Hey! How's it going?

4o

**You said:**

shell scripts tasks that i can expect at hashedin univeristy please - assignments please with solutions

**ChatGPT said:**

I don't have access to specific course assignments from HashedIn University, but I can help you by suggesting some typical shell script tasks that you might encounter in a university setting or a similar environment. Here are some common shell scripting tasks with solutions that would be relevant for learning and practicing:

**Task 1: Create a simple shell script to check if a file exists.**

Write a script that takes a filename as input and checks if the file exists in the current directory.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Check if file exists

echo "Enter filename: "

read filename

if [ -e "$filename" ]; then

echo "File $filename exists."

else

echo "File $filename does not exist."

fi

**Task 2: Write a script to count the number of lines, words, and characters in a file.**

The script should take a filename as input and output the number of lines, words, and characters in the file.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Take file name as input

echo "Enter filename: "

read filename

if [ -e "$filename" ]; then

# Count lines, words, and characters

lines=$(wc -l < "$filename")

words=$(wc -w < "$filename")

chars=$(wc -m < "$filename")

echo "Lines: $lines"

echo "Words: $words"

echo "Characters: $chars"

else

echo "File does not exist."

fi

**Task 3: Write a script to list all files in a directory.**

The script should accept a directory as input and list all files in that directory, including hidden files.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Take directory name as input

echo "Enter directory: "

read dir

if [ -d "$dir" ]; then

echo "Listing files in $dir:"

ls -la "$dir"

else

echo "Directory does not exist."

fi

**Task 4: Create a script that finds and replaces a string in a file.**

The script should take three inputs: filename, the string to find, and the string to replace. It should modify the file in place.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input parameters

echo "Enter filename: "

read filename

echo "Enter string to find: "

read find

echo "Enter string to replace: "

read replace

if [ -e "$filename" ]; then

sed -i "s/$find/$replace/g" "$filename"

echo "Replaced all occurrences of '$find' with '$replace' in $filename."

else

echo "File does not exist."

fi

**Task 5: Create a script that checks if a given number is prime.**

Write a shell script to check if a given number is prime.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input the number

echo "Enter a number: "

read num

# Prime check logic

if [ $num -le 1 ]; then

echo "$num is not a prime number."

exit 1

fi

for ((i=2; i<=num/2; i++))

do

if [ $((num%i)) -eq 0 ]; then

echo "$num is not a prime number."

exit 0

fi

done

echo "$num is a prime number."

**Task 6: Write a script to print the Fibonacci sequence up to a given number.**

Write a shell script to print the Fibonacci sequence up to a given number of terms.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input the number of terms

echo "Enter the number of terms in Fibonacci sequence: "

read terms

a=0

b=1

echo "Fibonacci sequence up to $terms terms:"

for ((i=0; i<terms; i++))

do

echo -n "$a "

fib=$((a + b))

a=$b

b=$fib

done

echo

**Task 7: Write a script to find the largest file in a directory.**

The script should take a directory as input and find the largest file in that directory.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory

echo "Enter directory: "

read dir

if [ -d "$dir" ]; then

largest\_file=$(find "$dir" -type f -exec du -h {} + | sort -rh | head -n 1)

echo "The largest file is: $largest\_file"

else

echo "Directory does not exist."

fi

**Task 8: Create a script that backs up a directory.**

Write a shell script that creates a backup of a directory by compressing it into a .tar.gz file.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory to backup

echo "Enter directory to backup: "

read dir

if [ -d "$dir" ]; then

timestamp=$(date +"%Y%m%d%H%M%S")

backup\_file="backup\_$timestamp.tar.gz"

tar -czf "$backup\_file" "$dir"

echo "Backup of $dir created as $backup\_file."

else

echo "Directory does not exist."

fi

These tasks will give you a solid foundation in shell scripting. Practice these scripts by modifying them and adding more complex features to reinforce your learning.

4o

**You said:**

continue please

**ChatGPT said:**

Sure! Here are more shell script tasks to help you improve your skills:

**Task 9: Create a script to monitor disk usage and alert when usage exceeds a threshold.**

The script should check the disk usage of the root directory and send an alert if usage exceeds a given percentage.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Set threshold for disk usage

threshold=80

# Get current disk usage percentage of the root directory

usage=$(df / | tail -1 | awk '{print $5}' | sed 's/%//')

# Check if disk usage exceeds threshold

if [ "$usage" -gt "$threshold" ]; then

echo "Alert: Disk usage has exceeded $threshold%. Current usage: $usage%."

else

echo "Disk usage is below $threshold%. Current usage: $usage%."

fi

**sqTask 10: Create a script to back up MySQL database.**

Write a shell script to back up a MySQL database to a .sql file.

**Solution:**

bash

CopyEdit

#!/bin/bash

# MySQL credentials

echo "Enter MySQL username: "

read user

echo "Enter MySQL password: "

read -s pass

echo "Enter database name: "

read dbname

# Backup file with timestamp

timestamp=$(date +"%Y%m%d%H%M%S")

backup\_file="${dbname}\_backup\_$timestamp.sql"

# Command to create backup

mysqldump -u "$user" -p"$pass" "$dbname" > "$backup\_file"

if [ $? -eq 0 ]; then

echo "Database $dbname backed up successfully to $backup\_file."

else

echo "Error during backup."

fi

**Task 11: Write a script to find duplicate files in a directory.**

This script should check for duplicate files in a given directory based on file size and content.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory

echo "Enter directory: "

read dir

# Check if directory exists

if [ -d "$dir" ]; then

echo "Finding duplicate files in $dir..."

# List files with their checksums (md5sum)

find "$dir" -type f -exec md5sum {} + | sort | uniq -d --check-chars=32 | awk '{print $2}' > duplicates.txt

if [ -s duplicates.txt ]; then

echo "Duplicate files found:"

cat duplicates.txt

else

echo "No duplicate files found."

fi

else

echo "Directory does not exist."

fi

**Task 12: Write a script to create a user account and set a password.**

This script should allow you to create a new user on a Linux system and assign a password.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input username

echo "Enter username: "

read username

# Check if user already exists

if id "$username" &>/dev/null; then

echo "User $username already exists."

else

# Create user

useradd "$username"

# Set password for the user

echo "Enter password for $username: "

read -s password

echo "$username:$password" | chpasswd

if [ $? -eq 0 ]; then

echo "User $username created successfully."

else

echo "Failed to create user."

fi

fi

**Task 13: Write a script that prints a countdown from 10 to 0 with a delay.**

The script should print each number from 10 down to 0 with a one-second delay between each number.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Countdown from 10 to 0

for ((i=10; i>=0; i--))

do

echo "$i"

sleep 1

done

echo "Countdown complete!"

**Task 14: Write a script that renames all .txt files in a directory by adding a prefix.**

The script should take a directory name and prefix as input, and rename all .txt files in that directory by adding the specified prefix.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory and prefix

echo "Enter directory: "

read dir

echo "Enter prefix: "

read prefix

# Check if directory exists

if [ -d "$dir" ]; then

# Loop through all .txt files in the directory

for file in "$dir"/\*.txt; do

if [ -f "$file" ]; then

new\_name="$dir/$prefix$(basename "$file")"

mv "$file" "$new\_name"

echo "Renamed $file to $new\_name"

fi

done

else

echo "Directory does not exist."

fi

**Task 15: Write a script that creates a report on the system's resource usage.**

The script should generate a report with details about CPU usage, memory usage, and disk usage.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Report filename

report\_file="system\_report\_$(date +"%Y%m%d%H%M%S").txt"

# Get CPU usage

cpu\_usage=$(top -bn1 | grep "Cpu(s)" | sed "s/.\*, \*\([0-9.]\*\)%\* id.\*/\1/" | awk '{print 100 - $1 "%"}')

# Get memory usage

memory\_usage=$(free -m | awk 'NR==2{printf "Memory Usage: %.2f%%\n", $3\*100/$2 }')

# Get disk usage

disk\_usage=$(df -h | grep '^/dev' | awk '{print $1 ": " $5}')

# Write report

echo "System Resource Report - $(date)" > "$report\_file"

echo -e "\nCPU Usage: $cpu\_usage" >> "$report\_file"

echo -e "Memory Usage: $memory\_usage" >> "$report\_file"

echo -e "\nDisk Usage:" >> "$report\_file"

echo "$disk\_usage" >> "$report\_file"

echo "Report generated as $report\_file."

