Day 1 – Introduction to Red Hat Linux and Basic Terminal Commands

Introduction

The first session of the **RHCSA – Red Hat Certified System Administrator Workshop** was conducted under the **IEEE Council at VVITU (Vasireddy Venkatadri International Technological University)**. This course aims to introduce students to essential Linux administration skills and prepare them for real-world system management using **Red Hat Enterprise Linux (RHEL)**. The workshop is specifically curated for B.Tech students to build foundational knowledge in a professional Linux environment.

During this session, we began by installing the **Red Hat Linux Operating System** in **VirtualBox** using the .iso installation file. We also gained theoretical knowledge about:

- What is **Linux**?
- What is **Red Hat Linux OS**?
- Differences between regular Linux distributions and enterprise-grade systems like **RHEL**
- Importance of terminal commands in system administration

Documentation for Day 1

Linux Terminal Practice - Day 1 Documentation

User and Root Account Switching

- su - root
  Switches from the current user account to the root (superuser) account. This provides administrative access.
- su - bhargavsai_2005
  Switches back from the root account to the user account named **bhargavsai_2005**.

Prompt Details (User Account Terminal)

- bhargavsai_2005 – This is the username
- @ – Separator between the username and hostname
- vbox – The hostname of the system (in this case, a VirtualBox environment)
- ~ – Represents the user's home directory
- $ – Indicates that the terminal is running as a normal user

- # – Indicates that the terminal is running as the root user

---

Basic Terminal Concepts and Commands

- whatis command_name – Displays a brief description of the command.
- touch filename – Creates an empty file with the given name.
- mkdir directoryname – Creates a new directory with the specified name.
- ls – Lists all files and directories in the current location.

Hostname Commands

- hostname -s – Displays the short name of the current host machine.
- hostname -i – Displays the IP address of the machine.

History Command

- history – Lists all the commands executed so far in the current session.

Types of Files in Linux

- **Regular Files (-)** – Standard files like text, data, and program files
- **Directory Files (d)** – Directories or folders that hold other files
- **Special Files:**
    - Character Files (c) – Device files that transmit data character by character
    - Block Files (b) – Device files that transmit data in blocks
    - Link Files (l) – Symbolic links to other files
    - Pipe Files (p) – Used for inter-process communication
    - Socket Files (s) – Used for communication between processes, usually over a network

---

File Management and Display

- cat filename – Displays the content of a file
- echo "Text" > filename – Creates a file and writes the given text into it
- touch --help – Displays help information and options for the touch command

---

Online Lab Credentials – Manual Login Through Chrome

- **Username:** student
- **Password:** redhat

RedHat OS Credentials (Virtual Box Machine)

- **Username:** bhargavsai_2005
- **Password:** Absv2005@
- **Root_password:** Absv2005@

Commands Executed – User Account (bhargavsai_2005)

1. exit – Exit from the current session or shell.
2. ls – List files and directories.
3. su - root – Switch to root user.
4. history – Show command history.
5. su - root – Switch to root again.
6. exit – Exit the root shell.
7. whoami – Show current user.
8. whatis whoami – Description of the whoami command.
9. whatis useradd – Description of the useradd command.
10. su - root – Switch to root user again.
11. clear – Clear the terminal screen.
12. history – Show previous command history.

Commands Executed – Root Account

1. whoami – Confirms logged-in user is root.
2. ls – Lists files and folders.
3. touch file1 – Creates a file named file1.
4. ls – Confirms file creation.
5. mkdir dir1 – Creates a directory named dir1.
6. ls – Lists new contents (should show file1 and dir1).
7. hostname -s – Shows the short hostname.

8. hostname -i – Shows the IP address.
9. -ls – Mistyped command, no output.
10. -l – Mistyped command, no output.
11. ls -l – Lists files with detailed information.
12. echo "Good Afternoon" – Prints "Good Afternoon" on screen.
13. echo "Good Afternoon" > file2 – Writes the text into a file named file2.
14. ls – Shows file2 created.
15. cat file2 – Displays contents of file2.
16. touch --help – Shows help manual for touch.
17. history – Lists previous commands.
18. wc f1 – Mistyped; file likely doesn’t exist.
19. wc file1 – Shows word, line, and character count of file1.
20. cat file1 – Displays contents of file1 (probably empty).
21. cat file2 – Displays "Good Afternoon".
22. wc file2 – Shows count of lines, words, characters in file2.
23. user – Invalid command; Linux returns an error.
24. wc -c file2 – Shows character count of file2.
25. ec -l file2 – Mistyped; should be wc -l file2.
26. wc -l file2 – Shows number of lines in file2.
27. wc -w file2 – Shows number of words in file2.
28. logout – Logs out of the root account.
29. history – Displays history again.
30. clear – Clears the terminal screen.
31. exit – Ends root session.
32. whoami – Confirms logged-in user.
33. mandb – Updates the manual page database.
34. whatis useradd – Shows brief description of useradd.
35. whatis sudo – Shows brief description of sudo.
36. which useradd – Displays location of useradd binary.
37. su - bhargavsai_2005 – Switches back to normal user.
38. exit – Exits the user session.

39. whoami – Verifies current user identity.
40. history – Final list of commands executed.

Practical Tasks Performed

- Switching between user and root accounts using su command
- Identifying terminal structure: username, host, home directory indicator ~, and shell prompt symbols ($ for user, # for root)
- Creating files and directories using touch and mkdir
- Viewing directory contents with ls and its options
- Using whatis to understand Linux command descriptions
- Displaying and redirecting text using echo and cat
- Checking system hostname and IP address using hostname -s and hostname -i
- Exploring types of Linux files (regular, directory, special files)
- Counting lines, words, and characters in files using wc
- Learning about terminal history and command tracking
- Accessing command help using --help

Conclusion

Day 1 laid the groundwork for working in a Linux environment, especially in a Red Hat-based distribution. We not only learned how to set up the OS in a virtual machine but also understood essential terminal commands, file types, and user management basics. This session provided a practical and conceptual starting point for deeper exploration into system administration.

The experience was interactive, informative, and exciting, setting the right tone for the rest of the RHCSA workshop.
We look forward to mastering Linux system operations and becoming confident Red Hat Certified System Administrators.

Title: Day 2: "Linux User and Group Management: Creating and Configuring Users, Groups, and Password Policies"

File Manager & File Operations

The **File Manager** is a system that organizes files and folders on a computer or system. It works by managing a bunch of folders, which in turn, contain a bunch of files. The following directories are commonly found in a Linux system and perform specific roles:

1. **Bin Directory**
    - This directory contains essential user commands that are required for system operation.
    - These commands are stored here to be accessed by users for regular system operations.
2. **Boot Directory**
    - The Boot directory is responsible for system startup.
    - It holds the files needed to initialize and boot the system or computer.
3. **Dev Directory**
    - The Dev directory stores files related to devices and disk management.
    - It contains device files, which allow the system to interface with hardware devices.
4. **Etc Directory**
    - This directory contains system-specific configuration files.
    - These files define system settings and application configurations that are critical for system operations.
5. **Home Directory**
    - The Home directory is where regular user files are stored.
    - Each user has a subdirectory in the **Home** directory to store personal files.
6. **Root Directory**
    - The Root directory is the directory for the superuser (admin).

    - This directory has elevated privileges and contains files critical for system maintenance and administration.

7. **Run Directory**
    - The Run directory stores runtime data for programs that need it while running.
    - Data in this directory is not persistent, meaning it gets erased after a reboot.
8. **Sbin Directory**
    - This directory contains commands that are used by the superuser (admin).
    - These commands are typically used for system maintenance and troubleshooting.
9. **Temp Directory**
    - The Temp directory stores temporary files.
    - Files in this directory are deleted automatically after 10 days if they are not modified.
10. **Usr Directory**
    - The Usr directory contains subdirectories such as **bin**, **local**, **sbin**, and **temp**.
    - This directory is essential for storing user-related data and system files that aren't part of the core operating system.
11. **Var Directory**
    - The Var directory contains variable data, including system logs and records of external activities.
    - It also has a **temp** directory, where temporary files are stored and automatically deleted after 30 days.

Summary of Each Block

- **Bin Directory:** Stores essential user commands.
- **Boot Directory:** Contains files required to boot the system.
- **Dev Directory:** Houses device-related files for disk management.
- **Etc Directory:** Stores system configuration files.
- **Home Directory:** Contains personal files for regular users.
- **Root Directory:** The superuser’s directory with critical files.
- **Run Directory:** Holds temporary runtime data that is erased after reboot.
- **Sbin Directory:** Contains system maintenance commands for the superuser.
- **Temp Directory:** Stores temporary files, deleted automatically after 10 days if not modified.
- **Usr Directory:** Includes subdirectories that store various user-related data.

- **Var Directory:** Contains variable files such as logs and external data, including a temporary files directory.

Here's the content structured and formatted properly for documentation purposes:

Changing Paths Between Directories: Relative and Absolute Paths

When navigating between directories in a file system, two types of paths are commonly used: **Relative Path** and **Absolute Path**. These two paths serve different purposes depending on how the user is navigating to their destination.

Relative Path

- A **Relative Path** refers to the path that is relative to the current working directory. The user does not have a predefined way to reach the destination directory, so they must move step-by-step through the file system.
- In the case of a relative path, the user manually changes directories one by one, checking each directory for the required files until they locate the destination.

Example (Relative Path Navigation):

1. Assume the user wants to check if the required files are present in a specific directory. The user starts from their current working directory and navigates to the destination directory manually, checking along the way.
2. **Commands for Relative Path Navigation:**

```
[root@vbox folks]# cd
[root@vbox ~]# cd downloads
[root@vbox downloads]# cd music
[root@vbox music]# cd folks
[root@vbox folks]# ls
song1  song2
[root@vbox folks]#
```

   - This shows the user manually navigating through directories (downloads -> music -> folks) and using the ls command to check for files.

Absolute Path

- An **Absolute Path** refers to the full path from the root directory to the target directory. The user knows exactly where the files are located, and they can reach the destination directly by specifying the full path.
- With an absolute path, the user doesn't need to navigate step-by-step. They can use one command to go directly to the required directory.

Example (Absolute Path Navigation):

1. Suppose the user knows the exact location of the required files. They can directly access the directory by specifying the full absolute path.
2. **Commands for Absolute Path Navigation:**

```
[root@vbox ~]# ls downloads/music/folks
song1  song2
```

- This example shows the user using the absolute path to directly list the files in the folks directory.

Creating Directories with Absolute Paths:

1. You can also use absolute paths to create directories at specific locations in the file system.
2. **Example of Directory Creation:**

```
[root@vbox ~]# mkdir -p IPL/MI/CSK/RCB
[root@vbox ~]# ls
anaconda-ks.cfg  dir1  downloads  file1  file2  file3
file4  IPL
[root@vbox ~]# cd IPL
[root@vbox IPL]# ls
MI
[root@vbox IPL]# cd MI
[root@vbox MI]# ls
CSK
[root@vbox MI]# cd CSK
[root@vbox CSK]# ls
RCB
[root@vbox CSK]# cd RCB
[root@vbox RCB]# ls
```

- Here, the user creates nested directories (IPL -> MI -> CSK -> RCB) and navigates to them step by step.

Summary of Key Points

- **Relative Path:** The path is relative to the current working directory, and the user must navigate step-by-step to reach the destination.
- **Absolute Path:** The full path is known, and the user can directly navigate to the destination using one command.
- **Directory Creation (Absolute Path Example):** You can create nested directories using the mkdir -p command and navigate to them using absolute paths.

File Operations and Management in Linux

In this section, we will explore file management and various commands used to handle files and directories in Linux.

Creating Directories Using mkdir -p

Using the mkdir -p command, you can create a parent directory along with its subdirectories in one go. Here's how it's done:

- The mkdir -p command ensures that the parent directories are created if they do not already exist.

Example:

[root@vbox ~]# mkdir -p IPL/MI/CSK/RCB

Modifying a File

To modify a file in Linux, you can use commands like cat with the >> operator to append text to an existing file. Here is an example of modifying the content of a file:

- Using cat to view the file content:

[root@vbox ~]# cat file4

Welcome to RedHat

- Using cat >> file4 to append content to the file:

[root@vbox ~]# cat >> file4

My name is Adhimulam Bhargav Sai Viswanath

I enrolled for this Red Hat Linux OS Course.

^C

- After appending, the file content will be:

[root@vbox ~]# cat file4

Welcome to RedHat

My name is Adhimulam Bhargav Sai Viswanath

I enrolled for this Red Hat Linux OS Course.

Reading File Content

There are several commands available to read files in Linux. The most common ones are cat, tac, head, tail, more, and less.

1. **Using cat**: This command displays the entire content of the file from top to bottom.
2. [root@vbox ~]# cat file4
3. **Using tac**: This command displays the content of the file from bottom to top.

4. [root@vbox ~]# tac file4
5. **Using head**: This command shows the top 10 lines of the file.
6. [root@vbox ~]# head file4
7. **Using tail**: This command shows the last 10 lines of the file.
8. [root@vbox ~]# tail file4
9. **Using more**: This command allows you to scroll through the file content.
10. [root@vbox ~]# more file4
11. **Using less**: This command is similar to more, but it allows backward navigation through the file.
12. [root@vbox ~]# less file4

Printing Specific Lines of a File

To print a specific number of lines from a file, you can use the cat command with the -n option, which allows you to specify the line number.

Example:

cat -2 file4

File Permissions and File Information

When listing files in Linux, the ls command shows the file permissions, owner, size, timestamp, and more. Here is an example of the output:

-rw-------. 1 root root 821 Apr 21 10:00 anaconda-ks.cfg

drwxr-xr-x. 2 root root 6 Apr 21 15:15 dir1

drwxr-xr-x. 3 root root 19 Apr 22 09:10 downloads

-rw-r--r--. 1 root root 0 Apr 21 15:15 file1

-rw-r--r--. 1 root root 15 Apr 21 15:24 file2

-rw-r--r--. 1 root root 29 Apr 22 09:19 file3

-rw-r--r--. 1 root root 106 Apr 22 09:26 file4

drwxr-xr-x. 3 root root 16 Apr 22 09:23 IPL

Here, the details include:

- **Permissions**: The file's access rights (read, write, execute).
- **Owner**: The user and group that own the file.
- **Size**: The size of the file.

- **Timestamp**: The last modification date and time.
- **Name**: The file or directory name.

Hidden Files and Detailed Listing

To list hidden files (those starting with a dot), you can use the ls -a command. For a detailed listing, including permissions and other file attributes, use ll.

Example:

ls -a

ll

Copying Files and Directories

To copy files from one location to another, you can use the cp command:

- Copy a file:

cp source_filename destination_filename

- Copy a directory:

cp -r source_directory destination_directory

Removing Files and Directories

To remove a file, use the rm command. To forcefully remove a file or directory without confirmation, you can use rm -f or rm -rf for directories.

Example:

- Remove a file:

rm filename

- Forcefully remove a file:

rm -f filename

- Remove a directory:

rm -r directory_name

- Forcefully remove a directory:

rm -rf directory_name

This documentation covers basic file operations in Linux, including navigating directories, modifying files, and using commands to read, copy, and remove files. These are essential skills for managing files in a Linux environment effectively.

Here is an explanation of the various commands and their usage in Linux for file management and linking, as mentioned in your document:

1. **Creating Links**:
    - **Hard Links**:
        - A hard link creates another reference (link) to the same inode, meaning both the original file and the hard link will point to the same data on the disk.
        - Hard links cannot span across different filesystems, and directories cannot be hard linked.
        - Hard links take up more memory because they occupy separate directory entries, but they don't create duplicate copies of the file.
        - If the original file is deleted, the content remains accessible through the hard link because both the original file and the hard link reference the same inode.
    - **Soft Links (Symbolic Links)**:
        - Soft links, or symlinks, are special files that point to another file or directory by name. A symlink can span different filesystems and can link to directories.
        - Unlike hard links, soft links take up less memory because they only store the path to the original file, not the data.
        - However, if the original file is deleted, the soft link becomes broken and no longer points to valid data.
2. **Creating and Testing Hard Links**:
    - Using the ln command creates hard links. In your example, the following commands were used:
    - ln file3 hardlink1
    - ln file3 hardlink2
    - ln file3 hardlink3
    - These commands create three hard links pointing to file3. When you modify one of the hard links, the change is reflected in all other links.
3. **Creating Soft Links**:
    - To create soft links, the ln -s command is used:
    - ln -s hardlink1 softlink1

- ln -s hardlink1 softlink2
- ln -s hardlink1 softlink3
- These soft links point to hardlink1. If hardlink1 is removed, the soft links become broken.

