LAB CYCLE:1

EXPERIMENT NO:1

Date:

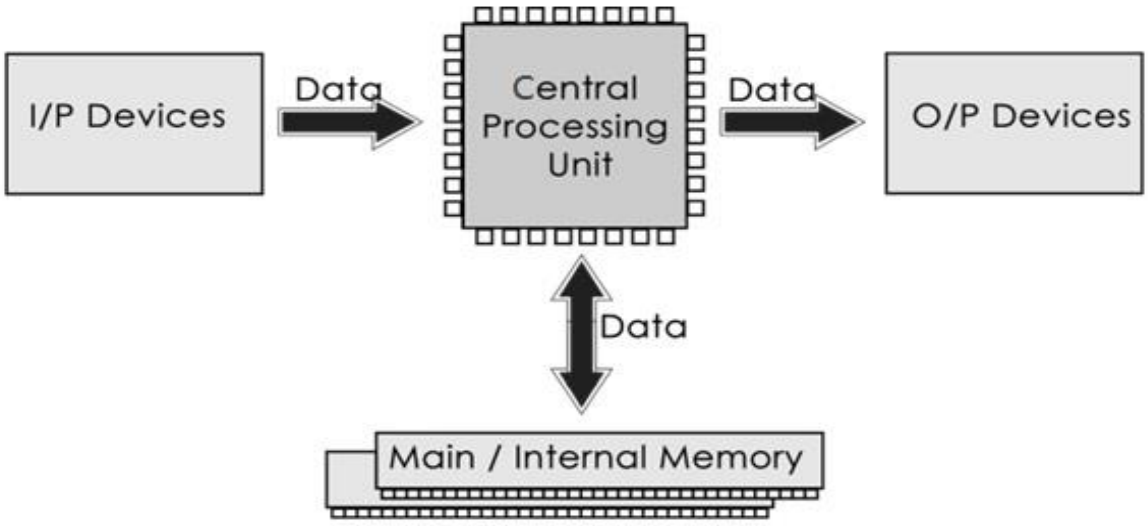
Aim: Familiarisation with computer hardware.

Introduction to Computer hardware: Physical identification of major components of a computer system such as mother board, RAM modules, daughter cards, bus slots, SMPS, internal storage devices, interfacing ports. Specifications of desktop and server class computers. Installation of common operating systems for desktop and server use.

Description:

a) Introduction to Computer hardware

Components of a computer system are the primary elements which make the functioning of an electronic device smooth and faster. Computer systems consist of three components as shown in below image: **Central Processing Unit, Input devices and Output devices**. Input devices provide data input to processor, which processes data and generates useful information that's displayed to the user through output devices. This is stored in computer's memory.



The operations of computer components are given below:

1) Inputting: It is the process of entering raw data, instructions and information into the computer. It is performed with the help of input devices.

2) Storing: The computer has primary memory and secondary storage to store data and instructions. It stores the data before sending it to CPU for processing and also stores the processed data before displaying it as output.

3) Processing: It is the process of converting the raw data into useful information. This process is performed by the CPU of the computer. It takes the raw data from storage, processes it and then sends back the processed data to storage.

4) Outputting: It is the process of presenting the processed data through output devices like monitor, printer and speakers.

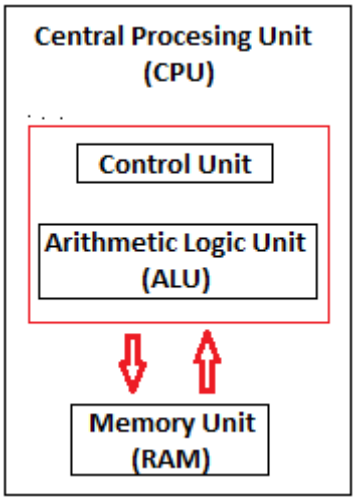
5) Controlling: This operation is performed by the control unit that is part of CPU. The control unit ensures that all basic operations are executed in a right manner and sequence.

Central Processing Unit (CPU)

A Central Processing Unit is also called a processor, central processor, or microprocessor. It carries out all the important functions of a computer. It receives instructions from both the hardware and active software and produces output accordingly. It stores all important programs like operating systems and application software. CPU also helps Input and output devices to communicate with each other. Owing to these features of CPU, it is often referred to as the brain of the computer.

CPU has three components:

- ALU (Arithmetic Logic Unit)
- Control Unit
- Memory or Storage Unit



b) Physical identification of major components

Hardware, which is abbreviated as HW, refers to all physical components of a computer system, including the devices connected to it. You cannot create a computer or use software without using hardware. The screen on which you are reading this information is also a hardware

1. Motherboard: The motherboard is generally a thin circuit board that holds together almost all parts of a computer except input and output devices. All crucial hardware like CPU, memory, hard drive, and ports for input and output devices are located on the motherboard. It is the biggest circuit board in a computer chassis. It allocates power to all hardware located on it and enables them to communicate with each other. It is meant to hold the computer's microprocessor chip and let other components connect to it. Each component that runs the

computer or improves its performance is a part of the motherboard or connected to it through a slot or port.

Fig.1

2. RAM Modules: RAM stands for Random Access Memory. It is also called the main memory. RAM is a temporary data storage device in computers and other devices. SRAM,DRAM, SDRAM, DDR etc are the various types of RAM available. A memory module or RAM stick is a narrow-printed circuit board that holds memory chips (RAM chips).

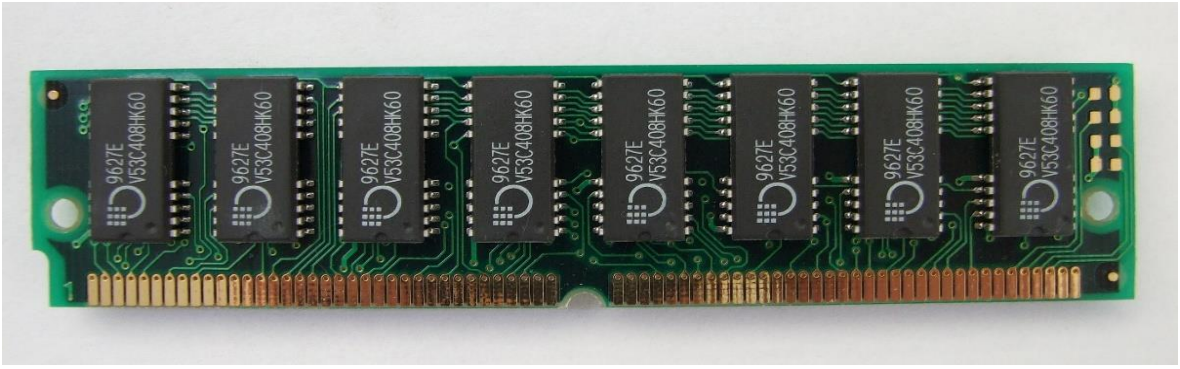


Fig.2

SRAM (static RAM) is random access memory (RAM) that retains data bits in its memory as long as power is being supplied. You can lose data if your SRAM is not powered. SRAM does not offer to refresh programs. SRAM has a low storage capacity(about 1MB).

Dynamic random access memory (DRAM) is a type of semiconductor memory that is typically used for the data or program code needed by a computer processor to function. The advantage of a DRAM is it only requires a single transistor compared to around six in a typical static RAM, SRAM memory cell. The costs of DRAM are much lower than those for SRAM,and they are able to provide much higher levels of memory density(about 1GB).

SDRAM (synchronous DRAM) is a generic name for various kinds of DRAM that are synchronized with the clock speed that the microprocessor is optimized for. That is same external clock pulse can be used to operate both SRAM and processor. This tends to increase the number of instructions that the processor can perform in a given time.

DDR Stands for "Double Data Rate." It is an advanced version of SDRAM, DDR-SDRAM can transfer data twice as fast as regular SDRAM chips. This is because DDR memory can send and receive signals twice per clock cycle. DDR operates about 2.5 V and DDR2 averages about 1.8 V, with DDR3 the voltage is reduced to 1.5 V.DDR3 has transfer rates between 800MT/s and 1600MT/s.DDR4 is the latest generation of DDR. It has the lowest operating voltage of 1.2 V and has higher transfer rates than previous generations. DDR5 launch speeds delivery nearly double the bandwidth of DDR4. It also enables scaling memory performance without degrading channel efficiency at higher speeds under real-world conditions. Crucial DDR5 memory will operate at 4800MT/s at launch, which is 1.5x the maximum standard DDR4 speed.

3. Daughter cards: A daughterboard (daughter card) is a type of circuit board that plugs in or is attached to the motherboard or similar expansion card to extend its features and services.

Fig.3

A daughterboard is connected directly to the motherboard. Like a motherboard, a daughterboard has sockets, pins, plugs and connectors to be attached to other boards.Today, these boards are not found or used in desktop computers. They were replaced with ISA card, PCI card and onboard options.With the rise of connective USB ports and other technology, it has become less necessary to upgrade devices with daughtercards or daughterboards.

4. Bus Slots: An expansion slot is a socket on the motherboard that is used to insert an expansion card, which provides additional features to a computer such as video, sound, advanced graphics, Ethernet or memory.

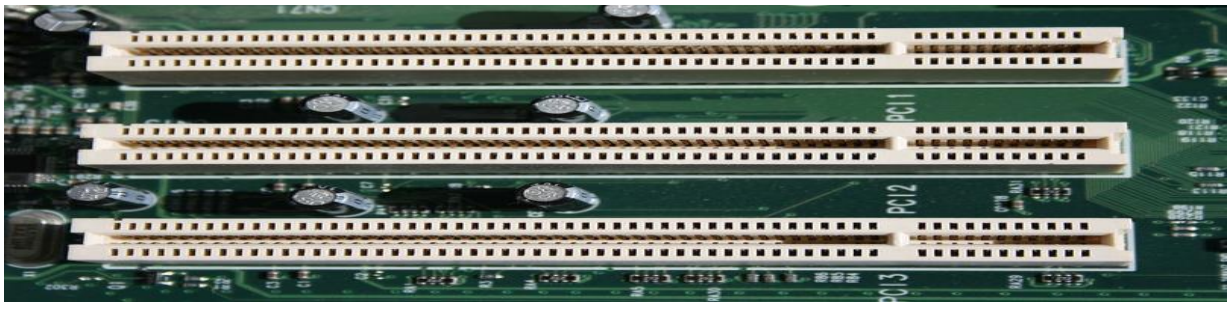


Fig.4

PATA stands for Parallel Advanced Technology Attachment and SATA stands for Serial Advanced Technology Attachment both are two bus interfaces used for connecting secondary storage devices like hard disks, optical drives.

Serial refers to the fact that data is sent one bit at a time down a single connection in each direction. There's a separate connection for data going in to and out of the device.Parallel refers to the fact that data is sent 16 bits at a time through a single 16-bit connection, which is used for data traveling in both directions.

SCSI stands for Small Computer System Interface is a set of standards for physically connecting and transferring data between computers and peripheral devices.

IDE stands for Integrated Drive Electronics is a standard interface for connecting a motherboard to storage devices such as hard drives and CD/DVD drives.

PCI (Peripheral Component Interconnect)is a bus standard that connects the computer motherboard and external devices. PCIe (peripheral component interconnect express) is an interface standard for connecting high-speed components. Every desktop PC motherboard has a number of PCIe slots.

5. SMPS: SMPS stands for Switched-Mode Power Supply. It is an electronic power supply that uses a switching regulator to convert electrical power efficiently.

Fig. 5

It is a power supply unit (PSU) generally used in computers to convert the voltage into the computer acceptable range. Technically briefing, an SMPS in a desktop system that converts 220V AC and 50HZ into +5V, -5V, +12V and +3.3 V DC at various electrical components in the computer.

6. Internal Storage: A storage device is an integral part of the computer hardware which stores information or data to process the result of any computational work. Internal storage is a storage device that's internal (inside the case) and is not a removable storage or external storage. For example, the hard drive inside your computer is an example of internal storage.HDD (Hard Disk Drive) also known as fixed disk uses magnetic tape for the storage of data. HDD has a moving read/write head to access data from storage like a gramophone player and slower to read and write. Unlike HDD, SSD (Solid State Drive) has no moving parts, and it obtains data from storage instantly. SSD can give faster performance than traditional magnetic-based computer storage devices. Although both of them perform the same task.

Fig 6.i

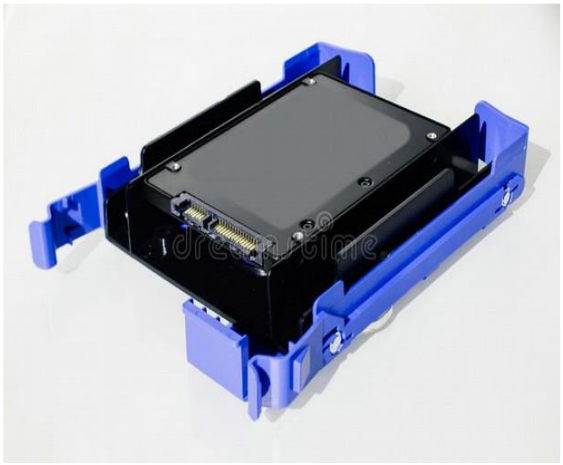


Fig 6.ii

7. Interfacing ports:A port is a connection or a jack provided on a computer to connect external or peripheral devices to the computer, for example, you will need a port on your device to connect a keyboard, mouse, pen-drives, etc. So, it acts as an interface or a point of attachment between computer and external devices. It is also called a communication port, as it is the point where you plug in a peripheral device to allow data transfer or communication between the device and computer. Generally, they are four to six in number and present on the back or sides of the computer.

Based on the type of protocol used for communication, computer ports can be of two types: Serial Ports and Parallel Ports.

i. Serial Ports

Fig 7.i

This type of ports provides an interface to connect to peripheral devices using a serial protocol. In this port, the rate of transmission of data is one bit at a time through a single communication line. For example, D-Subminiature or D-sub connector is a commonly used serial port, which carries RS-232 signals.

ii. Parallel Ports

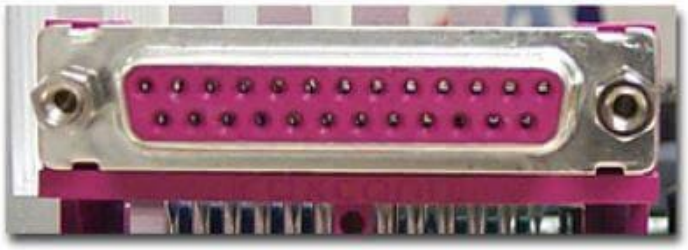


Fig 7.ii

As the name suggests, a parallel port is an interface that allows communication or data transfer between a computer and a device in a parallel manner through more than one communication line. For example, a printer port is a parallel port.

Some computer ports are:

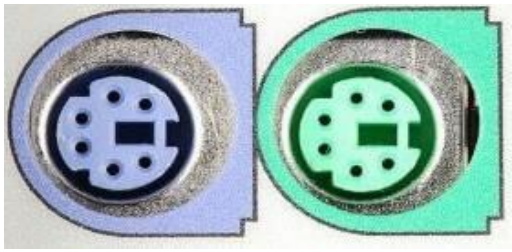


Fig:**PS/2**

Fig: **VGA**

Fig:**DVI**

Fig: **RJ-45**

Fig: **HDMI**

8. Specifications of desktop and server class:

Desktop:

Processor	Core i5
Processor speed	3.90 GHz
Number of cores	4
Typical Memory	32GB
Cache size	L1:8KB -1MB, L2:256KB - 3MB
Memory type	DDR4

Web Server:

Processor	Intel® Xeon® Bronze 3206R Processor (Multiple Processors)
Processor speed	1.90 GHz
Number of cores	8
Typical Memory	512GB
Cache size	L1:1-2MB, L2:8MB, L3:32-64MB
Memory type	DDR4

Result:

Familiarisation of computer and hardware done successfully.

LAB CYCLE:1

EXPERIMENT NO:2

Date:

Aim: Familiarisation of OS installation.

Oracle Virtual Box is already installed on your Desktop.

1. Create a Virtual Machine with sufficient RAM (nearly half the RAM of your Desktop).
2. Create a virtual hard disk (vmdk) of 20 GB (variable size).
3. Download Ubuntu 20.04 Desktop iso (64 bit) from https://ubuntu.com/download/desktop/thank-you?version=20.04.4&architecture=amd64 (or from http://10.2.7.253/downloads for quicker download)
4. Attach this iso to the Virtual Machine as CD Drive.
5. Boot from it. Execute gparted
6. Create 2 Primary partitions of equal size and install Ubuntu in one of them.
7. Make a copy of the vmdk file for later experiments.
8. You can also install another OS to the other partition.
   Write down a description of Virtualbox and vmdk format. Take the screenshots of Steps 1,2,4 and 6 and affix.

Solution:

VirtualBox is an open-source software for virtualizing the x86 computing architecture. It acts as a hypervisor, creating a VM (virtual machine) where the user can run another OS (operating system).

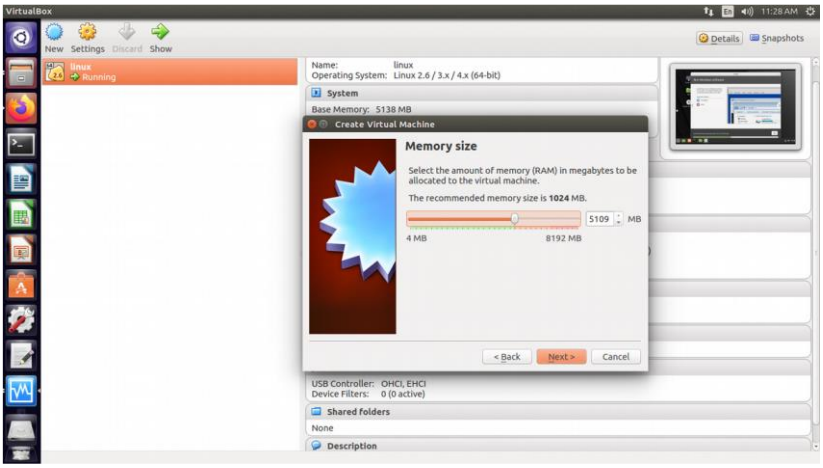
VMDK (short for Virtual Machine Disk) is a file format that describes containers for virtual hard disk drives to be used in VM.

Installation of Ubuntu using VirtualBox:

1.We have installed VirtualBox and you have downloaded the Ubuntu. Now set to install Linux in VirtualBox.

2.Start VirtualBox, and click on the New symbol. Give the virtual OS a relevant name.

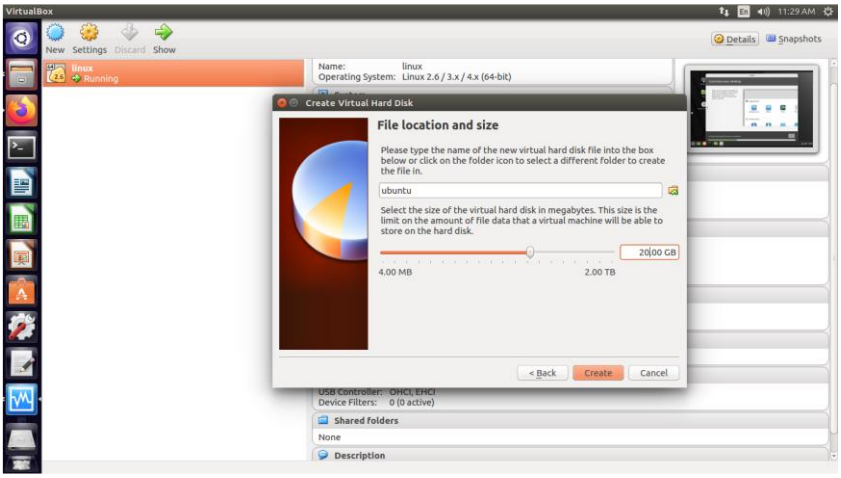
3.Allocate RAM to the virtual OS.



