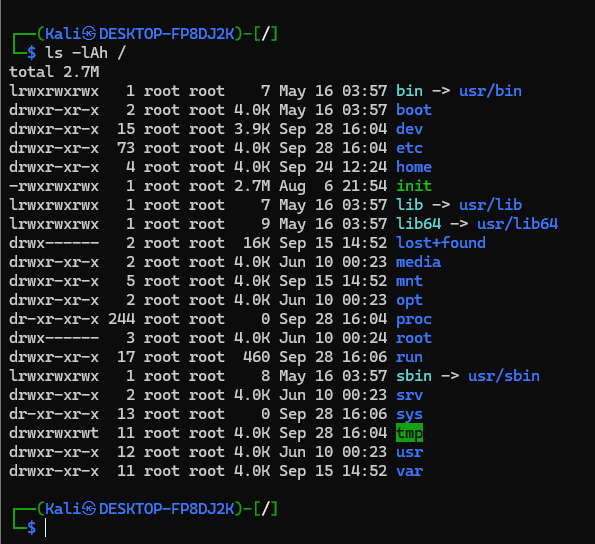
When analyzing a potentially compromised Linux system, each directory under / plays a specific role. Below is how they might be involved in an attack, with commands you can run as proof.

**1) / — Root overview**

**Command:**

ls -lAh /

**Explanation:**  
The root directory is the entry point for all other folders. Listing it confirms the presence of key system directories like /bin, /etc, /var, /usr, /tmp, /opt, /boot, and /home. This gives a quick map of areas that require closer investigation.

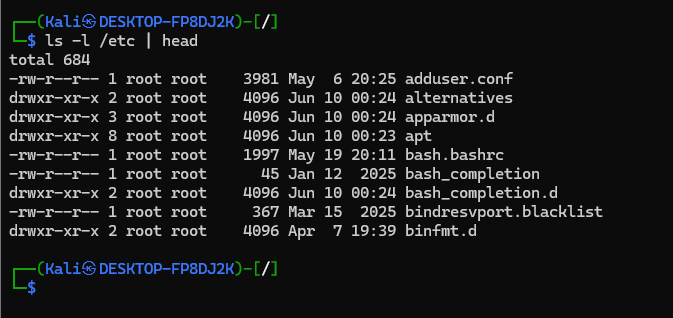


**2) /etc — System configuration files**

**Command:**

ls -l /etc | head

**Explanation:**  
This directory keeps critical system configs. Attackers often edit files like passwd, shadow, or ssh/sshd\_config to create hidden accounts, allow remote logins, or weaken security rules. Evidence of unauthorized changes here usually indicates a compromise.

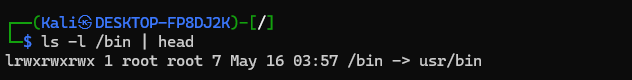


**3) /bin — Core system binaries**

**Command:**

ls -l /bin | head

**Explanation:**  
/bin stores everyday commands (ls, cp, cat). If an attacker replaces one of these binaries with a trojaned version, users might unknowingly run malicious code. Inspecting /bin ensures that essential tools haven’t been tampered with.

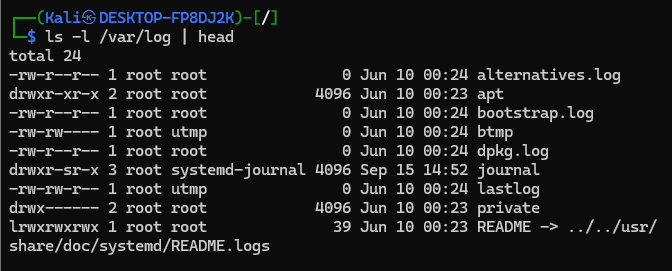


**4) /var/log — System and service logs**

**Command:**

ls -l /var/log | head

**Explanation:**  
The /var/log directory is where system and service logs are written. Authentication attempts, sudo usage, and kernel events are all recorded here. Intruders may erase or alter logs to hide traces, so checking this location is vital for forensic evidence.

