Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Linux Laboratory ENCS3130

Project#2 Report

Python Project

Prepared by: Sara Ewaida **ID:**1203048

Instructor: Dr. Mohammad Jubran

Section:1

Date:6/13/2024

Table of Contents

Introduction

This project aims to automate command execution in Python, focusing on a variety of tasks that can be orchestrated through predefined scripts. The project is designed to provide a foundation for automation testing scenarios across different environments. By implementing a range of commands and utilizing a structured approach to execution, the resulting software ensures robust and efficient task management.

The core functionality of this project includes commands such as moving the most recent file from one directory to another, categorizing files based on size, counting files in a directory, deleting specified files, renaming files, listing directory contents, and sorting files based on various criteria. These commands are executed through a main script that reads a predefined script file, parses the commands, and executes them accordingly.

Configuration settings are managed through a JSON file, which includes parameters such as the threshold size for categorizing files, the maximum number of commands to execute per script, the maximum number of log files to retain, whether to place PASSED and FAILED logs in the same directory, and the output format (CSV or log).

To facilitate this automation, the project employs the Factory Design Pattern for command creation, Python's logging module for detailed log management, and the argparse module for handling command-line arguments. The logging module is particularly useful for debugging and producing the final script results, ensuring that no print statements are used within the execution flow.

Components

Command Implementation

1. Mv_last <src_directory> <des_directory>:

Moves the most recent file from the source directory to the destination directory.

2. Categorize <directory>:

Splits files in the given directory into two types:

An inner directory with files less than a specified <threshold_size>.

An inner directory with files more than the specified <threshold_size>.

3. Count <directory>:

Counts the number of files in the specified directory.

4. Delete <file> <directory>:

Deletes a specified file from the specified directory.

5. Rename <old_name> <new_name> <directory>:

Renames a file in the specified directory.

6. List <directory>:

Lists all files and directories in the specified directory.

7. Sort <directory> <criteria>:

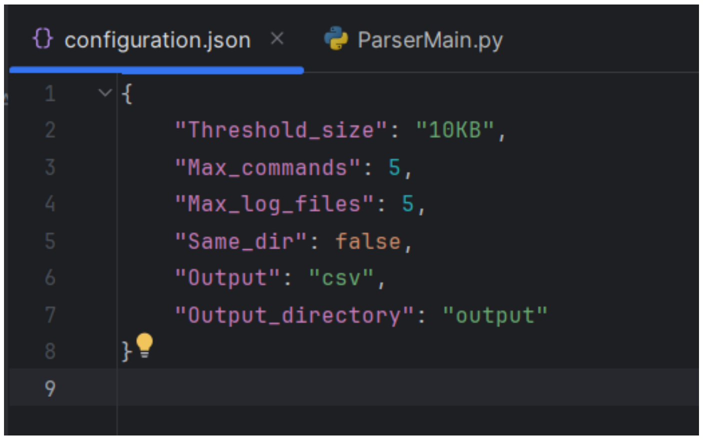
Sorts the files in the specified directory based on the specified criteria:

Supported criteria: "name", "date", "size".

Configuration

Utilizes a Python JSON configuration file (configuration.json) that contains essential values for running the application:

