Linux Backup and System Health Shell Scripting Report

Author: Ahmad Emad

GitHub: Linux Backup & System Health Shell Scripting
YouTube Video: Demonstration
Email: ahmademad995.ae@gmail.com

Objective Overview:

The primary objective of this project is to provide a robust, flexible, and user-friendly backup solution for Linux systems, along with a comprehensive system health check utility. The project encompasses two key scripts:

1. **Backup Script (backup.sh)**:
    - **Goal**: To automate the backup process of critical directories, ensuring data security through options for encryption, compression, and exclusion filters.
2. **System Health Script (system-health.sh)**:
    - **Goal**: To monitor and report on critical system metrics, including disk space, memory usage, running services, and system updates.

Table of Contents

Introduction

In this report, we explore shell scripting for system health checks and backup procedures. The scripts' objectives are to monitor system performance and guarantee data integrity through routine backups to prevent possible problems.

Users may safely backup important files and directories with the help of the backup script, which also handles failures, logs backup statuses, and compresses the data. The script minimizes human mistake and guarantees consistent and dependable backups by automating these activities. The system health check script also keeps an eye on a number of system metrics, including disk usage, CPU load, and memory consumption, and notifies users of any possible problems before they become serious ones.

The main features, implementation specifics, and error-handling procedures are described in this report.

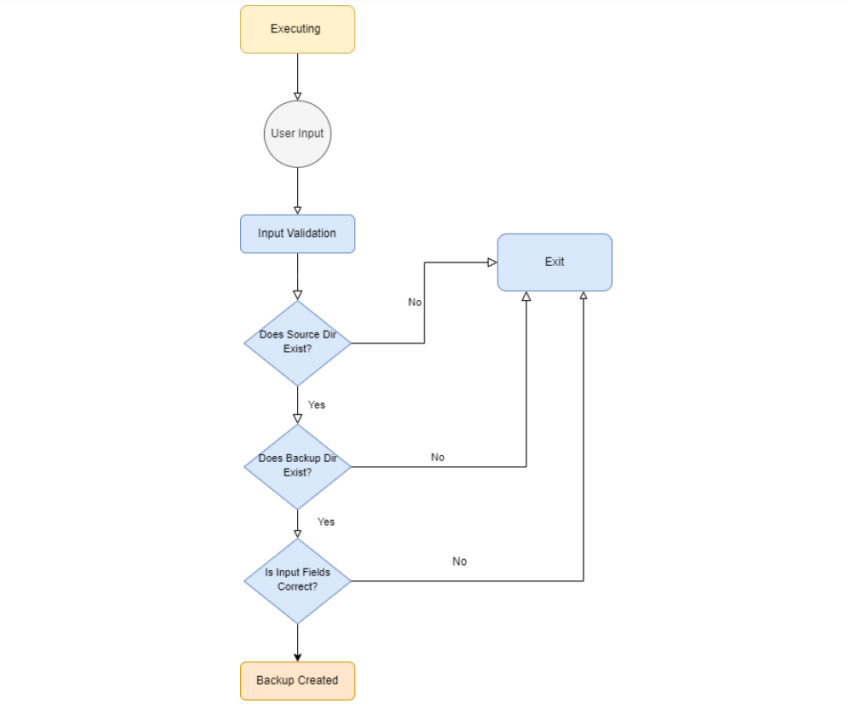
Backup Script

Overview

Users can designate where the backup should be saved, and which file paths should be backed up using the script. Furthermore, it provides choices for encrypting and compressing the backup, enhancing security and effectiveness.

By using this script, backups are guaranteed to be made frequently and dependably, with thorough logging and error handling to handle any potential problems.

Flowchart

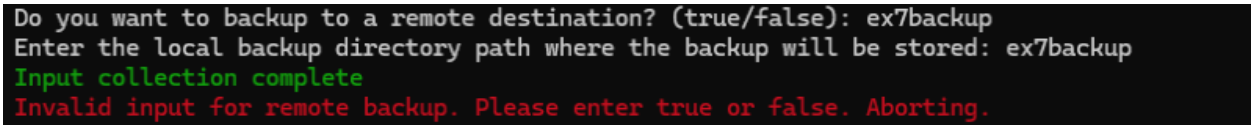


Optimizations

Interactivity: The script starts by determining the most effective backup method for the user. It prompts for the following inputs:

- Subdirectories to exclude from the backup
- Specific files to exclude, such as (not-important-file.txt)
- File extensions to exclude, such as (.Log)
- An option to enable compression for the backup
- An option to enable encryption for the backup
- An option to perform the backup to a remote destination, such as a remote server

Input Validation: The script verifies user inputs to ensure they are accurate and complete. It checks whether the source directory exists, ensures that the backup directory can be created (for local backups), and confirms that all necessary details are provided for remote backups and encryption.



