Shell Scripting

Assignment 4

Name: Mohammad Al-Qaisy

Date: 4th Jul 2023

Table of Contents

Introduction

The shell script presented in this report aims to provide a comprehensive analysis of files within a specified directory and its subdirectories. The script incorporates user-friendly features and customization options to enhance the usability and effectiveness of the analysis.

Upon executing the script, the user is prompted to provide a directory path as an argument. This allows for flexibility in selecting the desired directory for analysis. The script utilizes a command-line interface (CLI) with clear prompts and descriptive messages, ensuring ease of use and guiding the user throughout the process.

One of the key features of the script is the ability to search for files with a specific extension. By specifying the desired file extension, the user can narrow down the analysis to only include files of interest, such as ".txt" files or any other desired extension.

The script can generate a comprehensive report that includes vital details about each file found in the specified directory and its subdirectories. This information encompasses the size, owner, permissions, and last modified timestamp of each file. By consolidating these details into a single report, users can easily gain insights into their file system and track relevant information.

To facilitate better organization and readability of the report, the script groups the files based on their respective owners. This grouping enables users to quickly identify files associated with specific owners and better understand the distribution of file ownership within the analyzed directory.

Furthermore, the script provides the option to sort the file groups based on the total size occupied by each owner. Sorting the file groups allows users to identify owners with the largest file storage footprint, aiding in optimizing storage allocation and resource management.

Finally, to ensure easy access and future reference, the script saves the generated report in a file named "file_analysis.txt". This file serves as a valuable resource for users to review and analyze the collected information at their convenience.

In addition to the above features, the shell script incorporates robust error handling mechanisms. It displays informative error messages and suggests potential solutions to assist users in addressing any issues that may arise during the execution of the script. Additionally, the script validates user inputs to provide appropriate feedback for invalid or missing arguments, promoting a smooth and error-free experience.

In conclusion, the shell script presented in this report offers a user-friendly and customizable solution for analyzing files within a directory. By leveraging its features, users can gain valuable insights into their file system, optimize resource allocation, and effectively manage their data.

Task Analysis

The assigned task involves designing a shell script that performs a series of tasks to provide a comprehensive analysis of files within a specified directory and its subdirectories. Let's analyze each task in detail:

Task A: Accepts a directory path as an argument.

This task requires the shell script to accept a directory path as an argument when it is executed. The directory path serves as the starting point for the file analysis.

Task B: Searches for all files with a specific extension (e.g., .txt) in the given directory and its subdirectories.

The script needs to recursively search the specified directory and its subdirectories to identify files that match a specific extension provided by the user. This task involves efficiently traversing the directory tree and filtering files based on the given file extension.

Task C: Generates a comprehensive report that includes file details such as size, owner, permissions, and last modified timestamp.

For each file identified in Task B, the script must gather relevant details such as file size, owner, permissions, and last modified timestamp. These details need to be collected and organized to generate a comprehensive report that provides insights into the analyzed files.

Task D: Groups the files by owner.

After gathering the file details, the script should group the files based on their respective owners. This grouping enhances the organization and readability of the report, allowing users to easily identify files associated with specific owners.

Task E: Sorts the file groups by the total size occupied by each owner.

To provide further analysis, the script should sort the file groups obtained in Task D based on the total size occupied by each owner. This sorting feature enables users to identify owners with the largest file storage footprint and understand the distribution of file sizes among different owners.

Task F: Saves the report in a file named "file_analysis.txt".

Once the file analysis is complete in Task C, the script should save the generated report in a file named "file_analysis.txt". Saving the report ensures easy access and future reference, allowing users to review the analyzed file details at their convenience.

User-Friendly Features:

The shell script incorporates various user-friendly features to enhance the overall user experience:

- Clear prompts and descriptive messages in the command-line interface (CLI) guide users through the execution process, providing guidance and clarity.
- A dedicated help section provides instructions on script usage, explaining how to execute the script and listing available options, ensuring users can leverage the script's capabilities effectively.
- Informative error messages are displayed in case of any issues encountered during script execution. These messages not only alert users about errors but also offer potential solutions to address them by using the help command to see what the available options are.
- The script validates user inputs to ensure they are in the expected format and provides appropriate feedback for invalid or missing arguments.

