LINUX ASSIGNMENT

Bash Script Tool “Mojaks-Mix”

Done by:

Mohammad Alsabi

Supervisors:

Dr. Motasem Aldiab

ABSTRACT

This report introduces Mojaks-Mix tool, housed in the Mojaks-Mix folder, utilizes a main bash file ("Controller") to process commands and invoke specific action files (BackUpAction, HealthCheckAction, DisplayLogs). It enforces a consistent file structure. For backups, users can execute Mojaks-Mix -c –zip "source" "destination", creating a compressed copy. The tool also supports Mojaks-Mix -log to display previous logged actions. Health checks (Mojaks-Mix -hc) assess system health, issuing warnings for high CPU or memory usage. Error messages guide users to the help command.

ACKNOWLEDGMENT

I would like to express my appreciation and gratitude to everyone that supported me and provided me with the help I needed on my journey to finish this project.

I would also like to give special thanks to my project's supervisors, **Dr. Motasem Aldiab**, for his continued help and support.

Table of Contents

Table

List of Figures

CHAPTER ONE: Purpose and Functionality

1.1 Controller

The "Controller.sh" script serves as the central orchestrator for the Mojaks-Mix tool. It processes various commands, including creating a compressed copy of a folder ("-c" or "copy"), conducting a system health check ("-hc" or "health-check"), and displaying the content of the log file ("-log" or "logs"). The script provides a help option ("-h") for command usage guidance. It utilizes functions to handle backup actions, health check actions, and log displays. Additionally, the script defines regular expressions to validate input values and flags, ensuring proper command execution. The main script section checks for provided arguments and calls the appropriate functions based on user input. The script is designed to gracefully handle errors and display help messages when needed, but these messages will be common because the real handling will be in the file that is responsible of the action, so that will serve the purpose of the controller of being an orchestrator.

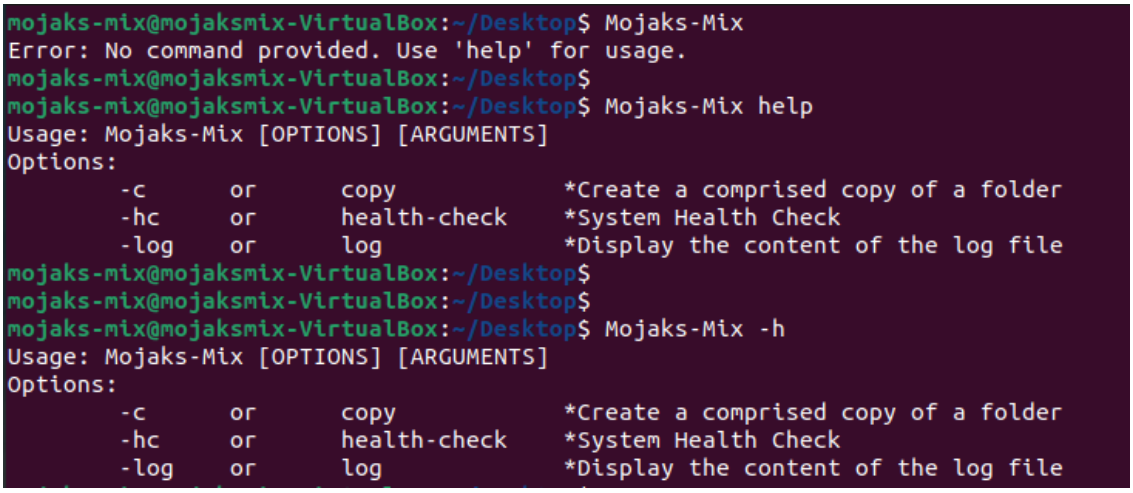

Figure 1: Calling Mojaks-Mix Tool and Help Flag

1.2 Health Check

The "health-check.sh" script evaluates the system's health by performing various checks. It includes functions to assess disk space, memory usage, CPU usage, running services, and recent system updates. The script generates a detailed PDF report with the findings, including a section on recent updates. Additionally, it checks for high memory and CPU usage, issuing warnings if thresholds are exceeded. The results are logged in the "health_check_log.txt" file, and the generated PDF report is opened for user reference. The script's modularity allows easy integration into the Mojaks-Mix tool for comprehensive system health monitoring.