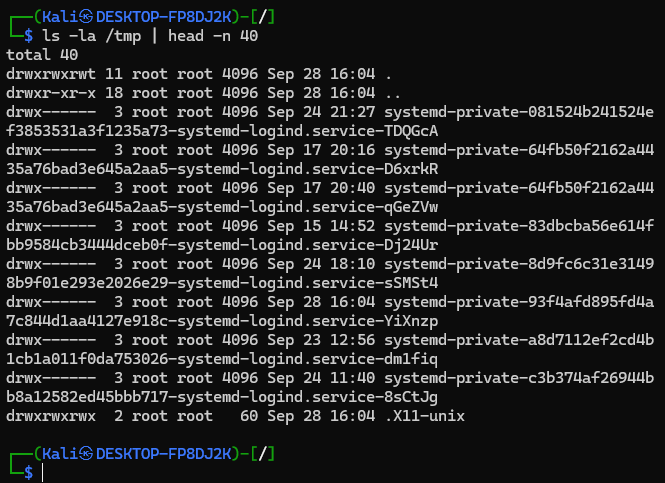


**5) /tmp — Temporary working space**

**Command:**

ls -la /tmp | head -n 40

**Explanation:**  
This directory is world-writable, meaning anyone can create files there. Attackers exploit /tmp to drop payloads or run scripts since permissions are usually open. Suspicious executables or scripts in /tmp are often a sign of intrusion.

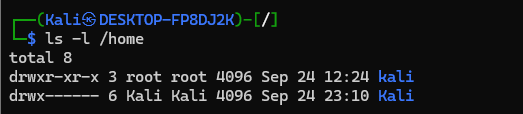


**6) /home — User directories**

**Command:**

ls -l /home

**Explanation:**  
Each user has a home directory under /home. Hackers may add SSH keys to ~/.ssh/authorized\_keys or modify hidden startup files like .bashrc to maintain persistence. Checking /home helps spot accounts that might have been targeted.

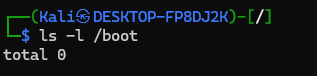


**7) /boot — Bootloader and kernel files**

**Command:**

ls -l /boot

**Explanation:**  
/boot contains the kernel image and GRUB configuration. While not as commonly attacked, a skilled intruder could backdoor the kernel or tamper with the bootloader to keep access after reboots. It’s less frequent but extremely critical if compromised.

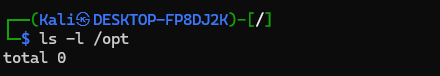


**8) /opt — Optional software installations**

**Command:**

ls -l /opt

**Explanation:**  
This is where third-party or custom applications are installed. Since it’s not as closely monitored as /bin or /usr, attackers might hide tools here. Inspecting /opt ensures that no unfamiliar software was dropped by an attacker.



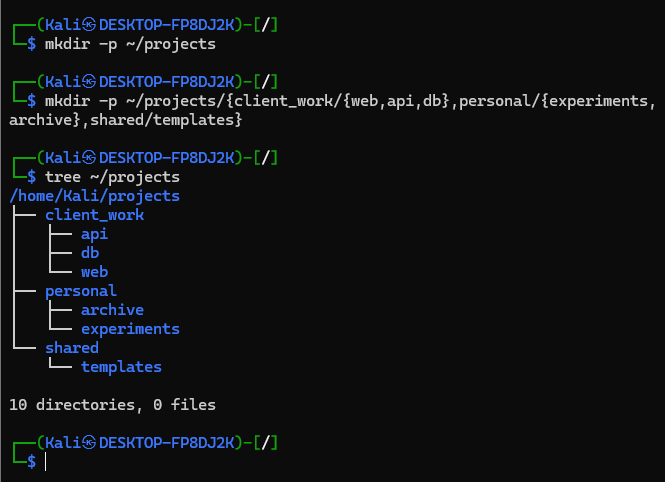
To create the structure efficiently, I used mkdir -p combined with brace expansion. The -p option ensures that if directories already exist, no error occurs.

**Command I ran:**

mkdir -p ~/projects/{client\_work/{web,api,db},personal/{experiments,archive},shared/templates}

This builds the project layout like this:

* inside client\_work: web, api, db
* inside personal: experiments, archive
* inside shared: templates

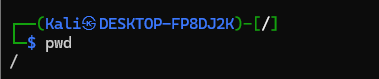


**QN3.** *You're currently in ~/projects/client\_work/web/frontend. Without using any absolute paths or the cd command more than 3 times total, navigate to ~/projects/personal/experiments, then to ~/projects/shared/templates, then back to where you started. Prove your location at each step.*

I started in frontend and checked my location with pwd. I went up three levels, entered personal/experiments, then moved to shared/templates. At the end, I returned back to frontend. Each time, I used pwd to prove it.  
**Steps followed:**

