ATYPON

Shell Scripting Assignment

Made by: Samer AlHamdan

Instructors:
Motasim Aldiab
Fahed Jubair

1. Introduction **3**

2. The Assignment **3**

3. Design Choices **4**

3.1 Script Overview 4

3.2 Command-Line Usage 4

3,3 Functions 5

3.4 Error Handling 8

3.5 Temporary File 8

3.6 Report Generation 8

3.7 Modularity and Readability 9

3.8 Flexibility with Filters 10

4. Optimizations **10**

5. Advanced Features **11**

6. User Friendliness **14**

7. Reflections **15**

7.1 Lessons Learned 16

7.2 Difficulties Encountered 16

7.3 Potential Future Improvements 17

1. Introduction

This report offers a comprehensive analysis and optimization strategy for a feature-rich shell script designed to efficiently perform complex tasks during Atypon's Training in May 2023. The script focuses on utilizing Linux commands, shell scripting techniques, and user-friendly features to enhance the overall experience. The assignment consists of three key parts: implementation, optimization, and advanced feature integration.

2. The Assignment

The purpose of this project is to develop a script that generates a report on files based on a given path and file type(s). The script aims to provide valuable insights into the files present in the specified directory by collecting and presenting information such as file size, owner, permissions, and last modified date.

The scope of the project encompasses the design and implementation of a robust and efficient script capable of handling large volumes of files while ensuring accurate and comprehensive reporting. The script offers flexibility by allowing users to specify the path and file type(s) of interest, enabling focused analysis on specific file categories.

The main functionality of the script involves traversing the directory structure, identifying files matching the specified file type(s), and collecting relevant information about each file. The collected data is then compiled into a report that presents a detailed overview of the files, enabling users to gain insights into their characteristics and attributes.

The objectives of the script include:

- Providing a clear and organized report on file information, facilitating analysis and decision-making.
- Optimizing performance by implementing efficient file traversal and data collection techniques.
- Incorporating user-friendly features to enhance the overall user experience.
- Allowing customization through filters and advanced options for tailored reporting.
- Enabling extensibility and future improvements by following modular and well-structured design principles.

By achieving these objectives, the script aims to provide users with a valuable tool for understanding and analyzing files in a given directory.

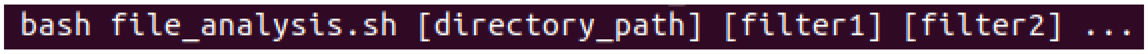
3. Design Choices

3.1 Script Overview

- The script is designed to analyze files in a given directory and its subdirectories.
- It generates a detailed report with file details such as size, owner, permissions, and last modified timestamp.
- The report is saved in a file named `file_analysis.txt`.

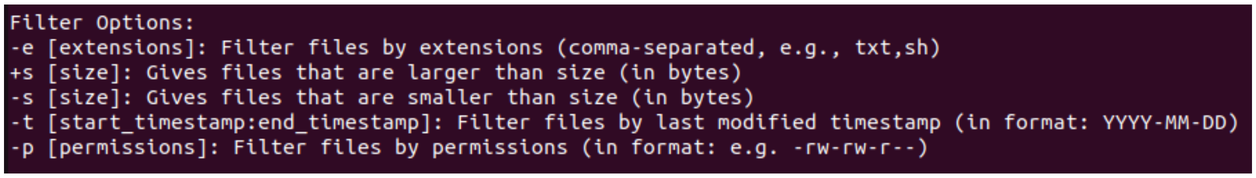
3.2 Command-Line Usage

- The script expects the directory path as the first argument and optional filters as subsequent arguments.



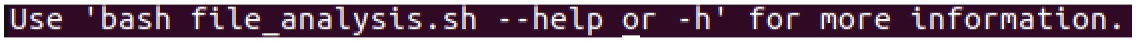
```
bash file_analysis.sh [directory_path] [filter1] [filter2] ...
```

- The filters allow users to narrow down the analysis based on file extensions, size, last modified timestamp, and permissions.



```
Filter Options:
-e [extensions]: Filter files by extensions (comma-separated, e.g., txt,sh)
+s [size]: Gives files that are larger than size (in bytes)
-s [size]: Gives files that are smaller than size (in bytes)
-t [start_timestamp:end_timestamp]: Filter files by last modified timestamp (in format: YYYY-MM-DD)
-p [permissions]: Filter files by permissions (in format: e.g. -rw-rw-r--)
```

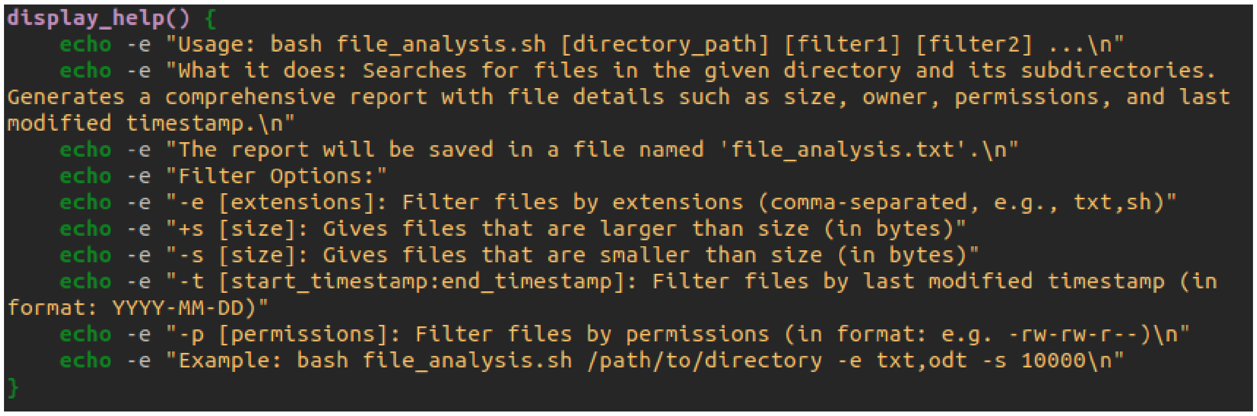
- Users can view the usage instructions by running the script with the `--help` or `-h` argument.



```
Use 'bash file_analysis.sh --help or -h' for more information.
```

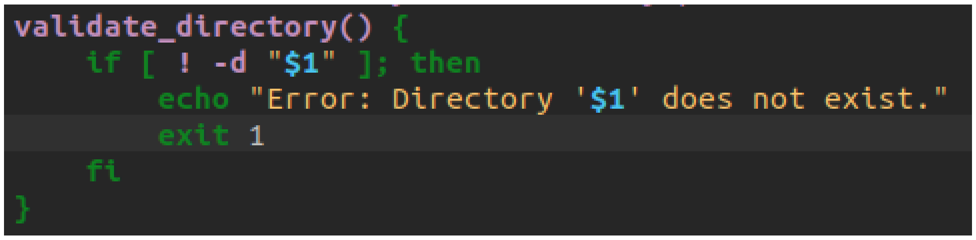
3,3 Functions

- display_help() : Displays the usage instructions and available filter options to the user.



