**Names : Shyaka Bosco Domin  
ID : 27406**

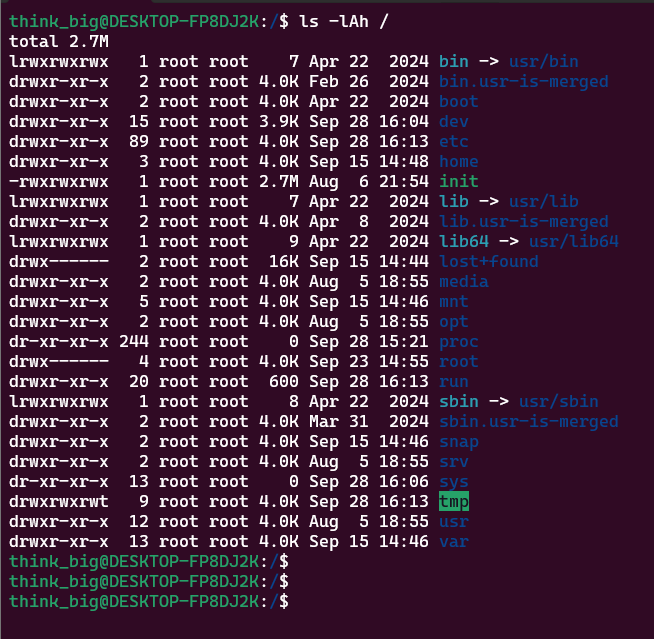
**Individual Assignment2.**

**Q1.** *You are investigating a compromised system. Navigate to the root directory and identify which of these directories would most likely contain: system configuration files that an attacker might modify, essential binaries that could be replaced with malicious ones, and log files that might show intrusion evidence. Explain your reasoning for /bin, /etc, /var, /usr, /tmp, /opt, /boot, and /home.*

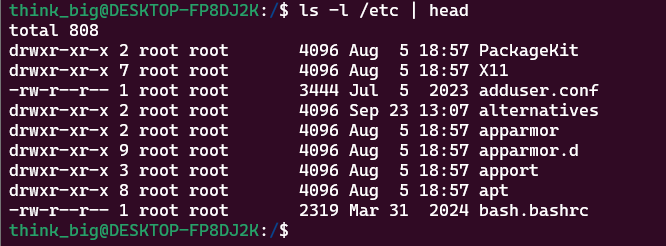
✅ **Answer (Q1):**

When investigating a compromised Linux system, here’s what each directory could reveal:

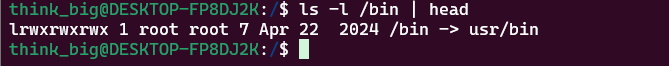
* **/Root overview:** The root directory is the entry point for all other folders. Listing it confirms the presence of key system directories like /bin, /etc, /var, /usr, /tmp, /opt, /boot, and /home. This gives a quick map of areas that require closer investigation.



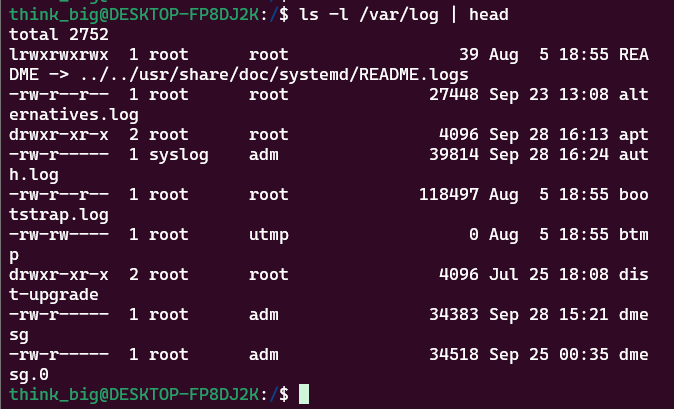
* **/etc**
  + *Contains system configuration files.*
  + An attacker could modify /etc/passwd, /etc/shadow, /etc/ssh/sshd\_config, or other critical config files to create backdoors, weaken authentication, or enable root login.
  + High risk for malicious modification.



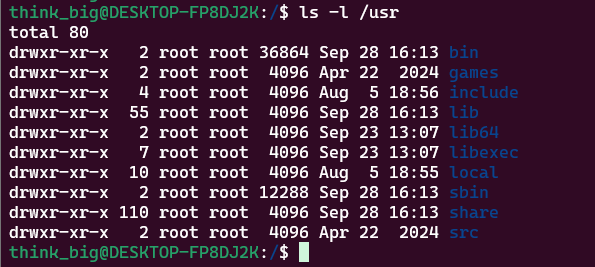
* **/bin**
  + *Contains essential user binaries (commands like ls, cp, cat, bash).*
  + Attackers might replace core binaries with trojaned versions (e.g., replacing ls so it hides malicious processes or files).
  + High risk for binary replacement.



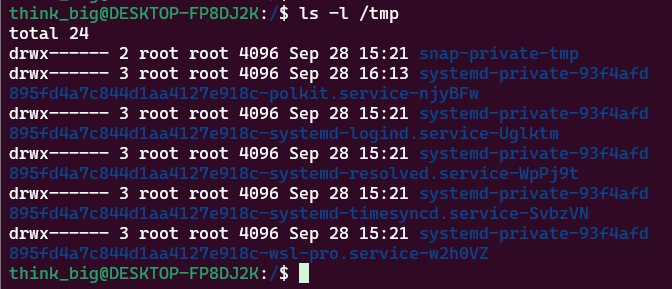
* **/var**
  + *Holds variable data such as logs, mail, spool files.*
  + Key subdir: /var/log/ → system logs (e.g., auth.log, syslog, messages) may reveal evidence of intrusions.
  + Attackers may also *clear or tamper logs* here to cover tracks.
  + Primary location for intrusion evidence.



