Shell Scripting Assignment

Yazan Abdullah

Contents

---

1 Introduction

In this report, I will present my solutions to the Linux assignment in Atypon Training.

This report is divided in two sections. The first section describes the backup shell script, and the second describes the system health check script.

Each section is further divided into subsections; one for every major optimization.

2 Backup Shell Script

2.1 General Overview

I wrote a shell script to handle backups, with the following features:

1. The ability to **incrementally** backup directories and files.
2. The ease to modify the tree hierarchy with minimal overhead.
3. Compression.
4. Encryption.
5. Logging
6. Extensible modular script.

User Guide

The following code block, demonstrates how to use the script:

```
backup.sh mode args...
```

There are several modes you can choose from:

1. **backup** mode: Allows you to add files to the backup image.
2. **update** mode: Allows you to update (replace) the backup image.
3. **clean** mode: to remove unreferenced files after an update.
4. **encrypt/decrypt** modes: Allows you to encrypt your data to enhance security, and decrypt them back again.
5. **compress/extract** modes: Allows you to compress your files to save some space, and then extract them back.

6. **stats** mode: displays the number of files in the backup images, and its size.

To backup a directory or a file, you can run the following command:

```
backup.sh backup path/to/file path/to/backup/dir
```

This will, **add** the specified file, to the image. Compare it, however, with the following command:

```
backup.sh update path/to/dir/to/backup path/to/backup/dir
# you may need to clean old files after updates
backup.sh clean path/to/backup/dir
```

The update mode does two things; it adds the files that you specified and removes those that you didn't; thus, it replaces the backup image, with the new data you provide.

However, note that this removal is shallow; that is, it only appears to the user that their files are deleted, while they will still be stored in the backup image; to actually removed the files, you need to explicitly call the clean mode. This feature aims to avoid data loss catastrophes due to simple mistakes.

You can also perform compression operations, to save some disk space as follows:

```
backup.sh compress path/to/backup/dir
backup.sh extract path/to/backup/dir
```

Similarly, you can encrypt/decrypt your files but you will need to add a password, as follows:

```
backup.sh encrypt password path/to/backup/dir
backup.sh decrypt password path/to/backup/dir
```

Note: in the current implementation, the encryption function does compression by default; so there is no need to chain operations, (more on this in Section 2.4).

Lastly, you can display some statistics about your backup image, as follows:

```
backup.sh stats path/to/backup/dir
```

Important Notes for Advanced Users

Here, I will list some under the hood details, which I hope to be helpful in debugging any errors.

First, All files and meta data are stored in the .data directory. There you can find a sqlite3 database: metadata.db, a logs file: logs.txt and the files directory where all the files are stored. Because every file can be referenced multiple times, there was no way for me to decide which name they should take, so I decided to name every file with the *original* hash value of it; I write original because this name doesn't change after compression, for example.

2.2 Optimizing Space

I decided to store all files in a hidden directory. The visible directories to the user are merely hard links to the actual files.

This setting, makes it very easy to manage duplicate files; I only need to store unique files in the hidden directory, and then use multiple hard links to refer to them.
Note: the backup image isn't meant to be changed manually by the user, if they want to change it anyway, they should do it out of the backup directory, then use the script to add the modified files.

2.3 Incremental Backups

There are two modes with which the user can backup their files: backup, and update mode, see Section 2.1.

In either mode, the script only copies the new files to the backup directory. Remember, the script stores all the files in a hidden directory, and keeps track of them using a database. So before adding any file to the backup image, it first checks whether the file already exist, if so no move operation is needed.

Each file has a unique ID that I get by using its hash digest; and for efficiency, I compare files by comparing their IDs.

2.4 Optimizing Compression

I added the ability to compress and decompress files in the backup script. following are the optimizations:

1. **Seamless compression**: compression only affects the size of the stored files, that is it will not disrupt other operations like backing up more files, or surfing the files tree.

