NAME SHREYA

ROLL NO 09

USN ENG24CY0051

SECTION C

**Linux Programming: Assignment-4**

**1.A system has a file /etc/passwd. How would you use grep + tee to extract usernames and save them to a file while also displaying them on screen?**

The file /etc/passwd has entries like this:

root:x:0:0:root:/root:/bin/bash

Shreya:x:1000:1000:Shreya:/home/Shreya:/bin/bash

* The **username** is the first field (before the first :).
* cut -d: -f1 normally extracts it, but since you asked about grep + tee, here’s how:

**Command:**

grep -o '^[^:]+' /etc/passwd | tee usernames.txt

**Explanation:**

* grep -o '^[^:]+' /etc/passwd
  + ^ → start of line
  + [^:]+ → one or more characters that are not : (so everything up to the first colon)
  + -o → print only the matched part (the username).
* tee usernames.txt
  + saves the output into usernames.txt **and** displays it on screen.

If you wanted to append instead of overwrite:

grep -o '^[^:]+' /etc/passwd | tee -a usernames.txt

**2. A binary isn’t found in $PATH. How would you use commands (which, find, locate) to troubleshoot and fix the issue?**

**1. Check if the command is in $PATH with which**

which mybinary

* If it prints a path (e.g. /usr/bin/mybinary), the binary is installed and in your $PATH.
* If it prints nothing, the shell can’t find it in the directories listed in $PATH.

**2. Search the filesystem with find**

If which fails, try locating it manually:

sudo find / -type f -name mybinary 2>/dev/null

* Searches the entire filesystem for mybinary.
* Redirecting errors (2>/dev/null) hides "permission denied" noise.
* If found, note the directory (e.g., /opt/myapp/bin/mybinary).

**3. Use locate for faster search**

locate mybinary

* Uses a prebuilt index of files (much faster than find).
* If results seem outdated, run:

sudo updatedb

locate mybinary

**4. Fix the issue**

Once you find the binary:

* **Option A:** Add its directory to your $PATH temporarily:

export PATH=$PATH:/opt/myapp/bin

* **Option B:** Make it permanent by adding the line to your shell config (~/.bashrc, ~/.zshrc):

echo 'export PATH=$PATH:/opt/myapp/bin' >> ~/.bashrc

source ~/.bashrc

* **Option C:** Move or symlink it to a standard directory already in $PATH, like /usr/local/bin:

sudo ln -s /opt/myapp/bin/mybinary /usr/local/bin/mybinary

**3. Write a command pipeline that finds all .log files modified in the last 24 hours in /var/log and saves results into log\_report.txt.**

find /var/log -type f -name "\*.log" -mtime -1 | tee log\_report.txt

**Breakdown:**

* find /var/log → start searching in /var/log.
* -type f → only files (skip dirs).
* -name "\*.log" → restrict to .log files.
* -mtime -1 → modified within the last 24 hours.
* | tee log\_report.txt → write output to log\_report.txt **and** display it on the screen.

If you want to **append** instead of overwrite:

find /var/log -type f -name "\*.log" -mtime -1 | tee -a log\_report.txt

**4. What is the difference between shutdown -r now and reboot?**

**shutdown -r now**

* shutdown is a **controlled system shutdown tool**.
* -r → means **reboot after shutdown**.
* now → means **start immediately** (you could also say +5 for 5 minutes later).
* It **notifies logged-in users** (via wall messages) and gracefully stops services before rebooting.
* Example:
* sudo shutdown -r now

**reboot**

* A **shortcut command** to reboot the system.
* It calls the system’s reboot syscall (via systemctl reboot on modern systems).
* Does **not send warnings or allow scheduling** (though clean service stop still happens).
* Example:
* sudo reboot

**Key Difference**

* shutdown -r now = **graceful, schedulable, user-aware reboot**.
* reboot = **immediate reboot** (faster, but less user-friendly).

On **modern systemd systems**, both end up calling systemctl reboot, so the difference is smaller — but shutdown still allows **scheduling and notifications**.

**5. How can you use the tee command to debug a script that generates both standard output and error messages?**

**Case 1: Capture only stdout**

./myscript.sh | tee debug.log

* Saves normal output (stdout) into debug.log **and** displays it on screen.
* Error messages (stderr) still go only to the screen.

**Case 2: Capture stdout + stderr together**

./myscript.sh 2>&1 | tee debug.log

* 2>&1 redirects file descriptor **2 (stderr)** to **1 (stdout)**.
* Now both outputs go through tee.
* debug.log will contain **everything** (stdout + stderr).

**Case 3: Capture them into separate files**

./myscript.sh > >(tee stdout.log) 2> >(tee stderr.log >&2)

* stdout.log → only standard output.
* stderr.log → only errors.
* You still see both on your terminal.