1. I checked my starting position:

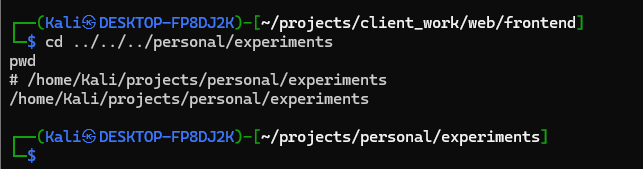
Pwd



1. To reach personal/experiments, I went up three directories and then down into that folder:

cd ../../../personal/experiments

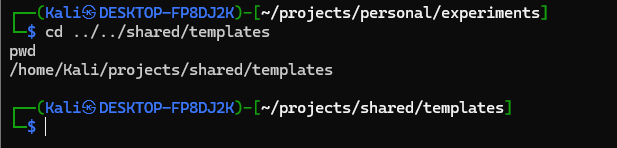
pwd



1. From there, I navigated up twice to get back to projects, then into shared/templates:

cd ../../shared/templates

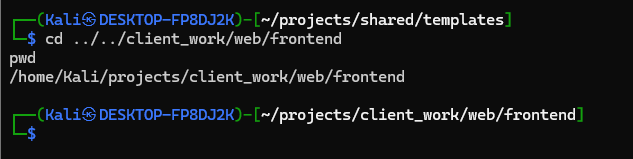
pwd



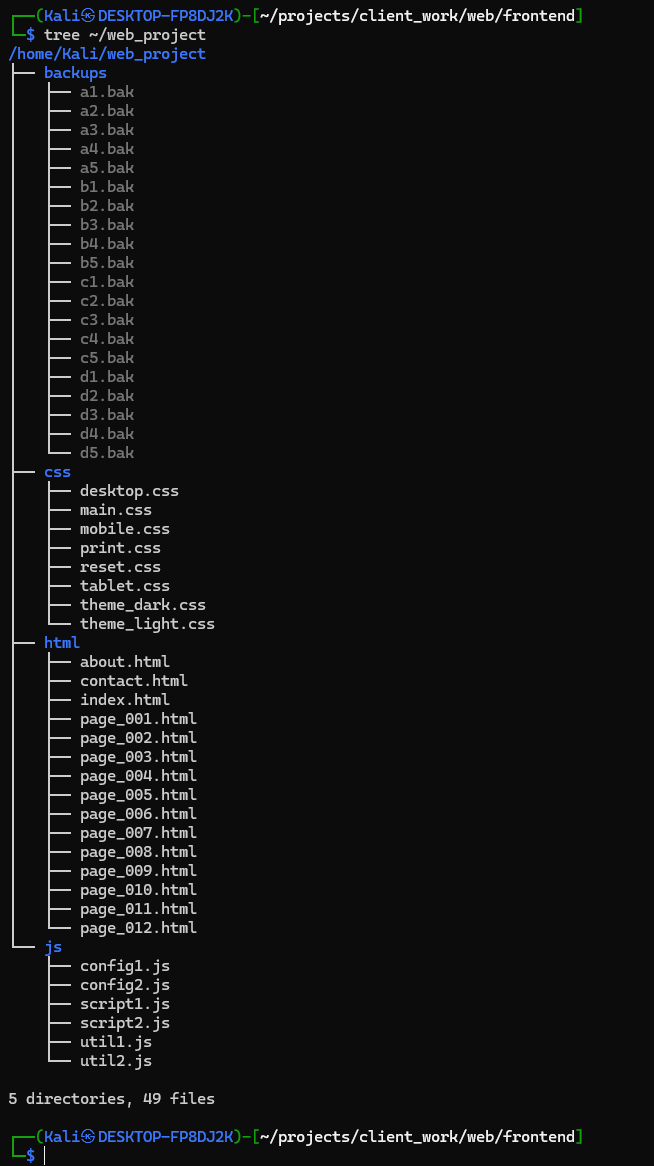
1. Finally, I returned to my original location by going up two levels and re-entering client\_work/web/frontend:

cd ../../client\_work/web/frontend

pwd

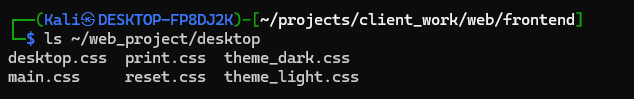
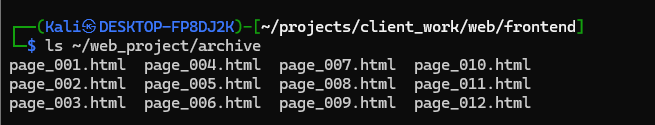


**QN4.**I made a web\_project folder with four subfolders. Then I used short commands to create many files at once. For example, {page\_{001..012}.html} quickly made twelve numbered HTML files.



