Module: 1 - Linux server - Understand and use essential tools

1. **What is the minimum number of partitions you need to install Linux?**

➔ To install Linux, the minimum number of partitions you need is **two**:

1. **Root partition ("/")**: This is where the core files of the operating system will be installed.
2. **Swap partition**: This acts like virtual memory. It's used when our physical RAM gets full.

These two are the minimum for a basic Linux installation, but there are more options we can use for organization and performance (like separate partitions for home or boot), but these two are the essential ones.

2. . Explain About Chmod Command

➔ The `chmod` command in Linux is used to **change the permissions** of files or directories. Permissions determine who can read, write, or execute a file. We can control these permissions for three types of users:

1. **Owner (User)**: The person who owns the file.
2. **Group**: Users who are in the same group as the file.
3. **Others**: Everyone else.

Permissions are represented by three letters:

- **r** (read): Allows viewing the content of the file.
- **w** (write): Allows modifying the file.
- **x** (execute): Allows running the file if it's a program.

How `chmod` works:

- We can change the permissions by using either **letters** or **numbers**.

Using letters:

- `chmod u+x file.txt` → Adds **execute** permission for the **owner** (u stands for user/owner).
- `chmod g-w file.txt` → Removes **write** permission for the **group** (g stands for group).
- `chmod o+r file.txt` → Adds **read** permission for **others** (o stands for others).

Using numbers:

Permissions can also be set using numbers, where:

- **r = 4**, **w = 2**, **x = 1**.
- The numbers are added together to represent different combinations of permissions.

For example:

- `chmod 755 file.txt` means:
    - **Owner** gets to read, write, and execute (7 = 4+2+1).
    - **Group** and **Others** get read and execute permissions (5 = 4+1).

In simple terms, `chmod` is used to decide who can do what with a file or folder!

3. How to check Linux memory utilization

➔ To check memory utilization in Linux, we can use a few simple commands. Here are the most common ones:

1. `free` Command

This is the easiest way to check memory usage.

➔ Just type:
```
free -h
```
- The `-h` flag makes the output easier to read by displaying memory in human-readable units (like MB or GB).
  we'll see:
    - **Total**: The total memory available.
    - **Used**: The memory that's currently in use.
    - **Free**: The unused memory.
    - **Buffers/Cache**: Memory used for caching and buffers, which can be freed if needed.

2. `top` Command

This command gives us a dynamic, real-time view of your system's memory usage, along with CPU and processes.

➔ Just type:
css
Copy
```
top
```
- In the top part of the screen, we'll see memory-related info like:
    - **Mem**: Total memory, used memory, free memory, and more.
    - **Swap**: Information about swap space (virtual memory).

3. `htop` Command

`htop` is a more user-friendly, colored version of `top`. It shows memory usage in a graphical way.

➔ Type:
```
htop
```

- If it's not installed, we can install it with `sudo apt install htop` (on Ubuntu) or `sudo yum install htop` (on CentOS).

We can use these commands to quickly understand how much memory is being used and whether we need to worry about running out of memory!.

4. · Use grep to search for specific patterns in files.

➔ The `grep` command in Linux is used to **search for specific patterns** (like words or phrases) in files. It's really useful when we need to find something quickly in a large file or across multiple files.

Here's how it works in simple terms:

Basic Syntax:

```
grep [pattern] [file]
```

- **[pattern]**: This is the word or phrase you are looking for.
- **[file]**: The file (or files) where we want to search for the pattern.

Examples:

Search for a word in a file: If we want to find the word "apple" in a file called `fruits.txt`, we would use:

```
grep "apple" fruits.txt
```

1. This will show us all the lines in `fruits.txt` that contain the word "apple".

2. Search for a word in multiple files: To search for "apple" in all `.txt` files in the current directory, use:

```
grep "apple" *.txt
```

This will search through all `.txt` files and show any lines that contain the word "apple".

3.**Case-insensitive search**: If we don't care about whether the word is capitalized, use the `-i` option to make the search case-insensitive:

```
grep -i "apple" fruits.txt
```

This will find "apple", "Apple", "APPLE", and so on.

4.**Show line numbers with results**: To see the line numbers where the word appears, add the `-n` option:

```
grep -n "apple" fruits.txt
```

This will show the line number along with the matching line.