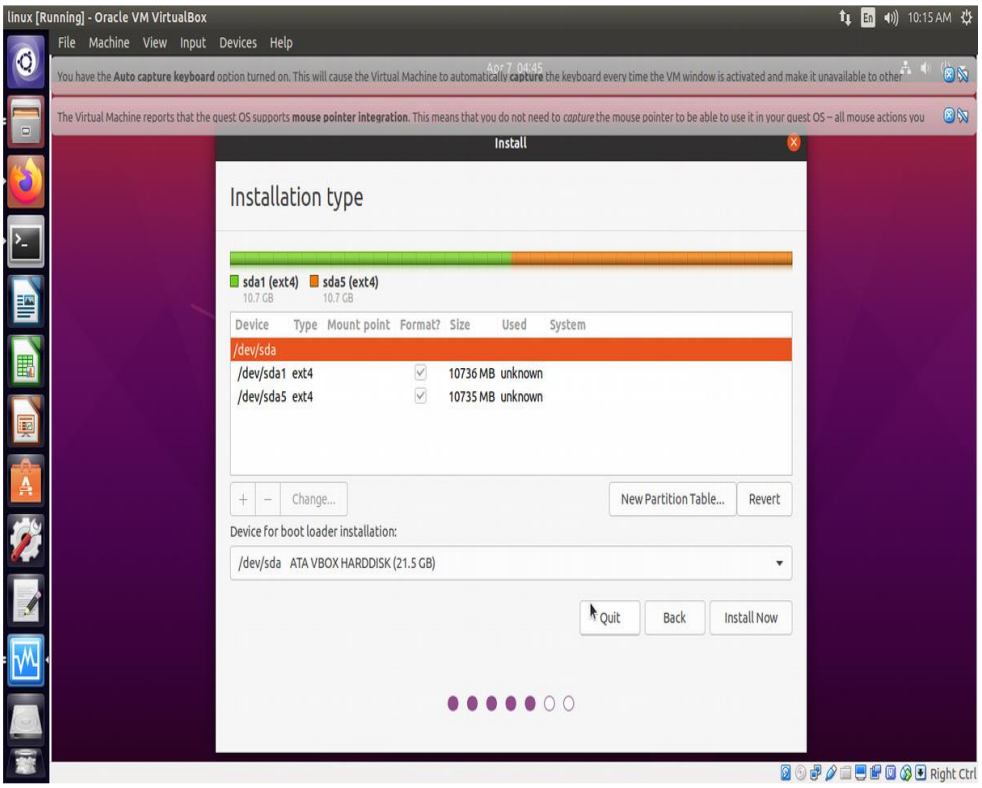
4.Create a virtual disk. This serves as the hard disk of the virtual Linux system.

5.Choose VDI file type as file type for new virtual hard disk.

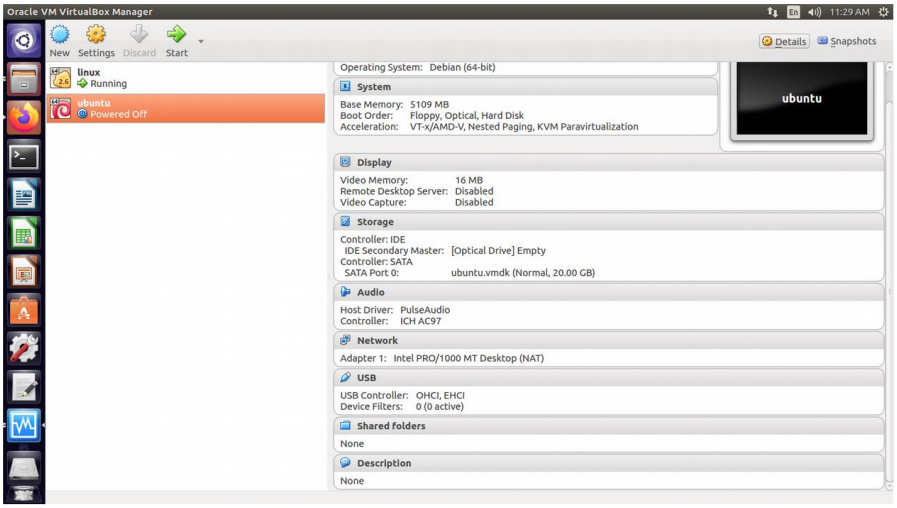
6.Choose either the “Dynamically allocated” or the “Fixed size” option for creating the virtual hard disk.
7.Allocate size for virtual hard disk (recommended size is 20GB).



8.Install Ubuntu
9.

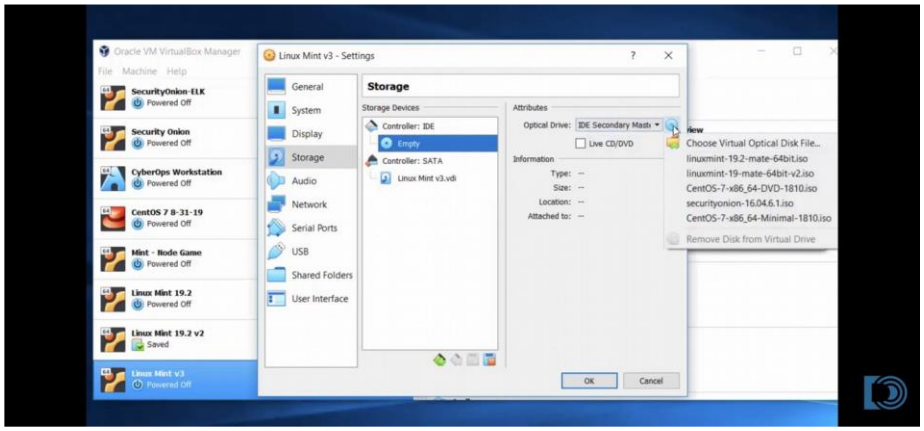


10. Open the VM box. It shows all the details and properties of the virtual box.

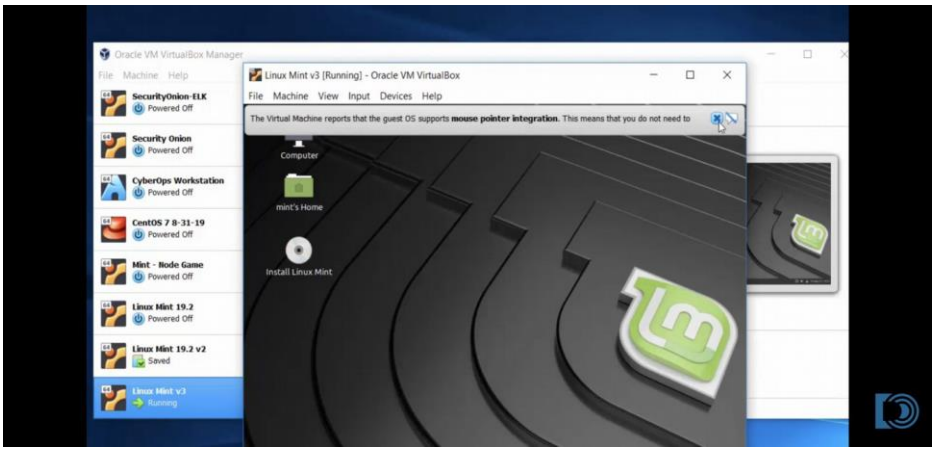


Installation of another OS (Linux Mint) to the other partition:

1. Download Linux Mint OS.
2. Choose



3.



4. Install linux mint.

Result:

Program executed successfully and output is verified.

LAB CYCLE:2

EXPERIMENT NO: 3

Date:

TEXT EDITOR

Aim: Study of a terminal based text editor such as Vim or Emacs.

There are many ways to edit files in Unix. Editing files using the screen-oriented text editor vi is one of the best ways. This editor enables you to edit lines in context with other lines in the file. An improved version of the vi editor which is called the VIM has also been made available now. Here, VIM stands for Vi IMproved. The vi editor was developed by William Joy as a more visual version of his own command line editor program called ex, which was becoming popular as a text editor.

The vi editor uses the terminal window for editing a file. For an example, run the following command to open a new file or an existing file of the same name:

vi filename

There are two ways of working in vi:

- Insert Mode: This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and placed in the file. To come out of the insert mode, press the Esc key, which will take you back to the command mode.
- Command Mode: This mode enables you to perform administrative tasks such as saving the files, executing the commands, moving the cursor, cutting (yanking) and pasting the lines or words, as well as finding and replacing. In this mode, whatever you type is interpreted as a command.

Commands and descriptions:

- vi filename
  #Creates a new file if it already does not exist, otherwise opens an existing file.
- vi -R filename
  #Opens an existing file in the read-only mode.
- view filename
  #Opens an existing file in the read-only mode.

Create a new file testfile if it already does not exist in the current working directory −

The above command will generate the following output −

The tilde (~) on each line following the cursor. A tilde represents an unused line. If a line does not begin with a tilde and appears to be blank, there is a space, tab, newline, or some other non-viewable character present.

a) Cursor operations

In command mode we can position the cursor anywhere in the file. Since you begin all basic edits (changing, deleting, and copying text) by placing the cursor at the text that you want to change, you want to be able to move the cursor to that place as quickly as possible.

There are vi commands to move the cursor:

- Up, down, left, or right—one character at a time.
- Forward or backward by blocks of text such as words, sentences, or paragraphs.
- Forward or backward through a file, one screen at a time.

An underscore marks the present cursor position. Circles show movement of the cursor from its current position to the position that would result from various vi commands.

Single Movements

The keys h, j, k, and l, right under your fingertips, will move the cursor:

h: Left, one space

j: Down, one line

k: Up, one line

l: Right, one space

You can also use the cursor arrow keys (←, ↓, ↑, →), + and - to go up and down, or the ENTER and BACKSPACE keys.

b) Manipulate text

1. Open a new or existing file with vim filename.
2. Type i to switch into insert mode so that you can start editing the file.
   Enter or modify the text with your file.
3. Once you're done, press the escape key Esc to get out of insert mode and back to command mode.
4. Type :wq to save and exit your file.

Vim commands for editing#:

Those who use Vim tend to use the term "yank" where most people would use the terms copy and paste. Therefore, the command for copying a word is yw, which stands for yank word, and the command for pasting whatever has been copied is p, meaning put. If you look up additional commands in the future, it can be confusing if you don't know what yank and put mean when using Vim.

You also have two options for how to select text. You can either use commands like dd, which deletes a single line, and yy, which copies a single line, or you can highlight text and then copy it to the unnamed register. The paste commands work the same whether you've highlighted text or used a command to automatically copy it.

As with movement commands, putting a number in front of the command can increase the number of times a task is completed. For instance, putting a number in front of yy will increase the number of lines copied, so 5yy will copy five lines.

yy - Copies a line

yw - Copies a word

y$ - Copies from where your cursor is to the end of a line

v - Highlight one character at a time using arrow buttons or the h, k, j, l buttons

V - Highlights one line, and movement keys can allow you to highlight additional lines

p - Paste whatever has been copied to the unnamed register

d - Deletes highlighted text

dd - Deletes a line of text

dw - Deletes a word

D - Deletes everything from where your cursor is to the end of the line

. - Repeats the last action

one

Vim commands for working with multiple files#:

You can also edit more than one text file at a time. Vim gives you the ability to either split your screen to show more than one file at a time or you can switch back and forth between documents. As with other functions, commands make going between documents or buffers, as they're referred to with Vim, as simple as a few keystrokes.

:bn - Switch to next buffer

:bp - Switch to previous buffer

:bd - Close a buffer

:sp [filename] - Opens a new file and splits your screen horizontally to show more than one buffer

:vsp [filename] - Opens a new file and splits your screen vertically to show more than one buffer

:ls - Lists all open buffers

Ctrl + ws - Split windows horizontally

Ctrl + wv - Split windows vertically

Ctrl + ww - Switch between windows

Ctrl + wq - Quit a window

Ctrl + wh - Moves your cursor to the window to the left

Ctrl + wl - Moves your cursor to the window to the right

Ctrl + wj - Moves your cursor to the window below the one you're in

Ctrl + wk - Moves your cursor to the window above the one you're in

c) Search for patterns:

Vim commands for searching text#:

Like many other text editors, Vim allows you to search your text and find and replace text within your document. If you opt to replace multiple instances of the same keyword or phrase, you can set Vim up to require or not require you to confirm each replacement depending on how you put in the command.

/[keyword] - Searches for text in the document where keyword is whatever keyword, phrase or string of characters you're looking for

?[keyword] - Searches previous text for your keyword, phrase or character string

n - Searches your text again in whatever direction your last search was

N - Searches your text again in the opposite direction

:%s/[pattern]/[replacement]/g - This replaces all occurrences of a pattern without confirming each one

:%s/[pattern]/[replacement]/gc - Replaces all occurrences of a pattern and confirms each

We can use the slash to search for a word, and then use the dot to replace it.

open the sample.txt file using the Vim editor:

$ vim sample.txt

we need to press the forward-slash(/) key, then search for the word “article”:

/article

This will highlight the first occurrence of the word “article” and we can press the Enter key to jump to it.

d) Global search and replace

We can use the slash to search for a word, and then use the dot to replace it.

open the file using the Vim editor:

$ vim filename

we need to press the forward-slash(/) key, then search for the word:

/word to be searched

This will highlight the first occurrence of the word and we can press the Enter key to jump to it.

We can then type in the cgn command combo:

cgn

This Vim editor command finds the last thing we searched for, deletes it, and then put us into insert mode.

We can then press the Esc key to return to normal mode.

Next, we need to press the “N” key to jump to the next occurrence of the word and press the Dot (.) key to auto-replace it with the word.

This is the simplest method to perform a basic search and replace in Vim editor.

Search and Replace Using the substitute Command

Basic syntax:

:s/<search_phrase>/<replace_phrase>/options

- All Occurrences
  :%s/article/tutorial/g
- Case-Insensitive
  :%s/vim/baeldung/gi
- With Confirmation
  :%s/article/tutorial/gc
- Within Specific Lines
  :start_line_number, end_line_number s/<search_term>/<replace_term>/g
- Whole Word
  :s/\<cover\>/go through/gi

Result:

Study of text editor has been done successfully.

LAB CYCLE:2

EXPERIMENT NO:4

Date:

BASIC LINUX COMMANDS

Aim: Basic Linux commands, familiarity with following commands/operations expected.

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

2. ls, echo, read

ls: is a Linux shell command that lists directory contents of files and directories. Some practical examples of ls command are shown below.

Options:

-t: It sorts the file by modification time, showing the last edited file first.

-l: To show long listing information about the file/directory.

-lh: To display file size in easy-to-read format.

-a: To show all the hidden files in the directory.

echo: used for displaying lines of text or string which are passed as arguments on the command line.

```
exam@CSCC2-19:~$ echo "welcome"
welcome
exam@CSCC2-19:~$
```

read: is used to read from a file descriptor.

```
exam@CSCC2-19:~$ read -p "Enter name" name
Enter name pradnya
exam@CSCC2-19:~$ echo "$name"
pradnya
exam@CSCC2-19:~$
```

3. more, less, cat

more: reads files and displays the text one screen at a time.

```
exam@CSCC2-19:~/mca-nsa$ more file1.txt
welcome
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
exam@CSCC2-19:~/mca-nsa$
```

less: used for filtering and viewing text files one screen page at a time.

```
exam@CSCC2-19:~/mca-nsa$ less file1.txt

[7]+  Stopped                 less file1.txt
exam@CSCC2-19:~/mca-nsa$
```

cat: used to display the content of a file, copy content from one file to another, concatenate the contents of multiple files, display the line number, display $ at the end of the line, etc.

```
exam@CSCC2-19:~/mca-nsa$ cat >file1.txt
welcome
exam@CSCC2-19:~/mca-nsa$ cat file1.txt
welcome
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
exam@CSCC2-19:~/mca-nsa$
```

4. cd, mkdir, pwd, find

cd: cd command in linux known as change directory command. It is used to change current working directory.

Syntax: cd [directory_name]

```
exam@CSCC2-19:~/mca-nsa$ cd ..
exam@CSCC2-19:~$ cd mca-nsa
exam@CSCC2-19:~/mca-nsa$
```

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

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

pwd: pwd stands for Print Working Directory. It prints the path of the working directory, starting from the root.

Syntax: pwd -L: Prints the symbolic path.

pwd -P: Prints the actual path.

```
exam@CSCC2-19:~/mca-nsa/d1$ pwd
/home/exam/mca-nsa/d1
exam@CSCC2-19:~/mca-nsa/d1$
```

find: The find command in UNIX is a command line utility for walking a file hierarchy. It can be used to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions.

Syntax: find [where to start searching from]

[expression determines what to find] [-options] [what to find]

```
exam@CSCC2-19:~/mca-nsa$ find file1.txt
file1.txt
exam@CSCC2-19:~/mca-nsa$
```

5. mv, cp, rm, tar

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

(i) It renames a file or folder.

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

Syntax: mv [Option] source destination

```
exam@CSCC2-19:~/mca-nsa$ mkdir d1
exam@CSCC2-19:~/mca-nsa$ mv file1.txt d1
exam@CSCC2-19:~/mca-nsa$ cd d1
exam@CSCC2-19:~/mca-nsa/d1$ ls
file1.txt
exam@CSCC2-19:~/mca-nsa/d1$
```

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

Syntax: cp [OPTION] Source Destination

```
exam@CSCC2-19:~/mca-nsa$ ls
d1  file.txt
exam@CSCC2-19:~/mca-nsa$ cp file.txt d1
exam@CSCC2-19:~/mca-nsa$ cd d1
exam@CSCC2-19:~/mca-nsa/d1$ ls
file1.txt  file.txt  mca-nsa
exam@CSCC2-19:~/mca-nsa/d1$
```

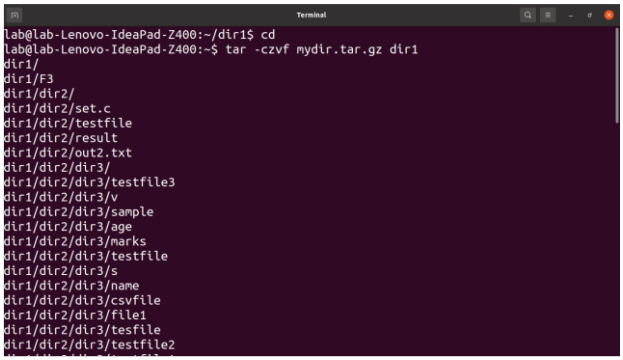
rm: rm stands for remove here. rm command is used to remove objects such as files, directories, symbolic links and so on from the file system like UNIX. To be more precise, rm removes references to objects from the filesystem, where those objects might have had multiple references.

Syntax: rm [OPTION]... FILE...

```
exam@CSCC2-19:~/mca-nsa/d1$ rm mca-nsa
exam@CSCC2-19:~/mca-nsa/d1$ ls
file1.txt  file.txt
exam@CSCC2-19:~/mca-nsa/d1$
```

tar: The Linux ‘tar’ stands for tape archive, is used to create Archive and extract the Archive files. tar command in Linux is one of the important commands which provides archiving functionality in Linux. We can use Linux tar command to create compressed or uncompressed Archive files and also maintain and modify them.

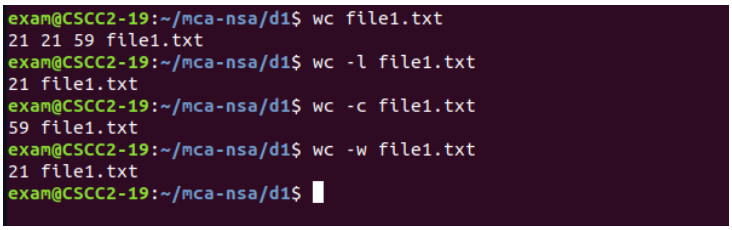
Syntax: tar [options] [archive-file] [file or directory to be archived]



6.wc, cut, paste

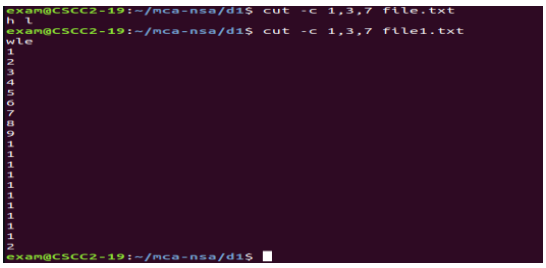
wc: wc stands for word count.It is used to find out number of lines, word count, byte and characters count in the files specified in the file arguments.