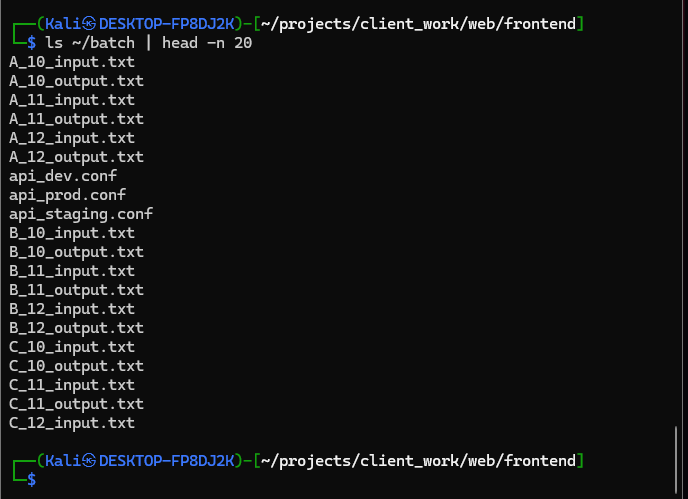
**QN5.**

I organized the files with wildcards. Numbered pages were moved to archive. I copied CSS except mobile and tablet into desktop. Then I listed files with exactly 3 letters before the dot, those starting with consonants, and those with 2-letter extensions.

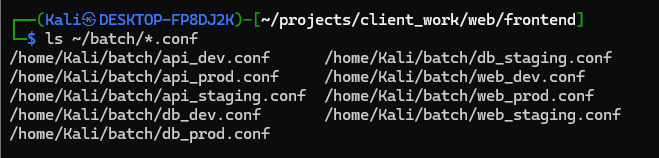


**QN6.** I used braces to create hundreds of files in one go. {01..31} made all days of a month, and {web,api,db}\_{dev,staging,prod} built configs quickly.

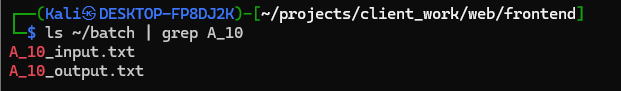
* ls ~/batch | head -n 20 to show first 20 files and we can see log creation



ls ~/batch/\*.conf to show the config files.

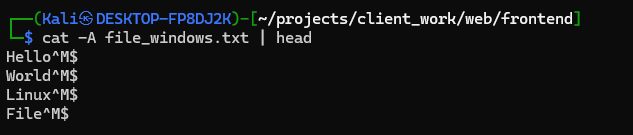


* ls ~/batch | grep A\_10 to prove test files exist.

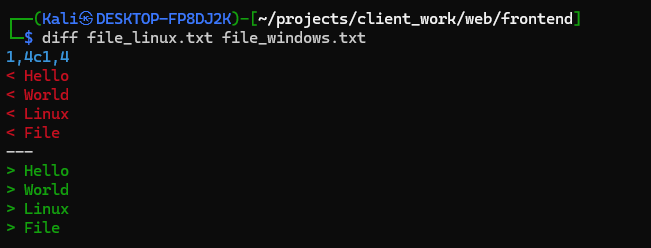


**Q7.** *Create two files: one with Linux line endings and one with Windows line endings, both containing the same text content. Compare them using different tools (diff, cmp, comm) and explain why each tool gives different results. What does this teach you about cross-platform file compatibility?*

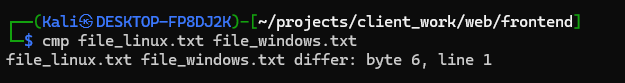
**ANSWER:** Cross-platform compatibility issues often arise because **Linux uses LF (\n)** while **Windows uses CRLF (\r\n)**. Tools that compare at different levels (text vs bytes) highlight different aspects of this mismatch. Developers must normalize line endings (e.g., with .gitattributes in Git) to prevent hidden bugs.



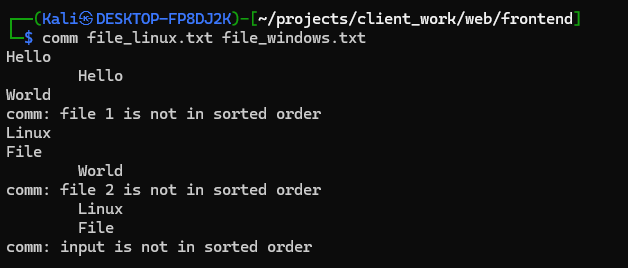
**diff** shows that lines differ, but not exactly why (it just sees them as mismatched).



