ATYPON

Linux Scripts Report

By Hashim Emad

Table of Contents

Introduction To Bash Scripting

Bash is a very powerful scripting language. It is one out of many Linux scripting languages. It gives us the ability to automate tasks, commands and many more. Let's say we need to write 30 commands every day at some o'clock, we can write them inside a bash script and schedule them at that time. Also, we can create some logic utilizing if statements and the different types of loops. In this report I will discuss two scripts,

The first one is a backup script that takes the paths of the files to backup, and the destination where to save the backup, and gives you the option to compress the files to be backed-up.

The second script is the health script, it displays system-related information such as storage usage, memory usage, running services and recent system updates

Backup Script

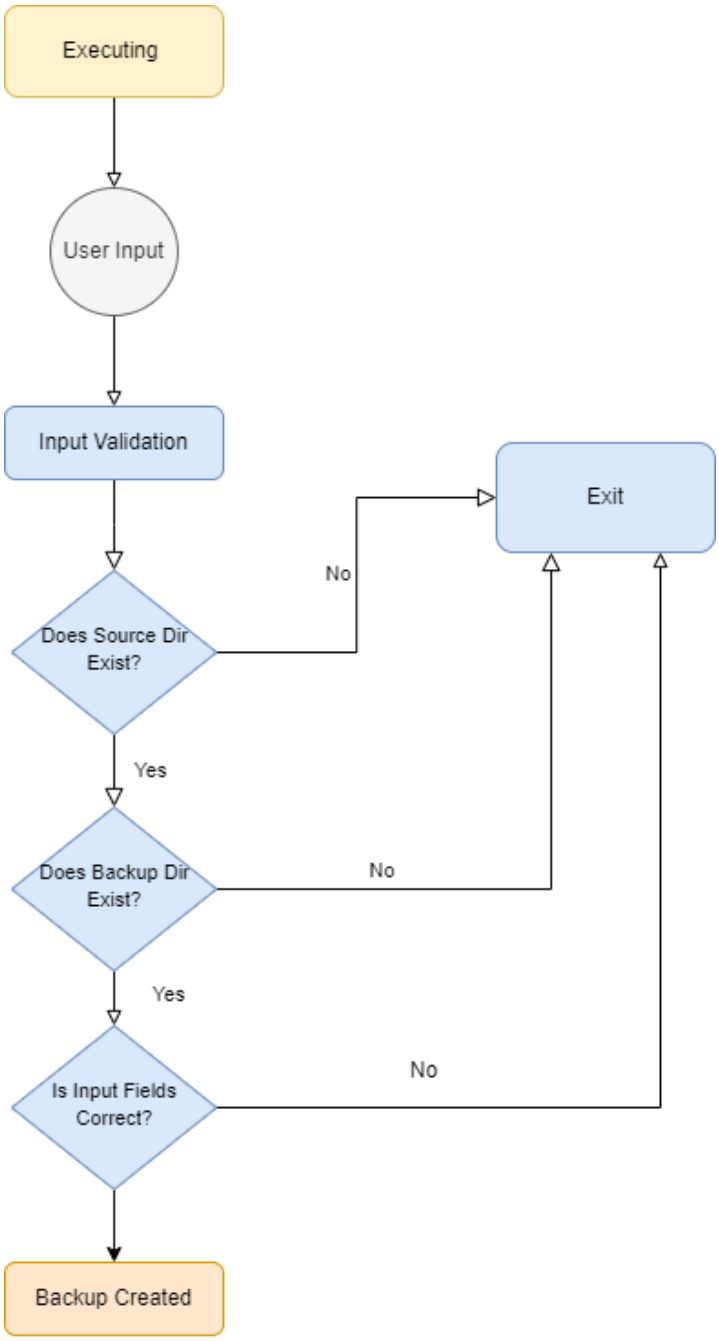
Purpose and Functionality:

Imagine you are managing a web server that hosts multiple websites. It's crucial to back up the website data and configuration files regularly to prevent data loss in case of hardware failure, accidental deletion, or cyber-attacks. By using the backup script, you can automate the process of backing up important files to a secure location, ensuring that you can quickly restore the websites from the backups if needed.