By carefully analyzing the assigned tasks and considering the incorporation of user-friendly features, we can design a shell script that meets the requirements, provides a seamless file analysis experience, and empowers users to gain valuable insights into their file system.

Available Options

The shell script offers several options that users can utilize to customize their analysis and obtain specific information. Below are the available options along with their descriptions:

1. **‘ls’**:

   Lists all files in the specified directory and its subdirectories. This option provides a comprehensive view of all files present in the directory tree.

   ‘[File extension]’:

   An optional parameter that allows users to lists files with a specific extension (e.g., .txt) in the specified directory and its subdirectories. By providing a file extension as an argument, users can filter the displayed files and focus on a particular type of file.

2. **‘as’**:

   Filters files based on size, permissions, or last modified timestamp. This option allows users to narrow down the analysis based on specific criteria, such as displaying files of a certain size range or files modified within a specific timeframe.

3. **‘rp’**:
   Generates a comprehensive report of file details on the terminal. This option extracts essential information about each file, such as size, owner, permissions, and last modified timestamp. The report is displayed directly on the terminal for easy viewing and analysis.

   ‘-s’ or **‘--save’**:

   An optional parameter that allows users to save the generated report in a file named "file_analysis.txt". This option ensures that the report can be accessed and reviewed at a later time.

4. **‘gp’**:
   Groups files by owner, providing an overview of ownership distribution. This option organizes the files based on their respective owners, enabling users to identify the owners associated with specific files easily.

   ‘-s’, **‘--sort’**:

   An optional parameter that sorts the file groups by the total size occupied by each owner. This sorting feature assists users in identifying owners with the largest file storage footprint.

5. **‘summary’** or **‘summ’**:
   Provides a summary of the directory contents and ownership details. This option offers a concise overview of the analyzed directory, including the total number of files, the number of files per owner, and other relevant ownership details.

6. **‘help’**:

   Displays a help section that provides instructions on how to use the script and lists the available options. This command serves as a quick reference guide for users who need assistance while using the shell script.

These available options empower users to customize their analysis, filter files based on specific criteria, generate comprehensive reports, and gain valuable insights into their file system and ownership distribution.

Functions

A key aspect of developing a shell script is the implementation of functions that promote modularity, extensibility, and maintainability. By designing functions with a single responsibility, you have created a framework that simplifies the addition and modification of features. This approach ensures that each function focuses on a specific task, enhancing code readability and reducing complexity.

These functions play a crucial role in organizing and structuring the shell script, making it easier to understand, maintain, and expand. With a clear separation of responsibilities, modifying existing features or introducing new ones becomes a straightforward process. By adhering to the principle of single responsibility, each function can be modified or extended independently, without affecting other parts of the script.

The implementation of these functions not only improves the flexibility and scalability of the shell script but also enables code reuse. With well-defined functions in place, developers can easily extract and reuse sections of code for similar tasks, saving time and effort in the development process.

The utilization of these functions demonstrates a commitment to developing a robust and adaptable shell script. It sets a foundation that fosters easy feature addition, efficient code maintenance, and encourages future enhancements to meet evolving requirements.

Functions:

1. **advance_search**

Description:

The advance_search function is responsible for listing files based on specific criteria such as size, permissions, or last modified timestamp. It allows users to narrow down their file analysis by applying filters and displaying files that meet the specified criteria.

Functionality:

1. Accepts arguments for file extension, minimum size, maximum size, permissions, and timestamp.
2. Constructs a find command based on the provided criteria.
3. Utilizes conditional statements to append filter options to the find command.
4. Evaluates and executes the constructed find command using the eval function.
5. Displays the list of files that match the specified criteria.

Usage:

advance_search $file_extension $min_size $max_size $permissions $timestamp

Parameters:

1. file_extension: The desired file extension for filtering files. If provided, only files with this extension will be listed.
2. min_size: The minimum size (in bytes) for filtering files. If provided, only files larger than this size will be listed.
3. max_size: The maximum size (in bytes) for filtering files. If provided, only files smaller than this size will be listed.
4. permissions: The desired permissions for filtering files. If provided, only files with these permissions will be listed.
5. timestamp: The maximum number of minutes since the last modified timestamp for filtering files. If provided, only files modified within this timeframe will be listed.