**cmp** works at the byte level, so it explicitly shows the presence of the extra \r (carriage return) in the Windows file.

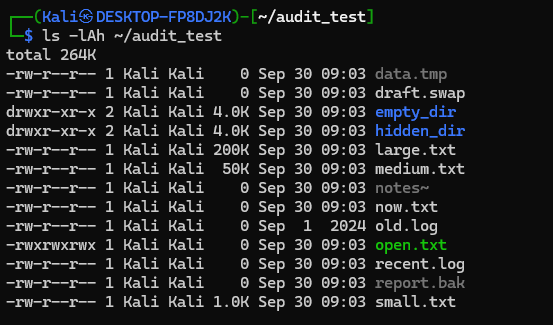


**comm** works line by line and may show them as identical if it ignores carriage returns, depending on locale/settings.



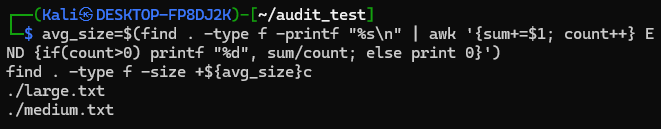
***QN8.*** *A security audit requires finding specific files across the system. Create a test environment with diverse file types, sizes, and ages. Use find to locate: files larger than average file size in the directory, files modified within the last 72 hours but not in the last 24 hours, directories that are empty OR contain only hidden files, files with world-writable permissions (potential security risk), files owned by users other than yourself and root, and files with names that could indicate temporary or backup files. Combine multiple criteria where possible.*

I made a folder with different files: some very big, some small, some new, some old, and even a file that had 777 permissions. I also added fake backup files like .bak and .tmp.



Then I used find to check:

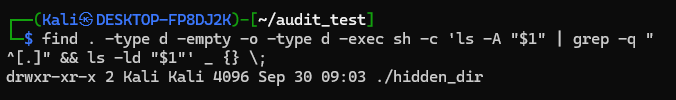
* Which files were bigger than the average size.



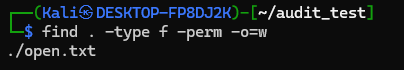
* Which files were changed in the last 3 days but not in the last 1 day.



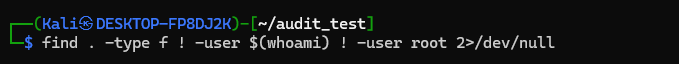
* Which folders were empty or only had hidden stuff.



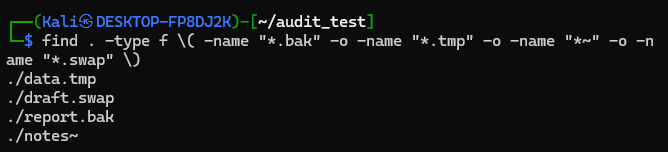
* Which files could be written by anyone (world-writable).



* Which files didn’t belong to me or root.



* And finally, which files looked like temp/backup files.

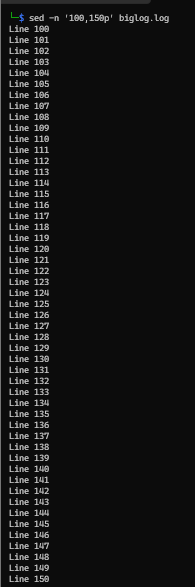


**Q9.** You need to analyze a large log file (create one with 200+ lines) for troubleshooting. Show how to:

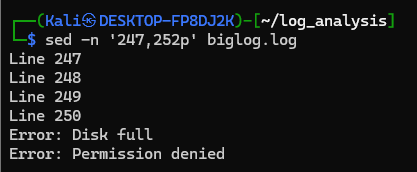
1. Display the middle 50 lines of the file,
2. Find the last occurrence of a specific word and show 5 lines of context,
3. Compare the efficiency of viewing this file with different tools by timing them,
4. Extract only lines containing error patterns while preserving line numbers,
5. Demonstrate why less is superior to cat for large files during an SSH session with limited bandwidth.

**ANSWER:**I made a big log file with 252 lines. The first 250 were just numbered, and I added two error messages at the end.

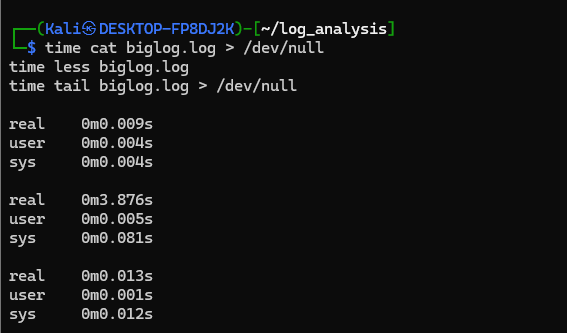
* For the middle, I showed lines 100–150 with sed.