The script allows users to specify the paths of files to be backed up and the destination where the backup should be stored. Additionally, it offers options for compressing and encrypting the backup, providing enhanced security and efficiency. Using this script ensures that backups are performed consistently

and reliably, with comprehensive logging and error handling to address any issues that may arise.

Flowchart Of The Script:



Optimizations:

Note: bonuses are extra none required optimizations I added to the script.

Dependency Check: The script checks for the required dependencies (rsync, tar, gpg, ssh) before executing the backup process. This ensures that the necessary tools are available and prevents potential errors.

Input Validation: The script validates user inputs to ensure they are valid and complete. It checks if the source directory exists, if the backup directory can be created (for local backups), and if the required information is provided for remote backups and encryption.

(BONUS) Remote Backup Destination: The script supports remote backups by using rsync over SSH. It requires users to have SSH access to the remote destination where the backup should be stored, enhancing flexibility in backup storage options.

(BONUS) Excluding Options: Users can specify directories, files, and file extensions to exclude from the backup, allowing for customization of the backup scope.

Cleanup Mechanism: The script includes a cleanup mechanism using a trap statement. If the script is interrupted, it removes any incomplete backup files created within the last 5 minutes to free up space and maintain a clean backup directory.

Logging: The script logs important information and status messages to a log file named with a timestamp. This helps in troubleshooting and monitoring the backup process.

Disk Space Check: Before creating a local backup, the script checks if there is sufficient disk space available in the backup directory. If the required space exceeds the available space, the script aborts the backup process to prevent disk space issues.

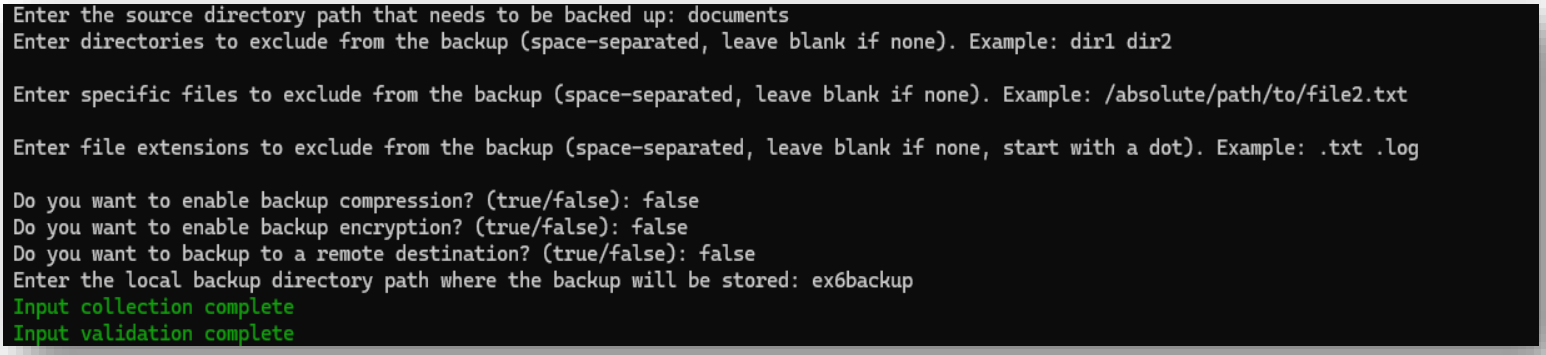
Compression: The script provides options for backup compression using tar. Used when we want efficient storage usage and faster transfer times during backups, especially beneficial for large datasets.

(BONUS) Encryption: The script provides options for Encryption using 'gpg' to ensures confidentiality of the backup data stored locally or transmitted to remote destinations, enhancing data security.

Performance Analysis:

The script utilizes rsync for efficient incremental backups, minimizing the amount of data transferred. It also provides options for excluding specific directories, files, and file extensions to further optimize the backup process. The script checks for available disk space before creating the backup to ensure sufficient storage capacity.

Why Backup Script Uses Question-Style Input?



