A. Now, after installing Apache, it's time to run Apache under a dedicated user named **apache_admin** to enhance security. Operating Apache under its own non-privileged user account isolates the processes from other system activities, which is a common security practice.

Create a dedicated user named **apache_admin** with no login shell access.

To create a dedicated user named `apache_admin` with no login shell access, follow these steps:

1. Create the user with a disabled login shell using the `useradd` command:

```
sudo useradd -r -s /usr/sbin/nologin apache_admin
```

- `-r`: Creates a system account, usually with a UID lower than 1000, which is ideal for service accounts.

- `-s /usr/sbin/nologin`: Ensures that this user cannot log in interactively, enhancing security.

2. Assign the necessary permissions for `apache_admin` to run Apache. Edit the Apache configuration to run under this new user by modifying the `User` and `Group` directives in the Apache configuration file.

Open the Apache configuration file:

```
sudo nano /etc/apache2/apache2.conf
```

Find the following lines:

```
User ${APACHE_RUN_USER}

Group ${APACHE_RUN_GROUP}
```

Replace them with:

```
User apache_admin

Group apache_admin
```

3. Change the ownership of Apache-related directories to the new user:

```
sudo chown -R apache_admin:apache_admin /var/www

sudo chown -R apache_admin:apache_admin /var/log/apache2
```

4. Finally, restart Apache for the changes to take effect:

```
sudo systemctl restart apache2
```

B. Enable the apache_admin user to restart the Apache web server without needing to enter a password, simplifying server management while maintaining security. Edit the sudoers file to attempt this task. Verify the configuration after updating the sudoers file to ensure it works as intended. Switch to the apache_admin user and attempt to restart the Apache service.

To allow the `apache_admin` user to restart the Apache web server without entering a password, you can configure the `sudoers` file to grant `apache_admin` the required privileges.

Here are the steps to achieve this:

1. **Edit the Sudoers File**

You need to edit the `sudoers` file safely using the `visudo` command, which checks for syntax errors before saving the file.

```
sudo visudo
```

2. **Add Apache Restart Privileges for `apache_admin`**

In the `sudoers` file, add a line to grant the `apache_admin` user permission to restart Apache without a password.

Scroll down to the section where user privileges are defined, and add this line:

```
apache_admin ALL=(ALL) NOPASSWD: /bin/systemctl restart apache2
```

This line means the `apache_admin` user can run the `systemctl restart apache2` command without being prompted for a password.

3. **Save and Exit `visudo`**

To save the changes in `visudo`, press `CTRL + O` (to write/save the file), and then press `CTRL + X` to exit.

4. **Switch to the `apache_admin` User**

Now, switch to the `apache_admin` user to test if the configuration works:

```
su - apache_admin
```

5. **Restart Apache Using `sudo` Without a Password**

Attempt to restart the Apache service using `sudo` as the `apache_admin` user:

```
sudo systemctl restart apache2
```

If everything is configured correctly, Apache should restart without prompting for a password.

6. **Verify Apache Status**

Check the status of Apache to ensure that it has restarted successfully:

```
sudo systemctl status apache2
```

Question 3

TASK 1: Create a New User

1. **Create the User**: To create a new user called audit_member with a home directory, run the following command:
   bash
   ```
   sudo useradd -m audit_member
   ```
   The -m option creates a home directory for the user at /home/audit_member.
2. **Set a Password for the User**: Set a password for the new user:
   bash
   ```
   sudo passwd audit_member
   ```
   You will be prompted to enter and confirm the new password.

TASK 2: Configure Sudo Privileges

1. **Edit the Sudoers File Using visudo**: It's safer to edit the sudoers file using visudo, but since you need to create a separate file for audit_member, you can do the following:
   ```
   sudo visudo -f /etc/sudoers.d/audit_member
   ```
   - You are instructing visudo to edit or create the file located at /etc/sudoers.d/audit_member.
   - This is useful for defining user-specific or command-specific sudo permissions in a separate file rather

than adding everything into the main sudoers file, which enhances organization and reduces the risk of errors.

2. **Specify Allowed Commands**: In the editor, add the following lines to allow audit_member to execute the specified commands without a password:
   ```
   audit_member ALL=(ALL) NOPASSWD: /bin/cat
   /var/log/alternatives.log, /bin/systemctl restart nginx,
   /bin/systemctl status nginx
   ```

   Command List:

- /bin/cat /var/log/alternatives.log: This allows the user to view the contents of the alternatives.log file.
- /bin/systemctl restart nginx: This allows the user to restart the nginx service.
- /bin/systemctl status nginx: This allows the user to check the status of the nginx service.

   This entry allows the audit_member to:
   - View the contents of /var/log/alternatives.log
   - Restart the nginx service
   - Check the status of the nginx service

3. **Save and Exit**: After adding the lines, save the file and exit the editor.

TASK 3: Validate Configuration

1. **Switch to the audit_member User Account**: You can switch to the audit_member user account using:
   ```
   su - audit_member
   ```
2. **Verify Each Specified Command**:
   - **View the contents of the /var/log/alternatives.log file**:
   ```
   sudo cat /var/log/alternatives.log
   ```
   - **Restart the nginx service**:

```
sudo systemctl restart nginx
```

- **Check the status of the nginx service**:

```
sudo systemctl status nginx
```

3. For each command, you should not be prompted for a password, and they should execute successfully.
4. **Attempt to Execute Other Commands with Sudo**: Try running a command that is not allowed, such as:
   bash
   Copy code
   sudo ls /root
   You should receive a message indicating that the user is not allowed to execute this command, confirming that audit_member is restricted to the specified commands only.

Question 4

In a shell script, -e is an option used with the test command (or [ ], which is a synonym for test) to check if a file exists.
Here's what it means:

- **-e**: This checks if a file (or directory) exists. It returns true if the file exists, regardless of the type (regular file, directory, symbolic link, etc.).
  Example:

```
if [ -e "/path/to/file" ]; then
   echo "The file exists."
else
   echo "The file does not exist."
fi
```

In this example, the -e flag is checking if /path/to/file exists.
If you need to check for a specific type of file, there are other flags, such as:

- **-f**: Checks if the file exists and is a regular file.

- **-d**: Checks if the file exists and is a directory.
- -r : Checks if the file exists is readable or not
- -z : The -z flag in shell scripting is used to check if a string is **empty**. Specifically, it returns true if the length of the string is zero (meaning the string is empty).