**Task 16: Write a script to search for a string in all files in a directory.**

The script should take a string and a directory as input and search for the string in all files in that directory.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input string and directory

echo "Enter string to search: "

read search\_string

echo "Enter directory: "

read dir

if [ -d "$dir" ]; then

echo "Searching for '$search\_string' in $dir..."

grep -r "$search\_string" "$dir"

else

echo "Directory does not exist."

fi

**Task 17: Write a script to check if a number is a palindrome.**

A script that checks if a given number is the same when reversed.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input number

echo "Enter a number: "

read number

# Reverse the number

reverse=$(echo "$number" | rev)

# Check if palindrome

if [ "$number" == "$reverse" ]; then

echo "$number is a palindrome."

else

echo "$number is not a palindrome."

fi

**Task 18: Write a script to create a simple to-do list.**

The script should allow users to add tasks to a to-do list and show the list.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Initialize to-do list file

todo\_file="todo\_list.txt"

# Check if file exists, create it if not

if [ ! -f "$todo\_file" ]; then

touch "$todo\_file"

fi

# Main loop

while true; do

echo "To-Do List Menu:"

echo "1. Add a task"

echo "2. Show tasks"

echo "3. Exit"

echo "Enter your choice (1/2/3): "

read choice

case $choice in

1)

echo "Enter task: "

read task

echo "$task" >> "$todo\_file"

echo "Task added."

;;

2)

echo "Your To-Do List:"

cat "$todo\_file"

;;

3)

echo "Exiting..."

break

;;

\*)

echo "Invalid choice."

;;

esac

done

These tasks should provide a solid foundation for practicing shell scripting, especially for system administration or automation purposes. Feel free to modify or extend the tasks based on your needs or interests!

4o

**You said:**

continue please

**ChatGPT said:**

Certainly! Here are even more shell scripting tasks for you to practice and enhance your skills:

**Task 19: Write a script to find and remove files older than a certain number of days.**

The script should take a directory and the number of days as input, and remove files that are older than the specified number of days.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory and number of days

echo "Enter directory: "

read dir

echo "Enter number of days: "

read days

if [ -d "$dir" ]; then

find "$dir" -type f -mtime +$days -exec rm -f {} \;

echo "Files older than $days days have been removed from $dir."

else

echo "Directory does not exist."

fi

**Task 20: Write a script to display system uptime.**

The script should display how long the system has been running, including the last boot time.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Get uptime and last boot time

uptime\_info=$(uptime -p)

last\_boot=$(who -b | awk '{print $3, $4}')

echo "System uptime: $uptime\_info"

echo "Last boot time: $last\_boot"

**Task 21: Write a script to create a timestamped backup of a file.**

The script should take a filename as input and create a backup with the current date and time as the suffix.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input filename

echo "Enter filename: "

read filename

# Check if file exists

if [ -e "$filename" ]; then

timestamp=$(date +"%Y%m%d%H%M%S")

backup\_file="${filename}\_backup\_$timestamp"

cp "$filename" "$backup\_file"

echo "Backup created as $backup\_file."

else

echo "File does not exist."

fi

**Task 22: Write a script to monitor a process and restart it if it is not running.**

The script should take a process name as input and check if it is running. If it’s not running, the script should restart it.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input process name

echo "Enter process name: "

read process

# Check if the process is running

if pgrep "$process" > /dev/null; then

echo "$process is running."

else

echo "$process is not running. Restarting..."

# Restart the process (replace with the correct command to start the process)

$process &

echo "$process restarted."

fi

**Task 23: Write a script to list files by modification time.**

The script should list all files in a given directory, sorted by their modification time.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory

echo "Enter directory: "

read dir

if [ -d "$dir" ]; then

echo "Listing files in $dir sorted by modification time:"

ls -lt "$dir"

else

echo "Directory does not exist."

fi

**Task 24: Write a script to count the number of directories in a given directory.**

The script should count and display the number of subdirectories in the specified directory.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory

echo "Enter directory: "

read dir

if [ -d "$dir" ]; then

dir\_count=$(find "$dir" -type d | wc -l)

echo "Number of directories in $dir: $dir\_count"

else

echo "Directory does not exist."

fi

**Task 25: Write a script to compare two files.**

The script should take two filenames as input and compare their contents. It should display whether they are identical or different.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input filenames

echo "Enter first file: "

read file1

echo "Enter second file: "

read file2

# Compare files

if cmp -s "$file1" "$file2"; then

echo "The files are identical."

else

echo "The files are different."

fi

**Task 26: Write a script to create a simple password generator.**

The script should generate a random password containing uppercase and lowercase letters, digits, and special characters.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Set password length

password\_length=12

# Generate a random password

password=$(< /dev/urandom tr -dc 'A-Za-z0-9@#$%^&\*()\_+' | head -c"$password\_length")

echo "Generated password: $password"

**Task 27: Write a script to display the last N lines of a file.**

The script should take a filename and a number N as input and display the last N lines of the file.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input filename and number of lines

echo "Enter filename: "

read filename

echo "Enter number of lines: "

read n

if [ -e "$filename" ]; then

tail -n "$n" "$filename"

else

echo "File does not exist."

fi

**Task 28: Write a script to check if a directory is empty.**

The script should check if the directory provided is empty or contains files.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory

echo "Enter directory: "

read dir

if [ -d "$dir" ]; then

if [ "$(ls -A $dir)" ]; then

echo "Directory is not empty."

else

echo "Directory is empty."

fi

else

echo "Directory does not exist."

fi

**Task 29: Write a script to compress multiple files into a single zip file.**

The script should take a list of filenames and compress them into a .zip file.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input filenames

echo "Enter files to compress (separate by space): "

read files

# Output zip file name

echo "Enter output zip filename: "

read zip\_file

# Compress files

zip "$zip\_file" $files

echo "Files compressed into $zip\_file."

**Task 30: Write a script to send an email using the mail command.**

The script should send an email with a subject and body to a specified email address.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input email details

echo "Enter recipient email: "

read recipient

echo "Enter subject: "

read subject

echo "Enter body of the email: "

read body

# Send email

echo "$body" | mail -s "$subject" "$recipient"

echo "Email sent to $recipient."

**Task 31: Write a script to display system's memory usage.**

The script should display the total memory, used memory, and free memory.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Display memory usage

free -h

**Task 32: Write a script to print the current user's home directory.**

The script should display the current user's home directory.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Display home directory

echo "Your home directory is: $HOME"

**Task 33: Write a script to change file permissions.**

The script should change the permissions of a given file to rwxr-xr-x (755).

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input filename

echo "Enter filename: "

read filename

if [ -e "$filename" ]; then

chmod 755 "$filename"

echo "Permissions of $filename changed to 755."

else

echo "File does not exist."

fi

**Task 34: Write a script to print the system's hostname.**

The script should display the hostname of the system.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Display hostname

echo "System hostname: $(hostname)"

These tasks will give you additional practice with shell scripting, especially for automating and managing system tasks. Keep modifying and testing the scripts, and try adding additional features to challenge yourself further.

4o

**You said:**

continue

**ChatGPT said:**

Sure! Here are more advanced and diverse shell scripting tasks to help you continue honing your skills:

**Task 35: Write a script to simulate a simple calculator.**

The script should allow users to perform basic arithmetic operations like addition, subtraction, multiplication, and division.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Function to perform calculations

calculate() {

case $operation in

"+")

result=$((num1 + num2))

;;

"-")

result=$((num1 - num2))

;;

"\*")

result=$((num1 \* num2))

;;

"/")

if [ $num2 -eq 0 ]; then

echo "Error: Division by zero is not allowed."

exit 1

fi

result=$((num1 / num2))

;;

\*)

echo "Invalid operation."

exit 1

;;

esac

}

# User input

echo "Enter first number: "

read num1

echo "Enter second number: "

read num2

echo "Enter operation (+, -, \*, /): "

read operation

# Perform calculation

calculate

echo "Result: $result"

**Task 36: Write a script to list all running processes and allow the user to kill a process by PID.**

This script should list running processes and allow the user to kill a process by providing the process ID (PID).

