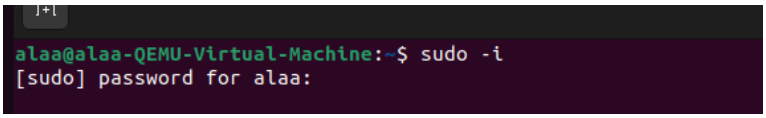
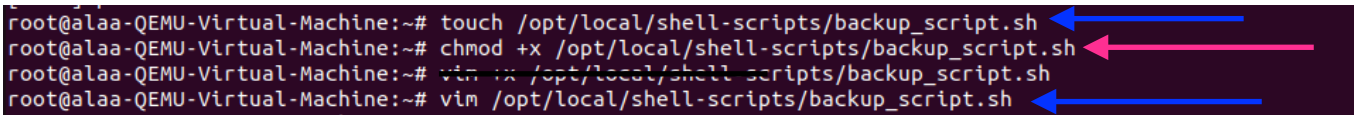
(Backup.sh) script file.

Switch to Root User:



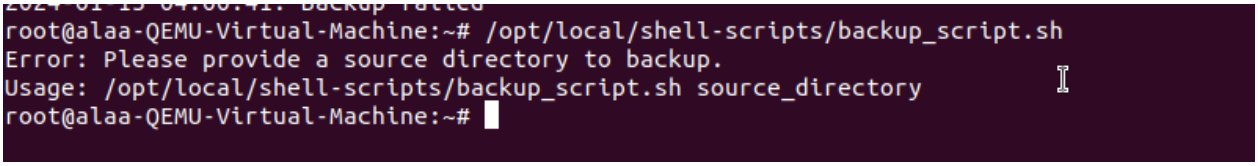
1) Create and Edit Backup Script:

reated an empty backup script file at /opt/local/shell-scripts/backup_script.sh and edited it using Vim.
make the file excutable (change mode)



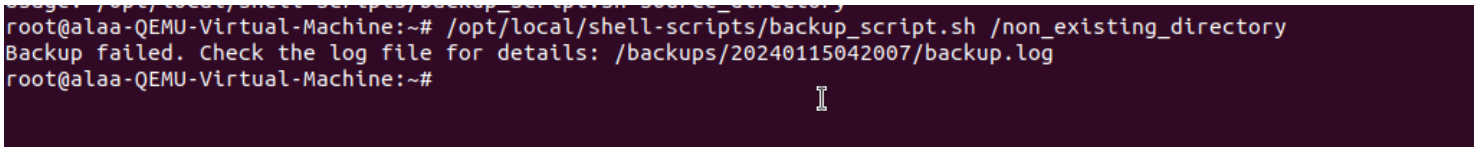
2) Run Backup Script Without a Source Directory:

display an error message prompting you to provide a source directory.



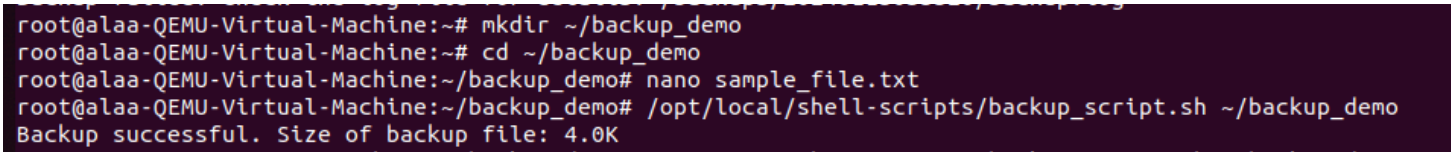
3) Run Backup Script with a Non-Existing Directory

result in a backup failure, and you can check the log for details.

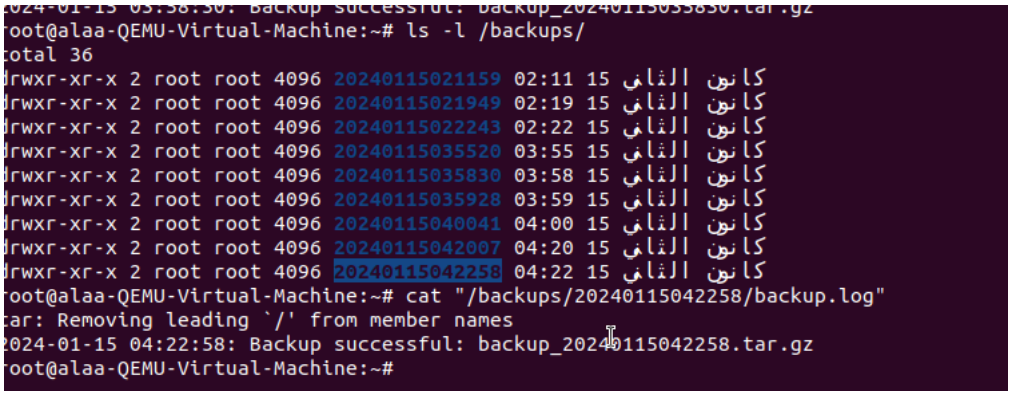


4) Create a Source Directory and Run Backup Script:

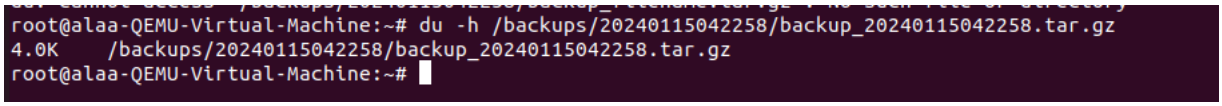
* created a source directory (`~/backup_demo`), added a file (`sample_file.txt`), and successfully ran the backup script.
* Ensure the script runs successfully and reports the size of the backup file.



