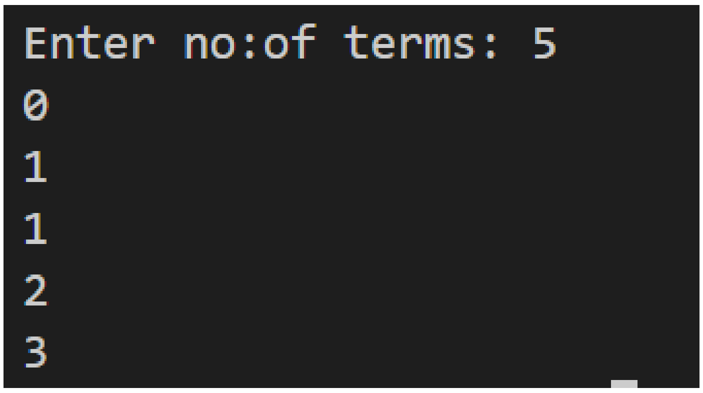
Experiment 17: **Fibonacci Series**

Aim: Write a Shell script to generate Fibonacci series.

Source Code:

```
read -p "Enter no:of terms: " n
a=0
b=1
for i  in $(seq 1 $n)
do
    c=$((a + b))
    echo $a
    a=$b
    b=$c
done
```

Output:



```
Enter no:of terms: 5
0
1
1
2
3
```

Result: The script has executed successfully and required output is obtained.

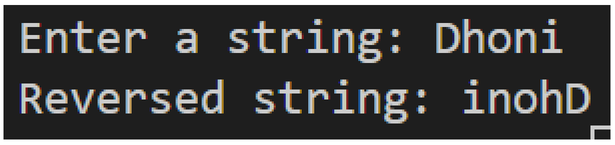
Experiment 18: **Reverse a String**

Aim: Write a Shell script to reverse a string.

Source Code:

```
read -p "Enter a string: " s
l=${#s}
ns=""
for i in $(seq $l -1 1)
do
    a=$(expr substr "$s" $i 1)
    ns="$ns$a"
done
echo "Reversed string: "$ns
```

Output:



```
Enter a string: Dhoni
Reversed string: inohD
```

Result: The script has executed successfully and required output is obtained.

Experiment 19: **File Handling-1**

Aim: Write a shell script to create a file. Follow the given instructions:

(i) Input a page profile to yourself, copy it into other existing file.

(ii) Start printing file at certain line.

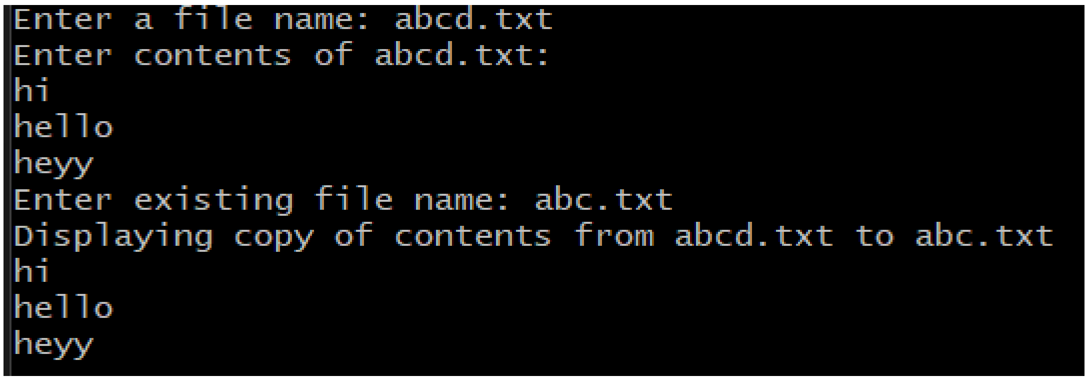
(iii) Print all the difference between two file, copy the two files.

(iv) Print lines matching certain word pattern.

Source Code:

(i) Input a page profile to yourself, copy it into other existing file.

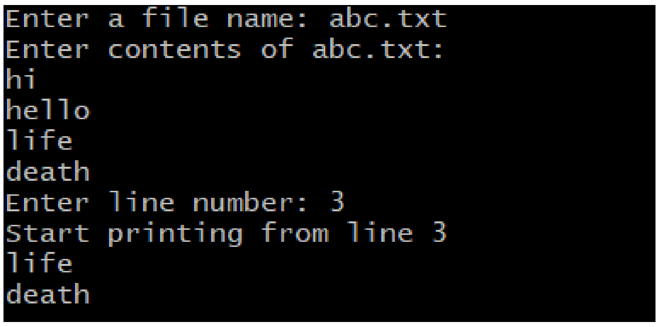
```
#!/bin/bash
read -p "Enter a file name: " file1
echo "Enter contents of $file1: "
cat > $file1
read -p "Enter existing file name: " file2
echo "Displaying copy of contents from $file1 to
$file2"
cp $file1 $file2
cat $file2
```



```
Enter a file name: abcd.txt
Enter contents of abcd.txt:
hi
hello
heyy
Enter existing file name: abc.txt
Displaying copy of contents from abcd.txt to abc.txt
hi
hello
heyy
```

(ii) Start printing file at certain line.

