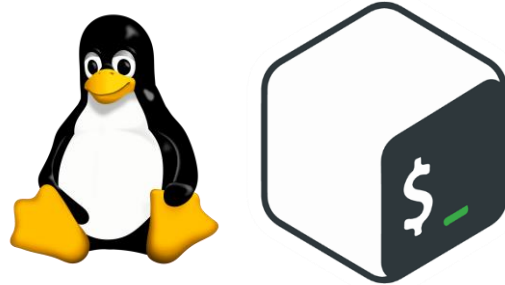


Atypoon

Training Program

Assignment #3

Linux Command & Script Project Report



Instructor: Prof. Motasem Al-Diab

Done by: Ahmad Al-Sou'b



Table of Contents:

❖	<u>Abstraction</u>	<u>3</u>
1.	<u>Introduction</u>	<u>3</u>
2.	<u>Project Overview</u>	<u>4</u>
3.	<u>Work Explanation</u>	<u>4</u>
4.	<u>Implementation.....</u>	<u>6</u>
	<u>4.1 Backup</u>	<u>6</u>
	<u>4.2 System Health Check</u>	<u>13</u>
5.	<u>Results</u>	<u>19</u>
6.	<u>Challenges</u>	<u>19</u>
7.	<u>Conclusions</u>	<u>20</u>



Abstraction:

The assignment focuses on automating critical system administration tasks through two distinct shell scripts: the **Backup Script** and the **System Health Check Script**. These scripts aim to streamline essential processes, improve efficiency, and reduce the manual effort required to maintain data integrity and system health.

Both scripts are designed with a focus on user-friendliness, efficiency, and reliability, making them valuable tools for system administrators and regular users alike.

- **The Backup Script** automates the backup of user-specified directories, compressing them into space-efficient .tar.gz files and logging the entire process to ensure transparency. It addresses common challenges such as error handling for invalid directories and ensures a user-friendly experience.
- **The System Health Check Script** provides a detailed system health report, covering disk space, memory utilization, service status, and system updates. It helps system administrators maintain optimal performance and stability by offering actionable insights.

Introduction:

In today's digital landscape, maintaining data integrity and system performance is critical. The assignment's goal was to develop two scripts that automate common yet crucial administrative tasks.

Together, these scripts reflect a comprehensive approach to automating routine tasks, enabling administrators to focus on higher-level responsibilities while ensuring system reliability and data security.

- **The Backup Script** addresses the need for reliable data backups to prevent data loss, which can have severe consequences. By automating the process of directory backups and compression, the script ensures that users can safeguard their important data with minimal effort.
- **The System Health Check Script** caters to the challenges of monitoring system health in real-time. It automates the assessment of key system metrics, such as storage usage and service status, providing administrators with a clear understanding of the system's current state.



Project Overview:

The project is centered on automating system administration tasks, with each script targeting a specific aspect of system maintenance:

1. Backup Script:

- **Objective:** Automate the backup of user-specified directories into compressed .tar.gz files.
- **Key Features:**
 - User-specified input for directories.
 - Compression to save storage space and facilitate file transfers.
 - Logging to provide a detailed record of the backup process.
 - Error handling for invalid directories or failed backups.

2. System Health Check Script:

- **Objective:** Provide a detailed system health report, consolidating key metrics for administrators.
- **Key Features:**
 - Disk space monitoring to prevent crashes due to full storage.
 - Memory utilization tracking to ensure optimal performance.
 - Service status checking for critical services like apache2 and cron.
 - Update notifications for maintaining security and functionality.

Both scripts share a focus on automation, error handling, and user feedback, ensuring they are robust, reliable, and easy to use.

Work Explanation:

Developing the **Backup Script** and **System Health Check Script** involved a structured approach, from understanding the requirements to implementing and testing the solutions. Here's how the work was carried out:

1. Requirement Analysis:

The first step was to clearly define the objectives and features for each script:

- For the **Backup Script**, the goal was to automate directory backups with compression and logging, ensuring user-specified flexibility and error handling.
- For the **System Health Check Script**, the goal was to monitor system metrics like disk usage, memory utilization, service statuses, and updates, providing clear feedback for administrators.



2. Script Design:

Based on the requirements, the scripts were designed with the following key considerations:

- **Backup Script:**
 - Input flexibility for directories to be backed up.
 - Usage of the **tar** command for creating (.tar.gz)archives.
 - Logging implementation with **tee** command to ensure transparency.
 - Error checks for invalid directories or failures.
- **System Health Check Script:**
 - Inclusion of checks for disk space, memory usage, and service statuses.
 - Compatibility across Linux distributions.
 - Informative yet concise update notifications.
- Modular design was applied to ensure each functionality (e.g., compression, logging, service checks) was self-contained and reusable.