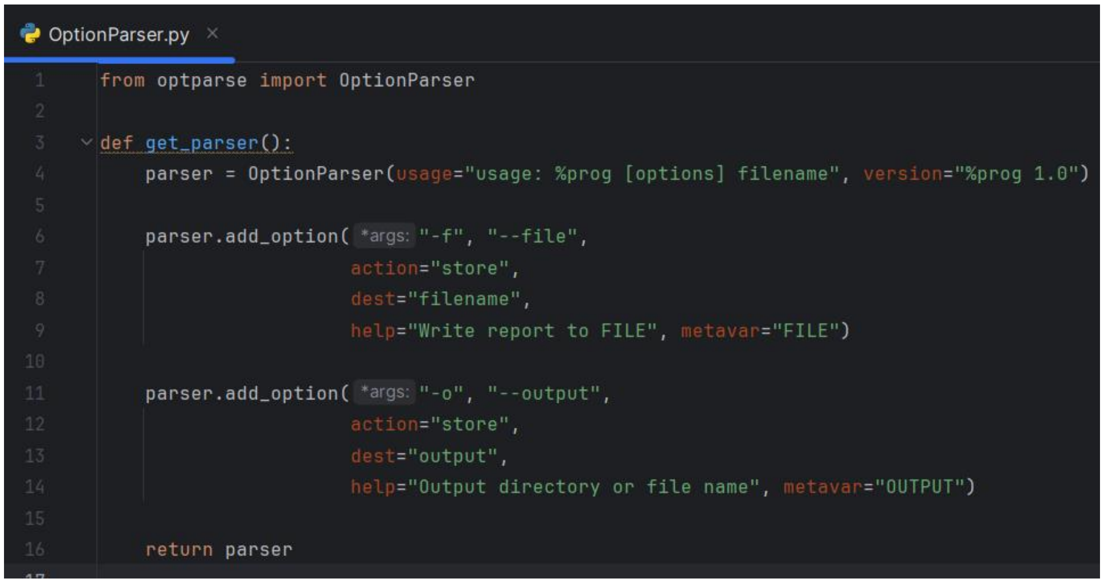
- Threshold_size: Value needed by the Categorize command.
- Max_commands: Maximum number of commands that should be executed per script.
- Max_log_files: Maximum number of files in the log directory; older files are deleted if the limit is exceeded.
- Same_dir: Specifies whether PASSED and FAILED should be placed in the same directory or in internal pass and fail subdirectories.
- Output: Supports two types of results – CSV and log files. If CSV is chosen, the output consists of two columns, each statement with the result. If log is chosen, each statement with its result is printed in the file.



```json
{
    "Threshold_size": "10KB",
    "Max_commands": 5,
    "Max_log_files": 5,
    "Same_dir": false,
    "Output": "csv",
    "Output_directory": "output"
}
```

Option Parser

In the project, the optparse module is utilized for command-line option parsing, allowing users to specify input and output file paths conveniently. This module provides a flexible way to handle command-line arguments, ensuring that the script can be run with different configurations without modifying the code.



```python
from optparse import OptionParser

def get_parser():
    parser = OptionParser(usage="usage: %prog [options] filename", version="%prog 1.0")

    parser.add_option( *args: "-f", "--file",
                      action="store",
                      dest="filename",
                      help="Write report to FILE", metavar="FILE")

    parser.add_option( *args: "-o", "--output",
                      action="store",
                      dest="output",
                      help="Output directory or file name", metavar="OUTPUT")

    return parser
```

usage: Specifies the command-line syntax for the script.

version: Defines the version of the script.

add_option: Adds a command-line option to the parser.

-f, --file: Option for specifying the input file path. The dest parameter indicates that the value will be stored in the filename attribute.

-o, --output: Option for specifying the output file path. The dest parameter indicates that the value will be stored in the output attribute.

To run the script using the specified input and output files, we can use the following command in your terminal:

```
python ParserMain.py -f script.txt -o output.log
```

-f script.txt: This option specifies the input file, in this case, script.txt. The -f flag is used to indicate the file path.

-o output.log: This option specifies the output log file, in this case, output.log. The -o flag is used to indicate the output file path.

Technologies and Methodologies Used

Logging Library in Python

The Python logging module was extensively used throughout the project to provide detailed insights into the program's execution flow and to aid in debugging. The logging setup included:

- Configuration: Logging was configured to write to a log file named CommandDebugger.log, capturing debug and higher-level messages. This setup ensured that all relevant information was logged for each command executed.
- Usage: Within each command class and the main script, logging statements were used to log the start and end of operations, success or failure messages, and any errors encountered. This provided a comprehensive record of the system's behavior during execution.
- Benefits: The use of logging enabled real-time tracking of the system's activities, facilitated easier debugging by providing detailed error messages, and ensured that the system's state could be reconstructed and analyzed post-execution.

JSON Files in Python

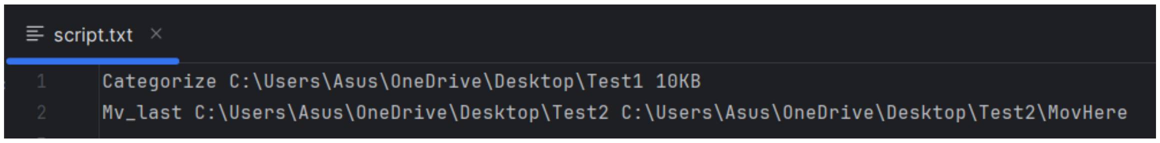
JSON files were used for configuration management, providing a flexible and easy-to-read way to store various settings and parameters required by the system. Key aspects of using JSON files included:

- Configuration Storage: The configuration.json file stored settings such as threshold size, maximum commands, maximum log files, whether to use the same directory for logs, and the desired output format.
- Loading and Parsing: The configuration file was loaded and parsed at the start of the main script, allowing the system to dynamically adjust its behavior based on the settings specified. This made the system highly configurable without changing the code.
- Flexibility: JSON's hierarchical structure allowed for organized storage of configuration data, and Python's built-in json module made it straightforward to read and write JSON data.