**Solution:**

bash

CopyEdit

#!/bin/bash

# List all running processes

ps aux

# Get PID to kill

echo "Enter the PID of the process to kill: "

read pid

# Kill the process

kill "$pid"

if [ $? -eq 0 ]; then

echo "Process $pid killed successfully."

else

echo "Failed to kill process $pid."

fi

**Task 37: Write a script to sort a list of numbers in ascending order.**

This script should accept a list of numbers as input and sort them in ascending order.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input list of numbers

echo "Enter numbers separated by space: "

read -a numbers

# Sort numbers

sorted\_numbers=($(for num in "${numbers[@]}"; do echo $num; done | sort -n))

# Display sorted numbers

echo "Sorted numbers: ${sorted\_numbers[@]}"

**Task 38: Write a script to generate a random number between 1 and 100.**

The script should generate and print a random number within the given range.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Generate random number between 1 and 100

random\_number=$((RANDOM % 100 + 1))

# Display random number

echo "Generated random number: $random\_number"

**Task 39: Write a script to find the largest and smallest numbers in a list.**

This script should find and display the largest and smallest numbers from a list of numbers provided by the user.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input list of numbers

echo "Enter numbers separated by space: "

read -a numbers

# Initialize largest and smallest

largest=${numbers[0]}

smallest=${numbers[0]}

# Loop through numbers

for num in "${numbers[@]}"; do

if [ "$num" -gt "$largest" ]; then

largest=$num

fi

if [ "$num" -lt "$smallest" ]; then

smallest=$num

fi

done

# Display results

echo "Largest number: $largest"

echo "Smallest number: $smallest"

**Task 40: Write a script to extract email addresses from a text file.**

This script should search for email addresses in a given text file and display them.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input file name

echo "Enter filename: "

read filename

# Check if file exists

if [ -f "$filename" ]; then

# Extract and display email addresses

grep -oP '[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}' "$filename"

else

echo "File does not exist."

fi

**Task 41: Write a script to calculate the factorial of a number.**

This script should take a number as input and calculate its factorial.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input number

echo "Enter a number: "

read num

# Initialize factorial

factorial=1

# Calculate factorial

for ((i=1; i<=num; i++)); do

factorial=$((factorial \* i))

done

# Display result

echo "Factorial of $num is $factorial"

**Task 42: Write a script to backup a directory and keep only the last 5 backups.**

This script should create backups of a directory and ensure that only the last 5 backups are kept.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory

echo "Enter directory to back up: "

read dir

# Check if directory exists

if [ -d "$dir" ]; then

# Get current timestamp

timestamp=$(date +"%Y%m%d%H%M%S")

backup\_dir="backup\_$timestamp"

# Create backup

cp -r "$dir" "$backup\_dir"

echo "Backup of $dir created as $backup\_dir."

# Keep only the last 5 backups

backups=($(ls -1d backup\_\* | sort -r | tail -n +6))

for backup in "${backups[@]}"; do

rm -rf "$backup"

echo "Deleted old backup: $backup"

done

else

echo "Directory does not exist."

fi

**Task 43: Write a script to monitor log files and send an email if a certain keyword is found.**

This script should check for a specific keyword in a log file and send an email notification if the keyword is found.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input log file and keyword

echo "Enter log file path: "

read log\_file

echo "Enter keyword to search for: "

read keyword

# Check if log file exists

if [ -f "$log\_file" ]; then

# Search for the keyword

if grep -q "$keyword" "$log\_file"; then

echo "Keyword '$keyword' found in $log\_file. Sending email notification..."

# Send email

echo "Alert: '$keyword' found in $log\_file" | mail -s "Log Alert" "admin@example.com"

else

echo "Keyword '$keyword' not found in $log\_file."

fi

else

echo "Log file does not exist."

fi

**Task 44: Write a script to monitor system resource usage and log it to a file.**

The script should monitor the system's CPU and memory usage and log it every minute to a file.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Log file

log\_file="system\_usage.log"

# Start logging

echo "Monitoring system resource usage. Press [CTRL+C] to stop."

while true; do

# Get current CPU and memory usage

cpu\_usage=$(top -bn1 | grep "Cpu(s)" | sed "s/.\*, \*\([0-9.]\*\)%\* id.\*/\1/" | awk '{print 100 - $1 "%"}')

memory\_usage=$(free -m | awk 'NR==2{printf "%.2f%%\n", $3\*100/$2 }')

# Log to file

echo "$(date) - CPU Usage: $cpu\_usage - Memory Usage: $memory\_usage" >> "$log\_file"

# Wait for 1 minute

sleep 60

done

**Task 45: Write a script to display disk usage and warn if usage exceeds a threshold.**

The script should monitor the disk usage of the root partition and warn the user if it exceeds 85%.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Set the threshold

threshold=85

# Get disk usage of the root partition

disk\_usage=$(df / | tail -1 | awk '{print $5}' | sed 's/%//')

# Check if disk usage exceeds threshold

if [ "$disk\_usage" -ge "$threshold" ]; then

echo "Warning: Disk usage is at ${disk\_usage}%, which exceeds the threshold of $threshold%."

else

echo "Disk usage is normal: ${disk\_usage}%."

fi

These advanced tasks will further solidify your understanding of shell scripting, covering areas such as file manipulation, resource monitoring, process management, and backups. Keep experimenting with variations and improvements to each script to deepen your knowledge!

4o

**You said:**

continue please some more tasks related to shell scripts that i can expect at HU - with moderrate and hard lvel

**ChatGPT said:**

Absolutely! Here are some additional moderate-to-advanced shell script tasks that you might encounter at HashedIn University (HU) or in similar environments. These tasks cover a variety of scenarios including file manipulation, process management, automation, and performance monitoring.

**Task 46: Write a script to monitor the memory usage of a specific process.**

The script should monitor the memory usage of a process by its name and alert if the memory usage exceeds a specified threshold.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input process name and memory threshold

echo "Enter process name: "

read process\_name

echo "Enter memory usage threshold (in MB): "

read threshold

# Get the memory usage of the process

mem\_usage=$(ps aux | grep "$process\_name" | awk '{print $6}' | sort -n | tail -n 1)

# Convert the memory usage from KB to MB

mem\_usage\_mb=$((mem\_usage / 1024))

# Check if memory usage exceeds threshold

if [ "$mem\_usage\_mb" -ge "$threshold" ]; then

echo "Warning: $process\_name is using $mem\_usage\_mb MB of memory, which exceeds the threshold of $threshold MB."

else

echo "$process\_name is using $mem\_usage\_mb MB of memory, which is below the threshold."

fi

**Task 47: Write a script to sync a directory to a remote server using rsync.**

This script should take a source directory and a destination server as input and sync the contents of the source directory to the destination server using rsync.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input source directory, destination server, and remote directory

echo "Enter source directory: "

read source\_dir

echo "Enter destination server (user@host): "

read dest\_server

echo "Enter remote directory: "

read remote\_dir

# Sync using rsync

rsync -avz --delete "$source\_dir" "$dest\_server:$remote\_dir"

echo "Sync complete from $source\_dir to $dest\_server:$remote\_dir"

**Task 48: Write a script to recursively change the ownership of all files in a directory.**

The script should take a directory path and a user/group as input, and recursively change the ownership of all files and subdirectories within that directory.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory, user, and group

echo "Enter directory path: "

read dir

echo "Enter user (e.g., username): "

read user

echo "Enter group (e.g., groupname): "

read group

# Change ownership recursively

chown -R "$user":"$group" "$dir"

echo "Ownership of files in $dir changed to $user:$group."

**Task 49: Write a script to create a cron job for automatic backups.**

This script should create a cron job that runs daily at midnight to back up a directory to a specified backup location.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input source directory and backup location

echo "Enter source directory for backup: "

read source\_dir

echo "Enter backup destination directory: "

read backup\_dir

# Set the cron job for daily backup at midnight

(crontab -l ; echo "0 0 \* \* \* cp -r $source\_dir $backup\_dir/backup\_$(date +\%Y\%m\%d)") | crontab -

echo "Cron job created to back up $source\_dir to $backup\_dir daily at midnight."

**Task 50: Write a script to find the most recently modified file in a directory.**

The script should accept a directory as input and find the most recently modified file in that directory.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory

echo "Enter directory: "

read dir

# Find most recently modified file

recent\_file=$(ls -t "$dir" | head -n 1)

echo "Most recently modified file in $dir is: $recent\_file"

**Task 51: Write a script to rotate log files.**

The script should compress and archive log files when they exceed a certain size (e.g., 10MB) and keep the last 5 compressed backups.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input log file and size threshold

echo "Enter log file path: "

read log\_file

echo "Enter size threshold in MB: "

read size\_threshold

# Get log file size in MB

log\_size=$(du -m "$log\_file" | cut -f1)

# Check if log file size exceeds threshold

if [ "$log\_size" -ge "$size\_threshold" ]; then

# Create a backup with timestamp

timestamp=$(date +"%Y%m%d%H%M%S")

gzip -c "$log\_file" > "${log\_file}\_$timestamp.gz"

echo "Log file compressed as ${log\_file}\_$timestamp.gz."