2. **Parallel compression**: instead of using `gzip` which is single threaded, I went with the multi-threaded app `plgz`. This resulted in faster compression and extraction times.

3. **Compression before Encryption**: one of the subtle but important features of my script is it only allows you to compress then encrypt files. This is the right order, because compression relies on the regularity in the data, and encryption aims to scramble the data and make it look random. Thus, it results in far smaller files when we apply these operations in this order. Also note, in my implementation, the encryption command compresses files by default, so the user needn't worry about compression when they encrypt files; however, if a different command is used, they will most certainly appreciate this feature, See Figure 1.

```
epoch@fedora:~/scripts/learning/atypon/backup/testing$ gpg --compression
 0 --symmetric abc
epoch@fedora:~/scripts/learning/atypon/backup/testing$ gzip abc.gpg
epoch@fedora:~/scripts/learning/atypon/backup/testing$ ls -lh
total 127M
-rw-r--r--. 1 epoch epoch 56M Jul 11 19:56 abc
-rw-r--r--. 1 epoch epoch 56M Jul 11 19:58 abc.gpg.gz
-rw-r--r--. 1 epoch epoch 15M Jul 11 19:57 abc.gz.gpg
```

Figure 1: Compression Efficiency and the Order of Compression and Encryption

2.5 SQLite Database

I used a sqlite database to store the ids of the files in the backup image and their meta data.

Using sqlite has two advantages; first, It is widely used and most developers know it, so if something goes wrong with the script, they will manage to retain their meta data in a format they can understand; second, It offers several features for optimizing querying the data, like database indexes and loading only the data it needs.

I decided to store the logs in a text file outside sqlite database; I did so, because I don't need to do complex operations on the files (just appending lines), and because text files are harder to corrupt, and easier to retrieve data from, which makes them better suited for storing the logs.

3 System Health Check Script

3.1 Health Checks

Following are the different system health checks my script does:

1. Check for Updates: In this window, I display the number of available updates. I made this script work for both `apt` and `dnf`.

2. Services: In this window, I display the statuses of user specified services using the `systemctl` command. Different Statuses are: Running, Inactive, Disabled, uninstalled.

3. Kernel Messages: In this window, I display the last 7 kernel messages, using the `dmesg` command. This window requires super user permissions, to get the logs.

4. CPU Utilization: In this window, I display the number of CPU cores in the system; and the real-time CPU utilization. I do so, by reading the `/proc/cpuinfo` file, and using the `mpstat` command.

5. Disk Utilization: In this window, I display the size and available space, of some user specified mount points or partitions, using the `df` command.

6. Memory Utilization: In this window, I display the total amount of and available space of the main memory and swap memory, using the `free` command.

7. Network Activity: In this window, I display the total amount of transmitted and received data through the network; typically, since the last boot, but it's possible for the user to reset stats. I used `ifstat` for this purpose.

8. Battery Health: In this window, I display the current capacity/charge of the battery in the computer, its status (Charging, Discharging, FULL), and its health. I calculated the health as the ration of current maximum capacity divided by the maximum capacity when the battery was new. I gathered these information from the `/sys/class/power_supply/BAT0` directory.

3.2 Enhancing the UI

Since this script is not dynamic, and fairly simple, I thought I should enhance the user experience and interface. I tried my best to keep the code very clean, so anyone can add to it with ease. I also decided to go with a TUI; for that, I found a repository on GitHub called `bashsimplecurses: https://github.com/metal3d/bashsimplecurses`; which provides a simple and intuitive interface, to build windows in the terminal as following, also See Figure 2.

```
# an example window
# window title, title color, width
window "Services" "yellow" "50%"
# 2 is for two columns and | is the seperator here
append_tabbed "Name|Status" 2 "|"
append_tabbed "Docker|docker_status" 2 "|"
append_tabbed "MySQL|mysql_status" 2 "|"
endwin
```