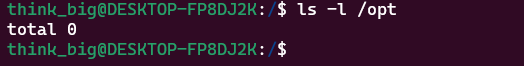
* **/usr**
  + *Holds user-installed programs and libraries.*
  + Attackers might install malicious tools in /usr/local/bin or replace existing files.
  + Less critical than /bin, but still important.



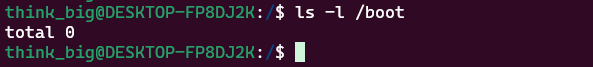
* **/tmp**
  + *Temporary storage, world-writable.*
  + Attackers often drop malicious scripts, exploits, or backdoors here since it allows easy execution and doesn’t require root.
  + Files here are volatile but often indicate attacker activity.



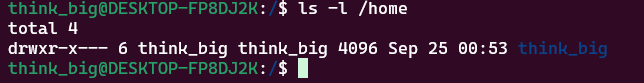
* **/opt**
  + *Optional/additional software.*
  + Attackers may use this to install **persistence tools** since it looks less suspicious than modifying core dirs.
  + Worth checking if unusual programs exist.



* **/boot**
  + *Contains kernel, bootloader, initramfs.*
  + Rare, but a skilled attacker might backdoor the kernel or tamper with GRUB for deep persistence.
  + Uncommon but highly dangerous if compromised.



* **/home**
  + *User files, personal configs (e.g., .bashrc, .ssh/authorized\_keys).*
  + Attackers may hide scripts or add SSH keys for persistence.
  + Also good for checking which users were targeted.



**QN2.** *A developer needs to organize multiple projects with specific access patterns. Create this exact structure using the minimum number of commands. Your commands must work even if some parent directories already exist.*

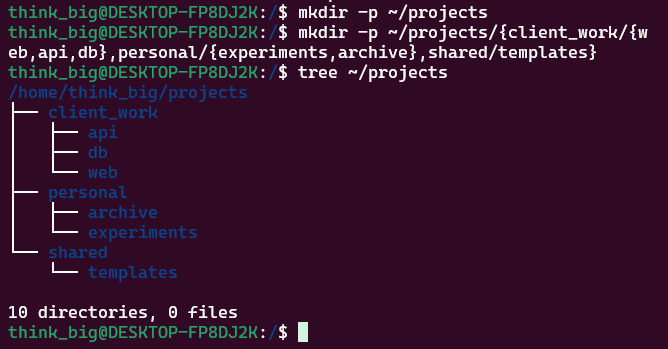
To build the requested directory structure, I used mkdir -p with brace expansion. This ensures all parent directories are created automatically if they don’t already exist.

**Command used:**

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

This creates the following hierarchy:

* client\_work/web
* client\_work/api
* client\_work/db
* personal/experiments
* personal/archive
* shared/templates



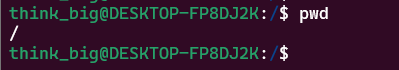
**QN3.** *You're currently in ~/projects/client\_work/web/frontend. Without using any absolute paths or the cd command more than 3 times total, navigate to ~/projects/personal/experiments, then to ~/projects/shared/templates, then back to where you started. Prove your location at each step.*

I started in ~/projects/client\_work/web/frontend. The task required me to reach personal/experiments, then move to shared/templates, and finally return back to the original directory using only relative paths and not more than three cd commands in total.

