Assignment-4

Name: Suraj P Das

USN: ENG25CY1006

Section: 3C

Roll no: 66

1. To extract usernames from /etc/passwd using grep and save them to a file while also displaying them on screen, the following command can be used.

   Command:

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

   Explanation:

   - grep " /etc/passwd → prints all lines.
   - cut -d: -f1 → extracts the first field.
   - tee usernames.txt → writes the usernames to usernames.txt while also printing them on the terminal.

2. To troubleshoot and fix a "command not found" error when a binary is not in your $PATH, you can use which, find, and locate as follows:

   1. Verify the binary's presence and location:
      - which: This command checks if the binary is found within any of the directories listed in your $PATH. If the binary is in $PATH, which will output its full path. If not, it will return nothing or an error message.
        Command: which <binary_name>

- find: If which fails, use find to search the entire filesystem for the binary. This is a more exhaustive search.
  Command: find / -name <binary_name> 2>/dev/null
- locate: If find is too slow, locate can be faster if its database is up-to-date. This command searches a pre-built database of files
  Command: locate <binary_name>
- If locate doesn't find it or gives outdated results, you might need to update its database first.
  Command: sudo updated

2. Add the binary's directory to $PATH:
   - Once you have identified the full path to the binary's directory you can add it to your $PATH.
     Command: export PATH=$PATH:/opt/my_app/bin
   - Edit your shell's configuration file (e.g., ~/.bashrc, ~/.zshrc, or ~/.profile) and add the export PATH line.
     Command: echo 'export PATH=$PATH:/opt/my_app/bin' >> ~/.bashrc
   - Then, apply the changes:
     Command: source ~/.bashrc
3. Verify the fix:
   - After adding the path, use which again to confirm that the binary is now found in your $PATH.
     Command: which <binary_name>

3.You can use the find command with -mtime (modification time) and then pipe to tee.

- Here's the pipeline:
  Command: find /var/log -type f -name "*.log" -mtime -1 | tee log_report.txt

Explanation:

- find /var/log → search inside /var/log.
- -type f → only files.
- -name "*.log" → match .log files.
- -mtime -1 → modified in the last 24 hours.
- tee log_report.txt → save results to log_report.txt while also displaying them on screen.

Example output inside log_report.txt:

- /var/log/syslog.log
  /var/log/auth.log
  /var/log/nginx/error.log

4. Difference between shutdown -r now and reboot:

1. shutdown -r now:

- It Tells the system to shut down cleanly and then reboot.
- -r → reboot after shutdown.
- now → means “do it immediately.
- It notifies all logged-in users.
- Sends signals to processes so they can close gracefully.
- Unmounts filesystems safely.

Example:

- Command: sudo shutdown -r now

2.reboot:

- Directly reboots the system.
- Modern Linux systems treat it as a shortcut for:
  Command: systemctl reboot

- It’s faster but may skip some user notifications.
- Still does a clean shutdown (not unsafe), but less flexible compared to shutdown.

5. The tee command can be used to debug a script that generates both standard output (stdout) and error messages (stderr) by capturing both streams to a file while simultaneously displaying them on the terminal. This allows for real-time monitoring and later analysis of the script's behavior.

- Here's how to use tee for this purpose:
  Command: your_script.sh 2>&1 | tee script_output.log

Explanation:

- your_script.sh: This represents the shell script you are debugging.
- 2>&1: This is a crucial redirection that merges the standard error stream into the standard output stream.
- After this, both stdout and stderr will be treated as a single stream of data.
- |:The pipe symbol redirects the combined output of your_script.sh to the standard input of the tee command.
- tee script_output.log:The tee command then reads this combined input. It simultaneously writes the input to its own standard output (which goes to your terminal, allowing you to see the output in real-time) and copies it to the specified file, script_output.log.