The backup script utilizes a question-style input method for several important reasons:

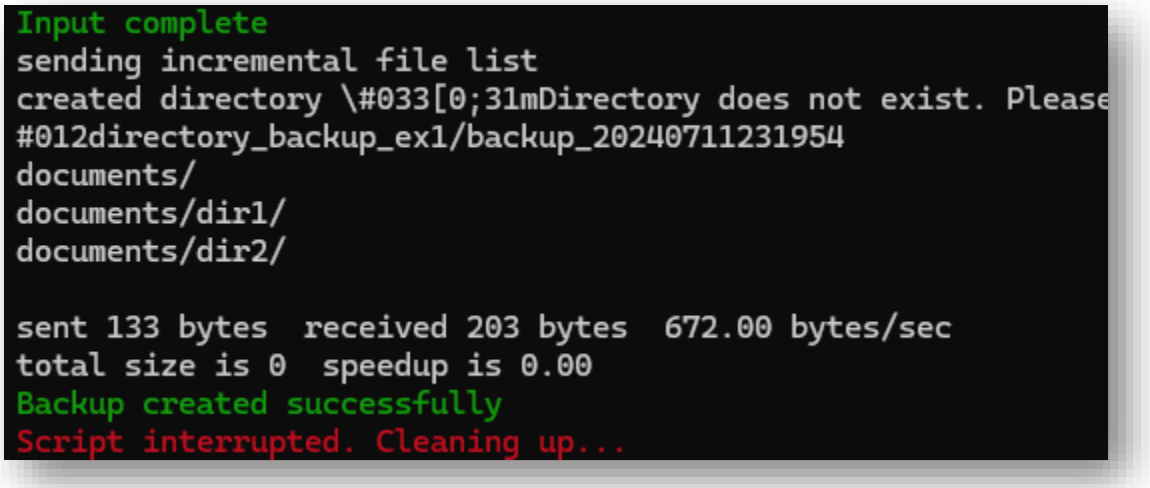
- **User Guidance:** By asking specific questions, the script guides users through the setup process, ensuring they understand what information is required and reducing the risk of errors.

- **Customization and Flexibility:** The script can collect detailed preferences from the user, such as which directories and files to include or exclude, whether to use compression or encryption, and whether to store backups locally or remotely.
- **Error Prevention:** By interacting with the user, the script can immediately validate the inputs, ensuring that all required information is provided and is valid before proceeding. Users receive instant feedback if their inputs are incorrect, allowing them to correct mistakes on the spot.
- **Ease of Use:** Even users with limited technical knowledge can follow the prompts and provide the necessary information without needing to edit the script directly. The interactive nature of question-style input makes the script more engaging and easier to use compared to editing configuration files or passing numerous command-line arguments.
- **Consistency and Reliability:** The script ensures that all required information is gathered in a consistent manner, leading to more reliable backups.

Exception Handling

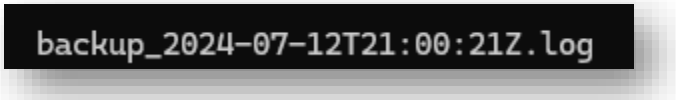
The backup script includes robust exception handling mechanisms to ensure smooth operation and graceful recovery from potential errors:

- **Trap Statement:**



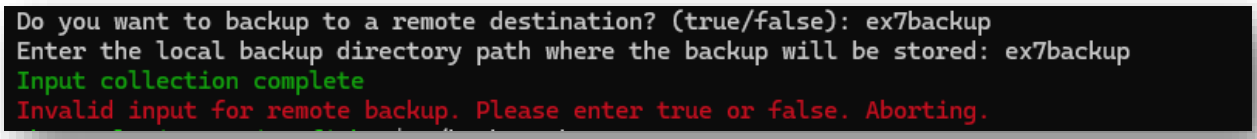
  Utilizes a trap statement to capture interrupts. Upon interruption, the script performs cleanup actions, such as removing incomplete backup files, to maintain a consistent and clean backup environment.

- **Error Logging:**



  Logs error messages and status updates to a designated log file (backup_<timestamp>.log). This allows administrators to review and troubleshoot issues encountered during the backup process, facilitating efficient resolution of errors.