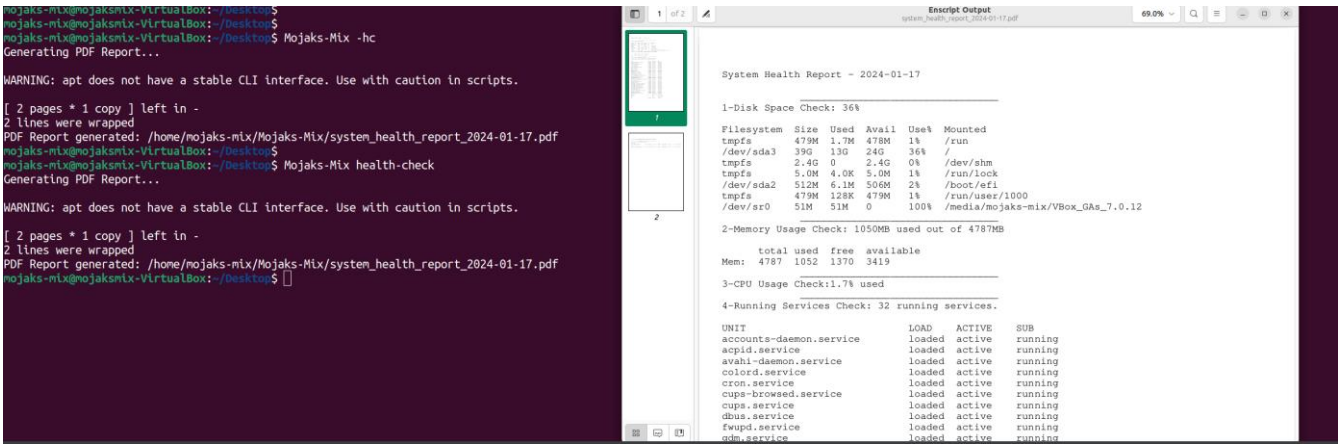

Figure 2: Health Check Functionality

1.3 Backup

The "back-up.sh" script facilitates the backup functionality within the Mojaks-Mix tool. It checks the size of the source file or directory, ensuring it is not greater than 1 GB. The script allows users to specify the source path and an optional backup directory. If no backup directory is provided, it defaults to "Mojaks_Back_Up/default_backup_directory." The script creates a tar archive of the source directory or compresses a file, depending on the user's specifications. Additionally, the script logs each backup action, including the source path and destination, in the "backup_log.txt" file. The script handles errors gracefully, notifying users of any issues encountered during the backup process.

Figure 3: Backup Functionality

1.4 Display Logs

The "display-logs.sh" script is designed to retrieve and display logs generated by the Mojaks-Mix tool. It takes a single argument, either "-back" or "-health," to determine which log file to access. If the argument is "-back," it displays the content of the "backup_log.txt" file, which records backup actions, including source paths and destinations. If the argument is "-health," it displays the content of the "health_check_log.txt" file, containing records of system health check actions. The script handles errors gracefully, notifying users if an invalid argument is provided, and exits with an appropriate message.