# Keep only the last 5 compressed backups

backups=($(ls -1t ${log\_file}\_\*.gz | tail -n +6))

for backup in "${backups[@]}"; do

rm "$backup"

echo "Deleted old backup: $backup"

done

else

echo "Log file size is below threshold. No action taken."

fi

**Task 52: Write a script to monitor a directory for changes.**

The script should monitor a directory for any new files or changes and log the event.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory to monitor

echo "Enter directory to monitor: "

read dir

# Monitor the directory using inotifywait

inotifywait -m "$dir" -e create -e modify |

while read path action file; do

echo "$(date) - $action - $file in $path" >> directory\_changes.log

done

**Task 53: Write a script to limit the CPU usage of a process.**

This script should limit the CPU usage of a given process to a specified percentage.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input process name and CPU usage limit

echo "Enter process name: "

read process\_name

echo "Enter CPU usage limit (in percentage): "

read cpu\_limit

# Find the process ID (PID) of the process

pid=$(pgrep "$process\_name")

if [ -z "$pid" ]; then

echo "Process $process\_name not found."

exit 1

fi

# Set CPU limit using cpulimit

cpulimit -p "$pid" -l "$cpu\_limit" &

echo "CPU usage of $process\_name (PID $pid) limited to $cpu\_limit%."

**Task 54: Write a script to create a user and set a password with validation.**

The script should create a new user and assign a password. It should also validate the password strength (e.g., at least 8 characters, one uppercase, one digit, and one special character).

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input username

echo "Enter new username: "

read username

# Input password and validate strength

while true; do

echo "Enter password (min 8 characters, 1 uppercase, 1 number, 1 special character): "

read -s password

if [[ ${#password} -ge 8 && "$password" =~ [A-Z] && "$password" =~ [0-9] && "$password" =~ [^a-zA-Z0-9] ]]; then

break

else

echo "Password does not meet the strength criteria. Please try again."

fi

done

# Create the user and set the password

useradd "$username"

echo "$username:$password" | chpasswd

echo "User $username created successfully with a secure password."

**Task 55: Write a script to check for zombie processes.**

This script should find and list any zombie processes on the system.

**Solution:**

bash

CopyEdit

#!/bin/bash

# List zombie processes

zombies=$(ps aux | grep 'Z' | awk '{print $2, $11, $12}')

if [ -z "$zombies" ]; then

echo "No zombie processes found."

else

echo "Zombie processes found:"

echo "$zombies"

fi

**Task 56: Write a script to calculate the average size of files in a directory.**

The script should calculate the average size of all files in a given directory.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory

echo "Enter directory: "

read dir

# Calculate total size of files and number of files

total\_size=0

file\_count=0

for file in "$dir"/\*; do

if [ -f "$file" ]; then

total\_size=$((total\_size + $(stat -c %s "$file")))

file\_count=$((file\_count + 1))

fi

done

if [ "$file\_count" -gt 0 ]; then

average\_size=$((total\_size / file\_count))

echo "Average file size in $dir: $average\_size bytes."

else

echo "No files found in $dir."

fi

**Task 57: Write a script to create a compressed archive of a directory excluding certain files.**

The script should take a directory and a list of file types to exclude (e.g., .log, .tmp) and create a compressed .tar.gz archive of the directory, excluding those file types.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory and exclude file types

echo "Enter directory: "

read dir

echo "Enter file types to exclude (e.g., .log .tmp): "

read -a exclude\_types

# Create the exclude argument for tar command

exclude\_args=""

for ext in "${exclude\_types[@]}"; do

exclude\_args="$exclude\_args --exclude=\*${ext}"

done

# Create the tar archive

tar -czf "${dir}\_backup.tar.gz" $exclude\_args "$dir"

echo "Directory $dir archived as ${dir}\_backup.tar.gz excluding ${exclude\_types[@]} files."

**Task 58: Write a script to compare two directories.**

The script should compare two directories and list the differences (files that are in one directory but not the other).

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input two directories

echo "Enter first directory: "

read dir1

echo "Enter second directory: "

read dir2

# Compare directories

diff -r "$dir1" "$dir2" > dir\_diff.log

if [ -s dir\_diff.log ]; then

echo "Differences found between $dir1 and $dir2. See dir\_diff.log for details."

else

echo "No differences found between $dir1 and $dir2."

fi

These tasks are designed to challenge you while also preparing you for practical, real-world use cases in system administration, file management, and automation. They require a moderate to high understanding of shell scripting, and they will help you refine your skills further.

4o

**You said:**

continue please

**ChatGPT said:**

Sure! Here are more challenging and moderate-to-hard shell script tasks you can expect, which are useful for enhancing your problem-solving and system administration skills.

**Task 59: Write a script to detect and report disk failures (SMART status check).**

This script should check the SMART (Self-Monitoring, Analysis, and Reporting Technology) status of all disks on the system and report any failing disks.

**Solution:**

bash

CopyEdit

#!/bin/bash

# List all disks

disks=$(lsblk -d -o name | grep -v "NAME")

# Check SMART status for each disk

for disk in $disks; do

smartctl -H /dev/$disk | grep -i "SMART overall-health"

done

**Task 60: Write a script to manage process priority using nice and renice.**

The script should allow the user to change the priority of a running process or start a new process with a specific priority.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Menu to choose action

echo "1. Start a new process with a specific priority"

echo "2. Change priority of an existing process"

echo "Enter your choice: "

read choice

if [ "$choice" -eq 1 ]; then

# Start a new process with a specific priority

echo "Enter command to start the process: "

read command

echo "Enter priority value (negative values increase priority, positive decrease): "

read priority

nice -n "$priority" $command

elif [ "$choice" -eq 2 ]; then

# Change the priority of an existing process

echo "Enter the PID of the process: "

read pid

echo "Enter new priority value (negative values increase priority, positive decrease): "

read priority

renice -n "$priority" -p "$pid"

else

echo "Invalid choice"

fi

**Task 61: Write a script to limit disk space usage per user (quota management).**

The script should take a username and limit the disk space usage for that user to a specified amount using setquota.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input username and disk space limit

echo "Enter username: "

read username

echo "Enter soft disk limit (in KB): "

read soft\_limit

echo "Enter hard disk limit (in KB): "

read hard\_limit

# Set the disk space limit using setquota

setquota -u "$username" "$soft\_limit" "$hard\_limit" 0 0 /

echo "Disk space limits set for user $username: Soft limit = $soft\_limit KB, Hard limit = $hard\_limit KB."

**Task 62: Write a script to schedule and automate file downloads using wget.**

The script should accept a URL and a time interval (in minutes) and download the file periodically.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input URL and time interval

echo "Enter URL to download from: "

read url

echo "Enter time interval (in minutes): "

read interval

# Download file periodically

while true; do

wget "$url"

echo "Downloaded $url at $(date)"

sleep $((interval \* 60)) # Sleep for the specified time interval

done

**Task 63: Write a script to monitor system logs and send an email if a critical error is found.**

The script should monitor a system log file (e.g., /var/log/syslog or /var/log/messages) for any critical errors and send an email notification if any are found.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Log file and email settings

log\_file="/var/log/syslog"

email="admin@example.com"

# Monitor the log file for critical errors

tail -f "$log\_file" | while read line; do

if echo "$line" | grep -q "CRITICAL"; then

echo "$line" | mail -s "Critical error detected in $log\_file" "$email"

echo "Sent email about critical error."

fi

done

**Task 64: Write a script to create incremental backups.**

The script should create incremental backups of a directory, keeping track of changes since the last backup.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input source directory and backup directory

echo "Enter source directory: "

read source\_dir

echo "Enter backup directory: "

read backup\_dir

# Create an incremental backup

backup\_file="$backup\_dir/backup\_$(date +%Y%m%d%H%M%S).tar.gz"

tar --create --file="$backup\_file" --listed-incremental="$backup\_dir/incremental.snar" "$source\_dir"

echo "Incremental backup created at $backup\_file."

**Task 65: Write a script to check for orphaned packages and remove them.**

The script should check for orphaned packages on a Debian-based system (e.g., Ubuntu) and remove them.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Check for orphaned packages and remove them

echo "Checking for orphaned packages..."

orphans=$(deborphan)

if [ -n "$orphans" ]; then

echo "Orphaned packages found: $orphans"

echo "Removing orphaned packages..."

sudo apt-get remove --purge $orphans

echo "Orphaned packages removed."

else

echo "No orphaned packages found."

fi

**Task 66: Write a script to monitor and limit bandwidth usage for specific processes.**

The script should allow users to monitor and limit the network bandwidth usage of specific processes using tc.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input process ID and bandwidth limit

echo "Enter process ID (PID): "

read pid

echo "Enter bandwidth limit (in KBps): "

read bandwidth

# Set bandwidth limit using tc (traffic control)

tc qdisc add dev eth0 root handle 1: htb default 30

tc class add dev eth0 parent 1: classid 1:1 htb rate ${bandwidth}kbit

tc filter add dev eth0 parent 1: protocol ip prio 1 handle 1 fw flowid 1:1

# Apply bandwidth limit to the specified PID

iptables -A OUTPUT -t mangle -p tcp --sport 80 -m owner --pid-owner "$pid" -j MARK --set-mark 1

echo "Bandwidth limit set for process $pid: $bandwidth KBps."