4. **Removing Files**:
    - The rm command is used to remove files, and rm -f forces the removal without asking for confirmation. Removing a hard link or a soft link does not remove the content of the file until all links pointing to the inode are deleted.
5. **File Management**:
    - The ls command is used to list files and directories.
        - ls -a lists hidden files and directories.
        - ls -lh shows detailed information about files, including permissions, ownership, size, and modification time.
        - ls *A and ls A* are used to match files starting with "A", but wildcard matching needs to be done carefully.
6. **Directory Management**:
    - The mkdir command creates directories. In your example, you created a directory glob and then added several files using the touch command:
    - mkdir glob
    - cd glob/
    - touch Arvind Anu Aarya Abhi Bhargav Bhanu Branhma Charlie Cat
    - The ls command was used to list the files, and wildcards (like ls A* or ls *A) were used to filter files based on their names.

These concepts help in efficiently managing files and directories within the Linux environment, and understanding links is crucial for file system integrity and saving storage.

Creating Multiple Files at Once

To create multiple files at once in the terminal, you can use brace expansion:

1. **Create multiple files in a range**:
2. touch file{5..10}.txt

This creates files from file5.txt to file10.txt.

3. **Create files with combinations**:
4. touch file{5..10}{a..c}.txt

This creates the following files: file5a.txt, file5b.txt, file5c.txt, file6a.txt, file6b.txt, file6c.txt, etc.

Variables and Echo Commands

- To define a variable and display its value, you use echo:
- username=Bhargav
- echo $username

This prints the value of the username variable (Bhargav).

Using Vim Editor

Vim is a powerful text editor with several modes. Here's an overview of how to use Vim:

1. **Modes in Vim**:
    - **Command Mode**: You can navigate the file, delete lines, copy, paste, etc.
    - **Insert Mode**: Press i to switch to Insert Mode where you can type content.
    - **Extended Mode**: Press : to access extended commands like saving or quitting.
2. **Basic Commands**:
    - To switch from Command Mode to Insert Mode: Press i.
    - To return to Command Mode: Press Esc.
    - Save and exit: In Command Mode, type :wq (write and quit).
    - Delete a single line: In Command Mode, type dd.
    - Delete multiple lines: In Command Mode, type ndd where n is the number of lines to delete.
    - Copy (yank): yy (yank a line).
    - Paste: p (paste the yanked content).
    - Delete a specific word: In Command Mode, type dw.
    - Set line numbers: Type :set nu in Extended Mode to show line numbers, and :set nonu to hide them.
3. **Navigation**:
    - Go to the beginning of the file: gg.
    - Go to the last line: G.
    - Go to a specific line number: Type :n where n is the line number.
4. **Visual Mode**:
    - Press v to select text character by character.
    - Press Ctrl+v for a visual block (select columns).
    - Press Shift+v to select entire lines.
5. **Manual Pages**:

- To get help on any command, use the man command:
- man command_name

For example, to learn about ls, you would type man ls.

Redirecting Output to Files

- To redirect the output of a command to a file, use the > operator:
- history > output1.txt

This will store the history of commands in the file output1.txt.

Input, Output, and Errors

- **Standard Input** (stdin): The input provided by the user, often from the keyboard (denoted by 0).
- **Standard Output** (stdout): The output shown on the screen (denoted by 1).
- **Standard Error** (stderr): Error messages shown on the screen (denoted by 2).

This guide covers essential file creation, Vim usage, output redirection, and basic command-line navigation. Let me know if you need further clarifications!

Users & Groups

In Linux, there are different types of user accounts, each with distinct privileges and roles. Here's a breakdown of the user and group management system:

User Accounts

1. **Root User**:
    - The root user, also known as the superuser, has complete access to the system.
    - This user can perform any action without restrictions (e.g., installing software, modifying system files).
    - It is represented by # in the terminal.
2. **Normal Users**:
    - These users have limited access to the system and require permission to perform certain actions.
    - It is represented by $ in the terminal.

Types of Users

1. **Root User**:
    - The root user can access and modify everything on the system.
2. **System Users**:
    - **System Static Users**:

        - These users manage background services that are essential for the system to run.
        - For example, system services running when you switch on your phone.
    - **System Dynamic Users**:
        - These users manage on-demand tasks or applications.
        - They have administrator permissions to install or delete applications (like the permission prompt when installing apps).
3. **Normal Users**:
    - These are regular users who do not have administrative privileges and require permission to perform certain system-level tasks.

Unique User IDs (UID)

Each user has a unique identifier (UID). Here are the typical UID ranges:

- **Root User**: UID = 0
- **System Users**: UID = 1 to 999
    - **System Static Users**: UID = 1 to 200
    - **System Dynamic Users**: UID = 200 to 999
- **Normal Users**: UID = 1000 to 65,000

Creating a User

To create a new user, you can use the following commands:

- useradd username
    - Or the alternative command: adduser username

For example:

useradd BhargavSai

You need to be a root user or have root privileges to create a new user. If you are a normal user, you may get a **Permission Denied** error.

Example of User Creation

1. Check which useradd command is being used:
2. which useradd
3. # Output: /usr/sbin/useradd
4. Attempt to add a user (without root privileges):
5. useradd u1
6. # Output: useradd: Permission denied.

7. If you are the root user, you can add a user:
8. whoami
9. # Output: root
10. useradd Bhargav
11. id Bhargav
12. # Output: uid=1001(Bhargav) gid=1001(Bhargav) groups=1001(Bhargav)
13. The system creates the following for the user:
    - **Unique ID (UID)**: 1001
    - **Primary Group**: A group is created with the same name as the username (Bhargav).
    - **Group ID (GID)**: 1001
    - **Home Directory**: /home/Bhargav
    - **Login Shell**: /bin/bash
14. You can see the user details in the /etc/passwd file:
15. cat /etc/passwd

Example output:

Bhargav:x:1001:1001::/home/Bhargav:/bin/bash

- **Bhargav**: Username
- **x**: Encrypted password (stored in /etc/shadow)
- **1001**: Unique User ID (UID)
- **1001**: Group ID (GID)
- **/home/Bhargav**: User's home directory
- **/bin/bash**: User's login shell

Summary

- **Root user** has full system access and is represented by #.
- **Normal users** have limited access and require permissions for certain actions, represented by $.
- Users are assigned unique IDs, and system users are categorized into static and dynamic users based on their tasks.
- A new user is created using useradd, and the system creates necessary details like UID, GID, home directory, and login shell.

Modifying Users in Linux

When managing users, you can modify existing users with the usermod command. Here are some ways to modify users and their attributes.

1. Modifying a User's Shell

To change the login shell for an existing user, use the following command:

usermod -s /bin/bash username

For example, to change the shell for the user Sai to /bin/bash, run:

usermod -s /bin/bash Sai

Then, you can switch to that user with:

su - Sai

2. Locking and Unlocking a User Account

- **Lock a User Account**:
  To lock an account, use the usermod command with the -L option:
- usermod -L username

For example, to lock the Sai user account:

usermod -L Sai

- **Unlock a User Account**:
  To unlock the account, you can use the passwd command. It will prompt for the new password:
- passwd username

For example, to unlock the Sai account:

passwd Sai

Enter the new password when prompted.

3. Setting Passwords for Multiple Users at Once

You can use the chpasswd command to set passwords for multiple users in one go. The syntax is:

chpasswd

For example, to set the passwords for multiple users like Bhargav, Sai, and Tingari, create an input file with the following format:

Bhargav:redhat123

Sai:redhat1020

Tingari:redhat2020

You can also provide this input directly in the command line:

echo -e "Bhargav:redhat123\nSai:redhat1020\nTingari:redhat2020" | chpasswd

Summary of Commands:

- **usermod -s /bin/bash username**: Changes the login shell for the user.
- **usermod -L username**: Locks the user account.
- **passwd username**: Unlocks and allows you to set a new password for the user.
- **chpasswd**: Allows you to set passwords for multiple users at once.

By using these commands, you can modify, lock, unlock, and manage user accounts effectively in a Linux system. Let me know if you'd like further clarification on any command or process!

Sudo User in Linux

A **sudo user** is a normal user who is granted privileges to execute commands typically reserved for the root user. This allows a non-root user to execute administrative tasks without being the root user.

Example in Real Life:

In a college, if the principal is the root user, the vice principal may temporarily take over some of the principal's privileges while the principal is on leave. This mirrors how a sudo user gets elevated privileges temporarily.

Granting Sudo Access to a Normal User

To give a normal user root-like access, you need to configure the /etc/sudoers file.

Steps:

1. **Edit the sudoers file**: Use the vim editor to edit the sudoers file.
2. sudo vim /etc/sudoers
3. **Add the required line**: In the file, you will see a line like this (typically around line 100):
4. ## Allow root to run any commands anywhere
5. root ALL=(ALL) ALL

To grant Sai (or any other user) sudo access, append the following line:

Sai ALL=(ALL) NOPASSWD: ALL

This line means:

    - Sai: the username.
    - ALL: can execute commands from any host.
    - (ALL): can run commands as any user.
    - NOPASSWD: does not need to enter a password to execute sudo commands.

6. **Save the changes**: Press Esc, then type :wq and hit Enter to save and quit.

Deleting a User

To remove a user, you can use the userdel command.

- To delete a user temporarily:

- userdel username
- To delete a user and their home directory:
- userdel -rf username

Groups in Linux

There are two types of groups in Linux:

- **Primary group**: The group automatically assigned to the user when created.
- **Supplementary group**: Additional groups that a user can be assigned to manually.

1. Creating a Group

To create a new supplementary group:

groupadd groupname

For example, to create a group called cloud:

groupadd cloud

2. Verifying Groups

To verify if a group has been created, you can check the /etc/group file:

cat /etc/group

For example, the output might look like this:

root:x:0:

bin:x:1:

daemon:x:2:

sys:x:3:

adm:x:4:

tty:x:5:

disk:x:6:

lp:x:7:

mem:x:8:

kmem:x:9:

wheel:x:10:bhargavsai_2005

cloud:x:1022:

web:x:1023:

bhargavsai_2005:x:1000:

Bhargav:x:1001:

Sai:x:1005:

Here:

- **Cloud** is the name of the group.
- **X** is the encrypted password field (not used for groups).
- **1022** is the group ID (GID).

Summary of Commands:

- **Grant sudo access**:
- sudo vim /etc/sudoers

Add the line for the user (e.g., Sai ALL=(ALL) NOPASSWD: ALL).

Here's how to approach the homework task step-by-step, followed by an explanation of each part:

Sure! Here's how the exercise can be rewritten as a formal problem statement:

Problem Statement:

You are tasked with performing user and group management tasks on a Linux system. The requirements for this task are as follows:

1. **Create a supplementary group:**
    - **Create a supplementary group named sysadms.**
2. **Create users and assign them to the group:**
    - **Create two users named natasha and harry.**
    - **Both natasha and harry should be members of the sysadms group.**
3. **Create a user with a non-interactive shell:**
    - **Create a user named sarah.**
    - **The sarah user should have a non-interactive shell (i.e., /usr/sbin/nologin or /bin/false).**
    - **The sarah user should not be a member of the sysadms group.**
4. **Set passwords:**
    - **Set the password for all three users (natasha, harry, and sarah) to trootent.**

Objectives:

- **Create the supplementary group and users as per the specified criteria.**
- **Assign users to the correct groups.**
- **Configure the shell for the non-interactive user.**

- **Ensure all users have the specified password.**

Deliverables:

1. **A set of commands used to create the group and users.**
2. **Verification of the group memberships and user shells.**
3. **Proof of setting the passwords for the users.**

Steps to Complete the Task:

1. Create a supplementary group named sysadms:

This group will be used to add the users natasha and harry.

sudo groupadd sysadms

2. Create users natasha and harry and add them to the sysadms group:

Use the useradd command with the -G option to specify supplementary groups. This will add both users to the sysadms group.

sudo useradd -G sysadms natasha

sudo useradd -G sysadms harry

3. Create the user sarah with a non-interactive shell:

For a non-interactive user, you can set the shell to /usr/sbin/nologin or /bin/false. This means sarah will not be able to log in interactively. Also, sarah should not be added to the sysadms group.

sudo useradd -s /usr/sbin/nologin sarah

4. Set the password for all users as trootent:

You can set the password for all the users with the passwd command. The password can be set by echoing the password into the passwd command using --stdin.

echo "trootent" | sudo passwd --stdin natasha

echo "trootent" | sudo passwd --stdin harry

echo "trootent" | sudo passwd --stdin sarah

Verification:

To check the results, you can run the following commands to verify the users, their groups, and the passwords.

1. Check the groups for natasha and harry:

These users should be part of the sysadms group.

id natasha

id harry

Expected Output:

- natasha and harry should both show sysadms under their groups.

2. Verify sarah is created with a non-interactive shell:

Check sarah's details to verify that the shell is set to /usr/sbin/nologin:

id sarah

cat /etc/passwd | grep sarah

Expected Output:

- sarah should have /usr/sbin/nologin as the shell and should not belong to the sysadms group.

3. Check the password (Shadow file):

You can also inspect the /etc/shadow file to verify that the password for all users has been set to trootent:

cat /etc/shadow

Expected Output:

- You should see entries for natasha, harry, and sarah with the encrypted password.

Explanation of the Commands:

- **groupadd sysadms**: This command creates a new group called sysadms, which can be used to assign users to a particular role or access group.
- **useradd -G sysadms natasha**: The -G option adds the user natasha to the sysadms group, in addition to their default group.
- **useradd -s /usr/sbin/nologin sarah**: This command creates the user sarah and assigns them a non-interactive shell (/usr/sbin/nologin). This means sarah cannot log in directly to the system, which is often used for service accounts.
- **passwd --stdin**: This method sets the password for a user without the need to interactively type it in. You pass the password through a pipeline (echo "password") to passwd.

Expected Final State:

- **Users**:
    - natasha and harry will be part of the sysadms group.

- sarah will have /usr/sbin/nologin as the shell and will **not** be in the sysadms group.

- **Passwords**:
  - All three users (natasha, harry, sarah) will have the password trootent.

Or we can use the commands simply AS:

Groupadd sysadms

Cat /etc/group

Useradd -G sysadms natasha

Useradd -G sysadms harry

Cat /etc/group

Useradd -s /sbin/nologin sarah

Cat /etc/passwd

Chpasswd

Cat /etc/shadow

Example Output for id Command:

For natasha and harry:

$ id natasha

uid=1001(natasha) gid=1001(natasha) groups=1001(natasha),1002(sysadms)

$ id harry

uid=1002(harry) gid=1002(harry) groups=1002(harry),1002(sysadms)

For sarah (non-interactive shell):

$ id sarah

uid=1003(sarah) gid=1003(sarah) groups=1003(sarah)

$ cat /etc/passwd | grep sarah

sarah:x:1003:1003::/home/sarah:/usr/sbin/nologin

Conclusion:

By following the steps and explanations above, you will have successfully created the users, added them to groups, set passwords, and configured a non-interactive shell for sarah.

Sure! Here's a summary of everything we've discussed and worked on so far, including the tasks for today:

Summary of Today:

1. **Linux User and Group Management**:
    - You worked with user management commands in Linux, such as creating users, adding users to groups, modifying user group memberships, and setting up sudo privileges.
    - You explored how to create and manage groups and users, specifically:
        - You used commands like useradd, usermod, and groupdel to manage users and groups.
        - You set up users in supplementary groups and assigned them to the correct primary group.
        - You learned about managing sudo privileges for users and groups by modifying the /etc/sudoers file.
        - You reviewed the /etc/shadow file to understand how user passwords are stored, including encrypted passwords and associated metadata like expiration dates.
        - You practiced setting password expiration policies using the chage command.
2. **Exercise Task (Homework)**:
    - You received a Linux user/group management exercise to complete:
        - **Task 1**: Create a supplementary group sysadms.
        - **Task 2**: Create two users natasha and harry and add them to the sysadms group.
        - **Task 3**: Create a user sarah with a non-interactive shell (i.e., no login access) and ensure she is not a member of sysadms.
        - **Task 4**: Set the password trootent for all users (natasha, harry, and sarah).
    - I provided a detailed problem statement and explained the commands to accomplish the task.
3. **Shadow File Understanding**:
    - You explored the /etc/shadow file in detail to understand how encrypted passwords are stored. We discussed different fields within the shadow file, including encryption types (e.g., SHA512) and password expiration information.

- We reviewed the structure of an encrypted password and its components (salt, hash, rounds).

4. **Password Expiration Policies**:
    - You learned how to set and manage password expiration policies using the chage command.
    - You practiced changing minimum days, maximum days, warning days, and inactive days for password changes.
5. **Commands and Syntax Explained**:
    - We discussed various Linux commands, such as:
        - useradd: For creating new users.
        - usermod: For modifying users, such as adding them to supplementary groups.
        - groupdel: For deleting groups.
        - chage: For modifying password expiration settings.
6. **Theoretical Explanation**:
    - We also reviewed the contents and purpose of files like /etc/group and /etc/shadow, which play an essential role in user and group management in Linux.
    - We clarified the purpose of fields in the /etc/shadow file, especially the encryption methods, and provided real-world examples to make sense of these concepts.