5.**Count the number of matches**: If we just want to know how many times the word appears in the file, use the `-c` option:

```
grep -c "apple" fruits.txt
```

So in simple terms, grep is a fast and powerful tool to search for specific words or patterns in files. we can even combine it with other commands to find exactly what we're looking for!

5.Get Connecting on a linux server by ssh

➔ To connect to a Linux server using SSH (Secure Shell), follow these simple steps:

Prerequisites:

1. **SSH Client**: Most Linux and macOS computers have an SSH client pre-installed. Windows users can use tools like **PuTTY** or the Windows Subsystem for Linux (WSL).

2. **Server IP Address**: we need the IP address or domain name of the Linux server we want to connect to.

3. **Username**: The username (usually something like root or user1) on the Linux server.

4. **Password or SSH Key**: we’ll need the password for the username or an SSH key if the server is set up for key-based authentication.

Steps:

1. **Open your Terminal**:

    - **Linux/macOS**: Open the Terminal app.

    - **Windows**: Open **Command Prompt**, **PowerShell**, or **Windows Terminal** (if we're using WSL).

2.Use the SSH command: The basic format of the SSH command is:

```
ssh username@server_ip
```

    - username: our login name on the server (e.g., root or user1).
    - server_ip: The IP address of the server (e.g., 192.168.1.10).

Example:

```
ssh root@192.168.1.10
```

3.Enter your password: If it's our first time connecting to the server, we might be asked to verify the server's authenticity (we can confirm and continue by typing "yes").

Then, enter the password for the specified username. If we're using SSH keys instead of a password, the connection will happen automatically if the key is correctly set up.

4.You're connected!: Once the password is entered (or key authentication is successful), we will be logged into the server's terminal, and we can begin running commands on the server.

Troubleshooting:

- **Connection timeout or refusal**: Make sure the server is running and SSH is enabled. we can check if the server allows SSH connections by making sure that the firewall or any security groups (in cloud services) are not blocking SSH traffic on port 22.

- **Permissions**: If we don’t have permission, contact the system administrator for access.

6. Create 5 files in the /tmp directory, and then use tar and gzip to bundle and compress the files.

➔ 1. Create 5 Files in the /tmp Directory

First, let's create some simple text files in the /tmp directory. we can do this using the touch command:

➔ cd /tmp
➔ touch file1.txt file2.txt file3.txt file4.txt file5.txt

This will create 5 empty files (file1.txt, file2.txt, file3.txt, file4.txt, and file5.txt) in the /tmp directory.

2. Bundle and Compress the Files with tar and gzip

2.1. Bundle the Files Using tar

The tar command is used to combine multiple files into one archive. Here’s how we can use it:

➔ tar -cvf files.tar file1.txt file2.txt file3.txt file4.txt file5.txt

- tar: The command to create an archive.
- -c: Create a new archive.
- -v: Verbose mode (shows which files are being added).
- -f: Followed by the name of the archive (files.tar in this case).

This will bundle all 5 files into a single file named files.tar.

2.2. Compress the Archive Using gzip

Now, let's compress the archive using gzip:

```
gzip files.tar
```

- gzip: The command to compress the archive.
- files.tar: The archive file we just created.

This will compress the files.tar file and create a new file called files.tar.gz. The original files.tar will be replaced by the compressed version.

3. Check the Result

After running the above commands, we should see a file named files.tar.gz in the /tmp directory. This is our bundled and compressed archive.

To check the contents of the compressed file, we can run:

➔ tar -tzf files.tar.gz

This will list all the files inside the files.tar.gz archive without extracting them.

7.Describe the root account

➔ The **root account** on a Linux system is like the "administrator" or "superuser" account. It has **unlimited control** over the entire system, meaning it can do anything and everything—whether it's installing or removing software, changing system settings, or accessing any file.

Here are some key things about the root account in simple language:

1. Full Control

- The root user has permission to do anything on the system. This includes modifying or deleting important system files, changing user permissions, and more.

- For example, the root can install software, shut down the system, or change network settings.

2. Root's Password

- The root account is protected by a **password**, just like other user accounts.
- You can only use the root account if we know its password (or use the sudo command to temporarily gain root privileges).