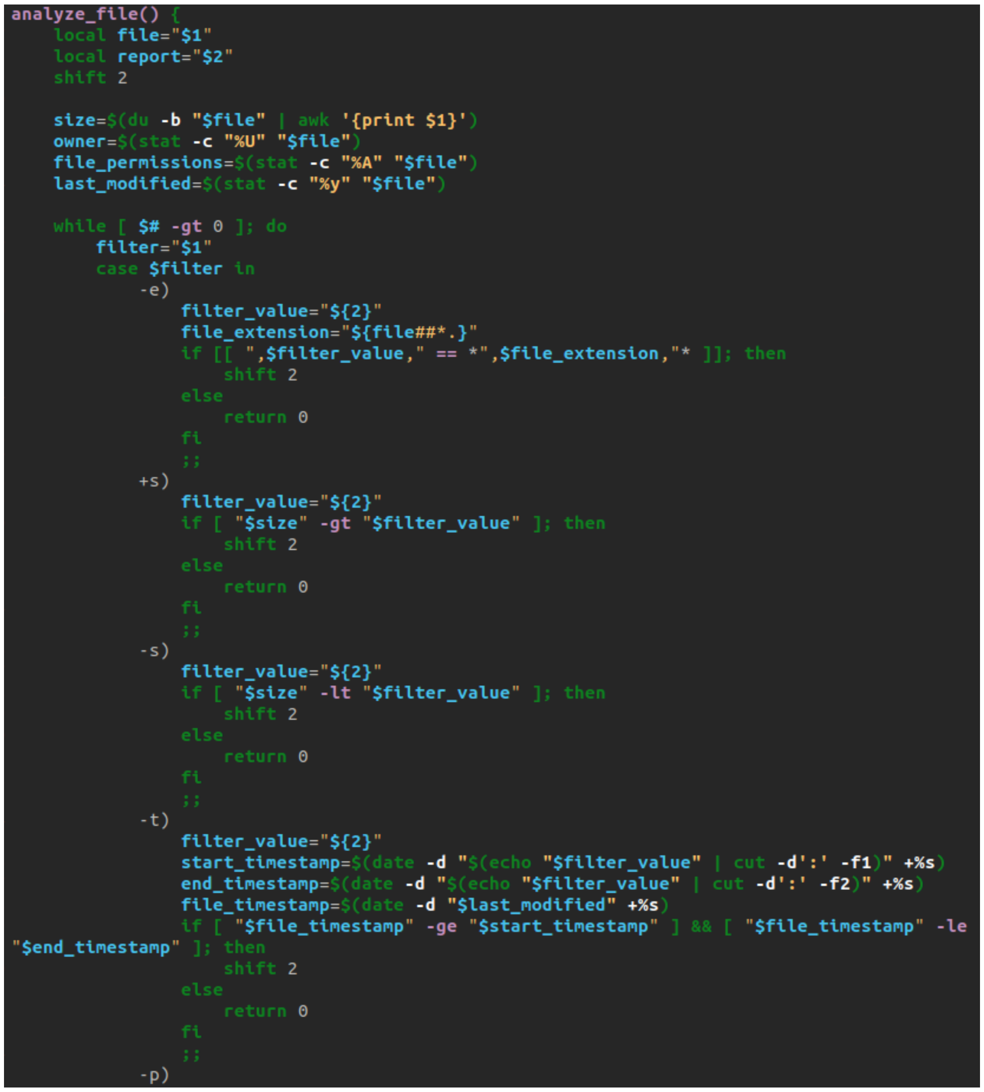
```
display_help() {
    echo -e "Usage: bash file_analysis.sh [directory_path] [filter1] [filter2] ...\n"
    echo -e "What it does: Searches for files in the given directory and its subdirectories. 
Generates a comprehensive report with file details such as size, owner, permissions, and last 
modified timestamp.\n"
    echo -e "The report will be saved in a file named 'file_analysis.txt'.\n"
    echo -e "Filter Options:"
    echo -e "-e [extensions]: Filter files by extensions (comma-separated, e.g., txt,sh)"
    echo -e "+s [size]: Gives files that are larger than size (in bytes)"
    echo -e "-s [size]: Gives files that are smaller than size (in bytes)"
    echo -e "-t [start_timestamp:end_timestamp]: Filter files by last modified timestamp (in 
format: YYYY-MM-DD)"
    echo -e "-p [permissions]: Filter files by permissions (in format: e.g. -rw-rw-r--)\n"
    echo -e "Example: bash file_analysis.sh /path/to/directory -e txt,odt -s 10000\n"
}
```

- validate_directory() : Checks if the specified directory path is valid. If the directory doesn't exist, an error message is displayed, and the script terminates.

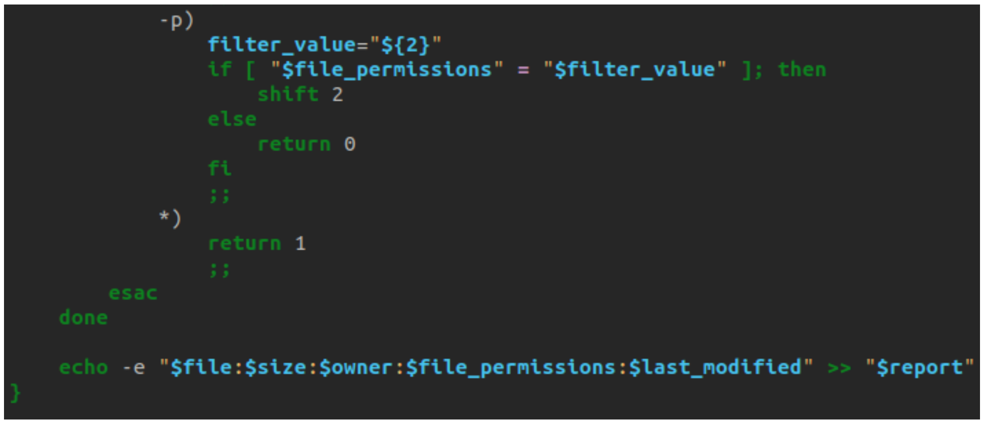


```
validate_directory() {
    if [ ! -d "$1" ]; then
        echo "Error: Directory '$1' does not exist."
        exit 1
    fi
}
```

- `analyze_file()`: Analyzes a file by retrieving its details (size, owner, permissions, last modified timestamp) and applying the specified filters. If the file matches all the filters, its details are added to the report.

