DAYANANDA SAGAR UNIVERSITY

LINUX PROGRAMMING

ASSIGNMENTS-4

NAME : AKHIL VENKAT D

USN : ENG24CY0079

ROLL NO :46

SEACTION : A

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?

Step 1: Extract Usernames

In /etc/passwd, each line looks like:

```
username:x:UID:GID:comment:home:shell
```

The **username** is the first field, separated by :.

You can use cut or awk to extract it. Example with cut:

```
cut -d: -f1 /etc/passwd
```

Step 2: Display and Save Using tee

Combine with tee to display on screen **and** save to a file:

```
cut -d: -f1 /etc/passwd | tee usernames.txt
```

Explanation

- cut -d: -f1 → extracts the first field (username) from each line.
- | → pipes the output to the next command.
- tee usernames.txt → displays output on screen **and writes to usernames.txt**.

✅ Example output on screen:

```
root
```

```
daemon
alice
bob
```

The same output is saved in usernames.txt.

Alternative using awk

```
awk -F: '{print $1}' /etc/passwd I tee usernames.txt
```

- -F: → sets field separator as :
- $1 → prints first field (username)

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

Step 1: Check if the command exists in $PATH

```
which binary_name
```

- Example:

```
which python3
```

- If it returns nothing → the shell cannot find it in the directories listed in $PATH.

Step 2: Search for the binary manually

Option A: Using find

```
sudo find / -type f -name binary_name 2>/dev/null
```

- / → search from root directory
- -type f → only files
- 2>/dev/null → suppress permission denied errors
- Example:

```
sudo find / -type f -name python3 2>/dev/null
```

Option B: Using locate

```
locate binary_name
```

- Very fast because it uses a prebuilt database.
- If the database is outdated, update it first:

```
sudo updatedb
```

- Example:

```
locate python3
```

Step 3: Add the binary's directory to $PATH

Suppose find shows the binary at /usr/local/bin/python3.

- Temporarily add to $PATH:

```
export PATH=$PATH:/usr/local/bin
```

- To make it permanent, add the line to ~/.bashrc or ~/.profile:

```
echo 'export PATH=$PATH:/usr/local/bin' >> ~/.bashrc
source ~/.bashrc
```

Step 4: Verify

```
which python3
```

- Should now show the full path.

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.

You can achieve this using the **find** command combined with **tee** (or output redirection) to save results.

Command:

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

Explanation:

- /var/log → directory to search in.
- -type f → only regular files.
- -name "*.log" → files ending with .log.
- -mtime -1 → modified in the last 1 day (24 hours).
- | tee log_report.txt → displays the results on screen **and saves to log_report.txt**.

Alternative using -exec ls -l for detailed info

```
find /var/log -type f -name "*.log" -mtime -1 -exec ls -l {} \; | tee log_report.txt
```

- Shows **permissions, size, and modification time** along with filenames.

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

1. shutdown -r now

- shutdown is a versatile command to **power off or reboot** the system.
- -r → tells it to **reboot** instead of shutting down.
- now → execute **immediately**.
- Behavior:
  - Sends a **notification to all logged-in users**.
  - Stops all processes **gracefully**, unmounts filesystems, then reboots.

Example:

```
sudo shutdown -r now
```

---

2. reboot

- reboot is a **simpler, direct command** to restart the system.
- Internally, it calls the **shutdown system call**, so it also stops processes and reboots.
- Often **faster** than shutdown -r now because it may skip some user notifications depending on system configuration.

Example:

```
sudo reboot
```

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

Basic Idea

- > → redirects stdout
- 2> → redirects stderr
- 2>&1 → redirects stderr to stdout
- Pipe everything to tee to save and display simultaneously

Command

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

Explanation

1. ./myscript.sh → runs your script.
2. 2>&1 → merges stderr (2) into stdout (1), so both streams go through the pipe.
3. | tee debug.log →
   - **Displays output on the screen** (real-time debugging).
   - **Saves output to debug.log** for later analysis.

Example

Suppose myscript.sh contains:

```
echo "Starting script..."

ls /nonexistent

echo "Script finished."
```

Running:

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

Output on screen and in debug.log:

```
Starting script...

ls: cannot access '/nonexistent': No such file or directory

Script finished.
```

Optional Variants

1. **Append to log instead of overwriting:**

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

2. **Separate stdout and stderr into different files (advanced):**

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

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

1. Web Servers and Cloud Computing

- **Use case:** Hosting websites, applications, and cloud services.
- **Example:**

    - Companies like **Google, Amazon (AWS), and Facebook** run Linux servers to handle massive web traffic.
    - Popular web servers like **Apache** and **Nginx** run primarily on Linux.
- **Why Linux?**
    - Stability and uptime for 24/7 services.
    - Security and flexibility for server management.
    - Open-source, cost-effective for large-scale deployments.

---

2. Embedded Systems and IoT Devices

- **Use case:** Operating system for devices with limited resources.
- **Example:**
    - **Smart TVs, routers, automotive infotainment systems, and smart appliances** often run Linux or Linux-based variants like **Android (Linux kernel)**.
- **Why Linux?**
    - Lightweight and customizable to hardware.
    - Supports a wide range of processors and architectures.
    - Large developer community and ready-made drivers.