3. Dangerous Powers

- Because the root account can make major changes to the system, it can also cause serious damage if used incorrectly.
- For example, accidentally deleting system files or running certain commands can break the system or make it unusable.

4. Common Usage

- Normally, regular users don't log in as root to avoid mistakes. Instead, they use **sudo** (Super User Do) to temporarily run commands with root privileges when needed.
- Example: If we want to install a program, we might type sudo apt install <program-name> on a Debian-based system. This gives we root-level access just for that specific command.

5. Root in Action

we can switch to the root account with the su (substitute user) command:

```
su
```

- It will prompt us for the root password. Once entered, we'll have full root access.

- On many modern systems, the root account is disabled by default for security reasons, and users are encouraged to use sudo for administrative tasks instead.

6. Root User vs. Regular User

- Regular users have limited access. They can create files and run programs but can't change system-wide settings.
- Root has **no restrictions** and can do everything.

8. What is shell?

A **shell** is like a command interpreter or a "messenger" between us and our computer's operating system. It allows us to interact with our computer by typing commands. Think of it as a **text-based interface** where we can tell the computer what to do, and it will execute those tasks for us.

Key Points About the Shell:

1. **Text-Based Interface**:
   - Unlike clicking on icons in a graphical user interface (GUI), in a shell, we type out commands and the computer does the work.
   - For example, instead of opening a file by double-clicking it, we might type something like cat file.txt to see its contents.
2. **Command Interpreter**:
   - The shell "interprets" the commands we type and then tells the operating system what to do. If we ask it to run a program, it will find that program and run it.
3. **Types of Shells**: There are different types of shells, but the most common ones are:
   - **Bash (Bourne Again Shell)**: This is the most popular shell in Linux. It's used by many Linux distributions by

default.

  - **Zsh (Z Shell)**: Another popular shell, often praised for its features and customization options.

  - **Sh (Bourne Shell)**: One of the original shells used in UNIX systems.

  - **Fish (Friendly Interactive Shell)**: A more user-friendly shell with features like auto-suggestions and syntax highlighting.

4. **How You Use It**:

➔ we interact with the shell by typing commands in a terminal. The terminal is the window where you see the shell, and it allows us to enter commands. It might look something like this:

```
$ ls
```

➔
```
file1.txt  file2.txt  file3.txt
```

  - In this example, ls is the command to list files in a directory, and the shell displays the files it finds.

5. **What You Can Do With a Shell**:

  - **Run Programs**: we can start programs, like running firefox to open the Firefox browser.

  - **Manage Files**: we can copy, move, delete, or rename files using commands like cp, mv, and rm.

  - **Automate Tasks**: we can write scripts (small programs) to automate tasks we do often. For example, a script might back up our files every night.

Example:

Here’s an example of interacting with the shell:

➔ we open the terminal and type a command like this:

- echo "Hello, World!"

➔ The shell will display:

- Hello, World!

Why It's Useful:

- **Speed**: If we know the commands, the shell can be faster than using a mouse and GUI.
- **Power**: we can do a lot more complex tasks with the shell than we can with a GUI, like automating things or running background tasks.
- **Flexibility**: Shells like Bash or Zsh have lots of features and customization options that make it easier to work efficiently.

9.What is Linux?

Linux is an **operating system**—like Windows or macOS—that runs on our computer, but it's free and open-source. It’s what makes our computer work, managing everything from running programs to handling files, and connecting to the internet.

Key Points About Linux:

1. **Free and Open Source**:
    - Unlike Windows or macOS, which are **closed-source** and we have to pay for them, Linux is **free** to use. Anyone can download it, modify it, and even share it with others.
    - Since it’s open-source, developers can look at the code behind it, make changes, and improve it.
2. **Linux is Everywhere**:
    - Even though we might not see it, Linux is used in **many devices**:
        - **Servers**: The majority of websites and online services run on Linux.

        - **Smartphones**: Android (which is the most popular mobile OS) is based on Linux.
        - **Embedded Systems**: Devices like smart TVs, routers, and even some cars run on Linux.