Question 5

TASK 1:

You need to check the file Existence and Verify if the file check.txt exists and inform the user if it does not. Output an appropriate message indicating whether the file exists.

If the file does exist:

Output message:

The file '/home/user/check.txt' exists.

If the file does not exist:

Output message:

The file '/home/user/check.txt' exists.

TASK 2:

Check File Type. Verify whether check.txt is a regular file and not a directory using appropriate flags. Continue editing the file_info.sh script and output an appropriate message indicating whether the file is a regular file.

If check.txt is a regular file:

Output message:

The file '/home/user/check.txt' is a regular file.

If check.txt is not a regular file:

Output message:

The file '/home/user/check.txt' does not correspond to a regular file.

TASK 3:

Check File Readability. Determine if check.txt is readable and notify the user accordingly. Output an appropriate message indicating whether the file is readable.

If check.txt is readable:

Output message:

The file '/home/user/check.txt' is readable.

If check.txt is not readable:

Output message:

The file '/home/user/check.txt' is not readable.

Note: You need to check the full path /home/user/check.txt of this file check.txt

ANS.

```
#!/bin/bash
```

```
file_path="/home/user/check.txt"

if [ -e $file_path ]; then
        echo "The file '$file_path' exists."
        if [ -f $file_path ]; then
                echo "The file '$file_path' is a regular file."
        else
                echo "The file '$file_path' is not a regular file."
        fi
        if [ -r $file_path ]; then
                echo "The file '$file_path' is readable."
        else
                echo "The file '$file_path' is not readable."
        fi
else
        echo "The file '$file_path' is does not exist."
fi
```

$(...): This syntax tells the shell to execute the command inside the parentheses and substitute it with the output.

Exit Status Codes:

- **exit 0**:
    - Indicates successful completion of the script.

    - By convention, a return code of 0 means "no error," signaling that the script executed successfully without encountering any issues.
- **exit 1**:
    - Indicates that the script encountered an error or abnormal termination.
    - By convention, a return code of 1 (or any non-zero value) signifies that something went wrong during the execution of the script.

Question 6

Write a script called **cleanup_temp.sh** that utilizes wildcards to locate and list all **.tmp files** in a directory **test_temp_dir**. The script should prompt the user for confirmation before deleting these files, and proceed with deletion only if the user approves. This assignment aims to develop a shell script to effectively manage file cleanup operations while enhancing user interaction and command execution skills.

TASK 1:

Use wildcards to find all .tmp files in the directory **test_temp_dir** and list them.

TASK 2:

Continue editing the cleanup_temp.sh script and Prompt the user for confirmation. Before deleting the files, prompt the user to confirm their intention to delete. Use the **echo** command to ask the user for confirmation to delete the listed .tmp files. Use the **read** command to capture the user's response.

TASK 3:

Now, you need to proceed with deletion only if the user confirms the action. Use an **if** statement to check the user's response. If the user confirms, delete the .tmp files. If the user does not confirm, exit the script without deleting the files. But for required task you need to delete the .tmp files.

Ans

```
#!/bin/bash


Directory="/home/user/test_temp_dir"

All_temp_file=$(ls "$Directory"/*.tmp 2>/dev/null)

if [ -z "$All_temp_file" ]; then

	echo "No. We have no temp file in '$Directory'"

	exit 0

fi

echo "Do you want delete temp file in '$Directory'"

read -p permission

if [ "y"=="$permission" ] || [ "Y" == "$permission" ]; then

	sudo rm "$Directory"/*.tmp

else

	exit 1

fi
```

Question 5

Objective

In the script file **calculate_area.sh** already present on path **/home/user** write a script that calculates the area of various geometric shapes based on user input. The script should support multiple interaction modes, including a help menu, interactive input, and command-line arguments.

NOTE: In this assignment you might require usage of bc library. The bc command in Unix/Linux is an arbitrary precision calculator language that supports interactive execution of mathematical expressions. It is often used for performing high-precision arithmetic operations and evaluating complex expressions from the command line.

Script Requirements

TASK - 1:

Help Flag (-h)

When the script is run with the **-h** flag, it should display a help message detailing the types of areas it can calculate. Example areas include **circles**, **squares**, and **rectangles**.

INPUT:

```
./calculate_area.sh -h
```

OUTPUT:

```
Usage: ./calculate_area.sh [option] [shape] [dimension1]
```

[dimension2]

Calculate the area of various geometric shapes.

Options:
-h Display this help message.
-i Interactive mode.

Shapes and dimensions:
circle radius Calculate the area of a circle.
square side Calculate the area of a square.
rectangle length width Calculate the area of a rectangle.

NOTE: Your script help flag should exactly match the above output, including the blank spaces.

TASK - 2:

Command Line Arguments

- If no flags are provided, the script should expect command-line arguments in the following format:

- **./calculate_area.sh shape dimension1 [dimension2]**

- The **shape** should be a **circle**, **square**, or **rectangle**.

- **dimension1** and **dimension2** (if needed) should be the numerical values for the calculations.

INPUT 1:

./calculate_area.sh circle 5

OUTPUT 1:

```
Area of the circle: 78.53975
```

INPUT 2:

```
./calculate_area.sh rectangle 5 10
```

OUTPUT 2:

```
Area of the rectangle: 50
```

TASK - 3:

Interactive Mode (-i)

- Running the script with the -i flag should initiate an interactive mode where the script prompts the user to choose the type of area they wish to calculate.

- Based on the selected type, the script should then ask for the necessary dimensions:

- Circle: Request the radius.

- Square: Request the side length.

- Rectangle: Request the length and width.

INPUT:

```
./calculate_area.sh -i
```

OUTPUT:

```
Choose the shape to calculate the area:
1. Circle
2. Square
3. Rectangle
Enter your choice (1/2/3):
```

Enter choice: 1

```
Enter the radius of the circle:
```

Enter the radius: 5

```
Area of the circle: 78.53975
```

Note: In this and upcoming assignments, you might want to make use of advanced commands to complete the assignment. One such command is getopts, in Unix-based systems getopts is a utility used to parse positional parameters in shell scripts, allowing scripts to accept and handle options and their arguments efficiently. It helps in processing command line arguments passed to a script, ensuring that options are recognized whether they are given as single letters prefixed with a hyphen or as concatenated groups.