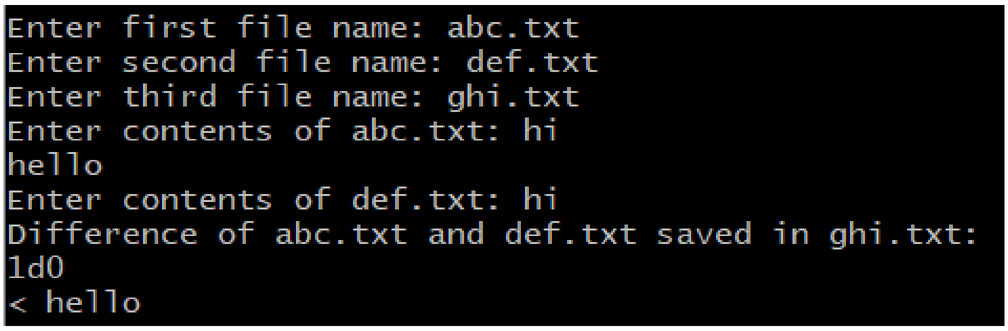
```
#!/bin/bash
read -p "Enter a file name: " file1
echo "Enter contents of $file1: "
cat > $file1
read -p "Enter line number: " l
echo "Start printing from line $l"
tail +$l $file1
```



```
Enter a file name: abc.txt
Enter contents of abc.txt:
hi
hello
life
death
Enter line number: 3
Start printing from line 3
life
death
```

(iii) Print all the difference between two file, copy the two files.

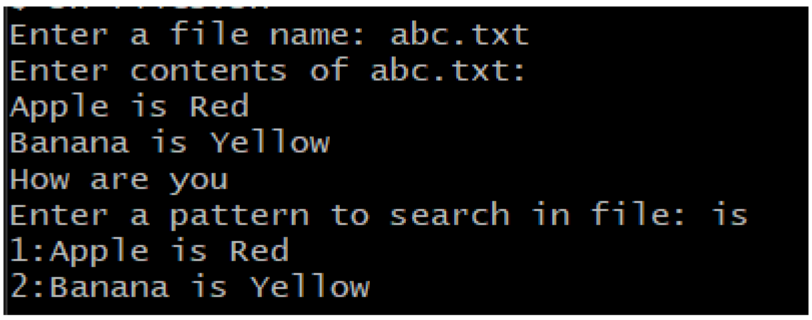
```
#!/bin/bash
read -p "Enter first file name: " file1
read -p "Enter second file name: " file2
read -p "Enter third file name: " file3
echo "Enter contents of $file1: "
cat > $file1
echo "Enter contents of $file2: "
cat > $file2
echo "Difference of $file1 and $file2 saved in
$file3: "
diff -a $file1 $file2 > $file3
cat $file3
```



```
Enter first file name: abc.txt
Enter second file name: def.txt
Enter third file name: ghi.txt
Enter contents of abc.txt: hi
hello
Enter contents of def.txt: hi
Difference of abc.txt and def.txt saved in ghi.txt:
1d0
< hello
```

(iv) Print lines matching certain word pattern.

```
#!/bin/bash
read -p "Enter a file name: " file1
echo "Enter contents of $file1: "
cat > $file1
read -p "Enter a pattern to search in file: " s
grep -ni $s $file1
```



```
Enter a file name: abc.txt
Enter contents of abc.txt:
Apple is Red
Banana is Yellow
How are you
Enter a pattern to search in file: is
1:Apple is Red
2:Banana is Yellow
```

Result: The script has executed successfully and required output is obtained.

Experiment 20: **System Information**

Aim: Write shell script for·

i) Showing the count of users logged in.

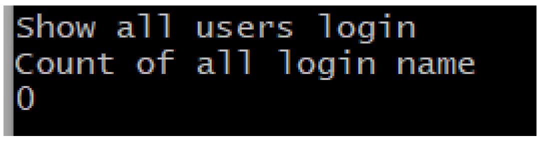
ii) Printing Column list of files in home directory.

iii) Listing job with below normal priority.

Source Code:

(i) Showing the count of users logged in.

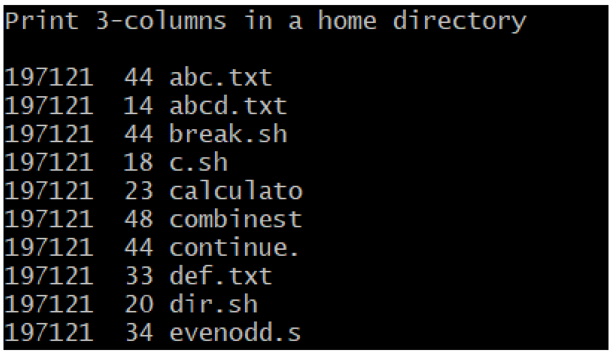
```
#!/bin/bash
echo "Show all users login"
who
echo "Count of all login name"
who | wc -l
```



```
Show all users login
Count of all login name
0
```

(ii) Printing Column list of files in your home directory.