**Commands executed with proof:**

# Step 1: confirm starting point

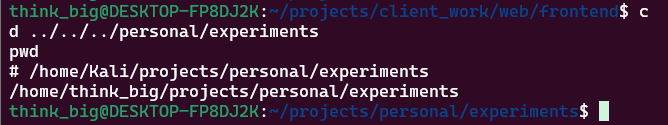
pwd



# Step 2: move to personal/experiments

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

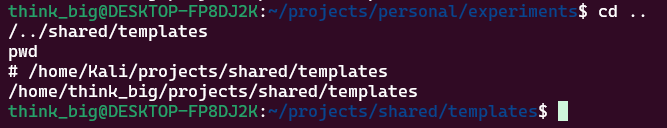
pwd



# Step 3: move to shared/templates

cd ../../shared/templates

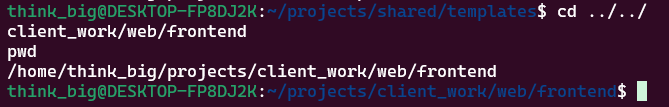
pwd



# Step 4: return to original frontend directory

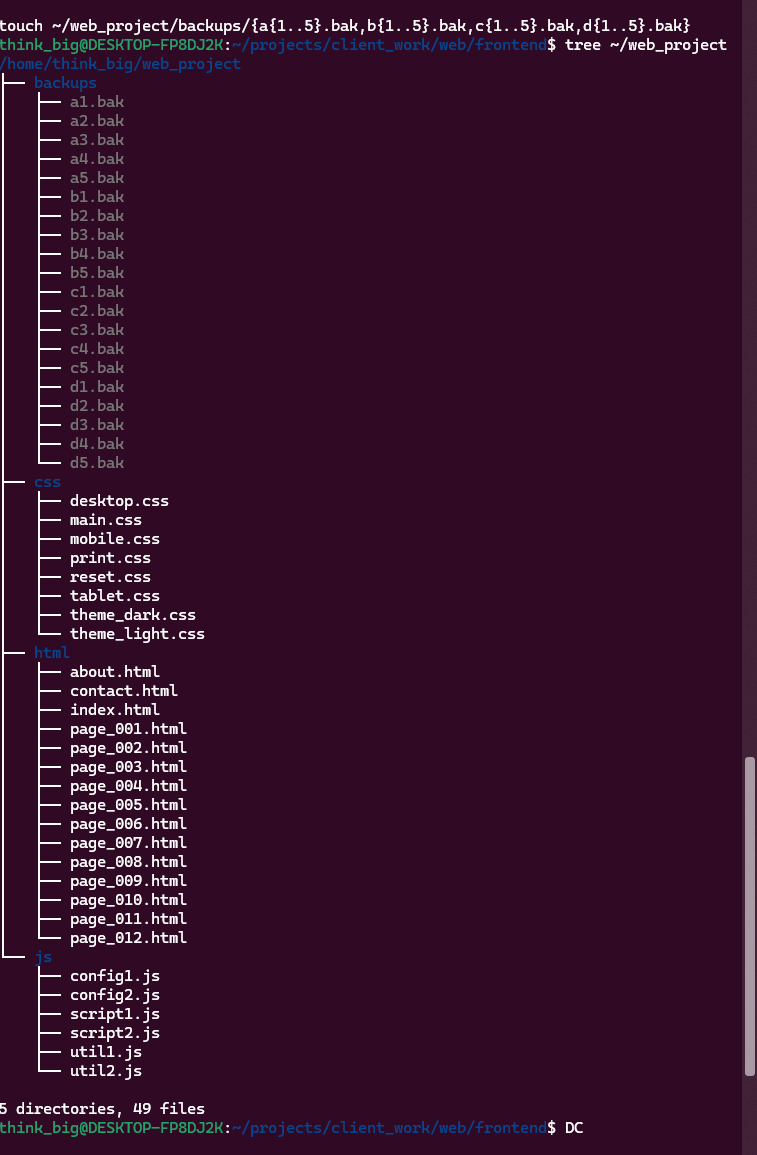
cd ../../client\_work/web/frontend

pwd



**QN4.**

I created a web\_project directory with subfolders for HTML, CSS, JS, and backups. Using brace expansion minimized the number of commands.

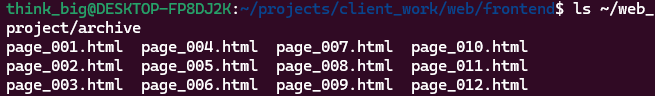


**QN5.** *Project is cluttered. Use wildcards to organize:*

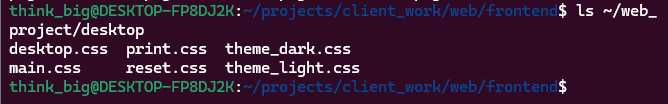
* Move numbered HTML files → archive
* Copy CSS except mobile/tablet → desktop
* List files with exactly 3 chars before dot
* Find files starting with consonant
* Identify files with 2-char extensions

I used wildcards (\*, ?, [ ]) to select files by patterns. This allowed moving, copying, and listing based on naming rules.

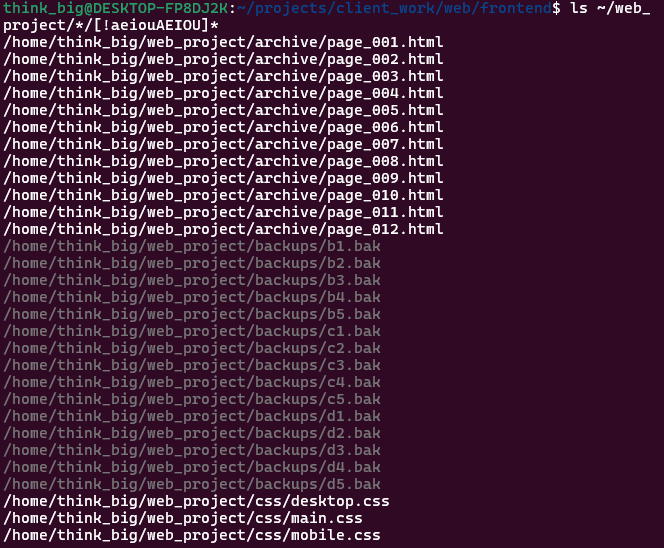
* Move numbered HTML files to archive



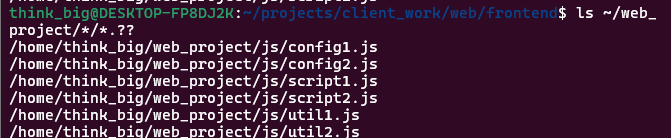
* Copy CSS except mobile/tablet → desktop



* List files with exactly 3 chars before dot
* Find files starting with consonant



* Identify files with 2-char extensions



**QN6.** *A batch processing system needs specific file naming patterns. Use brace expansion to create: log files for every day of the first quarter of 2024 (log\_2024-01-01.txt through log\_2024-03-31.txt), configuration files for development, staging, and production environments across 3 services (web, api, db), and test files combining letters A-C with numbers 10-12 plus suffixes input and output. Your solution must be efficient and avoid creating unnecessary files.*