Ans

```
#!/bin/bash


# Function to display help message

display_help() {
   echo "Usage: ./calculate_area.sh [option] [shape] [dimension1] [dimension2]"
   echo ""
   echo "Calculate the area of various geometric shapes."
   echo ""
   echo "Options:"
   echo "-h              Display this help message."
   echo "-i              Interactive mode."
   echo ""
   echo "Shapes and dimensions:"
   echo "  circle radius          Calculate the area of a circle."
   echo "  square side            Calculate the area of a square."
   echo "  rectangle lenth width  Calculate the area of a rectangle."
}

calculate_area(){
	shape=$1
	dim1=$2
	dim2=$3
case "$shape" in
	circle)
		if [[ -z $dim1 ]];then
			echo "Please enter radius of circle"
			exit 1
```

```
            fi
            area=$(echo "3.14159 * $dim1 * $dim1" | bc -l)
            echo "Area of the circle: $area"
            ;;
        square)
            if [[ -z $dim1 ]];then
                echo "Please enter lenth of square"
                exit 1
            fi
            area=$(echo "$dim1 * $dim1" | bc -l)
            echo "Area of the square: $area"
            ;;
        rectangle)
            if [[ -z $dim1  || -z $dim2 ]]; then
                echo "Please enter lenth and width of rectangle"
                exit 1
            fi
            area=$(echo "$dim1 * $dim2" | bc -l)
            echo "Area of the rectangle: $area"
            ;;
    esac
    }
    # Check if the -h flag is passed
    if [[ $1 == "-h" ]]; then
```

```
   display_help
   exit 0
fi
if [[ $1 == "-i" ]]; then
     echo "Choose the shape to calculate the area:
1. Circle
2. Square
3. Rectangle"
     read -p "Enter your choice (1/2/3):" option
     if [[ $option == "1" ]];then
           echo "Enter the radius of the circle:"
           read -p radius
           shape="circle"
           calculate_area "$shape" "$radius"
      elif [[ $option == "2" ]];then
           echo "Enter the lenth  of the square:"
           read -p lenth
           shape="square"
           calculate_area "$shape" "$lenth"
     elif [[ $option == "3" ]];then
           echo "Enter the lenht of the rectangular:"
           read -p lenth
           echo "Enter the width of the rectangular:"
           read -p width
```

```
            shape="rectangle"
            calculate_area "$shape" "$lenth" "$width"
        else
            echo "Invalid option"
        fi
fi

# If no flags are provided, treat the arguments as shape and dimensions
if [[ -z $1 ]]; then
    echo "Error: No shape provided. Use -h for help."
    exit 1
else
    shape=$1
    dimension1=$2
    dimension2=$3
    calculate_area "$shape" "$dimension1" "$dimension2"
fi
```

Question 6

Objective

In the script file **log_analyzer.sh** already present on path **/home/user** write a script that analyzes system logs to **identify**, **categorize**, and **summarize** error messages. This script will allow users to interactively choose the type of log analysis they want to perform and display the results based on their selection.

NOTE: The script should have a help message for better understanding.

NOTE: To clear this and upcoming assignment knowledge of debugging levels is crucial. Debugging levels categorize the severity and nature of messages logged during software execution, aiding in problem diagnosis and system monitoring. Common levels include INFO, which records routine information about program operation, DEBUG for detailed diagnostic data, ERROR for significant issues affecting program execution, and CRITICAL for severe conditions that may cause the program to terminate. There are others as well such as WARN showcasing warning messages.

Help Message

```
Usage: ./log_analyzer.sh [-h] [-i] [log_file_path]

Analyze system logs to identify and summarize error messages.

Options:
-h Display this help message.
-i Interactive mode.
log_file_path Specify the path to the log file. Default is
/var/log/syslog.
```

Interactive Mode

The script should start in an interactive mode that prompts the user to choose the type of analysis they wish to perform. Options should include:

- **Count Errors**: Count the log's total number of error messages.

- **List Errors**: List all unique error messages and their frequency.

- **Search Errors**: The user can enter a keyword to search for specific errors in the log.

TASK - 1:

Count the total number of log messages present in **/home/user/logs/log_file.log.**

NOTE: Provide the default log file path inside your script.

INPUT:

```
./log_analyzer.sh -i
```

OUTPUT:

```
Select the type of log analysis to perform:
1. Count Errors
2. List Errors
3. Search Errors
Enter your choice:
```

Enter the choice: 1

```
Total number of errors: 9
```

TASK - 2:

List all unique error messages and their frequency present in **/home/user/logs/log_file.log**.
NOTE: Provide the default log file path inside your script.

INPUT:

```
./log_analyzer.sh -i
```

OUTPUT:

```
Select the type of log analysis to perform:
1. Count Errors
2. List Errors
3. Search Errors
Enter your choice:
```

Enter the choice: 2

```
List of unique error messages and their frequencies:
1 error: Unauthorized access attempt.
1 error: System time synchronization failed.
1 error: Security certificate expired.
1 error: Failed to establish network connection.
1 error: Disk write failure.
1 error: Database connection timeout.
1 error: DNS resolution failure.
1 error: Could not load user profile.
1 error: Application failed to respond.
```

TASK - 3:

The user can enter a keyword to search for specific errors in the log present in **/home/user/logs/log_file.log**.

NOTE: Provide the default log file path inside your script.