Example:

advance_search "txt" 100 2000 775 1654661192

2. report

Description:

The report() function generates a comprehensive report of file details, including filename, owner, permissions, last modified timestamp, and size. It iterates over the files within the specified directory and its subdirectories, collects the required information using the stat command, and formats the details into a report format. Users have the option to save the report in a file named "file_analysis.txt".

Functionality:

1. Initializes the report_details variable to store the formatted file details.
2. Accepts an optional argument on_file to determine whether to save the report in a file or display it on the terminal.
3. Uses the find command to retrieve a list of files within the specified directory and its subdirectories.
4. Iterates over each file and extracts the filename, owner, permissions, last modified timestamp, and size using the stat command.
5. Formats the file details into a tabular format and appends them to the report_details variable.
6. If the on_file argument is provided as "-s" or "--save", the function saves the report in a file named "file_analysis.txt" by overwriting the file if it already exists.

7. If the on_file argument is empty or not provided, the function displays the report on the terminal.
8. If an invalid or unsupported option is provided as on_file, the function displays an error message.

Usage:

report $on_file

Parameters:

on_file: An optional argument to determine whether to save the report in a file or display it on the terminal. If provided as "-s" or "--save", the report will be saved in a file. If not provided or any other value is given, the report will be displayed on the terminal.

Example:

report "-s"

In this example, the report() function will generate a comprehensive report of file details and save it in a file named "file_analysis.txt". The report will include information such as filename, owner, permissions, last modified timestamp, and size for each file.

By utilizing the report function, users can easily obtain a detailed analysis of file attributes, facilitating further examination and decision-making based on file details.

3. owner_groups

Description:

The owner_groups() function is responsible for grouping files by owner and providing an overview of ownership distribution. It collects file details, such as owner, filename, and size, and organizes them into groups based on the owner. The function allows users to choose whether to sort the file groups by the total size occupied by each owner.

Functionality:

1. Initializes associative arrays groups and group_sizes to store file groups and their respective sizes.
2. Accepts an optional argument sorted to determine whether the file groups should be sorted by size.

3. Uses the find command to retrieve a list of files within the specified directory and its subdirectories.
4. Iterates over each file and extracts the owner, filename, and size using the stat command.
5. Populates the groups and group_sizes arrays with the file details, organizing them by owner.
6. If the sorted argument is provided as "-s" or "--sort", the function sorts the file groups based on the total size occupied by each owner.
7. Displays the grouped files, owner information, and optionally the total size occupied by each owner.
8. If an invalid or unsupported option is provided as sorted, the function displays an error message.

Usage:

owner_groups $sorted

Parameters:

sorted: An optional argument to indicate whether the file groups should be sorted by size. If provided as "-s" or "--sort", the groups will be sorted. If not provided or any other value is given, the groups will be displayed without sorting.

Example:

owner_groups "-s"

In this example, the owner_groups function will group the files by owner and sort the groups based on the total size occupied by each owner. It will display the owner's name, the total size occupied by the owner, and the list of filenames associated with each owner.

By utilizing the owner_groups function, users can obtain insights into ownership distribution and easily identify owners with the largest file storage footprint, facilitating further analysis and decision-making based on ownership information.describe.

4. describe

Description:

The describe function provides a summary of the directory contents and ownership details. It counts the total number of files, calculates the total size of files, determines the number of unique file groups, and identifies the number of unique owners within the specified directory and its subdirectories.

Functionality:

1. Initializes variables to keep track of file count, total size, group count, and owner count.
2. Uses the find command to retrieve a list of files within the specified directory and its subdirectories.
3. Iterates over each file and performs the following actions:
4. Increments the file count by one.
5. Retrieves the size of the file using the stat command and adds it to the total size.
6. Retrieves the group of the file using the stat command and checks if it is already recorded in the groups array. If not, increment the group count and adds the group to the groups array.
7. Retrieves the owner of the file using the stat command and checks if it is already recorded in the owners array. If not, increment the owner count and adds the owner to the owners array.
8. Displays the summary information including file count, total size, group count, and owner count.

Usage:

describe

Example:

describe

In this example, the describe function will provide a summary of the directory contents and ownership details. It will display the total number of files, the total size of files in bytes, the number of unique file groups, and the number of unique owners.

By utilizing the describe function, users can obtain a concise overview of the directory contents and ownership distribution, providing valuable insights for further analysis and decision-making based on the summary information.