**Task 67: Write a script to check the health of a web server.**

The script should check if a web server (e.g., Apache or Nginx) is running and accessible, and return the HTTP status code.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input server URL

echo "Enter web server URL (e.g., http://localhost): "

read url

# Check the HTTP status code

status\_code=$(curl -o /dev/null -s -w "%{http\_code}" "$url")

if [ "$status\_code" -eq 200 ]; then

echo "Web server is running. HTTP Status: 200 OK"

else

echo "Web server is not accessible. HTTP Status: $status\_code"

fi

**Task 68: Write a script to automate SSL certificate renewal using certbot.**

This script should automate the renewal of SSL certificates for a website using certbot.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Automate SSL certificate renewal

echo "Renewing SSL certificates..."

sudo certbot renew

# Restart the web server to apply the renewed certificate

echo "Restarting web server..."

sudo systemctl restart apache2

echo "SSL certificate renewal and web server restart completed."

**Task 69: Write a script to backup MySQL databases and store them with a timestamp.**

The script should back up all MySQL databases and store them with the current date and time as the filename.

**Solution:**

bash

CopyEdit

#!/bin/bash

# MySQL credentials

echo "Enter MySQL username: "

read user

echo "Enter MySQL password: "

read -s password

# Get the current timestamp

timestamp=$(date +"%Y%m%d%H%M%S")

backup\_dir="/path/to/backup/dir"

backup\_file="$backup\_dir/mysql\_backup\_$timestamp.sql"

# Backup all databases

mysqldump -u "$user" -p"$password" --all-databases > "$backup\_file"

if [ $? -eq 0 ]; then

echo "MySQL backup created at $backup\_file."

else

echo "Backup failed."

fi

**Task 70: Write a script to synchronize two directories (bidirectional sync).**

The script should synchronize files in both directions between two directories (e.g., local and remote) using rsync.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directories

echo "Enter the first directory: "

read dir1

echo "Enter the second directory: "

read dir2

# Sync both directories

rsync -av --delete "$dir1/" "$dir2/"

rsync -av --delete "$dir2/" "$dir1/"

echo "Directories $dir1 and $dir2 are synchronized."

These tasks will help you develop advanced shell scripting skills in various areas such as system administration, server management, security, backup automation, and traffic control. Each task presents a practical use case that you can adapt to your own needs or modify to address more complex requirements.

4o

**You said:**

continue please

**ChatGPT said:**

Absolutely! Here are more advanced shell scripting tasks, providing practical and real-world solutions for system administration, automation, and troubleshooting scenarios.

**Task 71: Write a script to check disk usage and automatically send an alert if it exceeds a threshold.**

The script should monitor disk usage and send an email alert when the disk usage exceeds a specified threshold.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Set the threshold (in percentage)

threshold=80

email="admin@example.com"

# Get the disk usage of the root partition

disk\_usage=$(df / | tail -1 | awk '{print $5}' | sed 's/%//')

# Check if disk usage exceeds threshold

if [ "$disk\_usage" -ge "$threshold" ]; then

echo "Disk usage is at ${disk\_usage}% which exceeds the threshold of ${threshold}%. Sending an alert to $email."

echo "Warning: Disk usage exceeds threshold! Current usage: ${disk\_usage}%" | mail -s "Disk Usage Alert" "$email"

else

echo "Disk usage is normal: ${disk\_usage}%."

fi

**Task 72: Write a script to find and delete empty directories.**

This script should scan a directory and its subdirectories to find empty directories and delete them.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory

echo "Enter directory to search for empty directories: "

read dir

# Find and delete empty directories

find "$dir" -type d -empty -exec rmdir {} \;

echo "Empty directories have been deleted from $dir."

**Task 73: Write a script to securely copy files to a remote server using SCP.**

This script should allow you to securely copy files from a local system to a remote server using scp.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input details for SCP

echo "Enter source file to copy: "

read source\_file

echo "Enter destination server (user@host): "

read dest\_server

echo "Enter destination path: "

read dest\_path

# Securely copy the file using SCP

scp "$source\_file" "$dest\_server:$dest\_path"

if [ $? -eq 0 ]; then

echo "File copied successfully to $dest\_server:$dest\_path."

else

echo "File copy failed."

fi

**Task 74: Write a script to create a new virtual machine (VM) using virt-install.**

This script should automate the creation of a new virtual machine using virt-install.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input virtual machine details

echo "Enter VM name: "

read vm\_name

echo "Enter CPU count: "

read cpu\_count

echo "Enter memory size (in MB): "

read memory\_size

echo "Enter disk size (in GB): "

read disk\_size

echo "Enter OS image path: "

read os\_image

# Create the VM using virt-install

virt-install --name "$vm\_name" --vcpus "$cpu\_count" --memory "$memory\_size" --disk size="$disk\_size" --cdrom "$os\_image" --os-type linux --os-variant ubuntu20.04 --network network=default --graphics none --console pty,target\_type=serial --autostart

echo "VM $vm\_name created successfully."

**Task 75: Write a script to download a file using HTTP and handle download failure retries.**

The script should attempt to download a file and retry a specified number of times if the download fails.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input URL and file name

echo "Enter URL to download: "

read url

echo "Enter the filename to save the file as: "

read file\_name

echo "Enter the number of retries: "

read retries

# Download the file with retries

count=0

while [ $count -lt $retries ]; do

curl -O "$url" -o "$file\_name"

if [ $? -eq 0 ]; then

echo "Download successful."

exit 0

else

echo "Download failed. Retrying... ($((count + 1))/$retries)"

fi

count=$((count + 1))

done

echo "Download failed after $retries retries."

**Task 76: Write a script to automatically update system packages and send an email notification.**

The script should update all installed packages on a system and send a notification email about the update status.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Email details

email="admin@example.com"

# Update system packages

echo "Starting system update..."

sudo apt-get update && sudo apt-get upgrade -y

# Check if the update was successful

if [ $? -eq 0 ]; then

echo "System update completed successfully." | mail -s "System Update Status" "$email"

else

echo "System update failed." | mail -s "System Update Status" "$email"

fi

**Task 77: Write a script to synchronize system time with NTP server.**

The script should sync the system time with an NTP server to ensure the system time is accurate.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Sync system time with NTP server

echo "Synchronizing system time with NTP server..."

sudo ntpdate pool.ntp.org

if [ $? -eq 0 ]; then

echo "System time synchronized successfully."

else

echo "Failed to synchronize system time."

fi

**Task 78: Write a script to clean up old log files and compress them.**

The script should delete log files older than a specified number of days and compress remaining logs.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Set the number of days to keep log files

days=30

log\_dir="/var/log"

# Delete logs older than the specified days

find "$log\_dir" -type f -name "\*.log" -mtime +$days -exec rm -f {} \;

# Compress remaining log files

find "$log\_dir" -type f -name "\*.log" -exec gzip {} \;

echo "Old log files deleted and remaining logs compressed."

**Task 79: Write a script to create an SSH key pair and deploy it to a remote server.**

The script should generate an SSH key pair and deploy the public key to a remote server for passwordless authentication.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input server details

echo "Enter remote server (user@host): "

read remote\_server

# Generate SSH key pair

echo "Generating SSH key pair..."

ssh-keygen -t rsa -b 2048 -f ~/.ssh/id\_rsa -N ""

# Deploy the public key to the remote server

ssh-copy-id "$remote\_server"

echo "SSH key pair created and deployed to $remote\_server."

**Task 80: Write a script to perform a vulnerability scan on the system using lynis.**

This script should run a security audit using lynis and save the results to a log file.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Run a Lynis security audit

echo "Performing security audit using Lynis..."

# Run the audit and save results to a log file

sudo lynis audit system > /var/log/lynis\_audit.log

echo "Security audit completed. Results saved to /var/log/lynis\_audit.log."

**Task 81: Write a script to monitor CPU and memory usage and log it periodically.**

The script should monitor CPU and memory usage every minute and log it to a file for analysis.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Set the log file location

log\_file="/var/log/cpu\_mem\_usage.log"

# Log system usage periodically

while true; do

echo "$(date) - CPU Usage: $(top -bn1 | grep "Cpu(s)" | sed "s/.\*, \*\([0-9.]\*\)%\* id.\*/\1/" | awk '{print 100 - $1 "%"}'), Memory Usage: $(free -h | awk 'NR==2{print $3 "/" $2 " (" $3/$2 \* 100 "%)"}')" >> "$log\_file"

sleep 60

done

**Task 82: Write a script to monitor and alert for high CPU usage.**

This script should monitor CPU usage and alert if the usage exceeds a certain threshold (e.g., 90%).