INPUT:

```
./log_analyzer.sh -i
```

OUTPUT:

```
Select the type of log analysis to perform:
1. Count Errors
2. List Errors
3. Search Errors
Enter your choice:
```

Enter the choice: 3

```
Enter a keyword to search for specific errors:
```

Enter keyword: error

```
Searching for errors containing the keyword 'error':
2024-07-09T08:11:10.123Z server2 network.error: Failed to establish
network connection.
2024-07-09T08:14:20.001Z server5 application.error: Application
failed to respond.
2024-07-09T08:15:25.234Z server6 database.error: Database
connection timeout.
2024-07-09T08:16:30.567Z server7 system.error: Disk write failure.
2024-07-09T08:18:40.123Z server9 security.error: Unauthorized
access attempt.
2024-07-09T08:20:50.789Z server11 network.error: DNS resolution
failure.
2024-07-09T08:21:55.012Z server12 system.error: System time
```

```
synchronization failed.
2024-07-09T08:23:00.345Z server13 application.error: Could not
load user profile.
2024-07-09T08:25:10.901Z server15 security.error: Security
certificate expired.
```

Ans

```
#!/bin/bash

file_path="/home/user/logs/log_file.log"

Display_help(){

        echo "Select the type of log analysis to perform:"
        echo "1. Count Errors"
        echo "2. List Errors"
        echo "3. Search Errors"
}
analyse(){
        path=$1
        choice=$2
if [[ $choice == "1" ]]; then
        count=$(grep -c "error" "$path")
        echo "Total number of errors: $count"
elif [[ $choice == "2" ]]; then
        list=$(grep "error" "$path")
        echo "List of unique error messages and their frequencies:"
        echo "$list"
else
        echo "List of unique error messages and their frequencies: "
        read -p "Enter keyword: " pattern
        list=$(grep "$pattern" "$path")
        echo "Searching for errors containing the keyword '$pattern':"
        echo "$list"

fi
}
if [[ $1 == "-h" && -z "$2" ]]; then
        Display_help
        echo "Please enter '-h' '-i' 'file path optional' with this format"
elif [[ $1 == "-h" || $2 == "-i" || $1 == "-i" ]]; then
    Display_help
    read -p "Enter your choice: " choice
    if [[ -z "$3" ]]; then
            analyse "$file_path" "$choice"
    else
            analyse "$3" "$choice"
    fi

fi
```

Question 7

Objective

Enhance the previously created **calculate_area.sh** script by integrating structured logging to improve debugging and traceability of the script's operations.

Script Requirements

TASK - 1:

Help Flag (-h)

When the script is run with the **-h** flag, it should display a help message detailing the types of areas it can calculate. Example areas include circles, squares, and rectangles.

INPUT:

```
./calculate_area.sh -h
```

OUTPUT:

```
Usage: ./calculate_area.sh [-h] [-i] [--debug] [--logfile filename]
[shape dimensions]

Calculate the area of various geometric shapes.

Options:
-h Display this help message.
-i Interactive mode.
--debug Enable detailed debug logging.
```

--logfile Specify the file to log to.

Shapes:
circle radius Calculate the area of a circle.
square side Calculate the area of a square.
rectangle length width Calculate the area of a rectangle.

TASK - 2:

Interactive Mode (-i)

- Running the script with the -i flag should initiate an interactive mode where the script prompts the user to choose the type of area they wish to calculate.

- Based on the selected type, the script should then ask for the necessary dimensions:

- Circle: Request the radius.

- Square: Request the side length.

- Rectangle: Request the length and width.

- Record the calculated value as a log entry in **/home/user/logs/calculate_area.log** file.

NOTE: Provide the default log file path inside your script, the value of default path should be /home/user/logs/calculate_area.log.

INPUT:

./calculate_area.sh -i

OUTPUT:

```
Select the type of area to calculate:
1. Circle
2. Square
3. Rectangle
Enter choice:
```

Enter choice: 1

Enter the radius:

Enter the radius: 5

The area of the circle is 78.53975 square units.

After the above steps a line should be added in **/home/user/logs/calculate_area.log** file, in the below given format:

[2024-07-16 22:06:50] [INFO] :: Calculated area of the circle with radius 5: 78.53975

TASK - 3:

Debug (-- debug)

- Implement the **--debug** flag. When this flag is used, include detailed debugging information in the logs.

- Default logging should be at the **INFO** level, providing general information about the script's operation.

INPUT:

```
./calculate_area.sh --debug rectangle 5 10
```

OUTPUT:

```
The area of the rectangle is 50 square units.
```

Log File Entry:

```
[2024-07-16 22:14:39] [INFO] :: Calculated area of the rectangle with length 5 and width 10: 50
```

TASK - 4:

Log Levels and Flags (--logfile)

If the **--logfile** flag is provided, user should be able to redirect log to a file.

INPUT:

```
./calculate_area.sh --logfile /home/user/logs/calculate_area.log square 5
```

OUTPUT:

```
The area of the square is 25 square units.
```

Log File Entry:

[2024-07-16 22:16:36] [INFO] :: Calculated area of the square with side 5: 25

Ans

```
#!/bin/bash

File_path="/home/user/logs/calculate_area.log"
mode=""

display_help() {
    echo "Usage: ./calculate_area.sh [-h] [-i] [--debug] [--logfile filename] [shape dimensions]"
    echo ""
    echo "Calculate the area of various geometric shapes."
    echo ""
    echo "Options:"
    echo "  -h              Display this help message."
    echo "  -i              Interactive mode."
    echo "  --debug         Enable detailed debug logging."
    echo "  --logfile FILE   Specify the file to log to."
    echo ""
    echo "Shapes:"
    echo "  circle radius     Calculate the area of a circle."
    echo "  square side       Calculate the area of a square."
    echo "  rectangle length width  Calculate the area of a rectangle."
    exit 0
}
log(){
        local log_message=$1
        local level=$2
        local timestamp=$(date +"%Y-%m-%d %H:%M:%S")
        if [[ $mode == "d" || $mode == "l" ]]; then
            echo "Log File Entry:"
            echo "[$timestamp] [$level] :: $log_message" | tee -a
"$File_path"
        else
                echo "[$timestamp] [$level] :: $log_message" >> tee -a
"$File_path"
        fi
        }
calculate () {
       local choice=$1
       local dim1=$2
       local dim2=$3
       case $choice in
                 1) if [[ $mode == "i" ]]; then
            read -p "Enter the radius: " radius
          else
           radius=$dim1
          fi

          area=$(echo "3.14159 * $radius * $radius" | bc -l)
          echo "The area of the circle is $area square units."
          log "Calculated area of the circle with radius $radius:
$area" "INFO"
          ;;
        2)
          if [[ $mode == "i" ]]; then
            read -p "Enter the length: " length
          else
           length=$dim1
          fi

          area=$(echo "$length * $length" | bc -l)
          echo "The area of the square is $area square units."
          log "Calculated area of the square with side $length:
$area" "INFO"
          ;;
        3)
          if [[ $mode == "i" ]]; then
```

```
                read -p "Enter the length: " length
                read -p "Enter the width: " width
              else
               length=$dim1
               width=$dim2
              fi

              area=$(echo "$length * $width" | bc -l)
              echo "The area of the rectangular is $area square units."
              log "Calculated area of the rectangular with length
$length and $width: $area" "INFO"
              ;;
            *)
             echo "Invalid Option"
             ;;
        esac
}
display_interactive() {
        echo "Select the type of area to calculate:"
        echo "1. Circle"
        echo "2. Square"
        echo "3. Rectangle"
        read -p "Enter choice: " choice
        calculate "$choice"
        }

    if [[ "$1" != "" ]]; then
        case $1 in
            -h)
                display_help
                mode="h"
                ;;
            -i)
                mode="i"
                display_interactive
                ;;
            --debug)
             mode="d"
             size=$2
             dim1=$3
             dim2=$4
             if [[ $size == "circle" ]]; then
                 calculate "1" "$dim1"
             elif [[ $size == "square" ]]; then
                 calculate "2" "$dim1"
             elif [[ $size == "rectangle" ]]; then
                 calculate "3" "$dim1" "$dim2"
             else
```

```
           echo "invalid area name"
         fi
         ;;
        --logfile)
         mode="l"
         path=$2
         size=$3
         dim1=$4
         dim2=$5
         File_path=$path
         if [[ $size == "circle" ]]; then
            calculate "1" "$dim1"
         elif [[ $size == "square" ]]; then
            calculate "2" "$dim1"
         elif [[ $size == "rectangle" ]]; then
            calculate "3" "$dim1" "$dim2"
         else
           echo "invalid area name"
         fi



esac
```