3. **The Kernel**:
    - Linux is technically the name of the **kernel**, which is the core part of the operating system. The kernel manages the hardware and software interaction.
    - But when people talk about "Linux," they often mean the full operating system that includes the kernel and all the tools that make it useful.
4. **Distributions (Distros)**:
    - There are many different **versions** of Linux, called **distributions** (or "distros"). Each distro has different features, tools, and user interfaces, but they all use the Linux kernel at their core.
    - Some popular Linux distros are:
        - **Ubuntu**: Known for being beginner-friendly, great for everyday use.
        - **Fedora**: A cutting-edge distro with the latest software.
        - **Debian**: Stable and reliable, used by many server administrators.
        - **Arch Linux**: A more advanced, minimalistic distro for experienced users.
5. **Command Line vs. Graphical User Interface (GUI)**:
    - Linux offers a **command line** (shell) for advanced users to control the system by typing commands, but it also has a

graphical user interface (GUI), like Windows or macOS, for people who prefer clicking on things.

  - Many Linux distros come with **desktop environments** (like GNOME, KDE, or Xfce) that make the system look more like a regular computer with windows, icons, and buttons.

6. **Why People Like Linux**:

  - **Customization**: Linux is very customizable. we can change almost everything—from the look of the desktop to how the system behaves.
  - **Security**: Linux is known for being more secure than some other operating systems. It's less targeted by viruses and malware.
  - **Stability and Performance**: Linux is very stable and can run on old hardware, making it a good choice for people with older computers.
  - **Community**: Since it's open-source, there's a huge global community of developers and users who help each other out. If you run into problems, there's a lot of support available.

Example of Linux in Action:

- When you use Linux, we can do things like:
  - Install software (using commands or an app store).
  - Manage files with commands or file managers (like Nautilus or Dolphin).
  - Browse the web, watch videos, play games, and more, just like any other operating system.

10.What is Bash?

➔ **Bash** (which stands for **Bourne Again Shell**) is a **command-line interface** or **shell** used in many Linux and macOS systems. It's the place where we can type commands to tell our computer what to do.

Key Points About Bash:

1. **Command Interpreter**:
    - Bash is a **command interpreter**—it takes the commands we type and makes our computer run them. For example, if we type ls, it will list the files in the current directory.
    - we use Bash to interact with the operating system, run programs, and manage files.
2. **Text-Based**:
    - Bash doesn't have fancy graphics like a desktop environment. Instead, we type text-based commands, and Bash shows we the results in the terminal.
    - For example, to see the contents of a folder, we type ls, and Bash will show the list of files in that folder.
3. **Bash vs Other Shells**:
    - Bash is just one of many types of shells. Others include **Zsh**, **Fish**, and the original **Bourne Shell** (sh). Bash is the most common shell used in Linux systems.
    - It's a **command-line shell**, so it's different from a graphical interface where we click on icons, but it's extremely powerful for users who want to perform tasks quickly and efficiently.
4. **Why People Use Bash**:
    - **Efficiency**: Once we get the hang of it, we can do things much faster than using a mouse and clicking around.

- **Automation**: we can write **scripts** (small programs) in Bash to automate repetitive tasks. For example, we can create a script to back up our files automatically every day.

- **Control**: Bash gives we a lot of control over our system, allowing we to manage files, install software, and run complex tasks with just a few lines of text.

5. **Bash Commands**:

    - Common commands we can use in Bash include:

        - ls – List the files in the current directory.
        - cd – Change the current directory.
        - mkdir – Create a new directory.
        - echo – Print text to the screen.
        - cp – Copy files.
        - rm – Remove (delete) files.

➔ You can also chain commands together, like this:

```
ls && echo "Done!"
```

- This will list the files and then print "Done!" if the ls command runs successfully.

6. **Bash Scripts**:

    - A **Bash script** is just a file that contains a series of Bash commands that will be executed in order. Scripts can be used to automate tasks.

➔ For example, a simple Bash script to say "Hello" could look like this:

```
#!/bin/bash
```

➔
```
echo "Hello, World!"
```

- When you run this script, it will print "Hello, World!" to the screen.

Example of Using Bash:

1. Open our terminal (Bash shell).

➔ Type a command like this to list files in our home directory:
ls

2. Bash will show we the files and folders in that directory.
3. we can also change directories (like moving between folders) with:
cd Documents

   Now we're in the **Documents** folder.