What Was Covered Earlier:

1. **User and Group Creation**:
    - You learned how to add users, modify their groups, and set group memberships.
2. **Working with /etc/group and /etc/shadow**:
    - We explored the /etc/group file, which lists groups and their members.
    - You also reviewed the /etc/shadow file, where encrypted passwords and user expiration policies are stored.
3. **Sudo Privileges**:
    - You saw how to assign sudo privileges to users through group membership and how to edit the /etc/sudoers file.
4. **Shadow File Understanding**:
    - We dove into the specifics of the /etc/shadow file format, detailing how encrypted passwords are stored and explaining the fields, like the password hash, the minimum and maximum days for password changes, and account expiration settings.

Next Steps:

- **Completing the Exercise**:
    - You can implement the user creation task with supplementary groups, set passwords, and adjust user settings as described in the exercise. Once you've completed it, you should verify it using commands like id, groups, and chage to ensure the configuration is correct.
- **Further Study**:
    - As you continue to explore Linux user and group management, consider experimenting with creating more complex user scenarios, setting up security policies, and managing multiple users for real-world application.

“Day 3: Deep Dive into Linux Security, Permissions, and Remote Access”

🔐 Security Layers in Linux

Linux provides multiple layers of security to control access and protect the system. These include:

1. **File Permissions**
2. **Special Permissions**
3. **Access Control Lists (ACLs)**
4. **Firewall**
5. **SELinux (Security-Enhanced Linux)**

📂 File Permissions in Linux

File permissions are the **first layer of security** in Linux. They define **who can access and perform actions** on files and directories.

There are **three basic types of permissions**:

Permission	Symbol	Description
Read	r	Allows viewing or reading the content of a file or listing files in a directory.
Write	w	Allows modifying file contents or changing the contents of a directory (add/remove files).
Execute	x	Allows executing a file as a program or script, or entering a directory using cd.

Let me know if you want an explanation of **special permissions** like SUID, SGID, and Sticky Bit, or examples of how to configure firewalls or SELinux policies!

📌 Events of a Permission on a File/Directory

1. **Read permission on a file**:
   Allows the user to **view or read** the contents of the file.
2. **Read permission on a directory**:
   Allows the user to **list all the files** and directories present within that directory.

3. **Write permission on a file**:
   Allows the user to **modify or update** the contents of the file.
4. **Write permission on a directory**:
   Allows the user to **perform operations** such as **adding, deleting, or copying files** within that directory.
5. **Execute permission**:
   - On **files**: Allows the file (such as a script or program) to be **executed as a command**.
   - On **directories**: Allows the user to **enter the directory** using commands like cd, and perform actions **within** it (requires read permission to list contents).

👥 File Permission Categories

Category	Explanation
User (Owner)	Has full control over the file (can read, write, and execute if allowed). The file owner is usually the one who created the file.
Group	Members of the same group can access the file as per the group permissions. The user must belong to that group.
Others	Any third-party user (not the owner or group member). Their access is controlled strictly by the "others" permission field.

📂 Example Output from Terminal

[root@vbox Files]# ls

testfile1.txt

[root@vbox Files]# ll

total 4

-rw-r--r--. 1 root root 127 Apr 22 11:14 testfile1.txt

🔍 Explanation of Permissions

- **rw-** → For **user (owner)** root
  - 🔹 Has **read and write** permissions.
  - 🔹 Can **view and modify** the file.

- **r--** → For **group** root
  - 🔷 Has **read-only** permission.
  - 🔷 Can **view** the file, but **not modify** it.
- **r--** → For **others**
  - 🔷 Also has **read-only** permission.
  - 🔷 Can **view** the file but **cannot write or execute**.

Let me know if you'd like to see how to modify these permissions with chmod or chown commands!

Permission	Effect on Files	Effect on Directories
r (read)	File contents can be viewed/read.	File names inside the directory can be listed (ls).
w (write)	File contents can be modified or overwritten.	Files can be **created**, **deleted**, or **renamed** within the directory.
x (execute)	File can be executed as a program or script.	Directory can be **entered** using cd. *(Needs r to list contents)*

Permission Type	Symbol	Numerical Value	Action	Explanation
User	u	—	—	Refers to the **owner** of the file
Read	r	4	Add/remove as +r/-r	Grants reading capability
Write	w	2	Add/remove as +w/-w	Grants writing/modifying capability
Execute	x / X	1	Add/remove as +x/-x	Grants executing or running the file
Others	o	—	—	Refers to users **other than owner/group**
All (User, Group, Others)	a	—	Use as a+r, a-w, etc.	Applies to all categories

---

how to **change file permissions using chmod** in **two methods: symbolically and numerically**, along with examples.

🔧 1. Symbolic Method

This method uses letters (u, g, o, a) and symbols (+, -, =) to modify permissions.

Syntax:

chmod [who][+/-/=][permission] filename

Example:

chmod u+x file.txt

🔷 *Adds execute (x) permission to the user (u) for file.txt.*

🔢 2. Numeric Method

This uses a **3-digit number** where each digit represents the permissions for **user**, **group**, and **others** respectively.

Permission Values:

- r = 4
- w = 2
- x = 1

Example:

chmod 755 file.txt

🔷 *Means:*

- **User**: 7 → rwx (4+2+1)
- **Group**: 5 → r-x (4+0+1)
- **Others**: 5 → r-x (4+0+1)

🔐 chown Command – Change File Owner and Group

Definition: The chown command (short for **change owner**) is used to **change the ownership** of a file or directory to a different **user** and/or **group**.

🧠 Basic Syntax:

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

- OWNER: The new user who will own the file.
- GROUP: (Optional) The new group the file will belong to.
- : separates owner and group.

- You can use -R to apply the change **recursively** to directories and their contents.

📄 **Example 1: Change File Owner Only**

chown john file.txt

🔹 *Changes the owner of file.txt to user john. Group remains unchanged.*

👥 **Example 2: Change File Owner and Group**

chown john:developers file.txt

🔹 *Changes the owner of file.txt to john and the group to developers.*

📂 **Example 3: Recursively Change Owner and Group of a Directory**

chown -R john:developers /home/john/

🔹 *Changes the owner and group of /home/john/ and all files/subdirectories inside it.*

Here's a clean and clear explanation of **special permissions** in Linux, with their meanings, symbolic notations, numeric values, and real-life examples:

🔐 **Special Permissions in Linux:**

Permission	Symbolic Notation	Numeric Value	What It Does	Real-Life Example
SUID (Set User ID)	u+s	4	Executes a file with the **owner's privileges**, not the user who runs it.	/usr/bin/passwd – lets normal users change their own password.
SGID (Set Group ID)	g+s	2	Executes a file with the **group's privileges** or sets group ownership for files in a directory.	/var/www – shared directory where all files inherit the same group.
Sticky Bit	o+t or +t	1	Applied to **directories**, allows only the file **owner** or **root** to delete or rename files.	/tmp – a public directory, but users can't delete others' files.

💡 **Example Commands:**

✅ **Set SUID on a file:**

chmod u+s filename

OR using numeric:

chmod 4755 filename

✅ **Set SGID on a file/directory:**

chmod g+s filename_or_directory

OR using numeric:

chmod 2755 directoryname

✅ **Set Sticky Bit on a directory:**

chmod +t directoryname

OR using numeric:

chmod 1777 /tmp

🎯 **Quick Reference Table (Numeric Form):**

Special Bit	Value	Total Permission Example
SUID	4	4755
SGID	2	2755
Sticky Bit	1	1777

Let me know if you want a small bash script to check or apply special permissions!

Sure! Here's a **complete explanation** of your situation along with the **correct process**, and how to **add multiple users at once** — all written in clean, professional Linux admin style:

🥷 **Why You Got a Permission Denied Error as a Normal User (Sai)**

When you tried this command:

[Sai@vbox ~]$ useradd NaturalLover

You received the error:

useradd: Permission denied.

useradd: cannot lock /etc/passwd; try again later.

🔒 This happens because **only the root user (or a user with sudo privileges)** can add new users to the system. Normal users like Sai **do not have the necessary permissions** to modify system files like /etc/passwd.

✅ **Solution: Use Root User or Sudo**

To fix this, you need to:

1. **Switch to the root user** using:
2. su -

Then enter the root password.

3. **Or** use sudo if Sai has sudo privileges:
4. sudo useradd NaturalLover

👨‍💻 **Root User Example**

[root@vbox ~]# useradd NaturalLover

[root@vbox ~]# id NaturalLover

uid=1028(NaturalLover) gid=1031(NaturalLover) groups=1031(NaturalLover)

🎉 User NaturalLover was successfully created by the **root user**.

➕ **How to Add Multiple Users at Once**

You can add multiple users using:

🔷 **Option 1: Add Users One by One in a Script or Command**

useradd user1

useradd user2

useradd user3

You can also combine them using &&:

useradd user1 && useradd user2 && useradd user3

🔷 **Option 2: Use a Loop with a List of Usernames**

for user in user1 user2 user3; do

  useradd "$user"

done

🔷 Option 3: Read from a File

If you have a file named users.txt with usernames like:

user1

user2

user3

You can run:

```
while read user; do
  useradd "$user"
done < users.txt
```

🔐 Sticky Bit in Linux

The **Sticky Bit** is a **special permission** used mostly on **directories**, not files.

When the **Sticky Bit is enabled** on a directory, **only the file owner** (or the root user) can **delete or rename** the files inside — **even if others have write permissions** on the directory.

📌 Example Use Case:

Imagine a shared folder like /shared where:

- The **owner** and their **friend** both have **write permissions**.
- But you want to **prevent the friend from deleting the owner's files**.

In that case, enable the **Sticky Bit**:

```
chmod +t /shared
```

Now, even if the friend can **write** in the folder, they **cannot delete or rename files** created by the owner.

Here is your content with corrected spelling, formatting, and clarity:

Umask: Brief Introduction

- **Full permission of a directory:** 777
- **Full permission of a file:** 666

Permission Type	Directory	File
Full Permission	777	666
Umask	022	022
Default	755	644

Example:

```
[root@vbox Files]# ls
d1  testfile1.txt

[root@vbox Files]# umask
0022

[root@vbox Files]# umask 042

[root@vbox Files]# umask
0042

[root@vbox Files]# ls
d1  testfile1.txt

[root@vbox Files]# ll
total 4
drwx-wxr-x. 2 root root  16 Apr 23 09:37 d1
-rw--w-r--. 1 root root 127 Apr 22 11:14 testfile1.txt
```

Brief Explanation Regarding the Concept of vim ~/.bashrc

The .bashrc file is a shell script that Bash runs whenever a user opens a new terminal session in interactive mode. It is commonly used to:

- Customize the shell environment (like setting aliases, environment variables).

- Set prompt styles.
- Automatically run commands at startup.

To edit the .bashrc file, use:

vim ~/.bashrc

After editing, apply changes using:

source ~/.bashrc

Calculation of Umask for Full Permission and Default Permissions

- **Full permissions:**
  - **Directory:** 777
  - **File:** 666
- **Umask:** Determines the default permission by subtracting its value from the full permission.

Calculation Example:

Type	Full Permission	Umask	Default Permission
Directory	777	022	755
File	666	022	644

Formula:
Default Permission = Full Permission - Umask

Controlling N Number of Servers – Monitoring and Managing Without Disturbance

To efficiently manage and monitor multiple servers, system administrators use configuration management and monitoring tools. These tools help in:

- Automating updates and deployments.
- Monitoring server health and logs.
- Managing configurations uniformly across all servers.

Popular Tools:

- **Ansible:** Agentless automation tool.
- **Nagios / Zabbix:** Monitoring tools for server performance.
- **Puppet / Chef / SaltStack:** Configuration management tools.

These tools ensure smooth operations, minimal downtime, and proactive issue resolution.

Process Lifecycle – Brief Introduction and Real-World Scenario

Lifecycle Stages:

1. **Created (New):** Process is created (forked).
2. **Running:** Actively executing instructions.
3. **Waiting (Blocked):** Waiting for resources (e.g., I/O).
4. **Terminated (Exit):** Process has completed or been killed.
5. **Zombie:** Finished execution but still in the process table.

Real-World Scenario:

Imagine a web server like Apache:

- When a user sends a request, a new process or thread is spawned.
- It runs, processes the request, and sends a response.
- Once the task is done, the process terminates.

This full cycle from request to response to termination follows the process lifecycle.

Zombie Process in Linux – Impact on System Performance

In Linux, **zombie processes** are processes that have completed execution but still remain in the **process table** because their parent process hasn't read their exit status using the wait() system call.

- **Why it causes lag:**
  Zombie processes **consume process table entries**, which are limited. If too many zombie processes accumulate, the system may run out of space in the process table, leading to:
    - System lag
    - Slow performance
    - In extreme cases, inability to create new processes
- **Solution:**
  Ensure the parent process is coded to correctly handle child process termination using proper wait() or waitpid() calls. If needed, restart the parent process or use kill to clean up.

First Process in Linux

- The **first process** that starts during the Linux boot process is:
- systemd
- **Process ID (PID):** 1
- **Role of systemd:**
    - Initializes the system

- Starts all necessary system services
- Manages the boot process and targets (runlevels)
- Becomes the ancestor of all other processes

System Monitoring in Linux

1. ps aux – Process Snapshot

The ps aux command provides a **snapshot of all running processes** on the system. It's commonly used for **monitoring and troubleshooting**.

Breakdown of ps aux:

- a – Shows processes for all users.
- u – Displays the user who owns the process.
- x – Lists processes not attached to a terminal.

Example:

ps aux

This displays detailed info like:

- User
- PID (Process ID)
- %CPU
- %MEM
- Time
- Command

Useful for identifying processes consuming high resources.

---

2. top Command – Real-Time Process Monitoring

The top command provides a **dynamic, real-time view** of system processes, CPU usage, memory usage, and more.

To use:

top

Key Features:

- Continuously updates the list of processes.

- Shows system load, uptime, number of users, and more.
- Use keys like P (sort by CPU), M (sort by memory), k (kill a process), and q (quit).

Example Output (partial):

top - 12:45:02 up 2:34, 1 user, load average: 0.29, 0.35, 0.32

Tasks: 159 total,   1 running, 158 sleeping,   0 stopped,   0 zombie

%Cpu(s): 3.1 us, 0.5 sy, 0.0 ni, 96.1 id, 0.2 wa, 0.0 hi, 0.1 si

MiB Mem :   7891.0 total,   1452.2 free,   2321.0 used,   4117.8 buff/cache

---

Devices Connected – Real-World Scenario (Bluetooth/Wi-Fi in Linux)

Real-World Scenario:

Imagine you are using a **Linux laptop** or **Raspberry Pi**, and you have:

- **Wi-Fi connected** to a router
- **Bluetooth connected** to a wireless speaker or smartphone

You may want to:

- See which devices are connected to your **Wi-Fi network**
- Check the **Bluetooth** devices your system is communicating with
- Verify the **IP address** of devices connected to your system or network

---

Tracking Bluetooth Devices in Linux

1. **Check Bluetooth Status:**
2. systemctl status bluetooth
3. **Scan for Devices:**
4. bluetoothctl
5. scan on
6. **Pairing Info:**
7. paired-devices
8. **Exit Tool:**
9. exit

This helps you track which Bluetooth devices are around or paired with your Linux system.

Tracking Wi-Fi Devices in Linux (Who is Connected to My Network)

To see all devices connected to your **Wi-Fi network**, log into your **router** (usually 192.168.0.1 or 192.168.1.1) OR use network scanning tools like:

1. **nmap (Network Mapper)**
   First, install it:
2. sudo apt install nmap

Then scan your network:

nmap -sn 192.168.1.0/24

This shows a list of devices connected to the network with their **IP and MAC addresses**.

How to Verify IP Address in Linux

1. **To check your own IP address:**
2. ip a

or simply:

ifconfig

3. **To view connected devices and their IPs (locally):**
4. arp -a

This command displays IP addresses and their associated MAC addresses that your system has communicated with recently.

Bonus Tool – netstat or ss:

Check who is connected and on which port:

netstat -tuln

or

ss -tuln

Great! Let's break down and **explain what each command and its output means** from your Linux terminal:

✅ **Command: w**

This command shows **who is logged in** and **what they are doing** on the system.

Example Output:

11:21:48 up 46 min, 2 users, load average: 0.04, 0.07, 0.08

USER   TTY   LOGIN@   IDLE   JCPU   PCPU WHAT

bhargavs seat0   10:36   0.00s 0.00s 0.00s /usr/libexec/gdm-wayland-session --regist

bhargavs tty2   10:36   46:15   0.14s 0.14s /usr/libexec/gnome-session-binary

Explanation of each part:

- **11:21:48** → Current system time.
- **up 46 min** → System has been running for 46 minutes since the last boot.
- **2 users** → Two user sessions are currently logged in (both by user bhargavs on different terminals).
- **load average: 0.04, 0.07, 0.08** → This shows the system load over the last **1, 5, and 15 minutes**.
    - These numbers represent the average number of processes waiting in the queue to run.
    - Lower values mean the system is under low load (which is good).

Columns in the w command:

- **USER** → The name of the user logged in.
- **TTY** → The terminal the user is connected to.
- **LOGIN@** → The time the user logged in.
- **IDLE** → Time since the user last interacted with the terminal.
- **JCPU** → Time used by all processes attached to the terminal.
- **PCPU** → Time used by the current process shown in the "WHAT" column.
- **WHAT** → The command the user is currently executing.