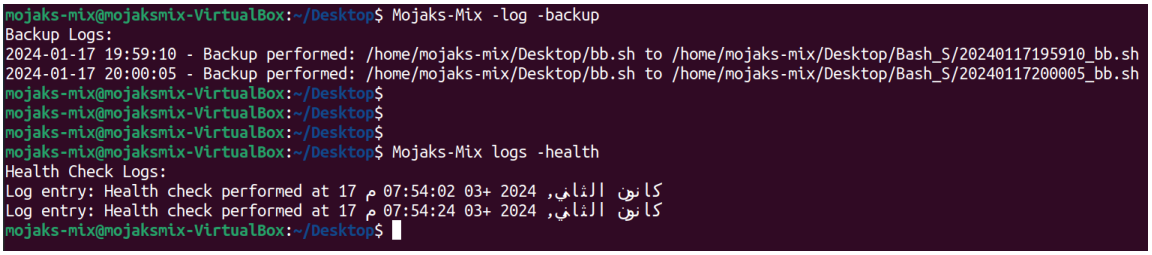

Figure 4: Display Log Files

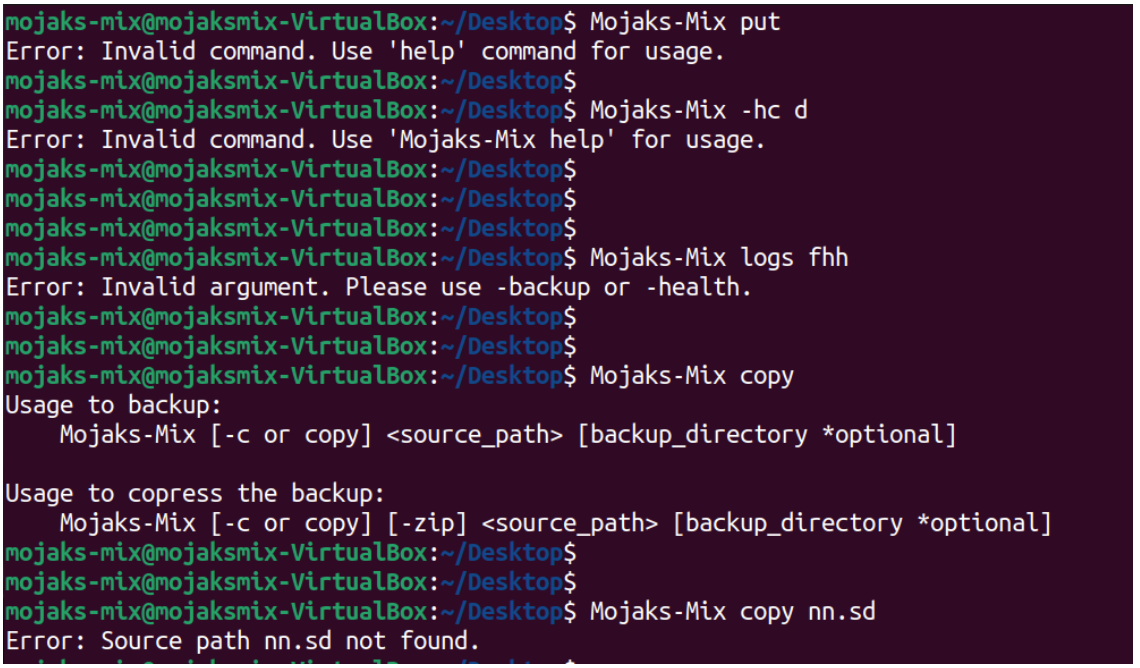

Figure 5: Possible Error Handling

Table 1: Tool Structure and Dynamics

Action	Purpose	Command Structure	Error Handling
Controller	• Controlling and redirecting user input to the proper action. • Separate the files and their logic achieving cleaner code structure.	**Flags to decide action :** • -h or help: Display help message for command usage. • -hc or health-check: System Health Check. • -c or copy: Create a backup copy of a folder or file. • -log or logs: Display the content of the log files.	• Invalid command, prompting help. • No command provided, prompting help.
Health Check	Execute health checks and generate a pdf report.	**No argument should be passed**	• Package manager log file not found. • High memory or CPU usage warnings. • Errors during the PDF report generation or opening, prompting: “install enscript!”
Backup	Create a backup of a file or directory which can be copressed.	• -c <source_path> [backup_dest] • -c **-zip** <source_path> [backup_dest]	• Source path or destination not found. • Size of the source path exceeds 1 GB. • Issues during tar archive creation or compression. • Errors during log entry creation or log file write.
Display Logs	Display the content of log files.	• -back: backup logs. • -health: health check logs.	• Log file not found or inaccessible. • Invalid argument, prompting help.

CHAPTER TWO: Insights the Performance Analysis

To insuring the performance I rely on built-in commands that already serve the system and the structure of the tool is what I call the AC "Action Controller", so that one file will be called or control what file or action will be done, sothe files structure will follow the AC principle that is explained in the tool structure, so all the project will be in the Mojaks-Mix folder.

The main bash file called "Controller" that will have a match statement that will differentiate between the actions of the commands and if the command is not there it will tell the user that there is no command and tell him to type "mojaks-mix help" that will print the flags and the allowed command and their purpose. so that command structure will be processed here if valid then call the file of the action to do the action else the help functionality will be called.

In conclusion, the performance is optimized mainly by using proper logic, use of built-in commands, and the AC principle that will reduce the effort of execution because what is used is called, so no loading for unused scripts.