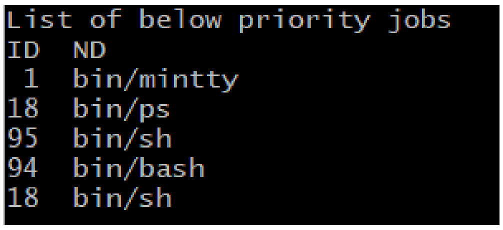
```
#!/bin/bash
echo "Print 3-columns in a home directory"
ls -l | cut -c 17-24,39-50,56-
```



```
Print 3-columns in a home directory

197121  44 abc.txt
197121  14 abcd.txt
197121  44 break.sh
197121  18 c.sh
197121  23 calculato
197121  48 combinest
197121  44 continue.
197121  33 def.txt
197121  20 dir.sh
197121  34 evenodd.s
```

(iii) Listing your job with below normal priority.

```
#!/bin/bash
echo "List of below priority jobs"
ps -al | cut -c 16-19,70-
```



```
List of below priority jobs
ID  ND
 1  bin/mintty
18  bin/ps
95  bin/sh
94  bin/bash
18  bin/sh
```

<u>Result:</u> The script has executed successfully and required output is obtained.

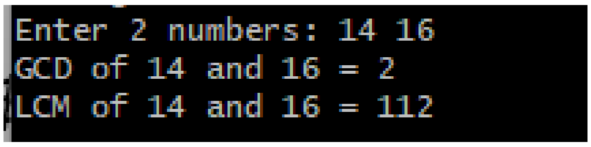
Experiment 21: **GCD and LCM**

Aim: Write a Shell script to compute GCD and LCM of 2 numbers.

Source Code:

```
#!/bin/bash
read -p "Enter 2 numbers: " a b
m=$a
if [ $b -lt $m ]
then
    m=$b
fi
while [ $m -gt 0 ]
do
    x=$(expr $a % $m)
    y=$(expr $b % $m)
    if [ $x -eq 0 -a $y -eq 0 ]
    then
        echo "GCD of $a and $b = $m"
    break
    fi
    m=$(expr $m - 1)
done

l=$(expr $a \* $b / $m)
echo "LCM of $a and $b = $l"
```

Output:



```
Enter 2 numbers: 14 16
GCD of 14 and 16 = 2
LCM of 14 and 16 = 112
```

Result: The script has executed successfully and required output is obtained.

Experiment 22: **Student Database**

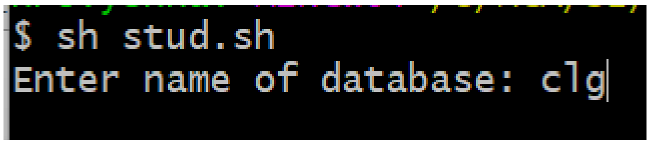
Aim: Shell script to perform database operations for student data like view, add and delete records.

Source Code:

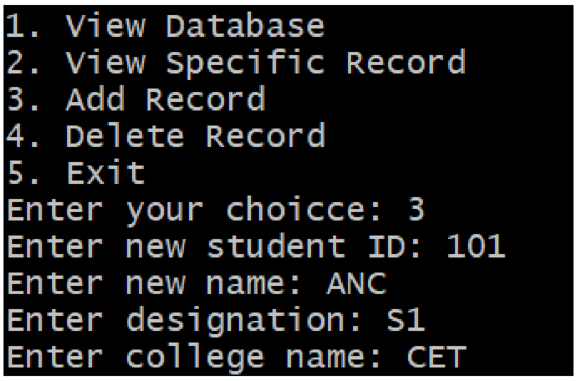
```
#!/bin/bash
i="y"
read -p "Enter name of database: " db
while [ $i = "y" ]
do
    clear
    echo "1. View Database"
    echo "2. View Specific Record"
    echo "3. Add Record"
    echo "4. Delete Record"
    echo "5. Exit"
    read -p "Enter your choicce: " ch
    case $ch in
    1) cat $db;;
    2) read -p "Enter ID: " id
       grep -i "$id" $db;;
    3) read -p "Enter new student ID: " tid
       read -p "Enter new name: " nm
       read -p "Enter designation: " des
       read -p "Enter college name: " college
       echo "$tid $nm $des $college">>$db;;
    4) read -p "Enter ID: " id
       grep -v "$id" $db > dbs1
       echo "Record is deleted"
       cat dbs1;;
    5) exit ;;
    *) echo "Invalid choice";;
    esac
    read -p "Do you want to continue? " i
    if [ $i != "y" ]
    then
```

```
            exit
        fi
done
```