✅ **Command: uptime**

Output:

11:22:10 up 46 min, 2 users, load average: 0.03, 0.07, 0.08

This gives a quick summary of:

- **Current time**
- **System uptime (46 minutes)**
- **Number of users (2)**
- **Load average (1, 5, 15 minutes)**

🔍 **Summary:**

- The system is running fine with **low load**.
- User bhargavs is logged in on two sessions (seat0 and tty2).
- No heavy tasks are running; the CPU is mostly idle.
- The system has been on for **46 minutes** since the last reboot.

🧠 **1. Understanding w and uptime command**

[root@vbox ~]# w

[root@vbox ~]# uptime

Both commands show:

- **System uptime** (how long the system has been running).
- **Number of users** logged in.
- **Load average**:
- load average: 0.03, 0.07, 0.08

These are the **average number of processes in the run queue** over:

  - 1 minute
  - 5 minutes
  - 15 minutes

📈 **2. How to Interpret Load Average**

Formula:

Load Average is **not** CPU usage percentage. It shows **how many processes are waiting** to be scheduled.

- **CPU(s)**: From your lscpu, you have:
- CPU(s): 2

That means your system can run **2 tasks simultaneously**.

Now:

load average: 0.03, 0.07, 0.08

This means:

- Over the last 1, 5, and 15 minutes, on average, **less than 1 task** was waiting to be executed.
- Since your system has **2 CPUs**, anything below 2.0 is generally good.

Rule of Thumb:

- **Load < CPU cores** → system is underutilized (good).
- **Load = CPU cores** → fully utilized (okay).
- **Load > CPU cores** → overloaded (can cause slowdowns).

⚙️ 3. lscpu Breakdown

Shows detailed info about your CPU:

- **Model**: Intel i5-10310U
- **Cores**: 2
- **Threads per core**: 1 (so no Hyper-Threading here)
- **Sockets**: 1 (only 1 physical CPU)
- **Speed**: ~2.2 GHz (dynamic)

💾 4. /proc/cpuinfo

This file shows per-core CPU info.

- You see 2 entries: processor: 0 and processor: 1 → matches lscpu saying 2 cores.
- CPU flags tell features like support for virtualization, SIMD, etc.

🧮 5. How is Load Average Calculated?

Linux uses the **exponential moving average** of the length of the run queue (processes waiting or running) to calculate load.

Behind the scenes:

- Every 5 seconds, the OS checks how many processes are active.
- It then updates the 1, 5, and 15-minute averages using a smoothing algorithm.

💡 **Example:**

Your system has 2 CPUs, and load average is:

load average: 1.00, 1.00, 1.00

This means:

- You are **using the CPU efficiently**.
- If it was 3.00, 3.00, 3.00 → you are trying to run **more than 2 tasks at once**, and **1 task is always waiting**.

🧠 **Bonus: Memory info /proc/meminfo**

From this:

MemTotal: 3746468 kB

MemFree: 947524 kB

SwapTotal: 4141052 kB

SwapFree: 4141052 kB

- You have ~3.7 GB of RAM.
- ~947 MB is totally free.
- No swap is used yet → great (means RAM is sufficient)

✅ **Summary**

Term	Meaning
Load Average	Avg. number of processes waiting to run
Ideal Load	Should be ≤ number of CPU cores
lscpu	Shows core/thread/socket info
/proc/cpuinfo	Per-core detailed info
/proc/meminfo	RAM/swap usage, caching, buffers

🦴 What is a Daemon?

A **daemon** is a background process that runs continuously without user interaction. It typically starts during the boot process and continues to run until the system shuts down.

There are **two types of daemons**:

1. **Automatic Daemon**
    - Starts automatically at boot time.
    - Can also stop automatically when the system shuts down.
2. **Manual Daemon**
    - You start and stop it manually using commands like systemctl start <service> and systemctl stop <service>.

🧠 systemd and systemctl

- systemd: It is the system and service manager for Linux.
- systemctl: It is the command-line tool used to manage systemd.

To view available unit types:

systemctl -t help

Output includes:

- service – For daemons
- mount – For mount points
- swap – For swap space
- socket – For socket-activated services
- timer – For scheduled tasks
- etc.

🛠 Common Commands

Action	Command Example
Start a service	systemctl start sshd
Stop a service	systemctl stop sshd
Enable service at boot	systemctl enable sshd

Action	Command Example
Disable service at boot	systemctl disable sshd
Status of a service	systemctl status sshd
Restart a service	systemctl restart sshd

❌ **Error Fixing in Your Commands**

You had a typo in the command:

dnf intall httpd* -y

✅ **Correct command:**

dnf install httpd* -y

⚠️ **Error: No Enabled Repositories**

Your system shows:

Error: There are no enabled repositories in "/etc/yum.repos.d", ...

This happens because:

- Your system is **not registered** with a subscription manager (like Red Hat).
- You don’t have proper repositories set up in your .repo files.

✅ Fix (if you're on RHEL):

subscription-manager register

subscription-manager attach --auto

subscription-manager repos --enable=<repo-name>

Or, for local setup:

- Create .repo files manually or use CentOS/AlmaLinux repos if you're not using an official RHEL license.

✅ **Your sshd.service Output**

Your SSH daemon (sshd) is working correctly. It's active and running:

● sshd.service - OpenSSH server daemon

Active: active (running) since Wed 2025-04-23 ...

This means you can SSH into your system.

🔁 **Enable/Disable Services**

To **enable** a service (start it automatically at boot):

systemctl enable sshd

To **disable** a service (manually start each time):

systemctl disable sshd

Here’s a rewritten version of your statement for clarity:

Differences Between Restart and Reboot:

- **systemctl reload httpd**: This command reloads the configuration files of the HTTPD (Apache) service without stopping it. It's used when you want to apply changes to configuration files while keeping the service running.
- **systemctl restart httpd**: This command stops and then restarts the HTTPD service. It's used when you want to completely restart the service, which can help resolve certain issues that a simple reload won't fix.
- **systemctl reload-or-restart httpd**: This command is a combination of both reload and restart. If the service supports reloading, it will reload; if not, it will restart the service. This command is useful if you're unsure whether reloading is sufficient and want to try reloading first, with a fallback to restarting if necessary.

---

systemctl mask Command - Example:

The systemctl mask command is used to completely disable a service, preventing it from being started manually or automatically. It's like blocking the service from running entirely.

For example, consider a payment app like **PhonePe** that integrates with multiple UPI apps. If you want to prevent one specific UPI app from being used for transactions (similar to uninstalling or restricting its use), you would mask the service of that particular app. By masking it, you ensure that no one can start that service (or app) until it is unmasked.

Sure! Here's a rewritten and clearer version of your SSH introduction with a real-life example:

Introduction to SSH (Secure Shell)

SSH (Secure Shell) is a protocol used to securely connect to remote computers or servers over a network. It ensures that the communication between the client and server is encrypted, protecting data from unauthorized access or tampering.

Real-life Example:

Imagine you're using a **Wi-Fi network** in a public place, like a café, and you want to connect from your **mobile device or laptop** to another computer (such as a personal server or cloud machine). If you simply use an unsecured method (like plain Telnet or FTP), anyone on the same network can potentially intercept your data — including your passwords or private files.

But when you use **SSH**, your entire session — including commands, passwords, and data — is encrypted. It's like creating a **private, secure tunnel** between your device and the remote system, even over a public or insecure Wi-Fi network.

So, just like you wouldn't want someone looking over your shoulder while typing a password, SSH makes sure that even if someone is "watching" the network, they won't be able to see your data.

Sure! Here's a rewritten and polished version of the "Prerequisites for SSH Connection" section:

Prerequisites for Establishing an SSH Connection

Before you can successfully connect to a remote system using SSH (Secure Shell), you need to ensure the following requirements are met:

1. **OpenSSH Package Installed**
    - The **OpenSSH** package must be installed and running on both the **client** (your local machine) and the **server** (remote machine you want to connect to).
2. **Remote Username and Password**
    - You need valid **login credentials** (i.e., the **username** and **password**) for the remote system.
    - If password authentication is disabled, you may require a **private SSH key** instead.
3. **Remote Hostname or IP Address**
    - You must know the **hostname** or **IP address** of the remote machine you want to connect to.
    - Example: 192.168.1.10 or server.example.com
4. **SSH Port (Default: 22)**
    - By default, SSH communicates over **port 22**.
    - If the server is configured to use a different port, you must specify it during the connection (e.g., ssh -p 2222 username@host).

Sure! Here's a clearer and more professional version of your Red Hat lab exercise:

Red Hat Lab Exercise: Connecting to a Remote Host via SSH

Objective:
Learn how to connect from your **Workstation host** to another host (e.g., servera) using the SSH (Secure Shell) protocol.

Step-by-Step Instructions:

1. **Open the terminal** on your **Workstation** machine.
2. **Use the following syntax to connect:**
3. ssh remoteusername@remotehostname_or_IP
    - remoteusername: The username of the remote system (e.g., root)
    - remotehostname_or_IP: The hostname (e.g., servera) or IP address of the remote machine

---

Example Command:

ssh root@servera

This command initiates a secure connection to the host servera as the root user.

---

workstation : ssh root@servera

servera : vim /etc/ssh/sshd_config

servera : systemctl restart servera

then go to server and you can login to servera

if facing any error like hostkey

ssh-keygen -R serverb

servera : systemctl restart servera → systemctl restart sshd

Note:

- Ensure that the **SSH service is running** on the remote machine.

- Port 22 should be open and reachable between both machines.
- If it's your first time connecting, you'll be prompted to accept the remote host's key.

Here's a well-structured version of your content, including a real-world example to explain both **symmetric** and **asymmetric encryption** methods:

🔐 **Encryption Methods in Cybersecurity**

1. Symmetric Encryption

- **Key Concept**: Uses **one single key** for both encryption and decryption.
- **Key Type**: Only a **private key** is used and shared between sender and receiver.
- **Example**:
    - Imagine you lock a box with a key and give that **same key** to your friend. Both of you can lock and unlock the box using the **same key**.
- **Real-world Usage**: Data encryption within closed systems, such as local file encryption.

✅ **Real-life Example:**

Suppose you and your friend both know the **password** to unlock a payment app (like PhonePe). If you give your friend that password, they can open the app and access it.
Here, **the same secret (password)** is shared — similar to a **symmetric key**.

---

2. Asymmetric Encryption

- **Key Concept**: Uses **two keys** – one **public key** for encryption and one **private key** for decryption.
- **Key Types**:
    - **Public Key**: Can be shared openly with anyone.
    - **Private Key**: Kept secret by the receiver.
- **Example**:
    - You lock a box using a **public key** (everyone has access to it), but **only the person with the private key** can unlock and read the contents.
- **Real-world Usage**: Secure communication over the internet, like SSL/TLS for websites and SSH for remote login.

✅ **Real-life Example:**

In a UPI app like PhonePe:

- You may share the **application password** with someone (like a public key).
- But only **you** know the **UPI PIN** (like a private key), which is required to complete a transaction.
- So even if someone can access the app, **they can't send money without your UPI PIN**.
  This illustrates **asymmetric encryption**, where access is public but actions are restricted to a private key.

Great! Let's break down these two concepts **before** diving into the Wi-Fi connection explanation, as they relate to how systems (including Linux) handle **secure authentication**, especially in remote or automated environments.

🔐 1. Passwordless Authentication

✅ **What it means:**

Passwordless authentication is a secure login method where users authenticate without entering a password. Instead, they use:

- SSH keys (public/private key pair)
- Biometrics (fingerprint/face ID)
- OTPs or authenticator apps
- Hardware tokens (like YubiKey)

💡 **Real-life example:**

You want to connect from your **laptop to a Linux server** using SSH:

- You generate a key pair on your laptop:
- ssh-keygen
- Then copy your public key to the server:
- ssh-copy-id user@server
- Now, whenever you ssh user@server, **no password is needed**.

This is common in cloud servers and DevOps environments for **secure automation**.

🧼 2. Password Cleanliness Authentication (Likely Misunderstood Term)

It sounds like you're referring to **"Password Hygiene"** or **clean practices in using passwords**, since **"password cleanliness authentication"** isn't a standard term.

✅ **What it implies:**

Password cleanliness (or hygiene) refers to the **proper handling and use of passwords** to keep systems secure.

🧹 **Best Practices Include:**

- Never reuse passwords across platforms
- Use complex passwords (mix of upper, lower, numbers, symbols)
- Store passwords securely (not in plain text or sticky notes)
- Change default passwords on devices (especially routers/Wi-Fi)
- Use password managers instead of remembering passwords

🔐 **Real-world scenario:**

Let's say you're setting up Wi-Fi or SSH access on a Linux machine:

- **Bad password hygiene:** Using the default password 123456 for your router or sharing your root password with friends.
- **Good password hygiene:** Changing router passwords to a secure one and using key-based login for SSH instead of plain passwords.

📌 **In Summary:**

Concept	Meaning
Passwordless Authentication	Authentication without passwords, using keys or biometrics
Password Hygiene	Proper creation, use, and management of passwords for better security

---

Conclusion:

This document , explores various **Linux security concepts** and **administrative practices**, including:

- File and directory permissions
- Special permissions (SUID, SGID, Sticky Bit)
- Ownership (chmod, chown)
- Umask and default permission calculations
- User and group management
- SSH configuration and real-life use cases
- Process lifecycle and system monitoring tools (ps, top, w, uptime)
- Daemon management and systemctl usage

- Symmetric vs asymmetric encryption
- Concepts of passwordless authentication and password hygiene

DAY 4: System Administration Essentials: File Management, Compression, and Package Management

Steps to Configure SSH Authentication Between servera and serverb

1. **Open the Red Hat Lab online environment.**
2. **Access the Workstation Console.**
3. **Connect to the servera host.**

🔒 **Editing SSH Configuration for Password Authentication**

Before proceeding, ensure you have **execute permission** to run commands and edit files. If not, use sudo before each command.

Step-by-step:

On servera:

1. Open the SSH configuration file:
2. vim /etc/ssh/sshd_config
3. Go to **line number 40** (you can use :40 in vim to jump to it).
4. Modify the following line:
5. #PasswordAuthentication no

to:

PasswordAuthentication yes

   - Press i to enter **insert mode**.
   - Remove the # to uncomment.
   - Change prohibit-password or no to yes.

6. Save and exit vim:
   - Press Esc, then type :wq and press Enter.

7. Reload the SSH service to apply changes:
8. systemctl reload sshd

On serverb:

Repeat the same process on serverb:

1. Connect to serverb.
2. Open the SSH config:
3. vim /etc/ssh/sshd_config
4. Go to **line number 40**, and **edit** the same line as done on servera.
5. Save the file and reload SSH:
6. systemctl reload sshd

✅ **Explanation**:

- Changing PasswordAuthentication to yes allows users to log in using passwords.
- If you want to **restrict login**, you can set the value to no.

Perfect! Here's a clean and detailed explanation of your next block, with corrections and formatting for better understanding:

Linux Logging Architecture Overview

In Linux, the entire **login and logging architecture** is managed primarily by **two key components**:

🔧 **1. systemd-journald**

- Acts as the **heart of the logging architecture**.
- Responsible for collecting and managing **runtime logs**.
- Stores logs **temporarily** in memory under:
- /run/log

📦 **2. rsyslog (using Syslog protocol)**

- Works with **syslog protocol** to forward and store logs **persistently**.
- Takes the logs from systemd-journald and stores them permanently in:
- /var/log

🔄 Real-World Example: System Boot Process

Let's break it down:

1. When you **start your system or laptop**, the OS begins the **booting process**.
2. During boot:
    - Many **background services** start automatically (like networking, time sync, ssh, etc.).
    - These services generate **log messages** about their status and actions.
3. These log messages are first collected by:
4. systemd-journald → /run/log
    - This location is **temporary** and used only for the **current runtime session**.
5. To make these logs **persistent** and available after reboot, they are passed to:
6. rsyslog (Syslog protocol) → /var/log
    - This stores logs in files like /var/log/messages, /var/log/secure, etc.

⚠️ Common Problems Without rsyslog

1. ❌ **You cannot read past logs** after a reboot — because /run/log is volatile.
2. ❌ **No persistent storage** of logs — important logs are lost after shutdown.

✅ That's why **rsyslog** is crucial — it makes logs **survive reboots** and allows for proper **log auditing and system monitoring**.