3. Implementation:

- Shell scripting was used due to its efficiency and compatibility with Linux environments.
- The scripts were written to be user-friendly, using clear prompts and outputs.
- Logging and error handling were implemented to ensure transparency and ease of debugging.

4. Testing and Debugging:

- Extensive testing was conducted in various scenarios:
 - Valid and invalid directory inputs for the Backup Script.
 - Monitoring disk space, memory, and services under different load conditions for the System Health Check Script.
 - Compatibility checks across multiple Linux distributions (e.g., Ubuntu, CentOS).
- Errors were logged for analysis, and fixes were applied to ensure robustness.

5. Optimization:

- For the Backup Script, compression levels and logging formats were optimized to balance performance and usability.
- For the System Health Check Script, commands were refined to produce concise yet informative outputs, avoiding unnecessary clutter.



Implementation:

Backup:

Scenario:

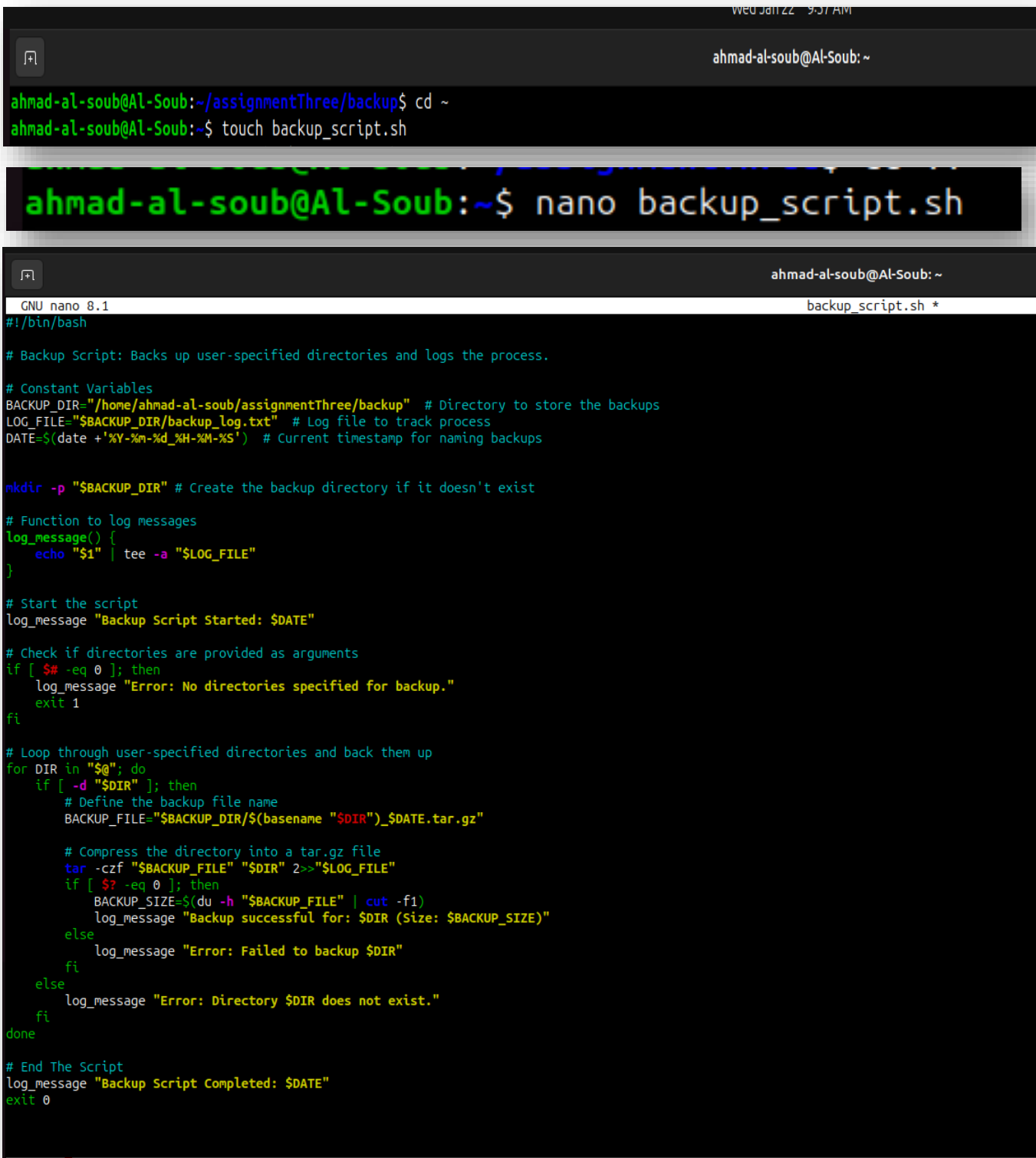
- I have Created Three backup directories:
 - /home/user/assignmentThree/docs.
 - /home/user/assignmentThree/photos.
 - /home/user/assignmentThree/songs.
- The backup will be stored in the /assignmentThree/backup.
- The Script will also log the process to log file: /backup/backup_log.txt
- And create files inside them: as in Figure 1.

```
ahmad-al-soub@Al-Soub:~$ mkdir assignmentThree
ahmad-al-soub@Al-Soub:~$ cd assignmentThree/
ahmad-al-soub@Al-Soub:~/assignmentThree$ mkdir docs photos songs
ahmad-al-soub@Al-Soub:~/assignmentThree$ ls
docs photos songs
ahmad-al-soub@Al-Soub:~/assignmentThree$ cd docs/