Syntax: wc [OPTION]... [FILE]...



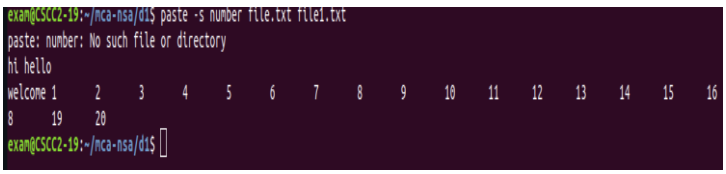
cut: The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by byte position, character and field.

Syntax: cut OPTION... [FILE]...



paste: paste command is one of the useful commands in Unix or Linux operating system. It is used to join files horizontally (parallel merging) by outputting lines consisting of lines from each file specified, separated by tab as delimiter, to the standard output.

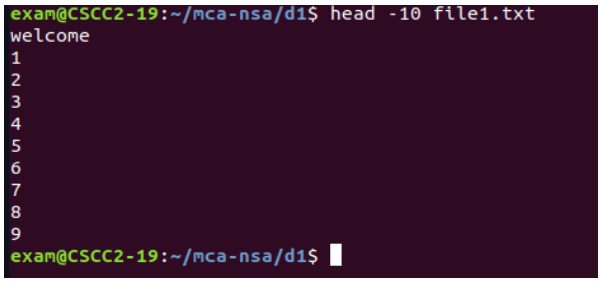
Syntax: paste [OPTION]... [FILES]...



7. head, tail, grep, expr:

head: it prints the first 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name.

Syntax: head [OPTION]... [FILE]...

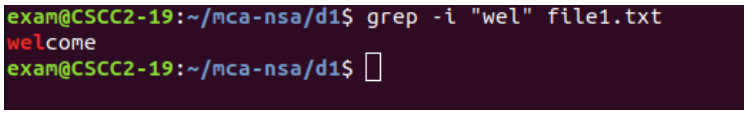


tail: it prints the last 10 lines of the specified files. If more than one file name is provided then data from each file is precedes by its file name.

Syntax: tail [OPTION]... [FILE]...

grep: the grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for global search for regular expression and print out).

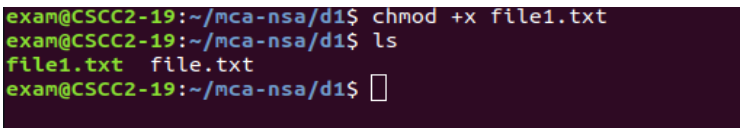
Syntax: grep [options] pattern [files]



8. chmod, chown:

chmod: the chmod command is used to change the access mode of a file.The name is an abbreviation of change mode.

Syntax: chmod [reference][operator][mode] file...



chown: it is used to set read, write, execute permission for user. To protect and secure files and directory in Linux we use permissions to control what a user can do with a file or directory.

Syntax: chown [OPTION]… [OWNER][:[GROUP]] FILE…

chown [OPTION]… –reference=RFILE FILE…

```
Number of days of warning before password expires       : 7
lab@lab-Lenovo-IdeaPad-Z400:/home$ sudo chown -R u_ser1:u_ser1 dir2
chown: cannot access 'dir2': No such file or directory
lab@lab-Lenovo-IdeaPad-Z400:/home$ sudo chown -R u_ser1:u_ser1 /home/lab/dir1/dir2
```

9. Redirections & Piping:

Redirection: whenever an individual runs a command, it can take input, give output, or do both. Redirection helps us redirect these input and output functionalities to the files or folders we want, and we can use special commands or characters to do so.

Append redirection

“>>” standard output

“<<” standard input

```
exam@CSCC2-19:~/mca-nsa/d1$ ls -l >>lsoutput
exam@CSCC2-19:~/mca-nsa/d1$ cat lsoutput
file1.txt
file.txt
lsoutput
total 12
-rwxrwxr-x 1 exam exam 59 Apr  5 10:43 file1.txt
-rw-rw-r-- 1 exam exam  9 Apr  5 10:54 file.txt
-rw-rw-r-- 1 exam exam 28 Apr  5 11:15 lsoutput
exam@CSCC2-19:~/mca-nsa/d1$
```

Overwrite Redirection

“>” standard output

“<” standard input

```
exam@CSCC2-19:~/mca-nsa/d1$ ls
file1.txt  file.txt
exam@CSCC2-19:~/mca-nsa/d1$ ls >lsoutput
exam@CSCC2-19:~/mca-nsa/d1$ cat lsoutput
file1.txt
file.txt
lsoutput
exam@CSCC2-19:~/mca-nsa/d1$
```

Piping: Pipe is used to combine two or more commands, and in this, the output of one command acts as input to another command, and this command’s output may act as input to the next command and so on. It can also be visualized as a temporary connection between two or more commands/ programs/ processes. The command line programs that do the further processing are referred to as filters.

Syntax: command_1 | command_2 | command_3 | .... | command_N

```
exam@CSCC2-19:~/mca-nsa/d1$ ls -l | more
total 12
-rwxrwxr-x 1 exam exam  59 Apr  5 10:43 file1.txt
-rw-rw-r-- 1 exam exam   9 Apr  5 10:54 file.txt
-rw-rw-r-- 1 exam exam 182 Apr  5 11:16 lsoutput
exam@CSCC2-19:~/mca-nsa/d1$
```

10. useradd, usermod, userdel, passwd:

useradd: is a command in Linux that is used to add user accounts to your system. It is just a symbolic link to adduser command in Linux and the difference between both of them is that useradd is a native binary compiled with system whereas adduser is a Perl script which uses useradd binary in the background.

Syntax: useradd [options] name_of_the_user

```
lab@lab-Lenovo-IdeaPad-Z400:/home$ sudo useradd -s /bin/bash -m -d /home/u_ser1 u_ser1
lab@lab-Lenovo-IdeaPad-Z400:/home$ sudo chage -l u_ser1
Last password change                                    : May 08, 2022
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : never
```

usermod: is a command in Linux that is used to change the properties of a user in Linux through the command line. After creating a user we have to sometimes change their attributes like password or login directory etc. so in order to do that we use the Usermod command.

Syntax: sudo usermod -c "This is test user" test_user

```
lab@lab-Lenovo-IdeaPad-Z400:/home$ sudo useradd -s /bin/bash -m -d /home/u_ser1 u_ser1
lab@lab-Lenovo-IdeaPad-Z400:/home$ sudo chage -l u_ser1
Last password change                                    : May 08, 2022
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : never
```

userdel: is used to delete a user account and related files. This command basically modifies the system account files, deleting all the entries which refer to the username LOGIN. It is a low-level utility for removing the users.

Syntax: userdel [options] LOGIN

```
lab@lab-Lenovo-IdeaPad-Z400:~$ sudo userdel u_ser1
```

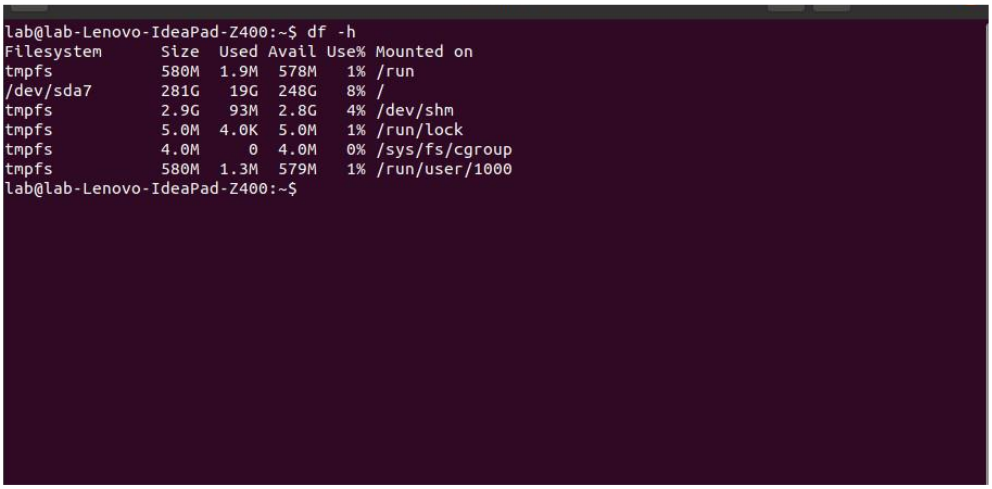
passwd: passwd command in Linux is used to change the user account passwords. The root user reserves the privilege to change the password for any user on the system, while a normal user can only change the account password for his or her own account.

Syntax: passwd [options] [username]

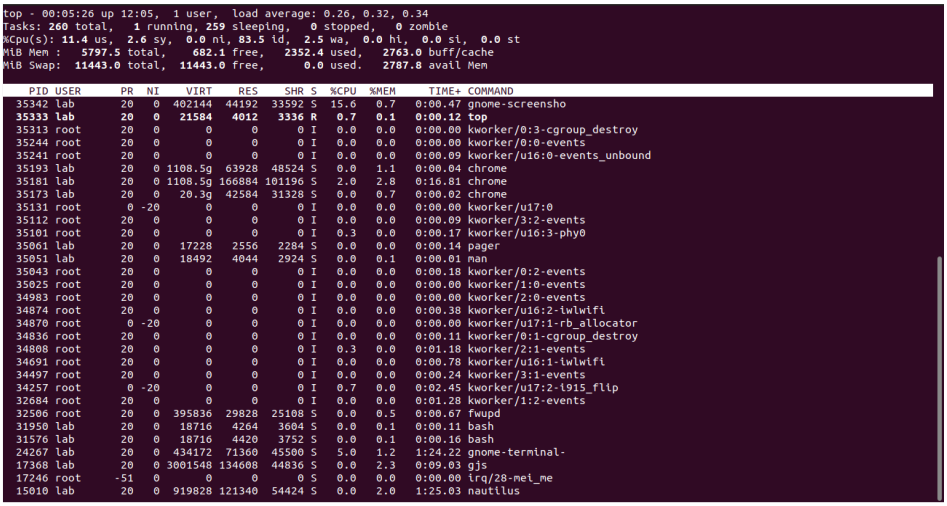
11. df,top, ps:

df: is used to displays the amount of disk space available on the file system containing each file name argument.

Syntax: df [OPTION]...[FILE]...

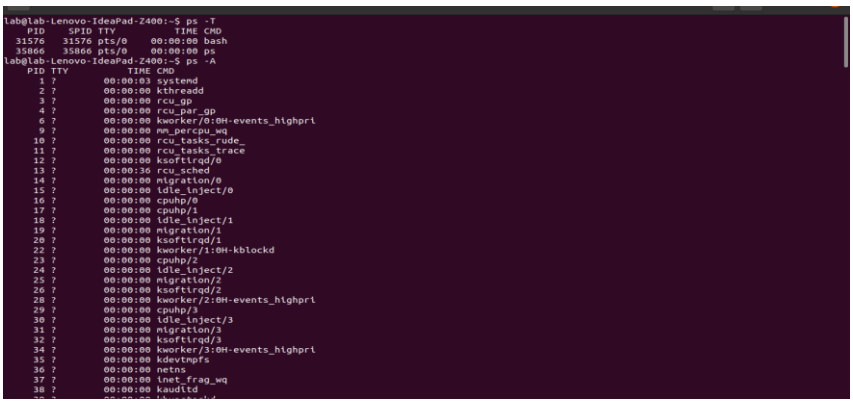


top: top command is used to show the Linux processes. It provides a dynamic real-time view of the running system. Usually, this command shows the summary information of the system and the list of processes or threads which are currently managed by the Linux Kernel.



ps: it allows multiple processes to operate simultaneously without interfering with each other. Process is one of the important fundamental concept of the Linux OS.

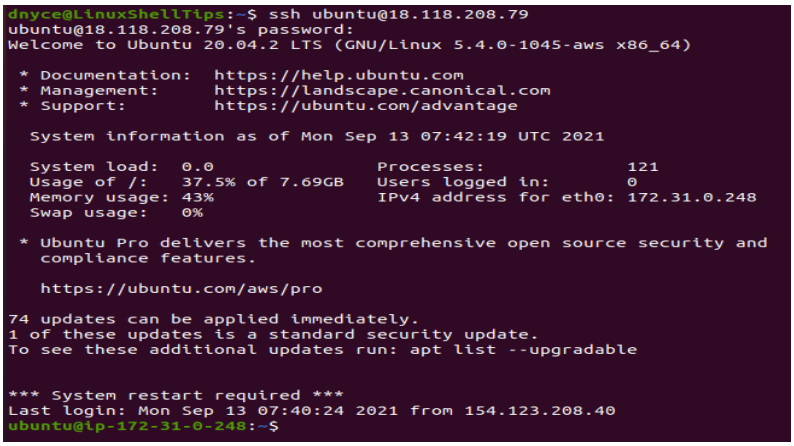
Syntax: ps [options]



12. ssh, scp, ssh-keygen, ssh-copy-id:

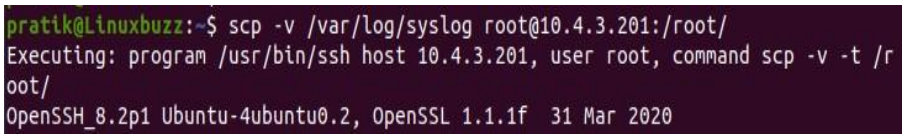
ssh: stands for “Secure Shell”. It is a protocol used to securely connect to a remote server/system. ssh is secure in the sense that it transfers the data in encrypted form between the host and the client. It transfers inputs from the client to the host and relays back the output. ssh runs at TCP/IP port 22.

Syntax: ssh user_name@host(IP/Domain_name)

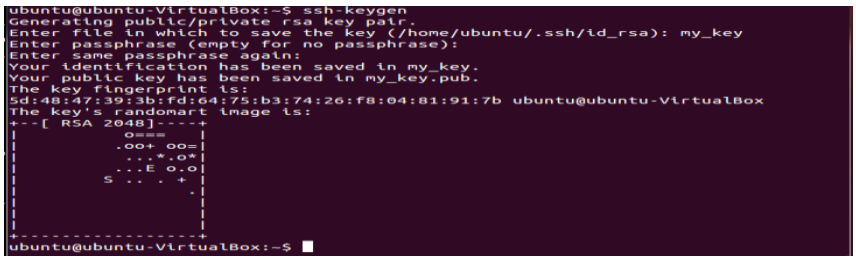


scp: scp (secure copy) command in Linux system is used to copy file(s) between servers in a secure way. The SCP command or secure copy allows secure transferring of files in between the local host and the remote host or between two remote hosts. It uses the same authentication and security as it is used in the Secure Shell (SSH) protocol. SCP is known for its simplicity, security and pre-installed availability.

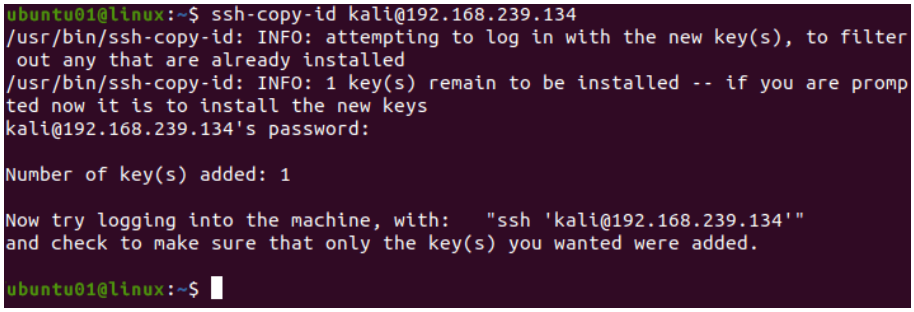
Syntax: scp [-346BCpqrTv] [-c cipher] [-F ssh_config] [-i identity_file] [-l limit] [-o ssh_option] [-P port] [-S program] [[user@]host1:]file1 … [[user@]host2:]file2



ssh-keygen: Secure Shell(SSH) is a cryptographic network protocol used for operating remote services securely. It is used for remote operation of devices on secure channels using a client-server architecture that generally operates on Port 22. SSH is the successor of Telnet. SSH uses public and private keys to validate and authenticate users. ssh-keygen is used to generate these key pairs.

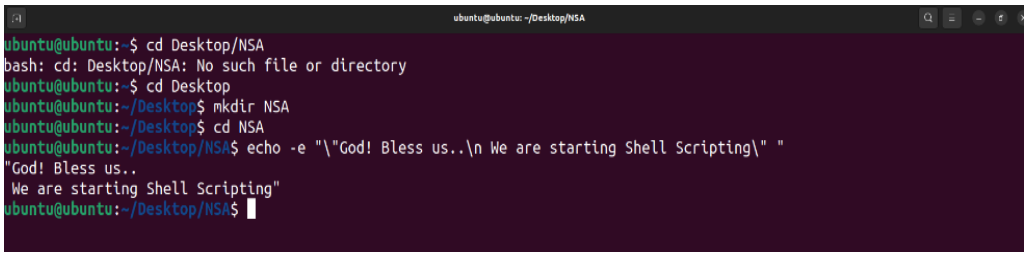


ssh-copy-id: is a command that copies the public key of the local host to the authorized keys file of the remote host, allowing password-less and automatic login via SSH.

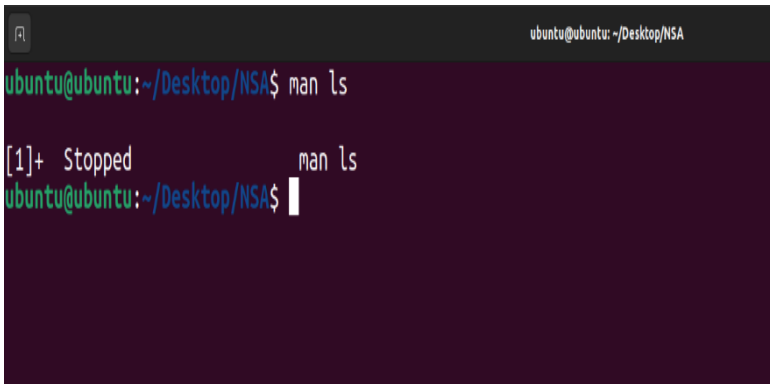


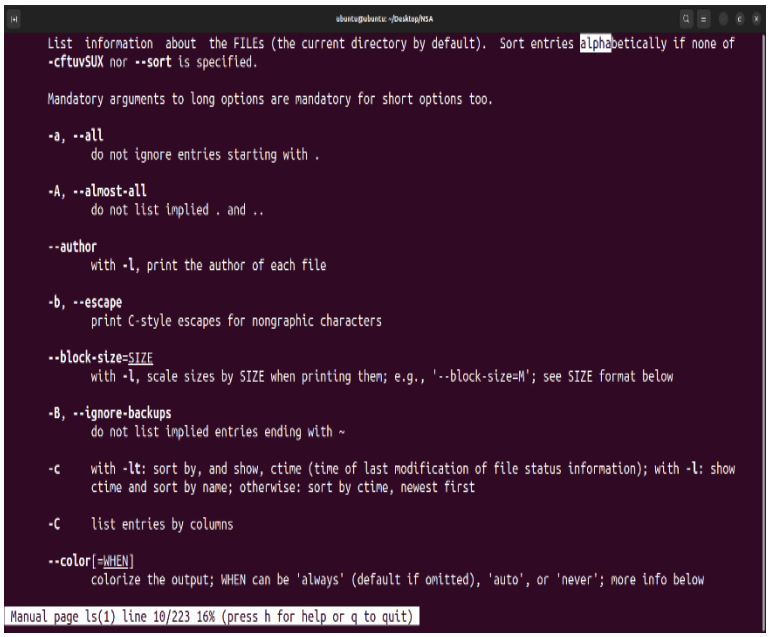
Basic Linux commands activity questions

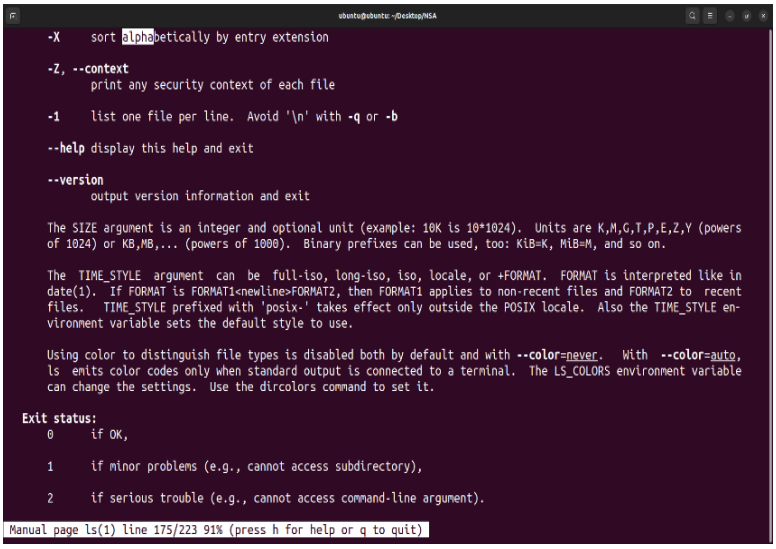
1. Command to display the following message as (Use “New line).
   “God!Bless us.
   We are starting Shell Scripting”



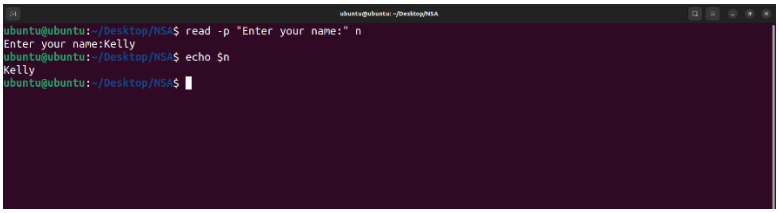
2. Get the manual page of ‘ls’ command. Search for the word “alpha”. Find the next occurrence and then find the previous occurrence.