5. help

Description:

The help() function displays instructions on how to use the shell script and provides a list of available options along with their descriptions. It serves as a quick reference guide for users, enabling them to understand the script's functionality and usage.

Functionality:

1. Prints usage instructions and options for the shell script.
2. Provides a clear and concise description for each option.

3. Includes any additional parameters or arguments associated with specific options.
4. Presents the information in a readable format for easy understanding.

Usage:

help

Example:

help

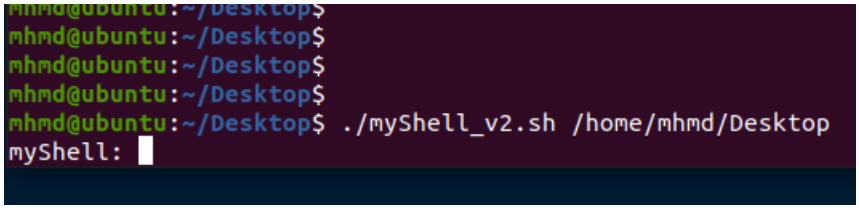
In this example, the help function will display the usage instructions and a list of available options along with their descriptions. It will provide users with the necessary information to utilize the shell script effectively.

By using the help function, users can quickly access the instructions and descriptions of available options, ensuring they have the necessary guidance to interact with the shell script and make informed choices.

Manual Testing

Implementation Part

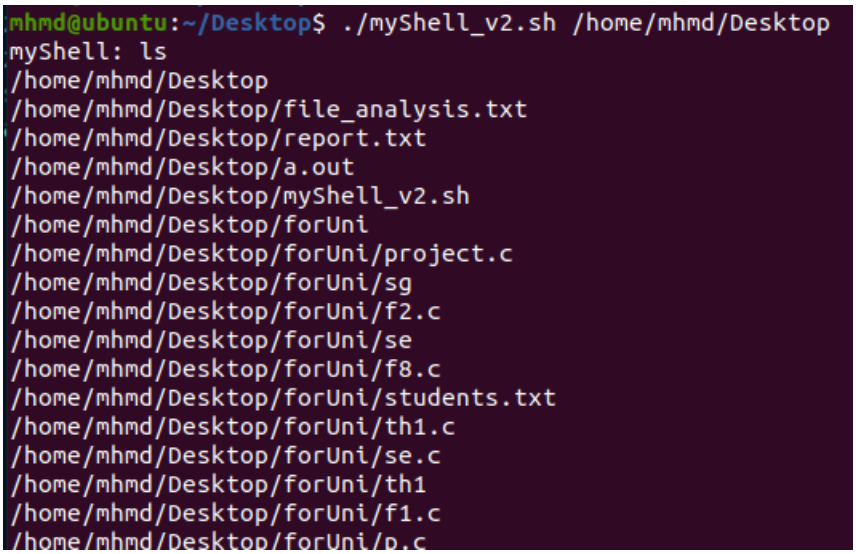
Task A: Accepts a directory path as an argument.



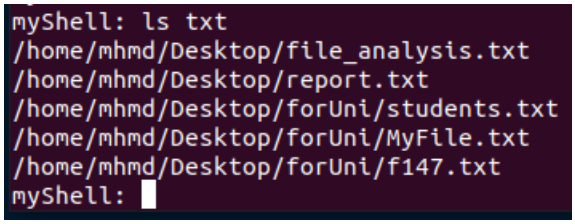
Result: Pass

Task B: Searches for all files with a specific extension (e.g., .txt) in the given directory and its subdirectories.