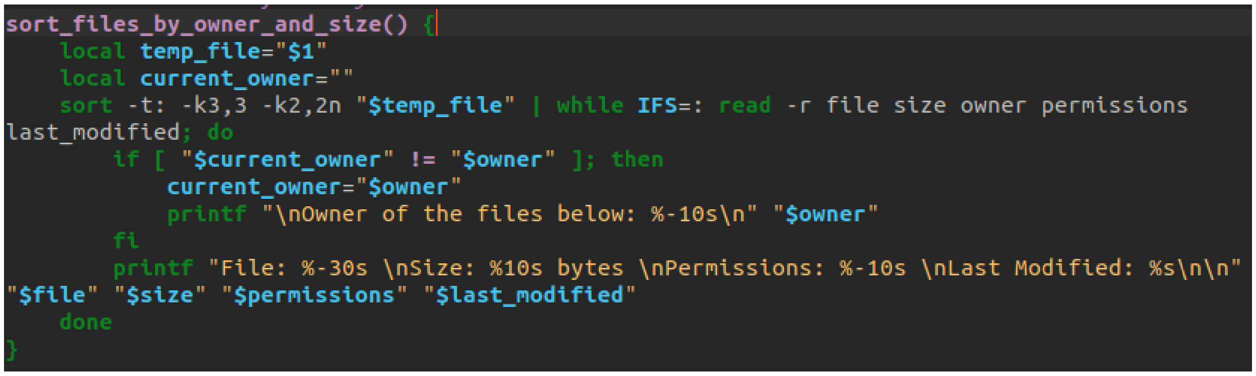


```
analyze_file() {
    local file="$1"
    local report="$2"
    shift 2

    size=$(du -b "$file" | awk '{print $1}')
    owner=$(stat -c "%U" "$file")
    file_permissions=$(stat -c "%A" "$file")
    last_modified=$(stat -c "%y" "$file")

    while [ $# -gt 0 ]; do
        filter="$1"
        case $filter in
            -e)
                filter_value="${2}"
                file_extension="${file##*.}"
                if [[ ",$filter_value," == *",$file_extension,"* ]]; then
                    shift 2
                else
                    return 0
                fi
                ;;
            +s)
                filter_value="${2}"
                if [ "$size" -gt "$filter_value" ]; then
                    shift 2
                else
                    return 0
                fi
                ;;
            -s)
                filter_value="${2}"
                if [ "$size" -lt "$filter_value" ]; then
                    shift 2
                else
                    return 0
                fi
                ;;
            -t)
                filter_value="${2}"
                start_timestamp=$(date -d "$(echo "$filter_value" | cut -d':' -f1)" +%s)
                end_timestamp=$(date -d "$(echo "$filter_value" | cut -d':' -f2)" +%s)
                file_timestamp=$(date -d "$last_modified" +%s)
                if [ "$file_timestamp" -ge "$start_timestamp" ] && [ "$file_timestamp" -le
"$end_timestamp" ]; then
                    shift 2
                else
                    return 0
                fi
                ;;
            -p)
```



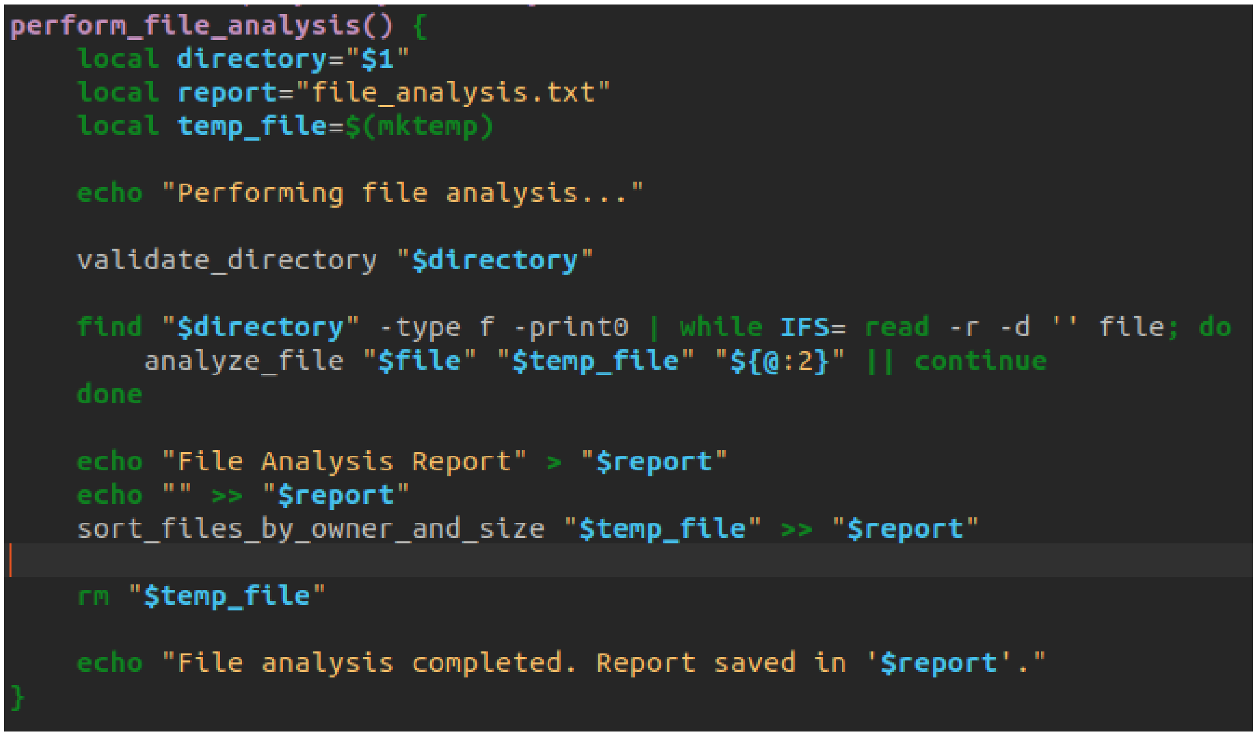
```
            -p)
                filter_value="${2}"
                if [ "$file_permissions" = "$filter_value" ]; then
                    shift 2
                else
                    return 0
                fi
                ;;
            *)
                return 1
                ;;
        esac
    done

    echo -e "$file:$size:$owner:$file_permissions:$last_modified" >> "$report"
}
```

- `sort_files_by_owner_and_size()`: Sorts the files in the report by owner and size. It groups the files by owner and displays the file details (name, size, permissions, last modified timestamp) in a sorted manner.