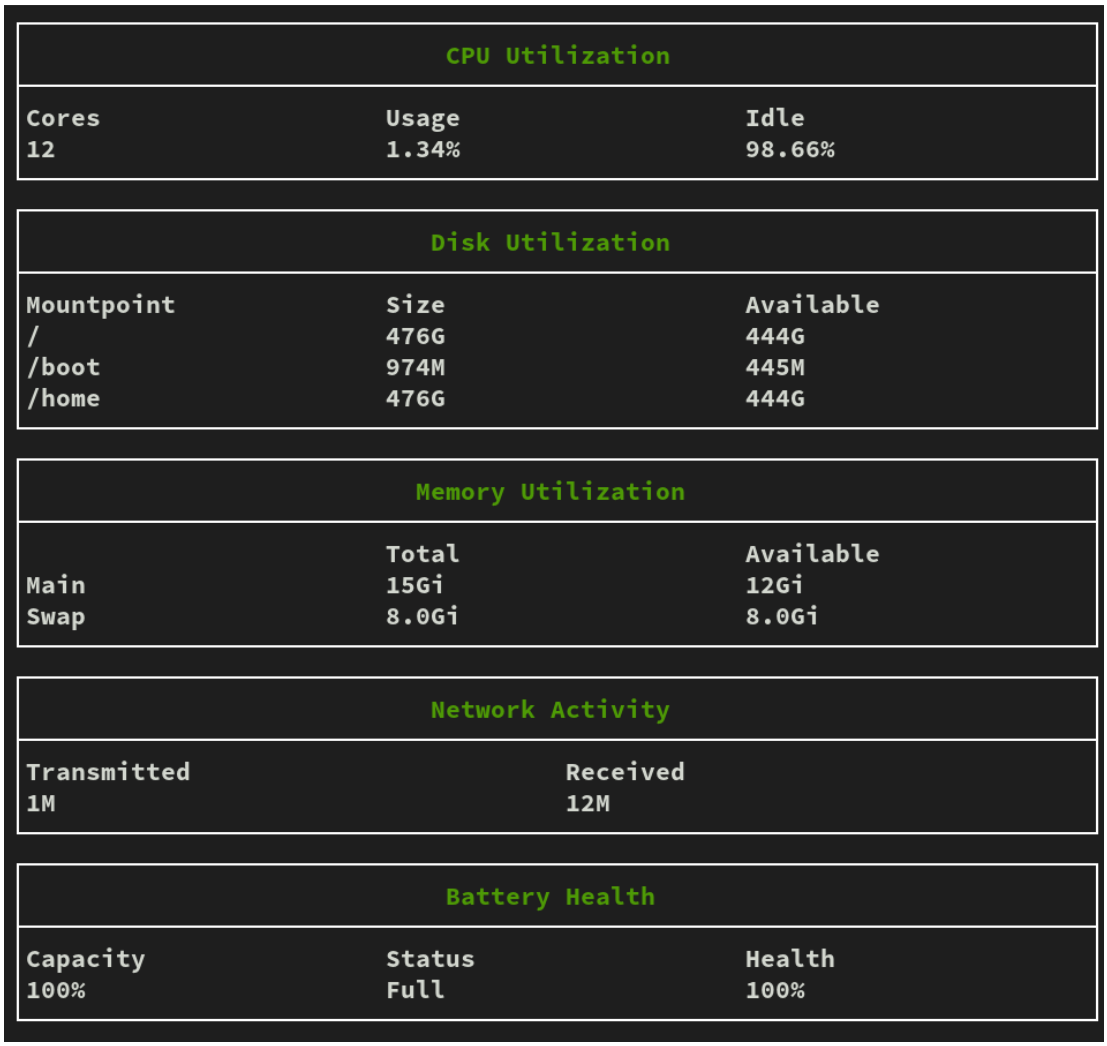

Figure 2: TUI interface using bash curses