Testing and Results

1.1Script Execution Outcome

The script here executed two main commands to manipulate files within directories "Test1" and "Test2" on the desktop:

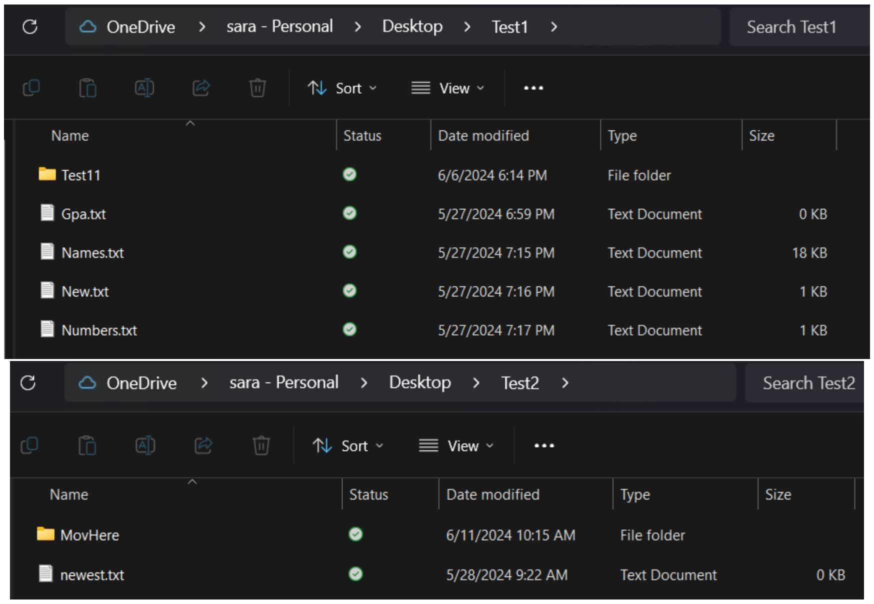


script.txt

```
Categorize C:\Users\Asus\OneDrive\Desktop\Test1 10KB
Mv_last C:\Users\Asus\OneDrive\Desktop\Test2 C:\Users\Asus\OneDrive\Desktop\Test2\MovHere
```

1.2 Before

Before state of the directories



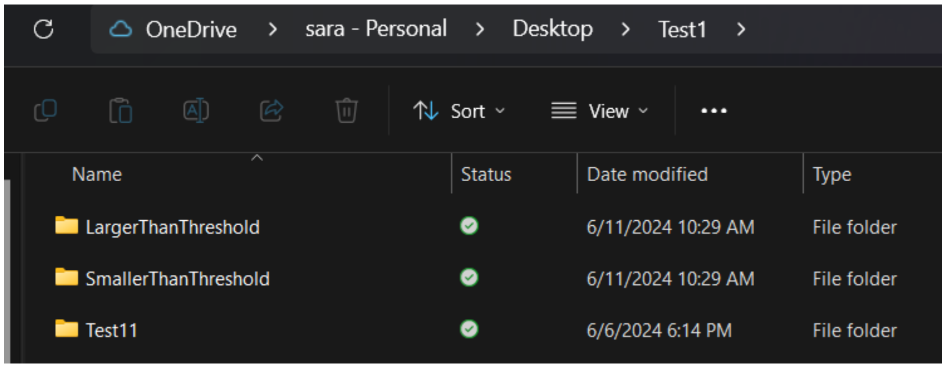
OneDrive > sara - Personal > Desktop > Test1 >

Name	Status	Date modified	Type	Size
Test11	✓	6/6/2024 6:14 PM	File folder	
Gpa.txt	✓	5/27/2024 6:59 PM	Text Document	0 KB
Names.txt	✓	5/27/2024 7:15 PM	Text Document	18 KB
New.txt	✓	5/27/2024 7:16 PM	Text Document	1 KB
Numbers.txt	✓	5/27/2024 7:17 PM	Text Document	1 KB