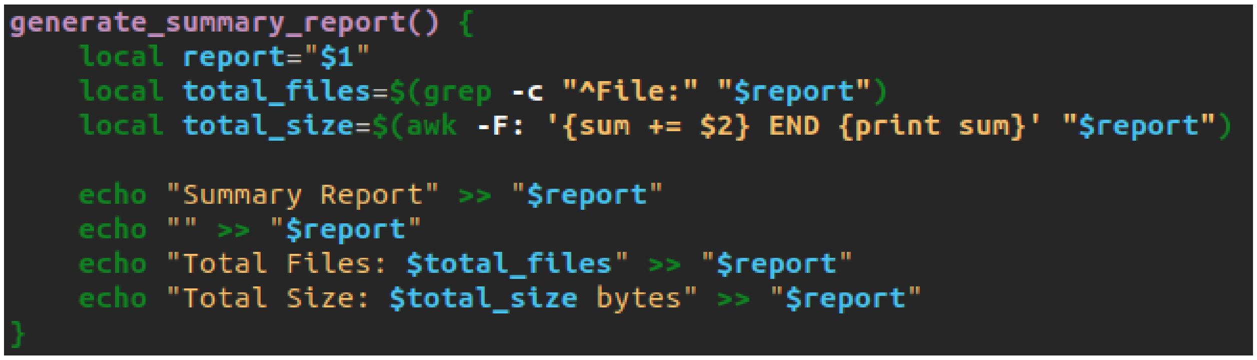
```
sort_files_by_owner_and_size() {
    local temp_file="$1"
    local current_owner=""
    sort -t: -k3,3 -k2,2n "$temp_file" | while IFS=: read -r file size owner permissions
last_modified; do
        if [ "$current_owner" != "$owner" ]; then
            current_owner="$owner"
            printf "\nOwner of the files below: %-10s\n" "$owner"
        fi
        printf "File: %-30s \nSize: %10s bytes \nPermissions: %-10s \nLast Modified: %s\n\n"
"$file" "$size" "$permissions" "$last_modified"
    done
}
```

- `perform_file_analysis()`: Executes the file analysis process. It validates the directory path, finds files in the directory and its subdirectories, analyzes each file, and generates a detailed report.



```
perform_file_analysis() {
    local directory="$1"
    local report="file_analysis.txt"
    local temp_file=$(mktemp)

    echo "Performing file analysis..."

    validate_directory "$directory"

    find "$directory" -type f -print0 | while IFS= read -r -d '' file; do
        analyze_file "$file" "$temp_file" "${@:2}" || continue
    done

    echo "File Analysis Report" > "$report"
    echo "" >> "$report"
    sort_files_by_owner_and_size "$temp_file" >> "$report"

    rm "$temp_file"

    echo "File analysis completed. Report saved in '$report'."
}
```

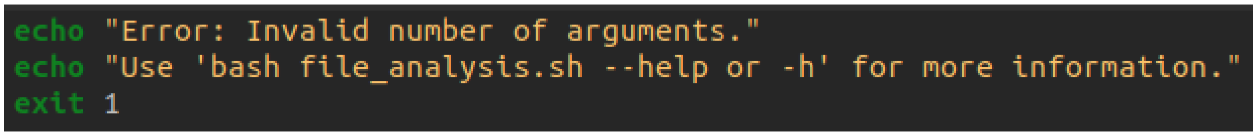
- `generate_summary_report()`: Appends a summary report to the detailed report. The summary includes the total number of files analyzed and their cumulative size.



```
generate_summary_report() {
    local report="$1"
    local total_files=$(grep -c "^File:" "$report")
    local total_size=$(awk -F: '{sum += $2} END {print sum}' "$report")

    echo "Summary Report" >> "$report"
    echo "" >> "$report"
    echo "Total Files: $total_files" >> "$report"
    echo "Total Size: $total_size bytes" >> "$report"
}
```

3.4 Error Handling

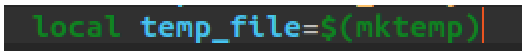
- The script checks the number of arguments provided and displays an error message if the count is incorrect. It prompts users to refer to the help information for correct usage.



```
echo "Error: Invalid number of arguments."
echo "Use 'bash file_analysis.sh --help or -h' for more information."
exit 1
```

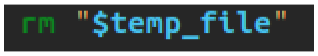
3.5 Temporary File

- A temporary file (`temp_file`) is created using the `mktemp` command to store the file details during the analysis process.



```
  local temp_file=$(mktemp)
```

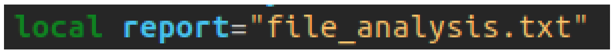
- Once the analysis is completed and the report is generated, the temporary file is removed.



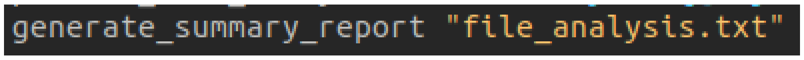
```
rm "$temp_file"
```

3.6 Report Generation

- The script generates a detailed report (`file_analysis.txt`) containing the file details for the analyzed files.

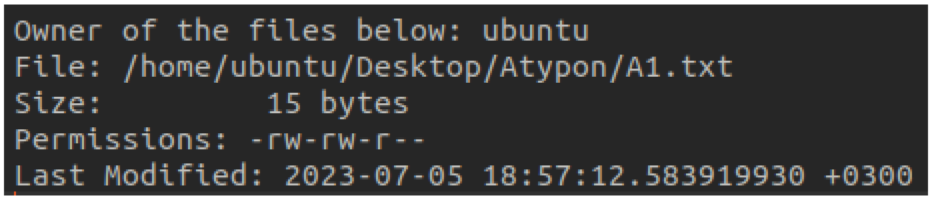


```
local report="file_analysis.txt"
```



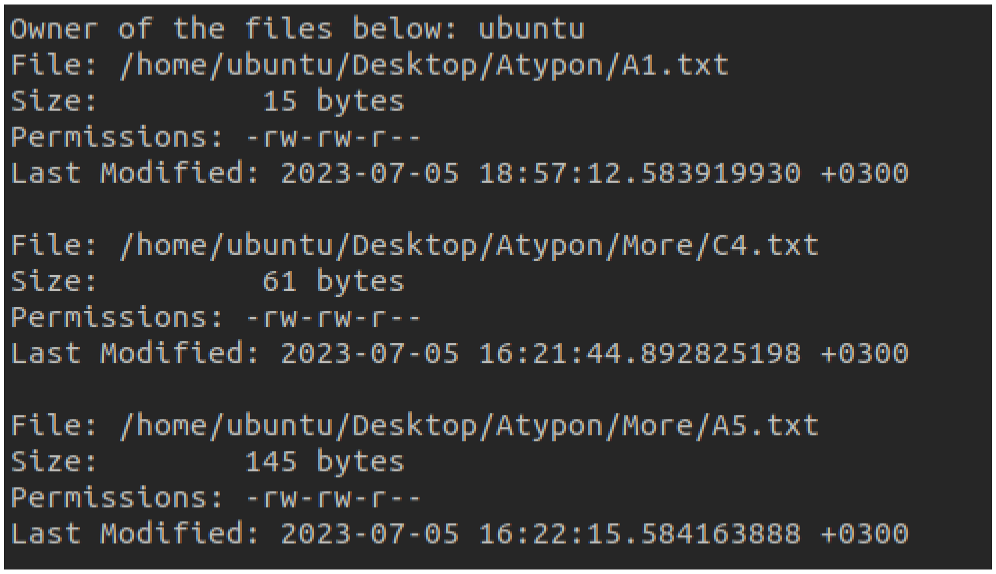
```
generate_summary_report "file_analysis.txt"
```

- The report includes the file name, size, owner, permissions, and last modified timestamp.



