**Linux Programming: Assignment-1**

**NAME-K.V.S Tanveesh USN:ENG24CY0119 ROLL:19 sec-CY(C)**

**Q1) what is Linux Operating System ?**

Ans: In general Linux is basically a open-source developed by Linus Torvalds in 1991. Linux acts as an Core component of many operating systems Called Linux Distributions.

Linux OS is widely used in servers, desktops, embedded systems, and mobile devices due to its stability, security, and flexibility.

**3-Pros of Linux :**

1. **Open Source and Free**  
   Linux is freely available and users can view, modify, and distribute its source code.
2. **Security and Stability**  
   Linux is less vulnerable to malware and viruses compared to other OSes, making it highly secure and stable for long-running systems.
3. **Highly Customizable**  
   Users can customize Linux extensively, from the kernel to the desktop environment and applications, to suit specific needs.

**3-Cons of Linux :**

1. **Steeper Learning Curve for Beginners**  
   New users, especially those familiar only with Windows or macOS, may find Linux commands and system management challenging initially.
2. **Limited Support for Some Commercial Software and Games**  
   Many popular commercial applications and games do not natively support Linux, which can limit usability for some users.
3. **Hardware Compatibility Issues**  
   Some hardware manufacturers do not provide Linux drivers, leading to compatibility problems with certain devices like printers or graphics cards.

**Q2: Differentiate between Linux, Mac, Android, and Windows OS with at least six unique features.**

**ANS:**

When we compare these four operating systems, we see that each one is made for a different purpose and user base. Here’s how they differ:

1. **Kernel and base design**
2. Linux runs on a monolithic kernel, it is flexible and you can add/remove modules.
3. macOS has the XNU hybrid kernel (Mach + BSD).

**2. and availability**

1. Linux is open-source (GPL license), anyone can use or modify it.
2. macOS is closed, only for Apple devices.
3. Android is partly open (AOSP is free) but most phones add their own skin and apps.

**3.User interface and experience**

1. Linux depends on the desktop environment — GNOME, KDE, XFCE etc. It’s very customizable but not always consistent.
2. macOS has a polished Aqua UI and works smoothly with iPhones and iPads.
3. Android is touch-based, Material Design style, focused on mobiles.
4. **Applications and ecosystem**
5. Linux software mostly comes through package managers (apt, dnf, pacman). Great for open-source apps but limited commercial ones.
6. macOS apps are via App Store or dmg/pkg downloads, with Xcode for developers.
7. Android apps are mostly from Google Play, also allows APK sideloading.
8. **Security model**
9. Linux has permissions (DAC), plus SELinux/AppArmor for strict control. It is considered very secure for servers.
10. macOS uses Gatekeeper and System Integrity Protection.
11. Android isolates each app in a sandbox, adds SELinux enforcing, but updates depend on phone brands.
12. **Main usage areas**
13. Linux: mostly servers, cloud, IoT, supercomputers, and also for developers.
14. macOS: preferred by creative professionals (design, video editing, coding for Apple ecosystem).
15. Android: number one in smartphones, tablets, smart TVs, also used in cars.

**Q3) Why is Linux preferred for Mainframe Servers for legacy applications? (3 technical reasons)**

**ANS---**

1. **Rock-solid stability and long uptimes**  
   Linux is famous for running non-stop for years without crashing. Mainframes handle critical workloads (like banks or government records) that can’t afford downtime. Linux kernels with LTS (long term support) patches are very reliable, which makes it safe for old legacy apps to keep running smoothly.
2. **Better workload isolation and resource control**  
   Linux has features like cgroups and namespaces that can isolate applications, limit how much CPU or memory they use, and prevent one old app from crashing the entire system. This makes it easier to run many legacy programs together on the same big mainframe hardware without interference.
3. **Virtualization and hardware support**  
   Modern Linux supports strong virtualization (KVM, z/VM) and can directly use mainframe hardware optimizations like high-speed I/O and multiple CPUs. This allows old legacy software to run in virtual environments without changing the original code, while still getting performance close to bare metal.