**Solution:**

bash

CopyEdit

#!/bin/bash

# Set the CPU usage threshold

threshold=90

# Get current CPU usage

cpu\_usage=$(top -bn1 | grep "Cpu(s)" | sed "s/.\*, \*\([0-9.]\*\)%\* id.\*/\1/" | awk '{print 100 - $1}')

# Check if CPU usage exceeds the threshold

if (( $(echo "$cpu\_usage > $threshold" | bc -l) )); then

echo "Warning: CPU usage is at $cpu\_usage%, which exceeds the threshold of $threshold%." | mail -s "CPU Usage Alert" "admin@example.com"

else

echo "CPU usage is normal: $cpu\_usage%"

fi

**Task 83: Write a script to archive and email a directory.**

This script should create a compressed archive of a directory and email it to a specified address.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory and email details

echo "Enter directory to archive: "

read dir

echo "Enter recipient email address: "

read email

# Create an archive of the directory

archive\_name=$(basename "$dir")\_$(date +"%Y%m%d%H%M%S").tar.gz

tar -czf "$archive\_name" "$dir"

# Send the archive via email

echo "Sending archive $archive\_name to $email..."

echo "Here is the requested archive: $archive\_name" | mail -s "Directory Archive" -A "$archive\_name" "$email"

echo "Archive sent successfully."

**Task 84: Write a script to clean the system cache (e.g., package manager and system cache).**

The script should clean the package manager cache and system cache to free up space.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Clean package manager cache

echo "Cleaning package manager cache..."

sudo apt-get clean

# Clean system cache

echo "Cleaning system cache..."

sudo rm -rf /var/cache/\*

echo "System cache cleaned."

These tasks involve a wide range of practical use cases that will challenge your understanding of shell scripting, system administration, automation, and security. Completing them will help you become more proficient in shell scripting and handling advanced system management tasks.

4o

**You said:**

continue with some hard task now that i can expect

**ChatGPT said:**

Sure! Here are some challenging shell scripting tasks that require a deep understanding of Linux systems, scripting techniques, and system administration. These tasks will test your ability to automate complex workflows, manage system resources, troubleshoot, and secure your system effectively.

**Task 85: Write a script to create a custom system monitoring tool.**

The script should display detailed system information like CPU usage, memory usage, disk usage, and network traffic every 10 seconds and log it to a file.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Log file location

log\_file="/var/log/system\_monitor.log"

# Function to get system stats

get\_system\_stats() {

# Get CPU usage

cpu\_usage=$(top -bn1 | grep "Cpu(s)" | sed "s/.\*, \*\([0-9.]\*\)%\* id.\*/\1/" | awk '{print 100 - $1 "%"}')

# Get memory usage

memory\_usage=$(free -h | awk 'NR==2{printf "%.2f%%\n", $3\*100/$2 }')

# Get disk usage

disk\_usage=$(df -h | grep "^/dev" | awk '{print $1 ": " $5}')

# Get network traffic (bytes sent/received)

network\_traffic=$(netstat -i | grep 'eth0' | awk '{print $3, $7}')

# Log system stats

echo "$(date) - CPU: $cpu\_usage, Memory: $memory\_usage, Disk: $disk\_usage, Network Traffic: $network\_traffic" >> "$log\_file"

}

# Monitor system stats every 10 seconds

while true; do

get\_system\_stats

sleep 10

done

**Task 86: Write a script to create a custom backup solution with versioning.**

The script should back up a specified directory, create a timestamped backup, and maintain a versioning system by keeping only the last 5 backups.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input directory and backup location

echo "Enter directory to back up: "

read source\_dir

echo "Enter backup destination directory: "

read backup\_dir

# Create a timestamped backup name

timestamp=$(date +"%Y%m%d%H%M%S")

backup\_name="backup\_$timestamp.tar.gz"

backup\_path="$backup\_dir/$backup\_name"

# Create the backup using tar

tar -czf "$backup\_path" "$source\_dir"

# Maintain versioning: Keep only the last 5 backups

backups=($(ls -1t $backup\_dir/backup\_\*.tar.gz | tail -n +6))

for backup in "${backups[@]}"; do

rm "$backup"

echo "Deleted old backup: $backup"

done

echo "Backup created: $backup\_path"

**Task 87: Write a script to securely wipe a disk or directory.**

The script should securely erase all data from a specified disk or directory by overwriting it multiple times with random data, making recovery impossible.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input the directory or disk to wipe

echo "Enter the directory or disk to securely erase: "

read target

# Check if it's a valid path

if [ ! -e "$target" ]; then

echo "Invalid directory or disk."

exit 1

fi

# Overwrite the data multiple times

if [ -d "$target" ]; then

echo "Wiping directory $target..."

# Using shred command to securely delete files

find "$target" -type f -exec shred -u -z {} \;

rmdir "$target"

else

echo "Wiping disk $target..."

# Using dd to overwrite disk with random data

dd if=/dev/urandom of="$target" bs=1M status=progress

sync

echo "Disk $target securely wiped."

fi

**Task 88: Write a script to automatically generate SSH keys for all users.**

This script should generate SSH keys for all users on the system who don’t already have an SSH key pair and deploy them to a central location.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Directory to store SSH public keys

keys\_dir="/home/ssh\_keys"

# Create the directory if it doesn't exist

mkdir -p "$keys\_dir"

# Loop through all users

for user in $(awk -F: '{print $1}' /etc/passwd); do

user\_dir="/home/$user"

# Check if the user has a home directory and doesn't already have an SSH key

if [ -d "$user\_dir" ] && [ ! -f "$user\_dir/.ssh/id\_rsa" ]; then

echo "Generating SSH key for user $user..."

# Generate SSH key pair for the user

sudo -u "$user" ssh-keygen -t rsa -b 2048 -f "$user\_dir/.ssh/id\_rsa" -N ""

# Copy the public key to the central keys directory

cp "$user\_dir/.ssh/id\_rsa.pub" "$keys\_dir/$user.pub"

echo "SSH key for $user generated and stored in $keys\_dir."

fi

done

echo "SSH key generation and deployment completed."

**Task 89: Write a script to create and configure an isolated Docker container for a specific application.**

The script should automate the process of creating a Docker container for a specified application, install the necessary dependencies, and start the container.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input application name and version

echo "Enter application name: "

read app\_name

echo "Enter application version (e.g., 1.0): "

read app\_version

# Docker image and container names

image\_name="$app\_name:$app\_version"

container\_name="$app\_name"\_container

# Check if Docker is installed

if ! command -v docker &> /dev/null; then

echo "Docker is not installed. Please install Docker first."

exit 1

fi

# Pull the Docker image

echo "Pulling Docker image for $app\_name:$app\_version..."

docker pull "$image\_name"

# Create and start the Docker container

echo "Creating and starting container for $app\_name..."

docker run -d --name "$container\_name" "$image\_name"

# Install necessary dependencies (if any)

echo "Installing dependencies inside the container..."

docker exec -it "$container\_name" apt-get update

docker exec -it "$container\_name" apt-get install -y dependency1 dependency2

echo "$app\_name container setup and started successfully."

**Task 90: Write a script to implement a system health check and report all critical issues.**

The script should check the system for critical issues such as disk usage, CPU load, memory usage, and failed services, then generate a detailed report.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Output file for the health report

report\_file="/var/log/system\_health\_report\_$(date +%Y%m%d%H%M%S).log"

# Create the report header

echo "System Health Check Report - $(date)" > "$report\_file"

echo "-------------------------------------" >> "$report\_file"

# Check disk usage

echo "Checking disk usage..." >> "$report\_file"

disk\_usage=$(df / | tail -1 | awk '{print $5}' | sed 's/%//')

if [ "$disk\_usage" -ge 85 ]; then

echo "Warning: Disk usage is at ${disk\_usage}%." >> "$report\_file"

else

echo "Disk usage is normal: ${disk\_usage}%" >> "$report\_file"

fi

# Check CPU load

echo "Checking CPU load..." >> "$report\_file"

cpu\_load=$(uptime | awk -F'load average:' '{print $2}' | cut -d, -f1)

if (( $(echo "$cpu\_load > 1.0" | bc -l) )); then

echo "Warning: CPU load is high: $cpu\_load." >> "$report\_file"

else

echo "CPU load is normal: $cpu\_load." >> "$report\_file"

fi

# Check memory usage

echo "Checking memory usage..." >> "$report\_file"

memory\_usage=$(free -h | awk 'NR==2{print $3 "/" $2}')

if [[ $memory\_usage == \*"G"\* ]]; then

echo "Memory usage is $memory\_usage" >> "$report\_file"

else

echo "Memory usage is normal: $memory\_usage" >> "$report\_file"

fi

# Check for failed services

echo "Checking failed services..." >> "$report\_file"

failed\_services=$(systemctl --failed)

if [ -n "$failed\_services" ]; then

echo "Warning: The following services have failed:" >> "$report\_file"

echo "$failed\_services" >> "$report\_file"

else

echo "No failed services found." >> "$report\_file"

fi

# Display report

cat "$report\_file"

**Task 91: Write a script to create a custom user permission management system.**

The script should automate the creation of a user, set specific file permissions for their directories, and log changes for auditing.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Input username and directories

echo "Enter username: "

read username

echo "Enter directories to assign permissions (separate with space): "

read -a directories

# Create user if doesn't exist

if ! id "$username" &>/dev/null; then

echo "Creating user $username..."

useradd "$username"

echo "User $username created."

fi

# Set permissions for each directory

for dir in "${directories[@]}"; do

if [ -d "$dir" ]; then

echo "Setting permissions for $dir..."

chmod 700 "$dir"

chown "$username:$username" "$dir"

echo "Permissions set for $dir"

else

echo "Directory $dir does not exist. Skipping."

fi

done

# Log the changes

log\_file="/var/log/user\_permissions\_$username.log"

echo "User $username permissions update - $(date)" > "$log\_file"

for dir in "${directories[@]}"; do

echo "$dir - Permissions: $(ls -ld $dir)" >> "$log\_file"

done

echo "Permission changes logged to $log\_file."