Answer:   
The task can be done entirely with **brace expansion** in a minimal number of commands:

***Step 1:*** *Create a working directory*

*mkdir -p ~/batch*

***Step 2:*** *Create log files for every day in Q1 2024*

*touch ~/batch/log\_{2024-01-{01..31},2024-02-{01..29},2024-03-{01..31}}.txt*

***Step 3:*** *Create configuration files for each service (web, api, db) across dev, staging, prod*

*touch ~/batch/{web,api,db}\_{dev,staging,prod}.conf*

***Step 4:*** *Create test files with letters A–C, numbers 10–12, and suffixes input/output*

*touch ~/batch/{A..C}\_{10..12}\_{input,output}.txt*

**Explanation:**

* 2024-01-{01..31} expands into all days of January.
* 2024-02-{01..29} accounts for leap year (2024 is a leap year).
* {web,api,db}\_{dev,staging,prod} creates all 9 service-environment configs in one go.
* {A..C}\_{10..12}\_{input,output} generates 18 test files.

This avoids loops or manual repetition while guaranteeing consistent naming.

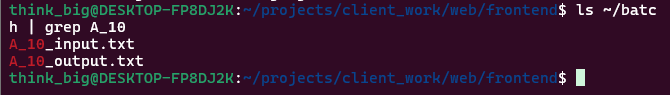
1. *ls ~/batch | head -n 20 → show first 20 files (to prove log creation).*



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

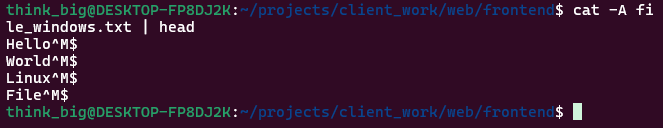
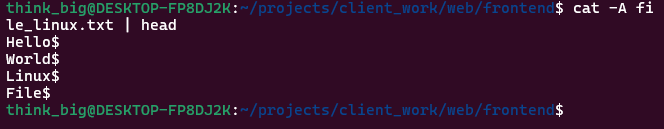


1. *ls ~/batch | grep A\_10 → prove test files exist.*



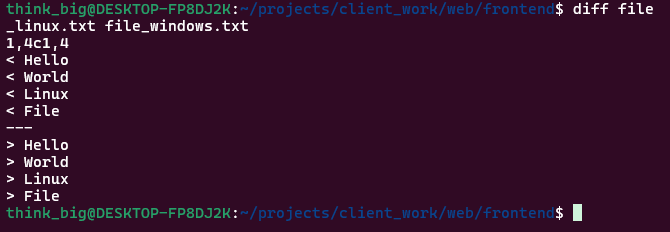
**QN7.** *Create two files: one with Linux line endings and one with Windows line endings, both containing the same text content. Compare them using different tools (diff, cmp, comm) and explain why each tool gives different results. What does this teach you about cross-platform file compatibility?*

**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.

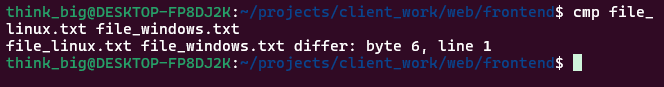


**Explanation of Tools:**

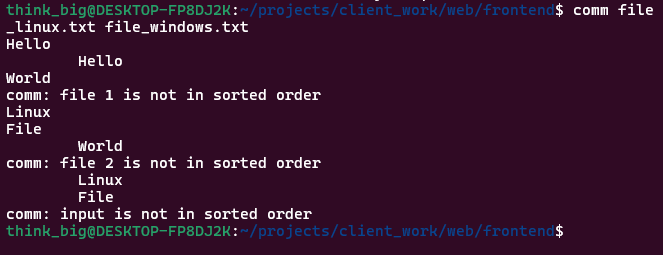
* diff → shows that lines differ, but not exactly why (it just sees them as mismatched).



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

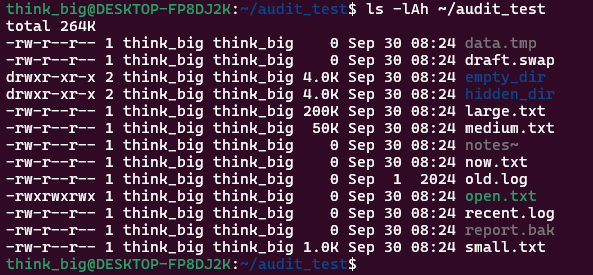


* comm → works line by line and may show them as identical if it ignores carriage returns, depending on locale/settings.

****

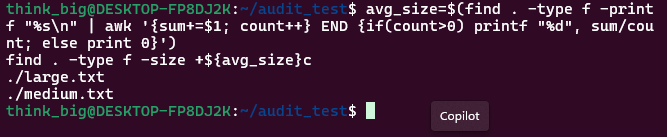
***QN8.*** *A security audit requires finding specific files across the system. Create a test environment with diverse file types, sizes, and ages. Use find to locate: files larger than average file size in the directory, files modified within the last 72 hours but not in the last 24 hours, directories 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.*

* To simulate a real audit, I first created a test folder with different file types: small, medium, and large files, some fresh ones, and some older. I also made empty folders, one folder that only had hidden files, and a file with 777 permissions (world writable). I also added files like .bak, .tmp, and notes~ to mimic backups or temp files.