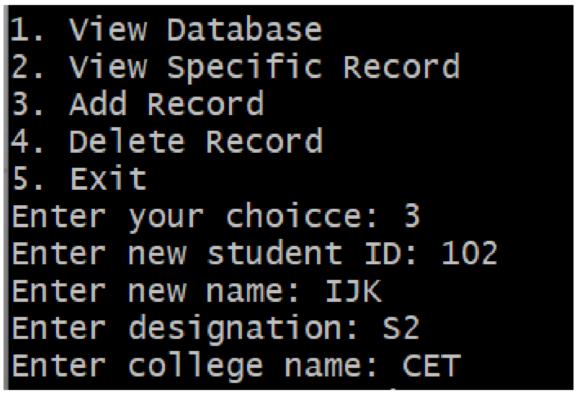
Output:



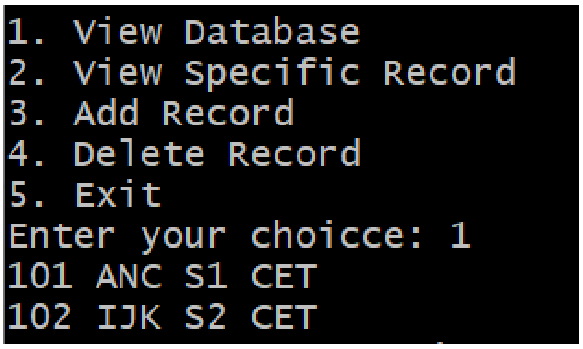
```
$ sh stud.sh
Enter name of database: clg
```



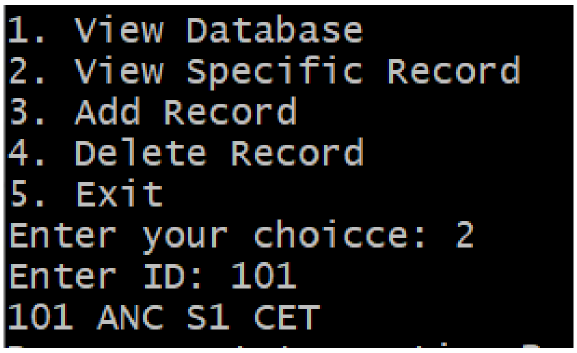
```
1. View Database
2. View Specific Record
3. Add Record
4. Delete Record
5. Exit
Enter your choicce: 3
Enter new student ID: 101
Enter new name: ANC
Enter designation: S1
Enter college name: CET
```



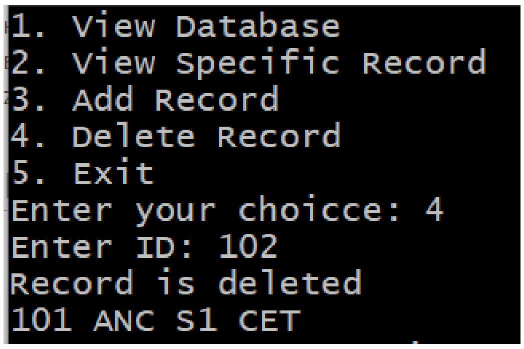
```
1. View Database
2. View Specific Record
3. Add Record
4. Delete Record
5. Exit
Enter your choicce: 3
Enter new student ID: 102
Enter new name: IJK
Enter designation: S2
Enter college name: CET
```



```
1. View Database
2. View Specific Record
3. Add Record
4. Delete Record
5. Exit
Enter your choicce: 1
101 ANC S1 CET
102 IJK S2 CET
```



```
1. View Database
2. View Specific Record
3. Add Record
4. Delete Record
5. Exit
Enter your choicce: 2
Enter ID: 101
101 ANC S1 CET
```



```
1. View Database
2. View Specific Record
3. Add Record
4. Delete Record
5. Exit
Enter your choicce: 4
Enter ID: 102
Record is deleted
101 ANC S1 CET
```

Result: The script has executed successfully and required output is obtained.

Experiment 23: **File Handling-2**

Aim:

1. Use the **cat** command to create a file containing the following table. Call it **myfile.**

1001	RAM	97
1002	NEHA	89
1010	DIVYA	77
1025	RAHUL	65
1012	ARUN	99
1013	NISHA	76

2. Print the first 2 rows of the file myfile.
3. Print the last 2 rows of the file myfile.
4. Copy the contents of "myfile" into another file called **mycopyfile**.
5. Count the number of lines, words, and characters of myfile.
6. Print the contents of the 3rd row only of the file myfile.
7. Append the contents of myfile into a file called **myappendfile** without rewriting the contents of myappendfile.