5) Check Log File and Backup Size



This indicates that the backup file has been successfully created and has a size of 4.0 kilobytes.



the directory `/backups/` contains backups from different timestamps. The backup log indicates a successful backup at `2024-01-15 04:22:58` & the associated backup file is `backup_20240115042258.tar.gz`. The du command is used to check the disk usage of the backup file, showing a size of 4.0K.

(Backup.sh) script file.

The purpose and functionality:
This script is designed to create backups of specified source directories using the `tar` command in a compressed (`targz`) format. And the purpose is to provide a simple and automated way to back up data and to ensure that each backup is timestamped & logged for tracking purposes.

Variables:
`BACKUP_DST`: The destination directory where backups will be stored
`BACKUP_DATE`: A timestamp generated using the `date` command (`%Y%m%d%H%M%S`)
`BACKUP_FILENAME`: The name of the backup file, which includes the timestamp (`backup_20240115042007.tar.gz`)

Usage Function:
`display_usage()`: A function to display the correct usage of the script when the user provides incorrect.

Argument Check:
this script checks if a source directory is provided as an argument.
If not, it displays an error message and the correct usage.

Backup Directory Creation:
Use to create a backup directory with the timestamp under the specified destination directory

Log Function:
`log_message()`: used to log messages with timestamps into a `backup`.

Backup Process:
The source directory specified as an argument is archived using `tar` creating a compressed tarball (`tar.gz`) , the archive is stored in the backup directory with the designated filename.

Verification and Output:

- The script checks the exit status of the `tar` command to determine if the backup was successful. If successful it logs a success message with the filename and displays information about the size of the backup file. If unsuccessful, it logs a failure message and advises the user to check the log file for details.

Insights from the performance analysis:

The script works well by using the "tar" command to pack and save the chosen source folder. It also includes timestamps, which are like time stamps on files, to help keep everything organized and easy to track. And The script includes error handling to notify the user of any issues during the backup process.

Optimizations and Impact:

- **Timestamping**: The use of it in the backup directory names and filenames ensures uniqueness.
- **Destination Directory Creation**:The script creates the destination directory (`/backups`) if it doesn't exist.
- **Logging**: Logging messages with timestamps provides a detailed record of each backup's if it’s success or not :)

Impacts:

- Troubleshooting:
  - Detailed error messages in the log file assist users in identifying and addressing issues during the backup process.
- Usability:
  - Users can easily back up specific directories by providing the source directory as an argument.
  - Error messages and log files aid users in understanding the outcome of the backup process.
- Organizational:
  - The timestamped directories and filenames contribute to an organized and easily navigable backup structure.

System Health Check Script:

The script designed for checking and reporting various aspects of system health.
It consists of several functions, each responsible to gather specific information (disk space, memory usage, running services, and recent system updates)
Its a system to generat a comprehensive health report.

Disk Space info

- Purpose: Displays the current disk space usage on the system.
- Functionality: Utilizes the `df-h` command to show the disk space details.

- ### Memory Usage info
  - Purpose: Shows the current memory usage statistics.
  - Functionality: Employs the `free-h` command to present human-readable memory details.

- ### Running Services
- Purpose: Lists the services that are currently running on the system.
  - Functionality: Uses `systemctl` to list active running services.

- ### Check System Updates
- Purpose: Provides information about recent system updates.
    - **Functionality:**
    - 1) Updates the package lists using `sudo apt update.`
    - 2) Lists upgradable packages using `sudo apt list --upgradable.`

- ### Display Overall System Health
  - Purpose: Integrates all the previously defined functions to present a comprehensive system health report.
  - Functionality: Calls each function to collect specific information.

- ### Run the System Health Check
  - Purpose: Initiates the overall system health check.

Impact of Optimizations:

- Easier to Read:
  - By using functions and organizing information, the script becomes easier to understand and work with.
- Less Messy Output:
  - Sending unnecessary updates to a virtual trash can (/dev/null) cleans up the report, making it more focused and helpful.
- Building Blocks:
  - Breaking down the script into smaller functions makes it simpler to take care of and can be used again in different scripts."
  -

Insights and Optimizations:

The script uses basic Linux commands like df, free, systemctl, and apt to collect information about the system.

It creates an easy-to-read report that highlights important details about the system’s.performance.

Improvements:

To make the script better, we can add more advanced and more error handling and reporting features.
Breaking down the code into functions makes it easier to understand and organize (readable).
By sending the output of 'apt update' to /dev/null,
we can prevent unnecessary information from cluttering the report, making it cleaner."