fi

Question 8

Objective:

In the script file log_analyzer.sh already present on path /home/user write a script that utilizes regular expressions to perform log analysis tasks on system log files. This script should allow users to apply various regex-based operations to filter and categorize data from log files.

Script Requirements

TASK - 1:

Help Flag (-h)

When the script is run with the -h flag, it should display a help message detailing the available operations and their usage.

INPUT:

./log_analyzer.sh -h

OUTPUT:

Usage: ./log_analyzer.sh [-h] [-i] [--file filename] [operation criteria]

Perform regex-based log analysis on system log files.

Options:
-h Display this help message.
-i Interactive mode.

--file filename Specify the log file to operate on.

Operations:
filter level Filter logs by level (INFO, WARN, ERROR, DEBUG).
categorize Categorize logs by level and display counts.

TASK - 2:

Interactive Mode for filter option (-i)

Filter: Use regex to filter log entries based on criteria like error levels (INFO, ERROR, WARN, DEBUG), timestamps, or specific text content.

INPUT:

```
.log_analyzer.sh -i
```

OUTPUT:

```
Enter the log filename:

Enter the complete path to the file from where logs will fetched:
/home/user/log_data/sample_log.log

Choose operation (filter, categorize):

Choose operation:
filter

Enter criteria (for filter: ERROR/INFO/WARN/DEBUG):
```

```
Enter criteria:
ERROR

2024-07-09 08:12:31 ERROR Failed to load configuration.
2024-07-10 09:00:00 ERROR Unable to reach database server.
```

TASK - 3:

Interactive Mode for categorize option (-i)

Categorize: Use regex to categorize log entries by modules or features, displaying counts for each category.

INPUT:

```
.log_analyzer.sh -i
```

OUTPUT:

```
Enter the log filename:

Enter the complete path to the file from where logs will fetched:
/home/user/log_data/sample_log.log

Choose operation (filter, categorize):

Choose operation:
categorize

2 WARN
2 ERROR
1 INFO
```

1 DEBUG

TASK - 4:

Command Line Arguments

The script should accept command-line arguments for automation purposes, allowing operations to be specified without interactive prompts. This should include the file path, operation type, and necessary regex patterns or criteria.

INPUT:

./log_analyzer.sh --file /home/user/log_data/sample_log.txt filter "ERROR"

OUTPUT:

The area of the square is 25 square units.

Ans

```
display_help() {
    echo "Usage: ./log_analyzer.sh [-h] [-i] [--file filename]
[operation criteria]"
    echo ""
    echo "Perform regex-based log analysis on system log files."
    echo ""
    echo "Options:"
    echo "  -h              Display this help message."
    echo "  -i              Interactive mode."
    echo "  --file filename   Specify the log file to operate on."
    echo ""
    echo "Operations:"
    echo "  filter level      Filter logs by level (INFO, WARN, ERROR, DEBUG)."
    echo "  categorize        Categorize logs by level and display counts."
    exit 0
}
filter(){
    path=$1
    keyword=$2
    list=$(grep "$keyword" "$path")
    echo "$list"
}
categorize(){
    path=$1
    echo "Counting log levels in $path:"
    warn_count=$(grep -c "WARN" "$path")
    error_count=$(grep -c "ERROR" "$path")
    info_count=$(grep -c "INFO" "$path")
    debug_count=$(grep -c "DEBUG" "$path")
```

```
  {
      echo "$warn_count WARN"
      echo "$error_count ERROR"
      echo "$info_count INFO"
      echo "$debug_count DEBUG"
   } | sort -rn
}

# Check if help flag is present
if [[ "$1" == "-h" ]]; then
    display_help
elif [[ "$1" == "-i" ]]; then
   echo "Enter the log filename:"
   read -p "Enter the complete path to the file from where logs will
fetched:" File_path
   read -p "Choose operation (filter, categorize):" option
   if [[ $option == "filter" ]]; then
       read -p "Enter criteria:" keyword
       filter "$File_path" "$keyword"
    elif [[ $option == "categorize" ]]; then
        categorize "$File_path" "$keyword"
    else
        echo "Invalid option"
   fi
elif [[ "$1" == "--file" ]]; then
        path=$2
        operation=$3
        keyword=$4
       if [[ $operation == "filter" ]]; then
         filter "$path" "$keyword"
       elif [[ $operation == "categorize" ]]; then
       categorize "$path"
       else
       echo "Invalid option"
       fi
else
   echo "Invalid option"
fi
```

Question 9

Raw Problem

****Raw Problem****

Objective:

In the script file regex_tool.sh already present on path /home/user write a script that performs regex-based operations on text files, including searching, replacing, and extracting data, either interactively or via command-line arguments. It includes a help message, interactive prompts, and ensures the specified file is accessible and readable before

executing operations.