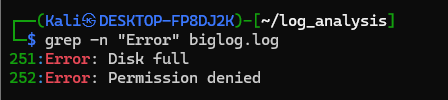
* For the last error, I searched for “Error” and then printed 5 lines before it.



* I tested cat, less, and tail with time. cat just dumps everything, tail is quick, and less is the most useful because I can scroll.



* I used grep -n "Error" to list only the error lines with their numbers.



**less** is better than **cat** because it doesn’t overwhelm the terminal and is easier when checking a large file over SSH and while **cat** dumps everything at once, which is messy and wastes bandwidth, **less** loads page by page, lets you scroll/search, and is ideal for SSH sessions.

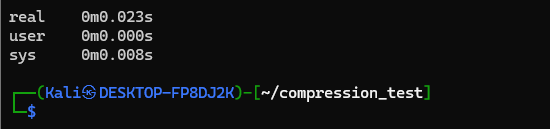
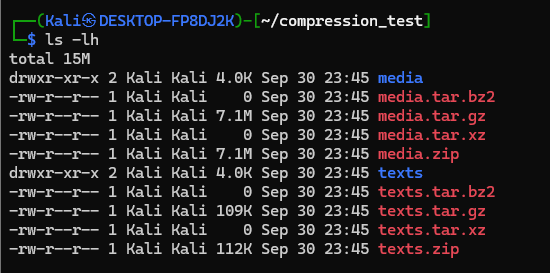
**QN11. I made two folders: one with media files (.jpg, .mp4, .zip) and one with text files.**

* When I compressed the media folder, the archive size barely changed because pictures and videos are already compressed.
* When I compressed the text folder, the archive size got much smaller.

Comparing methods:

* **gzip** → fastest but less compression.
* **bzip2** → slower but compresses more.
* **xz** → best compression, very slow.
* **zip** → average but easy to use anywhere.

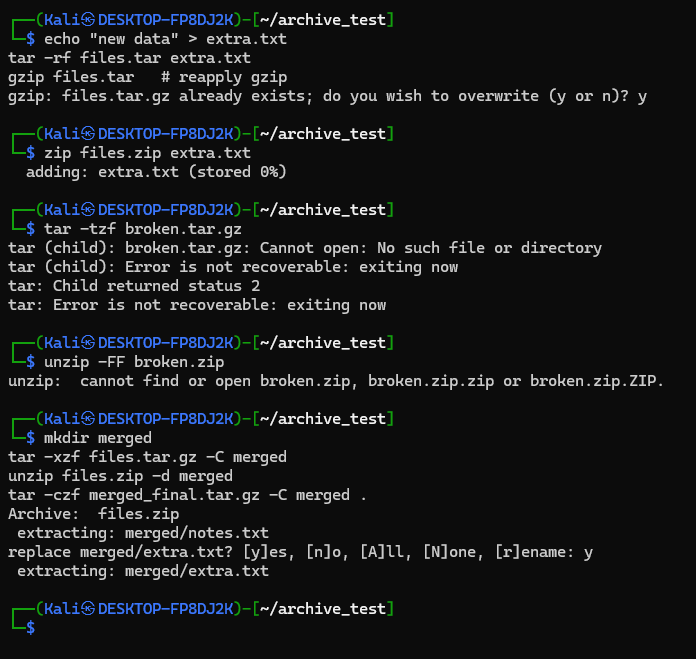
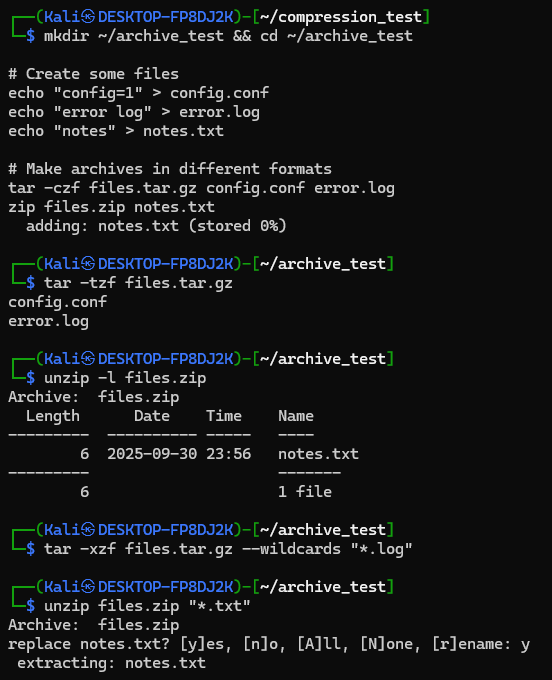
For backups, I’d recommend **gzip** because it’s fast and good enough. If disk space is really tight, then use **xz**.