ls which will listed all files



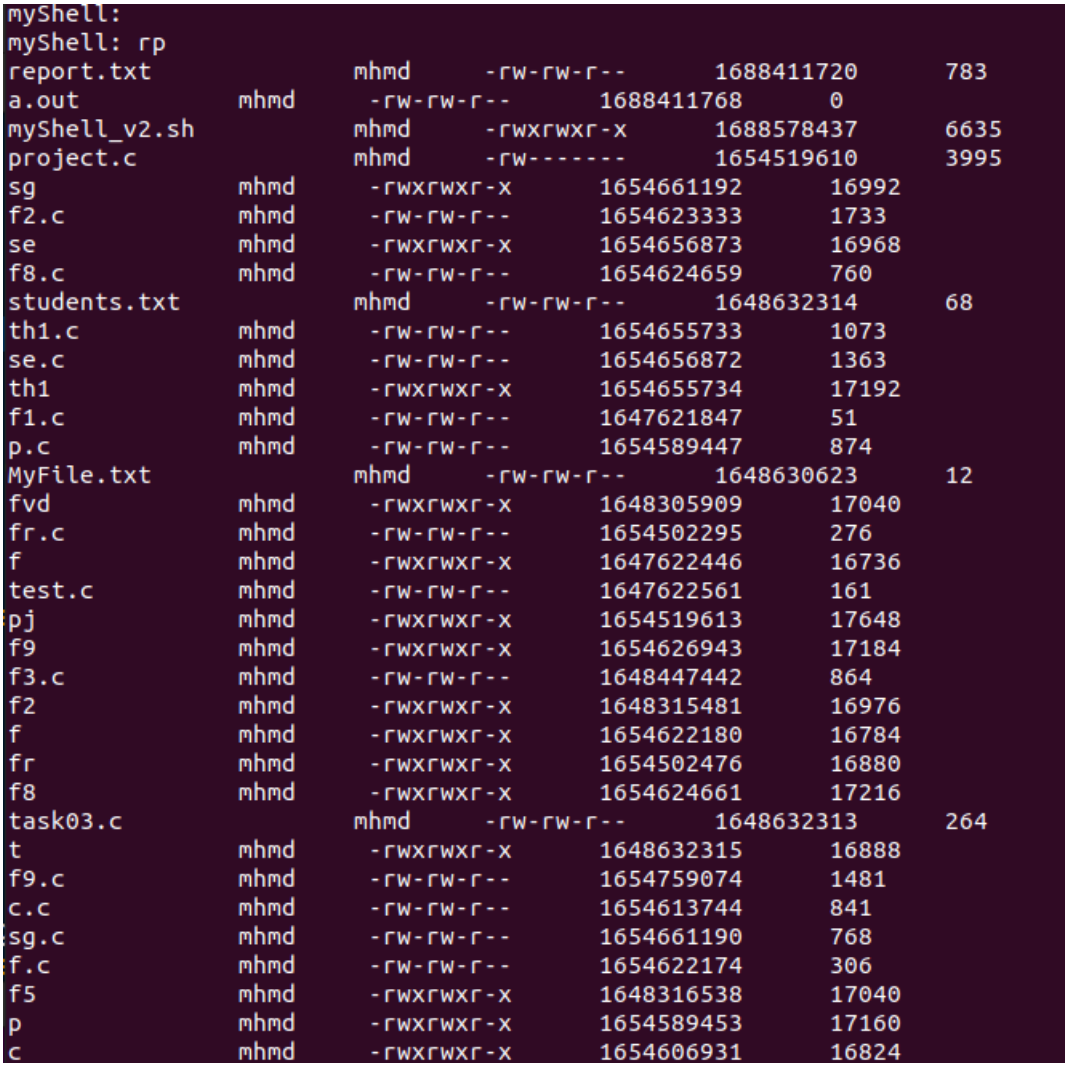
ls [File extension]



Listed all files with a specific extension

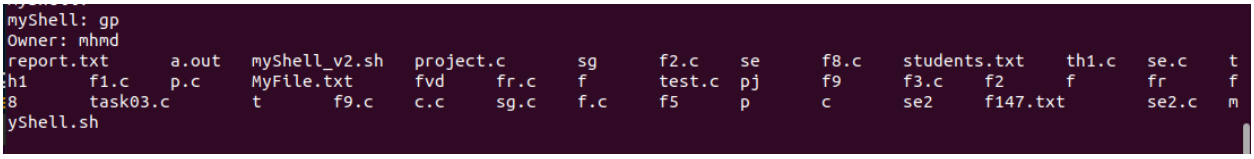
Result: Pass

Task C: Generates a comprehensive report that includes file details such as size, owner, permissions, and last modified timestamp.



Result: Pass

Task D: Groups the files by owner.