OneDrive > sara - Personal > Desktop > Test2 >

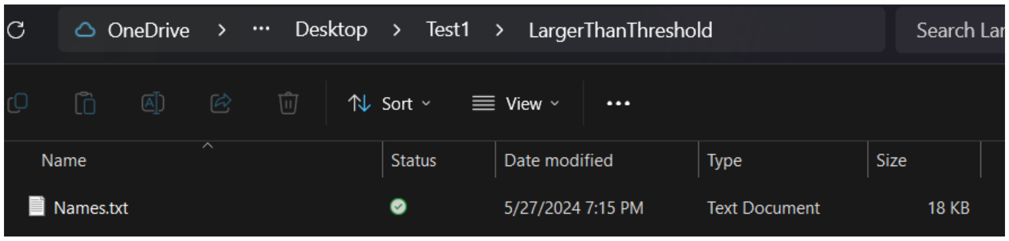
Name	Status	Date modified	Type	Size
MovHere	✓	6/11/2024 10:15 AM	File folder	
newest.txt	✓	5/28/2024 9:22 AM	Text Document	0 KB

1.3 After running



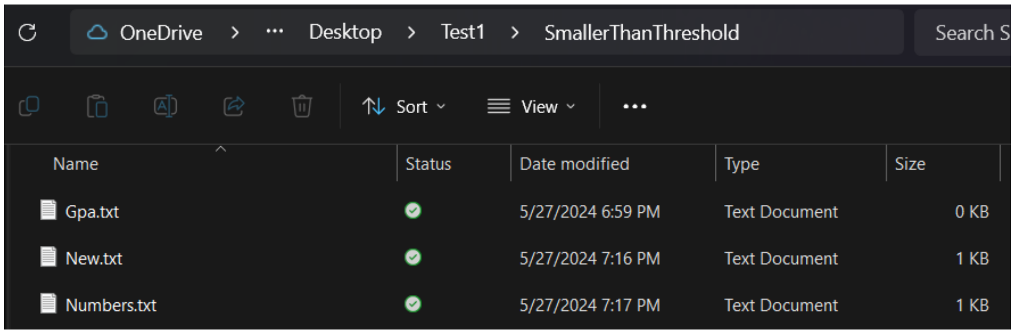
OneDrive > sara - Personal > Desktop > Test1 >

Name	Status	Date modified	Type
LargerThanThreshold	✓	6/11/2024 10:29 AM	File folder
SmallerThanThreshold	✓	6/11/2024 10:29 AM	File folder
Test11	✓	6/6/2024 6:14 PM	File folder



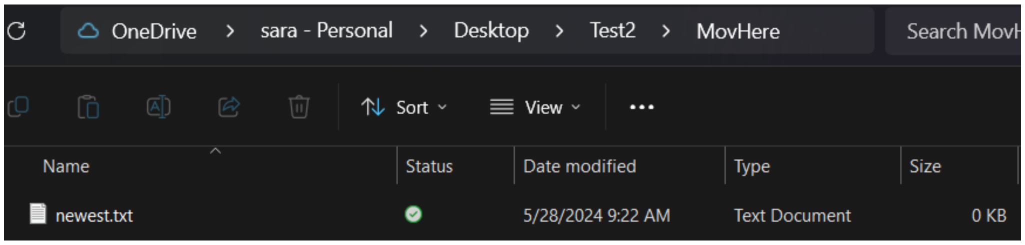
OneDrive > ··· > Desktop > Test1 > LargerThanThreshold

Name	Status	Date modified	Type	Size
Names.txt	✓	5/27/2024 7:15 PM	Text Document	18 KB



OneDrive > ··· > Desktop > Test1 > SmallerThanThreshold

Name	Status	Date modified	Type	Size
Gpa.txt	✓	5/27/2024 6:59 PM	Text Document	0 KB
New.txt	✓	5/27/2024 7:16 PM	Text Document	1 KB
Numbers.txt	✓	5/27/2024 7:17 PM	Text Document	1 KB



OneDrive > sara - Personal > Desktop > Test2 > MovHere

Name	Status	Date modified	Type	Size
newest.txt	✓	5/28/2024 9:22 AM	Text Document	0 KB