**Task 92: Write a script to automate a disaster recovery plan.**

The script should automatically back up critical data, including databases and configuration files, and store them in a safe location.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Backup directories and databases

backup\_dir="/backups/$(date +%Y%m%d)"

mkdir -p "$backup\_dir"

# Backup configuration files

echo "Backing up configuration files..."

cp -r /etc "$backup\_dir/etc\_config"

# Backup database (assuming MySQL)

echo "Backing up MySQL databases..."

mysqldump -u root -p --all-databases > "$backup\_dir/mysql\_backup.sql"

# Backup home directories

echo "Backing up user home directories..."

cp -r /home "$backup\_dir/home"

# Compress the backup directory

tar -czf "$backup\_dir.tar.gz" -C /backups "$(basename "$backup\_dir")"

# Send notification

echo "Backup completed successfully. Archive created: $backup\_dir.tar.gz"

These advanced tasks involve automating critical system administration tasks, handling backups, securing data, and performing system health checks, which are essential for managing complex infrastructures and ensuring system stability and security. Completing these tasks will give you a deep understanding of shell scripting and its application in real-world environments.

4o

**You said:**

continue withbasic and moderate shell scripts assignmetn that i can expect please

**ChatGPT said:**

Certainly! Here are some basic to moderate shell script assignments you might expect, covering a wide range of common tasks. These assignments will help you build foundational skills and enhance your understanding of scripting in a Unix/Linux environment.

**Task 93: Write a script to print "Hello, World!"**

A simple script that outputs "Hello, World!" to the terminal.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Hello, World!"

**Task 94: Write a script to display the current date and time.**

This script will print the current date and time in the format Day Month Date Time Year.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Current date and time: $(date)"

**Task 95: Write a script to check if a file exists.**

The script should take a filename as input and check if the file exists, outputting a message accordingly.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter filename: "

read filename

if [ -e "$filename" ]; then

echo "File exists."

else

echo "File does not exist."

fi

**Task 96: Write a script to list all files in a directory.**

This script should list all files in a specified directory, excluding directories, and print each file name.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter directory path: "

read dir

if [ -d "$dir" ]; then

echo "Files in $dir:"

for file in "$dir"/\*; do

if [ -f "$file" ]; then

echo "$(basename "$file")"

fi

done

else

echo "Directory does not exist."

fi

**Task 97: Write a script to count the number of words in a text file.**

The script should take a file as input and output the number of words in the file.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter the filename: "

read filename

if [ -e "$filename" ]; then

word\_count=$(wc -w < "$filename")

echo "Number of words in $filename: $word\_count"

else

echo "File does not exist."

fi

**Task 98: Write a script to calculate the factorial of a number.**

This script should take a number as input and output the factorial of that number.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter a number: "

read num

factorial=1

for (( i=1; i<=num; i++ )); do

factorial=$((factorial \* i))

done

echo "Factorial of $num is $factorial"

**Task 99: Write a script to check if a user is logged in.**

This script should check if a specific user is logged into the system.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter username: "

read username

if who | grep -q "$username"; then

echo "$username is logged in."

else

echo "$username is not logged in."

fi

**Task 100: Write a script to reverse a string.**

This script should reverse a string provided by the user.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter a string: "

read str

reversed\_str=$(echo "$str" | rev)

echo "Reversed string: $reversed\_str"

**Task 101: Write a script to check if a number is even or odd.**

This script should check if a number is even or odd and print the result.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter a number: "

read num

if (( num % 2 == 0 )); then

echo "$num is even."

else

echo "$num is odd."

fi

**Task 102: Write a script to find the largest number among three numbers.**

This script should accept three numbers from the user and print the largest one.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter three numbers:"

read num1

read num2

read num3

if (( num1 >= num2 && num1 >= num3 )); then

echo "$num1 is the largest."

elif (( num2 >= num1 && num2 >= num3 )); then

echo "$num2 is the largest."

else

echo "$num3 is the largest."

fi

**Task 103: Write a script to replace a word in a file.**

This script should take a word to find and replace in a file. It should replace all occurrences of the word.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter the file name: "

read filename

echo "Enter the word to find: "

read find\_word

echo "Enter the word to replace with: "

read replace\_word

if [ -e "$filename" ]; then

sed -i "s/$find\_word/$replace\_word/g" "$filename"

echo "All occurrences of '$find\_word' have been replaced with '$replace\_word'."

else

echo "File does not exist."

fi

**Task 104: Write a script to generate a random password.**

This script should generate a random password of a given length, using uppercase letters, lowercase letters, numbers, and special characters.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter the length of the password: "

read length

# Generate random password

password=$(< /dev/urandom tr -dc 'A-Za-z0-9@#$%^&\*()\_+' | head -c"$length")

echo "Generated password: $password"

**Task 105: Write a script to create a directory and change into it.**

The script should create a directory with a given name and then change into that directory.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter directory name: "

read dir\_name

mkdir "$dir\_name" && cd "$dir\_name"

echo "Changed to directory $dir\_name"

**Task 106: Write a script to copy a file to a backup directory with timestamp.**

The script should copy a file to a backup directory and append a timestamp to the filename.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter the filename to back up: "

read filename

echo "Enter backup directory: "

read backup\_dir

if [ -e "$filename" ]; then

timestamp=$(date +"%Y%m%d%H%M%S")

cp "$filename" "$backup\_dir/$(basename "$filename")\_$timestamp"

echo "Backup of $filename created as $(basename "$filename")\_$timestamp in $backup\_dir."

else

echo "File does not exist."

fi

**Task 107: Write a script to display a countdown timer.**

This script should accept a number of seconds from the user and display a countdown to zero.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter number of seconds for countdown: "

read seconds

while [ $seconds -gt 0 ]; do

echo "$seconds seconds remaining"

((seconds--))

sleep 1

done

echo "Time's up!"

**Task 108: Write a script to search for a file in a directory.**

The script should search for a file by its name in a given directory and print if it exists or not.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter directory to search in: "

read dir

echo "Enter filename to search for: "

read filename

if [ -e "$dir/$filename" ]; then

echo "File $filename found in $dir."

else

echo "File $filename not found in $dir."

fi

**Task 109: Write a script to change the extension of a file.**

The script should take a filename and change its extension to a new one.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter the filename: "

read filename

echo "Enter new extension: "

read new\_extension

new\_filename="${filename%.\*}.$new\_extension"

mv "$filename" "$new\_filename"

echo "File extension changed to $new\_filename."

**Task 110: Write a script to display the number of files in a directory.**

This script should count and display the number of files in a specified directory.

**Solution:**

bash

CopyEdit

#!/bin/bash

echo "Enter directory path: "

read dir

if [ -d "$dir" ]; then

file\_count=$(find "$dir" -type f | wc -l)

echo "Number of files in $dir: $file\_count"

else

echo "Directory does not exist."

fi

These assignments cover a variety of fundamental tasks that help you get comfortable with shell scripting. They focus on file manipulation, loops, conditionals, and basic operations, which are core concepts of shell programming. Once you master these, you can move on to more advanced tasks that involve automation and system administration.

4o

**You said:**

some shell scirpts that are used in industries like for automation montiroing

**ChatGPT said:**

Certainly! In industrial settings, shell scripts are often used for automation, monitoring, and managing various system tasks efficiently. Below are several examples of shell scripts commonly used in industries for these purposes. These scripts help in tasks such as log management, system health checks, backup automation, and network monitoring.