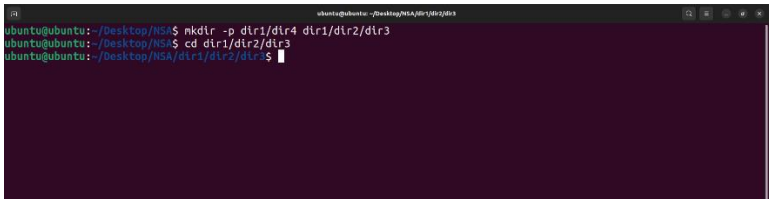




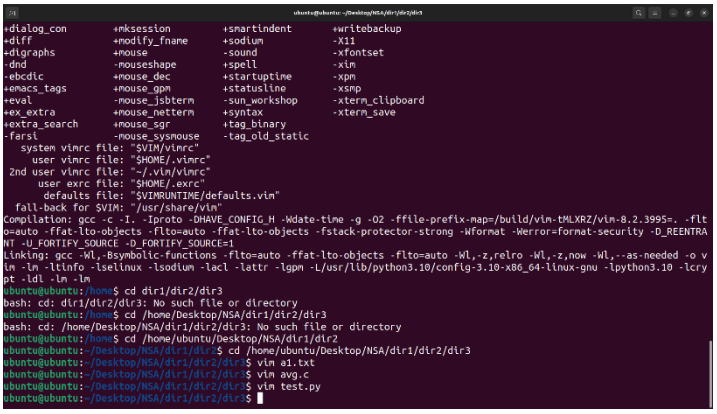
3. Read your name from the keyboard and display it.



4. Create the directory structure dir1/dir4 and dir1/dir2/dir3 with a single command and then change directory to dir3.



5. Create some files using **V**im.
   - Creating a file using vim:
   - Start vim by typing vim filename
   - To insert text press i
   - Now start editing text. Add new text or delete unwanted text.
   - One can press Esc and type :wq to save changes to a file and exit from vim]



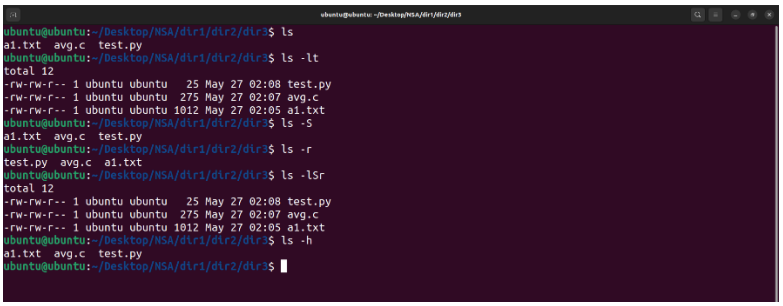
6. Display the current directory.

7. Listing files and folders.

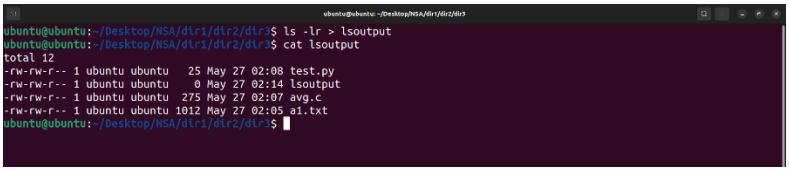
a. Listing the contents of dir1 (Qn.4) and all its descendants.

b. Listing the contents of dir3 (Qn.4) in

i. Alphabetical order
ii. Sorted on Time of modification, newest first
iii. Sorted on Size
iv. Reverse of all above
v. Long listing of files Sorted on Size with smallest first and size
vi. Displayed in human readable form

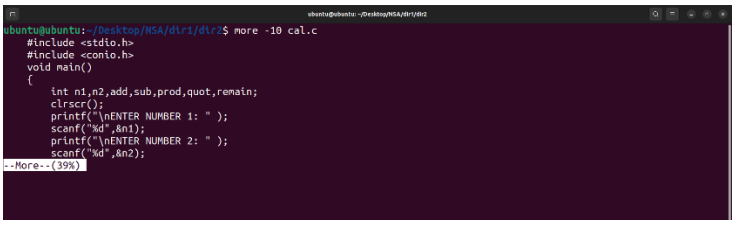


8. Execute **ls** and store the output to a file lsoutput .



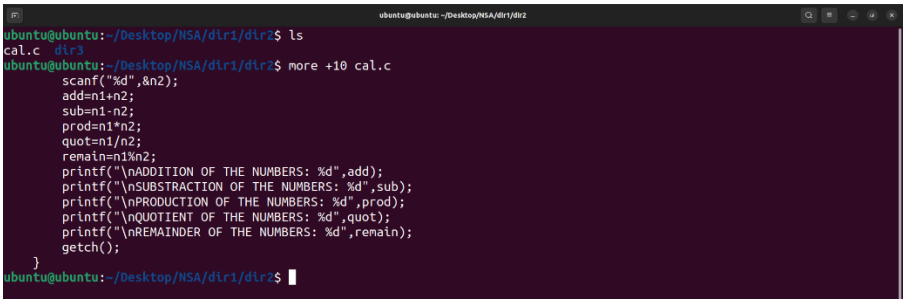
9. Display the file

a. starting with the first 10 lines and

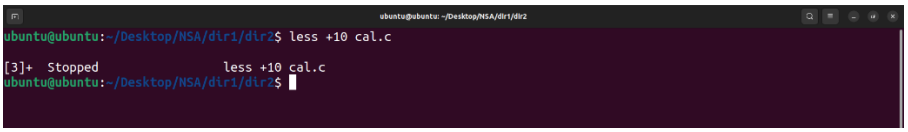


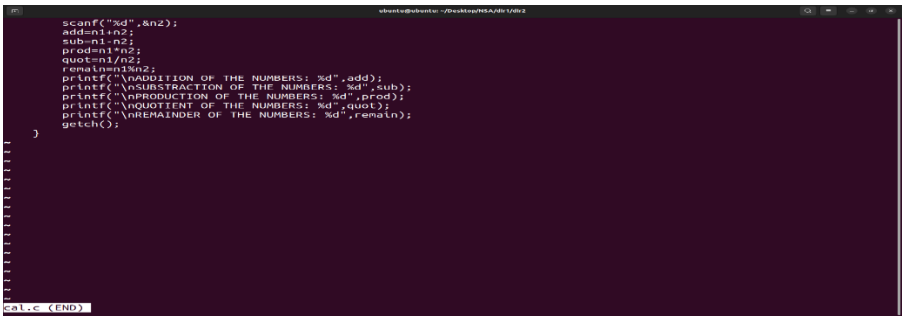
b. starting with the 10th line with provision for

i. Scrolling Up

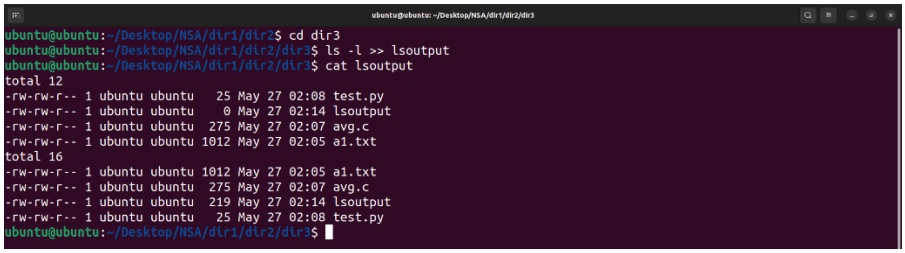


ii. Scrolling Up and Down

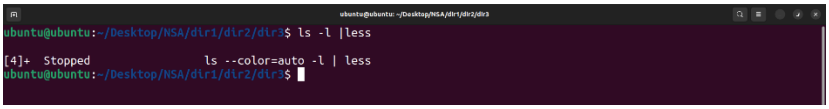


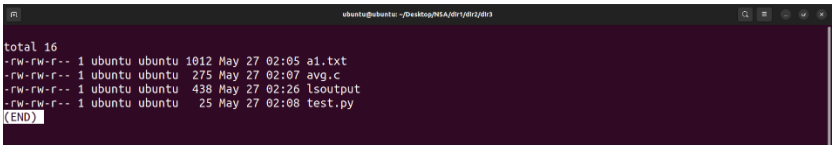


10. Execute **ls -l** and add the output to lsoutput, at the end.

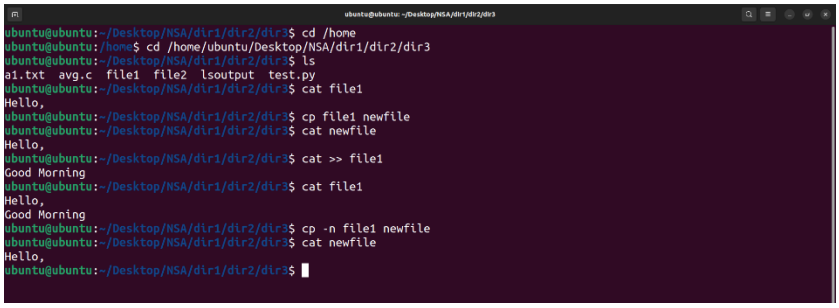


11. Execute **ls -l** and feed the result to less command, to scroll through the directory listing.

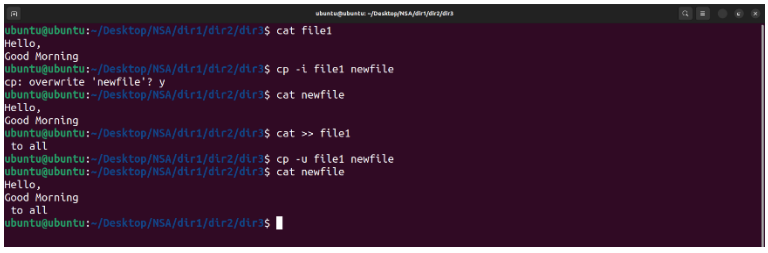




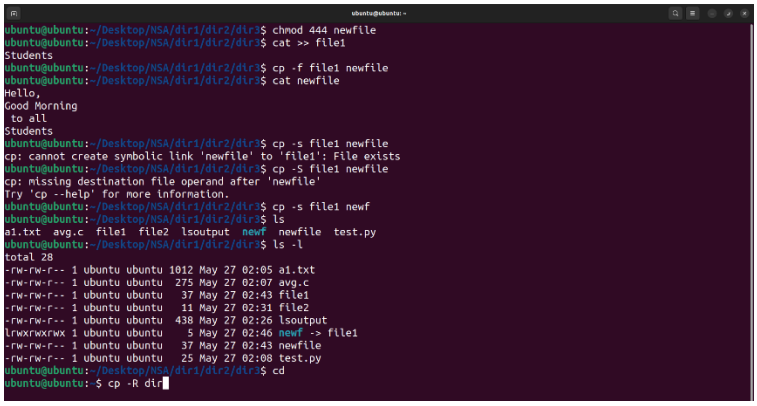
12. Copy the file file1 to newfile.
    a) If newfile already exists, it should be replaced.
    b) If newfile already exists, it should not be replaced.



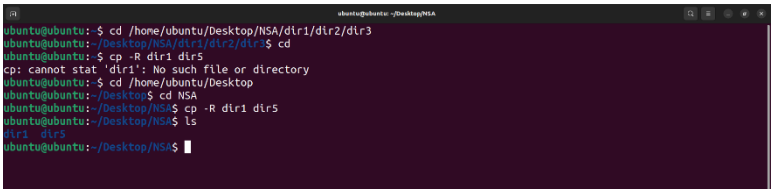
    c) If newfile already exists, it should be replaced, but only with the consent of the user.
    d) If newfile already exists, it should be replaced only if its contents is older than that of file1.



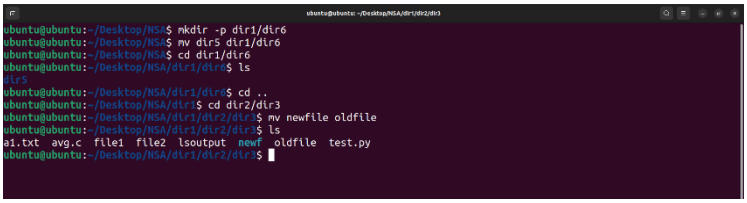
    e) Even if newfile is read only.
    f) Create a link instead of copying.



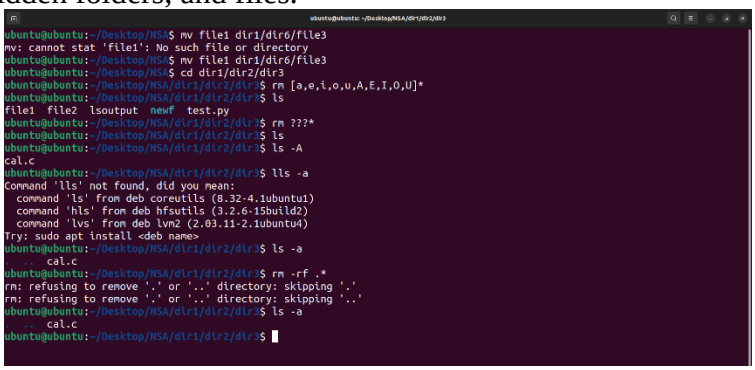
    g) Copy the entire directory tree from dir1 of Qn.4 to a new directory dir5



13. Create a new directory, dir6 inside dir1
    a) Move all files in dir5 into it.
    b) Delete all files where the name starts with a vowel character, upper or lower case.



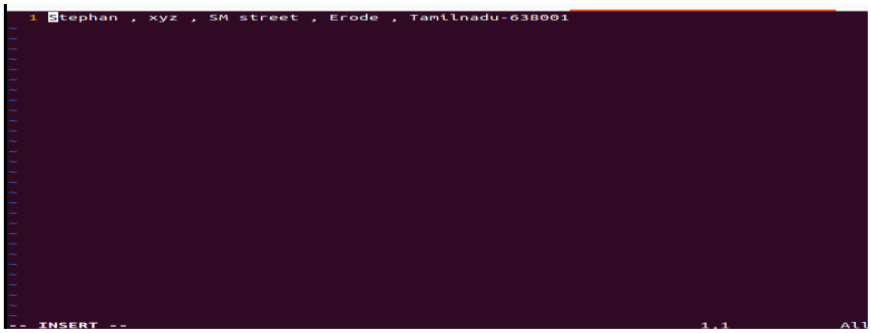
    c) Delete all files where the name is at least 3 characters long.
    d) Delete all hidden folders, and files.



14. Create a file testfile1 using Vim
    vim testfile1

a) Set line number
    vim testfile1
    Press esc
    Type :set number

b) Type your name and address with district and pincode

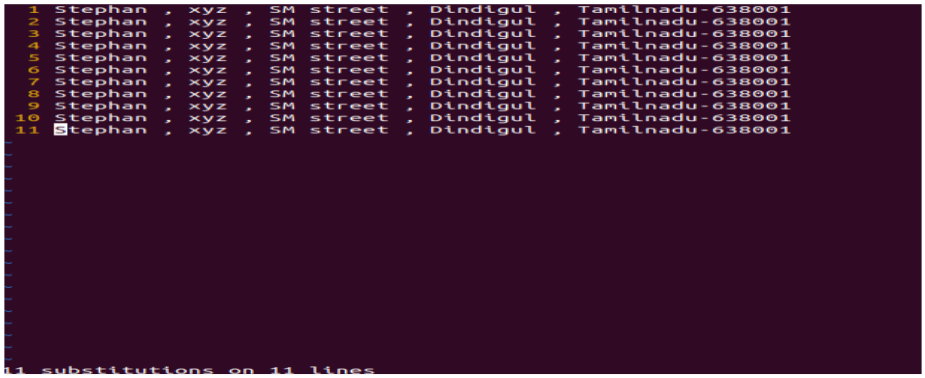


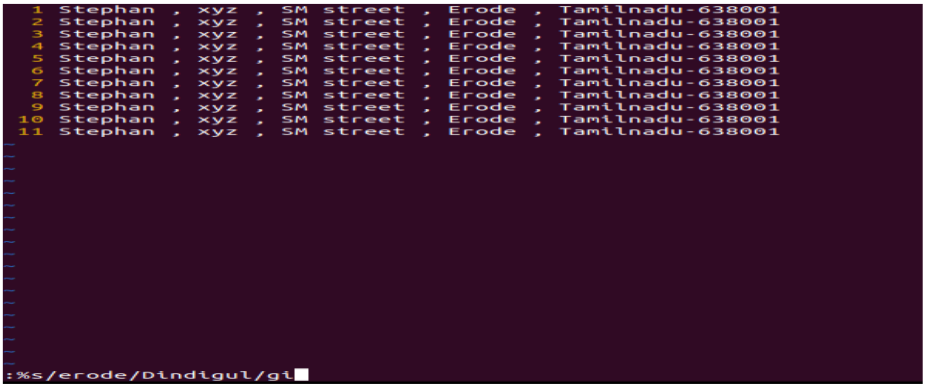
c) Copy paste the contents 10 times.

1. Place the cursor in the desired location(at beginning) .
2. Press Esc key then press ‘v’ [To change to visual mode] .
3. Press the ‘y’ key followed by movement command ,‘$’ .
4. Move the cursor to the location where you want to paste the contents.
5. Press ‘10p’ to paste the contents.

d) Replace all occurrence of your district with a neighbouring district.

1. Press Esc key [To change to normal mode].
2. Type “:%s/Erode/Dindigul/gi”





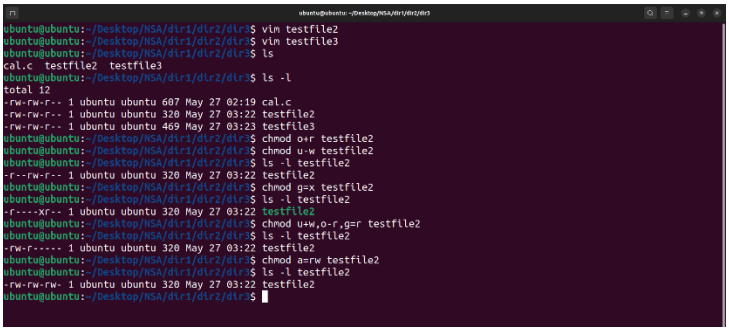
15. Create 2 files testfile2 and testfile3 using **Vim**.

a. Modify the permissions of testfile2 using symbolic mode.

i. Add read permission to others
ii. Revoke write from owner
iii. Set only execute to Group.

iv. Add write to owner, revoke read from others and set read only to group.