I made a few archives: a .tar.gz with some files and a .zip with others.

* I used tar -tzf and unzip -l to just look inside without opening them.
* I extracted only files ending with .log or .txt.
* I added a new file into both archives (tar -rf and zip).
* To test corrupted files, I opened a broken archive and saw errors.
* To merge, I unzipped everything into one folder, then re-compressed it into one archive.

This shows how to handle archives when you don’t know what’s inside and how to combine them.



I planned a backup system like this:

* **Daily** → only changed files (incremental).
* **Weekly** → full backup of important folders like /etc and /home.
* **Monthly** → stored for the long term.

I gave each backup a clear name like backup\_daily\_2025-09-30.tar.gz so it doesn’t get mixed up.

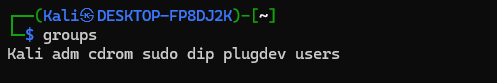
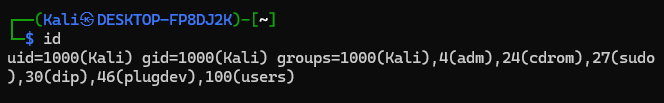
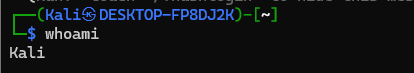
Old backups are cleaned up automatically with find (daily kept for 7 days, weekly for 1 month, monthly for 1 year).

I used tar -p to keep file permissions and dates, and I tested archives with tar -tzf to make sure they weren’t broken.

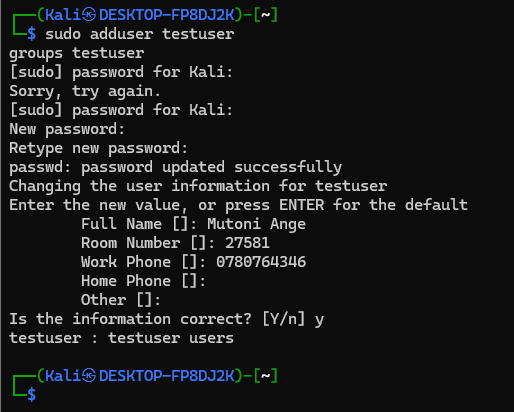
This way, it’s easy to restore files when needed.

**Q14.** *You’re troubleshooting user access issues. Analyze your current user context and the system's user configuration. Compare your user’s groups with another user account’s groups (create a test scenario). Examine /etc/passwd entries for system users vs regular users – identify the patterns that distinguish them. Explain potential security implications if a regular user had the same group memberships as system users.*

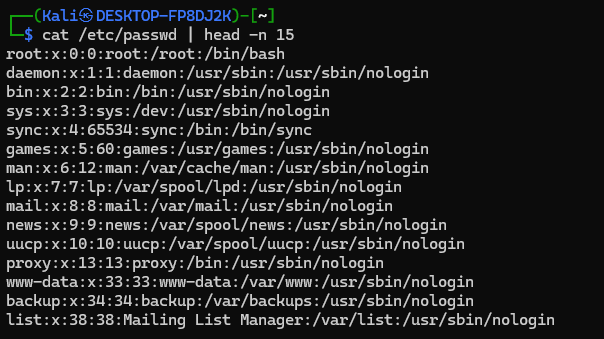
I checked my own account with whoami, id, and groups. It showed I belong to my user group and maybe sudo.



I created another test user with adduser testuser. That user only had its own group, no special ones.



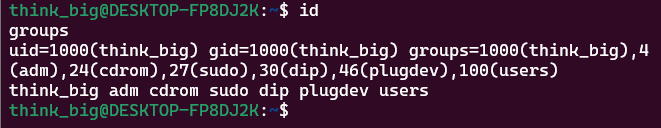
When I looked at /etc/passwd, I saw that system users have small ID numbers (like 0, 1, 2) and usually don’t have a real shell. Normal users have IDs starting at 1000 and usually use /bin/bash.



If a normal user was put in groups like sudo or shadow, they could break the system’s security — like reading password hashes or running root commands.