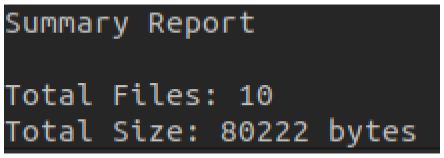
```
Owner of the files below: ubuntu
File: /home/ubuntu/Desktop/Atypon/A1.txt
Size:          15 bytes
Permissions: -rw-rw-r--
Last Modified: 2023-07-05 18:57:12.583919930 +0300
```

- The file entries in the report are sorted by owner and size using the `sort_files_by_owner_and_size()` function.



```
Owner of the files below: ubuntu
File: /home/ubuntu/Desktop/Atypon/A1.txt
Size:          15 bytes
Permissions: -rw-rw-r--
Last Modified: 2023-07-05 18:57:12.583919930 +0300

File: /home/ubuntu/Desktop/Atypon/More/C4.txt
Size:          61 bytes
Permissions: -rw-rw-r--
Last Modified: 2023-07-05 16:21:44.892825198 +0300

File: /home/ubuntu/Desktop/Atypon/More/A5.txt
Size:         145 bytes
Permissions: -rw-rw-r--
Last Modified: 2023-07-05 16:22:15.584163888 +0300
```

- A summary report is also appended to the detailed report, displaying the total number of files and their cumulative size.



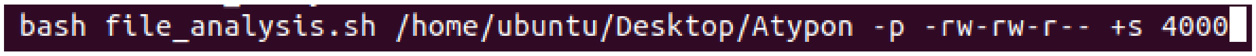
```
Summary Report

Total Files: 10
Total Size: 80222 bytes
```

3.7 Modularity and Readability

- The script is divided into multiple functions, each with a specific responsibility. This improves code modularity and readability.
- Each function performs a single task, such as displaying help information, validating the directory, analyzing files, sorting files, performing analysis, or generating reports.

3.8 Flexibility with Filters

- The script allows users to specify filters to customize the file analysis.
- Filters can be used to filter files based on extensions, size, last modified timestamp, and permissions.
- Multiple filters can be combined to refine the analysis further.



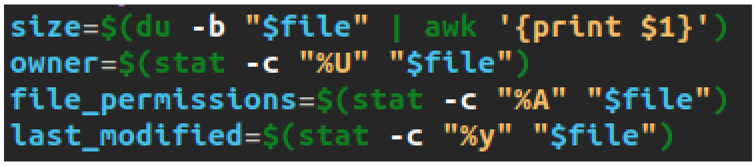
```
bash file_analysis.sh /home/ubuntu/Desktop/Atypon -p -rw-rw-r-- +s 4000
```

4. Optimizations

These optimization techniques aim to improve the performance and efficiency of the script, reducing resource usage and providing faster results during file analysis.

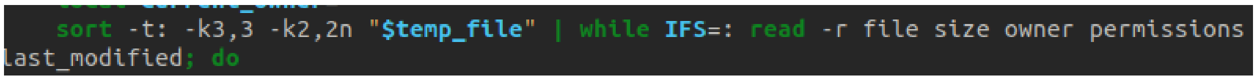
here are a few optimization techniques employed:

1. Early Filtering: The script incorporates filtering techniques in `analyze_file()` to exclude files early in the analysis process. As files are encountered, they are checked against the specified filters. If a file doesn't match a filter, it is skipped, reducing the processing time for unnecessary files.
2. Efficient File Details Retrieval: The `analyze_file()` function retrieves file details (size, owner, permissions, last modified timestamp) using the `du` and `stat` commands. By using these efficient system commands, the script minimizes the resource usage and performs the analysis swiftly.



```
size=$(du -b "$file" | awk '{print $1}')
owner=$(stat -c "%U" "$file")
file_permissions=$(stat -c "%A" "$file")
last_modified=$(stat -c "%y" "$file")
```

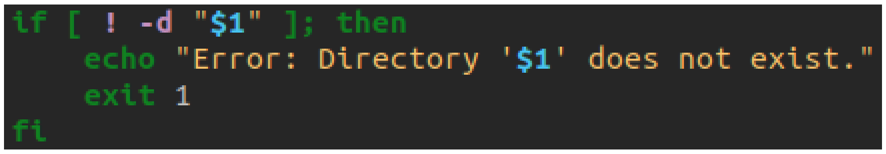
3. Temporary File Usage: During file analysis, the script stores the file details in a temporary file (`temp_file`). This approach avoids repeatedly opening and closing the output file while processing each file. It reduces I/O operations and enhances performance.
4. Sorting Optimization: The `sort_files_by_owner_and_size()` function utilizes the `sort` command to sort the files in the report. By specifying the sorting fields and delimiter, the sorting operation is optimized for efficiency, allowing for faster processing of the report.



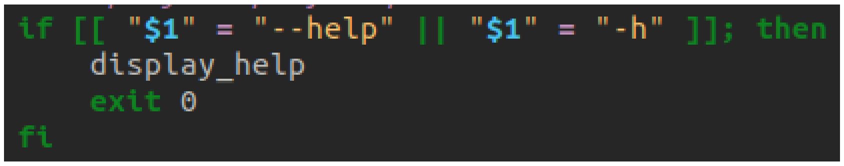
```
    sort -t: -k3,3 -k2,2n "$temp_file" | while IFS=: read -r file size owner permissions
last_modified; do
```

5. Early Exit: The script employs early exit strategies in several places. For example, when the provided directory path is invalid, an error message is displayed, and the script exits immediately. Similarly, when a file fails to match a filter, the script exits

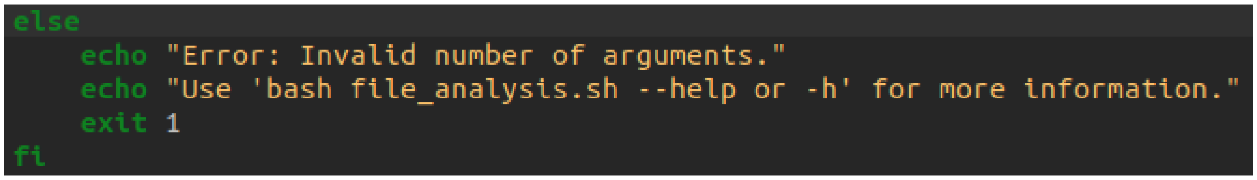
the current loop iteration, avoiding unnecessary processing.



```
if [ ! -d "$1" ]; then
    echo "Error: Directory '$1' does not exist."
    exit 1
fi
```