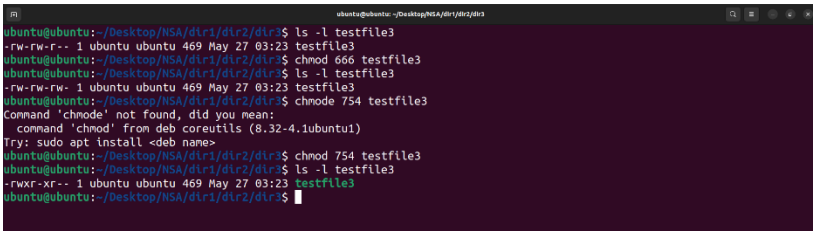
v. Set read and write to all.



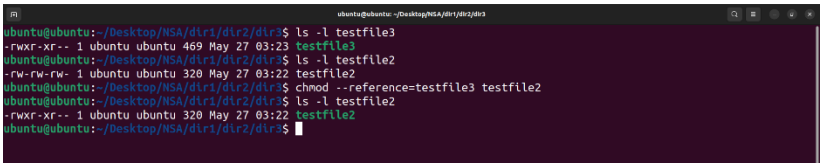
b. Modify the permissions of testfile3 using numeric mode

i. Set read and write to all.

ii. set read,write and execute to owner, read and execute to group and read only to others.



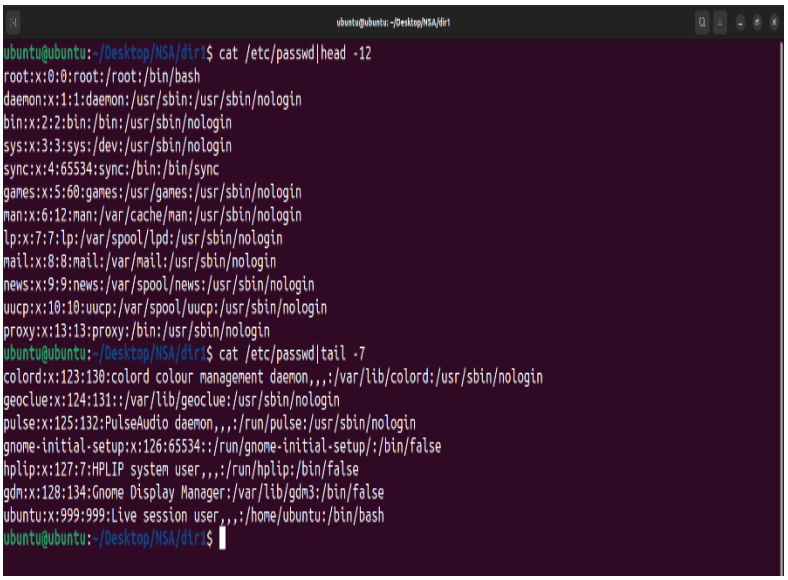
c. Set the permissions of testfile2 the same as that of testfile3



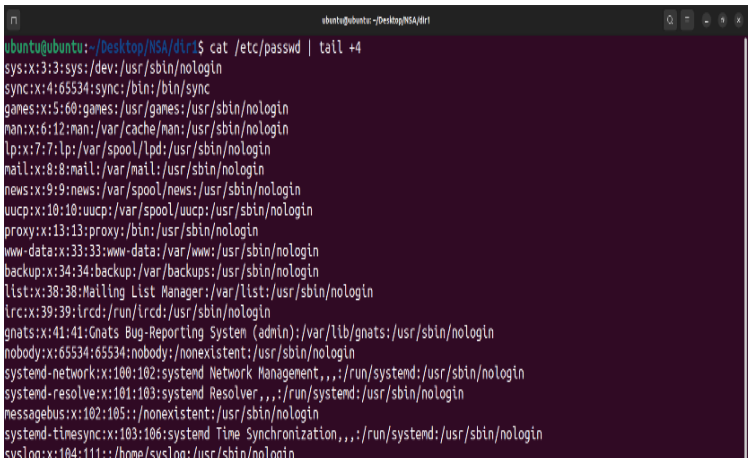
16. Use **head** and **tail** piped with cat /etc/passwd to display the details of

a. The first 12 users in the system.

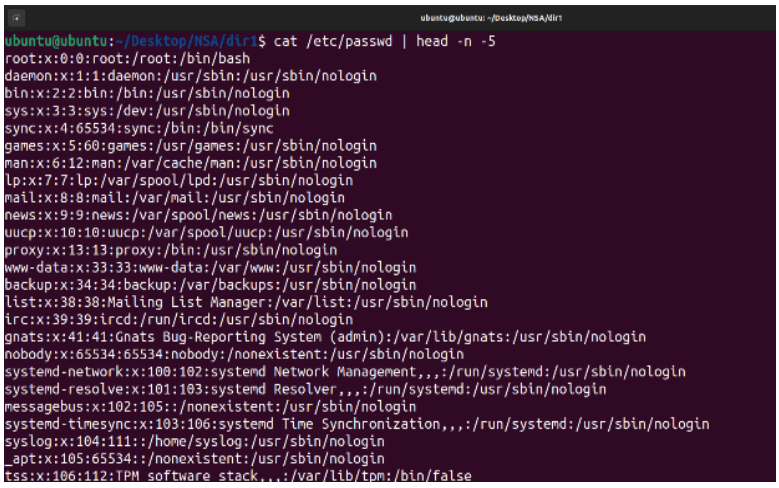
b. The last 7 users in the system.



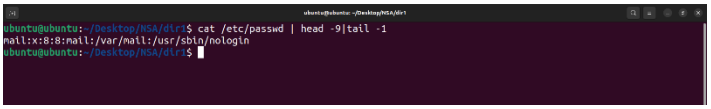
c. All but the first 3.



d. All but the last 5.



e. Only the 9 th.



17. Use **grep** to
    a. Display all lines in a file that contains the string “abc”
    b. Display all lines in a file that *does not* contain the string “abc”

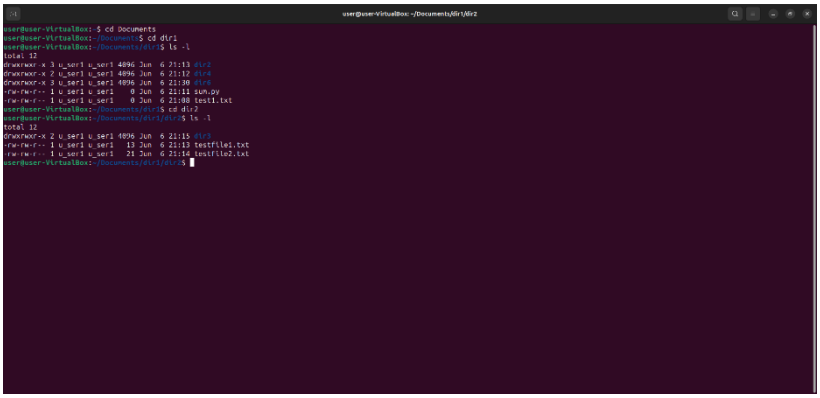
18. Using **expr**
    a. Read two integers X and Y. Display the sum, difference, product, quotient and remainder of these variables.
    b. Read a string, S, a position, p and a length l. Display the substring of length l starting at position p from the string S.

```
lab@lab-Lenovo-IdeaPad-Z400:~/dir1/dir6/dir5/dir2$ read -p "enter two numbers: " x y
enter two numbers: 12 5
lab@lab-Lenovo-IdeaPad-Z400:~/dir1/dir6/dir5/dir2$ echo "sum is : `expr $x + $y`"
sum is : 17
lab@lab-Lenovo-IdeaPad-Z400:~/dir1/dir6/dir5/dir2$ echo "Difference is : `expr $x - $y`"
Difference is : 7
lab@lab-Lenovo-IdeaPad-Z400:~/dir1/dir6/dir5/dir2$ echo "Product is : `expr $x \* $y`"
Product is : 60
lab@lab-Lenovo-IdeaPad-Z400:~/dir1/dir6/dir5/dir2$ echo "Quotient is : `expr $x / $y`"
Quotient is : 2
lab@lab-Lenovo-IdeaPad-Z400:~/dir1/dir6/dir5/dir2$ echo "Reminder is : `expr $x % $y`"
Reminder is : 2
lab@lab-Lenovo-IdeaPad-Z400:~/dir1/dir6/dir5/dir2$ read -p "enter a string: " s
enter a string: notebook
lab@lab-Lenovo-IdeaPad-Z400:~/dir1/dir6/dir5/dir2$ read -p "enter the position: " p
enter the position: 5
lab@lab-Lenovo-IdeaPad-Z400:~/dir1/dir6/dir5/dir2$ read -p "enter the length: " l
enter the length: 4
lab@lab-Lenovo-IdeaPad-Z400:~/dir1/dir6/dir5/dir2$ echo "Substring: `expr substr $s $p $l`"
Substring: book
lab@lab-Lenovo-IdeaPad-Z400:~/dir1/dir6/dir5/dir2$
```

19. a. Add a normal user, user1. Create (if it does not exist) the folder /user1 and set /user1 as the home directory of user1. Also set /bin/bash as the login shell *(Use a single command)*.
    b. Modify the user account of user1, to expire it after a specific date.

```
user@user-VirtualBox:~$ sudo useradd -s /bin/bash -m -d /home/u_ser1 u_ser1
[sudo] password for user:
user@user-VirtualBox:~$ sudo change -l u_ser1
sudo: change: command not found
user@user-VirtualBox:~$ sudo chage -l u_ser1
Last password change                                    : Jun 06, 2023
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : never
Minimum number of days between password change          : 0
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
user@user-VirtualBox:~$ sudo change -E 2022-06-9 u_ser1
sudo: change: command not found
user@user-VirtualBox:~$ sudo chage -E 2022-06-9 u_ser1
user@user-VirtualBox:~$ sudo chage -l u_ser1
Last password change                                    : Jun 06, 2023
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : Jun 09, 2022
Minimum number of days between password change          : 0
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
user@user-VirtualBox:~$ sudo chown -R u_ser1:u_ser1 /home/Documents/dir1/dir2
chown: cannot access '/home/Documents/dir1/dir2': No such file or directory
user@user-VirtualBox:~$ sudo chown -R u_ser1:u_ser1 /Documents/dir1/dir2
chown: cannot access '/Documents/dir1/dir2': No such file or directory
user@user-VirtualBox:~$ sudo chown -R u_ser1:u_ser1 /home/user/Documents/dir1/
user@user-VirtualBox:~$ cd ..
user@user-VirtualBox:/home$ ls -l
total 8
drwxr-x--- 18 user   user   4096 Jun  6 21:22 user
drwxr-x---  2 u_ser1 u_ser1 4096 Jun  6 22:06 u_ser1
user@user-VirtualBox:/home$ cd Documents
bash: cd: Documents: No such file or directory
user@user-VirtualBox:/home$ cd ..
user@user-VirtualBox:/$ cd Documents
bash: cd: Documents: No such file or directory
user@user-VirtualBox:/$
```

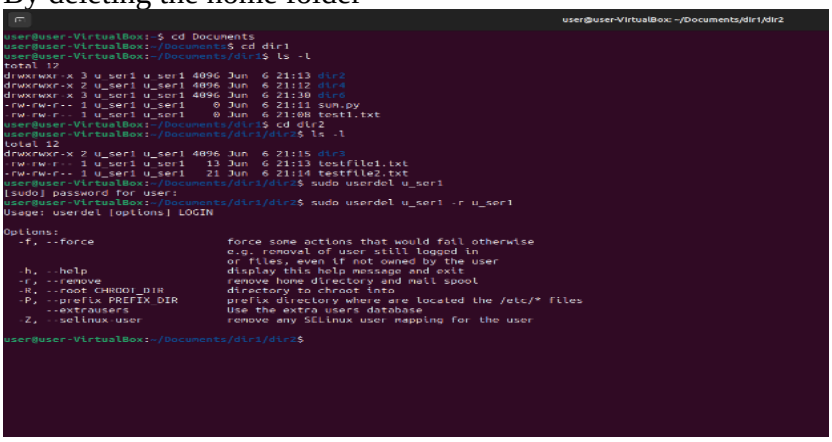
c. Change the owner and group of the directory tree from dir2 and all its contents to user1.



d. Delete the user account user1

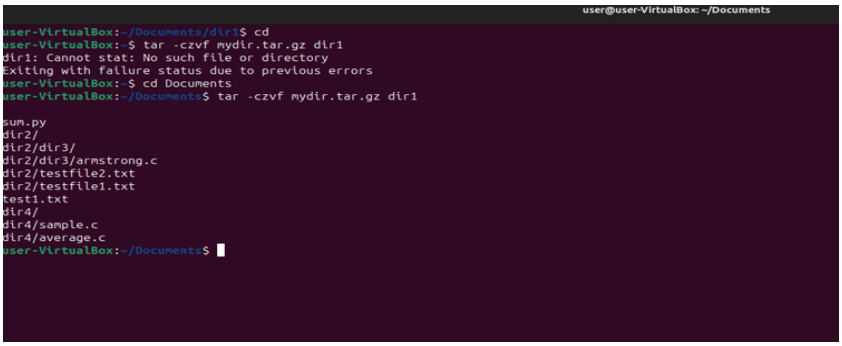
i. By retaining the home folder

ii. By deleting the home folder



20. Miscellaneous

a. Using **tar** create a tar.gz file of the folder dir1 of Qn.4 with the name mydir.tar.gz.



b. Extract the contents of mydir.tar.gz to dir6 of Qn.14.

```
user@user-VirtualBox:~/Documents$ tar -xvzf mydir.tar.gz -C dir1/dir6
dir1/
dir1/sum.py
dir1/dir2/
dir1/dir2/dir3/
dir1/dir2/dir3/armstrong.c
dir1/dir2/testfile2.txt
dir1/dir2/testfile1.txt
dir1/test1.txt
dir1/dir4/
dir1/dir4/sample.c
dir1/dir4/average.c
user@user-VirtualBox:~/Documents$
```

c. Use **top** to display processes sorted on

i. Process id

1. Type top

2. Press N

```
top - 21:34:09 up 49 min,  2 users,  load average: 0.09, 0.09, 0.09
Tasks: 231 total,   1 running, 230 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   4685.0 total,   1654.5 free,   1713.1 used,   1613.8 buff/cache
MiB Swap:   4096.0 total,   4096.0 free,      0.0 used.   2971.8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1642 user      20   0 5081956 525256 168664 S   1.7  10.9   2:36.78 gnome-shell
   2259 user      39  19  784244  32700  18944 S   0.3   0.7   0:01.05 tracker-miner-f
   2782 user      20   0  643092  57804  46032 S   0.3   1.2   0:05.48 gnome-terminal-
   5375 user      20   0   23024   5504   3328 R   0.3   0.1   0:00.02 top
      1 root      20   0  168440  12724   8884 S   0.0   0.3   0:01.77 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
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     11 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_kthread
     12 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_rude_kthread
     13 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_trace_kthread
     14 root      20   0       0      0      0 S   0.0   0.0   0:00.03 ksoftirqd/0
     15 root      20   0       0      0      0 I   0.0   0.0   0:00.78 rcu_preempt
     16 root      rt   0       0      0      0 S   0.0   0.0   0:00.02 migration/0
     17 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
     18 root      20   0       0      0      0 I   0.0   0.0   0:00.18 kworker/0:1-events
     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   0:00.26 migration/1
     23 root      20   0       0      0      0 S   0.0   0.0   0:00.03 ksoftirqd/1
     24 root      20   0       0      0      0 I   0.0   0.0   0:00.22 kworker/1:0-events
     25 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0H-events_highpri
     26 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/2
     27 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/2
     28 root      rt   0       0      0      0 S   0.0   0.0   0:00.26 migration/2
     29 root      20   0       0      0      0 S   0.0   0.0   0:00.03 ksoftirqd/2
     31 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/2:0H-events_highpri
     32 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/3
     33 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/3
```

ii. CPU%

1. Type top

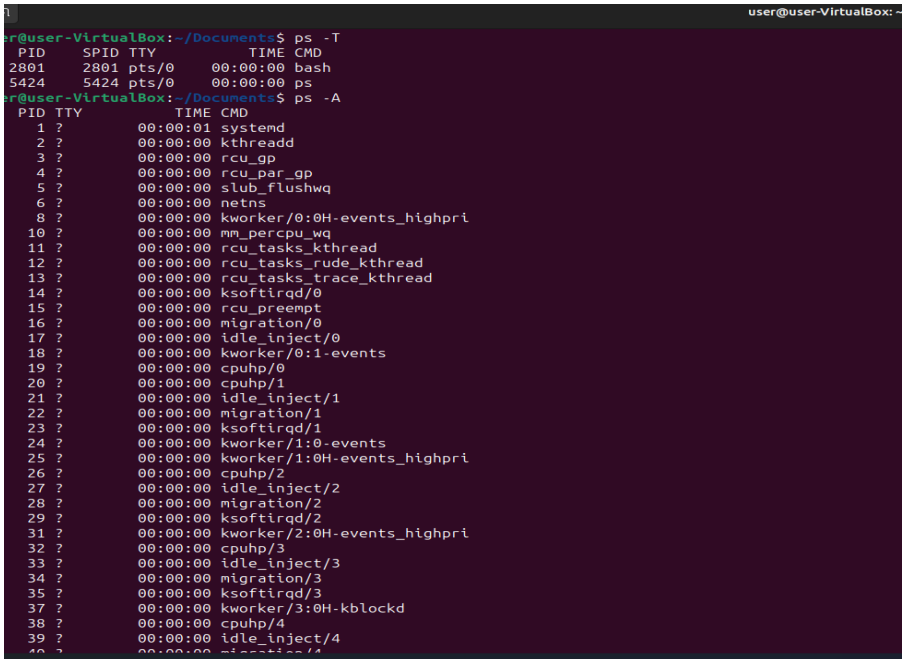
2. Press P

```
top - 21:33:24 up 48 min,  2 users,  load average: 0.09, 0.09, 0.10
Tasks: 229 total,   1 running, 228 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   4685.0 total,   1652.0 free,   1716.6 used,   1611.8 buff/cache
MiB Swap:   4096.0 total,   4096.0 free,      0.0 used.   2968.3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   5351 user      20   0   23024   5632   3456 R   0.3   0.1   0:00.03 top
   5350 root      20   0   16884   2176   2048 S   0.0   0.0   0:00.00 anacron
   5295 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/u12:4-ext4-rsv-conversion
   5294 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/u12:1-flush-8:0
   5258 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/4:1-events
   5250 root      20   0       0      0      0 I   0.0   0.0   0:00.01 kworker/3:1-events
   5248 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/1:2-cgroup_destroy
   5191 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/u12:5-flush-8:0
   5163 root      20   0       0      0      0 I   0.0   0.0   0:00.10 kworker/u12:0-events_power_efficient
   5153 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0-cgroup_destroy
   5107 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/5:1-events
   4230 user      20   0 2519536 242452 107860 S   0.0   5.1   0:24.80 gnome-text-edit
   4219 root      20   0       0      0      0 I   0.0   0.0   0:00.18 kworker/u12:2-events_power_efficient
   3539 user      20   0  868160  26368  14848 S   0.0   0.5   0:00.06 snap
   3269 root      20   0       0      0      0 I   0.0   0.0   0:00.16 kworker/u12:3-events_unbound
   3137 root      20   0       0      0      0 I   0.0   0.0   0:00.42 kworker/5:2-events
   3133 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/2:0
   2801 user      20   0   19604   4864   3584 S   0.0   0.1   0:00.06 bash
   2782 user      20   0  642860  57676  46032 S   0.0   1.2   0:05.30 gnome-terminal-
   2751 user      20   0  396408   8064   7296 S   0.0   0.2   0:00.01 gvfsd-recent
   2672 user      20   0 3029392  64592  48508 S   0.0   1.3   0:00.87 gjs
   2661 user      20   0 2673300 319420 124752 S   0.0   6.7   0:14.34 nautilus
   2552 user      20   0  575972  31492  24196 S   0.0   0.7   0:00.31 update-notifier
   2449 root      20   0  479412  33728  28232 S   0.0   0.7   0:00.62 fwupd
   2373 user      20   0  244796   6400   5888 S   0.0   0.1   0:00.01 gvfsd-metadata
   2302 user      20   0  160652   2076   1792 S   0.0   0.0   0:00.20 VBoxClient
   2301 user      20   0   28548   1564   1280 S   0.0   0.0   0:00.00 VBoxClient
   2299 user      20   0  227352   2824   2560 S   0.0   0.1   0:02.67 VBoxClient
   2294 user      20   0   28548   1544   1280 S   0.0   0.0   0:00.00 VBoxClient
   2288 user      20   0  226836   3080   2688 S   0.0   0.1   0:01.50 VBoxClient
   2286 user      20   0   28548   1544   1280 S   0.0   0.0   0:00.00 VBoxClient
   2266 user      20   0 1576112  86496  66968 S   0.0   1.8   0:00.65 mutter-x11-fram
   2259 user      39  19  784244  33008  18944 S   0.0   0.7   0:00.99 tracker-miner-f
   2251 user      20   0  275376  24700  18116 S   0.0   0.5   0:00.21 ibus-x11
```

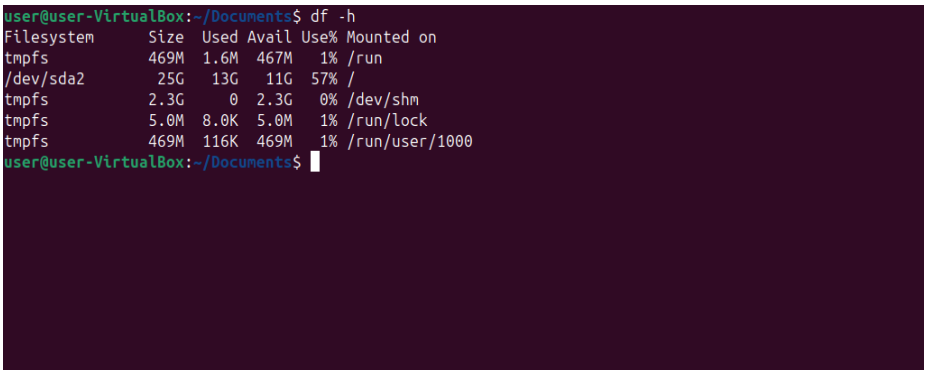
d. Use **ps** to display

i. Processes associated with the current terminal.

ii. All processes in the system.



e. Use **df** to display the storage available in each partition in human readable form.