---

3. Supercomputing and Scientific Research

- **Use case:** High-performance computing (HPC) for simulations, data analysis, and research.
- **Example:**
    - **Most of the top 500 supercomputers** (like those in CERN or NASA) run Linux.
    - Used in **weather modeling, genome analysis, and AI research**.
- **Why Linux?**
    - Open-source nature allows optimization for high-speed computing.
    - Excellent support for parallel computing and clustering.
    - Stability for long-running computations

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

1. Application Software

- **Definition:** Programs designed to perform **specific tasks** for the user.
- **Purpose:** Solve user problems or provide functionality like editing, browsing, or gaming.
- **Examples in Linux:**
    - Web browsers → Firefox, Chrome
    - Office suite → LibreOffice
    - Media players → VLC, Rhythmbox
- **Key Points:**
    - User-oriented.
    - Not essential for system operation.

---

2. System Software

- **Definition:** Software that **manages and controls hardware**, providing a platform for running application software.
- **Purpose:** Ensures smooth operation of hardware and system resources.
- **Examples in Linux:**
    - Linux kernel → core of the OS
    - System daemons → systemd, cron
    - Device drivers → for printers, graphics cards, network interfaces
- **Key Points:**
    - Essential for the OS to function.
    - Operates in the background.

---

3. Utility Software

- **Definition:** Programs that **perform maintenance and optimization tasks** on the system.
- **Purpose:** Improve efficiency, manage files, monitor performance, and troubleshoot.
- **Examples in Linux:**
    - File management → cp, mv, rm, ls
    - Disk management → df, du, fsck
    - System monitoring → top, htop, uptime

- **Key Points:**
  - Supports both users and system administrators.
  - Often command-line based in Linux.

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

1. Definition

- **Open-source OS:**
  - The **source code is freely available** to anyone.
  - Users can **view, modify, and distribute** it.
  - Example: **Linux, FreeBSD**
- **Proprietary OS:**
  - The **source code is closed** and owned by a company.
  - Users can **only use it under license**; modification or redistribution is prohibited.
  - Example: **Windows, macOS**

---

2. Cost

Feature	Open-Source OS	Proprietary OS
Licensing	Usually free	Paid or license-based
Updates	Free, community-supported	Often paid, may require subscriptions

---

3. Customization

- **Open-source:** Highly customizable; you can modify kernel, add/remove features.
- **Proprietary:** Limited or no customization; users depend on vendor updates.

---

4. Support & Community

- **Open-source:**
  - Supported by **community forums**, wikis, and volunteers.
  - Example: Ubuntu forums, Stack Exchange.

- **Proprietary:**
    - Supported by **official company support**, helpdesks, or premium plans.

---

5. Security

- **Open-source:**
    - Security issues can be quickly patched by the community.
    - Transparent source code allows auditing.
- **Proprietary:**
    - Security patches depend on vendor release cycles.
    - Source code not visible for auditing.

---

6. Examples

Open-Source OS	Proprietary OS
Linux (Ubuntu, Fedora)	Windows 10/11
FreeBSD	macOS
Android (AOSP)	iOS

9. Write the command to display the system's kernel version.

1. Using uname

```
uname -r
```

- -r → shows the **kernel release version**.

  ✅ Example output:

```
5.19.0-46-generic
```

---

2. Using uname -a (full system info)

```
uname -a
```

- Displays kernel version, hostname, architecture, and more.
  ✅ Example output:

```
Linux ubuntu-pc 5.19.0-46-generic #47-Ubuntu SMP Fri Sep 8 12:00:00 UTC 2025 x86_64
x86_64 x86_64 GNU/Linux
```

---

3. Using /proc/version

```
cat /proc/version
```

- Reads kernel version directly from the **proc filesystem**.
  ✅ Example output:

```
Linux version 5.19.0-46-generic (buildd@ubuntu) (gcc version 12.2.0) #47-Ubuntu SMP Fri
Sep 8 12:00:00
```

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

1. head Command

- **Purpose:** Displays the **first part of a file** (top lines).
- **Default behavior:** Shows the **first 10 lines**.
- **Syntax:**

```
head filename
head -n 5 filename   # first 5 lines
```

- **Example:**

```
head /var/log/syslog
```

Shows the **beginning of the file**.

---

2. tail Command

- **Purpose:** Displays the **last part of a file** (bottom lines).
- **Default behavior:** Shows the **last 10 lines**.
- **Syntax:**

```
tail filename
tail -n 5 filename   # last 5 lines
```

- **Example:**

```
tail /var/log/syslog
```

Shows the **end of the file**.

- **Extra feature:**
  tail -f filename → continuously monitors a file in real-time (useful for logs).

Key Differences

Feature	head	tail
Part of file	Beginning (top)	End (bottom)
Default lines	10	10
Use case	Preview start of file	Monitor recent or live updates
Real-time mode	No	Yes (-f)