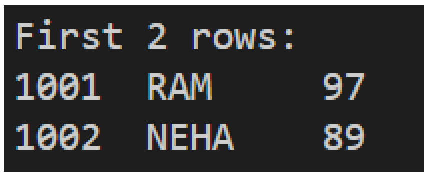
Source Code:

1. Using the **cat** command to create **myfile.**

```
#!/bin/bash
echo "1001  RAM    97" >> myfile
echo "1002  NEHA   89" >> myfile
echo "1010  DIVYA  77" >> myfile
echo "1025  RAHUL  65" >> myfile
echo "1012  ARUN   99" >> myfile
echo "1013  NISHA  76" >> myfile
```

2. Print the first 2 rows of the file myfile.

```
echo "First 2 rows:"
head -2 myfile
```



```
First 2 rows:
1001  RAM    97
1002  NEHA   89
```

3. Print the last 2 rows of the file myfile.

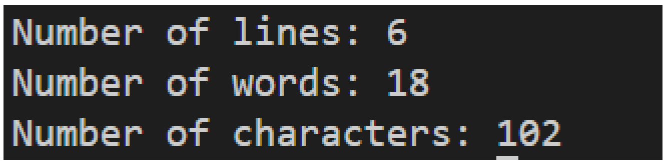
```
echo "First 2 rows:"
tail -2 myfile
```

4. Copy the contents of “myfile” into another file called **mycopyfile**

```
cp myfile mycopyfile
```

5. Count the number of lines, words, and characters of myfile.

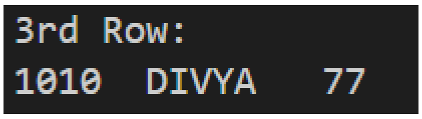
```
#!/bin/bash
echo "Number of lines: $(wc -l < myfile)"
echo "Number of words: $(wc -w < myfile)"
echo "Number of characters: $(wc -c < myfile)"
```



```
Number of lines: 6
Number of words: 18
Number of characters: 102
```

6. Print the contents of the 3rd row only of the file myfile.

```
#!/bin/bash
echo "3rd Row:"
sed -n '3p' myfile
```



```
3rd Row:
1010  DIVYA   77
```

7. Append the contents of myfile into a file called **myappendfile** without rewriting the contents of myappendfile.

```
cat myfile >> myappendfile
```

Result: The script has executed successfully and required output is obtained.

Experiment 24: Installation of LAMP

A **LAMP** stack is a group of open-source software that is typically installed together in order to enable a server to host dynamic websites and web apps written in **PHP**. This term is an acronym which represents the **Linux** operating system with the **Apache** web server. The site data is stored in a **MySQL** database, and dynamic content is processed by **PHP**.

Step 1: Installing Apache and Updating the Firewall

The Apache web server is among the most popular web servers in the world. It's well documented, has an active community of users, and has been in wide use for much of the history of the web, which makes it a great choice for hosting a website.

Start by updating the package manager cache:

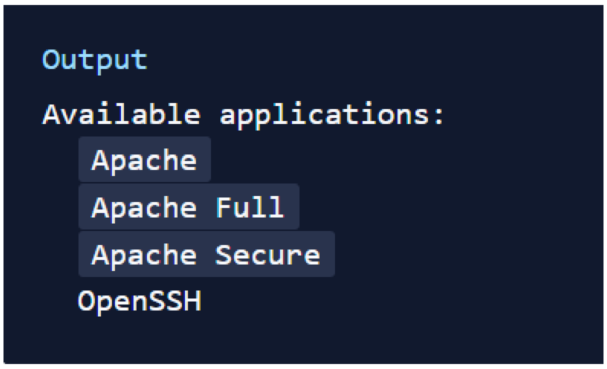
```
➢ sudo apt update
```

Then, install Apache with:

```
➢ sudo apt install apache2
```

Once the installation is finished, firewall settings to allow HTTP traffic needs to be adjusted. Ubuntu's default firewall configuration tool is called **Uncomplicated Firewall (UFW)**. It has different application profiles that can be leveraged. To list all currently available UFW application profiles, execute this command:

```
➢ sudo ufw app list
```



```
Output
Available applications:
  Apache
  Apache Full
  Apache Secure
  OpenSSH
```

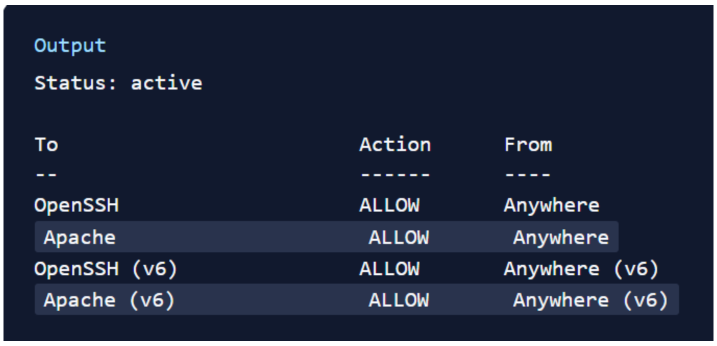
- **Apache**: This profile opens only port 80 (normal, unencrypted web traffic).
- **Apache Full**: This profile opens both port 80 (normal, unencrypted web traffic) and port 443 (TLS/SSL encrypted traffic).
- **Apache Secure**: This profile opens only port 443 (TLS/SSL encrypted traffic).