Remote Backup Destination: The script facilitates remote backups using rsync over SSH, allowing users to store backups on a remote server. It requires SSH access to the remote destination, providing greater flexibility in choosing backup storage locations.

Logging: The script records key information and status messages in a log file, which is named with a timestamp. This feature aids in troubleshooting and allows for effective monitoring of the backup process.

Disk Space Check: Prior to creating a local backup, the script verifies whether there is enough available disk space in the backup directory. If the required space exceeds the available capacity, the script halts the backup process to avoid potential disk space problems.

Compression: The script offers an option to compress backups using tar. This feature is useful for optimizing storage usage and reducing transfer times, particularly when dealing with large datasets, making the backup process more efficient.

Encryption: The script includes an option to encrypt backups using 'gpg', ensuring the confidentiality of backup data, whether stored locally or transmitted to remote destinations. This feature enhances data security by protecting sensitive information.

Exception Handling

Cleanup Mechanism: The script incorporates a robust cleanup mechanism using a trap statement. In the event of an interruption or error during the backup process, the script ensures that any incomplete backup files created within the last 5 minutes are deleted. This helps to free up space and maintain a clean and organized backup directory, minimizing the impact of errors and ensuring no partial or corrupted backups are left behind.

```
Input complete
sending incremental file list
created directory \#033[0;31mDirectory does not exist. Please
#012directory_backup_ex1/backup_20240711231954
documents/
documents/dir1/
documents/dir2/

sent 133 bytes  received 203 bytes  672.00 bytes/sec
total size is 0  speedup is 0.00
Backup created successfully
Script interrupted. Cleaning up...
```

Error Logging: The script logs error messages and status updates to a designated log file (backup_<timestamp>.log). This allows administrators to monitor and review any issues that arise during the backup process, enabling quick identification and resolution of errors for smoother operation and minimal downtime.

```
backup_2024-07-12T21:00:21Z.log
```