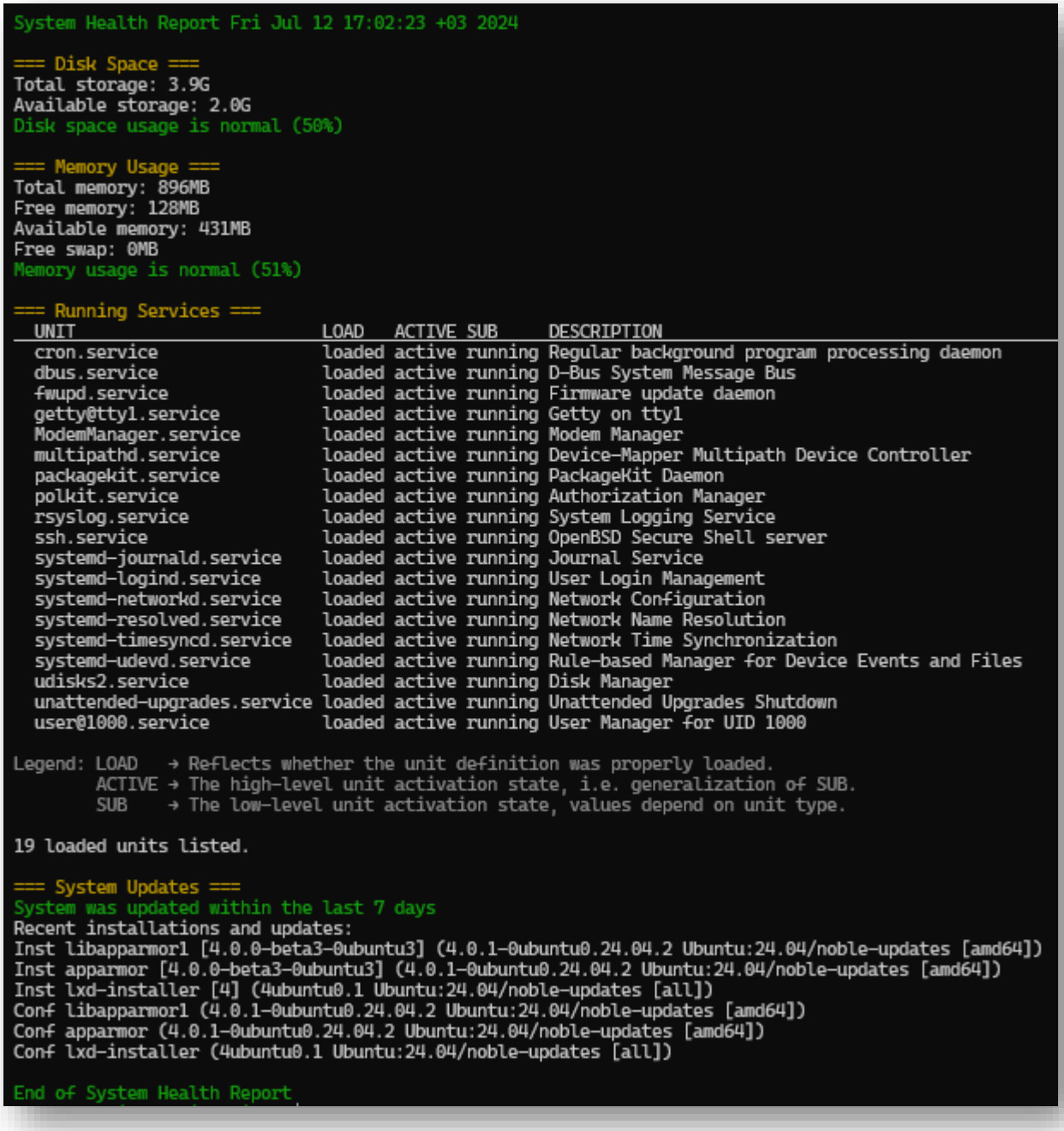
- **Input Validation:**



  Validates user inputs rigorously to prevent execution with invalid or incomplete parameters. If any required parameter is missing or incorrectly formatted, the script provides informative error messages and prompts the user to correct the input before proceeding with the backup.

- **Dependency Check:** Verifies the availability of essential dependencies (rsync, tar, gpg, ssh) before executing backup operations. If any dependency is missing, the script terminates execution with an error message, guiding users to install the required software for successful backup execution.
- **Disk Space Check:** Before initiating a backup, the script checks the availability of adequate disk space in the designated backup directory. If insufficient space is detected, the script aborts the backup process and notifies the user to free up disk space or allocate a larger storage volume for uninterrupted backup operations.