Result:

Commands executed successfully and output is verified.

LAB CYCLE:3

EXPERIMENT NO: 5

Date:

FILE SYSTEM HIERARCHY, PERMISSION, CONFIGURATION FILES AND LOG FILES

Aim: a) File system hierarchy in a common Linux distribution,

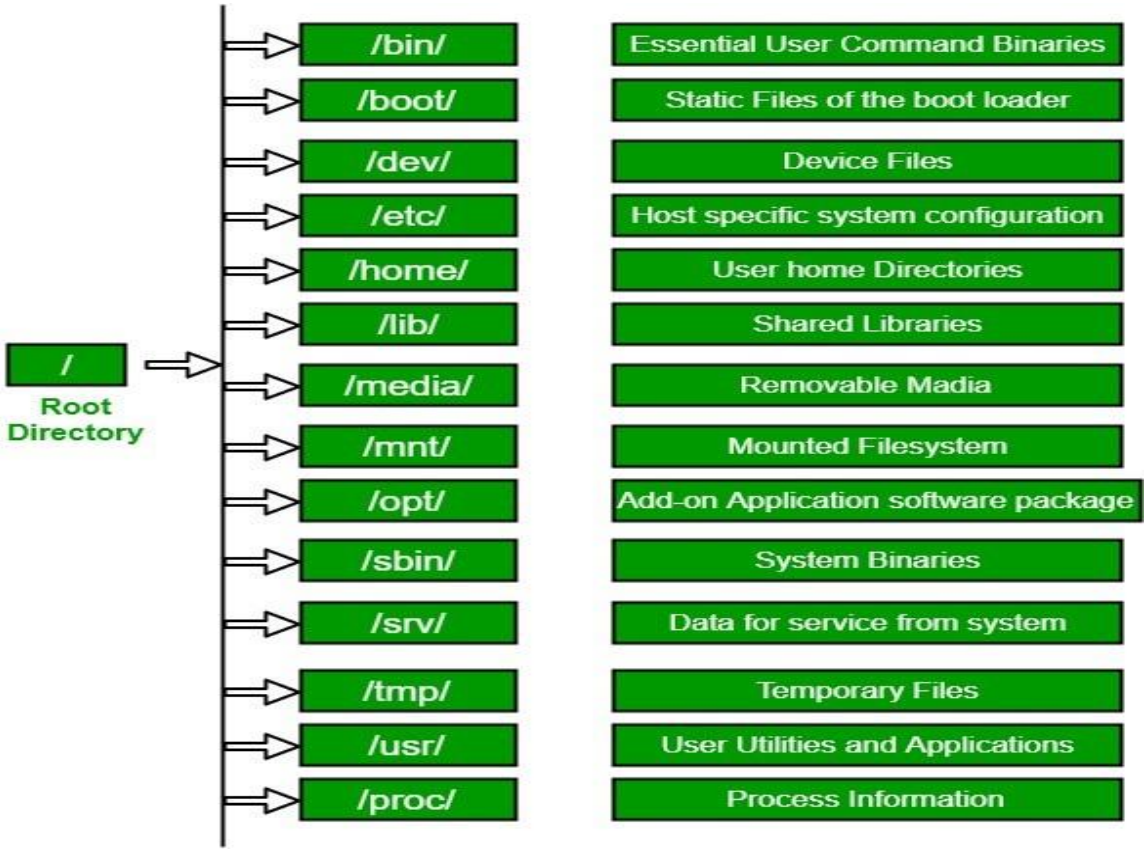
b) File and device permissions,

c) Study of system configuration files in /etc,

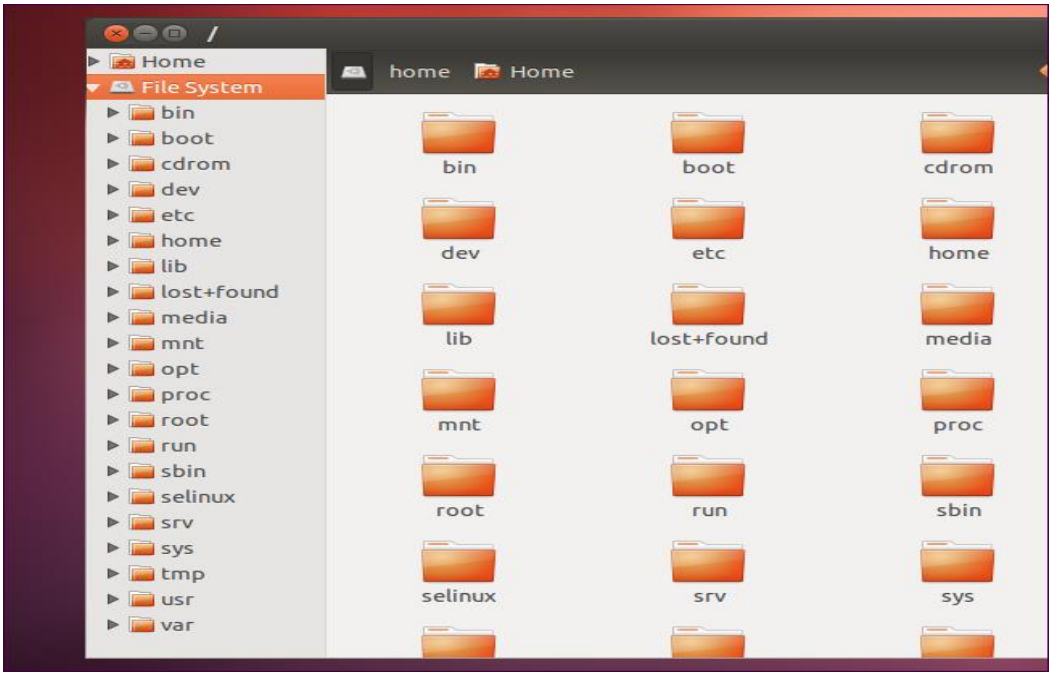
d) Familiarizing log files for system events, user activity,network events.

a) File system hierarchy in a common Linux distribution

The Linux File Hierarchy Structure or the File System Hierarchy Standard (FHS) defines the directory structure and directory contents in Unix-like operating systems. It is maintained by the Linux Foundation. In the FHS, all files and directories appear under the root directory /, even if they are stored on different physical or virtual devices.

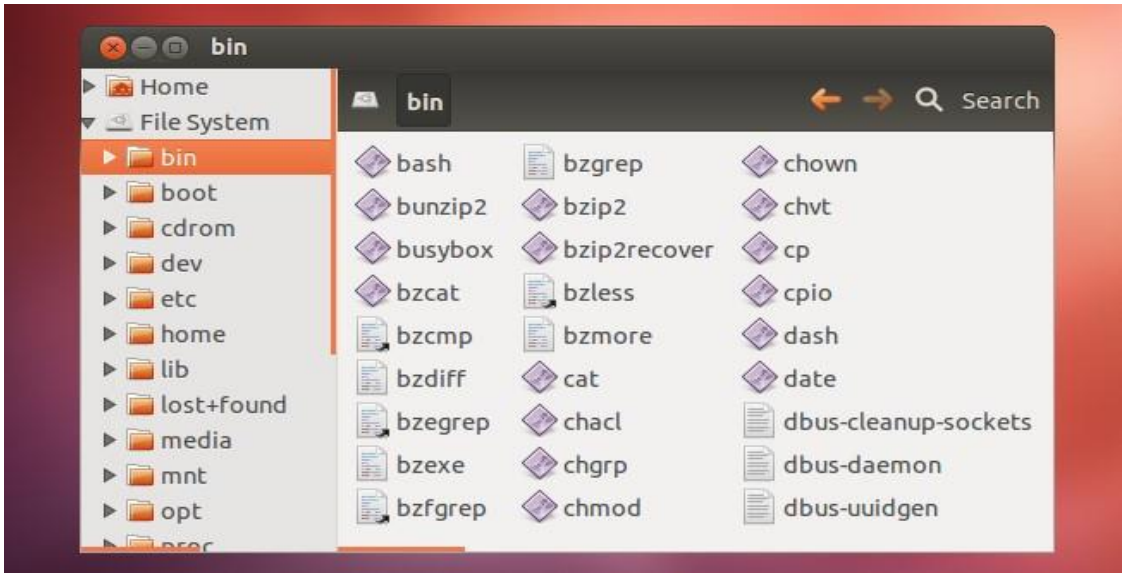


1. The Root Directory



- Every single file and directory starts from the root directory.
- The only root user has the right to write under this directory.
- /root is the root user's home directory, which is not the same as /.

2. /bin- User binaries

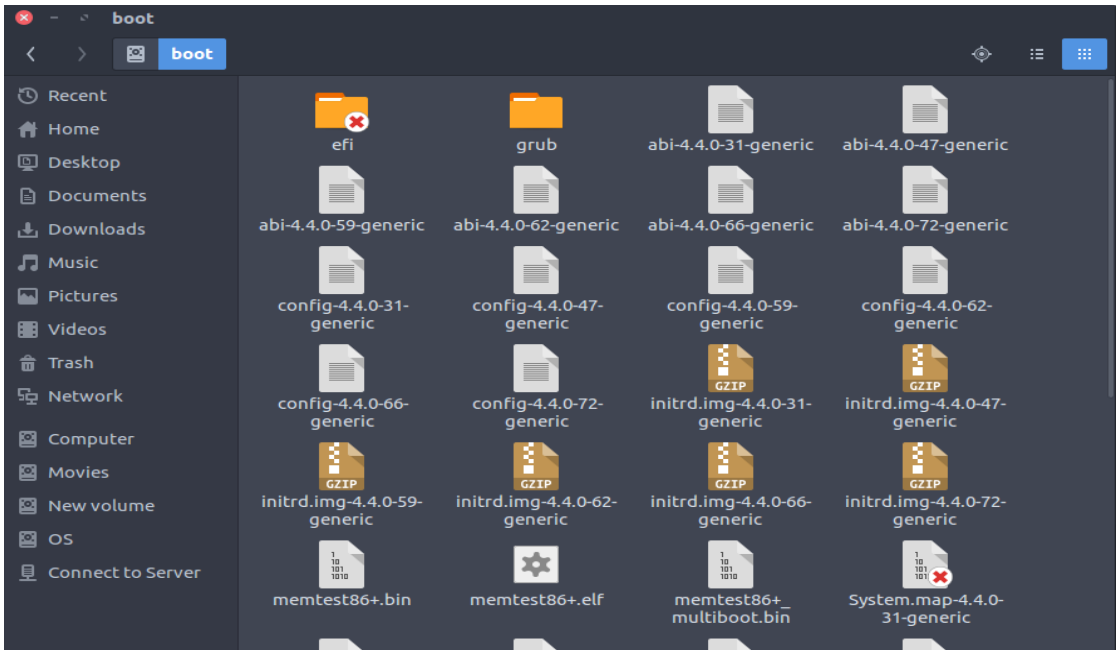


- Contains binary executables
- Common Linux commands you need to use in single-user modes are located under this directory.

- Commands used by all the users of the system are located here

  e.g. ps, ls, ping, grep, cp, cat, ls.

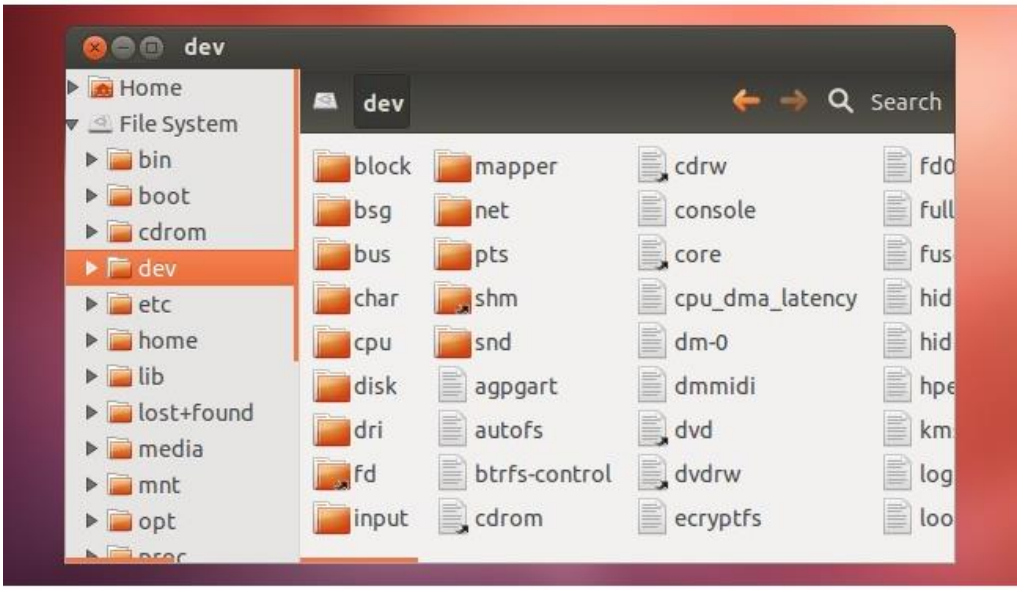
3. /boot – Static Boot Files



- Boot loader files
- Kernel initrd, vmlinux, grub files are located under /boot

  Example: initrd.img-2.6.32-24-generic, vmlinuz-2.6.32-24-generic

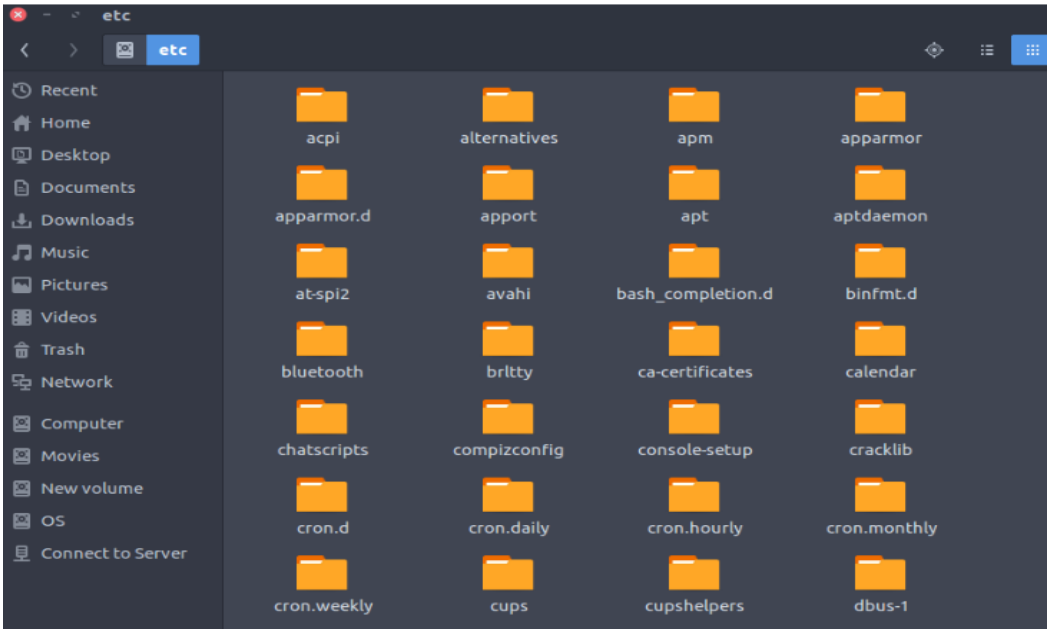
4. /dev-Device Files



- Contains device files
- These include terminal devices, usb, or any device attached to the system.

- These are not actual files as we know them, but they appear as files – for example, /dev/sda represents the first SATA drive in the system. If you wanted to partition it, you could start a partition editor and tell it to edit /dev/sda.
- This directory also contains pseudo-devices, which are virtual devices that don't actually correspond to hardware.

5. /etc- Configuration Files



- Contains configuration files required by all programs.
- This also contains start-up and shutdown shell scripts used to start/stop individual programs.

  Example: /etc/resolv.conf

b) File and device permissions

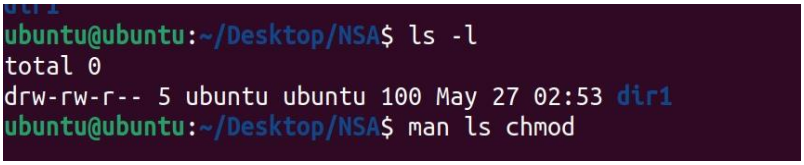
Since Ubuntu is a multi-user operating system, so it has security to prevent people from accessing each other's confidential files. When you do so, each file will be listed on a separate line in a long format.

$ls -l

1. There's a lot of information in those lines.
2. The first character = '-', which means it's a file

   'd', which means it's a directory.

3. The next nine characters = (rw-r–r–) show the security
4. The next column shows the owner of the file. (Here it is `root`)
5. The next column shows the group owner of the file.
6. The next column shows the size of the file in bytes.
7. The next column shows the date and time the file was last modified.
8. Last Column = File_name or Directory_name.



Security Permissions

rwx rwx rwx

user group other

Letters Definition

‘r’: “read” the file’s contents.

‘w’: “write”, or modify, the file’s contents.

‘x’ : “execute” the file. This permission is given only if the file is a program.

Operators Definition

`+`: Add permissions

`-`: Remove permissions

`=`: Set the permissions to the specified values

Reference Class Description

`u`(user): The user permissions apply only to the owner of the file or directory, they will not impact the actions of other users.

`g`(group): The group permissions apply only to the group that has been assigned to the file or directory, they will not affect the actions of other users.

`o`(others): The other permissions apply to all other users on the system, this is the permission group that you want to watch the most.

`a`(All three): All three (owner, groups, others)

Changing security permissions

The command you use to change the security permissions on files is called “chmod“, which stands for “change mode” because the nine security characters are collectively called the security “mode” of the file.