ahmad-al-soub@Al-Soub:~/assignmentThree/docs$ echo "This is The First File" >firstFile.txt
ahmad-al-soub@Al-Soub:~/assignmentThree/docs$ echo "This is The Second File" >secondFile.txt
ahmad-al-soub@Al-Soub:~/assignmentThree/docs$ cd ..
ahmad-al-soub@Al-Soub:~/assignmentThree$ cd photos/
ahmad-al-soub@Al-Soub:~/assignmentThree/photos$ echo "This is The Photo 1" >firstPhoto.jpg
ahmad-al-soub@Al-Soub:~/assignmentThree/photos$ echo "This is The Photo 2" >secondPhoto.jpg
ahmad-al-soub@Al-Soub:~/assignmentThree/photos$ cd ..
ahmad-al-soub@Al-Soub:~/assignmentThree$ cd songs/
ahmad-al-soub@Al-Soub:~/assignmentThree/songs$ echo "Coolio - Gangsta's Paradies" > firstSong.mp3
ahmad-al-soub@Al-Soub:~/assignmentThree/songs$ echo "Eminem - Not Afraid" > secondSong.mp3
ahmad-al-soub@Al-Soub:~/assignmentThree/songs$ cd ..
ahmad-al-soub@Al-Soub:~/assignmentThree$ cd docs/
ahmad-al-soub@Al-Soub:~/assignmentThree/docs$ ls
firstFile.txt secondFile.txt
ahmad-al-soub@Al-Soub:~/assignmentThree/docs$ cd ..
ahmad-al-soub@Al-Soub:~/assignmentThree$ cd photos/
ahmad-al-soub@Al-Soub:~/assignmentThree/photos$ ls
firstPhoto.jpg secondPhoto.jpg
ahmad-al-soub@Al-Soub:~/assignmentThree/photos$ cd ..
ahmad-al-soub@Al-Soub:~/assignmentThree$ cd songs/
ahmad-al-soub@Al-Soub:~/assignmentThree/songs$ ls
firstSong.mp3 secondSong.mp3
ahmad-al-soub@Al-Soub:~/assignmentThree/songs$ cd ..
ahmad-al-soub@Al-Soub:~/assignmentThree$ mkdir backup
ahmad-al-soub@Al-Soub:~/assignmentThree$ ls
backup docs photos songs
ahmad-al-soub@Al-Soub:~/assignmentThree$ cd backup/
ahmad-al-soub@Al-Soub:~/assignmentThree/backup$ touch backup_log.txt
ahmad-al-soub@Al-Soub:~/assignmentThree/backup$ ls
backup_log.txt
ahmad-al-soub@Al-Soub:~/assignmentThree/backup$
```