Absolutely! Here's a practical **hands-on list of essential Linux commands** used for checking **log files**, **runtime logs**, and related information — especially useful when managing or debugging login/logging architectures:

🛠️ Log File and Runtime Log Checking Commands

🔍 1. View Logs with journalctl (systemd-journald logs)

Command	Purpose
journalctl	View all logs collected by systemd-journald
journalctl -xe	View logs with priority and error context
journalctl -b	Show logs from the current boot

Command	Purpose
journalctl -k	View kernel logs only
journalctl --since "1 hour ago"	View logs from the past 1 hour
journalctl -u sshd	View logs related to a specific service (e.g., sshd)
journalctl --disk-usage	Check the size of the journal logs on disk
journalctl --vacuum-time=2d	Delete logs older than 2 days

---

2. View Traditional Syslog Files (rsyslog logs)

File/Command	Purpose
cd /var/log	Navigate to the log directory
ls -l /var/log	List all log files
cat /var/log/messages	View general messages and system logs
cat /var/log/secure	View authentication and authorization logs
cat /var/log/dmesg	Boot-time kernel logs
tail -f /var/log/messages	Live monitoring of new log entries
less /var/log/messages	Scrollable view for easier reading

---

3. Check Service Status and Errors

Command	Purpose
systemctl status	General status of systemd
systemctl status sshd	View sshd service status
systemctl restart sshd	Restart ssh service
systemctl reload sshd	Reload ssh configuration (used after editing /etc/ssh/sshd_config)

---

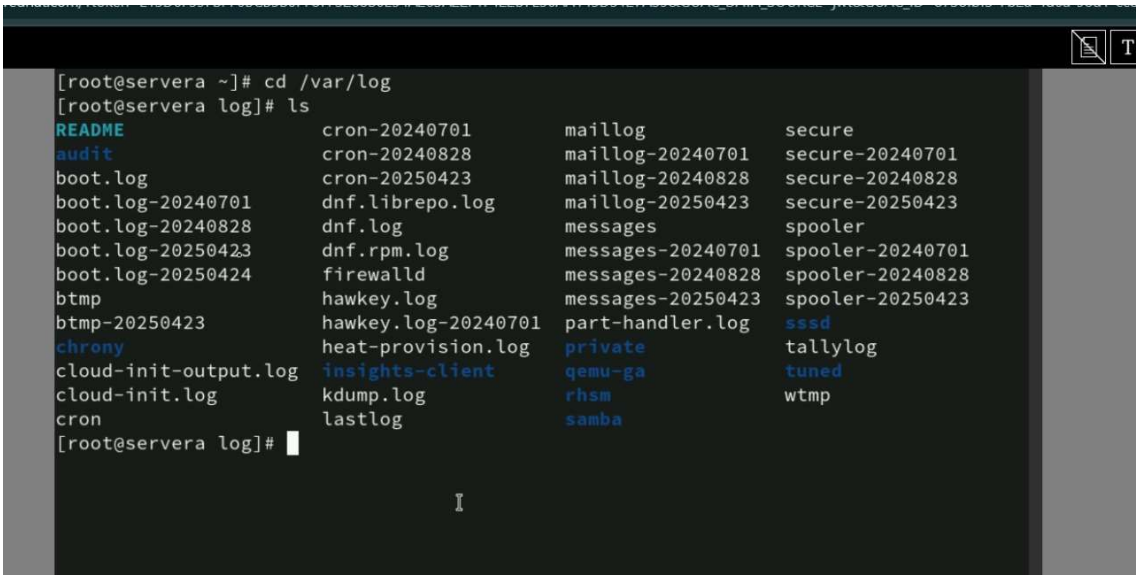
4. Check /run directory (temporary runtime files)

Command	Purpose
ls /run	View contents of the runtime directory
ls /run/log	Check runtime log storage (if journald writes here)
df -h /run	Check usage of the /run directory

---

5. Clear/Manage Log Storage (Advanced)

Command	Purpose
logrotate	Log management tool (auto rotates large logs)
journalctl --rotate	Force rotation of journal logs
journalctl --vacuum-size=100M	Clear logs if journal exceeds 100MB
journalctl --vacuum-time=7d	Keep logs from last 7 days only



Great question! Let's break it down simply and clearly:

How Logs are Stored in Linux (Syslog Architecture)

Logs in Linux are stored based on **two main properties**:

1️⃣ Facility

What is Facility?
It's **where** the log is coming from — the system component or application generating the message.

Facility	Source Description
auth or authpriv	Authentication logs (e.g., login, sudo)
cron	Logs from scheduled jobs (crontab)
daemon	Background services (e.g., system daemons)
kern	Kernel messages
user	General user-level messages
mail	Mail system logs
local0 to local7	Reserved for custom applications

📝 **Example:**

- Logs from sshd (SSH Daemon) → use **auth** facility.
- Logs from cron job → use **cron** facility.

2️⃣ Priority (Severity Level)

What is Priority?
It's **how important or urgent** the log message is — its severity.

Priority Level	Description	Example
emerg (0)	System is unusable	Disk failure
alert (1)	Action must be taken immediately	Database down
crit (2)	Critical conditions	Application crashed
err (3)	Error conditions	Failed service
warning (4)	Warning	High memory usage
notice (5)	Normal but significant	Configuration change
info (6)	Informational	Service started
debug (7)	Debug-level messages	Developer logs

📝 **Example:**

- crit means you must **immediately act** — like fixing a stopped service.
- info just tells you what happened, no urgent action.

🔗 **Combined Format**

Logs are categorized like this:

<facility>.<priority> action

Example in /etc/rsyslog.conf:

authpriv.* /var/log/secure

cron.* /var/log/cron

*.crit /var/log/critical.log

👉 This means:

- All messages from authpriv go to /var/log/secure
- All cron messages go to /var/log/cron
- Any message with crit priority (from **any** facility) goes to /var/log/critical.log

---

Perfect! You're on the right track with journalctl. It's the command used to **view logs managed by systemd-journald**.

Let's go step by step with a **realistic example output** and explain each field clearly.

✅ **Example journalctl Output:**

$ journalctl -xe

You might see something like:

Apr 24 09:31:45 serverb sshd[1305]: Accepted password for bhargav from 192.168.1.101 port 54432 ssh2

🧩 **Breakdown of Fields:**

Field #	Field Name	Example	Explanation
1️⃣	**Timestamp**	Apr 24 09:31:45	Date and time when the log entry was recorded
2️⃣	**Hostname**	serverb	Name of the machine/system (your host)

Field #	Field Name	Example	Explanation
3️⃣	**Process (PID)**	sshd[1305]	The name of the service (e.g., sshd) and its process ID (here, 1305)
4️⃣	**Message**	Accepted password for bhargav from 192.168.1.101 port 54432 ssh2	Actual log content / event description

---

🛠️ **Useful journalctl Command Variations**

Here are a few extra examples for hands-on experience:

🔹 **See all logs**

journalctl

🔹 **Follow logs in real time (like tail -f)**

journalctl -f

🔹 **Show only logs for sshd service**

journalctl -u sshd

🔹 **Show logs from today**

journalctl --since=today

🔹 **Show logs by a specific priority (e.g., critical)**

journalctl -p crit

Perfect! You're understanding the core of how **systemd journaling** works. Let me help you structure and simplify everything you've written about journald.conf for better clarity and hands-on use.

📁 **Configuration File**

vim /etc/systemd/journald.conf

🔍 **Key Parameter: Storage**

- Located near **line 18** of the config file.
- It controls how and **where log data is stored**.

✏️ **To Modify:**

1. Open the file:
2. sudo vim /etc/systemd/journald.conf
3. Locate the line:
4. #Storage=auto
5. Press i to go into insert mode.
6. Remove the # and set your preferred option:
    - For **permanent log storage**:
    - Storage=persistent
    - For **temporary log storage** (cleared on reboot):
    - Storage=volatile
    - For **automatic behavior**:
    - Storage=auto
        - If /var/log/journal/ exists → behaves as **persistent**
        - If not → behaves as **volatile**
7. Press Esc, then type :wq to save and exit.

🔄 **Restart the journald service:**

sudo systemctl restart systemd-journald

This applies your new storage setting immediately.

✅ **Summary Table:**

Storage Option	Behavior
persistent	Stores logs **permanently** in /var/log/journal/
volatile	Stores logs **temporarily** in /run/log/ (lost after reboot)
auto	Chooses persistent if the folder exists, else volatile

That's an excellent and practical explanation of how **time synchronization** works in Linux using **Chrony**! Here's your real-world example cleaned up, well-structured, and easy to understand — ready for reports, presentations, or quick revision.

⏱️ **Real-World Example: Time Synchronization Between Server and Clients (Using Chrony)**

🧠 **Scenario:**

You have a **main server** and multiple **client systems** located at different geographical locations. Your goal is to **synchronize the time of all clients** with the **main server's time**.

✅ **Step-by-Step Procedure**

🔷 **Step 1: Check the Chrony Service Status**

systemctl status chronyd

- chronyd is the **daemon** (service) that runs Chrony.
- If it's not active, enable it with:
- sudo systemctl start chronyd
- sudo systemctl enable chronyd

🔷 **Step 2: Check Time Synchronization Status**

timedatectl

If NTP service shows **inactive**, enable it:

timedatectl set-ntp true

This ensures time synchronization is managed by NTP.

🔷 **Step 3: Configure the Chrony Client**

Open the configuration file:

sudo vim /etc/chrony.conf

Inside this file:

- Add the **main server's hostname or IP address** followed by iburst, for faster initial synchronization:

server classroom.example.com iburst

The keyword iburst allows quicker communication with the time server.

🔷 **Step 4: Restart the Chrony Service**

sudo systemctl restart chronyd

🔷 Step 5: Verify Chrony Sources

chronyc sources -v

This command shows the list of **time sources** your system is syncing with and their status.

🔷 Step 6: Final Verification

timedatectl

Check if:

- **NTP service** is active
- **Time is synchronized**
- Time is the same on **both client and server machines**

🧪 Use Case Example:

In a **banking system**, where servers across various branches need to log transactions at exactly the same time. Even a small time mismatch can cause **data inconsistency** or **security concerns**. That's why proper time sync is critical.

Absolutely! You're spot on ✅ — the timedatectl command is powerful for managing system **date, time, and timezone settings** in Linux systems.

Here's a clearer breakdown for your notes or revision:

🛠️ timedatectl – Manage Date, Time, and Timezone

1️⃣ To Set the System Time

sudo timedatectl set-time "YYYY-MM-DD HH:MM:SS"

📝 **Example:**

sudo timedatectl set-time "2025-04-24 15:30:00"

This will manually change the system time and date.

2️⃣ To Change the Timezone

sudo timedatectl set-timezone <Region/City>

📝 **Example:**

sudo timedatectl set-timezone Asia/Kolkata

You can list all available time zones using:

timedatectl list-timezones

✅ **Bonus Tip:**

Always make sure NTP is off before setting the time manually, otherwise it might auto-correct itself:

sudo timedatectl set-ntp false

Archieve:

Perfect explanation! You've covered the archiving concept and command flow in Linux **very clearly**. Here's a nicely formatted recap with clean steps and examples so you can use it for revision, notes, or sharing:

📦 **Linux Archiving with tar**

✅ **What is Archiving?**

Archiving is the process of **grouping multiple files into one** using the tar command. This is useful for:

- Backup
- File transfers
- Storage optimization

🔧 **tar Command Syntax**

tar [options] destination_filename source_files

Option	Meaning
c	Create archive
f	Use archive file
t	List files inside
x	Extract archive
v	Verbose (show steps)

📂 **Practical Example**

1️⃣ Create Files and Archive

mkdir archiving

cd archiving

touch file{1..5}.txt

tar cf archive.tar file{1..5}.txt

➡️ Creates archive.tar containing 5 text files.

2️⃣ List Files in Archive

tar tf archive.tar

➡️ Shows:

file1.txt

file2.txt

file3.txt

file4.txt

file5.txt

3️⃣ Move Archive to Another Directory

mkdir test

mv archive.tar test/

4️⃣ Extract Archive

cd test

tar xf archive.tar

ls

➡️ Output:

archive.tar file1.txt file2.txt file3.txt file4.txt file5.txt

---

🧠 **Notes:**

- The archive **doesn't compress** by default, it's just a bundle. For compression, use:

- .tar.gz with tar czf
- .tar.bz2 with tar cjf

Concept: compressing the files,

Great! You're on the right track about compression and archiving. Here's a clean and **well-organized guide** on compressing files using tar along with **real command-line examples** and how to **check sizes** using du -sh.

🗜️ Concept: Compressing Files in Linux

📦 Archive + Compression

To compress files, we **combine** tar with a compression method:

Compression	Command Option	Output Extension	Notes
gzip	z	.tar.gz	Fast, common compression
bzip2	j	.tar.bz2	Slower, but better compression
xz	J	.tar.xz	Best compression rate

🧪 Example: Hands-on with Compression Types

🛠️ Step-by-step:

```
# 1. Create a directory and sample files
mkdir compress_demo
cd compress_demo
touch file{1..5}.txt
```

📏 Check original size:

```
du -sh .
```

🌀 1. gzip Compression

```
tar cvfz files.tar.gz file{1..5}.txt
du -sh files.tar.gz
```

🦋 2. bzip2 Compression

tar cvfj files.tar.bz2 file{1..5}.txt

du -sh files.tar.bz2

❄ **3. xz Compression (Best compression ratio)**

tar cvfJ files.tar.xz file{1..5}.txt

du -sh files.tar.xz

📊 **Compare Sizes**

You can now compare:

ls -lh files.tar.*

➡ You'll clearly see that:

- .tar is uncompressed
- .tar.gz is compressed
- .tar.bz2 is better compressed
- .tar.xz is **best compressed**

💡 **Notes:**

- Use du -sh filename to check any file or folder size in human-readable format.
- If you're backing up or sharing large files, prefer .tar.xz for best compression.

The scp (secure copy) command is used to securely transfer files between hosts over a network. It uses SSH for data transfer and provides the same authentication and security as SSH.

📜 **Syntax for scp Command**

scp <source_file> <user>@<destination_host>:<destination_path>

- <source_file>: The file you want to copy from the local machine.
- <user>: The username of the destination machine.
- <destination_host>: The IP address or hostname of the destination machine.
- <destination_path>: The path on the destination machine where you want to copy the file.

🧪 **Example Usage:**

1. Copy a file from local to remote:

scp file1.txt user@192.168.1.100:/home/user/remote_folder/

This copies file1.txt from the local machine to the /home/user/remote_folder/ directory on the remote machine at IP 192.168.1.100.

2. Copy a file from remote to local:

scp user@192.168.1.100:/home/user/remote_folder/file1.txt /local/folder/

This copies file1.txt from the remote machine to the /local/folder/ on the local machine.

3. Copy an entire directory recursively:

scp -r /local/directory/ user@192.168.1.100:/home/user/remote_folder/

The -r option is used to recursively copy an entire directory.

4. Copy files between two remote hosts:

scp user1@192.168.1.100:/path/to/file user2@192.168.1.101:/path/to/destination

This copies a file from one remote server to another remote server directly.

⚙ Additional Options:

- **-v**: Enable verbose mode to display detailed information about the transfer.
- **-C**: Enable compression to speed up the transfer (useful for large files).
- **-P**: Specify a different port (useful if the SSH server is running on a non-standard port).

Example with verbose and compression:

scp -v -C file1.txt user@192.168.1.100:/home/user/

Sfte: secure file transfer program:-

SFTP: Secure File Transfer Protocol

SFTP stands for **Secure File Transfer Protocol**, and it is a secure version of the traditional File Transfer Protocol (FTP). Unlike FTP, which sends data in plain text, SFTP encrypts both commands and data during the transfer, ensuring confidentiality and security. SFTP runs over an SSH (Secure Shell) connection, providing a higher level of security for transferring files over potentially unsecured networks (like the internet).

Key Features of SFTP:

1. **Encryption**: SFTP encrypts all traffic, including passwords and files, making it a secure method for transferring sensitive data.
2. **Authentication**: It uses SSH for authentication, allowing you to connect to remote servers securely.

3. **Data Integrity**: It ensures that the data transferred is not tampered with, maintaining its integrity.
4. **File Management**: SFTP not only transfers files but also allows you to manage files on remote systems. This includes renaming, deleting, and navigating directories.

🌍 **Real-World Example:**

Imagine you're a system administrator for a company, and you need to securely transfer a set of confidential financial reports from your local machine to the company's server. Using FTP could expose the sensitive data to potential attackers, but with SFTP, you ensure that the entire process is encrypted, protecting both the files and the credentials used for authentication.

For example, transferring customer data, financial statements, or software updates in a corporate environment using SFTP helps prevent data breaches or unauthorized access.

💻 **SFTP Command Example:**

To use SFTP, you'll typically start by connecting to a remote server using the following syntax:

sftp username@hostname

Example:

sftp user@192.168.1.100

This command connects to the remote machine 192.168.1.100 using the user account. You will then be prompted to enter the password for the user account. After authenticating, you can use SFTP commands to transfer files.

Common SFTP Commands:

1. **put**: Upload a file from local to remote.
2. sftp> put localfile.txt /remote/directory/

This uploads localfile.txt to /remote/directory/ on the remote server.

3. **get**: Download a file from remote to local.
4. sftp> get remotefile.txt /local/directory/

This downloads remotefile.txt from the remote server to /local/directory/ on your machine.