Script Requirements

TASK - 1:

Help Flag (-h)

When the script is run with the -h flag, it should display a help message detailing the available operations and their usage.

INPUT:

```
./regex_tool.sh -h
```

OUTPUT:

```
Usage: ./regex_tool.sh [-h] [-i] [--file filename] [operation regex
[replacement]]

Perform regex-based operations on text files.

Options:
-h Display this help message.
-i Interactive mode.
--file filename Specify the file to operate on.

Operations:
search regex Search for patterns using regex and display all
matching lines.
replace regex new Replace occurrences of regex with new string.
```

TASK - 2:

Interactive Mode for search option (-i)

Search: Use regex to search for patterns in the file and display all matching lines.

INPUT:

```
./regex_tool.sh -i
```

OUTPUT:

```
Enter the filename:

Enter the complete path to the file from where logs will fetched:
/home/user/sample_data/sample_text.txt

Choose operation (search, replace):

Choose operation:
search

Enter regex:

Enter regex:
error 404
```

OUTPUT:

```
Error 404: Page not found
```

TASK - 3:

Interactive Mode for replace option (-i)

Replace: Use regex to replace occurrences of a pattern with a new string.

INPUT:

```
./regex_tool.sh -i
```

OUTPUT:

```
Enter the log filename:

Enter the complete path to the file from where logs will fetched:
/home/user/log_data/sample_log.log

Choose operation (search, replace):

Choose operation:
replace

Enter regex:

Enter regex:
Error 404

Enter replacement text:

Enter replacement text:
CHECK 200
```

Before performing above task:

```
Error 404: Page not found
```

After performing above task:

```
CHECK 200: Page not found
```

TASK - 4:

Command Line Arguments

The script should accept command-line arguments for automation purposes, allowing operations to be specified without interactive prompts. This should include the file path, operation type, and necessary regex patterns or criteria.

INPUT:

```
./regex_tool.sh --file /home/user/sample_data/sample_text.txt search "error"
```

OUTPUT:

```
Error 404: Page not found
Error 500: Internal Server Error
```

Ans

```
#!/bin/bash


display_help() {
    echo "Usage: $0 [-h] [-i] [--file filename] [operation regex [replacement]]"
    echo
    echo "Perform regex-based operations on text files."
    echo
    echo "Options:"
    echo "  -h              Display this help message."
    echo "  -i              Interactive mode."
    echo "  --file filename  Specify the file to operate on."
    echo
    echo "Operations:"
    echo "  search regex          Search for patterns using regex and
display all matching lines."
    echo "  replace regex new     Replace occurrences of regex with
new string."
}

interactive_mode() {
    echo "Enter the filename:"
    read -r filename

    if [[ ! -f "$filename" ]]; then
        echo "Error: File not found."
        exit 1
    fi

    if [[ ! -r "$filename" ]]; then
```

```
        echo "Error: File is not readable."
        exit 1
    fi

    echo "Choose operation (search or replace):"
    read -r operation
    echo "Enter regex:"
    read -r regex

    case $operation in
        search)
            grep -iP "$regex" "$filename"
            ;;
        replace)
            echo "Enter replacement text:"
            read -r replacement
            sed -i -r "s/$regex/$replacement/g" "$filename"
            ;;
        *)
            echo "Invalid operation."
            exit 1
            ;;
    esac
}
```

```
while [[ "$1" =~ ^- && ! "$1" == "--" ]]; do
    case $1 in
        -h | --help)
            display_help
            exit
            ;;
        -i | --interactive)
            INTERACTIVE_MODE=1
            ;;
        --file)
            shift
            FILE="$1"
            ;;
    esac
    shift
done
if [[ "$1" == '--' ]]; then shift; fi

if [[ "$INTERACTIVE_MODE" == "1" ]]; then
    interactive_mode
else
    OPERATION="$1"
    REGEX="$2"
```

```
REPLACEMENT="$3"

if [ -z "$FILE" ]; then
    echo "File not specified. Use --file to specify the file."
    display_help
    exit 1
fi

if [[ ! -f "$FILE" ]]; then
    echo "Error: File not found."
    exit 1
fi

if [[ ! -r "$FILE" ]]; then
    echo "Error: File is not readable."
    exit 1
fi

if [[ -z "$OPERATION" || -z "$REGEX" ]]; then
    echo "Invalid operation or insufficient arguments."
    display_help
    exit 1
fi
```

```
    case $OPERATION in
        search)
            grep -iP "$REGEX" "$FILE"
            ;;
        replace)
            if [ -z "$REPLACEMENT" ]; then
                echo "Replacement text required for replace operation."
                exit 1
            fi
            sed -i -r "s/$REGEX/$REPLACEMENT/g" "$FILE"
            ;;
        *)
            echo "Invalid operation or insufficient arguments."
            display_help
            exit 1
            ;;
    esac
fi
```

****************Question***

***The goal is to write a script in /home/user/server_health_check.sh that monitors system processes and logs memory usage. The script will accomplish the following tasks:**

TASK: Logging Process ID and Memory Usage

The script should continuously monitor every 2 sec the system processes and log the Process ID (PID) and memory usage for each process.
The information should be logged in the format PID_ID Memory in a file located at /home/user/health_reports/server_health.log.
This log file will help track the memory usage of various processes over time.

Expected Output:
A file named server_health.log located in /home/user/health_reports/, which contains entries in the format PID_ID Memory for each monitored process.

Sample Output:

```
1234 45MB
5678 30MB
9101 100MB
```

Ans

```
#!/bin/bash

while true; do

   top -b -n 1 | awk 'NR>7 {printf "%s %sMB\n", $1, $10*100}' >>
/home/user/health_reports/server_health.log

   sleep 2

done
```

**************Question*********************

Optimize the previous script to alert whenever certain memory limit exceeds.

Task: Alerting on Memory Usage Exceeding 20MB

The script should monitor the memory usage of each process and log any process that consumes more than 20MB of memory.
If a process exceeds 20MB of memory usage, the script should log the Process ID (PID) and memory usage in a file named /home/user/health_reports/alert.log.

Only processes that exceed the 20MB threshold should be logged in the alert file. Processes below this threshold should not appear in the alert log.

Expected Output:
A file named alert.log located in /home/user/health_reports/, which logs entries in the format PID_ID Memory for processes that exceed 20MB of memory usage.