Categorize Command

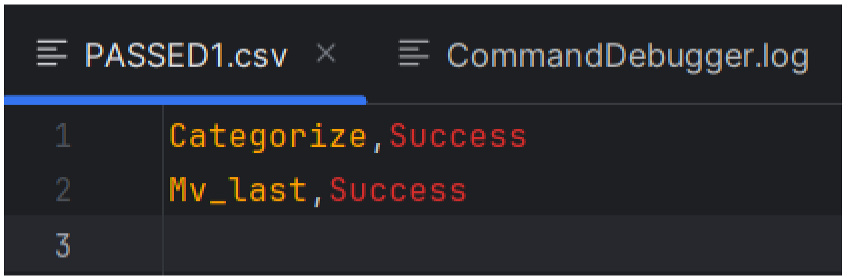
- Command: Categorize C:\Users\Asus\OneDrive\Desktop\Test1 10KB
- Purpose: This command categorizes files in the "Test1" directory based on their size threshold of 10KB. Files smaller than 10KB are moved to a new directory named "SmallerThanThreshold," and files larger or equal to 10KB are moved to another directory named "LargerThanThreshold."
- Result: As shown in the 'After running' screenshot, within the "Test1" directory, files have been sorted into the appropriate new directories according to their size, demonstrating the script's effectiveness in file organization based on size criteria.

Mv_last Command

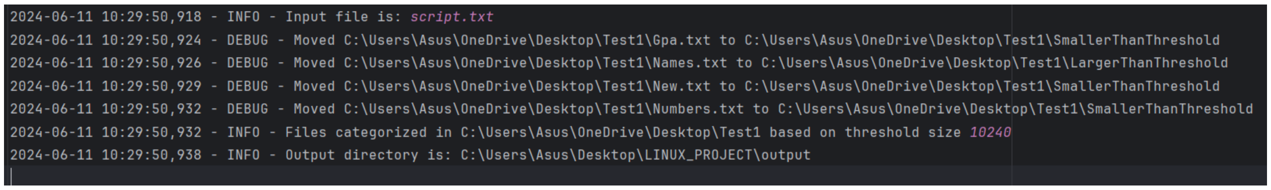
- Command:Mv_lastC:\Users\Asus\OneDrive\Desktop\Test2 C:\Users\Asus\OneDrive\Desktop\Test2\MovHere
- Purpose: Moves the most recently modified file from the "Test2" directory to the "MovHere" subdirectory.

- Result: The newest file from "Test2" directory has been successfully moved to "MovHere," showing the script's capability to automate file management tasks by handling the most recently updated files.

1.4 Test Output



```
Categorize,Success
Mv_last,Success
```

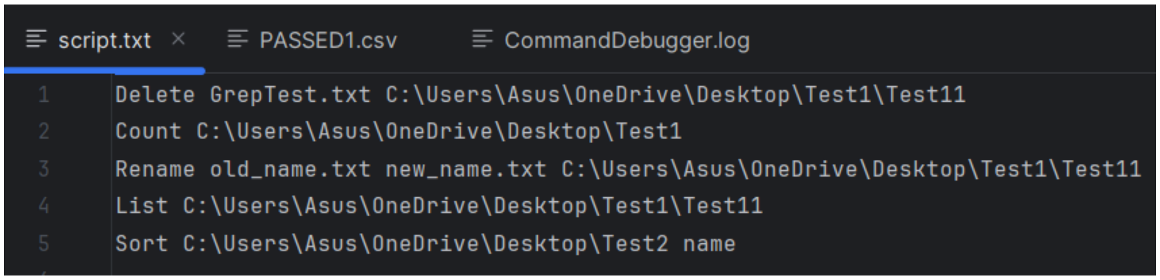


```
2024-06-11 10:29:50,918 - INFO - Input file is: script.txt
2024-06-11 10:29:50,924 - DEBUG - Moved C:\Users\Asus\OneDrive\Desktop\Test1\Gpa.txt to C:\Users\Asus\OneDrive\Desktop\Test1\SmallerThanThreshold
2024-06-11 10:29:50,926 - DEBUG - Moved C:\Users\Asus\OneDrive\Desktop\Test1\Names.txt to C:\Users\Asus\OneDrive\Desktop\Test1\LargerThanThreshold
2024-06-11 10:29:50,929 - DEBUG - Moved C:\Users\Asus\OneDrive\Desktop\Test1\New.txt to C:\Users\Asus\OneDrive\Desktop\Test1\SmallerThanThreshold
2024-06-11 10:29:50,932 - DEBUG - Moved C:\Users\Asus\OneDrive\Desktop\Test1\Numbers.txt to C:\Users\Asus\OneDrive\Desktop\Test1\SmallerThanThreshold
2024-06-11 10:29:50,932 - INFO - Files categorized in C:\Users\Asus\OneDrive\Desktop\Test1 based on threshold size 10240
2024-06-11 10:29:50,938 - INFO - Output directory is: C:\Users\Asus\Desktop\LINUX_PROJECT\output
```