**Q4) Structure of the Linux File System**

**ANS--** Linux follows a **hierarchical file system** structure, meaning everything starts from a single root directory / and branches out like a tree. Unlike Windows which has drives like C:\ or D:\, Linux treats everything (files, devices, configs) as part of one unified directory tree.

**Key directories inside /:**

1. /bin → Essential user commands (ls, cp, mv, etc.)
2. /sbin → System admin commands (mount, shutdown, etc.)
3. /etc → Configuration files of the system
4. /lib → Shared libraries needed by programs in /bin and /sbin
5. /usr → User programs, libraries, documentation (like /usr/bin, /usr/lib)
6. /var → Variable data like logs, mail, cache, spool
7. /home → Personal directories for users
8. /tmp → Temporary files (auto-cleared)
9. /boot → Bootloader files, kernel image
10. /dev → Device files (disks, USB, etc.)
11. /proc → Virtual filesystem that shows process and kernel info
12. /sys → Exposes kernel and device info

**Q5) If Linux is open-source, how does Red Hat make money?**

**ANS-- 1) What are customers actually buying?**

1. RHEL subscriptions  
   Same Linux you can compile yourself, but curated, tested, and supported for 10 years. Stable kernels, backported fixes, predictable updates. Enterprises hate surprises.
2. OpenShift (Kubernetes platform)  
   Red Hat’s opinionated K8s stack: cluster lifecycle, upgrades, security, logging, service mesh, operators. Basically Kubernetes that won’t blow up your weekend.
3. Automation (Ansible)  
   Enterprise automation + content catalogs + controller/analytics. Teams pay to standardize how they configure, patch, and deploy thousands of nodes.
4. Certifications & compliance  
   FIPS/CIS/STIG baselines, signed updates, provenance. Also “this OS is certified with SAP/Oracle/NVIDIA/VMware/your hardware.” That certification matrix is the moat.

**2) Who pays and why?**

1. Banks, telecom, govt, healthcare, manufacturing—places where downtime is expensive or illegal.
2. They pay because:
   1. Risk transfer: “If it breaks at 2 AM, someone accountable fixes it.”
   2. Lifecycle guarantees: 8–10 years of updates on the same major release.
   3. Ecosystem lock-in (the positive kind): Works out-of-the-box with their storage, NICs, GPUs, HSMs, databases.
   4. Audit comfort: Security baselines mapped to regulations. Auditors like boring, predictable platforms.

**3) How do they price it ?**

1. Per node / per core subscriptions for RHEL (standard vs premium support tiers).
2. Per cluster / per worker style for OpenShift (different SKUs for datacenter, edge, cloud).
3. Enterprise license for Ansible Automation Platform (nodes + controllers).
4. Upsell motion: start with RHEL → add Ansible → land OpenShift → expand to managed OpenShift.

**4) Why does open-source not kill their business?**

1. Curation > code. The raw code is free; the predictable, tested, certified build is not.
2. Integration tax. Pulling hundreds of packages together safely across CPU/GPU/storage/network stacks is real work.
3. Security & backports. They ship CVE fixes without forcing you to jump to a new upstream version; that’s harder than it looks.
4. Enterprise rituals. Change windows, long-term support, break-glass patches, and someone to sign off on risk.

**5) Competitors and substitutes (so you sound market-aware)**

1. Canonical (Ubuntu), SUSE—similar open-source + support play.
2. Cloud-native substitutes: AWS/Azure/GCP managed Kubernetes instead of OpenShift; AMIs and cloud images instead of on-prem RHEL.

**Q6) Write the command to display today’s date and time**

In Linux we can get it if we type:

Bash programming --- date

Then the Output Looks like:

Sat Sep 21 08:42:31 IST 2025

This shows day, month, date, current time, time zone, and year

**Q7) Which command is used to check how long the system has been running?**

The most common command is:

In Bash programming we type ---- uptime

output:

08:55:03 up 3 days, 4:17, 2 users, load average: 0.12, 0.08, 0.01

This means that:

1. Current time → 08:55:03
2. System uptime → 3 days, 4:17
3. Number of logged-in users → 2 users
4. Load average (system load over 1, 5, 15 minutes)

**Q8) Difference between shutdown -h now and halt**

1. shutdown -h now  
   This command tells the system to *shutdown immediately*. It first sends a warning message to all logged-in users, then stops running processes in an orderly way, unmounts file systems safely, and finally powers off the machine.
2. halt  
   The halt command simply stops all CPU functions. On modern Linux systems (with systemd), it also goes through a safe shutdown sequence, but traditionally it justhalts the CPU and doesn’t always power off the machine. Sometimes after halt, you may still need to press the power button to switch it off.