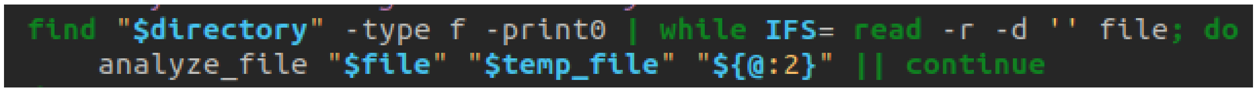


```
if [[ "$1" = "--help" || "$1" = "-h" ]]; then
    display_help
    exit 0
fi
```



```
else
    echo "Error: Invalid number of arguments."
    echo "Use 'bash file_analysis.sh --help or -h' for more information."
    exit 1
fi
```

6. Limited File System Access: The script uses the `find` command with the `-type f` option to search for files specifically, excluding directories and other non-file entities. This targeted approach ensures that the analysis is focused solely on files, minimizing unnecessary operations on directories.



```
 find "$directory" -type f -print0 | while IFS= read -r -d '' file; do
     analyze_file "$file" "$temp_file" "${@:2}" || continue
```

7. Streamlined Reporting: The script generates two reports: a detailed file analysis report and a summary report. By appending the summary to the same file, rather than creating a separate file, the script avoids additional file operations and keeps the reporting process efficient.

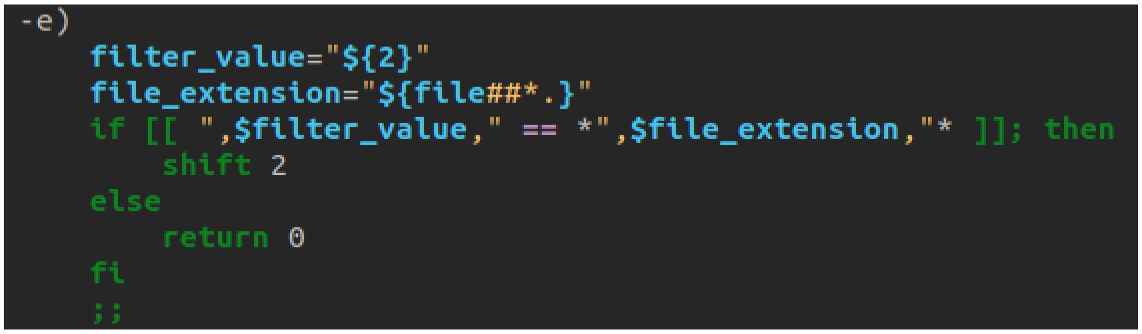
5. Advanced Features

The script incorporates advanced features to enhance the file analysis process. These features include the ability to apply filters and generate summary reports, providing users with greater control and a more concise overview of the analysis results.

- Filters: One advanced feature is the availability of filters, enabling users to narrow down the analysis based on specific criteria. Filters allow users to focus on files that meet their requirements and exclude irrelevant ones. With options to filter by file extensions, size, last modified timestamp, and permissions, users can customize the analysis to their specific needs. This flexibility empowers users to conduct targeted investigations and gain valuable insights from the files that match their specified criteria.

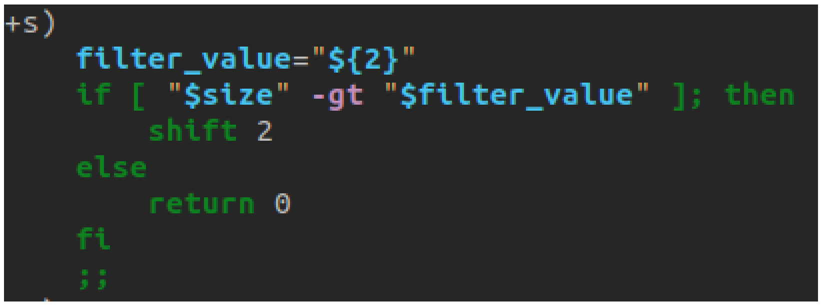
Filter Options: The script provides several filter options to narrow down the file analysis based on specific criteria:

- `-e [extensions]`: Allows filtering files by extensions, such as "txt" or "sh". Multiple extensions can be specified, separated by commas.



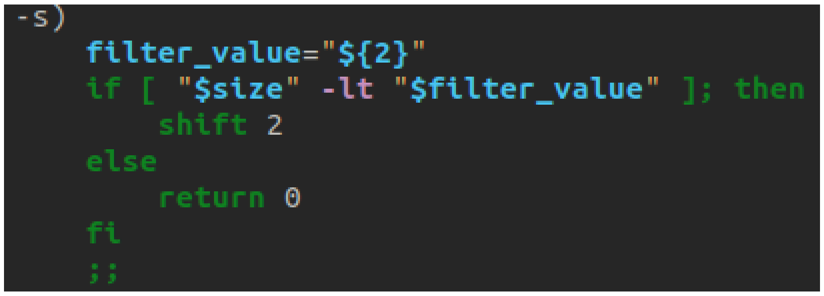
```
-e)
    filter_value="${2}"
    file_extension="${file##*.}"
    if [[ ",$filter_value," == *",$file_extension,"* ]]; then
        shift 2
    else
        return 0
    fi
    ;;
```

- `+s [size]`: Filters files that are larger than the specified size (in bytes).



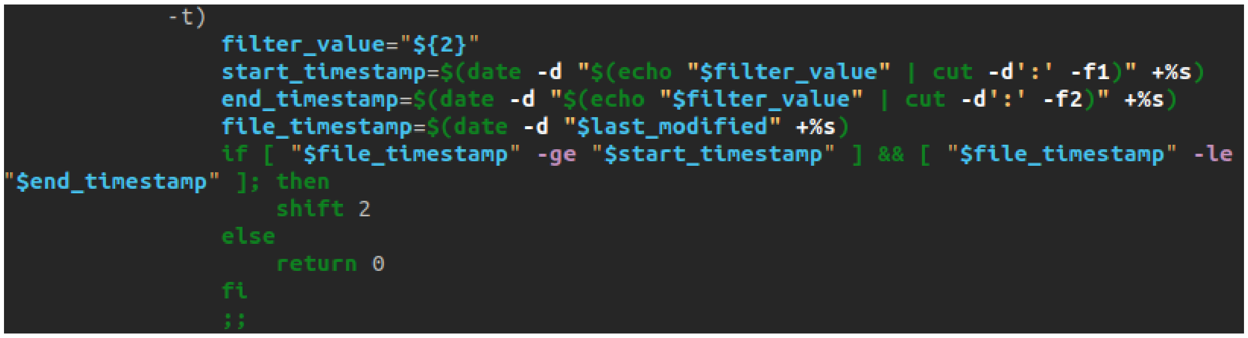
```
+s)
    filter_value="${2}"
    if [ "$size" -gt "$filter_value" ]; then
        shift 2
    else
        return 0
    fi
    ;;
```

- `-s [size]`: Filters files that are smaller than the specified size (in bytes).