2.1Script Execution Outcome

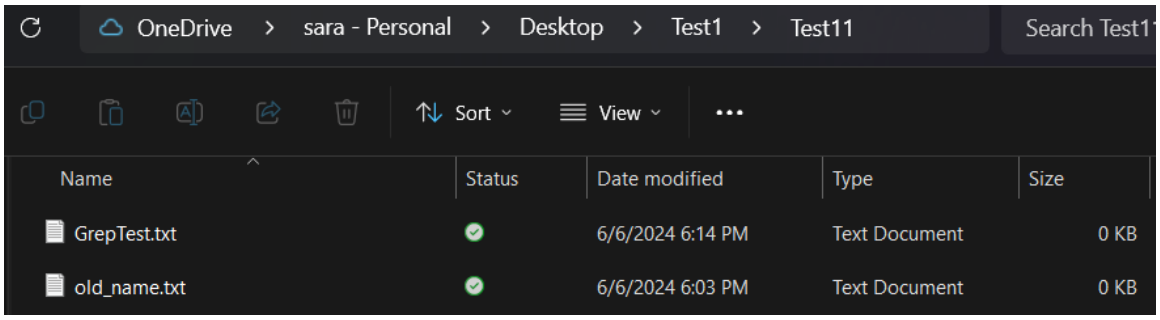
The script contained commands designed to manipulate files across two test directories (Test1 and Test2) on the system. The commands included:

- Delete a specific file.
- Count the number of files in a directory.
- Rename a file.
- List all files and directories.
- Sort files by name.

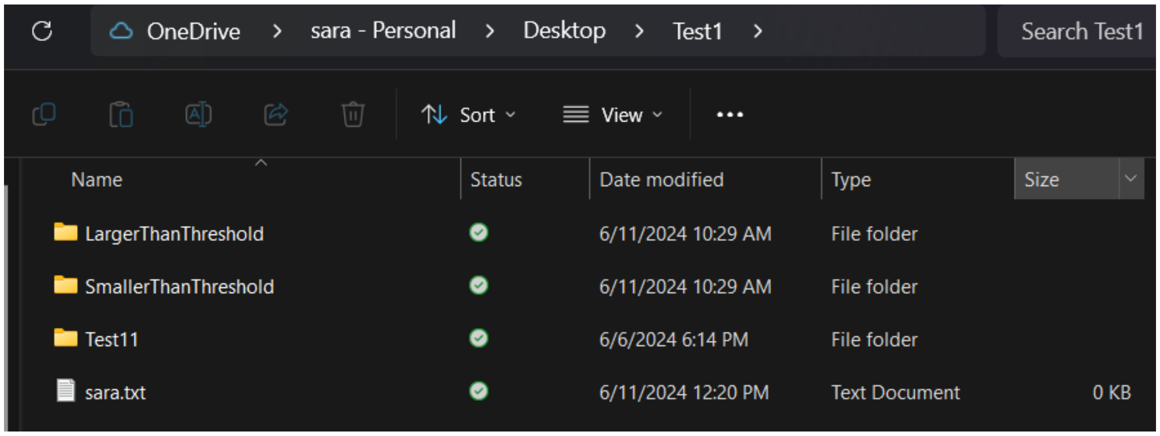


```
Delete GrepTest.txt C:\Users\Asus\OneDrive\Desktop\Test1\Test11
Count C:\Users\Asus\OneDrive\Desktop\Test1
Rename old_name.txt new_name.txt C:\Users\Asus\OneDrive\Desktop\Test1\Test11
List C:\Users\Asus\OneDrive\Desktop\Test1\Test11
Sort C:\Users\Asus\OneDrive\Desktop\Test2 name
```