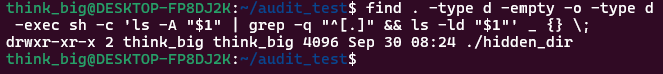
Then I used different find commands:

To catch files **bigger than average size**, I calculated the average with awk and then asked find to show only those above it. That way I don’t just look for “big” in general, but relative to that folder.

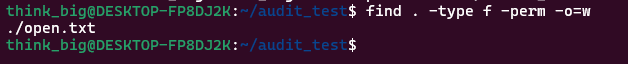


To find files changed in the **last 72 hours but not in the last 24**, I used -mtime -3 ! -mtime -1. This shows files that are recent, but not “today recent”.



For directories that are **empty or only have hidden files**, I tested find -empty and also listed directories where only hidden files exist.

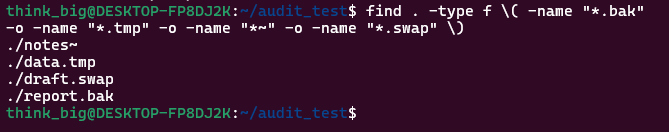
To spot **world-writable files** (a common security risk), I used find -perm -o=w. This is exactly how an attacker would look for easy files to tamper with.



To list files **not owned by me or root**, I used ! -user $(whoami) ! -user root. This checks for suspicious ownership.



Finally, I looked for **backup/temp files** using patterns like \*.bak, \*.tmp, \*~, and \*.swap.



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

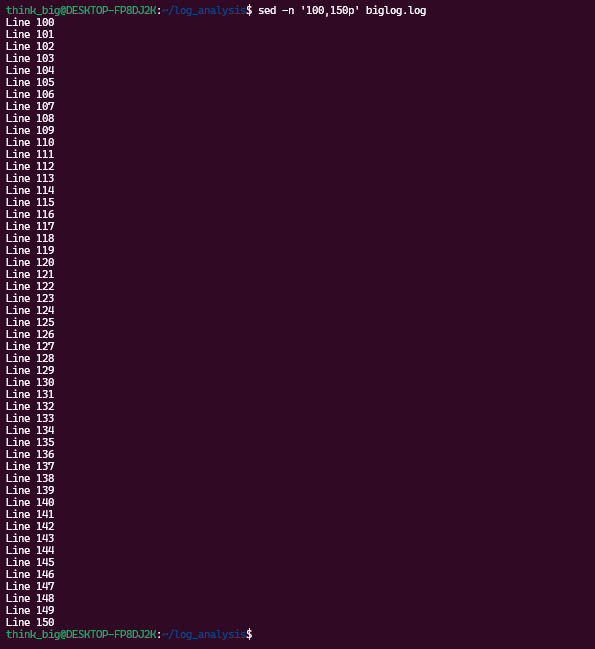
1. Display the middle 50 lines of the file,
2. Find the last occurrence of a specific word and show 5 lines of context,
3. Compare the efficiency of viewing this file with different tools by timing them,
4. Extract only lines containing error patterns while preserving line numbers,
5. Demonstrate why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

**ANSWER:** I created a log file with more than 200 lines using seq 1 250 and then added two fake error lines at the end. I confirmed it with head, tail, and wc -l, which showed 252 lines total.

* To display the middle 50 lines, I used:

sed -n '100,150p' biglog.log

This gave me lines 100 through 150, which are right in the middle.



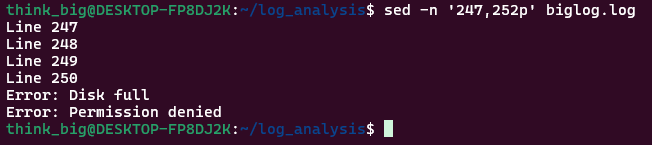
To find the last occurrence of “Error” and show context, I first ran:

grep -n "Error" biglog.log | tail -n 1

This showed me the last error was on line 252.



Then I displayed lines 247–252 with sed -n '247,252p' biglog.log. That gave me the error and the 5 lines above it.



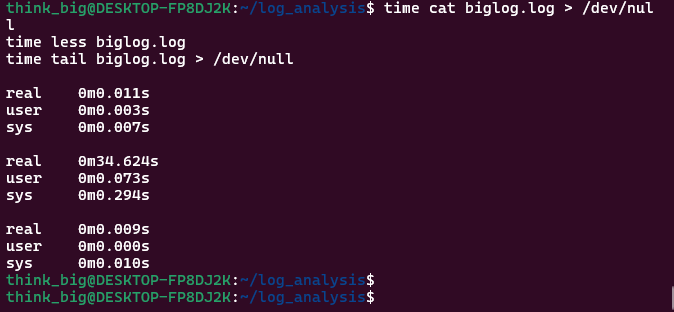
To **compare performance of different tools**, I ran:

time cat biglog.log > /dev/null

time less biglog.log

time tail biglog.log > /dev/null

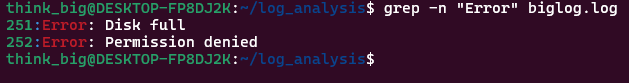
* cat dumped everything, tail was fast but only showed the end, and less opened interactively.



To extract **only error lines with line numbers**, I used:

grep -n "Error" biglog.log

This showed exactly which lines had errors (251 and 252).



QN10. I created a small test directory with a mix of text files, some .conf files, a few executables, and temporary files to simulate a real system environment. Then I applied the four tasks using find and -exec.