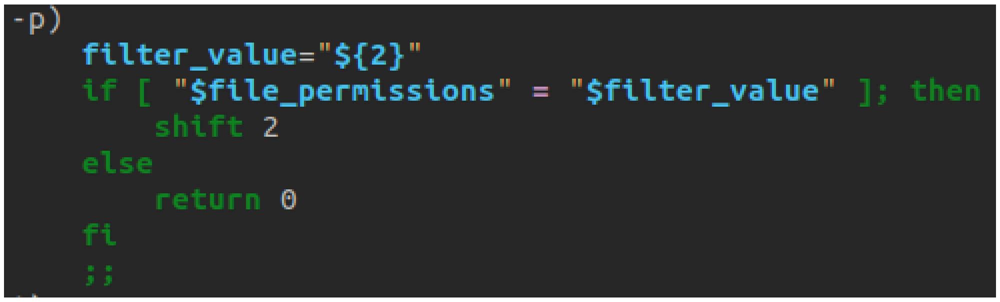
```
-s)
    filter_value="${2}"
    if [ "$size" -lt "$filter_value" ]; then
        shift 2
    else
        return 0
    fi
    ;;
```

- `-t [start_timestamp:end_timestamp]`: Filters files by the last modified timestamp within the specified range (in the format YYYY-MM-DD).



```
            -t)
                filter_value="${2}"
                start_timestamp=$(date -d "$(echo "$filter_value" | cut -d':' -f1)" +%s)
                end_timestamp=$(date -d "$(echo "$filter_value" | cut -d':' -f2)" +%s)
                file_timestamp=$(date -d "$last_modified" +%s)
                if [ "$file_timestamp" -ge "$start_timestamp" ] && [ "$file_timestamp" -le
"$end_timestamp" ]; then
                    shift 2
                else
                    return 0
                fi
                ;;
```

➢ `-p [permissions]`: Filters files by permissions, using the format (e.g., `-rw-rw-r--`).

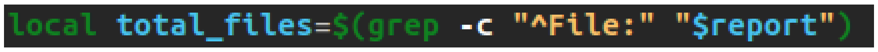


```
-p)
    filter_value="${2}"
    if [ "$file_permissions" = "$filter_value" ]; then
        shift 2
    else
        return 0
    fi
    ;;
```

- Summary Reports: Another advanced feature is the generation of summary reports. Alongside the detailed file analysis report, the script generates a summary report that provides a condensed overview of the analysis results. The summary report presents high-level information, including the total number of files analyzed and their cumulative size. This summary offers users a quick snapshot of the analysis outcome, allowing them to grasp the overall impact and scope of the analyzed files without delving into every individual file's details. By providing this concise summary, users can efficiently assess the magnitude and significance of the analysis results, making informed decisions based on the summarized information.

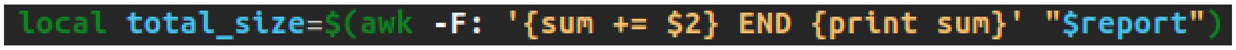
  Summary Report Generation: The script generates a summary report that provides an overview of the analysis results:

  ➢ The `generate_summary_report()` function calculates the total number of files analyzed using `grep -c "^File:" "$report"`.

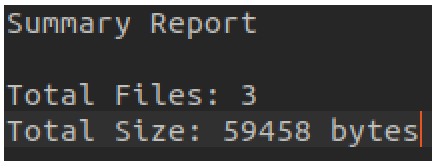
```
local total_files=$(grep -c "^File:" "$report")
```

  ➢ The cumulative size of the analyzed files is computed using `awk -F: '{sum += $2} END {print sum}' "$report"`.

```
local total_size=$(awk -F: '{sum += $2} END {print sum}' "$report")
```

  ➢ The summary report is then appended to the detailed report, displaying the total number of files and their cumulative size.

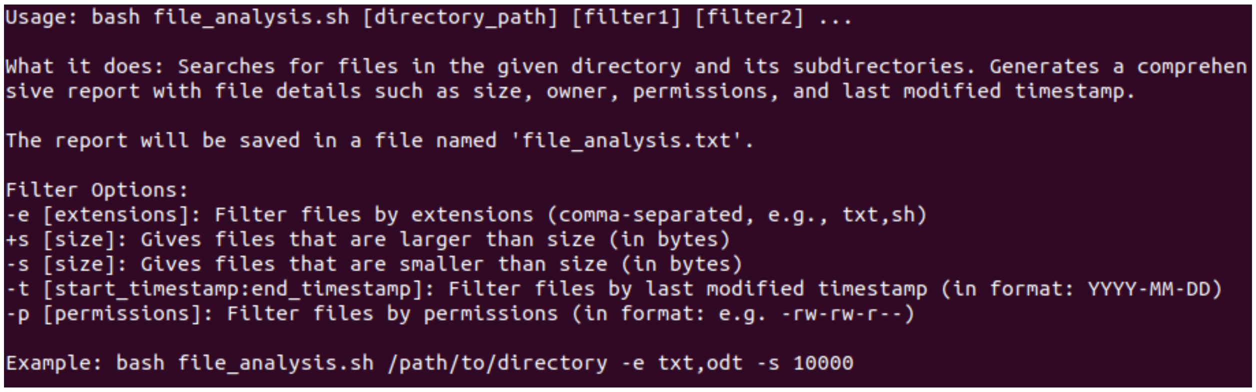
```
Summary Report

Total Files: 3
Total Size: 59458 bytes
```

6. User Friendliness

The user-friendly advantages of the code aim to make the file analysis process accessible, customizable, and informative for users. The clear usage instructions, comprehensive reports, customization options, and error handling contribute to an improved user experience and ease of use.

Some user friendly features that has been employed

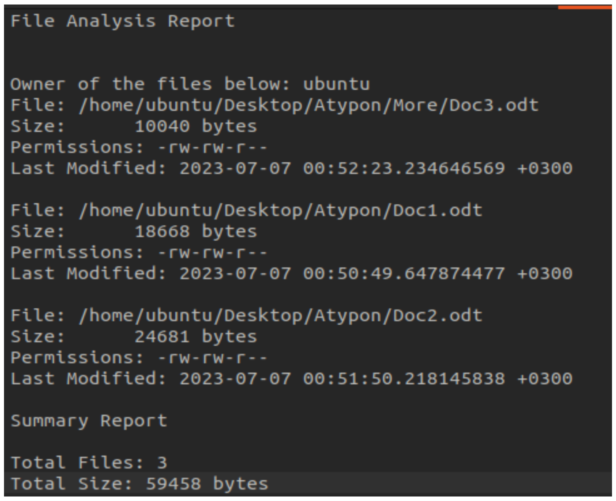
1. Easy Usage and Help Information: The script includes a `display_help()` function that provides clear usage instructions and explains the available filter options. Users can easily understand how to use the script effectively by running it with the `--help` or `-h` argument. The help information enhances user-friendliness by providing clear guidance.



```
Usage: bash file_analysis.sh [directory_path] [filter1] [filter2] ...

What it does: Searches for files in the given directory and its subdirectories. Generates a comprehen
sive report with file details such as size, owner, permissions, and last modified timestamp.

The report will be saved in a file named 'file_analysis.txt'.

Filter Options:
-e [extensions]: Filter files by extensions (comma-separated, e.g., txt,sh)
+s [size]: Gives files that are larger than size (in bytes)
-s [size]: Gives files that are smaller than size (in bytes)
-t [start_timestamp:end_timestamp]: Filter files by last modified timestamp (in format: YYYY-MM-DD)
-p [permissions]: Filter files by permissions (in format: e.g. -rw-rw-r--)

Example: bash file_analysis.sh /path/to/directory -e txt,odt -s 10000
```