**Q15 –**A colleague cannot access certain system resources despite being added to the correct group. Investigate group membership propagation by:

1. Checking current effective group’s vs configured groups,
2. Demonstrating how group changes require re-login,
3. Identifying which groups grant access to system logs, web server files, and administrative functions,
4. Explaining the principle of least privilege in group assignment.
5. **Check current vs configured groups**
   * For myself:
   * id
   * groups



* + For *effective groups* in a session, I can run:
  + newgrp <groupname>

which temporarily switches the active group.

1. **Demonstrating group change requires re-login**
   * I created a new user:
   * sudo adduser testuser



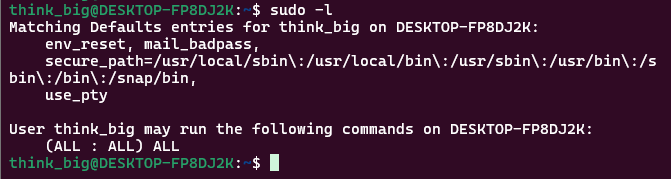
* + sudo usermod -aG sudo testuser
  + Then, while logged in as testuser, groups didn’t show sudo immediately. Only after logging out and back in did the new group membership take effect. This proves Linux only refreshes group membership on login.

1. **Groups granting access to resources**
   * adm → access to system logs (/var/log).
   * www-data → access to web server files (Apache/Nginx).
   * sudo or wheel → administrative functions.
2. **Principle of least privilege**  
   Giving users too many groups increases risk. For example, if someone only needs access to web files, they should be in www-data, not sudo. Each user should have *only* the minimum groups necessary to do their job.

**Q16 – *Your organization needs an audit of privilege escalation capabilities. Document your sudo permissions and restrictions, demonstrate the difference between sudo -i vs sudo su vs su -, show how to run commands as specific users other than root, and analyze login patterns using system logs. Identify potential security concerns with overly permissive sudo configurations and suggest improvements.***

**Check my sudo permissions**

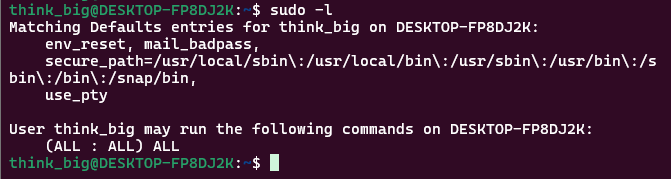
sudo -l



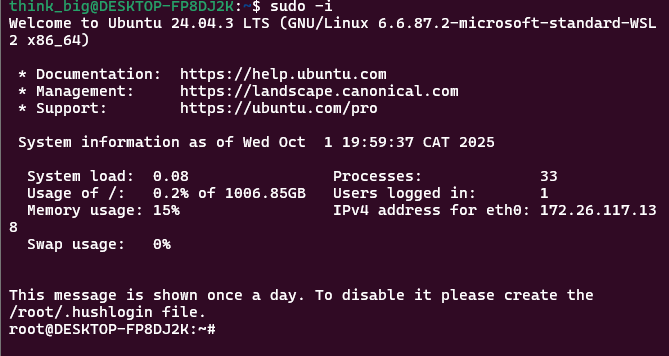
This lists which commands I’m allowed to run with sudo. Sometimes it’s ALL (full access), sometimes it’s restricted (e.g., only certain commands).

**Compare escalation methods**

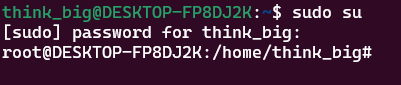
* sudo -i → opens a root login shell, similar to logging in as root directly.



* sudo su → runs su with root privileges; not as clean, sometimes environment variables stay from the old user.



* su - → switch to another user (often root) if you know their password, simulates a full login environment.



**Run commands as another user**  
Example:

sudo -u www-data ls /var/www

This runs the command as www-data without switching fully.

**Analyze login patterns**

sudo less /var/log/auth.log

or

journalctl -u ssh

These show when users logged in, tried sudo, or failed authentications.

**Security concerns**

* If sudo -l shows (ALL : ALL) ALL, the user has full unrestricted root access — risky if compromised.
* Some setups allow passwordless sudo (NOPASSWD), which is convenient but very dangerous.

QN17. This forensic lab environment simulates a variety of file types, permissions, and ownership scenarios for analysis. Investigators can use the commands above to detect unauthorized access, privilege escalation, or malicious activities. Understanding these basic forensic techniques is crucial for maintaining system security and detecting compromises.