What is the difference difference both:

1. shutdown -h now = graceful and safe shutdown + power off.
2. halt = just stops the system, may not cut power depending on settings.

**Q9) Compare init 0 and shutdown -h. Which is safer? Why?**

1. init 0  
   This command changes the system runlevel to 0, which is the runlevel for system halt. It stops all processes and powers down the system. It works, but it doesn’t always notify logged-in users or give a clean timed shutdown.
2. shutdown -h  
   This command is the standard way to shut down. It safely brings the system to a halt by:
   1. Sending warning messages to all users
   2. Stopping services in the proper order
   3. Unmounting filesystems cleanly
   4. Finally powering off the machine

Which is safer?  
shutdown -h is safer because it does everything in an orderly way — notifies users, prevents new logins, cleans up processes, and avoids file system corruption. init 0 just switches the runlevel, so it can be harsher and less user-friendly.

**Q10) If a server is powered off without proper shutdown, what problems can occur?**

1. File system corruption  
   Since the system didn’t unmount disks properly, there may be incomplete writes left in memory. This can damage file system structures, forcing fsck checks at the next boot.
2. Data loss  
   Any unsaved data sitting in cache or RAM will be lost. Databases or applications may lose recent transactions or need recovery.
3. Service disruptions  
   Critical services (like web servers, databases, mail servers) stop abruptly. On restart, they may take extra time to recover or rebuild indexes.
4. Hardware stress  
   Repeated sudden power cuts can affect storage devices (especially HDDs and SSDs) because heads or flash controllers don’t get a chance to park or flush data.
5. Security and logs  
   Important security logs or audit trails may not get written fully, creating gaps in monitoring or compliance reports.

**Linux Programming: Assignment-2**

**NAME-K.V.S Tanveesh USN:ENG24CY0119 Roll:19 SEC-CY(C)**

**Q1) What does the command pwd, whoami, and hostname display?**

1. **pwd (Print Working Directory):**  
   This command shows the exact path of the directory where we are currently working. For example, if we are inside a folder called *Documents* under your user account, it will display /home/username/Documents. It helps the user know their present location in the Linux file system.
2. **whoami:**  
   This command prints the username of the person who is logged in. It is useful when multiple users are working on the same machine or when switching between accounts.
3. **hostname:**  
   This command displays the name given to the computer (the host) in the network. For example. It helps in identifying the machine especially when it is connected in a network with many other systems.

**Q2) Write the command to create a directory named “project” inside the /home/student folder and keep three .txt file into it. Give output snapshot.**

Step 1: Create the directory

bash --- mkdir /home/student/project

This makes a new folder called project inside /home/student.

Step 2: Move into that directory

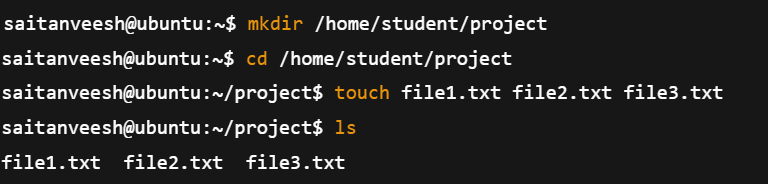
Bash---- cd /home/student/project

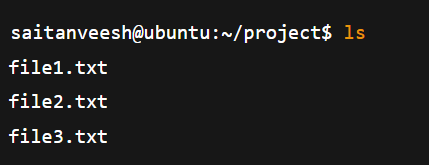
Now you are inside the project folder.

Step 3: Create three text files

Bash--- touch file1.txt file2.txt file3.txt

This will create three empty text files in one command.