Now, for test will try Groups the files by group:

```
myShell:
myShell: gp
Owner: lxd
f.c

Owner: sambashare
sg

Owner: mhmd
report.txt      a.out   myShell_v2.sh   project.c       f2.c    se      f8.c    th1.c   se.c    th1     f1.c    p.c     M
yFile.txt       fvd     fr.c    f       test.c  pj      f9      f3.c    f2      f       fr      f8      task03.c        t
f9.c    sg.c    f5      p       c       se2     f147.txt        se2.c   myShell.sh

Owner: sudo
students.txt

Owner: adm
c.c

myShell:
```

Result: Pass

Task E: Sorts the file groups by the total size occupied by each owner.

```
myShell: gp -s
Owner: mhmd
Size: 301257
report.txt      a.out   myShell_v2.sh   project.c       sg      f2.c    se      f8.c    students.txt    th1.c   se.c    t
h1      f1.c    p.c     MyFile.txt      fvd     fr.c    f       test.c  pj      f9      f3.c    f2      f       fr      f
8       task03.c        t       f9.c    c.c     sg.c    f.c     f5      p       c       se2     f147.txt        se2.c   m
yShell.sh

myShell:
```

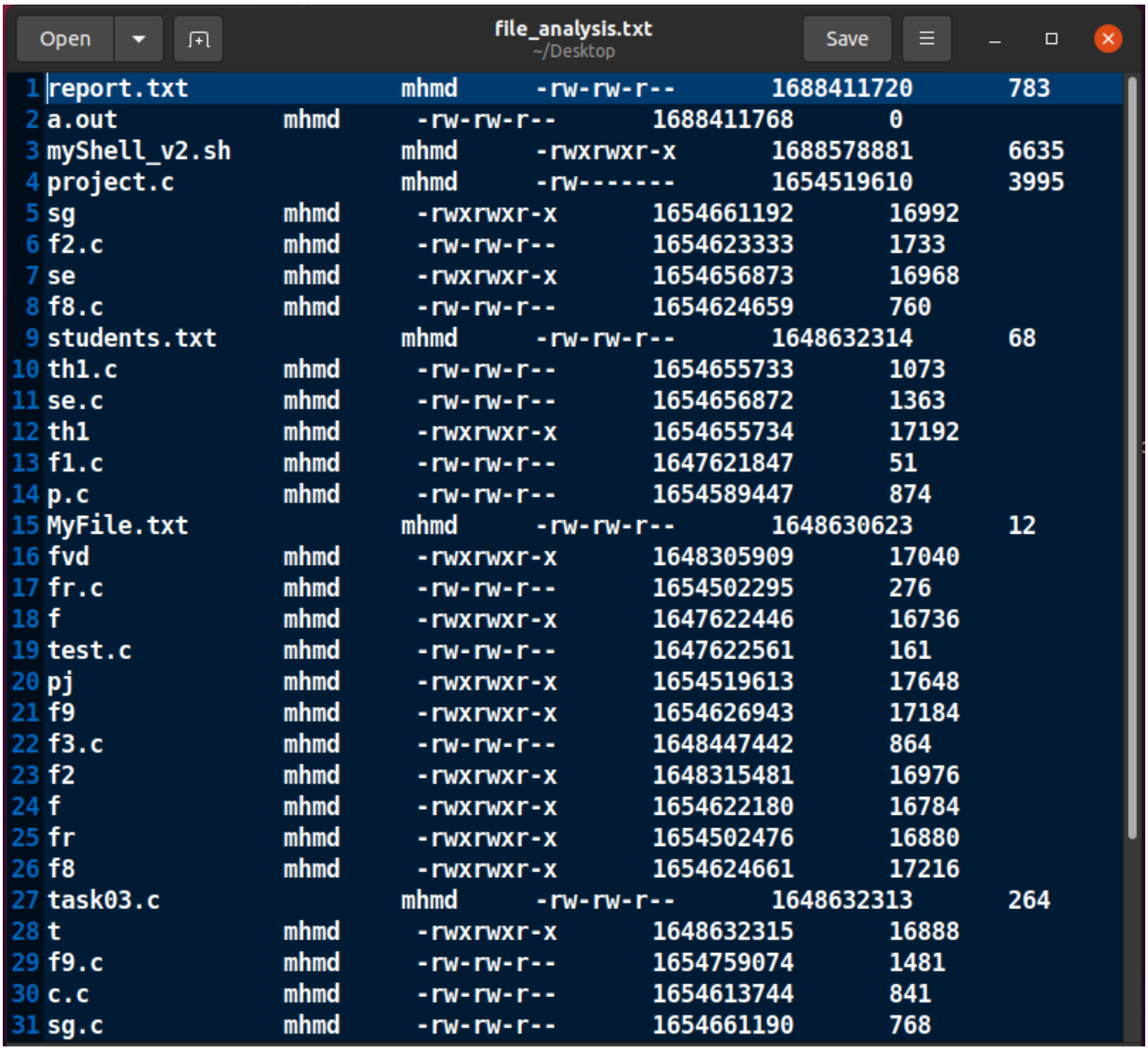
Now, for test will try Groups the files by group:

```
myShell: gp -s
Owner: mhmd
Size: 283050
report.txt      a.out   myShell_v2.sh   project.c       f2.c    se      f8.c    th1.c   se.c    th1     f1.c    p.c     M
yFile.txt       fvd     fr.c    f       test.c  pj      f9      f3.c    f2      f       fr      f8      task03.c        t
f9.c    sg.c    f5      p       c       se2     f147.txt        se2.c   myShell.sh

Owner: sambashare
Size: 16992
sg

Owner: adm
Size: 841
c.c

Owner: lxd
Size: 306
f.c

Owner: sudo
Size: 68
students.txt

myShell:
```