Figure 1



- **Create The Backup Script:**
 - Create a backup_script.sh file and start writing commands as in figure 2.



```

ahmad-al-soub@Al-Soub: ~/assignmentThree/backup$ cd ~
ahmad-al-soub@Al-Soub: ~$ touch backup_script.sh

ahmad-al-soub@Al-Soub: ~$ nano backup_script.sh

GNU nano 8.1 backup_script.sh *
#!/bin/bash

# Backup Script: Backs up user-specified directories and logs the process.

# Constant Variables
BACKUP_DIR="/home/ahmad-al-soub/assignmentThree/backup" # Directory to store the backups
LOG_FILE="$BACKUP_DIR/backup_log.txt" # Log file to track process
DATE=$(date +%Y-%m-%d_%H-%M-%S) # Current timestamp for naming backups

mkdir -p "$BACKUP_DIR" # Create the backup directory if it doesn't exist

# Function to log messages
log_message() {
    echo "$1" | tee -a "$LOG_FILE"
}

# Start the script
log_message "Backup Script Started: $DATE"

# Check if directories are provided as arguments
if [ $# -eq 0 ]; then
    log_message "Error: No directories specified for backup."
    exit 1
fi

# Loop through user-specified directories and back them up
for DIR in "$@"; do
    if [ -d "$DIR" ]; then
        # Define the backup file name
        BACKUP_FILE="$BACKUP_DIR/$(basename "$DIR")_$DATE.tar.gz"

        # Compress the directory into a tar.gz file
        tar -czf "$BACKUP_FILE" "$DIR" 2>>"$LOG_FILE"
        if [ $? -eq 0 ]; then
            BACKUP_SIZE=$(du -h "$BACKUP_FILE" | cut -f1)
            log_message "Backup successful for: $DIR (Size: $BACKUP_SIZE)"
        else
            log_message "Error: Failed to backup $DIR"
        fi
    else
        log_message "Error: Directory $DIR does not exist."
    fi
done

# End The Script
log_message "Backup Script Completed: $DATE"
exit 0

```

Figure 2



- **Breakdown of the Script:**

1. **In this line**, I write the location of the shell where I want Linux to execute the commands, as in Figure 3.



Figure 3

2. **In Figure 4,**

- I created three constant variables to store the backup files, store the log file for the process, and finally create a current timestamp to label the backup.
- The command "**mkdir -p**" used to create a directory and ensures Any Parent directories in the path are created if they do not already exist. and It does not throw an error if the directory already exists.
- The log_message function Definition:
 - I. **(echo "\$1")**: This command prints the first argument passed to the function.
 - II. **(|)**: Sends the output of {echo \$1} to next command (tee -a "\$LOG_FILE")
 - III. **(tee -a "\$LOG_FILE")**: This command reads input and writes it to the standard output and appends the message to the file specified by (\$LOG_FILE) instead of overwriting it. And used this function to start the script.

```
# Backup Script: Backs up user-specified directories and logs the process.

# Constant Variables
BACKUP_DIR="/home/ahmad-al-soub/assignmentThree/backup" # Directory to store the backups
LOG_FILE="$BACKUP_DIR/backup_log.txt" # Log file to track process
DATE=$(date +%Y-%m-%d_%H-%M-%S) # Current timestamp for naming backups

mkdir -p "$BACKUP_DIR" # Create the backup directory if it doesn't exist

# Function to log messages
log_message() {
    echo "$1" | tee -a "$LOG_FILE"
}

# Start the script
log_message "Backup Script Started: $DATE"
```

Figure 4



3. In Figure 5,

- **Check if the directories are provided as arguments.**
- **(for DIR in "\$@"; do):** Represents all the arguments passed to the script, with each argument treated as a separate item. And the **for** loop assigns each argument (one by one) to the variable **DIR**.
- **(if [-d "\$DIR"]; then):** Checks if the value of DIR is an existing directory. And If DIR is a directory, the commands inside the if block are executed.
- **(BACKUP_FILE="\$BACKUP_DIR/\${basename"\$DIR")}_\$DATE.tar.gz"):**
A variable that holds the directory where backup files will be saved. And extract the name of the directory (without its path). Then the variable that holds the current date or timestamp (e.g., 2025-01-22).

```
# Check if directories are provided as arguments
if [ $# -eq 0 ]; then
    log_message "Error: No directories specified for backup."
    exit 1
fi

# Loop through user-specified directories and back them up
for DIR in "$@"; do
    if [ -d "$DIR" ]; then
        # Define the backup file name
        BACKUP_FILE="$BACKUP_DIR/${basename "$DIR"}_$DATE.tar.gz"
```

Figure 5

4. In Figure 6,

- **(tar -czf "\$BACKUP_FILE" "\$DIR" 2>>"\$LOG_FILE"):** The **tar** command is used to create an archive. And **-czf** options is used to create a new archive, compress the archive using gzip (.tar.gz format), and Specifies the name of the archive file.
 - ✓ **(\$BACKUP_FILE):** This is the backup file where the compressed archive will be saved.
 - ✓ **(\$DIR):** The directory that is being backed up.
 - ✓ **(2>>"\$LOG_FILE"):** Redirects any error messages to a log file. To ensures that if any error occurs during the tar command, it is logged into \$LOG_FILE.