Sample Output:
```
534 23MB
1129 58MB
```

Ans

```
#!/bin/bash
total=$(awk '/MemTotal/ {print $2}' /proc/meminfo) # Total memory in kB
LOG_DIR="/home/user/health_reports"
ALERT_LOG="$LOG_DIR/alert.log"
SERVER_LOG="$LOG_DIR/server_health.log"

# Ensure the directory and log files exist
mkdir -p "$LOG_DIR"
touch "$ALERT_LOG" "$SERVER_LOG"
```

```
# Monitor memory usage
while true; do
    # Run top and process memory usage
    top -b -n 1 | awk -v Total_Mem="$total" -v
ALERT_LOG="$ALERT_LOG" -v SERVER_LOG="$SERVER_LOG" '
    NR > 7 {
        mem_used_kb = ($10 * Total_Mem) / 100;
        if (mem_used_kb > 20 * 1024) {
            # Overwrite alert log for processes > 20MB
            printf "%s %dMB\n", $1, mem_used_kb / 1024 >
ALERT_LOG;
        } else {
            # Append server log for processes <= 20MB
            printf "%s %dMB\n", $1, mem_used_kb / 1024 >>
SERVER_LOG;
        }
    }'

    # Pause for 2 seconds before repeating
    sleep 2
done
```

*****************Question************************

Objective:

Extending the previous question, regular logs are being stored in the logfile.log file. Your task is to write a script in website_health_check.sh file that fulfills the below tasks:

TASK - 1:

Redirect only the errors from this log file to the error_reports/website_health.log file. Remember, error storing should be continuous and dynamic.

Tip: Create the report before running the test cases.

Output:

```
2024-07-22 23:11:15 2024-07-22 23:10:55 ERROR: This is error number 1.
2024-07-22 23:11:15 2024-07-22 23:11:00 ERROR: This is error number 2.
2024-07-22 23:11:15 2024-07-22 23:11:05 ERROR: This is error number 3.
```

TASK - 2:

Based on the above errors received, prepare an error report and store the necessary details in the error_reports/website_report.log file.

Output Format:

```
Error Report - Mon Jan 2 20:10:20 UTC 2023
Total Errors: 9
Latest Error: 2023-01-02 20:10:16 2023-01-02 20:10:16 ERROR: This is error number 21.
```

```
#!/bin/bash
```

```
# File and directory paths
LOG_FILE="/home/user/logfile.log"
ERROR_REPORT_DIR="/home/user/error_reports"
ERROR_LOG="$ERROR_REPORT_DIR/website_health.log"
REPORT_LOG="$ERROR_REPORT_DIR/website_report.log"

# Create error reports directory if it doesn't exist
mkdir -p "$ERROR_REPORT_DIR"

# Ensure ERROR_LOG and REPORT_LOG exist
if [ ! -f "$ERROR_LOG" ]; then
  touch "$ERROR_LOG"
fi

if [ ! -f "$REPORT_LOG" ]; then
  touch "$REPORT_LOG"
fi

# Check if LOG_FILE exists
if [ ! -f "$LOG_FILE" ]; then
  echo "Log file not found: $LOG_FILE"
  exit 1
fi
```

```
# Function to update the error report dynamically
update_report() {
   local count=$(grep -oi "error" "$ERROR_LOG" | wc -l)
   local last_line=$(tail -n 1 "$ERROR_LOG")

   echo "Error Report - $(date)" > "$REPORT_LOG"
   echo "Total Errors: $count" >> "$REPORT_LOG"
   echo "Latest Error: $last_line" >> "$REPORT_LOG"
}

# Start capturing errors from LOG_FILE
trap 'kill $(jobs -p); exit' SIGINT SIGTERM

tail -f "$LOG_FILE" | grep --line-buffered -i "error" | tee -a
"$ERROR_LOG" &

# Dynamically update the report as new errors are logged
while true; do
   update_report
   sleep 5 # Adjust interval as needed
done
```

the --line-buffered option in the grep command ensures that the script processes and handles data dynamically and continuously.

How --line-buffered Works:

1. **Buffered Output by Default:**
   - **By default, grep and many other command-line utilities buffer their output, especially when writing to a pipe. This means they may wait until a certain amount of data is collected before processing or outputting it.**
   - **Without --line-buffered, grep might delay processing lines from tail -f until the buffer is full, causing significant delays in real-time scenarios.**
2. **Immediate Line Processing:**
   - **The --line-buffered option forces grep to process each line as soon as it is received.**
   - **In this script, as tail -f streams lines from the log file (LOG_FILE), grep immediately filters lines containing "error" and sends them to the next command (tee).**

Why It Ensures Dynamic and Continuous Logging:

- **Errors are detected and logged to ERROR_LOG in real time, without delays caused by buffering.**
- **This behavior is critical for scenarios requiring continuous monitoring, such as real-time error detection and dynamic report generation in the script.**

Without --line-buffered:

- **You might experience noticeable delays in detecting and logging errors, especially if the log file (LOG_FILE) is not being updated frequently.**

In summary, --line-buffered is essential in this script to ensure dynamic, real-time error processing and continuous logging functionality.

*****************QUESTION****************************

Raw Problem

There is script present in your system server_setup.sh, you need to edit this script in order to do these below tasks.
Task 1: Software Installation and System Update
Objective: Update the server with necessary software and update all existing packages.

- Update and Upgrade: Start by refreshing the package lists and upgrading all installed packages to enhance both security and functionality.
- Software Installation: Install essential software such as Nginx for web serving, Apache2-utils for Apache server management, UFW for firewall configurations, Fail2ban for security against brute-force attacks, and Vim as a versatile text editor.

Task 2: User Setup
Objective: Establish user accounts with predefined configurations to manage access control efficiently.

- User Creation: Create specific user accounts like `johndoe` and `janedoe`, setting them up with initial passwords and necessary user details, ensuring they have the correct permissions for their roles.