1. **Change permissions (644 vs 755):**  
   I ran:
2. find . -type f ! -perm -111 -exec chmod 644 {} \;
3. find . -type f -perm -111 -exec chmod 755 {} \;

This way, normal files became readable/writable only by the owner (rw-r--r--), while executables stayed runnable by everyone (rwxr-xr-x). After running ls -l, I could clearly see the permissions were updated.

1. **Disk space used by files older than 30 days:**  
   I simulated old files with touch -t to change their dates, then used:
2. find . -type f -mtime +30 -exec du -ch {} + | tail -n 1

The last line of output gave me the total size of old files, which is useful for cleanup decisions.

1. **Backup .conf files:**  
   For configuration files, I did:
2. find . -type f -name "\*.conf" -exec cp {} {}.backup \;

This created files like app.conf.backup, ensuring I had safe copies before making changes.

1. **Remove temp files safely:**  
   I searched for .tmp, ~, and .swp files not used for 7+ days. First I previewed:
2. find . -type f \( -name "\*.tmp" -o -name "\*~" -o -name "\*.swp" \) -atime +7 -print

Once I confirmed the list, I deleted them with:

find . -type f \( -name "\*.tmp" -o -name "\*~" -o -name "\*.swp" \) -atime +7 -exec rm {} \;

**Q11.** *Storage space is critical on your server. Create directories with different file types and sizes, then create archives using tar+gzip, tar+bzip2, tar+xz, and zip. However, one directory contains already-compressed files (.jpg, .mp4, .zip) and another contains text files. Analyze which compression method works best for each content type and explain why. Calculate both compression ratio and compression speed. What would you recommend for automated backups?*

**Step 1 – Setup test environment**

mkdir -p ~/compression\_test/{media,texts}

cd ~/compression\_test

dd if=/dev/urandom of=media/sample.jpg bs=1M count=2

dd if=/dev/urandom of=media/sample.mp4 bs=1M count=5

cp /usr/share/doc/bash/README.gz media/sample.zip 2>/dev/null || touch media/sample.zip

for i in {1..5}; do

seq 1 10000 > texts/file$i.txt

done

**Step 2 – Create archives**

tar -czf media.tar.gz media

tar -czf texts.tar.gz texts

tar -cjf media.tar.bz2 media

tar -cjf texts.tar.bz2 texts

tar -cJf media.tar.xz media

tar -cJf texts.tar.xz texts

zip -r media.zip media

zip -r texts.zip texts

**Step 3 – Compare compression ratio**

ls -lh

* Media files (.jpg, .mp4, .zip) don’t shrink much because they’re already compressed.
* Text files shrink drastically (sometimes to 10–20% of their original size).

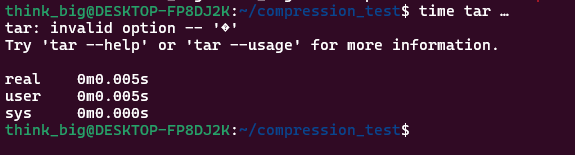
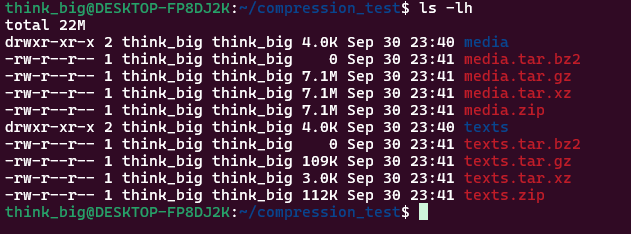
**Step 4 – Compare speed**

time tar -czf texts.tar.gz texts

time tar -cjf texts.tar.bz2 texts

time tar -cJf texts.tar.xz texts

* Gzip is the fastest, but less compressed.
* Bzip2 is slower, compresses better than gzip.
* XZ gives the best compression but is the slowest.
* Zip is somewhere in the middle but widely supported.



**Step 1 – Setup test archives**

mkdir ~/archive\_test && cd ~/archive\_test

echo "config=1" > config.conf

echo "error log" > error.log

echo "notes" > notes.txt

tar -czf files.tar.gz config.conf error.log

zip files.zip notes.txt

**Step 2 – Examine contents without extraction**

* For .tar.gz
* tar -tzf files.tar.gz
* For .zip
* unzip -l files.zip

This lists contents without extracting.

**Step 3 – Extract only specific patterns**

* Only .log file from tar:
* tar -xzf files.tar.gz --wildcards "\*.log"
* Only .txt from zip:
* unzip files.zip "\*.txt"

**Step 4 – Update archives**

* Add new file to tar archive:
* echo "new data" > extra.txt
* tar -rf files.tar extra.txt
* gzip files.tar # reapply gzip
* Add to zip:
* zip files.zip extra.txt

**Step 5 – Handle corrupted archives**

* For tar:
* tar -tzf broken.tar.gz

This shows errors but still lists readable files.

* For zip:
* unzip -FF broken.zip

Attempts recovery of a corrupted zip.