- (if [\$? -eq 0]; then):
 - ✓ (\$? -eq 0): This variable holds the exit status of the last executed command. And checks if the exit status is equal to 0 (success).
 - ✓ If the **tar** command was successful, the script proceeds to the **then** block.
- (BACKUP_SIZE=\$(du -h "\$BACKUP_FILE" | cut -f1)):
 - ✓ The **du** command reports the disk usage of the backup file. And convert to the Human-readable format. And extracts the first field (the file size) from the **du** output. And captures the output (the file size) and stores it in the **BACKUP_SIZE** variable.
- Calls the **log_message** function to log a success message, including the directory name (**\$DIR**) and the backup size (**\$BACKUP_SIZE**).
- If the **exit** status of **tar** is not 0, meaning the backup failed, the script executes the **else** block.
- Call the **log_message** function to log an error message indicating the failure of the backup for the specified directory (**\$DIR**).
- End The Script.

```
# Compress the directory into a tar.gz file
tar -czf "$BACKUP_FILE" "$DIR" 2>>"$LOG_FILE"
if [ $? -eq 0 ]; then
    BACKUP_SIZE=$(du -h "$BACKUP_FILE" | cut -f1)
    log_message "Backup successful for: $DIR (Size: $BACKUP_SIZE)"
else
    log_message "Error: Failed to backup $DIR"
fi
else
    log_message "Error: Directory $DIR does not exist."
fi
done

# End The Script
log_message "Backup Script Completed: $DATE"
exit 0
```

Figure 6



5. After saving the script to a file, Make the script executable, and run the script with the directories as arguments, as in figure 7.

```
ahmad-al-soub@Al-Soub:~$ ls  
assignmentThree backup_script.sh Desktop Documents Downloads etc Music Pictures Public snap Templates Videos  
ahmad-al-soub@Al-Soub:~$ chmod +x backup_script.sh  
ahmad-al-soub@Al-Soub:~$ ls  
assignmentThree backup_script.sh Desktop Documents Downloads etc Music Pictures Public snap Templates Videos  
ahmad-al-soub@Al-Soub:~$ ./backup_script.sh /home/ahmad-al-soub/assignmentThree/docs /home/ahmad-al-soub/assignmentThree/p
```

Figure 7

6. Expected Output:

- The script will create backup files for the docs, photos, and songs directories as in figure 8.

```
ahmad-al-soub@Al-Soub:~$ ./backup_script.sh /home/ahmad-al-soub/assignmentThree/docs /home/ahmad-al-soub/assignmentThree/p  
Backup Script Started: 2025-01-22_10-16-39  
Backup successful for: /home/ahmad-al-soub/assignmentThree/docs (Size: 4.0K)  
Backup successful for: /home/ahmad-al-soub/assignmentThree/photos (Size: 4.0K)  
Backup successful for: /home/ahmad-al-soub/assignmentThree/songs (Size: 4.0K)  
Backup Script Completed: 2025-01-22_10-16-39  
ahmad-al-soub@Al-Soub:~$
```

Figure 8

- A .tar.gz file will be created for each directory inside the /backup directory as in figure 9.

```
ahmad-al-soub@Al-Soub:~$ cd assignmentThree/backup/  
ahmad-al-soub@Al-Soub:~/assignmentThree/backup$ ls  
backup_log.txt photos_2025-01-22_10-16-39.tar.gz  
docs_2025-01-22_10-16-39.tar.gz songs_2025-01-22_10-16-39.tar.gz  
ahmad-al-soub@Al-Soub:~/assignmentThree/backup$
```

Figure 9

- A log file (backup_log.txt) will be created to record the status of the backup process as in figure 10.

```
ahmad-al-soub@Al-Soub:~/assignmentThree/backup$ cat backup_log.txt  
Backup Script Started: 2025-01-22_10-16-39  
Backup successful for: /home/ahmad-al-soub/assignmentThree/docs/ (Size: 4.0K)  
Backup successful for: /home/ahmad-al-soub/assignmentThree/photos/ (Size: 4.0K)  
Backup successful for: /home/ahmad-al-soub/assignmentThree/songs/ (Size: 4.0K)  
Backup Script Completed: 2025-01-22_10-16-39  
  
ahmad-al-soub@Al-Soub:~/assignmentThree/backup$
```

Figure 10



7. Error Test:

A. Test Case: No Arguments Passed:

- **Objective:** Ensure the script displays an error message when no directories are specified as arguments as in figure 11.

```
ahmad-al-soub@Al-Soub:~$ ./backup_script.sh
Backup Script Started: 2025-01-23_13-20-18
Error: No directories specified for backup.
```