Task 3: Security Hardening and Logging

Objective: Enhance the security measures to safeguard the server. There is firewall Configured in your background. You need to do these tasks to ensure security measures:

- **Disable Unused Services: Disable services that are not in use, such as Apache2 if it's redundant, to reduce potential security vulnerabilities.**
- **Secure SSH Access: Adjust SSH configurations to improve security measures, including disabling root login to prevent direct remote access to the server's root account.**

**********************ANS***************

```
#!/bin/bash
user_exist(){
   id -u "$1" &>/dev/null
}

sudo apt update
sudo apt upgrade -y

sudo apt install nginx -y
sudo apt install apache2 -y
sudo apt install ufw -y
sudo apt install fail2ban -y
sudo apt install vim -y
sudo apt install openssh-server -y
```

```
# Enable and configure UFW

sudo ufw allow 'Nginx Full'

sudo ufw enable


# Restart Fail2Ban to apply configurations

sudo systemctl restart fail2ban


if ! getent group DEVELOPER &>/dev/null; then

   sudo groupadd DEVELOPER

fi


if ! user_exist "johndoe"; then

   sudo useradd -m -p $(openssl passwd -6 "johndoe") johndoe

   sudo usermod -aG DEVELOPER johndoe

fi


if ! user_exist "janedoe"; then

   sudo useradd -m -p $(openssl passwd -6 "janedoe") janedoe

   sudo usermod -aG DEVELOPER janedoe

fi


sudo systemctl disable --now apache2 || error_exit "Failed to
disable Apache2."
```

```
secure_ssh() {
  sshd_config_file="/etc/ssh/sshd_config"

  # Backup the existing sshd_config
  sudo cp "$sshd_config_file" "${sshd_config_file}.bak"

  # Disable root login
if grep -q "^PermitRootLogin" $sshd_config_file; then
    sudo sed -i 's/^PermitRootLogin.*/PermitRootLogin no/'
"$sshd_config_file"
elif grep -q "^#PermitRootLogin" $sshd_config_file; then
    sudo sed -i 's/^#PermitRootLogin.*/PermitRootLogin no/'
"$sshd_config_file"
else
    echo "PermitRootLogin no" | sudo tee -a $SSHD_CONFIG
fi



  # Restart SSH service

  sudo systemctl restart ssh
```

}

secure_ssh

****
Question*******************

Task 1: Create HTML Report

Write a script that makes a system health report in HTML format. The script should check if the file system_health_report.html is saved correctly in the /var/log/ folder.

Task 2: Check HTML Report Content

The script should check that the HTML report has all the important sections. It should include:

- "System Health Report"
- "Uptime"
- "CPU and Memory Usage"
- "Disk Usage"
- "Recently Installed Security Patches"

Each section should provide clear information about the system's health and make sure to include these keywords and check these details as well.

Task 3: Verify Email Delivery

Set up the script to automatically send the report to the janedoe. The script should check if the email with the HTML report has been sent and received in the right mailbox, making sure it includes all the important details.

Expected Results

- Task 1: The task is successful if the HTML file is found in the right folder, showing that the report was created and saved correctly.
- Task 2: The task is complete if the HTML report has all the necessary sections, giving a full picture of the system's health.
- Task 3: The task is successful if the report is in the recipient's mailbox with the correct information, confirming it was sent and received properly.

*****************************Ans*****************

```
#!/bin/bash

recipient="janedoe@locakhost"

subject="System Health Report"

file_path="/var/log/"

file_name="system_health_report.html"

full_path="${file_path}${file_name}"

mail_file="/var/mail/janedoe"

if [[ -d $mail_file ]]; then

  sudo mkdir -p "$mail_file"

fi

sudo apt install mailutils  -y

if [[ -d $file_path ]]; then

      touch "$full_path"

fi
```

```
uptime=$(uptime)
memryuse=$(free -m | awk 'NR=2 {print "Total: %s MB Used: %s
MB Free: %s MB Avaiable: % MB", $2,$3,$4,$7}')
cpu=$(top -bn1 | grep "Cpu(s)" | sed "s/,/ /g")
disk=$(df -h)

{
 echo "<!DOCTYPE html>"
 echo "<html lang=\"en\">"
 echo "<head>"
 echo "<meta charset=\"UTF-8\">"
 echo "<title>System Health Report</title>"
 echo "</head>"
 echo "<body>"
 echo "<h1>System Health Report</h1>"
 echo "<h2>Uptime</h2>"
 echo "<pre> $uptime </pre>"
 echo "<h2>CPU and Memory Usage</h2>"
 echo "<h3> Memory Usage </h3>"
 echo "<pre> $memryuse </pre>"
 echo "<h3> CPU Usage </h3>"
 echo "<pre> $cpu </pre>"
 echo "<h2> Disk Usage</h2>"
 echo "<pre> $disk </pre>"
```

```
  echo "<h2>Recently Installed Security Patches</h2>"
  echo "<pre>$(grep "install " /var/log/dpkg.log | tail -n 10)</pre>"
  echo "</body>"
  echo "</html>"
} > $full_path
cp $full_path $mail_file

if command -v mailx &>/dev/null; then
  mailx -a "Content–type:text/html;" -s "$subject" "$recipient" <
$full_path
  if [[ $? -eq 0 ]]; then
    echo "System health report sent to $recipient."
  else
    echo "Failed to send the system health report." >&2
  fi
else
  echo "mailx is not installed. Please install it to enable email
functionality." >&2
  exit 1
fi
```

**********************question*******************

Task: Create a script system_maintenance.sh that updates system packages, removes unnecessary files, and cleans up the system.

Objective: Automate regular updates and maintenance tasks on Linux servers to maintain their security and performance.

Requirements:

- **Package Updates: Update all installed packages using the system's package manager.**
- **Cleanup Tasks: Clean up old cached packages and unused dependencies. Remove old kernel versions, keeping only the current and one backup kernel.**
- **Logging: Log all performed actions and any issues encountered during the process.**

Learning Goals: This assignment focuses on scripting for system maintenance and update procedures.

Deliverable: Submit the system_maintenance.sh script that fulfills the above requirements.

********************ans**********************

```
#!/bin/bash


log_file="/var/log/system_maintenance.log"


if [[ ! -f "$log_file" ]]; then
	touch $log_file
fi


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


log(){
        echo "[$Date] $1" | tee -a "$log_file"
}

log "Updating package"
if sudo apt update && sudo apt upgrade -y; then
       log "update pakage succesfully"
else
     log "update not done"
fi
log "removing not required things"
if sudo apt autoremove -y && sudo apt autoclean; then
        log "Clean up old cached package and remove old kernel
succesfully"
else
        log "Old file and cached not remove"
fi
log "System maintain succsesfully"
****************************************
```