**Q3) Explain the difference between absolute path and relative path with proper examples.**

1. Absolute Path
   1. Definition: An absolute path is the complete path of a file or directory starting from the root /.
   2. It always begins with / and shows the full location in the file system, no matter where you currently are.

Example:

* + 1. /home/saitanveesh/project/file1.txt
    2. Here, it starts from the root / → home → saitanveesh → project → file1.txt.

Use case: Useful when you want to be sure of the exact location of a file, independent of the current working directory.

1. Relative Path
   1. Definition: A relative path is the location of a file or directory relative to the directory you are currently working in.
   2. It does not start with /. Instead, it uses . (current directory) or .. (parent directory).

Example:

* + 1. Suppose your current location is /home/saitanveesh.
    2. The relative path to file1.txt inside the project folder will be:  
       project/file1.txt
    3. To go one level up: ../Downloads/file2.txt

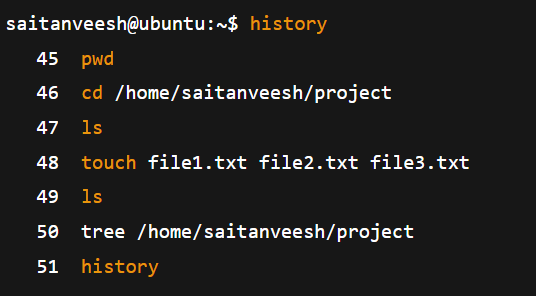
|  |  |
| --- | --- |
| **Absolute Path** | **Relative Path** |
| 1. Always starts with / | 1. Does not start with / |
| 1. Complete location from root | 1. Location depends on where you are (current directory) |
| 1. Works anywhere in the system | 1. Only works correctly if you are in the expected directory |
| 1. Example: /home/saitanveesh/project/file1.txt | 1. Example: project/file1.txt |

**Q4) What command will give you the already executed command traces in the terminal. Give output snapshot**

In Linux, the command used is:

Bash programming command --------- history

1. It shows a numbered list of all the commands that have been executed in the current shell session (and also from previous sessions, depending on settings).
2. You can re-run a command directly using !<number> from the history list.



**Q5) Compare the working functionality of find and locate command. Which one is faster and why?**

**ANS--- 1.find command**

* 1. Functionality: Searches for files and directories by walking through the filesystem in real time.
  2. It can search by name, type, size, permission, owner, date, etc.
  3. Works even if the file was just created a few seconds ago.
  4. Example:
  5. find /home/saitanveesh -name "file1.txt"

→ This will search the entire /home/saitanveesh directory for file1.txt.

**2.locate command**

* 1. Functionality: Searches files by checking a pre-built database (mlocate.db) instead of scanning the disk live.
  2. Faster because it does not look at the actual filesystem each time.
  3. But if the database is not updated (sudo updatedb), newly created files may not appear.
  4. Example:
  5. locate file1.txt

→ This will instantly display the path of file1.txt (if it is already recorded in the database).

**Q6)Which command is used to modify file permissions in Linux? Give an example.**

* The command used is chmod (change mode).
* It allows you to set or change the read (r), write (w), and execute (x) permissions for the owner, group, and others on a file or directory.

Example 1: Using symbolic method

chmod u+x file1.txt

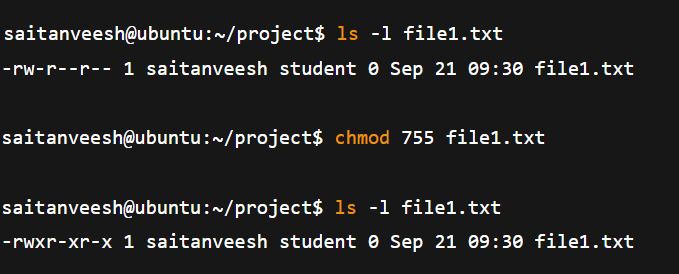
This gives the user (owner) execute permission on file1.txt.