To only allow traffic on port 80, use the Apache profile:

```
➢ sudo ufw allow in "Apache"
```

Verify the change with:

- **sudo** ufw status



```
Output
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
 Apache                     ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
 Apache (v6)                ALLOW       Anywhere (v6)
```

You can do a spot check right away to verify that everything went as planned by visiting your server's public IP address in your web browser

- **http://your_server_ip**

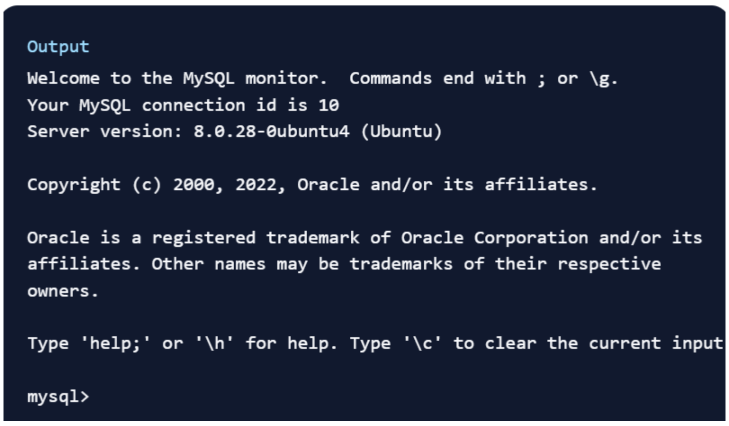
Step 2: Installing MySQL

With a web server operational, the next step is to install a database system to store and manage website data.

- **sudo** apt install mysql-server

Test by logging in to MySQl console:

- **sudo** mysql



```
Output
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.28-0ubuntu4 (Ubuntu)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input

mysql>
```

This will connect to the MySQL server as the administrative database user root, which is inferred by the use of sudo when running this command.

Step 3: Installing PHP

PHP is the component that processes code to display dynamic content to the final user. In addition to the **php** package, the **php-mysql** module is required. This module enables PHP to communicate with MySQL-based databases. **Libapache2-mod-php** is also necessary to allow Apache to handle PHP files. Core PHP packages are automatically installed as dependencies.

To install these packages, run the following command:

```
➢ sudo apt install php libapache2-mod-php php-mysql
```

Step 4: Creating a Virtual Host for the Website

Apache web server allows creating virtual hosts (similar to Nginx server blocks) to manage configuration details and host multiple domains on a single server.

Apache on Ubuntu has one virtual host enabled by default that is configured to serve documents from the ***/var/www/html directory*.** Apache typically uses ***/var/www/html*** as the default directory for website content. This is suitable for a single site. However, managing multiple sites becomes difficult with this approach. To address this, we can adopt a better structure: keep /var/www/html as the default directory to serve content that doesn't match any specific virtual host configuration. For additional sites, create dedicated directories within **/var/www** for each site. For instance, to host a site named **your_domain**, create a directory named **your_domain** inside **/var/www**. This way, we can manage multiple sites with a cleaner organization.

Create the directory for **your_domain** as follows:

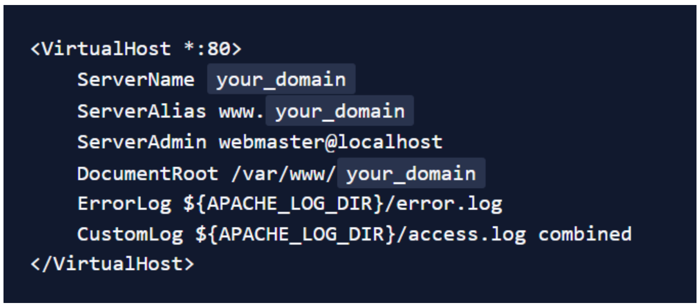
```
➢ sudo mkdir /var/www/your_domain
```

Next, assign ownership of the directory with the $USER environment variable, which will reference the current system user:

```
➢ sudo chown -R $USER:$USER /var/www/your_domain
```

Then, open a new configuration file in Apache’s sites-available directory

```
➢ sudo nano /etc/apache2/sites-available/your_domain.conf
```