**Task 111: Automated System Health Check Script**

This script checks the health of the system by monitoring CPU usage, memory usage, disk space, and network connectivity. It can be scheduled to run periodically (e.g., via cron job).

**Solution:**

bash

CopyEdit

#!/bin/bash

# Define output log file

log\_file="/var/log/system\_health\_check.log"

# Get current date and time

current\_time=$(date "+%Y-%m-%d %H:%M:%S")

# Check CPU usage

cpu\_usage=$(top -bn1 | grep "Cpu(s)" | sed "s/.\*, \*\([0-9.]\*\)%\* id.\*/\1/" | awk '{print 100 - $1 "%"}')

# Check Memory usage

memory\_usage=$(free -h | awk 'NR==2{printf "%.2f%%\n", $3\*100/$2 }')

# Check disk usage

disk\_usage=$(df -h | grep "^/dev" | awk '{print $1 ": " $5}')

# Check network connectivity (Ping Google DNS server)

ping -c 1 8.8.8.8 > /dev/null

if [ $? -eq 0 ]; then

network\_status="Network is up"

else

network\_status="Network is down"

fi

# Log all system health info

echo "$current\_time - CPU Usage: $cpu\_usage, Memory Usage: $memory\_usage, Disk Usage: $disk\_usage, Network Status: $network\_status" >> "$log\_file"

# Optionally, send an email alert if something is critical (CPU > 90%, memory > 90%, disk usage > 90%)

if [ "$(echo $cpu\_usage | sed 's/%//g')" -gt 90 ]; then

echo "Critical: High CPU Usage ($cpu\_usage) detected at $current\_time" | mail -s "System Alert: High CPU Usage" admin@example.com

fi

if [ "$(echo $memory\_usage | sed 's/%//g')" -gt 90 ]; then

echo "Critical: High Memory Usage ($memory\_usage) detected at $current\_time" | mail -s "System Alert: High Memory Usage" admin@example.com

fi

echo "System health check completed at $current\_time."

**Task 112: Automated Backup Script**

This script automates the backup of important directories (e.g., databases, configuration files) to a remote server, keeping only a specified number of backups.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Define directories to back up

source\_dir="/home/user/data"

backup\_dir="/home/user/backups"

remote\_server="user@remote\_server:/path/to/backup"

# Create a timestamp for the backup file

timestamp=$(date +"%Y%m%d%H%M%S")

backup\_filename="backup\_$timestamp.tar.gz"

# Create the backup

tar -czf "$backup\_dir/$backup\_filename" "$source\_dir"

# Sync backup to remote server

scp "$backup\_dir/$backup\_filename" "$remote\_server"

# Delete backups older than 7 days

find "$backup\_dir" -type f -name "backup\_\*.tar.gz" -mtime +7 -exec rm -f {} \;

echo "Backup completed and old backups deleted."

**Task 113: Automated Log Rotation Script**

This script rotates logs by compressing old logs and keeping only a defined number of backups. It can be scheduled to run regularly via cron job.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Define log directory and retention policy

log\_dir="/var/log/myapp"

max\_backups=5

# Get current date for filename

current\_date=$(date +"%Y%m%d")

# Compress the old log file

log\_file="$log\_dir/myapp.log"

gzip -c "$log\_file" > "$log\_dir/myapp\_$current\_date.log.gz"

# Clear the original log file

> "$log\_file"

# Delete old backups if they exceed the retention policy

old\_backups=$(ls -t "$log\_dir/myapp\_\*.log.gz" | tail -n +$(($max\_backups + 1)))

for old\_backup in $old\_backups; do

rm -f "$old\_backup"

done

echo "Log rotation completed. Compressed logs and old backups cleaned up."

**Task 114: Automated Network Monitoring and Alerting**

This script monitors network connectivity, and if it fails (e.g., unable to reach a target server), it sends an alert.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Define target server to ping (e.g., Google DNS)

target="8.8.8.8"

log\_file="/var/log/network\_monitor.log"

# Ping the target

ping -c 1 $target > /dev/null

# Check if ping was successful

if [ $? -ne 0 ]; then

# Log failure and send an alert

current\_time=$(date "+%Y-%m-%d %H:%M:%S")

echo "$current\_time - Network down, unable to reach $target" >> "$log\_file"

echo "Network down alert: Unable to reach $target" | mail -s "Network Alert" admin@example.com

else

# Log network status

current\_time=$(date "+%Y-%m-%d %H:%M:%S")

echo "$current\_time - Network is up" >> "$log\_file"

fi

echo "Network check completed."

**Task 115: Automated Disk Usage Alert Script**

This script checks the disk usage of all mounted file systems and sends an email alert if any file system exceeds a threshold.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Set threshold (in percentage)

threshold=90

email="admin@example.com"

# Check disk usage for all mounted file systems

df -h | grep '^/dev' | while read line; do

usage=$(echo $line | awk '{print $5}' | sed 's/%//')

filesystem=$(echo $line | awk '{print $1}')

# Check if disk usage exceeds threshold

if [ $usage -ge $threshold ]; then

echo "Warning: Disk usage for $filesystem is at $usage%, which exceeds the threshold of $threshold%" | mail -s "Disk Usage Alert for $filesystem" "$email"

fi

done

echo "Disk usage check completed."

**Task 116: Automated Database Backup Script (MySQL)**

This script automates the backup of MySQL databases and sends the backup file to a remote server. It also rotates backups by deleting backups older than 7 days.

**Solution:**

bash

CopyEdit

#!/bin/bash

# MySQL credentials

MYSQL\_USER="root"

MYSQL\_PASS="your\_password"

MYSQL\_HOST="localhost"

# Define backup directory and remote server

backup\_dir="/home/user/db\_backups"

remote\_server="user@remote\_server:/path/to/backup"

timestamp=$(date +"%Y%m%d%H%M%S")

backup\_filename="mysql\_backup\_$timestamp.sql"

# Backup all databases

mysqldump -u "$MYSQL\_USER" -p"$MYSQL\_PASS" --host="$MYSQL\_HOST" --all-databases > "$backup\_dir/$backup\_filename"

# Compress the backup

gzip "$backup\_dir/$backup\_filename"

# Transfer the backup to the remote server

scp "$backup\_dir/$backup\_filename.gz" "$remote\_server"

# Remove backups older than 7 days

find "$backup\_dir" -type f -name "mysql\_backup\_\*.sql.gz" -mtime +7 -exec rm -f {} \;

echo "Database backup completed and old backups removed."

**Task 117: File Integrity Monitoring Script**

This script checks for changes to critical files in a system by comparing their hashes (using md5sum or sha256sum). If any changes are detected, an alert is sent.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Define critical files and hash storage file

file1="/etc/passwd"

file2="/etc/hosts"

hash\_file="/var/log/file\_integrity\_hashes.txt"

current\_time=$(date "+%Y-%m-%d %H:%M:%S")

# Generate new hashes

new\_hash1=$(sha256sum "$file1" | awk '{print $1}')

new\_hash2=$(sha256sum "$file2" | awk '{print $1}')

# Read old hashes from file

if [ -f "$hash\_file" ]; then

old\_hash1=$(grep "$file1" "$hash\_file" | awk '{print $2}')

old\_hash2=$(grep "$file2" "$hash\_file" | awk '{print $2}')

else

old\_hash1=""

old\_hash2=""

fi

# Compare old and new hashes

if [ "$new\_hash1" != "$old\_hash1" ]; then

echo "$current\_time - File $file1 has been modified!" >> "$hash\_file"

echo "File $file1 modified" | mail -s "File Integrity Alert: $file1" admin@example.com

fi

if [ "$new\_hash2" != "$old\_hash2" ]; then

echo "$current\_time - File $file2 has been modified!" >> "$hash\_file"

echo "File $file2 modified" | mail -s "File Integrity Alert: $file2" admin@example.com

fi

# Save new hashes to the file

echo "$file1 $new\_hash1" > "$hash\_file"

echo "$file2 $new\_hash2" >> "$hash\_file"

echo "File integrity check completed."

**Task 118: Automated System Update Script**

This script automatically performs system updates, installs security patches, and reboots the system if necessary.

**Solution:**

bash

CopyEdit

#!/bin/bash

# Update package lists and upgrade installed packages

echo "Starting system update..."

sudo apt-get update -y

sudo apt-get upgrade -y

# Install security updates

sudo apt-get dist-upgrade -y

# Check if a reboot is required

if [ -f /var/run/reboot-required ]; then

echo "System requires a reboot. Rebooting now..."

sudo reboot

else

echo "System is up to date. No reboot required."

fi

echo "System update completed."

These are some real-world automation and monitoring scripts commonly used in industries for system administration, security monitoring, backups, and alerting. You can modify these scripts to suit your specific environment and use them as part of larger automation workflows.