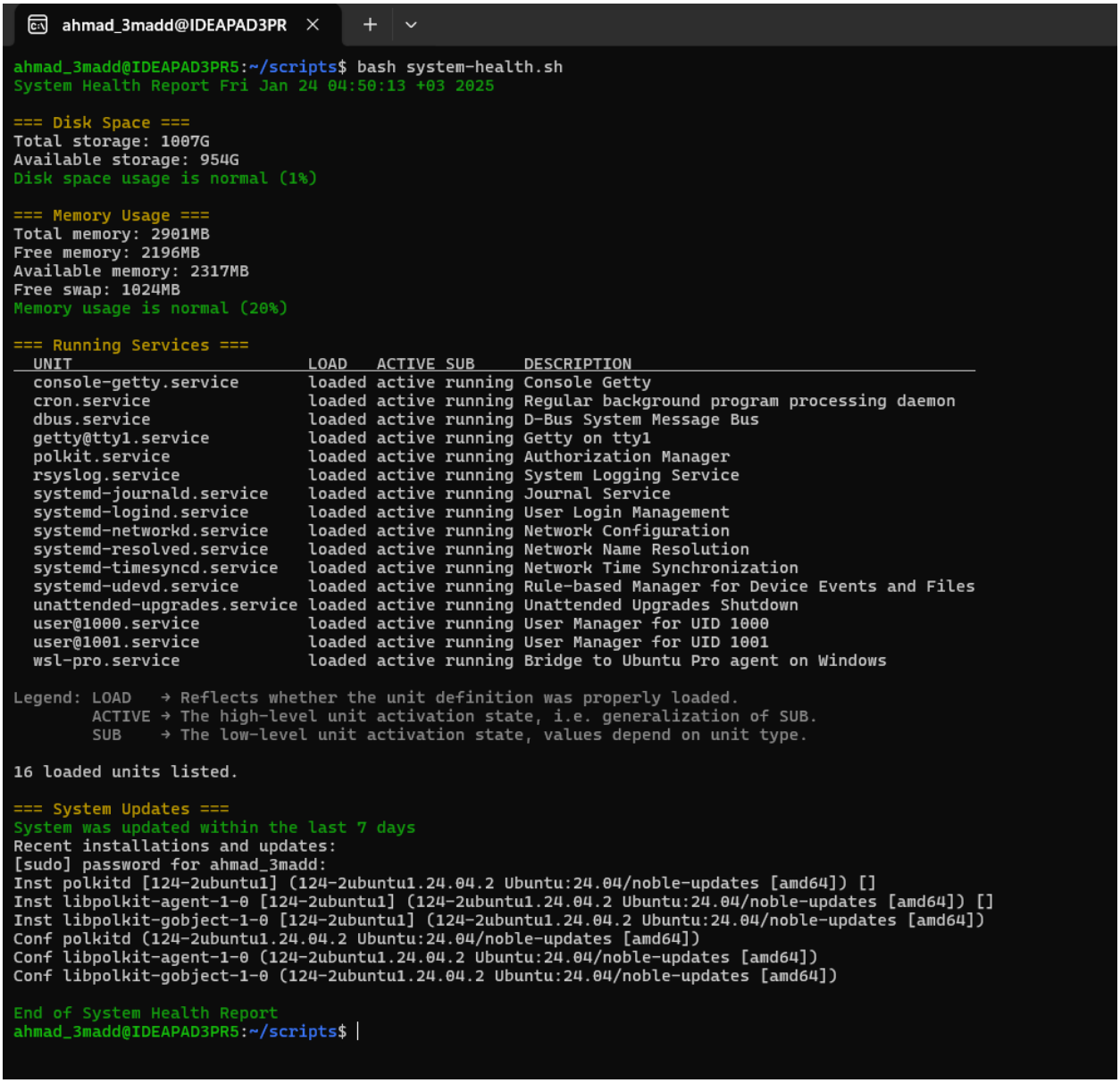
Performance Analysis

The script leverages rsync to perform efficient incremental backups, reducing the amount of data that needs to be transferred. It also offers options to exclude specific directories, files, and file extensions, further optimizing the backup process. Additionally, the script checks available disk space before initiating the backup to ensure there is enough storage capacity. This combination of features helps to maximize efficiency and minimize unnecessary resource usage during the backup process.

System Health Check Script

Overview

The System Health Script (system_health.sh) is designed to provide an automated and comprehensive report on the critical health metrics of a Linux system. Its primary purpose is to help system administrators quickly assess the status of key resources and identify potential issues before they impact system performance or reliability.



Features

Disk Space Monitoring: The script checks the disk space usage on the root filesystem (/) and provides a report on total storage, available storage, and usage percentage. It raises a warning if disk space usage exceeds 90%, suggesting actions to free up space or expand storage capacity.

Memory Usage Monitoring: The script tracks both physical memory (RAM) and swap usage, providing a breakdown of total, free, and available memory. A warning is issued if memory usage exceeds 90%, advising users to close unnecessary applications or consider adding more RAM.

Service Monitoring: The script checks for running services using either systemctl or service commands, depending on the system configuration. It ensures that all critical services are running and available. This helps administrators ensure that essential services are operational, minimizing downtime.

System Update Status: The script checks if the system has been updated recently by examining package manager cache files (using apt-get or yum). It notifies users if the system hasn't been updated in a week or more, providing recommendations to run system updates and displaying recent installations or updates.

Customizable Warnings: The script provides warnings when system resource usage (disk space, memory, or system updates) exceeds predefined thresholds, making it easier to manage system health proactively.

Optimizations

Command Existence Check: The script performs a check to determine whether critical commands such as systemctl, service, apt-get, and yum are available on the system before attempting to execute them. This step prevents errors caused by missing dependencies and ensures the script functions properly on different distributions. If one command isn't available, the script falls back to another method (e.g., using service when systemctl isn't available), which improves its robustness and compatibility across various systems.

Colored Output: The script employs color codes to enhance the readability and appeal of the output. Warnings are highlighted in **red**, informational messages in **green**, and section headers in **yellow**. This color-coding improves the user experience, making it easier to spot issues or important information at a glance. For instance, if disk space usage exceeds the 90% threshold, it will be displayed in red, drawing attention to potential problems.

Update Check: The script includes logic to check for available system updates. It displays the time elapsed since the last update (for systems using apt-get) or shows the available updates (for systems using yum). This feature ensures that administrators stay informed about the status of system packages and can act promptly to keep the system up-to-date. It also lists recent installations and updates, allowing administrators to review recent changes to the system.

Header Function: To improve the visual structure of the output, the script uses a helper function called print_header to print section headers in a consistent, visually appealing format. This function helps break the report into distinct sections (e.g., Disk Space, Memory Usage, System Updates) and makes it easier for users to navigate the generated health report. This organization improves the overall readability of the output, especially for larger systems with many resources to monitor.

Performance Analysis

The script efficiently collects system data using native Linux commands (df, free, systemctl, service, apt-get, yum). It checks disk space, memory usage, running services, and system updates with minimal overhead. The script offers real-time analysis and actionable recommendations, ensuring fast execution and low resource consumption.

Conclusion

Both the backup and system health check scripts are designed with efficiency and flexibility in mind. The backup script allows users to create secure backups with options for compression, encryption, and remote destinations, while ensuring disk space and input validation for reliable performance. The system health check script provides an overview of key system metrics, such as disk space, memory usage, and running services, along with available updates, helping administrators maintain system health.

Optimizations like command checks, color-coded output, and structured functions improve usability and performance, making these scripts valuable tools for Linux system management.