Result: Pass

Task F: Saves the report in a file named "file_analysis.txt".

```
myShell: rp -s
done
myShell:
```



Result: Pass

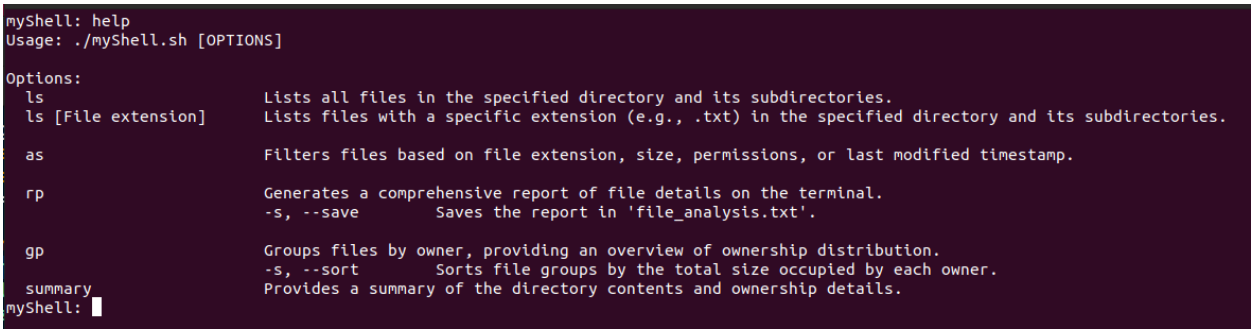
User-friendly Features Part:

Task A: CLI with clear prompts and descriptive messages.

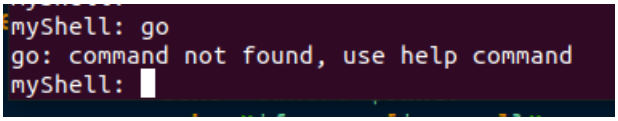
```
myShell: cc
cc: command not found, use help command
myShell:
```

```
myShell:
myShell:
myShell:
myShell:
myShell:
myShell: clear
```

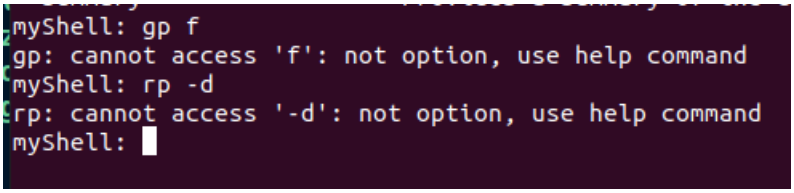
Task B: Provide a help section.



Task C: Handle errors by displaying informative error messages.