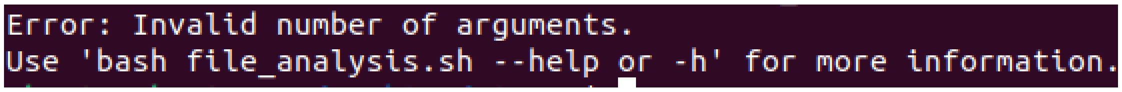
2. Customizable File Analysis: The script allows users to customize the file analysis based on their specific requirements. Users can specify various filters such as file extensions, size, last modified timestamp, and permissions to narrow down the analysis. This flexibility empowers users to focus on specific files of interest and obtain tailored analysis results.

3. Comprehensive Reports: The script generates comprehensive reports on analyzed files, providing detailed information such as size, owner, permissions, and last modified timestamp. These reports are saved as "file_analysis.txt." In addition to the detailed report, a summary report is generated, offering high-level information on the total number of files analyzed and their cumulative size. Users can quickly grasp the overall analysis results through the summary report, while the detailed report allows for in-depth insights into individual file attributes. Together, these reports enable users to make informed decisions based on the comprehensive file analysis.



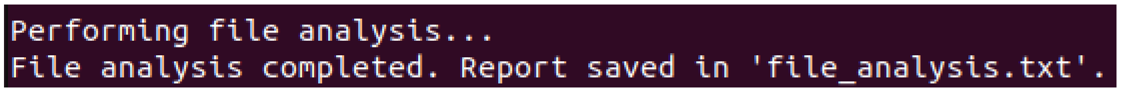
```
File Analysis Report


Owner of the files below: ubuntu
File: /home/ubuntu/Desktop/Atypon/More/Doc3.odt
Size:      10040 bytes
Permissions: -rw-rw-r--
Last Modified: 2023-07-07 00:52:23.234646569 +0300

File: /home/ubuntu/Desktop/Atypon/Doc1.odt
Size:      18668 bytes
Permissions: -rw-rw-r--
Last Modified: 2023-07-07 00:50:49.647874477 +0300

File: /home/ubuntu/Desktop/Atypon/Doc2.odt
Size:      24681 bytes
Permissions: -rw-rw-r--
Last Modified: 2023-07-07 00:51:50.218145838 +0300

Summary Report

Total Files: 3
Total Size: 59458 bytes
```

4. Error Handling: The script incorporates error handling to guide users in case of incorrect usage. If the script is executed with an incorrect number of arguments, it displays an error message indicating the invalid number of arguments and prompts users to refer to the help information. This error handling helps users understand and rectify incorrect usage, improving the overall user experience.



```
Error: Invalid number of arguments.
Use 'bash file_analysis.sh --help or -h' for more information.
```

5. Progress Updates: The script provides progress updates by printing informative messages to the console. It notifies users when the file analysis is being performed, and once the analysis is completed, it informs users about the location where the report is saved. These progress updates keep users informed about the script's execution status, adding to the user-friendliness.



```
Performing file analysis...
File analysis completed. Report saved in 'file_analysis.txt'.
```

7. Reflections

Developing the entire project encompassing the script and its functionalities has provided valuable insights, highlighted challenges, and opened avenues for potential future improvements. Let's reflect on the project as a whole, considering the lessons learned, difficulties encountered, and potential future improvements.

7.1 Lessons Learned

- Familiarity with Ubuntu OS: Throughout the project, one valuable lesson is that I was gaining familiarity with the Ubuntu operating system. Working with Ubuntu provided insights into its features, functionalities, and command-line interface (CLI). Acquiring knowledge about the OS expanded the skill set and enabled efficient utilization of Ubuntu-specific tools and commands.

- Command-Line Interface (CLI) Proficiency: The project emphasized the significance of CLI proficiency. Becoming proficient in CLI commands allowed for efficient navigation, file operations, and executing scripts. Learning various commands enhanced productivity, facilitated troubleshooting, and enabled the effective use of the Ubuntu terminal.

- Value of Modularity and Code Reusability: Implementing modular components and writing reusable code promotes maintainability and extensibility. Modular design enables easier updates and enhancements, allowing for seamless integration of new features and adaptability to changing requirements.

- Understanding Shell Scripting: Developing the script enhanced understanding of shell scripting concepts. Learning about variables, loops, conditional statements, and file operations in the context of shell scripting provided a foundation for automating tasks and writing efficient scripts. This knowledge can be applied to future projects and automation needs.

7.2 Difficulties Encountered

- Handling Complex File Operations: Dealing with file-related operations and managing file attributes, such as size, permission, and time stamp, presented challenges. Ensuring efficient file handling, handling edge cases, and managing performance for large-scale file processing required careful consideration and testing.

- User Input Validation: Validating user inputs and handling various input scenarios, such as different file types, filter options, and date formats, proved to be a complex task. Implementing robust input validation mechanisms while providing meaningful error messages posed challenges that required attention to detail.

7.3 Potential Future Improvements

- Error Handling and User Input Validation: Enhance error handling to provide more specific and informative error messages when incorrect or invalid inputs are provided. Validate user inputs rigorously to ensure that the script gracefully handles various scenarios and prevents unexpected behavior.

- Error Recovery and Resilience: Implement error recovery mechanisms to handle exceptional cases gracefully. For example, if a file analysis fails for a specific file, consider options to handle the error, skip the file, and continue the analysis for other files without terminating the entire process.

- Performance Optimization: Assess the script's performance and identify areas where optimization can be applied. Analyze the performance of resource-intensive commands and explore alternative methods or tools that could improve efficiency. Additionally, consider optimizations to minimize I/O operations, reduce unnecessary processing, and optimize resource usage.

- Enhanced Filter Options: Expand the available filter options based on specific use cases. Consider adding additional filtering criteria, such as file creation timestamp or file content, to provide users with more flexibility and customization options.

- Enhanced Reporting and Visualization: Explore options to improve the presentation and visualization of the analysis results. Consider integrating graphing or charting capabilities to provide visual representations of the data, enabling users to quickly understand patterns and trends within the analyzed files.

By considering these areas for improvement, you can enhance the functionality, performance, and user experience of the script in the future iterations, making it even more robust and user-friendly.