Example 2: Using numeric (octal) method

chmod 755 file1.txt

This sets permissions as:

* 7 → owner = read + write + execute (rwx)
* 5 → group = read + execute (r-x)
* 5 → others = read + execute (r-x)

****

**Q7) A file has permissions -rw -r- -r- -. What does this mean ?**

In Linux, file permissions are shown in 10 characters. It means like this:

Bash command is -rw-r--r--

1. First character (-)
   1. Shows the type of file.
   2. - = regular file
2. Next 3 characters (rw-) → Owner (User) permissions
   1. r = read allowed
   2. w = write allowed
   3. - = no execute permission
3. Next 3 characters (r--) → Group permissions
   1. r = read allowed
   2. -- = no write, no execute
4. Last 3 characters (r--) → Others (world) permissions
   1. r = read allowed
   2. -- = no write, no execute

**Q8) Explain the difference between chown and chgrp with an example ?**

1. chown (Change Owner)
   1. Used to change the owner of a file or directory.

Syntax:

chown new\_owner filename

Example:

* + sudo chown saitanveesh file1.txt

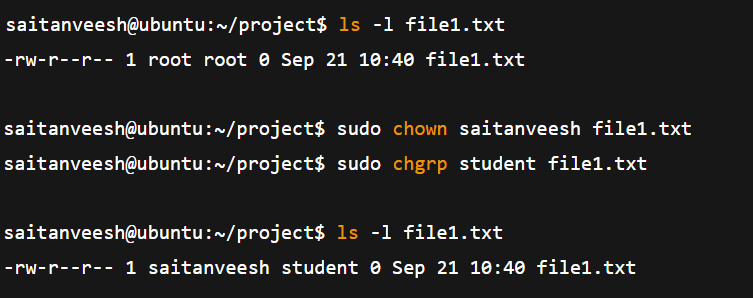
1. chgrp (Change Group)
   1. Used to change the group ownership of a file or directory.

Syntax:

chgrp new\_group filename

Example:

* + sudo chgrp student file1.txt



**Q9) A file needs to be accessible by multiple users but only writable by the owner. How will you set permissions?**

Requirement:

* 1. Owner → should have *read and write* access.
  2. Group & Others (all users) → should have *read-only* access.
  3. No one except the owner should be able to modify the file.

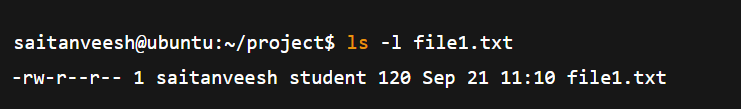
Using chmod (symbolic method)

**Bash command---- chmod 644 filename**

1. 6 → Owner = read (4) + write (2) = rw-
2. 4 → Group = read only = r--
3. 4 → Others = read only = r--

final permission will be:

**-rw-r--r—**

****

**10. How do you check the manual page for any Linux commands?.**

1. In Linux, almost every command has a manual (man) page that explains what the command does, its syntax, options, and examples.
2. To open the manual page, we use the man command followed by the command name.

**Syntax:**

**man <command>**

**Example:**

**man ls**

now it will open the manual page for the ls command, showing:

1. Description of the command
2. Usage / syntax
3. Available options (like -l, -a, -h)
4. Exit status and related commands

**Linux Programming: Assignment-3**

**NAME-K.V.S Tanveesh USN:ENG24CY0119 Roll:19 SEC-CY(C)**

**Q1) Distinguish between man and whatis commands. Justify with proper example.**

1. **man command**
   1. Stands for *manual*. It shows the complete manual page of a command, including description, syntax, options, examples, and related commands.
   2. Example: man ls

This will open the manual page for ls, showing all available options like -l, -a, -h etc.

1. **whatis command**
   1. Gives only a one-line short description of the command.

Example: whatis ls

Output:

ls (1) - list directory contents

Difference:

man = full manual (detailed explanation).

whatis = short one-line summary.

**Q2) Use the tee command to save the output of ls -l into a file while also displaying it.**

* The tee command is used when we want to see the output on the screen *and* save it into a file at the same time.

Example: ls -l | tee output.txt

This will:

1. Display the result of ls -l on the terminal.
2. Save the same output into a file called output.txt.

If you want to append (add) instead of overwriting:

ls -l | tee -a output.txt

**Q3) Explain with an example how the tee command can be used in logging.**

The tee command is special in Linux because it can do two things at once:

1. It shows the output of a command on the screen (standard output).
2. It also saves the same output into a file at the same time.