5. **ls**: List files in the current remote directory.
6. sftp> ls
7. **cd**: Change the directory on the remote server.
8. sftp> cd /path/to/directory
9. **exit**: Disconnect from the remote server.

10. sftp> exit

⚙ Example of a Secure File Transfer with SFTP:

Let's say you want to upload a file report.pdf to a remote server at IP 192.168.1.200 under the /home/user/documents/ directory:

1. Open the terminal and initiate the SFTP connection:
2. sftp user@192.168.1.200
3. Once connected, upload the file:
4. sftp> put report.pdf /home/user/documents/
5. After the transfer is complete, you can exit the SFTP session:
6. sftp> exit

This ensures the file report.pdf is transferred securely to the remote server without exposing sensitive data during the process.

SFTP Command Example: Using put and get

Let's walk through an example using **SFTP** to **upload** and **download** files securely from a remote server.

---

1. Upload a File Using put Command:

The **put** command is used to upload a file from your **local machine** to a **remote server**.

Steps:

1. **Connect to the remote server**: Open your terminal and use the following command to connect to the remote server:
2. sftp user@192.168.1.100

This connects you to the server at IP 192.168.1.100 using the user account. You'll be prompted to enter the password for the user account.

3. **Upload a file using put**: Suppose you have a local file named report.txt that you want to upload to the remote server. You can use the put command:
4. sftp> put report.txt /home/user/documents/

This command uploads the file report.txt from your local machine to the /home/user/documents/ directory on the remote server.

5. **Verify the upload**: Once the upload is complete, you can list the files in the remote directory to verify:
6. sftp> ls /home/user/documents/

This should show report.txt in the remote directory.

7. **Exit SFTP session**: After the file has been uploaded, you can exit the SFTP session:
8. sftp> exit

2. Download a File Using get Command:

The **get** command is used to **download** a file from a **remote server** to your **local machine**.

Steps:

1. **Connect to the remote server again** (if not already connected):
2. sftp user@192.168.1.100
3. **Download a file using get**: Suppose there is a file named data.txt on the remote server, located at /home/user/documents/, and you want to download it to your local machine. You can use the get command:
4. sftp> get /home/user/documents/data.txt

This command downloads the file data.txt from the remote server to your current working directory on the local machine.

5. **Verify the download**: After the download is complete, you can check the local directory to verify:
6. ls

This should show data.txt in the local directory.

7. **Exit SFTP session**: Once you're done, you can exit the SFTP session:
8. sftp> exit

Summary:

- **put** uploads a file from your local machine to the remote server.
- **get** downloads a file from the remote server to your local machine.

Example Commands Recap:

1. **Upload** a file to the remote server:
2. sftp> put localfile.txt /remote/directory/
3. **Download** a file from the remote server:
4. sftp> get /remote/directory/remotefile.txt

This is a simple demonstration of how to transfer files securely between a local machine and a remote server using SFTP's put and get commands.

.

RSYNC:-

Rsync: Overview

Rsync is a fast and versatile file-copying tool used in Linux and Unix systems. It allows you to efficiently transfer and synchronize files between two locations, whether those locations are on the same machine or across a network. Rsync is particularly efficient because it only transfers the differences between the source and the destination files, rather than copying entire files every time.

Key Features of Rsync:

1. **Delta Transfer Algorithm**: It only transfers the changed parts of files (i.e., blocks that differ), minimizing bandwidth usage.
2. **Incremental Backups**: It can be used for creating backups by copying only new or modified files.
3. **Compression**: It supports compression during file transfer, reducing network usage.
4. **Synchronization**: Rsync ensures the source and destination directories are synchronized by comparing the files.
5. **Remote Synchronization**: It can be used to synchronize files across different machines over SSH or other protocols.

Basic Syntax:

rsync [options] source destination

- **source**: The path to the file or directory you want to copy.
- **destination**: The location where you want to copy the file or directory.
- **Options**: Control the behavior of the rsync command (e.g., -r, -a, -v, -z).

Common Rsync Options:

- -a: Archive mode (preserves symbolic links, permissions, timestamps, and recursive copying).
- -v: Verbose mode (provides detailed output).
- -z: Compress file data during transfer.

- -r: Recursive mode (to copy directories and their contents).
- -e: Specify remote shell (e.g., SSH).
- --delete: Deletes files in the destination directory that no longer exist in the source.

Real World Scenario Example:

Let's consider an example where you want to back up a directory from your local machine to a remote server. You want to ensure that only the new or modified files are transferred, and you want the transfer to be compressed to save bandwidth.

Scenario: Backing up a local directory to a remote server

- **Local Directory**: /home/user/data
- **Remote Server**: remote.example.com
- **Remote Directory**: /home/user/backup

Rsync Command Example:

rsync -avz /home/user/data/ user@remote.example.com:/home/user/backup/

Explanation:

- **-a**: Archive mode (preserves file permissions, timestamps, symbolic links, etc.).
- **-v**: Verbose mode (prints detailed information about the files being transferred).
- **-z**: Compresses the files during the transfer to reduce network bandwidth usage.
- **/home/user/data/**: The source directory on the local machine.
- **user@remote.example.com:/home/user/backup/**: The destination directory on the remote server.

Real-World Usage Scenarios:

1. **Backup Strategy**:
    - Many system administrators use rsync for creating backups of important directories or servers. With rsync, backups are incremental, meaning only changes are transferred, which makes the backup process much faster and reduces the required storage space.
2. **Website Synchronization**:
    - If you manage a website that has multiple environments (e.g., development, staging, production), rsync can be used to synchronize files across different servers, ensuring that the latest version of the website is available on each.
3. **Remote File Synchronization**:

- Rsync can be used to synchronize files between a local machine and a remote server. This is particularly useful for situations where you need to make sure files are always up-to-date across different locations (e.g., syncing user data to a cloud server).

Example of Rsync with Remote Host:

Let's assume you want to synchronize the contents of the local directory /home/user/docs with the remote directory /home/user/docs_backup on a remote server.

rsync -avz /home/user/docs/ user@remote.example.com:/home/user/docs_backup/

Explanation:

- **-a**: Archive mode for preserving file structure and attributes.
- **-v**: Verbose mode to get more information.
- **-z**: Compression to minimize data transfer time.
- **/home/user/docs/**: Local directory you want to sync.
- **user@remote.example.com:/home/user/docs_backup/**: Remote server and destination directory.

Example of Rsync with SSH:

If you want to ensure the transfer is secure, you can use the -e option to specify SSH as the remote shell.

rsync -avz -e ssh /home/user/docs/ user@remote.example.com:/home/user/docs_backup/

This ensures the data is securely transferred over SSH.

Summary:

Rsync is a powerful tool for syncing files, creating backups, and transferring data efficiently. Its ability to only transfer changed files and its support for compression makes it ideal for remote backups, website synchronization, and large file transfers. Whether you're working on local directories or remote servers, rsync is a versatile solution that saves time and bandwidth.

Red hat package manager

Red Hat Package Manager (RPM): Overview

The **Red Hat Package Manager (RPM)** is a powerful package management system used by Red Hat-based Linux distributions such as **RHEL (Red Hat Enterprise Linux)**, **CentOS**, **Fedora**, and **Oracle Linux**. RPM is used to manage the installation, removal, upgrading, and querying of software packages. It helps automate the installation of software, resolve dependencies,

and maintain the integrity of the system by ensuring that installed packages are correct and up-to-date.
RPM packages have the .rpm file extension, which contains the software along with metadata (e.g., version, architecture, dependencies, and installation scripts).

Key RPM Commands

1. **Installing an RPM Package**:
    - To install a package using RPM, you use the following command:
    - rpm -ivh package-name.rpm
        - -i: Install the package.
        - -v: Verbose (gives detailed information).
        - -h: Shows hash marks (#) as a progress bar.
2. **Upgrading an RPM Package**:
    - To upgrade an existing package:
    - rpm -Uvh package-name.rpm
        - -U: Upgrade the package (install if not already installed).
3. **Removing an RPM Package**:
    - To remove an installed package:
    - rpm -e package-name
        - -e: Erase (remove) the package.
4. **Querying RPM Packages**:
    - To check if a package is installed and get details:
    - rpm -q package-name
        - -q: Query the package.
    - To list all installed packages:
    - rpm -qa
        - -qa: Query all installed packages.
5. **Verifying an RPM Package**:
    - To verify the integrity of an installed package:
    - rpm -V package-name
        - -V: Verify the integrity of the package files.
6. **Listing Files in an RPM Package**:
    - To list the files contained in an RPM package:
    - rpm -ql package-name
        - -ql: List files in the installed package.
7. **Building RPM Packages**:
    - You can create your own RPM packages for custom software installations:
    - rpmbuild -ba spec-file.spec

Example of RPM Usage

Scenario: Installing a custom application on a Red Hat-based system

Let's say you have a custom application packaged as an RPM file named example-app-1.0-1.x86_64.rpm that you want to install.

1. **Install the RPM Package**:
2. rpm -ivh example-app-1.0-1.x86_64.rpm
    - This installs the application on the system. The -ivh option provides a verbose installation with a progress bar.
3. **Verify the Installation**: To check if the package is installed:

4. rpm -q example-app
    - This confirms the version of the installed package.
5. **List Files in the Installed Package**: To see what files were installed by this package:
6. rpm -ql example-app
7. **Remove the RPM Package**: If you no longer need the application:
8. rpm -e example-app
    - This removes the example-app package from the system.

Real-World Scenario

Scenario: Managing Web Server Software

Imagine you're managing a web server (e.g., Apache) on a Red Hat-based system. You're using RPM to manage software packages related to the server.

1. **Installing Apache Web Server**: You may want to install the Apache web server (httpd package):
2. rpm -ivh httpd-2.4.6-93.el7_9.1.x86_64.rpm
    - This installs Apache from the .rpm package.
3. **Upgrading Apache**: When a new version of Apache is released, you might upgrade using:
4. rpm -Uvh httpd-2.4.6-94.el7_9.2.x86_64.rpm
    - This command upgrades Apache to the new version.
5. **Querying the Installed Package**: To check if Apache (httpd) is installed and its version:
6. rpm -q httpd
7. **Verifying Installed Package**: You may want to ensure that all files installed by Apache are in place and have not been modified:
8. rpm -V httpd
9. **Removing Apache**: If you need to remove Apache:
10. rpm -e httpd
    - This removes Apache from your system.

Conclusion

RPM is an essential tool for managing packages on Red Hat-based Linux systems. It offers a straightforward way to install, upgrade, remove, and verify software packages. In a real-world scenario, system administrators use RPM commands to manage software packages, ensuring that the system is up-to-date, secure, and running the required software.

By using RPM, organizations can automate software deployment, ensure consistency across systems, and manage custom packages effectively.

Package name :

It si divided into 4 parts:

For eamcple: xz-java-1.8-14.e19.noarch.rpm (explain these terms also)

In the example **xz-java-1.8-14.e19.noarch.rpm**, the package name is divided into four parts that provide important information about the package. Here's a breakdown of each part:

1. Name of the Package (xz-java)

- This is the actual name of the package being installed. In this case, xz-java refers to the software package, which is a Java-related package built using the xz compression format.
- **Explanation**: The name usually represents the software or application, and in this case, it's a Java package, possibly a library or a utility related to the xz compression algorithm.

2. Version of the Package (1.8)

- This part specifies the version of the package. Here, 1.8 is the version number, which tells you which release or iteration of the software you're dealing with.
- **Explanation**: The version is used to track updates and changes in the software. The version number usually increments when new features, improvements, or fixes are added. For example, version 1.8 might include significant updates compared to version 1.7.

3. Release Date of the Package (14.e19)

- This part contains the release number, which usually indicates a specific build or version of the package, which may differ from the general version number.
    - 14 is the release number.
    - .e19 often represents a specific build number or a unique identifier used by the package maintainers. In this case, it likely indicates that this is the 14th release of this package, with e19 potentially denoting a specific build or date.
- **Explanation**: The release number helps distinguish between different builds or iterations of the same version of a package. It's useful when the package has been updated multiple times under the same version number, perhaps for bug fixes or minor tweaks.

4. Architecture of the Package (noarch)

- This part indicates the architecture for which the package is built. Here, noarch stands for "no architecture," meaning that the package is architecture-independent and can run on any platform (e.g., x86_64, i386, ARM, etc.).
- **Explanation**: When a package is labeled as noarch, it means that the software doesn't depend on specific hardware architecture and can run on any machine, regardless of whether it's a 32-bit or 64-bit system. This is often the case with software that is written in an interpreted language (like Java) or doesn't require direct interaction with hardware.

Complete Breakdown of the Package Name:

- **xz-java-1.8-14.e19.noarch.rpm**:
    - **xz-java**: The name of the package (Java-related software).
    - **1.8**: Version of the package.
    - **14.e19**: Release number and build identifier.
    - **noarch**: Architecture-independent package.

This structure helps system administrators and package managers understand key information about a package, such as what the software is, which version it is, when it was released, and whether it's architecture-specific or not.

Here’s a breakdown of the dnf commands you've mentioned and an explanation of how they work in a real-world scenario:

1. dnf install package_name*

- **Description**: This command is used to install one or more packages on your system. The * is a wildcard that allows you to install multiple packages that match the specified name pattern. For example, if you want to install all packages that start with "xz," you can use dnf install xz* to install packages like xz-java, xz-utils, etc.
- **Example**:
- dnf install xz*
- **Real-world scenario**: If you’re working with Java-related compression utilities, you may want to install all xz-related packages on your system. This command simplifies installing all matching packages without needing to specify each one individually.

2. dnf info package_name

- **Description**: This command shows detailed information about a specific package. It provides metadata such as the version, release date, repository information, dependencies, and more. This is useful for checking the specifics of a package before installing or updating it.
- **Example**:
- dnf info xz-java
- **Real-world scenario**: If you are unsure about the version or details of a package you want to install or update, this command helps you review the package details to ensure it meets your requirements before proceeding.

3. dnf list all

- **Description**: This command lists all the available packages in the system, showing whether they are installed or available in the enabled repositories. It can be a very useful tool to view all software available for installation.
- **Example**:
- dnf list all
- **Real-world scenario**: As a system administrator, you may want to check what software is available on your system, especially when troubleshooting issues or verifying that certain packages are installed.

4. dnf group info package_name

- **Description**: This command provides information about a specific package group. Package groups are collections of related packages, such as all the packages needed for a web server or a desktop environment. For example, dnf group info "Web Server" would provide a list of packages related to web server installation.

- **Example**:
- dnf group info "Development Tools"
- **Real-world scenario**: When setting up a development environment on a server, you can use this command to check all the tools related to development, such as compilers, libraries, and debugging tools, before installing them.

5. dnf update package_name

- **Description**: This command updates a specific package to the latest version available in the repository. It ensures that you are running the most current version of the package, which may include bug fixes, security patches, or new features.
- **Example**:
- dnf update xz-java
- **Real-world scenario**: As part of system maintenance, you may want to keep your software packages up-to-date to ensure security patches and new features are applied. Running this command ensures the specific package you are interested in is updated.

6. dnf remove package_name* -y

- **Description**: This command removes a package or group of packages from the system. The * wildcard can be used to match multiple packages, and the -y option automatically confirms the removal, bypassing the confirmation prompt. This can be useful when removing multiple related packages or when performing system cleanup.
- **Example**:
- dnf remove xz* -y
- **Real-world scenario**: If you want to remove all xz-related packages from your system (for example, during a cleanup or after deprecating certain software), this command allows you to do it efficiently without needing to manually confirm each removal.

Summary of dnf commands:

Command	Description
dnf install package_name*	Installs multiple packages matching the specified name pattern.
dnf info package_name	Displays detailed information about a package (version, dependencies, etc.).
dnf list all	Lists all packages available or installed on the system.
dnf group info package_name	Provides information about a package group (a collection of related packages).
dnf update package_name	Updates the specified package to the latest available version.
dnf remove package_name* -y	Removes a specified package or group of packages, with automatic confirmation.

These commands are essential for package management on Red Hat-based Linux distributions (such as RHEL, CentOS, Fedora) and provide a powerful way to manage software installation, removal, and maintenance. They are particularly useful in environments where automation, efficiency, and package control are key to smooth system management.

Day 5: Linux System Management & Networking Essentials: From Partitioning to IP Configuration

---

How to Configure a Local Repository in RHEL 9.3

1. **Open the student workstation console.**
2. **Connect to the server named servera.**
3. **Navigate to the yum repository configuration directory** by running:
4. cd /etc/yum.repos.d/
5. **List the files in the directory** using:
6. ls

You'll see files like redhat.repo and rhel_dvd.repo.