**6. Explain any three real-world applications of Linux in industries.**

**Web Servers & Hosting**

* **Where used**: Companies like Google, Facebook, Amazon, and most web hosting providers.
* **Why Linux**:
  + Stable and secure for 24/7 uptime.
  + Runs Apache, Nginx, MySQL, PHP, etc. efficiently.
  + Open-source (no licensing costs).
* **Example**: Over **70% of web servers** on the internet run Linux-based operating systems.

**2. Embedded Systems & IoT**

* **Where used**: Smart TVs, routers, drones, cars (infotainment), medical devices.
* **Why Linux**:
  + Lightweight and customizable (e.g., Yocto, Buildroot).
  + Real-time capabilities for devices.
  + Open-source drivers and community support.
* **Example**: **Android** (built on the Linux kernel) powers billions of smartphones and IoT devices.

**3. Supercomputing & Scientific Research**

* **Where used**: NASA, CERN, weather forecasting centers, AI research labs.
* **Why Linux**:
  + Scales easily to thousands of processors.
  + Supports high-performance computing (HPC) clusters.
  + Reliable under heavy workloads.
* **Example**: **All of the world’s top 500 supercomputers** (as of 2025) run on Linux.

**7. Differentiate application, system and utility software in the context of Linux environment.**

**System Software**

* **Definition**: Core software that manages hardware and provides a platform for other programs.
* **In Linux**: The **Linux kernel + operating system components** (device drivers, memory manager, process scheduler).
* **Example**: Linux kernel, systemd, device drivers.

**2. Application Software**

* **Definition**: End-user programs designed to perform specific tasks.
* **In Linux**: Installed by users to achieve goals like browsing, editing, coding, etc.
* **Example**: Firefox (web browser), LibreOffice (office suite), GIMP (image editor).

**3. Utility Software**

* **Definition**: Small programs that support system maintenance, monitoring, or configuration.
* **In Linux**: Enhance performance, troubleshoot, or manage resources.
* **Example**: grep, top, ls, tar, rsync.

**Key Difference Table**

| **Category** | **Purpose** | **Linux Examples** |
| --- | --- | --- |
| **System Software** | Controls hardware & core OS functions | Linux kernel, systemd, drivers |
| **Application** | Provides user-level functionality | Firefox, LibreOffice, VLC |
| **Utility** | Maintenance, monitoring, support tasks | grep, top, tar, rsync, fdisk |

**8. What are the key differences between open-source and proprietary operating systems?**

Here’s a clear comparison of **open-source vs proprietary operating systems** (with Linux in mind):

**Key Differences**

| **Aspect** | **Open-Source OS** | **Proprietary OS** |
| --- | --- | --- |
| **Source Code** | Freely available; anyone can view, modify, and distribute. | Closed; only the vendor has access. |
| **Cost** | Usually free or low cost. | Generally requires purchase or licensing fees. |
| **Customization** | Highly customizable (users can tweak kernel, UI, features). | Limited customization; restricted to vendor settings. |
| **Support** | Community-driven (forums, wikis, contributors); optional paid support from vendors like Red Hat. | Official vendor support (helpdesk, updates, patches). |
| **Security** | Transparent code → faster vulnerability detection; patches from community. | Relies on vendor for security fixes; slower response possible. |
| **Examples** | Linux (Ubuntu, Fedora), FreeBSD, Android. | Windows, macOS, Solaris. |

**9. Write the command to display the system’s kernel version.** (CO3)

You can check the **Linux kernel version** with:

uname -r

**Other useful variants:**

* Full system info (including kernel):
* uname -a
* Kernel version details from /proc:
* cat /proc/version
* On systemd-based systems:
* hostnamectl | grep Kernel

The most common and exam-friendly answer is:

uname -r

```

**10. What is the difference between head and tail commands in text processing?**

**Here’s the clear distinction between head and tail in Linux text processing:**

**1. head**

* **Purpose**: Displays the **first part** of a file (by default, the first 10 lines).
* **Example**:
* head /etc/passwd
  + Shows the first 10 lines of /etc/passwd.
* **Options**:
  + -n N → show first N lines, e.g., head -n 5 file.txt.
  + -c N → show first N bytes.

**2. tail**

* **Purpose**: Displays the **last part** of a file (by default, the last 10 lines).
* **Example**:
* tail /var/log/syslog
  + Shows the last 10 lines of the syslog.
* **Options**:
  + -n N → show last N lines, e.g., tail -n 20 file.txt.
  + -f → follow the file in real-time (useful for logs).

**Key Difference**

| **Command** | **Shows** | **Default lines** | **Special feature** |
| --- | --- | --- | --- |
| **head** | Beginning of file | 10 | Useful to preview files |
| **tail** | End of file | 10 | -f option for live updates |