**Exercises**

**Exercise 1: File Permissions and Security Script**

Write a Bash script that:

1. Checks if a file provided as a command-line argument has permissions more permissive than rw-r--r-- (0644). If it does:
   * Display a warning message: “The file has insecure permissions.”
   * Correct the permissions to rw-r--r--.
2. Scans a directory provided as the second argument and identifies all files with the .sh extension. For each file found:
   * Check if the file has the executable bit set (chmod +x).
   * If the executable bit is missing, display a message: “The script [file\_name] is not executable.”

**Hint**:

* Use the ls -l command to view file permissions.
* Compare permissions using stat -c "%a" and utilize if statements for conditions.
* Use the find command to scan directories.

**Exercise 2: File Integrity Checker**

Write a Bash script that:

1. Takes two arguments:
   * The first argument is a file to check.
   * The second argument is a directory to scan.
2. Computes the MD5 hash of the file provided and stores it in a variable.
3. Scans the specified directory for files with the same name as the first argument and compares their hashes with the stored one.
   * If a match is found, display: “File integrity verified for [file\_name].”
   * If no match is found, display: “File integrity compromised or file not found.”

**Hint**:

* Use the md5sum command to compute file hashes.

The find command will help you locate files in the directory.

### Exercise 3: Disk Space Monitor

Write a Bash script that:

1. Checks the disk usage of the system’s main partitions (/, /home, etc.) using the df command.
2. If any partition is over 80% usage, display a warning:
   * “Warning: Partition [partition\_name] is above 80% usage.”
3. Write the results into a log file (disk\_usage.log) with a timestamp for each check.

**Hint**:

* Use the df -h command to view disk usage.
* Parse the output with awk to extract relevant columns.
* Use the date command to add timestamps.

**Exercise 4: System Process Tracker**

Write a Bash script that:

1. Lists all currently running processes on the system, sorted by memory usage.
2. Highlights the top 5 processes consuming the most memory.
3. Saves the output into a log file (process\_log.txt) with a timestamp.

**Hint:**

* Use the ps aux command to list processes.
* Use sort to order by memory usage.
* Use head to extract the top results.
* Use date to add timestamps to the log file.

**Exercise 5: Service Status Checker**

Write a Bash script that:

1. Accepts a list of service names as arguments.
2. Checks if each service is running on the system.
3. For any service not running, restart it and log the action to service\_status.log.

**Hint:**

* Use the systemctl command to check and restart services.
* Parse the output of systemctl status.
* Use echo to log actions.

**Exercise 6: Backup Automation Script**

Write a Bash script that:

1. Accepts a directory path as an argument.
2. Compresses the directory into a .tar.gz file with a timestamped name.
3. Moves the compressed file to a predefined backup location.

**Hint:**

* Use the tar command to create compressed archives.
* Use mv to relocate the archive.
* Use basename to extract directory names for file naming.

**Exercise 7: Network Connectivity Checker**

Write a Bash script that:

1. Pings a list of IP addresses or domain names provided in a file (targets.txt).
2. Logs the results of each ping (success/failure) with timestamps into connectivity\_log.txt.
3. If any host is unreachable, display a warning message.

**Hint:**

* Use the ping command with the -c flag to limit the number of attempts.
* Use while loops to read from the file.
* Use grep or awk to parse ping results.

**Exercise 8: Log Cleaner Script**

Write a Bash script that:

1. Scans a directory for log files (\*.log) older than 7 days.
2. Deletes these log files after prompting the user for confirmation.
3. Logs the names of deleted files into a cleanup.log.

**Hint:**

* Use the find command with the -mtime option to identify old files.
* Use read for user input.
* Use echo to log the names of deleted files.

**Exercise 9: Cron Job Status Monitor**

Write a Bash script that:

1. Lists all active cron jobs for a specific user.
2. Checks the last execution time of each job from the system logs (/var/log/syslog).
3. Logs any cron jobs that failed to execute in the past 24 hours to cron\_failures.log.

**Hint:**

* Use crontab -l to list cron jobs.
* Use grep with timestamps to filter recent log entries.
* Use awk or cut to extract job details.

**Exercise 10: RAM Usage Analyzer**

Write a Bash script that:

1. Monitors the system’s RAM usage in real-time.
2. Logs the total, used, and free memory to ram\_usage.log every minute for 10 minutes.
3. Sends an alert if memory usage exceeds 50%.

**Hint:**

* Use the free -m command to view RAM usage.
* Use sleep in a loop to run checks periodically.
* Use mail or echo for alerts.

**Exercise 11: Open Ports Scanner**

Write a Bash script that:

1. Scans the system for open network ports using netstat or ss.
2. Identifies and logs which services are listening on these ports to open\_ports.log.
3. Alerts the user if any unexpected ports (e.g., not in a predefined list) are open.

**Hint:**

* Use netstat -tuln or ss -tuln to list open ports.
* Use awk or grep to filter specific services.
* Use an array or file to maintain a list of allowed ports.

### Exercise 12: Directory Size Auditor

Write a Bash script that:

1. Analyzes the size of all subdirectories in a given directory.
2. Lists the top 5 largest directories.
3. Logs the results to dir\_size\_audit.log.

**Hint:**

* Use the du -sh command to calculate directory sizes.
* Use sort and head to find the largest directories.
* Use basename to clean up directory names.