Figure 11-A

```
ahmad-al-soub@Al-Soub:~/assignmentThree/backup$ cat backup_log.txt
Backup Script Started: 2025-01-22_10-16-39
Backup successful for: /home/ahmad-al-soub/assignmentThree/docs/ (Size: 4.0K)
Backup successful for: /home/ahmad-al-soub/assignmentThree/photos/ (Size: 4.0K)
Backup successful for: /home/ahmad-al-soub/assignmentThree/songs/ (Size: 4.0K)
Backup Script Completed: 2025-01-22_10-16-39
Backup Script Started: 2025-01-23_13-20-18
Error: No directories specified for backup.
```

Figure 11-B

B. Test Case: Non-Existent Directory Arguments

- **Objective:** Verify the script handles cases where the provided directory paths do not exist and logs appropriate error messages as in figure 12.

```
ahmad-al-soub@Al-Soub:~$ ./backup_script.sh /home/ahmad-al-soub/Desktop/Not-Directory
Backup Script Started: 2025-01-23_13-21-11
Error: Directory /home/ahmad-al-soub/Desktop/Not-Directory does not exist.
Backup Script Completed: 2025-01-23_13-21-11
ahmad-al-soub@Al-Soub:~$
```

Figure 12-A

```
ahmad-al-soub@Al-Soub:~/assignmentThree/backup$ cat backup_log.txt
Backup Script Started: 2025-01-22_10-16-39
Backup successful for: /home/ahmad-al-soub/assignmentThree/docs/ (Size: 4.0K)
Backup successful for: /home/ahmad-al-soub/assignmentThree/photos/ (Size: 4.0K)
Backup successful for: /home/ahmad-al-soub/assignmentThree/songs/ (Size: 4.0K)
Backup Script Completed: 2025-01-22_10-16-39
Backup Script Started: 2025-01-23_13-20-18
Error: No directories specified for backup.
Backup Script Started: 2025-01-23_13-21-11
Error: Directory /home/ahmad-al-soub/Desktop/Not-Directory does not exist.
Backup Script Completed: 2025-01-23_13-21-11
```

Figure 12-B



System Health Check:

- ◆ **The script generates a detailed health report when executed. And it performs the following:**
 1. Disk Space Check: Displays the used and available disk space on the root (/) partition.
 2. Memory Usage Check: Reports the current memory usage.
 3. Running Services Check: Checks the status of specific services (e.g., apache2 and **cron**). You can modify the services array to include any other services you'd like to monitor.
 4. System Updates Check: Identifies if there are any available updates using the **apt** Package Manager.
- ◆ **Create The System Health Check Script:**
 - Create a system-health-check_script.sh file and start writing commands as in figure 13 & 14.

```
ahmad-al-soub@Al-Soub:~$ pwd
/home/ahmad-al-soub
ahmad-al-soub@Al-Soub:~$ touch system-health-check_script.sh
ahmad-al-soub@Al-Soub:~$ nano system-health-check_script.sh
```

Figure 13



```

GNU nano 8.1                                system-health-check_script.sh *
#!/bin/bash

#System Health Script

#Function to check disk space.
function check_disk_space(){
    echo "Checking disk space..."
    df -h | awk 'NF==5'/{print "Disk Usage: "$5 " (" $3 " used out of "$2")"}'
}

#Function to check memory usage.
function check_memory_usage(){
    echo "Checking memory usage..."
    free -h | awk ' /^Mem:/ {print "Memory Usage: "$3 " used out of "$2 " (" $3 / $2 * 100 "%")"}'
}

#Function to check running services.
function check_running_services(){
    echo "Checking running services..."
    services=("cron" "apache2")
    for service in "${services[@]}";
    do
        if systemctl is-active --quiet "$service"; then
            echo "Service $service is running."
        else
            echo "Service $service is NOT running. Consider Starting it: sudo systemctl start $service"
        fi
    done
}

#Function to check recent system updates.
function check_system_updates(){
    echo "Checking for recent updates..."
    updates=$(apt list --upgradable 2>/dev/null | grep -v "Listing" | wc -l)
    if [ "$updates" -gt 0 ]; then
        echo "There are $updates update available. Consider running: sudo apt update && sudo apt upgrade"
    else
        echo "System is up-to-date."
    fi
}

#Generate Health report
function health_report(){
    echo "--- System Health Report ---"
    check_disk_space
    check_memory_usage
    check_running_services
    check_system_updates
    echo "--- End of Report ---"
}

#Then, Execute the health report
health_report

```

Figure 14



◆ Breakdown of the Script:

1. Disk Space Check:

- Checks the disk usage of the root (/) partition.
- Displays how much space is used, how much is available, and the percentage used as in figure 15.

```
echo "Checking disk space..."
df -h | awk '$NF=="/{print "Disk Usage: "$5 " ("$3" used out of "$2")"}'
```

Figure 15

- **df -h**: Lists disk space usage in a human-readable format.
- **awk '\$NF=="/{...}**: Filters the output to show only the root partition (/).
- Displays the disk usage as a percentage and the actual size used vs. total capacity.