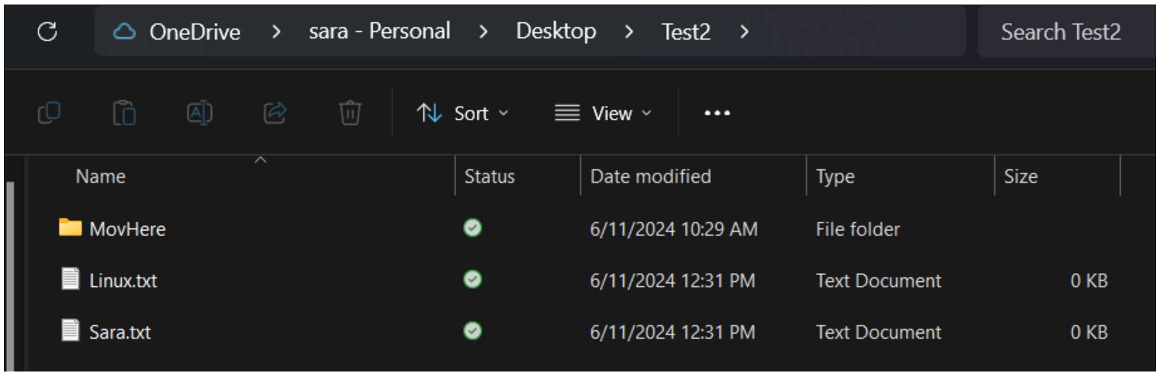
2.2 Before


OneDrive > sara - Personal > Desktop > Test1 > Test11

Name	Status	Date modified	Type	Size
GrepTest.txt	✓	6/6/2024 6:14 PM	Text Document	0 KB
old_name.txt	✓	6/6/2024 6:03 PM	Text Document	0 KB

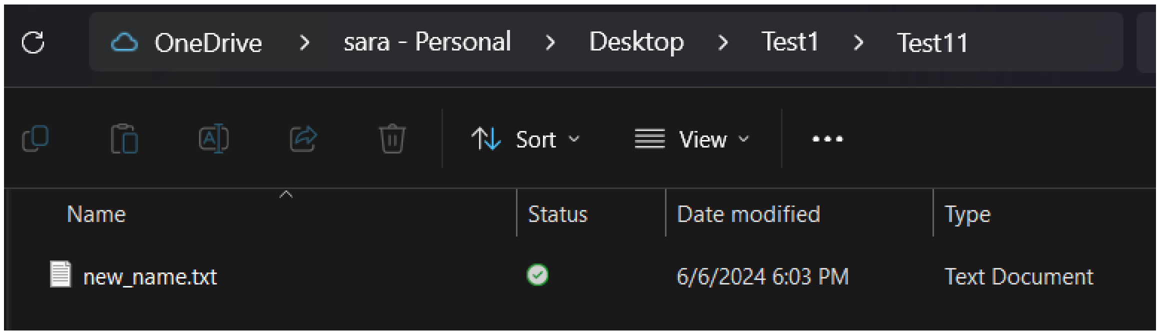

OneDrive > sara - Personal > Desktop > Test1 >

Name	Status	Date modified	Type	Size
LargerThanThreshold	✓	6/11/2024 10:29 AM	File folder	
SmallerThanThreshold	✓	6/11/2024 10:29 AM	File folder	
Test11	✓	6/6/2024 6:14 PM	File folder	
sara.txt	✓	6/11/2024 12:20 PM	Text Document	0 KB


OneDrive > sara - Personal > Desktop > Test2 >

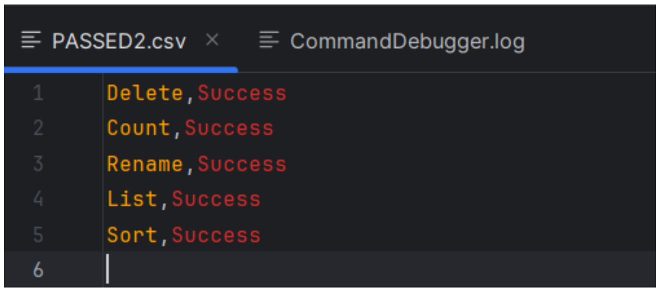
Name	Status	Date modified	Type	Size
MovHere	✓	6/11/2024 10:29 AM	File folder	
Linux.txt	✓	6/11/2024 12:31 PM	Text Document	0 KB
Sara.txt	✓	6/11/2024 12:31 PM	Text Document	0 KB