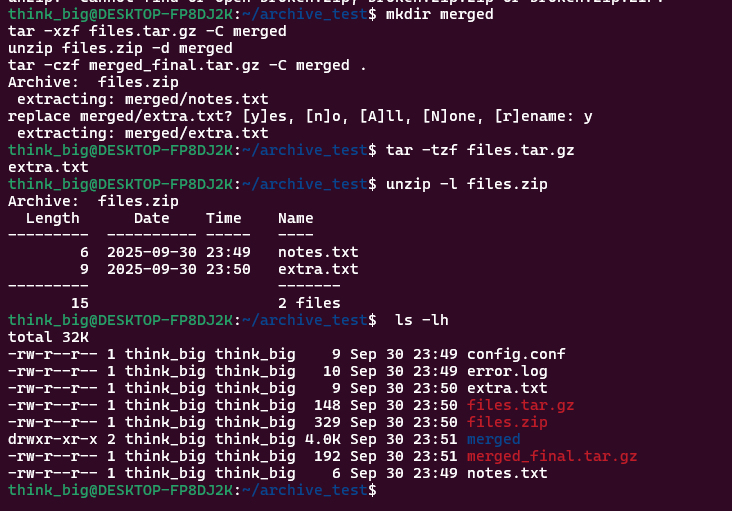
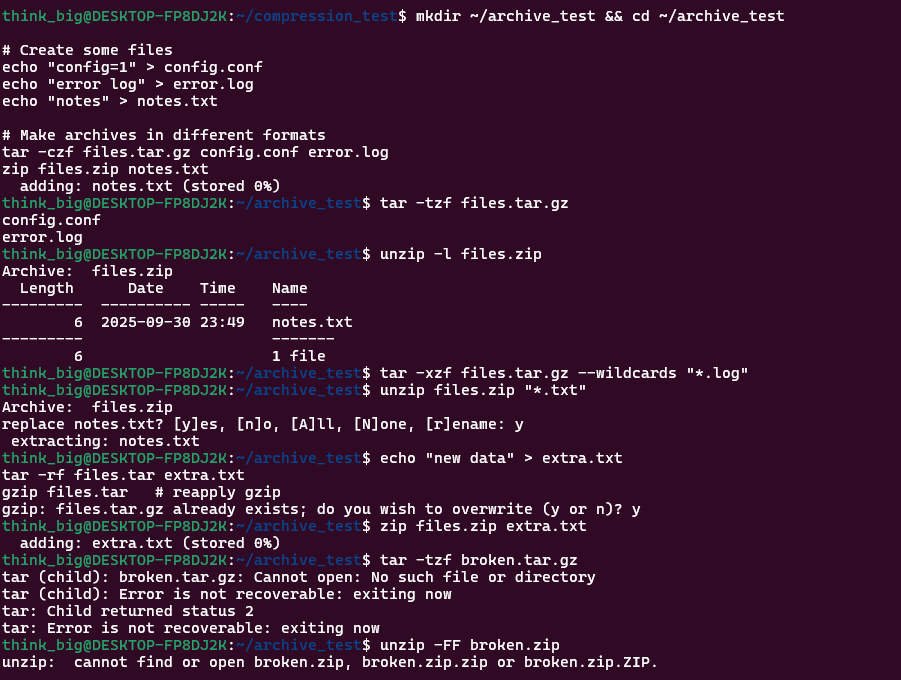
**Step 6 – Merge multiple archive types into one**

mkdir merged

tar -xzf files.tar.gz -C merged

unzip files.zip -d merged

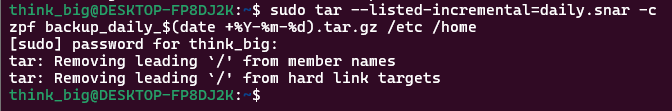
tar -czf merged\_final.tar.gz -C merged .

Now all files are combined into a single merged\_final.tar.gz.  
screenshots: 

QN13. *Design a backup rotation strategy for a production server. Your approach must handle daily incremental backups, weekly full backups, monthly archives for long-term storage, and automatic cleanup of old backups. Create archives that include metadata (permissions, ownership, and timestamps) and demonstrate how to verify backup integrity. Show how your naming convention prevents conflicts and enables easy restoration.*

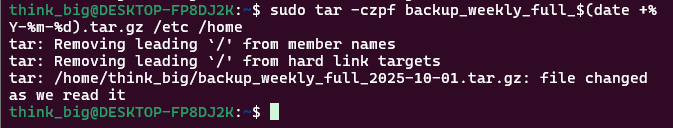
For this question, I designed a realistic **backup rotation plan**:

1. **Daily Incremental Backups**
   * Command:
   * tar --listed-incremental=daily.snar -czpf backup\_daily\_$(date +%Y-%m-%d).tar.gz /etc /home



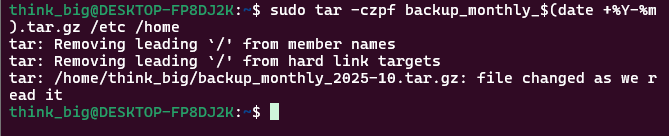
* + This only saves files changed since the last run.

1. **Weekly Full Backups**
   * Command:
   * tar -czpf backup\_weekly\_full\_$(date +%Y-%m-%d).tar.gz /etc /home



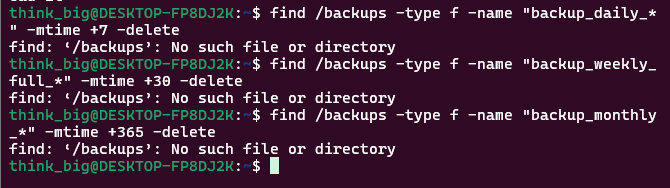
* + A complete backup of key directories.