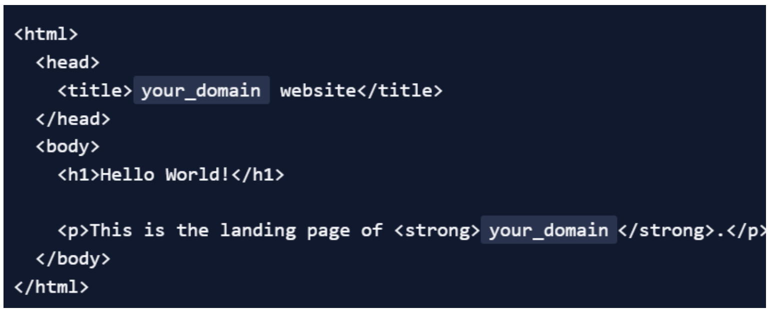
```
<VirtualHost *:80>
    ServerName your_domain
    ServerAlias www. your_domain
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/ your_domain
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

This will create a new blank file. Add in the following bare-bones configuration:

With this VirtualHost configuration, we're telling Apache to serve your_domain using /var/www/your_domain as the web root directory.

The new website is now active, but the web root **/var/www/your_domain** is still empty. Create an **index.html** file in that location to test that the virtual host works as expected:

➢ `nano /var/www/your_domain/index.html`

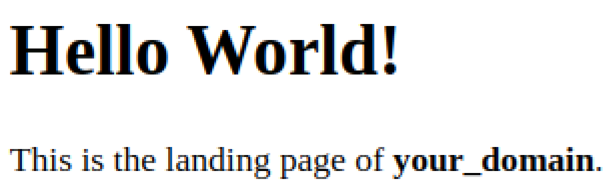


```
<html>
  <head>
    <title> your_domain  website</title>
  </head>
  <body>
    <h1>Hello World!</h1>

    <p>This is the landing page of <strong> your_domain </strong>.</p>
  </body>
</html>
```

Save and close the file, in the browser access the server's domain name or IP address:

➢ `http://server_domain_or_IP`

The web page should reflect the contents in the file just edited:



Hello World!

This is the landing page of **your_domain**.

Experiment 25: Installation of Laravel

Laravel is a web application framework known for its expressive and elegant syntax. It provides a foundation and structure for building web applications, allowing developers to focus on core functionalities while the framework handles common tasks.

Laravel prioritizes a positive developer experience alongside powerful features like robust dependency injection, a clear database abstraction layer, queue and scheduled job management, unit and integration testing capabilities, and more.

Furthermore, Laravel boasts a vibrant ecosystem supported by a vast community and extensive documentation, making it accessible to developers of all levels of expertise. With its built-in authentication and authorization functionalities, Laravel enhances the security of web applications, ensuring that sensitive data remains protected. In essence, Laravel continues to empower developers worldwide, enabling them to create robust and scalable web applications with unparalleled efficiency and ease.

Why Laravel?

- **Progressive Framework:** Laravel's vast library of documentation, guides, and video tutorials will help to learn the ropes without becoming overwhelmed.

- **Scalable Framework:** Laravel is incredibly scalable. Thanks to the scaling-friendly nature of PHP and Laravel's built-in support for fast, distributed cache systems like Redis, horizontal scaling with Laravel is a breeze. In fact, Laravel applications have been easily scaled to handle hundreds of millions of requests per month.

- **Community Framework:** Laravel combines the best packages in the PHP ecosystem to offer the most robust and developer friendly framework available. In addition, thousands of talented developers from around the world have contributed to the framework.

Steps to install Laravel on Ubuntu 20.04

Step 1: Install Apache web server

To host the Laravel program, the first step is to install a web server which can either be **Nginx** or **Apache** web servers.

To install the **Apache web server:**

```
➢ sudo apt install apache2
```

After installing Apache, verify if it's running. If not, use the following command to start it:

```
➢ sudo systemctl start apache2
```

Step 2: Installing PHP and its additional plugins

Laravel requires PHP along with some specific extensions. Install PHP and the required extensions using the following commands:

```
➢ sudo apt install php php-cli php-common php-mbstring php-xml
  php-zip php-mysql php-pgsql php-sqlite3 php-json php-bcmath
  php-gd php-tokenizer php-xmlwriter
```

Step 3: Download and Install a Database Manager

Laravel applications require a database to store data. MariaDB is one of several options supported by Laravel, including MySQL, SQLite, Postgres, and SQL Server.

```
➢ sudo apt install mariadb-server
```

To access the MariaDB prompt after installing the database server:

```
sudo mysql -u root -p
```

Sign in, then make a database and a user for it; grant the database user the necessary permissions.