7. **Display the contents of rhel_dvd.repo:**
8. cat rhel_dvd.repo

You'll find entries like:

[rhel-9.3-for-x86_64-baseos-rpms]

baseurl = http://content.example.com/rhel9.3/x86_64/dvd/BaseOS

enabled = true

gpgcheck = false

name = Red Hat Enterprise Linux 9.3 BaseOS (dvd)

[rhel-9.3-for-x86_64-appstream-rpms]

baseurl = http://content.example.com/rhel9.3/x86_64/dvd/AppStream

enabled = true

gpgcheck = false

name = Red Hat Enterprise Linux 9.3 AppStream (dvd)

9. **Create a new repo file** (for example: xyz.repo) using:
10. cat > xyz.repo

Then enter the following content:

[BaseOS]

name=BaseOS

baseurl=http://content.example.com/rhel9.3/x86_64/dvd/BaseOS

enabled=1

gpgcheck=0

[AppStream]

name=AppStream

baseurl=http://content.example.com/rhel9.3/x86_64/dvd/AppStream

enabled=1

gpgcheck=0

Press Ctrl + D to save and exit.

11. **Verify the configured repositories** by running:
12. dnf repolist all

You should see the following enabled repos:

- AppStream
- BaseOS
- rhel-9.3-for-x86_64-appstream-rpms
- rhel-9.3-for-x86_64-baseos-rpms

Real-World Scenario Analogy:

Consider how we **download apps from the Google Play Store** on our smartphones. The Play Store acts as a **repository**, where all apps (packages) are stored. Just like how you browse, find, and install an app, your Linux system uses repositories to find and install software.

- **Play Store = Linux Repositories**
- **Apps = Software Packages**
- **Internet connection to access Play Store = baseurl in repo config**
- **Enabling/disabling apps in settings = enabling/disabling repos**

By configuring the repository in Linux, we are basically telling our system **where to look** for software and giving it **permission to access and install it**, just like allowing your phone to access the Play Store and install apps.

How to Access the Linux File System

1. Why Files May Not Be Accessible

By default, Linux does **not automatically access files from external devices** like USB drives, CDs, or DVDs. Unlike Windows, Linux requires a few steps before using files from such devices.

---

2. Steps to Access Files in Linux:

To access files from storage devices (internal or external), you must follow **three steps**:

1. **Partitioning**
    - Divide the disk into sections called *partitions* using tools like fdisk, parted, or gparted.
2. **File System Creation**
    - Format each partition with a file system like ext4, xfs, or vfat.
3. **Mounting**
    - Attach the partition to a directory (known as the **mount point**) using the mount command so that the files become accessible in the Linux directory structure.

---

3. Device Files – /dev Directory

All hardware devices in Linux are represented as **device files** in the /dev directory. For example:

- /dev/sda – First physical hard disk
- /dev/vda – First virtual hard disk
- /dev/cdrom – CD/DVD drive

These files act as an interface to the actual hardware.

---

4. Useful Command – lsblk

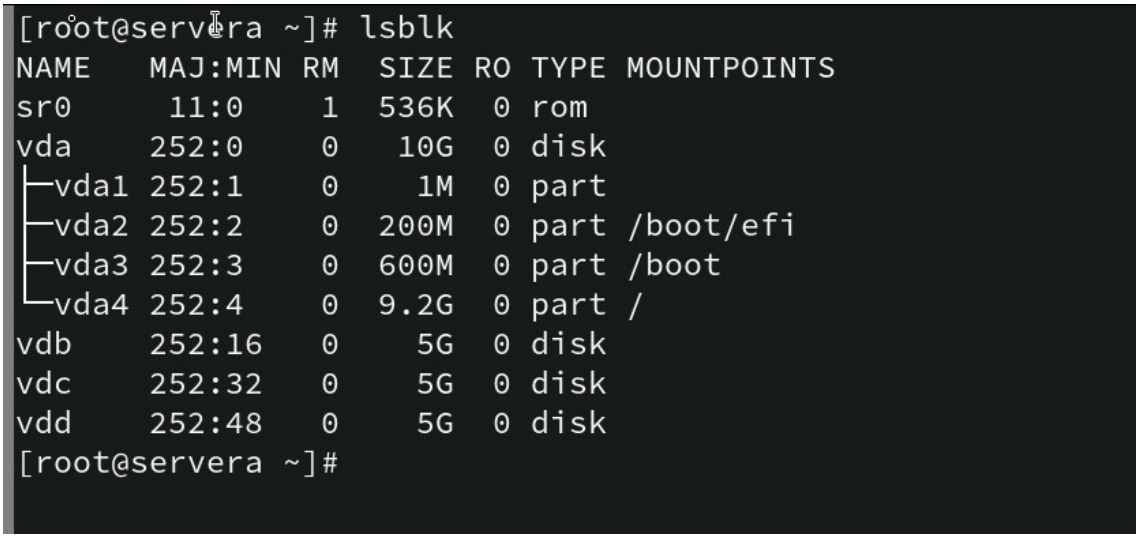
The command lsblk lists all **block devices** (like hard disks and partitions). It shows information like:

- Device names
- Sizes
- Types (disk/partition)
- Mount points

---

5. Types of Disks

Type	Device Names
Physical HDDs	sda, sdb, sdc
Virtual HDDs	vda, vdb, vdc
SSDs (NVMe)	nvme0n1, nvme0n2



📸 **Explanation of the Image:**

You ran the lsblk command on a virtual machine. Here's what it shows:

- vda (10 GB virtual disk) is split into 4 partitions:
    - vda1 – 1M (not mounted)
    - vda2 – 200M mounted on /boot/efi
    - vda3 – 600M mounted on /boot
    - vda4 – 9.2G mounted on / (root directory)
- vdb, vdc, and vdd – These are additional 5 GB virtual disks, not yet partitioned or mounted.

Partitioning of a Drive

1. **Partitioning a Virtual Hard Disk:**
    - If you want to **partition a virtual hard disk** like vda, it can be split into partitions such as:
        - vda1
        - vda2
        - vda3

        - etc.
    - ✅ Yes, **this is possible and commonly done** in both physical and virtual environments.

2. **Naming Convention:**
    - **Disk Name** → Always ends with **an alphabet**, for example:
        - sda, sdb, vda, vdb, etc.
    - **Partition Name** → Always ends with a **number**, for example:
        - sda1, sda2, vda1, vda2, etc.

✅ Purpose of Partitioning a Disk

Partitioning a hard disk is important for several reasons:

- 🎯 **Organization of Data**
  Helps organize files and directories in a structured way.
- 📂 **Separation of System and User Files**
  Example: You can have one partition for the system (/), one for boot files (/boot), and another for personal files (/home).
- 🔐 **Improved Security and Access Control**
  Limits access or damage if a particular partition fails or is compromised.
- 📊 **Efficient Memory Management**
  Allocates space more efficiently for different use cases and improves performance.
- 🔄 **Easy Backup and Restore**
  You can back up only specific partitions without affecting the entire disk.
- 💻 **Support for Multi-OS**
  Allows installation of multiple operating systems on the same disk by using different partitions.

🔰 What is a Boot Loader?

A **Boot Loader** is a small program responsible for **loading the operating system** (OS) into memory when the system starts. It is the **first software** that runs after the BIOS/UEFI finishes its tasks.

📦 To create partitions on a disk, we need to understand two types of partitioning methods:

1. MBR (Master Boot Record)

- **Definition:** MBR is an older partitioning method used to manage disk partitions.
- **Max Partitions:** Allows **up to 4 primary partitions** only.

    - If more than 4 partitions are needed:
        - Create **3 primary partitions + 1 extended partition**.
        - Inside the extended partition, you can create **logical partitions**.
        - Logical partitions always **start from the 5th partition (e.g., sda5)**.
- **Partition Size Limit:** Each partition can be up to **2 TB**.
- **Storage of Partition Table Info:**
    - Stored at the **beginning of the disk**.
    - ⚠️ If the partition table is corrupted, the system **won't boot**, and data **may be lost**.

🧪 **Example Structure:**

Disk: vdb

Partitions:

vdb1 - Primary (P)

vdb2 - Primary (P)

vdb3 - Primary (P)

vdb4 - Extended (E)

vdb5 - Logical (inside Extended)

---

2. GPT (GUID Partition Table)

- **Definition:** GPT is a modern replacement for MBR, part of the UEFI standard.
- **Max Partitions:** Supports **up to 128 partitions** directly (no need for primary/extended).
- **Partition Size Limit:** Each partition can be up to **8 ZB (Zettabytes)** (theoretical limit).
- **Partition Table Storage:**
    - Stored at **both the beginning and end** of the disk.
    - ✔️ If the beginning gets corrupted, the system can **recover using the backup at the end**.

⚖️ **Comparison Summary:**

Feature	MBR	GPT
Max Partitions	4 primary (or 3P + 1E + logic)	128 partitions

Feature	MBR	GPT
Partition Size Limit	2 TB	8 ZB
Partition Table Stored	Beginning of the disk	Beginning + End of the disk
Recovery Option	No	Yes, from backup
Partition Types	Primary, Extended, Logical	No such classification

---

💻 **How to Create, Format & Mount a Partition in Linux**

📌 **1. What is Partitioning?**

Partitioning is the process of dividing a hard disk into multiple logical sections so that each section can be managed separately. It helps in:

- Organizing data
- Managing memory efficiently
- Running multiple OS or separating system files from user files

---

🧰 **2. Tools Required:**

- fdisk → For creating partitions (MBR format)
- partprobe → To update the kernel with the new partition table
- lsblk → To list all available block devices
- mkfs → To format partitions with a file system
- mount → To mount a partition
- /etc/fstab → For permanent mounting (optional)

---

3. Step-by-Step Process

✅ **STEP 1: Identify the Disk**

Use the lsblk command to view all disks and partitions:

lsblk

Example output:

NAME SIZE TYPE MOUNTPOINT

vda 10G disk

vdb 5G disk

Let's say you want to create a partition on /dev/vdb.

✅ **STEP 2: Create a Partition Using fdisk**

fdisk /dev/vdb

Inside fdisk, follow these steps:

- Press n → to create a new partition
- Select partition type → p for primary
- Choose partition number (1, 2, etc.)
- Set the start and end size (example: +2G for 2GB)
- Press w → to write and save the changes

Example:

Command (m for help): n

Partition type:

p primary

e extended

Select (default p): p

Partition number (1-4): 1

First sector: (default)

Last sector: +2G

Command (m for help): w

✅ **STEP 3: Update the Kernel**

After partitioning, run:

partprobe /dev/vdb

This tells the system to reload the partition table.

✅ **STEP 4: Format the New Partition**

Now format the newly created partition (/dev/vdb1) with a file system like ext4:

mkfs.ext4 /dev/vdb1

Other file system options:

- mkfs.xfs /dev/vdb1 → XFS format
- mkfs.vfat /dev/vdb1 → FAT format

✅ **STEP 5: Mount the Partition**

Create a mount point (directory) to access the partition:

mkdir /mnt/mydisk

mount /dev/vdb1 /mnt/mydisk

Now your partition is accessible at /mnt/mydisk.

✅ **STEP 6: Verify the Mount**

To confirm:

lsblk

Or:

df -h

🔒 **STEP 7 (Optional): Permanent Mounting via /etc/fstab**

To make the mount permanent (auto-mount on boot):

1. Get the UUID of the partition:

blkid /dev/vdb1

2. Edit /etc/fstab:

nano /etc/fstab

Add this line at the bottom:

UUID=your-uuid-here /mnt/mydisk ext4 defaults 0 0

Save and exit. Test it:

mount -a

💡 **Summary**

Step	Action	Command
1	View disks	lsblk
2	Partition disk	fdisk /dev/vdb
3	Apply changes	partprobe /dev/vdb
4	Format partition	mkfs.ext4 /dev/vdb1
5	Create mount point	mkdir /mnt/mydisk
6	Mount partition	mount /dev/vdb1 /mnt/mydisk
7	Verify	df -h or lsblk
8	Auto-mount	/etc/fstab entry

Great! Here's the **complete guide** based on your input, structured neatly and clearly for **college notes, revision, or hands-on use**. I've corrected the typos and explained each concept properly.

🧹 How to Delete a Partition in Linux

📜 Steps to Delete a Partition Using fdisk

1. Open the partition tool:
2. fdisk /dev/vdb
3. Press d → to delete a partition.
4. It will ask for the partition number (e.g., 1, 2, etc.). Enter the number you want to delete.
5. You will get a confirmation message like:
6. Partition 2 has been deleted successfully
7. Press w → to write the changes and exit fdisk.
8. (Optional) Run partprobe to update the kernel:
9. partprobe /dev/vdb

🏗️ How to Create a File System

📂 Step-by-Step:

1. After partition creation, format it using mkfs.

Example for XFS:

mkfs.xfs /dev/vdb1

Example for EXT4:

mkfs.ext4 /dev/vdb1

2. When you run this command:
    - It **allocates inodes** (called **anode allocation**)
    - It arranges data in **blocks**
3. To verify and view filesystem type and UUID:
4. blkid

📦 Mounting a Partition

📄 What is Mounting?

Mounting is the process of **attaching a file system** (like a partition or USB) to an **empty directory** (called the **mount point**) so that you can access its contents.

🔨 Mounting Process:

Let's say we want to mount /dev/vdb1 to a directory called cloud.

✅ Steps:

1. Create a mount point (empty directory):
2. mkdir /mnt/cloud
3. Format the partition as ext4 (if not already done):
4. mkfs.ext4 /dev/vdb1
5. Mount the partition to the directory:
6. mount /dev/vdb1 /mnt/cloud
7. Check if it's mounted:
8. lsblk
9. df -h

📌 Example Use-Case:

Goal:
Mount a partition /dev/vdb1 to a folder called cloud with ext4 file system.

mkfs.ext4 /dev/vdb1 # Step 1: Format

mkdir /mnt/cloud # Step 2: Create directory

mount /dev/vdb1 /mnt/cloud # Step 3: Mount

df -h # Step 4: Check

🗑️ **How to Delete a Mounted Disk in Linux**

✅ **Step-by-Step Procedure:**

---

1. Unmount the Mount Point

Before deleting a partition, you must unmount it first.

umount /mnt/cloud

Replace /mnt/cloud with your actual mount point.

---

2. Verify Unmounting

Check whether the partition is still mounted or not:

lsblk

Look for your disk name and confirm it is no longer mounted.

---

3. Open fdisk Tool

Start working with the disk using fdisk:

fdisk /dev/vdb

Replace /dev/vdb with the actual disk name.

---

4. Delete the Partition

- Press d to delete.
- It will ask for the partition number (e.g., 1, 2, etc.)
- Confirm deletion.

---

5. Write and Exit

- Type w to **write changes** and exit.

---

6. Final Verification

Use lsblk again to verify the partition is deleted:

lsblk

Here's a clear explanation of the **locate command** in Linux, along with its usage:

🔍 **locate Command – Usage & Explanation**

📌 **What is locate?**

The locate command is used to **quickly find the location of files and directories** on your Linux system. It searches through a pre-built database rather than scanning the entire file system, which makes it extremely fast.

🧾 **Syntax:**

locate [filename or keyword]

✅ **Example:**

locate test.txt

This will list all paths where test.txt is found.

🛠️ **How It Works:**

- It uses a database stored in /var/lib/mlocate/mlocate.db.
- This database is updated using the updatedb command.

🔄 **To Update the Database:**

sudo updatedb

Run this if locate doesn't find recently created files.

🧠 **Tip:**

Use locate when you need to find file locations without navigating directories manually.

🔎 **find Command – Usage & Explanation**

📌 **What is find?**

The find command is used to **search for files and directories** in a specified location **based on different conditions** like name, size, type, modified time, permissions, etc.

📜 **Syntax:**

find [path] [options] [expression]

✅ **Examples:**

1. **Find a file by name:**

find /home -name "file.txt"

Searches for file.txt in the /home directory and its subdirectories.

---

2. **Find all .txt files in current directory:**

find . -name "*.txt"

---

3. **Find all empty files:**

find / -type f -empty

---

4. **Find files modified in the last 1 day:**

find /home -mtime -1

---

5. **Delete all .log files (use with caution):**

find /var/log -name "*.log" -delete

🧠 **Key Differences from locate:**

- find is **real-time** (searches the disk directly).
- locate is **faster** but uses an **outdated database** unless refreshed with updatedb.

📊 **Difference Between find and locate**

Feature	find	locate
🔍 **Search Type**	Real-time search from the file system	Searches from a pre-built database
⚡ **Speed**	Slower (scans disk each time)	Very fast
🕒 **Updated Results**	Always up to date	May be outdated (needs updatedb)
🛠️ **Installation**	Pre-installed on most systems	May need to install mlocate or locate
🎯 **Search Criteria**	Can filter by name, size, time, type, etc.	Only by file/directory name
🗃️ **Database Used**	No database, live scan	Uses /var/lib/mlocate/mlocate.db
📜 **Command Example**	find /home -name "file.txt"	locate file.txt
🧠 **Advanced Options**	Supports actions (e.g., -delete, -exec)	Basic listing only

---

🧠 **In Summary:**

- Use **find** for **precise, powerful, and live searches**.
- Use **locate** for **quick filename lookups** when speed matters

---

🌐 **Networking Basics Explained**

🖥️ **Networking Definition**

Networking refers to **connecting two or more devices** (like computers, mobiles, etc.) to **share resources** like files, internet, printers, etc. over a **wired or wireless medium**.