2. Memory Usage Check:

- Checks the system's memory usage.
- Displays the total memory, used memory, and available memory as in figure 16.

```
echo "Checking memory usage..."
free -h | awk '/^Mem:/{print "Memory Usage: "$3" used out of "$2" ("$3/$2*100 "%")"}'
```

Figure 16

- **free -h**: Shows memory usage in a human-readable format.
- **awk '/^Mem:/{...}**: Filters the line showing memory usage statistics.
- Outputs the used memory vs. the total memory.

3. Running Services Check:

- Monitors specific services (e.g., **apache2**, **cron**).
- Checks if they are running or stopped and provides recommendations if they are not running as in figure 17.

```
echo "Checking running services..."
services=("cron" "apache2")
for service in "${services[@]";
do
    if systemctl is-active --quiet "$service"; then
        echo "Service $service is running."
    else
        echo "Service $service is NOT running. Consider Starting it: sudo systemctl start $service"
    fi
done
```

Figure 17



- **services=("cron" "apache2")**: Defines a list of services to monitor.
- **systemctl is-active --quiet "\$service"**: Checks if a service is active without outputting extra information.
- Suggests starting the service if it's not running.

4. System Updates Check:

- Check if there are any available updates for the system.
- Advises the user to run an update command if updates are found as in figure 18.

```
echo "Checking for recent updates..."
updates=$(apt list --upgradable 2>/dev/null | grep -v "Listing" | wc -l)
if [ "$updates" -gt 0 ]; then
    echo "There are $updates update available. Consider running: sudo apt update && sudo apt upgrade"
else
    echo "System is up-to-date."
fi
```

Figure 18

- **apt list --upgradable**: Lists packages that can be updated.
- **wc -l**: Counts the number of lines in the output to determine the number of updates.
- Suggests updating if there are packages available for upgrade.

5. Health Report Logging:

- Consolidates the results of all checks into a comprehensive health report.
- Outputs the report to the console as in figure 19.

```
#Generate Health report
function health_report(){
    echo "--- System Health Report ---"
    check_disk_space
    check_memory_usage
    check_running_services
    check_system_updates
    echo "--- End of Report ---"
}
```

Figure 19

- Calls all the individual functions (check_disk_space, check_memory_usage, etc.).
- Groups the results under a "System Health Report" heading.



◆ Run the Script:

- First, ensure the script is saved as system-health-check_script.sh and made executable as in figure 20:

```
ahmad-al-soub@AL-Soub:~$ ls
assignmentThree backup_script.sh Desktop Documents Downloads etc Music Pictures Public snap system-health-check_script.sh Templates Videos
ahmad-al-soub@AL-Soub:~$ chmod +x system-health-check_script.sh
ahmad-al-soub@AL-Soub:~$ ls
assignmentThree backup_script.sh Desktop Documents Downloads etc Music Pictures Public snap system-health-check_script.sh Templates Videos
```

Figure 20

- Output: First, I created a file and stored the output value from the shell file inside it as shown in Figure 21.

```
ahmad-al-soub@AL-Soub:~$ ./system-health-check_script.sh >/home/ahmad-al-soub/Desktop/health_reprot_before.txt
ahmad-al-soub@AL-Soub:~$ cd Desktop
ahmad-al-soub@AL-Soub:~/Desktop$ cat health_reprot_before.txt
--- System Health Report ---
Checking disk space...
Disk Usage: 8% (19G used out of 274G)
Checking memory usage...
Memory Usage: 3.9Gi used out of 14Gi (27.8571%)
Checking running services...
Service cron is running.
Service apache2 is NOT running. Consider Starting it: sudo systemctl start apache2
Checking for recent updates...
There are 91 update available. Consider running: sudo apt update && sudo apt upgrade
--- End of Report ---
ahmad-al-soub@AL-Soub:~/Desktop$
```