```
➢ CREATE DATABASE laravel_db;
➢ CREATE USER 'laravel_user'@'localhost' IDENTIFIED BY
  'secretpassword';
➢ GRANT ALL ON laravel_db.* TO 'laravel_user'@'localhost';
➢ FLUSH PRIVILEGES;
➢ QUIT;
```

Step 4: Install Composer

Composer is a package manager and prerequisite management tool for PHP and manages the libraries and dependencies required by PHP based on the particular framework. It is required to download the Composer installer (composer. phar file) by executing the curl command and move the file to the *usr/local/bin/* directory:

- ➢ `curl -sS https://getcomposer.org/installer | php`

To move the file to the usr/local/bin/ path, run the following command:

- ➢ `sudo mv composer.phar /usr/local/bin/composer`

Assign authority to execute:

- ➢ `sudo chmod +x /usr/local/bin/composer`

Step 5: Install Laravel 8 on Ubuntu

Now that Composer is installed, we can use it to install Laravel. Open a terminal and navigate to the document root of the web server. For Apache, the document root is typically located at */var/www/html*.

To install Laravel :

- ➢ `sudo composer create-project laravel/laravel laravelapp`

Once the installation is complete, navigate to the project directory:

- ➢ `cd your-project-name`

Step 6: Configure Apache to serve Laravel

To host a Laravel site, it is necessary to configure the Apache web server. To launch the virtual host file:

- ➢ `sudo vim /etc/apache2/sites-available/laravelapp.conf`

Add the following content to the configuration file:

```
<VirtualHost *:80>
    ServerName your-domain-or-ip
    DocumentRoot /var/www/html/laravelapp/public
    <Directory /var/www/html/ laravelapp>
        AllowOverride All
    </Directory>
</VirtualHost>
```

Enable the Apache rewrite module:

- ➢ `sudo a2enmod rewrite`

Enable the virtual host:

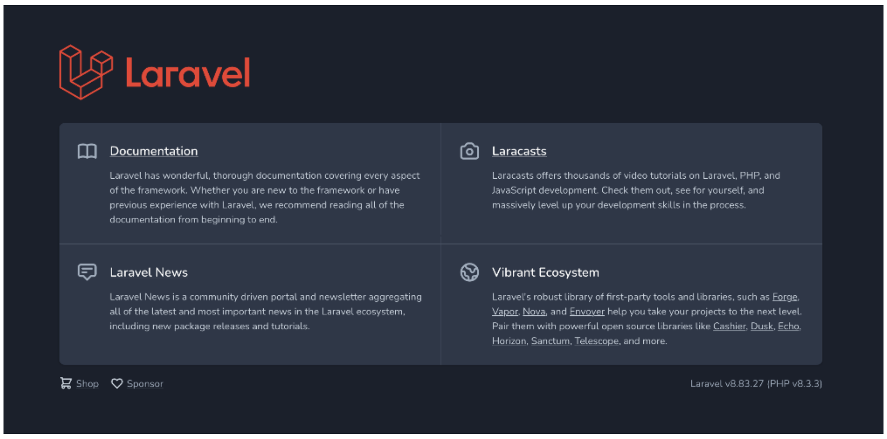
- `sudo a2ensite laravelapp.conf`

Restart Apache for the changes to take effect:

- `sudo systemctl restart apache2`

Step 7: Run Laravel in a web browser

Finally, to access Laravel, refer to the Fully Qualified Domain Name (FQDN) or IP address of the server where Laravel is installed. If URL is not specified, Laravel will use the default URL.



Documentation

Laravel has wonderful, thorough documentation covering every aspect of the framework. Whether you are new to the framework or have previous experience with Laravel, we recommend reading all of the documentation from beginning to end.

Laracasts

Laracasts offers thousands of video tutorials on Laravel, PHP, and JavaScript development. Check them out, see for yourself, and massively level up your development skills in the process.

Laravel News

Laravel News is a community driven portal and newsletter aggregating all of the latest and most important news in the Laravel ecosystem, including new package releases and tutorials.

Vibrant Ecosystem

Laravel's robust library of first-party tools and libraries, such as Forge, Vapor, Nova, and Envoyer help you take your projects to the next level. Pair them with powerful open source libraries like Cashier, Dusk, Echo, Horizon, Sanctum, Telescope, and more.

Shop Sponsor Laravel v8.83.27 (PHP v8.3.3)

Experiment 26: **Installation of Wireshark**

Wireshark is software that is widely used in the analysis of data packets in a network. Wireshark is completely free and open source. This packet analyzer is used for a variety of purposes like troubleshooting networks, understanding communication between two systems, developing new protocols, etc.

Step 1: Update Package Repository

To ensure that the package repository is up to date.

```
➢ sudo apt update
```

Step 2: Install Wireshark

Install Wireshark using the package manager specific to the Linux distribution. For Ubuntu and Debian based system we can use **apt.**

```
➢ sudo apt install wireshark
```

Step 3: Configure Wireshark

If **non-superusers** need to capture packets then these users must be added to the **wireshark** group.

```
➢ sudo usermod -aG wireshark $USER
```

Step 4: Start Wireshark by running `sudo wireshark` in the terminal.

Step 5: Select the network interface you want to capture packets from in the Wireshark GUI.

Step 6: Click on the green shark fin icon to start capturing packets.

Step 7: Analyze the captured packets using various filters, dissectors, and other features provided by Wireshark.

Step 8: Stop the capture by clicking the red square "Stop" button in the toolbar.

Step 9: To save the captured packets for later analysis, select "File" -> "Save" from the menu.