6. Here are three strong real-world applications of Linux in industries, explained clearly with examples:

1. Web and Application Servers:

- Use case: Hosting websites, APIs, and enterprise apps.
- Why Linux?
  Stable, secure, and highly customizable.
  Supports powerful server software like Apache, Nginx, MySQL, PHP, Node.js.
  Scales easily for cloud platforms.
- Industry Example:
  Google, Facebook, Amazon all run thousands of Linux-based servers to handle global traffic.

2. Cybersecurity & Networking:

- Use case: Firewalls, intrusion detection systems, penetration testing, VPN servers.
- Why Linux?
  Comes with strong built-in security tools.
  Open-source, so security researchers can inspect and harden it.
  Distros like Kali Linux are industry standards for penetration testing.
- Industry Example:
  Banks and telecom companies use Linux-based firewalls and routers to protect sensitive data.

3. Embedded Systems & IoT:

- Use case: Smart devices, automotive systems, industrial controllers.
- Why Linux?
  Lightweight versions like Yocto, Buildroot, and Android (Linux-based) can run on limited hardware.

Supports a wide variety of CPUs and devices.
Long-term support kernels ensure reliability.

- Industry Example:
  Automobiles (Tesla, BMW) use Linux-based systems for infotainment and automation.
  Smart TVs, routers, and IoT devices often run on Linux kernels.

7. Difference between application, system, and utility software in the context of a Linux environment.

Software Type	Purpose	Examples in Linux Environment	Key Characteristics
Application Software	To perform specific tasks for the user	LibreOffice, Firefox, GIMP, VLC.	User-facing, solves specific problems, runs on top of the operating system.
System Software	To manage and operate computer hardware and provide a platform for applications.	Linux Kernel (/boot/vmlinuz), systemd (init system), device drivers, GNU core utilities	Runs at the core level, interacts directly with hardware, enables communication between hardware.
Utility Software	To perform system maintenance, optimization, or management tasks.	grep (search), tar (archive), top (process monitoring), chmod (permission management)	Usually small programs or commands, often part of GNU core utilities or Linux distributions.

8. Key differences between open-source and proprietary operating systems**:**

Aspect	Open-source OS	Proprietary OS
Source Code	Fully available to the public for viewing, modification, and distribution	Source code is closed — users cannot see, modify, or distribute it.
Cost	Usually free to use, though support or enterprise editions may cost money	Paid licenses, subscriptions, or purchase fees.
Customization	Highly customizable since source code is available. Users and organizations can tailor the OS to their needs.	Limited customization — modifications are restricted to what the vendor provides**.**
Security	Transparent — community can inspect and fix vulnerabilities. Security depends on community vigilance and patch updates.	Vendor-controlled — security updates depend on the vendor; code is not inspectable by the public.
Support	Community-driven support (forums, docs) or paid vendor support.	Vendor-provided support is standard; professional support is usually guaranteed.
Examples	Linux (Ubuntu, Fedora), FreeBSD, OpenSolaris	Microsoft Windows, macOS, iOS, Android.

9. To display the system's kernel version in Linux, use the following command in the terminal:

- Command: uname -r

This command will output the kernel release version, such as 5.15.0-53-generic.

For more detailed information about the system, including the kernel name, hostname, kernel release, kernel version, machine architecture, processor architecture, hardware platform, and operating system, use:

- Command: uname -a

10. The head and tail commands are fundamental utilities in text processing, primarily used in Unix-like operating systems, to view specific portions of a file or input stream. The key difference between them lies in which part of the data they display:

- **Head:**
  This command displays the beginning of a file or input. By default, it outputs the first 10 lines. You can specify a different number of lines using the -n option.
  Command: head filename.txt # Displays the first 10 lines
  head -n 20 filename.txt # Displays the first 20 lines.
- **tail:**
  This command displays the end of a file or input. By default, it outputs the last 10 lines. Similar to head, you can specify a different number of lines using the -n option.
  Command: tail filename.txt # Displays the last 10 lines
  tail -n 20 filename.txt # Displays the last 20 lines
  tail -f logfile.log # Continuously displays new lines added to logfile.log.