2️⃣ IP Address of a Device

An IP (Internet Protocol) address is a **unique identifier** given to each device on a network to communicate with other devices.

📜 **Example**: 192.168.1.1

3️⃣ Hostname of a Device

Hostname is the **name** assigned to a device on a network. It helps to **identify devices by name** instead of remembering IPs.

📜 **Example**: my-computer.local

✅ How to Verify IP Address

You can check your system's IP using:

- ip addr → shows IP address & interface details
- ifconfig → older command to see network info

🌎 DNS – Domain Name Service

DNS converts **human-readable names** (like www.google.com) into **IP addresses** (like 142.250.192.110). It's like a **phonebook** of the internet.

💻 Hostname

To see your hostname, use:

hostname

🔔 Important Networking Commands

- ip addr → Displays all IP-related details
- ping 8.8.8.8 → Tests internet connectivity (Google DNS)
- ifconfig → Shows interfaces and IPs

📶 IP Address Types

Type	Description	Example Use Case
Static	Fixed IP that doesn't change	LAN, Servers, CCTV

Type	Description	Example Use Case
Dynamic	IP assigned temporarily via DHCP	Mobile devices, WiFi

---

🔢 **IPv4**

- 32-bit address (e.g., 192.168.0.1)
- Divided into **4 octets**
- Used widely today

🔢 **IPv6**

- 128-bit address (e.g., fe80::1)
- Allows **billions** of IPs
- Used as an upgrade to IPv4 due to IP shortage

Summary:

✅ **Linux File System & Partitioning Summary**

- **Device Names**:
  - Disk ends with a letter (e.g., vda)
  - Partition ends with a number (e.g., vda1)
- **Purpose of Partitioning**:
  - Organizes data, improves access and memory management

💽 **Partition Types**

- **MBR (Master Boot Record)**:
  - Max 4 primary partitions
  - Supports extended and logical partitions
  - Max size per partition: 2TB
  - Partition info stored at beginning only
- **GPT (GUID Partition Table)**:
  - Max 128 partitions
  - No primary/extended concepts

- Max size per partition: 8ZB
- Partition info stored at beginning and end

🛠️ Partitioning Steps

1. fdisk /dev/diskname
2. n → create new partition
3. Set number and size
4. w → write changes
5. partprobe /dev/diskname
6. lsblk → verify

❌ Delete Partition

- fdisk /dev/diskname
- d → delete partition
- w → write
- lsblk → verify

📄 Create File System

- Command: mkfs.ext4 /dev/vdb1
- blkid → check filesystem

📂 Mounting and Unmounting

- Create mount point: mkdir /mountname
- Mount: mount /dev/vdb1 /mountname
- Unmount: umount /mountname

🔍 Search Commands

- locate filename
  - Uses database, faster, may be outdated
- find /path -name filename
  - Real-time, accurate, slower

Networking Basics

- **IP Address**: Unique address for a device
- **Hostname**: Name of device on network
- **DNS**: Resolves domain names to IPs

Network Commands

- ip addr – view IP info
- hostname – show device name
- ping 8.8.8.8 – test connectivity
- ifconfig – view interfaces (older)

IP Types

- **Static IP** – fixed, stable (e.g., LAN)
- **Dynamic IP** – changes, assigned by network
- **IPv4** – 32-bit, 4 octets
- **IPv6** – 128-bit, supports more devices

DAY 6: IPv4 Addressing and Static IP Configuration in Linux: A Practical Guide

---

IPv4 Configuration

- IPv4 uses a **32-bit address** system for identifying devices on a network.
- The 32 bits are typically divided into four **8-bit octets**, represented in **dotted decimal format** (e.g., 192.168.0.1).

Address Classes and Their Properties

Class	Range	Default Subnet Mask	Purpose
A	1.0.0.0 to 126.255.255.255	255.0.0.0	Large networks
B	128.0.0.0 to 191.255.255.255	255.255.0.0	Medium networks
C	192.0.0.0 to 223.255.255.255	255.255.255.0	Small networks
D	224.0.0.0 to 239.255.255.255	N/A	Multicasting
E	240.0.0.0 to 255.255.255.255	N/A	Experimental

Note: Class D and E are not used for regular IP addressing.

Binary Octet Representation

- Each octet in an IPv4 address can represent values from **0 to 255**.
- This is due to the sum of 8-bit binary values:

20+21+22+23+24+25+26+27=2552^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 = 255

Address Class Bit Allocation

Class	Network Bits
A	8 bits
B	16 bits

Class	Network Bits
C	24 bits

How to Configure a Static IP Address

There are **two methods** to configure a static IP address:

1. **Graphical User Interface (GUI)**
2. **Command Line Interface (CLI)**

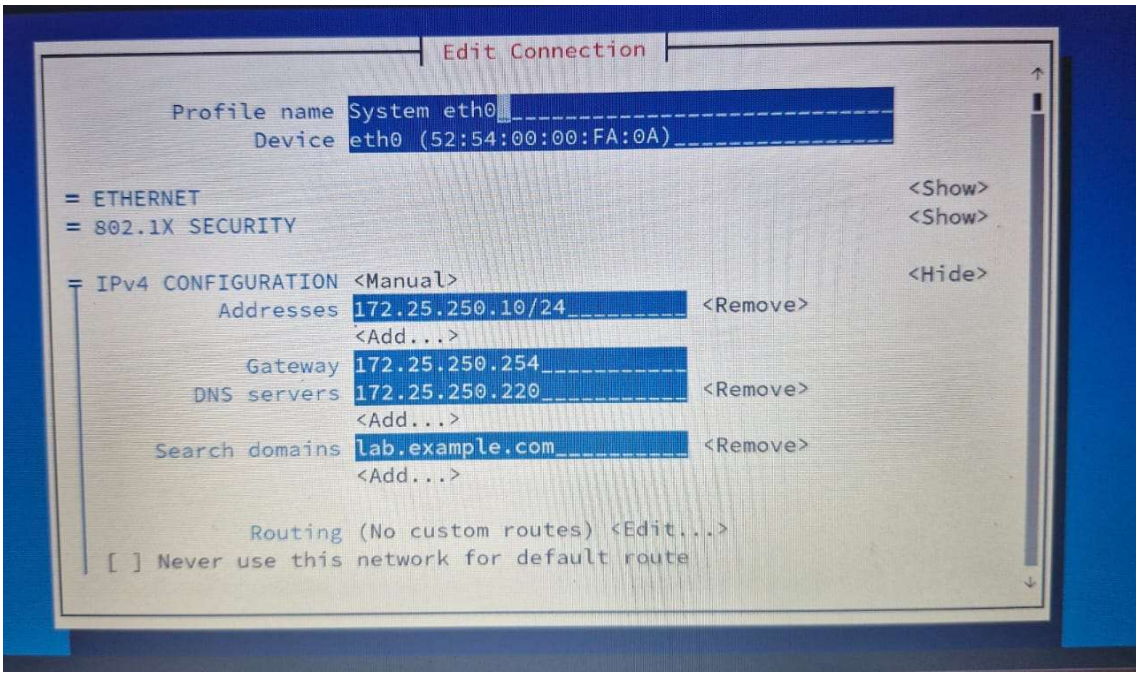
---

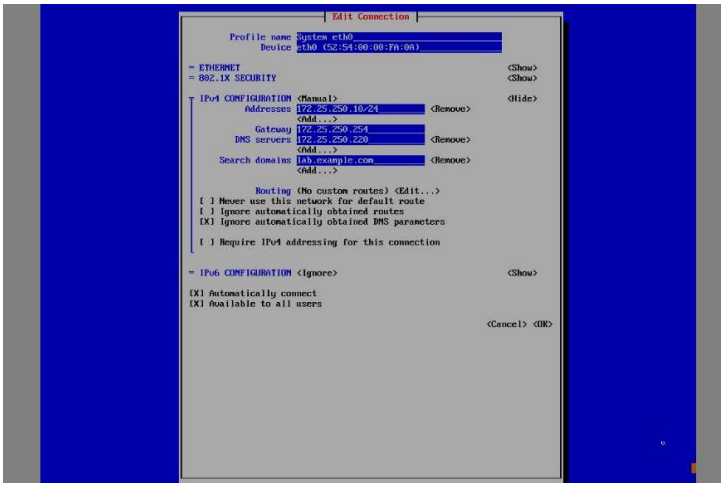
1. Configuration Using Graphical User Interface (GUI)

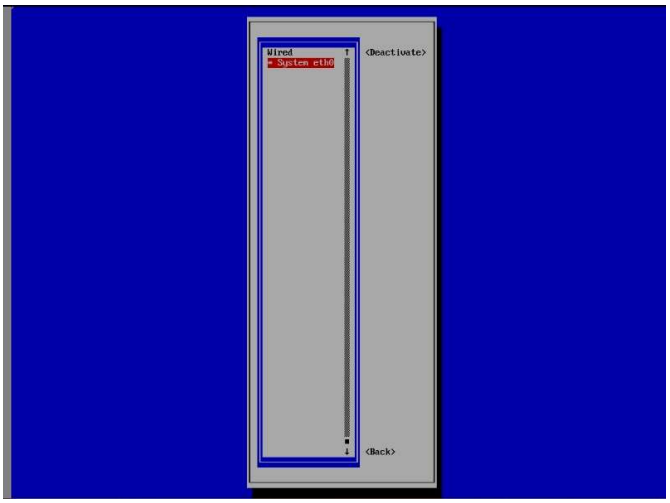
Step-by-step process:

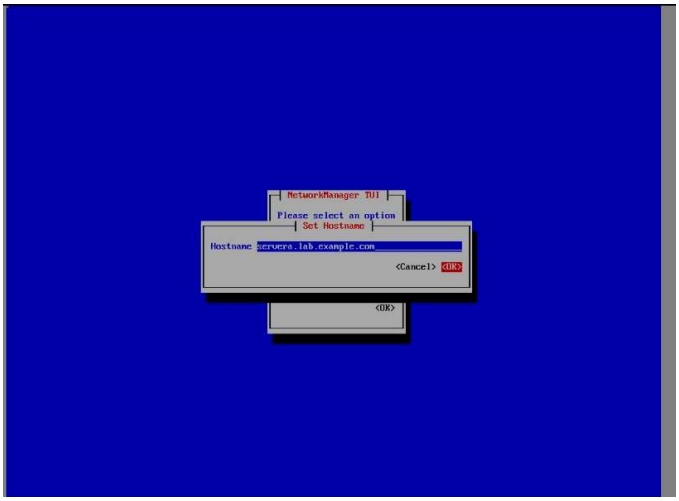
1. **Open Terminal**, log in as the **root user**, and type the following command:
2. nmtui
3. **Follow the prompts**:
    - Select **"Edit a connection"**
    - Choose the desired connection
    - Go to **IPv4 Configuration**
    - Set the method to **Manual**
    - Click on **"Show"** and provide the following details:
        - **Address**: 172.25.250.10/24
        - **Gateway**: 172.25.250.254
        - **DNS Server**: 172.25.250.254
        - **Domain Name**: lab.example.com
4. Uncheck or ignore the **“Automatically obtained DNS”** option.
5. Click **OK**.
6. Go **Back**, then choose **“Activate a connection”**:
    - **Deactivate** the current connection
    - **Reactivate** it (press Enter twice)
7. Select **"Set system hostname"** and enter:
8. servera.lab.example.com

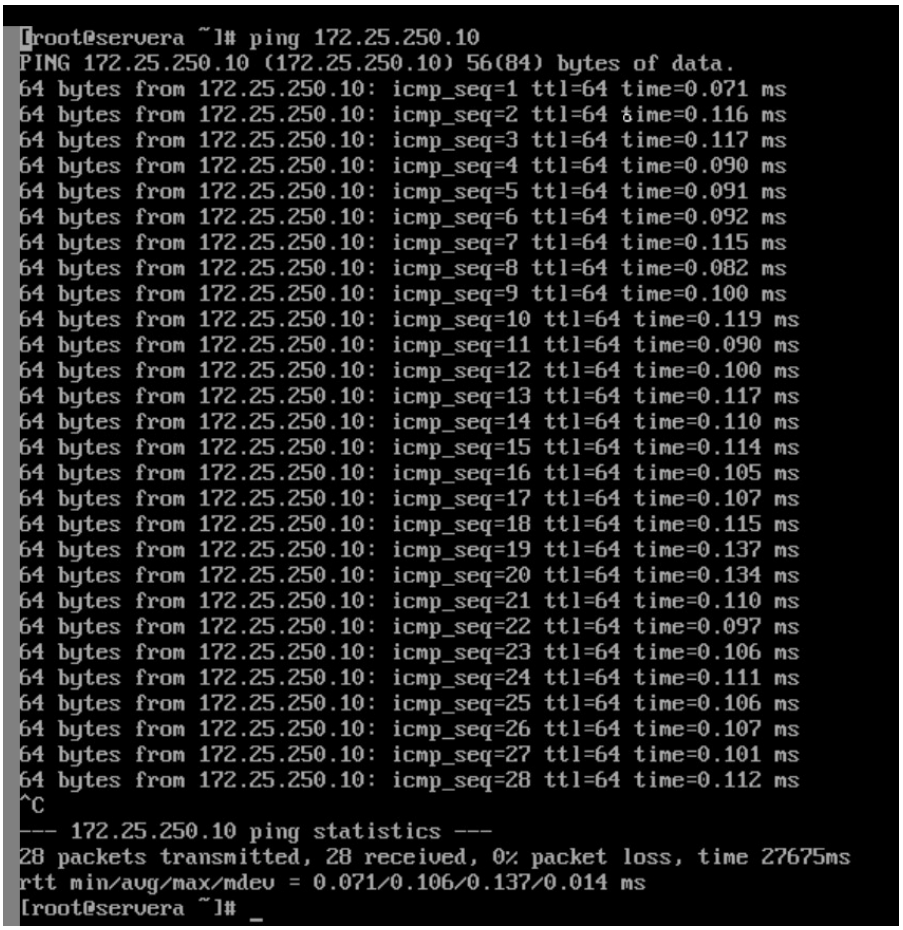
9. Choose **OK → OK → Quit**.
10. Press Ctrl + L to clear the screen.
11. Test the configuration by typing:
12. ping 172.25.250.10






Wired
System eth0
<Deactivate>
<Back>


NetworkManager TUI
Please select an option
Set Hostname
Hostname server1.lab.example.com
<Cancel> <OK>
<OK>



---

2. Configuration Using Command Line Interface (CLI)

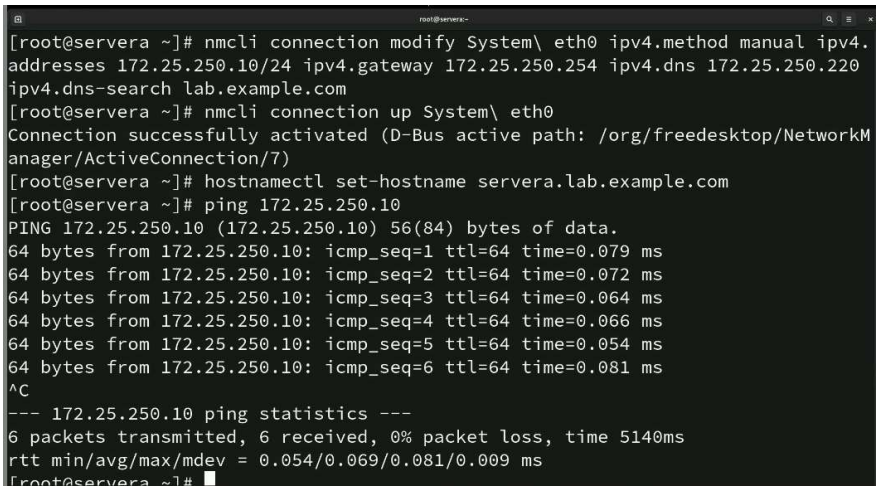
Step-by-step process:

1. Check the available network connections:
2. nmcli connection show
3. Modify the connection with the static IP configuration:
4. nmcli connection modify "System eth0" ipv4.method manual ipv4.addresses 172.25.250.10/24 ipv4.gateway 172.25.250.254 ipv4.dns 172.25.250.220 ipv4.dns-search lab.example.com

(Note: Use escape character \ if there's a space in the connection name, like System\ eth0)

5. Activate the connection:
6. nmcli connection up "System eth0"
7. Set the hostname:
8. hostnamectl set-hostname servera.lab.example.com
9. Verify the configuration by pinging the gateway:

10. ping 172.25.250.254



Conclusion

Understanding IPv4 configuration is fundamental to managing network communication effectively. With its 32-bit address system and class-based structure, IPv4 enables unique identification of devices across networks of varying sizes. Configuring a static IP address is essential for stable network setups, especially in server environments. Whether using the user-friendly **Graphical User Interface (GUI)** or the more flexible **Command Line Interface (CLI)**, both methods offer precise control over network settings. Mastery of these techniques ensures reliable connectivity, enhanced network management, and better troubleshooting capabilities in real-world networking scenarios.