Figure 21

- As we can see, Ubuntu comes with several built-in services that are usually running by default. Among them is the **cron service** that handles scheduled tasks (cron jobs). But **apache2** is inactive by default unless it is manually installed and enabled. Also, we have **91 update** available
- So, we need to install, enable, update, and enable a service in Linux via the command line, as in figure 22.

```
ahmad-al-soub@AL-Soub:~$ sudo apt update
Get:1 https://dl.google.com/linux/chrome/debian InRelease [18.0 kB]
Hit:2 http://jo.archive.ubuntu.com/ubuntu InRelease
Get:3 http://jo.archive.ubuntu.com/ubuntu InRelease
Get:4 http://security.ubuntu.com/ubuntu InRelease
Get:5 http://jo.archive.ubuntu.com/ubuntu InRelease
Get:6 https://dl.google.com/linux/chrome/debian InRelease [18.0 kB]
Get:7 http://jo.archive.ubuntu.com/ubuntu InRelease
```

Figure 22-A

```
ahmad-al-soub@AL-Soub:~$ sudo apt install apache2
Installing:
  apache2

Installing dependencies:
  apache2-bin apache2-data apache2-utils libapr1

Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-selinux
```

Figure 22-B

```
ahmad-al-soub@AL-Soub:~$ sudo apt upgrade
The following package was automatically installed, is no longer required, and will be removed:
python3-netifaces
Use 'sudo apt autoremove' to remove it.

Upgrading:
  alsa-ucm-conf gir1.2-polkit-1.0
  bluez          gnome-bluetooth
  bluez-cups     gnome-bluetooth
  bluez-obexd    gnome-settings-daemon
```

Figure 22-C



◆ **Final Output:**

```
ahmad-al-soub@Al-Soub:~$ ./system-health-check_script.sh >/home/ahmad-al-soub/Desktop/health_report_after.txt
ahmad-al-soub@Al-Soub:~$ cd Desktop/; cat health_report_after.txt
--- System Health Report ---
Checking disk space...
Disk Usage: 8% (19G used out of 274G)
Checking memory usage...
Memory Usage: 3.9Gi used out of 14Gi (27.8571%)
Checking running services...
Service cron is running.
Service apache2 is running.
Checking for recent updates...
There are 32 update available. Consider running: sudo apt update && sudo apt upgrade
--- End of Report ---
ahmad-al-soub@Al-Soub:~/Desktop$
```

Figure 23

- For the 32 packages are held back from upgrading due to possible version mismatches.



Results:

The scripts performed as expected, automating their respective tasks effectively.

- **Backup Script:**

- Created compressed backups of specified directories, stored with clear naming conventions (e.g., /backup/docs_2025-01-22_10-16-39.tar.gz).
- Maintained a detailed log file, recording successful backups, their sizes, and any errors encountered.
- Example Log Output:

```
Backup Script Started: 2025-01-22_12-30-00
Backup successful for: /home/user/docs (Size: 1.2M)
Backup successful for: /home/user/photos (Size: 500K)
Backup Script Completed: 2025-01-22_12-30-00
```

Figure 24

- **System Health Check Script:**

- Provided a comprehensive health report, including:
 - Disk usage as a percentage and storage breakdown.
 - Current memory utilization out of total available memory.
 - Status of critical services, identifying inactive ones and recommending action.
 - Information on available updates with clear guidance on update commands.

Challenges:

Developing and testing the scripts involved addressing several challenges:

1. Backup Script:

- **Invalid Directory Handling:** Ensuring the script gracefully handled cases where directories were missing or invalid.
- **Compression Efficiency:** Balancing compression speed and effectiveness to ensure minimal backup time without compromising storage savings.
- **Log File Management:** Preventing the log file from growing excessively large by appending new entries while retaining old logs.



2. System Health Check Script:

- **Service Monitoring:** Ensuring compatibility across various Linux distributions for detecting service statuses.
- **Command Variability:** Accounting for differences in command outputs (df, free, etc.) across distributions.
- **Error Handling:** Adding safeguards for missing permissions or failed commands to prevent script crashes.
- **Update Notifications:** Simplifying update information to avoid overwhelming users while still being informative.

By addressing these challenges, we ensured the scripts were reliable and user-friendly across different environments.

Conclusions:

The assignment successfully demonstrates the power of automation in system administration. Both the Backup Script and the System Health Check Script simplify complex tasks, making them accessible and efficient for users.

1. **The Backup Script** ensures data security through automated, compressed backups with detailed logging and error handling.
2. **The System Health Check Script** empowers administrators to monitor system performance and stability with ease, consolidating critical metrics into a single report.

Together, these scripts highlight the importance of automation, error handling, and user feedback in improving system administration workflows. By streamlining these tasks, they contribute to enhanced reliability, reduced manual effort, and better system management overall.

🌈 I hope this report is technical and understandable to anyone with basic knowledge of Linux.

The End ...