Adding execute permission

chmod o+x filename

Change multiple permissions at once

chmod ugo-rwx filename

Revoke execute(x) permission from others(o)

chmod ug+rw,o-x filename

Octal values

When Linux file permissions are represented by numbers, it's called numeric mode. In numeric mode, a three-digit value represents specific file permissions (for example, 744.) These are called octal values. The first digit is for owner permissions, the second digit is for group permissions, and the third is for other users. Each permission has a numeric value assigned to it:

r (read): 4

w (write): 2

x (execute): 1

In the permission value 744, the first digit corresponds to the user, the second digit to the group, and the third digit to others. By adding up the value of each user classification, you can find the file permissions.

For example, a file might have read, write, and execute permissions for its owner, and only read permission for all other users. That looks like this:

Owner: rwx = 4+2+1 = 7

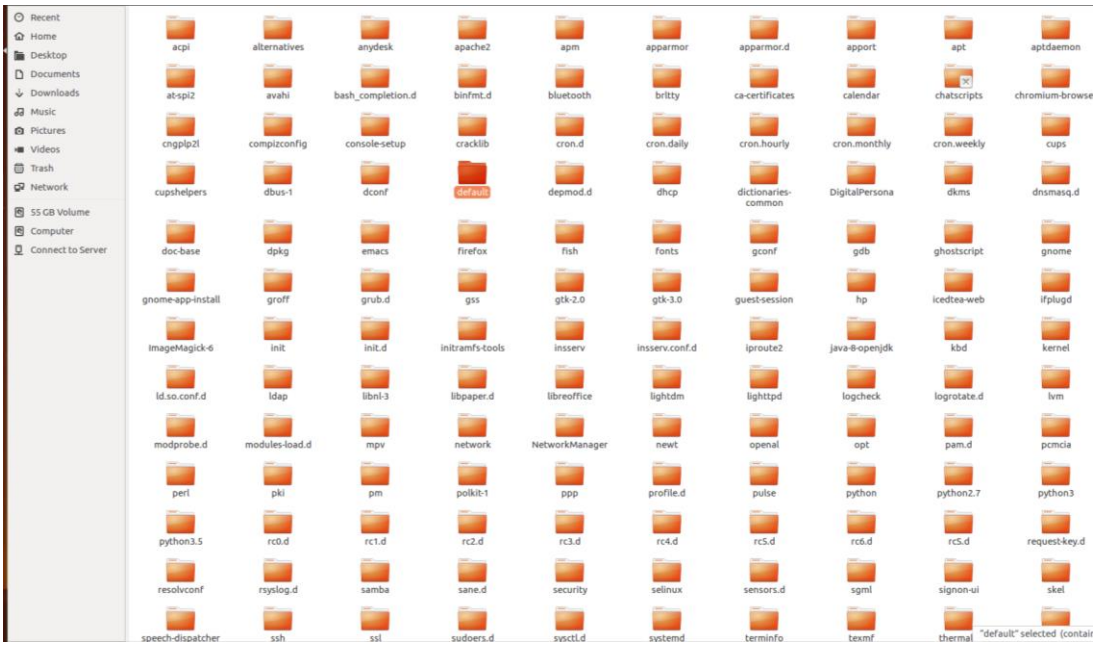
Group: r-- = 4+0+0 = 4

Others: r-- = 4+0+0 = 4

The results produce the three-digit value 744.

c) Study of system configuration files in /etc

Linux configuration files enable the Kernel to know about users, their login state, and manage file permissions and user groups. Most of the configuration files in Linux are usually under the /etc folder./etc(et-see) directory is where a linux configuration file live. The configuration files are static and cannot be executable. They are used to control the operation of various Linux programs.



Types of Configuration Files in /etc

1. Access files

2. Booting and login/logout

3. File system

4. System administration

5. Networking

6. System commands

7. Daemons

1. Access files

Access files tell the network domain how to look up host names.

Access files include:

- /etc/hosts.config
- /etc/hosts
- /etc/hosts.allow
- /etc/hosts.deny

2. Booting and login/logout

These files contain configuration information for booting up the system.

/etc/issue & /etc/issue.net

/etc/rc.d/rc

/etc/rc.d/rc.sysinit

/etc/rc.d/rc/rc X.d

3. File system

Linux provides /proc, a virtual file system that can be used to display various system data structures and parameters.

Many programs access this file system to gain statistical information about the system, such as the devices mounted on the system, the memory usage, etc.

/etc/mtab

/etc/fstab

/etc/mtools.conf

4. System administration

This group of files contains information about the users and user groups, as well as the file permissions and credentials of all users. These files include the following configuration files:

/etc/group

/etc/nologin

/etc/passwd

/etc/securetty

/etc/shadow

5. Networking

/etc/networks

Lists names and addresses of networks accessible from the network to which the machine is connected. Used by route command. Allows use of name for network.

/etc/protocols

Lists the currently available protocols

/etc/services

Translates network service names to port number/protocol.

6. System commands

System commands are meant to be used exclusively by the system. Programs like login or bash are all system commands. These are important files containing information about the system commands and include the following files.

/etc/lilo.conf

/etc/logrotate.conf

/etc/ld.so.conf

/etc/inittab

/etc/termcap

7. Daemons

Daemons are programs that run in the background without user interference.

These are often related to networking stacks, where they wait for connections to arrive so that they can provide services through them.

/etc/syslogd.conf

/etc/httpd.conf

/etc/conf.modules

d) Familiarizing log files for system events, user activity,network events.

Log files are a set of records that Linux maintains for the administrators to keep track of important events.

They contain messages about the server, including the kernel, services and applications running on it.

Linux provides a centralized repository of log files that can be located under the /var/log directory.

When issues arise, analyzing log files is the first thing an administrator needs to do.

How can Linux Log files help

1. Troubleshooting

When something goes wrong on a Linux system, logs can help pinpoint the issue.

By examining system logs, application logs, and service logs, it's possible to identify errors, warnings, and other messages that indicate what went wrong.

2. Diagnosing Performance Issues

System logs can help identify performance issues like memory leaks or disk I/O bottlenecks.

Examining application logs can also help identify performance issues with specific applications.

3. Monitoring System Health

Linux logs can be used to monitor system health and detect issues before they become critical.

By monitoring system logs, administrators can identify trends and patterns that could indicate a problem is brewing.

4. Compliance and Auditing

Many organizations are required to maintain logs for compliance and auditing purposes.

Linux logs can help organizations meet these requirements by providing a record of system activity.

5. Security

Linux logs are an essential tool for monitoring and detecting security issues.

System logs can be used to detect unauthorized access attempts, while application logs can help identify suspicious activity within specific applications.

By monitoring logs, administrators can quickly identify and respond to security incidents.

The log files generated in a Linux environment can typically be classified into four different categories:

- Application Logs
- Event Logs
- Service Logs

- System Logs

1. System Logs

These logs contain information about the system's operation, such as boot messages, kernel messages, and hardware events.

System logs are essential for troubleshooting system issues, and monitoring system performance.

2. Application Logs

These logs contain information about the behaviour of an application, including errors, warnings, and other messages.

Application logs are used to diagnose problems with applications and to analyze application performance.

3. Service Logs

These logs contain information about services running on the system, including network services and daemons.

Service logs are used to monitor service activity, and optimize service performance.

4.Event Logs

These logs contain information about events on the system, such as user logins, system shutdowns, and security events.

Event logs are used to audit system activity, track user activity, and investigate security incidents

Which Linux log files to monitor?

Monitoring and analyzing all of them can be a challenging task.

The sheer volume of logs can sometimes make it frustrating just to drill down and find the right file that contains the desired information.

1. /var/log/messages

2. /var/log/auth.log

3. /var/log/secure

4. /var/log/boot.log

5. /var/log/dmesg

6. /var/log/kern.log

7. /var/log/faillog

8. /var/log/cron

9. /var/log/yum.log

10. /var/log/maillog or /var/log/mail.log

11. var/log/httpd/

12. /var/log/mysqld.log or /var/log/mysql.log

Result:

Familiarisation with file system hierarchy,permission, configuration files and log files has been done successfully.

LAB CYCLE:4

EXPERIMENT NO: 6

Date:

SHELL SCRIPTING

Aim: a) Shell scripting: study bash syntax, environment variables, variables, control constructs such as if, for and while, aliases and functions, accessing command line arguments passed to shell scripts.

b) Study of startup scripts, login and logout scripts, familiarity with systemd and system 5 init scripts is expected

a) Shell scripting: study bash syntax, environment variables, variables, control constructs such as if, for and while, aliases and functions, accessing command line arguments passed to shell scripts.

Shell scripting refers to the process of writing and executing scripts using a shell, which is a command-line interface or command interpreter on Unix-like operating systems. The most commonly used shell scripting languages are Bash (Bourne Again Shell) and sh (Bourne Shell). Shell scripts can automate tasks, execute commands, and perform various operations on a Unix or Linux system.

Here are some key concepts and features of shell scripting:

1.Script file extension: Shell scripts typically have file extensions like ".sh" to indicate that they are shell scripts.

2.Shebang line: The first line of a shell script starts with a shebang (#!) followed by the path to the shell interpreter. For example, #!/bin/bash specifies that the script should be interpreted using Bash.

3.Variables: Shell scripts can define and use variables. Variables are typically assigned values using the assignment operator (=).

For example :

```
name="John"
age=25
```

4.Command execution: Shell scripts can execute commands using various command-line utilities. Commands can be executed directly or stored in variables for later use. For example:

```
ls
result=$(ls)
```

5.Control structures: Shell scripting supports control structures like conditionals (if-else statements) and loops (for loops, while loops). These structures allow you to control the flow of execution based on conditions or iterate over a set of values.

6.Input/output redirection: Shell scripts can redirect input and output streams. For example, you can redirect the output of a command to a file using the ">" operator or read input from a file using the "<" operator.

7.Functions: Shell scripts can define functions to encapsulate reusable code blocks. Functions help in modularizing the script and improving code organization.

8.Command-line arguments: Shell scripts can accept command-line arguments. These arguments can be accessed using special variables like $1, $2, etc., where $1 represents the first argument, $2 represents the second argument, and so on.

9.Environment variables: Shell scripts can access and modify environment variables, which are system-wide variables that hold information about the environment in which the script is running.

10.Error handling: Shell scripts can handle errors using conditional statements and error codes returned by commands. This allows scripts to take appropriate actions based on the success or failure of commands.

Shell scripting is a powerful tool for automation, system administration, and repetitive tasks in Unix-like operating systems. It provides a flexible and efficient way to execute commands, manipulate data, and control the behavior of the shell.

Environment Variables

Environment variables are an essential part of shell scripting as they allow you to store and retrieve information that can be used by various processes and scripts running in the shell. Here are some key points about environment variables in shell scripting:

1. Defining Environment Variables: Environment variables are typically defined using uppercase letters. To set an environment variable, you can use the export command followed by the variable name and its value. For example:

```
export MY_VAR="Hello, World!"
```

2. Accessing Environment Variables: To access the value of an environment variable, you can use the $ symbol followed by the variable name. For example:

```
echo $MY_VAR
```

3. System-defined Environment Variables: The shell automatically sets some environment variables that contain useful information. Examples include:

$HOME: The current user's home directory.

$PATH: A list of directories where the shell looks for executable files.

$USER: The username of the current user.

$PWD: The present working directory.

4. Shell Script-defined Environment Variables: You can define your own environment variables within shell scripts. These variables are local to the script by default, but you can export them to make them available to other processes. For example:

```
# Define a local variable
MY_LOCAL_VAR="Local variable"
```

```
# Export the variable to make it available to other processes
export MY_LOCAL_VAR
```

5. Using Environment Variables in Scripts: Environment variables can be used within shell scripts just like any other variables. Here's an example:

```
#!/bin/bash
echo "The value of MY_VAR is: $MY_VAR"
```

6. Unsetting Environment Variables: To remove an environment variable, you can use the unset command followed by the variable name. For example:

```
unset MY_VAR
```

7. Persistent Environment Variables: Environment variables defined in a shell session are not persistent and will be lost when the session ends. To make them persistent, you can define them in shell startup files such as .bashrc or .bash_profile (for Bash) or in the system-wide /etc/environment file.

Variables

Variables in shell scripting are used to store and manipulate data within a script. Here are some important points about variables in shell scripting:

1. Variable Declaration: In shell scripting, variables are typically declared without specifying a data type. You can assign a value to a variable using the syntax variable_name=value. For example:

```
name="John"
age=25
```

2. Variable Naming: Variable names in shell scripting can contain letters (a-z, A-Z), digits (0-9), and underscores (_). They must start with a letter or an underscore. Variable names are case-sensitive, so my_var and MY_VAR are considered different variables.

3. Accessing Variable Values: To access the value of a variable, you can use the $ symbol followed by the variable name. For example:

```
echo $name
```

4. Variable Scope: By default, variables in shell scripts have a global scope and can be accessed from anywhere within the script. If you want to limit the scope of a variable to a specific block of code, you can use functions.

5. Special Variables: Shell scripting provides some predefined variables that have special meanings. Examples include:

$0: The name of the script itself.

$1, $2, etc.: Command-line arguments passed to the script.

$#: The number of command-line arguments.

$?: The exit status of the last command.

$$: The process ID of the current script.

6. Variable Manipulation: You can perform various operations on variables, such as concatenation, substring extraction, and arithmetic calculations. Some commonly used operators and syntax include:

Concatenation: result=$var1$var2

Substring extraction: substring=${string:position:length}

Arithmetic calculations: result=$((num1 + num2))

7. Quoting Variables: When accessing variable values, it's a good practice to enclose them in double quotes ("") to handle cases where the value contains spaces or special characters. For example:

echo "$name is $age years old."

8. Variable Assignment from Command Output: You can assign the output of a command to a variable using command substitution. The syntax is variable=$(command). For example:

date=$(date +%Y-%m-%d)

These are some fundamental aspects of variables in shell scripting. They allow you to store and manipulate data, perform calculations, and pass information to and from your script.

Control Structures

In shell scripting, control structures are used to control the flow of execution based on certain conditions or to repeat a set of instructions. The commonly used control structures in shell scripting include:

1. if-else: The if-else statement allows you to perform different actions based on a condition. It has the following syntax:

```
if condition
then
   # statements to execute if condition is true
else
   # statements to execute if condition is false
fi
```

2. for loop: The for loop allows you to iterate over a list of values or elements. It has the following syntax:

```
for variable in list
do
   # statements to execute for each iteration
done
```

The variable takes the value of each item in the list on each iteration.

3. while loop: The while loop allows you to repeat a set of statements as long as a condition is true. It has the following syntax:

```
while condition
do
    # statements to execute while condition is true
done
```

The loop continues until the condition evaluates to false.

4. until loop: The until loop is similar to the while loop, but it continues until a condition becomes true. It has the following syntax:

```
until condition
do
    # statements to execute until condition becomes true
done
```

The loop continues until the condition evaluates to true.

5. case statement: The case statement allows you to perform different actions based on the value of a variable. It has the following syntax:

```
case variable in
    pattern1)
        # statements to execute for pattern1
        ;;
    pattern2)
        # statements to execute for pattern2
        ;;
    pattern3)
        # statements to execute for pattern3
        ;;
    *)
        # statements to execute for all other patterns
        ;;
esac
```

The value of variable is compared against each pattern, and the corresponding statements are executed for the matching pattern.

These control structures provide you with the flexibility to conditionally execute code, iterate over a set of values, and make decisions based on different conditions in shell scripting.

Aliasing

In Linux, an alias is a shortcut that references a command. An alias replaces a string that invokes a command in the Linux shell with another user-defined string. Aliases are mostly used to replace long commands, improving efficiency and avoiding potential spelling errors.

The alias command provides a string value that replaces a command name when it is encountered. The alias command lets you create shortcuts for long commands, making them easier to remember and use. It will have the same functionality as if the whole command is run.

How to Create Your Own Linux Commands?

Using the alias command, you'll be able to create your own commands. It's so simple to create your own command.

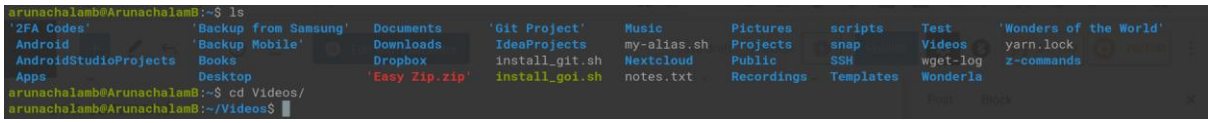
Here's the syntax for the alias command:

alias [alias-name[=string]...]

Let's look at an example of creating your own command.

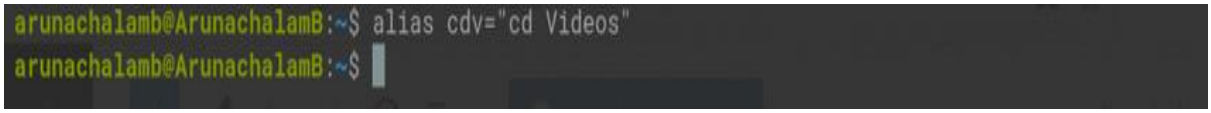
Let's assume you want to create a command called cdv, and entering the command in the terminal should take you to the Videos directory.

Usually, to navigate to a directory, we use cd command. To navigate to Videos we need to use cd Videos as shown in the below screenshot:



Let's create our command called cdv to navigate to the Videos directory. To achieve that, you have to enter the following command in your terminal:

alias cdv="cd Videos“



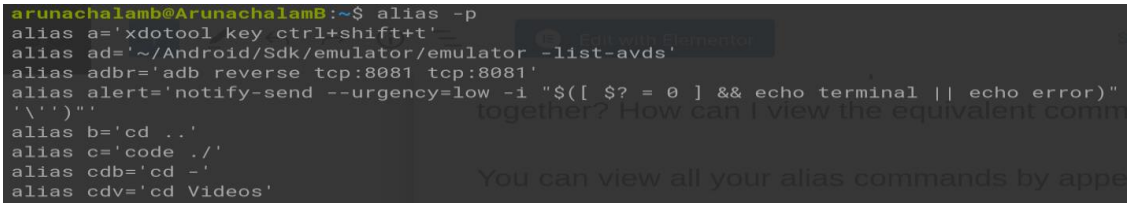
We have created our command. From the above screenshot, you can see that it does not return anything.

Run the cdv command on your terminal to see what happens:

How to View Created Alias Commands

You can view all your alias commands by appending the -p flag to the alias command like this:

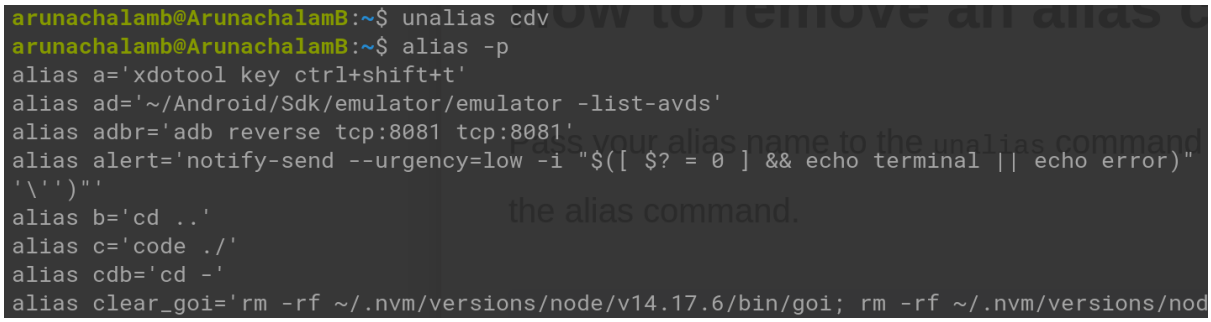
alias -p



How to Remove an Alias Command in Linux

Pass your alias name to the unalias command as an argument to remove the alias command.