11.You have a new empty hard drive that you will use for Linux. What is the first step you use.

➔ When we have a **new empty hard drive** and we want to use it with Linux, the **first step** is to **identify the drive** and **prepare it**. Here's the simplest way to start:

Step 1: Identify the New Hard Drive

Before we can do anything with the new hard drive, we need to find out what it's called on our system.

1. **Open a terminal**.

➔ Run the command to list all storage devices:
lsblk

2. This will show a list of all connected drives and their partitions.

   - The new hard drive will probably be listed as something like /dev/sdb, /dev/sdc, or /dev/nvme0n1 (depending on the type of drive).

     we should be able to identify it by its size or based on what is currently on the other drives (e.g., if our main drive is /dev/sda, the new one might be /dev/sdb).

Next Steps After Identifying the Drive:

Once we know which device is our new hard drive (let’s assume it’s /dev/sdb), the next steps will be:

1. **Partition the drive** (create sections of space on it).
2. **Format the partition** (so Linux can use it).
3. **Mount the partition** (make it accessible like a folder).

But the **first step** is always identifying our new hard drive with the lsblk command.

Let me know if we'd like help with the next steps, like partitioning and formatting!

12.Write the Linux command to show the current working directory.

➔ To show the **current working directory** in Linux, you can use the command:

```
pwd
```

What it does:

- pwd stands for "**print working directory**".
- When we type pwd and press Enter, it shows we the full path of the directory we are currently in.

Example:

If we’re in the home directory of a user called "john", running pwd might output something like:

```
/home/john
```

This means you're currently in the **/home/john** directory.

13. write the Linux command to get help with various options

- ➔ To get help with various options for a command in Linux, we can use the --help option or the man (manual) command. Here's how they work:

1. Using --help:

Most Linux commands come with a built-in help option. To use it, simply type the command followed by --help:

```
command --help
```

For example, to get help on the ls command (which lists files), type:

```
ls --help
```

This will show we a list of all options we can use with the ls command, like -l for detailed listing or -a to show hidden files.

2. Using man (Manual Pages):

The man command shows a detailed manual for any Linux command. To use it:

```
man command
```

For example, to see the manual for the ls command:

```
man ls
```

This will show we a detailed guide on how the ls command works, including all options and usage examples.

14. Write the linux comman! to display what all users are currently doing.

➔ To see what all users are currently doing on a Linux system, we can use the w command or the who command. These commands show the users logged in and what they are doing.

1. Using the w Command:

The w command provides detailed information about the users who are currently logged in and what they are doing.

```
w
```

What it shows:

- **User**: The username of the logged-in user.
- **TTY**: The terminal from which the user is logged in.
- **FROM**: The IP address or hostname of the user's machine.
- **IDLE**: How long the user has been idle.
- **JCPU**: The time used by all processes connected to the user's terminal.
- **PCPU**: The time used by the user's current process.
- **WHAT**: The command or process the user is currently running.

Example:

```
$ w

 10:00:00 up 5 days,  2:12,  3 users,  load average: 0.18, 0.25, 0.30

USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT

john     pts/0    192.168.1.5      09:50    0.00s  0.01s  0.00s w
```

```
alice    pts/1    192.168.1.6      09:55    2.00s  0.03s  0.01s bash
bob      pts/2    192.168.1.7      09:57    3.00s  0.04s  0.01s top
```

2. Using the who Command:

If we want just a list of the users who are logged in without the extra details, we can use the who command:

```
who
```

Example:

```
$ who
john     tty1         2025-04-24 08:10
alice    pts/0        2025-04-24 09:50 (:0)
bob      pts/1        2025-04-24 09:55 (192.168.1.7)
```

15. write the Linux command to get information about the operating system

➔ To get information about the Linux operating system, we can use the following commands:

1. Using uname Command:

The uname command shows basic information about the system.

To get general information, use:

```
uname
```

- This will show the kernel name (e.g., Linux).