This makes it very useful for **logging purposes**, because you don’t lose the live output while still keeping a permanent record of it.

**Example:**

Suppose you want to check the network connectivity by running a ping command:

ping -c 5 google.com | tee pinglog.txt

1. Here, the output of the ping will be printed on the screen so you can watch it live.
2. At the same time, the same result will be written into a file called pinglog.txt.
3. Later, you can open that file with cat pinglog.txt or any editor to check the details.

**Why is tee important for logging?**

1. It avoids losing important information.
2. Useful in **debugging scripts, troubleshooting, or auditing** system changes.
3. Helps when you need both *real-time monitoring* and *saved records*.

**Q4) List the steps involved in installing Ubuntu 25.04 LTS on Oracle VirtualBox.**

Installing Ubuntu inside VirtualBox is basically like setting up a computer inside existing computer.

**Step 1: Download Ubuntu ISO**

1. Go to the official [Ubuntu website](https://ubuntu.com/download).
2. Download the ISO image of **Ubuntu 25.04 LTS (64-bit)**.
3. Save it somewhere you can find easily (like Downloads).

**Step 2: Create a New Virtual Machine in VirtualBox**

1. Open Oracle VirtualBox and click on **“New.”**
2. Give a name → Ubuntu25.04.
3. Type = **Linux**, Version = **Ubuntu (64-bit)**.
4. Assign **RAM** (at least 4GB, better 8GB if system allows).
5. Assign **CPU cores** (2 or more).

**Step 3: Create a Virtual Hard Disk**

1. Choose **VDI (VirtualBox Disk Image)**.
2. Storage type → Dynamically allocated.
3. Disk size → Minimum 25GB

**Step 4: Mount the Ubuntu ISO**

1. Go to **Settings → Storage**.
2. Under the optical drive, select the Ubuntu ISO you downloaded.
3. This makes VirtualBox boot from the ISO as if it were a DVD.

**Step 5: Boot the Virtual Machine**

1. Click **Start**.
2. The VM will boot from the Ubuntu ISO.
3. Choose **“Install Ubuntu”**

**Step 6: Installation Wizard**

1. Select **Language** and **Keyboard layout**.
2. Choose **Normal Installation**
3. Partitioning: Pick **Erase disk and install Ubuntu**
4. Set up **username and password** → e.g. saitanveesh.

**Step 7: Installation Process**

1. The installer will copy files and install Ubuntu inside the virtual hard disk you created.
2. Once finished, it will ask you to restart.
3. **Important:** When rebooting, remove the ISO from the virtual drive, otherwise it will boot into the installer again.

**Step 8: Post-Installation Steps**

1. Log in with your username and password.
2. Update the system:

sudo apt update && sudo apt upgrade

**Q5) During Ubuntu OS installation, you face a Kernel Panic Error. How would you troubleshoot it?**

A **Kernel Panic** means the Linux kernel has encountered a critical error and cannot continue safely. During Ubuntu installation in VirtualBox,

**Step 1: Verify the ISO file**

1. Sometimes the downloaded ISO is incomplete or corrupted.
2. Check the file’s **checksum (SHA256)** with the one given on the Ubuntu website.
3. If it doesn’t match → re-download the ISO.

**Step 2: Recreate Bootable Media**

1. If installing on real hardware, remake the bootable USB using tools like Rufus (Windows) or dd (Linux).
2. If inside VirtualBox, just re-attach the correct ISO image.

**Step 3: Adjust VirtualBox Settings**

1. Make sure virtualization is enabled in BIOS/UEFI (**Intel VT-x** or **AMD-V**).
2. In VM settings →
   1. **System → Processor:** Allocate at least 2 CPUs.
   2. **System → Motherboard:** Enable I/O APIC.
   3. **Display:** Increase Video Memory to 128MB.

**Step 4: Boot with Safe Options**

1. When the GRUB menu appears, press e to edit boot parameters.
2. Add options like:
   1. nomodeset → bypass graphics driver issues.
   2. acpi=off → disable advanced power features if causing panic.
3. Then continue booting with these safe parameters.