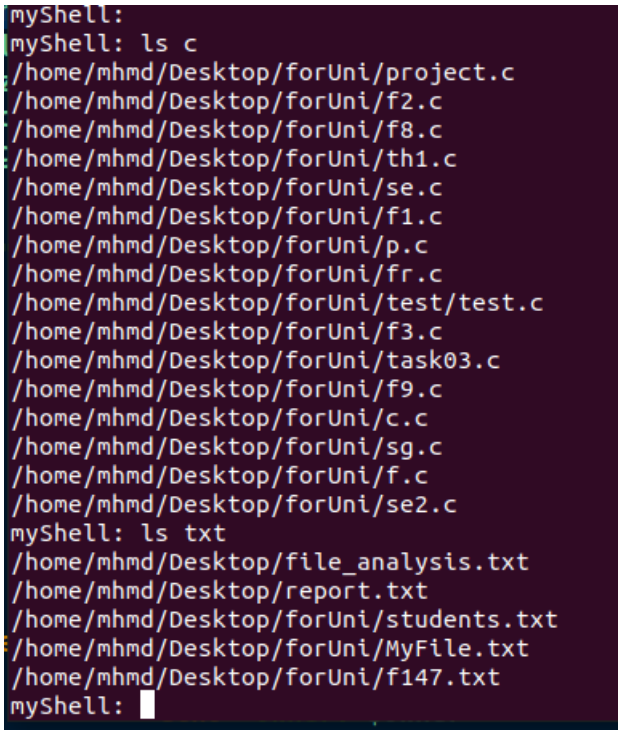


Task D: Validate user inputs and provide appropriate feedback for invalid or missing arguments.



Advanced Feature Integration Part

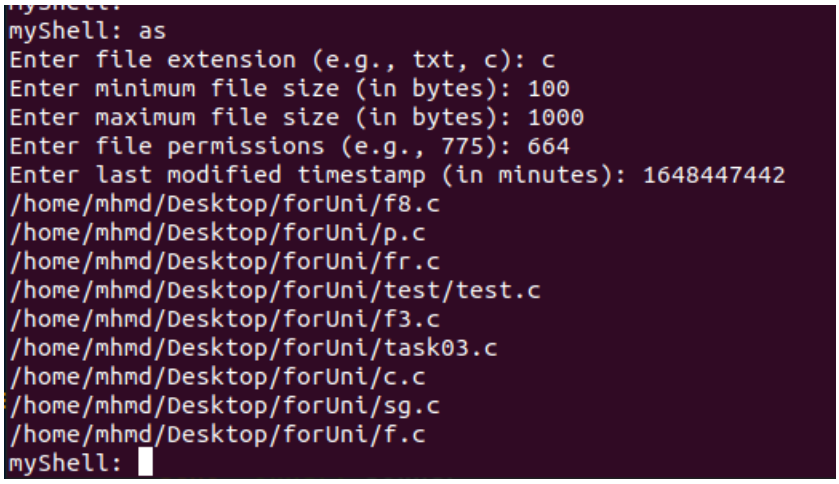
Task A: Support for multiple file extensions.



Can support any extensions.

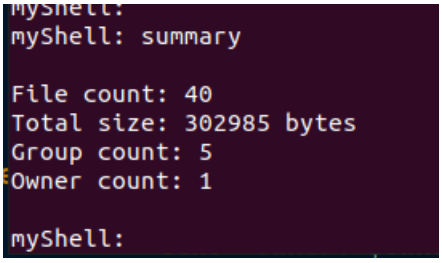
Result: Pass

Task B: Filter files based on size, permissions, or last modified timestamp, allowing users to customize their search criteria.



Result: Pass

Task C: generate a summary report that displays total file count, total size, and other relevant statistics.



Result: Pass

Summary

The shell script is designed to accomplish various tasks related to file analysis. It accepts a directory path as an argument, searches for files with a specific extension, generates a comprehensive report of file details, groups files by owner, sorts file groups by size, and saves the report in a file named "file_analysis.txt". The script also incorporates user-friendly features, including a command-line interface with clear prompts, a help section explaining usage and available options, informative error handling, and validation of user inputs.

In terms of functions, the script includes the following:

1. list_files: Lists all files in the specified directory and its subdirectories.
2. advance_search: Filters files based on size, permissions, or last modified timestamp.
3. owner_groups: Groups files by owner, providing an overview of ownership distribution.
4. report: Generates a comprehensive report of file details.
5. describe: Provides a summary of the directory contents and ownership details.
6. help: Displays instructions and descriptions of available options.

The implementation of these functions ensures modularity and maintainability, allowing for easy addition or modification of features. Each function has a clear responsibility, promoting code readability and facilitating future enhancements. Overall, the shell script offers a powerful and user-friendly solution for analyzing files and extracting meaningful insights from the file system.