System Health Check Script



```
System Health Report Fri Jul 12 17:02:23 +03 2024

=== Disk Space ===
Total storage: 3.9G
Available storage: 2.0G
Disk space usage is normal (50%)

=== Memory Usage ===
Total memory: 896MB
Free memory: 128MB
Available memory: 431MB
Free swap: 0MB
Memory usage is normal (51%)

=== Running Services ===
  UNIT                         LOAD   ACTIVE SUB     DESCRIPTION
  cron.service                 loaded active running Regular background program processing daemon
  dbus.service                 loaded active running D-Bus System Message Bus
  fwupd.service                loaded active running Firmware update daemon
  getty@tty1.service           loaded active running Getty on tty1
  ModemManager.service         loaded active running Modem Manager
  multipathd.service           loaded active running Device-Mapper Multipath Device Controller
  packagekit.service           loaded active running PackageKit Daemon
  polkit.service               loaded active running Authorization Manager
  rsyslog.service              loaded active running System Logging Service
  ssh.service                  loaded active running OpenBSD Secure Shell server
  systemd-journald.service     loaded active running Journal Service
  systemd-logind.service       loaded active running User Login Management
  systemd-networkd.service     loaded active running Network Configuration
  systemd-resolved.service     loaded active running Network Name Resolution
  systemd-timesyncd.service    loaded active running Network Time Synchronization
  systemd-udevd.service        loaded active running Rule-based Manager for Device Events and Files
  udisks2.service              loaded active running Disk Manager
  unattended-upgrades.service loaded active running Unattended Upgrades Shutdown
  user@1000.service            loaded active running User Manager for UID 1000

Legend: LOAD   → Reflects whether the unit definition was properly loaded.
        ACTIVE → The high-level unit activation state, i.e. generalization of SUB.
        SUB    → The low-level unit activation state, values depend on unit type.

19 loaded units listed.

=== System Updates ===
System was updated within the last 7 days
Recent installations and updates:
Inst libapparmor1 [4.0.0-beta3-0ubuntu3] (4.0.1-0ubuntu0.24.04.2 Ubuntu:24.04/noble-updates [amd64])
Inst apparmor [4.0.0-beta3-0ubuntu3] (4.0.1-0ubuntu0.24.04.2 Ubuntu:24.04/noble-updates [amd64])
Inst lxd-installer [4] (4ubuntu0.1 Ubuntu:24.04/noble-updates [all])
Conf libapparmor1 (4.0.1-0ubuntu0.24.04.2 Ubuntu:24.04/noble-updates [amd64])
Conf apparmor (4.0.1-0ubuntu0.24.04.2 Ubuntu:24.04/noble-updates [amd64])
Conf lxd-installer (4ubuntu0.1 Ubuntu:24.04/noble-updates [all])

End of System Health Report
```

Purpose and Functionality:

The system health check script provides a comprehensive overview of the system's health by checking various aspects such as disk space usage,

memory usage, running services, and available system updates. It generates a report with the collected information and provides recommendations based on the findings.

Optimizations:

Command Existence Check: The script checks if the required commands (systemctl, service, apt-get, yum) are available on the system before executing them. This prevents potential errors and provides fallback mechanisms if certain commands are not found.

Colored Output: The script uses color codes to highlight important information and warnings, making the report more visually appealing and easier to read.

Update Check: The script checks for available system updates and provides recommendations based on the findings. It displays the number of days since the last update (for apt-get) or the availability of updates (for yum), along with the list of recent installations and updates.

Header Function: The script includes a helper function called print_header to print section headers in a consistent and visually appealing format. This enhances the readability of the generated report.

Performance Analysis:

The script efficiently retrieves system information using built-in Linux commands and utilities. It analyzes disk space usage, memory utilization, and the status of running services. The script also checks for available system updates using the appropriate package manager (apt-get or yum).

Conclusion:

Both the backup script and the system health check script are well-designed and optimized for their respective purposes. The backup script offers a flexible and secure way to create backups of specified directories, with options for compression, encryption, and remote backup destinations. The system health check script provides a comprehensive overview of the system's health, including disk space usage, memory utilization, running services, and available updates. The optimizations implemented in both scripts enhance their reliability, efficiency, and user-friendliness, making them valuable tools for system administrators and users with basic Linux knowledge.