**Step 5: Try an Older Kernel or Recovery Mode**

1. Sometimes the newest kernel doesn’t play well with VirtualBox or specific hardware.
2. If installer gives the option, select a different kernel or recovery mode.
3. If you can reach a shell, view logs using:

dmesg | less

**Q6) Write the command to display the system’s hostname. How to change hostname using sysctl command?**

**1. Displaying the hostname**

* The **hostname** command is used to show the current hostname of the system.

Example:

hostname

Output might look like:

saitanveesh-pc

This is the name by which the machine identifies itself in a network.

**2. Changing the hostname using sysctl**

1. The hostname is controlled by the kernel parameter kernel.hostname.
2. You can temporarily change it using the sysctl command:

sudo sysctl kernel.hostname=newname

Example:

sudo sysctl kernel.hostname=ubuntu-server

**Important Note:**

1. This change is **temporary**. Once you reboot, the hostname will go back to the old one.
2. To make it **permanent**, you must edit the file /etc/hostname and also update /etc/hosts.

sudo hostnamectl set-hostname newname

**Q7) Which command is used to show the calendar of the year 1984 with August month?**

In Linux, the **cal command** is used to display a calendar in the terminal.

**1. Show a specific month of a year**

The syntax is:

cal <month> <year>

For August 1984, the command will be:

cal 8 1984

**Example Output (August 1984)**

August 1984

Su Mo Tu We Th Fr Sa

1 2 3 4

5 6 7 8 9 10 11

12 13 14 15 16 17 18

19 20 21 22 23 24 25

26 27 28 29 30 31

**Here**:

1. Su Mo Tu We Th Fr Sa = Days of the week (Sunday to Saturday).
2. Dates are arranged under each day.

**2. Show the whole year 1984**

If instead of one month you want the entire year’s calendar:

cal 1984

This will print all 12 months of the year 1984 together.

**Q8) Write a command to display system uptime and logged-in users together.**

In Linux, there are different commands that give you both uptime and the users who are currently logged in.

**1. Using the w command**

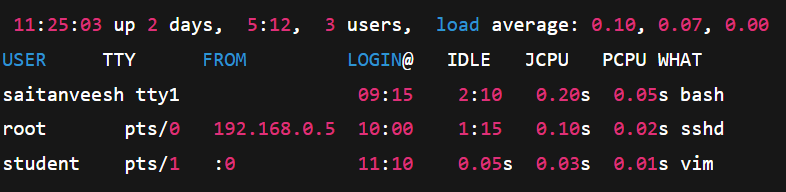
The simplest way is:

w

This command shows:

1. The current time
2. How long the system has been up (uptime)
3. List of logged-in users with what they are doing

**Sample Output:**

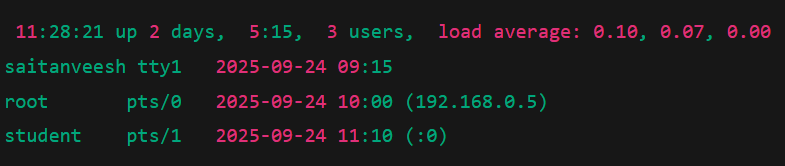


**2. Using uptime and who together**

You can also combine two commands:

uptime && who

1. uptime → shows system uptime, number of users, and load average.
2. who → shows list of logged-in users.

**Sample Output:**

**Q9) Use the find command to list all “.c” files in /home/user.**

The **find command** is one of the most powerful tools in Linux for searching files and directories. It scans through directories and lists files that match the conditions you specify.

**Syntax:**

find <path> -name <pattern>

**For this question:**

To find all .c files inside /home/user:

find /home/user -name "\*.c"

Explanation:

1. /home/user → the starting directory where the search begins.
2. -name "\*.c" → tells find to look for files ending with .c (C language source code files).
3. The \* (wildcard) means "match any filename ending with .c".

**Sample Output (example)**

**/home/user/project1/main.c**

**/home/user/project1/utils/helper.c**

**/home/user/project2/test.c**

**Final Answer**

1. Command: find /home/user -name "\*.c"
2. It will list all C source files in /home/user and its subdirectories.