To get more detailed information, like the kernel version and architecture, use:

```
uname -a
```

- This will give us a lot of details, such as:
    - Kernel name
    - Kernel version
    - Architecture (e.g., x86_64 for 64-bit)
    - Hostname (name of our machine)

Example output:

```
Linux mycomputer 5.4.0-104-generic #118-Ubuntu SMP Thu Apr 8
12:43:02 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```

2. Using lsb_release Command:

If we want information about the **Linux distribution** (like Ubuntu, Fedora, etc.), we can use the lsb_release command:

```
lsb_release -a
```

This will give us details about our distribution, such as:

- **Distributor ID**: Name of the distro (e.g., Ubuntu)
- **Description**: A full description of the OS version
- **Release**: The version number (e.g., 20.04)
- **Codename**: The codename of the version (e.g., focal)

Example output

```
Distributor ID: Ubuntu

Description:    Ubuntu 20.04 LTS

Release:        20.04

Codename:       focal
```

3. Using /etc/os-release File:

For a quick view of Linux OS details, can read the /etc/os-release file. This file contains information about our system's distribution.

```
cat /etc/os-release
```

This will show the OS name, version, and other details.

Example output:

```
NAME="Ubuntu"

VERSION="20.04 LTS (Focal Fossa)"

ID=ubuntu

ID_LIKE=debian

VERSION_ID="20.04"
```

16. Write the Linux command to create a hard link of a file.

- ➔ In Linux, a **hard link** allows us to create another name for an existing file. Both names (the original and the hard link) point to the same data on the disk, so changes made to one will affect the other.

To create a hard link, use the ln command:

Syntax:

```
ln original_file hard_link_name
```

Example:

Let's say we have a file called file.txt, and we want to create a hard link to it named file_link.txt. we would run:

```
ln file.txt file_link.txt
```

What happens:

- Both file.txt and file_link.txt will point to the **same data** on the disk.
- If we edit file.txt, the changes will be reflected in file_link.txt, and vice versa.
- Even if you delete one of the files (e.g., file.txt), the data is still accessible through file_link.txt because both links point to the same data.

Important Notes:

- Hard links cannot be created for directories (except for the special . and .. entries).
- Hard links are limited to the same **filesystem**—we can't link files across different drives or partitions.

Let me know if you need further details!

17. Write the Linux command to create a soft link of a file as well as Directory.

➔ **For a file:**

```
ln -s /path/to/original/file /path/to/link
```

For a directory:

```
ln -s /path/to/original/directory /path/to/link
```

Explanation:

- ln -s is the command to create a symbolic (soft) link.
- /path/to/original/file or /path/to/original/directory is the path to the file or directory we're linking to.

- /path/to/link is the path where we want to create the symbolic link.

For example:

If we have a file file.txt in /home/user/docs/, and we want to create a soft link to it in /home/user/links/, the command would be:

```
ln -s /home/user/docs/file.txt /home/user/links/file_link.txt
```

- If we have a directory dir1 in /home/user/folder/ and want to create a symbolic link to it in /home/user/links/, the command would be:
  ```
  ln -s /home/user/folder/dir1 /home/user/links/dir1_link
  ```

18. Write the Linux command! to search for specific pattern in a file.

➔ **Command:**

```
grep "pattern" /path/to/file
```

Explanation:

- grep: This is the command to search for text patterns in files.
- "pattern": The text or pattern we're searching for (it can be a word, number, or regular expression).
- /path/to/file: The file in which we're searching for the pattern.

  Example:

If we want to search for the word "apple" in a file called fruits.txt, we would use:

```
grep "apple" fruits.txt
```

This command will print all lines in fruits.txt that contain the word "apple".

Additional options:

Case-insensitive search: Use -i to ignore case differences:
grep -i "apple" fruits.txt

- **Search recursively in a directory**: Use -r to search all files in a directory:

  ```
  grep -r "apple" /home/user/documents/
  ```

19. Write the Linux command to show the use of basic regular expressions using grep command.

➔ Simple Linux command that shows how to use **basic regular expressions** with the grep command:

```
grep '^hello' filename.txt
```

What this does:

- grep: the command to search text using patterns.
- '^hello': a **basic regular expression**. The ^ means "start of a line", so this pattern matches any line that **starts with** the word hello.
- filename.txt: the file in which grep will search.

Example:

If filename.txt contains:

```
hello world

say hello

hello again

goodbye
```

Then the command will output:

hello world

hello again