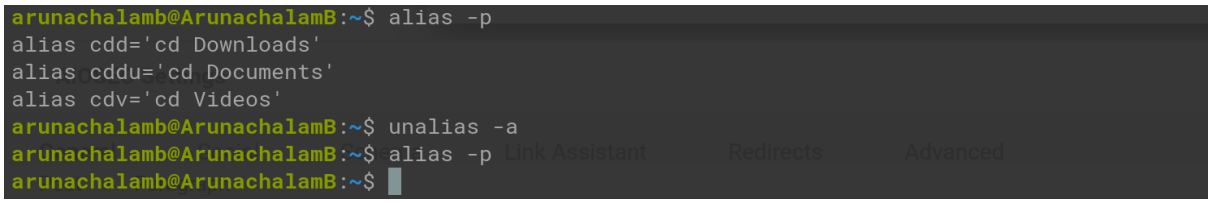
unalias alias_name



How to Remove All Alias Commands in Linux

We have a command to achieve that:

unalias –a



If you create an alias command, it'll be active only for the particular instance of the terminal. It'll not be created permanently, so you won't be able to access it in two different terminal windows unless you run the alias command on both terminals.

b) Study of startup scripts, login and logout scripts, familiarity with systemd and system 5 init scripts is expected

Startup scripts, login scripts, and logout scripts are important components of the initialization process in a Unix-like operating system. They allow administrators to automate tasks and configure system behavior during system startup, user login, and user logout. Familiarity with systemd and System V init scripts is crucial for managing these processes efficiently. Let's explore each of these topics in more detail:

1. Startup Scripts:

Startup scripts are executed during the boot process to initialize the system. In modern Linux distributions that use systemd as the init system, startup scripts are typically managed through systemd unit files. These unit files define services, targets, and other units that control the behavior of the system. The main directory for systemd unit files is /etc/systemd/system/.

2. Login Scripts:

Login scripts are executed when a user logs into a system. These scripts provide a way to set environment variables, configure user-specific settings, and perform additional tasks upon login. The specific login script executed depends on the user's shell and the system configuration. For example, in the Bash shell, the login script is usually ~/.bash_profile or ~/.bash_login for login shells, and ~/.bashrc for non-login shells.

3. Logout Scripts:

Logout scripts are executed when a user logs out of a system. These scripts allow for cleanup tasks, such as removing temporary files or logging session statistics. The specific logout script executed depends on the user's shell and the system configuration. In Bash, the logout script is typically ~/.bash_logout.

4. Systemd:

Systemd is a modern init system and service manager that has replaced the traditional System V init system in many Linux distributions. Systemd provides enhanced features for managing services, including parallel startup, dependency-based service control, and centralized logging. Systemd unit files, such as service unit files (.service), target unit files (.target), and timer unit files (.timer), are used to define and control services.

5. System V Init Scripts:

System V init scripts were the traditional initialization mechanism used in Unix-like systems before the introduction of systemd. Init scripts are stored in /etc/init.d/ directory and are typically written in shell scripting languages (e.g., Bash). These scripts are responsible for

starting, stopping, and managing system services and can be run manually or automatically during the system startup.

Familiarity with both systemd and System V init scripts is important for understanding and managing the initialization process in different Linux distributions. The choice of which init system to use depends on the specific distribution and its configuration. It's beneficial to have knowledge of both to be able to work with various systems effectively.

Shell script program questions.

1. Write a script to show current date,time and current directory.

```
#!/bin/bash
echo "Current date is `date`"
echo "Current directory is `pwd`"
```

Output:

```
exam@cec-H110M-S2:~$ cd mca_nsa
exam@cec-H110M-S2:~/mca_nsa$ ./dtdir.sh
Current date is Wed Apr  5 11:52:41 IST 2023
Current directory is /home/exam/mca_nsa mj
exam@cec-H110M-S2:~/mca_nsa$
```

2.Write shell script to find reverse of a number.

```
#!/bin/bash
echo enter the no:
read n
num=0
while [ $n -gt 0 ]
do
num=$(expr $num \* 10)
k=$(expr $n % 10)
num=$(expr $num + $k)
n=$(expr $n / 10)
done
echo Reversed number is $num
```

Output:

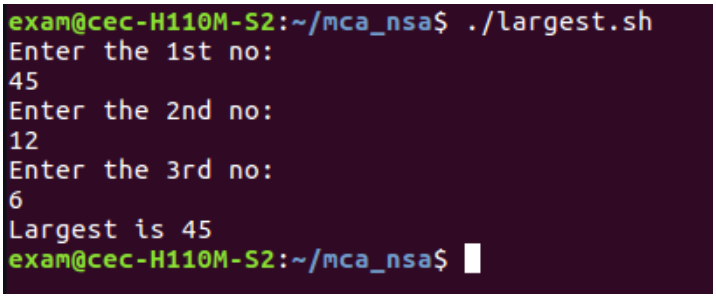
```
exam@cec-H110M-S2:~/mca_nsa$ ./reverse.sh
enter the no:
458
Reversed number is 854
exam@cec-H110M-S2:~/mca_nsa$
```

3. Write a script to find largest among three numbers.

```
#!/bin/bash
echo Enter the 1st no:
```

```
read m
echo Enter the 2nd no:
read n
echo Enter the 3rd no:
read o
if [ $m -gt $n ] && [ $m -gt $o ]
then
echo Largest is $m
elif [ $n -gt $m ] && [ $n -gt $o ]
then
echo Largest is $n
else
echo Largest is $o
fi
```

Output:

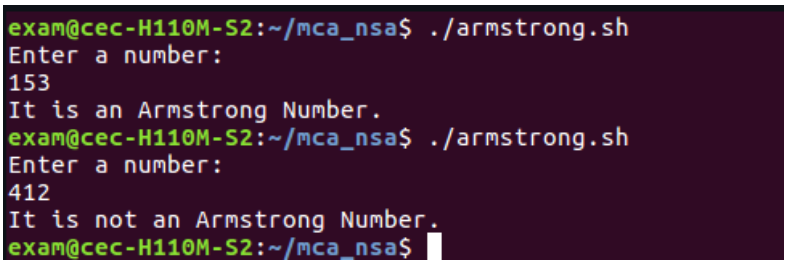


4. Write a script to check whether a number is armstrong or not.

```
#!/bin/bash
echo "Enter a number: "
read c
x=$c
sum=0
r=0
n=0
while [ $x -gt 0 ]
do
r=`expr $x % 10`
```

```
n=`expr $r \* $r \* $r`
sum=`expr $sum + $n`
x=`expr $x / 10`
done
if [ $sum -eq $c ]
then
echo "It is an Armstrong Number."
else
echo "It is not an Armstrong Number."
fi
```

Output:



5. Write a script to check password and login.

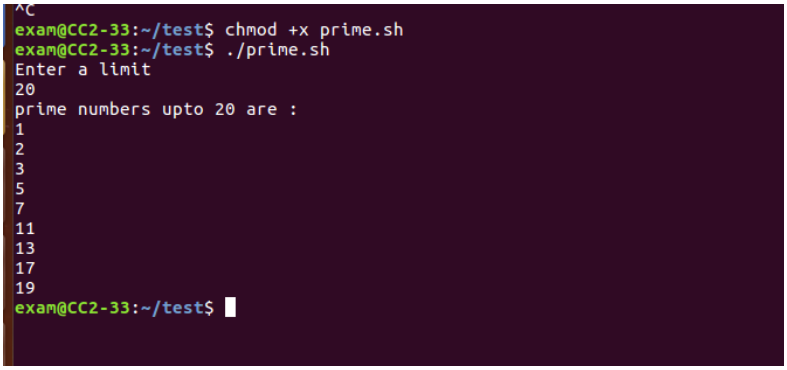
```
# !/bin/bash
read -p 'Username: ' user
read -sp 'Password: ' pass
if (( $user == "Admin" && $pass == "admin123" ))
then
        echo -e "\nWelcome! You are Sucessfull login\n"
else
        echo -e "\nUnsuccessful login\n"
fi
```

Output:

6. Write a script to count the prime numbers in specific range

```
#!/bin/bash
echo "Enter a  lower limit"
read i
echo "Enter a  upper limit"
read limit
echo "prime numbers upto $limit are :"
while [ $i -le $limit ]
do
   flag=1
   j=2
   while [ $j -lt $i ]
   do
       rem=$(( $i % $j ))
      if [ $rem -eq 0 ]
      then
        flag=0
        break
      fi
   j=$(( $j+1 ))
   done
   if [ $flag -eq 1 ]
   then
     echo "$i"
   fi
i=$(( $i+1 ))
done
```

Output:

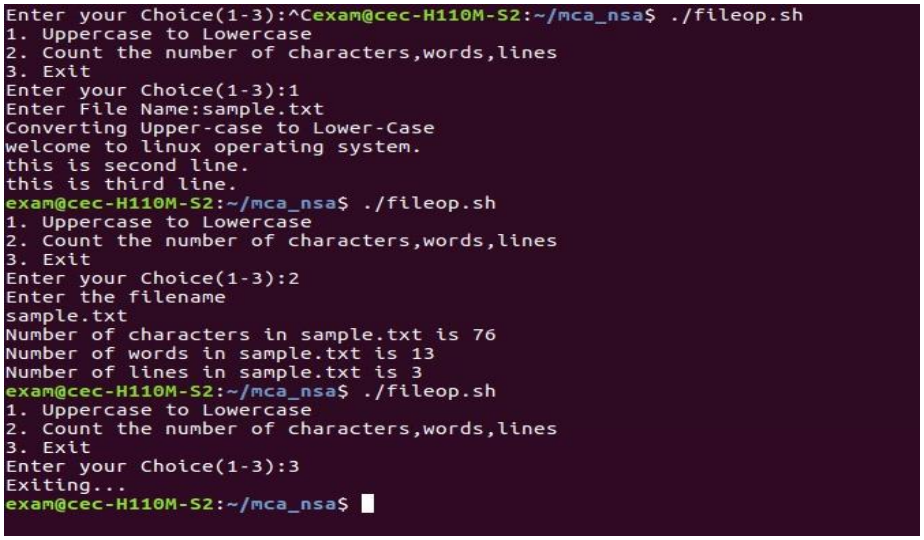


7. Write a script to convert the contents of agiven file from uppercase to lowercase and also count the number of lines,words and characters of the resultant file. Also display the resultant file in descending order.

```
#!/bin/bash
getFile()
{
   # Reading txtFileName to convert it's content
   echo -n "Enter File Name:"
   read filename
   # Checking if file exist
   if [ ! -f $filename ]; then
      echo "File Name $filename does not exists."
      exit 1
   fi
}
 echo "1. Uppercase to Lowercase "
 echo "2. Count the number of characters,words,lines"
 echo "3. Exit "
 echo -n "Enter your Choice(1-3):"
 read ch

 case "$ch" in
   1)
```

```
    # Function Call to get File
    getFile
    # Converting to lower case if user choose 1
    echo "Converting Upper-case to Lower-Case "
    tr '[A-Z]' '[a-z]' <$filename
  ;;

 2)
  echo Enter the filename
  read file
  c=`cat $file | wc -c`
  w=`cat $file | wc -w`
  l=`grep -c "." $file`
  echo Number of characters in $file is $c
  echo Number of words in $file is $w
  echo Number of lines in $file is $l
  ;;

  *) # exiting for all other cases
     echo "Exiting..."
     exit
  ;;
esac
```

Output:



8. Write a script to perform following basic math operation as:

Addition, subtraction, multiplication, division

```
#!/bin/sh
echo "Enter the two numbers to perform arithmetic operations "
read a b

val=`expr $a + $b`
echo "a + b : $val"

val=`expr $a - $b`
echo "a - b : $val"

val=`expr $a \* $b`
echo "a * b : $val"

if [ $a -gt $b ]
then
        val=`expr $a / $b`
        echo "a / b : $val"
else
```

```
        val=`expr $b / $a`
        echo "b / a : $val"

fi
```

Output:

```
exam@cec-H110M-S2:~/mca_nsa$ ./arithmetic.sh
Enter the two numbers to perform arithmetic operations
20 10
a + b : 30
a - b : 10
a * b : 200
a / b : 2
exam@cec-H110M-S2:~/mca_nsa$
```

9. Read 3 marks of a student and find the average. Display the grade of the student based on the average. (if..then..elif..fi)

S >= 90%

A < 90%, but >= 80%

B < 80%, but >= 60%

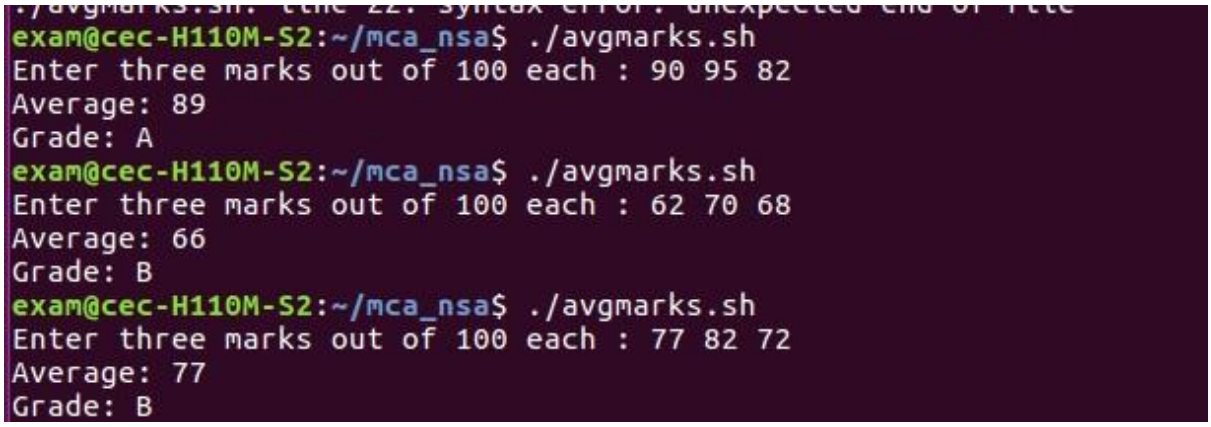
P < 80%, but >= 40%

F < 40%

```
#! /bin/bash
read -p "Enter three marks out of 100 each : " m1 m2 m3
s=$(($m1+$m2+$m3))
avg=$(($s / 3|bc))
echo -e "Average: $avg"

if [ $avg -ge 90 ]
then
        echo "Grade: S"
elif [[ $avg -lt 90  &&  $avg -ge 80  ]]
then
        echo "Grade: A"
elif [[ $avg -lt 80  &&  $avg -ge 60  ]]
```

```
then
        echo "Grade: B"
elif [[ $avg -lt 80  &&  $avg -ge 40  ]]
then
        echo "Grade: P"
else
        echo "Grade: F"
fi
```

Output:



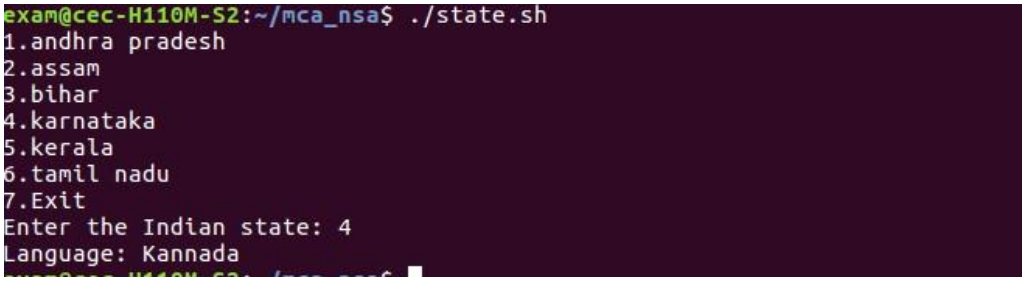
10. Read the name of an Indian state and display the main language according to the table. For other states, the output may be “Unknown”. Use “|” to separate states with same language *(case..esac)*

State	Main Language
Andhra Pradesh	Telugu
Assam	Assamese
Bihar	Hindi
Himachal Pradesh	Hindi
Karnataka	Kannada
Kerala	Malayalam
Lakshadweep	Malayalam
Tamil Nadu	Tamil

```
#!/bin/bash
echo -e "1.andhra pradesh \n2.assam \n3.bihar \n4.karnataka \n5.kerala \n6.tamil nadu \n7.Exit"
read -p "Enter the Indian state: " state
#state=$(echo $state | tr '[:upper:]' '[:lower:]')
```

```
case $state in
        1)
                echo "Language: Telugu"
                ;;
        2)
                echo "Language: Assamese"
                ;;
        3)
                echo "Language: Hindi"
                ;;
        4)
                echo "Language: Kannada"
                ;;
        5)
                echo "Language: Malayalam"
                ;;
        6)
                echo "Language: Tamil"
                ;;
        *)
                echo "Language: Unknown";;
esac
```

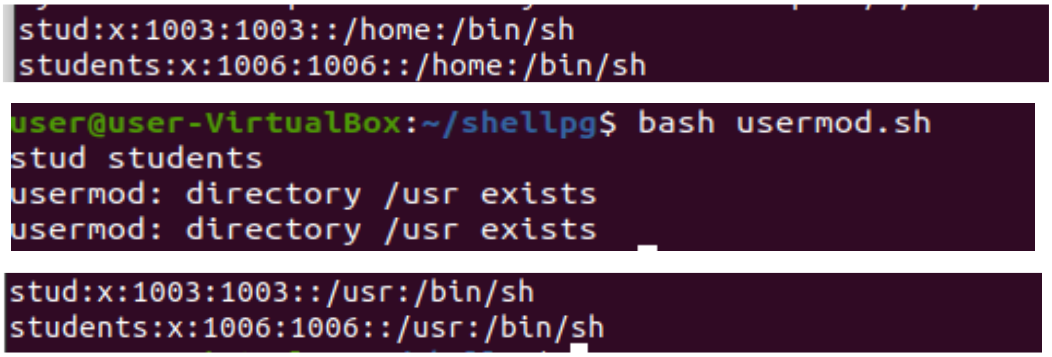
Output:



11.Change the home folder of all users whose name start with stud from /home/username to /usr/username. Also change the password of username to username123 (e.g., /home/stud25 changes to /usr/stud25 and his/her password changes to stud25123) - (Use for .. in)

```
#! /bin/bash
result=$(grep stud* /etc/passwd)
result=$(echo "${result}"| cut -d: -f 1)
echo $result
for f in $result
do
        p="${f}123"
        sudo usermod -p $(echo $p | openssl passwd -1 -stdin) $f
        sudo usermod -m -d /usr $f
done
```

Output:

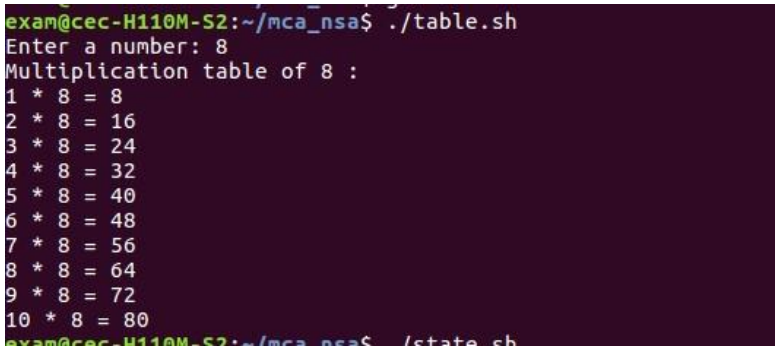


12. Read a number and display the multiplication table of the number up to 10 lines. - (Use for((..)))

```
#! /bin/bash
read -p "Enter a number: " num
echo "Multiplication table of $num : "
for (( i=1; i<=10; i++))
do
        val=$(( num * i ))
```

```
    echo "$i * $num = $val"
done
```

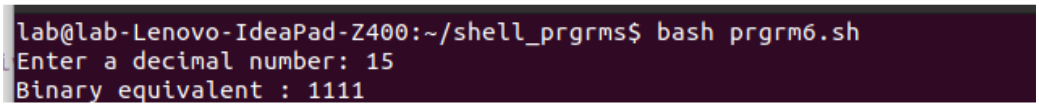
Output:



13. Read a Decimal number. Convert it to Binary and display the result. - (Use while)

```
#! /bin/bash
read -p "Enter a decimal number: " n
val=0
power=1
while [ $n -ne 0 ]
do
    r=`expr $n % 2`
    val=`expr $r \* $power + $val`
    power=`expr $power \* 10`
    n=`expr $n \/ 2`
done
echo "Binary equivalent : $val"
```

Output:



Result:

Shell script program has done successfully and output is verified.