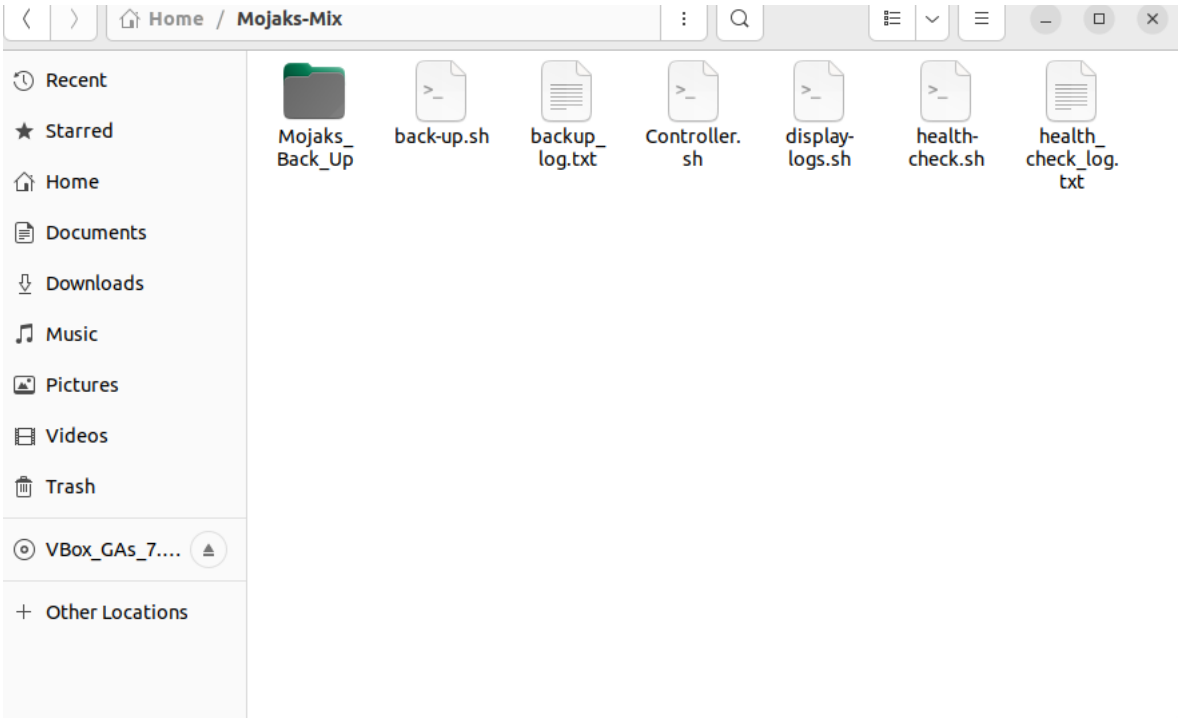

Figure 6: Tool Structure

2.1 Controller

Built-in Commands Used:

- awk, column, dirname, realpath, source, tar

2.2 Health Check

Built-in Commands Used:

- awk, column, date, df, echo, enscript, free, grep, mkdir, ps2pdf, realpath, systemctl, top, xdg-open

2.3 Backup

Built-in Commands Used:

- awk, cp, date, du, echo, gzip, mkdir, realpath, tar

2.4 Display Logs

Built-in Commands Used:

- cat, echo

CHAPTER THREE: Optimizations

3.1 Controller Script

Optimizations:

Modular Structure: The Controller script adopts a modular structure, separating functionality into distinct functions. This promotes code organization, readability, and maintainability. Each function handles a specific action, facilitating easy updates or additions without affecting other parts of the script.

Centralized Constants: The use of a constant for the script's path (SCRIPT_DIR) ensures consistency and facilitates maintenance. It allows for quick adjustments if the script is relocated, enhancing code robustness.

Error Handling: The script incorporates a centralized error handling function (handle_error()), reducing redundancy and promoting a consistent approach to error messages. This enhances the script's reliability and makes it easier to maintain.

Impact:

The modular structure and centralized constants contribute to improved readability and scalability. Maintenance becomes more straightforward as each function encapsulates a specific action, making it easier to troubleshoot or enhance individual features. Error handling consistency enhances the robustness of the script, providing clear and standardized error messages.

3.2 Health Check Script

Optimizations:

Modular Health Checks: The Health Check script modularizes different health checks, organizing them into separate functions (check_disk_space(), check_memory_usage(), etc.). This approach simplifies maintenance, allowing easy addition or modification of specific health checks without affecting the entire script.

Dynamic PDF Generation: The script dynamically generates a PDF report using enscript and ps2pdf. This ensures that the system health report is consistently formatted and easily shareable. The script also utilizes xdg-open to automatically open the generated PDF, enhancing user convenience.

Impact:

The modular health checks enhance the script's readability and maintainability. Future updates or additions to health checks can be seamlessly integrated. Dynamic PDF generation ensures standardized, shareable health reports, while automatic opening enhances the user experience.

3.3 Backup Script

Optimizations:

Size Checking Function: The script includes a dedicated function (check_size()) to check the size of files or directories, preventing backups of items exceeding 1 GB. This size constraint ensures the script's efficiency and prevents potential issues with large backups.

Default Directory Handling: The script intelligently handles default backup directories, creating them if they don't exist. This minimizes user intervention and enhances the script's usability.

Impact:

The size checking function prevents potential performance issues by restricting backups of excessively large files or directories. The handling of default backup directories improves user experience and reduces the likelihood of errors.

3.4 Display Logs Script

Optimizations:

Centralized Argument Handling: The script centralizes argument handling using a case statement, ensuring clarity and maintainability. It simplifies the addition of new log types in the future.

Impact:

The centralized argument handling improves script readability and allows for easy expansion of log types without significant modifications. This optimization supports long-term maintainability and adaptability.