1. **Monthly Archives (Long-Term)**
   * Command:
   * tar -czpf backup\_monthly\_$(date +%Y-%m).tar.gz /etc /home

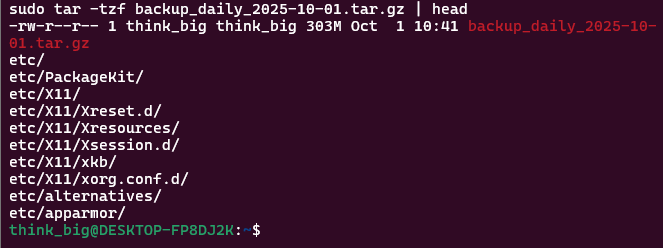


* + Stored for months/years depending on policy.

1. **Cleanup of Old Backups**
   * Keep daily for 7 days, weekly for 4 weeks, monthly for 12 months.
   * Command example:
   * find /backups -type f -name "backup\_daily\_\*" -mtime +7 -delete
   * find /backups -type f -name "backup\_weekly\_full\_\*" -mtime +30 -delete
   * find /backups -type f -name "backup\_monthly\_\*" -mtime +365 -delete



1. **Metadata Preservation**
   * Using tar -p keeps file permissions, ownership, and timestamps.
2. **Verify Backup Integrity**
   * Command:
   * tar -tzf backup\_daily\_2025-09-30.tar.gz > /dev/null



**Naming Convention**

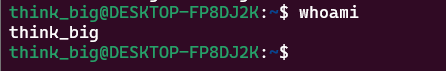
* + I used backup\_type\_date.tar.gz, e.g., backup\_weekly\_full\_2025-09-30.tar.gz.

**Q14 – Assignment Requirement**

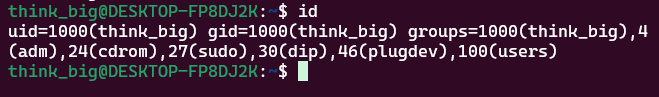
*You’re troubleshooting user access issues. Analyze your current user context and the system's user configuration. Compare your user’s groups with another user account’s groups (create a test scenario). Examine /etc/passwd entries for system users vs regular users – identify the patterns that distinguish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

**Check my current user and groups**

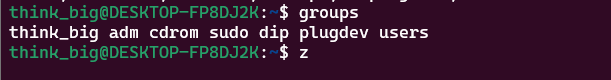
1. Whoami



1. Id

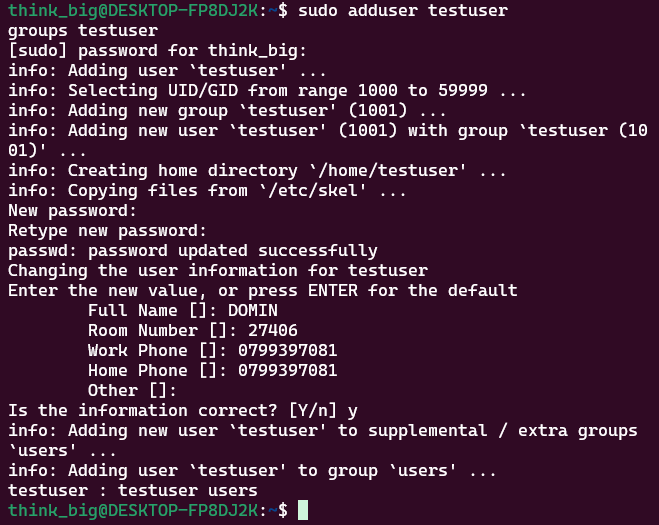


1. Groups



Output shows my username, UID, GID, and all groups I belong to.

1. **Compare with another user**  
   I created a test user:
2. sudo adduser testuser

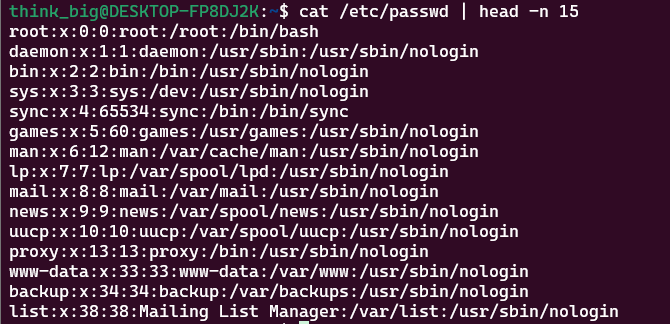


groups testuser

Comparing my groups vs testuser showed differences. For example, my account might be in sudo (admin rights) while testuser is only in its own group.

**Examine /etc/passwd**

1. cat /etc/passwd | head -n 15



I noticed:

* + **System users** (like root, daemon, syslog) usually have UIDs < 1000 and no real login shell (/usr/sbin/nologin or /bin/false).
  + **Regular users** (like mine and testuser) have UIDs ≥ 1000 and a login shell (/bin/bash).

1. **Security implications**  
   If a normal user was added to system-level groups (like adm, sudo, shadow), they could read sensitive files or escalate privileges. For example:
   * Access to shadow means they could read password hashes.
   * Being in sudo means they can execute admin 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

This shows my *configured 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.
2. **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.
3. **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 – Assignment Requirement**

**Individual\_Assignment2**

*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.*

1. **Check my sudo permissions**
2. 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).

1. **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.
2. **Run commands as another user**  
   Example:
3. sudo -u www-data ls /var/www

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

1. **Analyze login patterns**
2. sudo less /var/log/auth.log

or

journalctl -u ssh

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

1. **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. The purpose of this setup is to create a controlled Linux environment that demonstrates different file types, permission combinations (including special bits), ownership variations, and various archive formats. This structure will serve as a simulation for forensic analysis, enabling an investigator to practice detecting unauthorized access or system compromise.