2.3 After Running

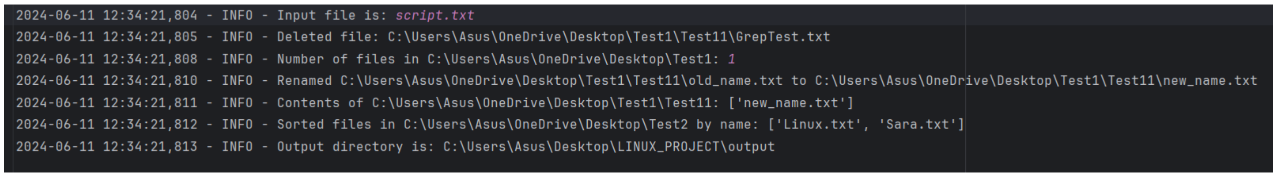

OneDrive > sara - Personal > Desktop > Test1 > Test11

Name	Status	Date modified	Type
new_name.txt	✓	6/6/2024 6:03 PM	Text Document

2.4 Test Output



```
Delete,Success
Count,Success
Rename,Success
List,Success
Sort,Success
```



```
2024-06-11 12:34:21,804 - INFO - Input file is: script.txt
2024-06-11 12:34:21,805 - INFO - Deleted file: C:\Users\Asus\OneDrive\Desktop\Test1\Test11\GrepTest.txt
2024-06-11 12:34:21,808 - INFO - Number of files in C:\Users\Asus\OneDrive\Desktop\Test1: 1
2024-06-11 12:34:21,810 - INFO - Renamed C:\Users\Asus\OneDrive\Desktop\Test1\Test11\old_name.txt to C:\Users\Asus\OneDrive\Desktop\Test1\Test11\new_name.txt
2024-06-11 12:34:21,811 - INFO - Contents of C:\Users\Asus\OneDrive\Desktop\Test1\Test11: ['new_name.txt']
2024-06-11 12:34:21,812 - INFO - Sorted files in C:\Users\Asus\OneDrive\Desktop\Test2 by name: ['Linux.txt', 'Sara.txt']
2024-06-11 12:34:21,813 - INFO - Output directory is: C:\Users\Asus\Desktop\LINUX_PROJECT\output
```

- **Delete Command:**

Functionality: Removes a specified file from the filesystem.

Execution: Delete GrepTest.txt C:\Users\Asus\OneDrive\Desktop\Test1\Test11

Result: The file GrepTest.txt was successfully deleted from the directory Test1\Test11.

Log Entry: "Deleted file: C:\Users\Asus\OneDrive\Desktop\Test1\Test11\GrepTest.txt"

- **Count Command:**

Functionality: Counts the number of files in a specified directory.

Execution: Count C:\Users\Asus\OneDrive\Desktop\Test1

Result: There is 1 file in the directory Test1.

Log Entry: "Number of files in C:\Users\Asus\OneDrive\Desktop\Test1: 1"

- **Rename Command:**

Functionality: Changes the name of a file within a directory.

Execution:Renameold_name.txtnew_name.txt
C:\Users\Asus\OneDrive\Desktop\Test1\Test11

Result: The file old_name.txt was successfully renamed to new_name.txt.

Log Entry:"Renamed C:\Users\Asus\OneDrive\Desktop\Test1\Test11\old_name.txt to C:\Users\Asus\OneDrive\Desktop\Test1\Test11\new_name.txt"

- **List Command:**

Functionality: Lists all files and directories within the specified directory.

Execution: List C:\Users\Asus\OneDrive\Desktop\Test1\Test11

Result: The directory contains the file new_name.txt.

LogEntry:"Contents of C:\Users\Asus\OneDrive\Desktop\Test1\Test11: ['new_name.txt']"

- **Sort Command:**

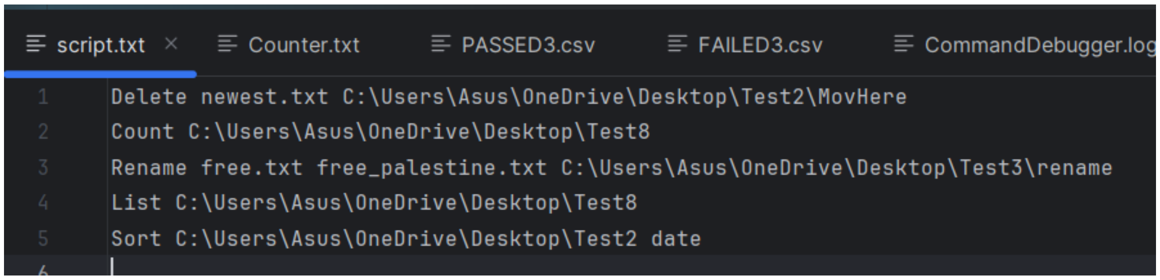
Functionality: Sorts the files in a directory based on a specified criterion, in this case by name.

Execution: Sort C:\Users\Asus\OneDrive\Desktop\Test2 name

Result: The files in Test2 are sorted by name, listed as ['Linux.txt', 'Sara.txt'].

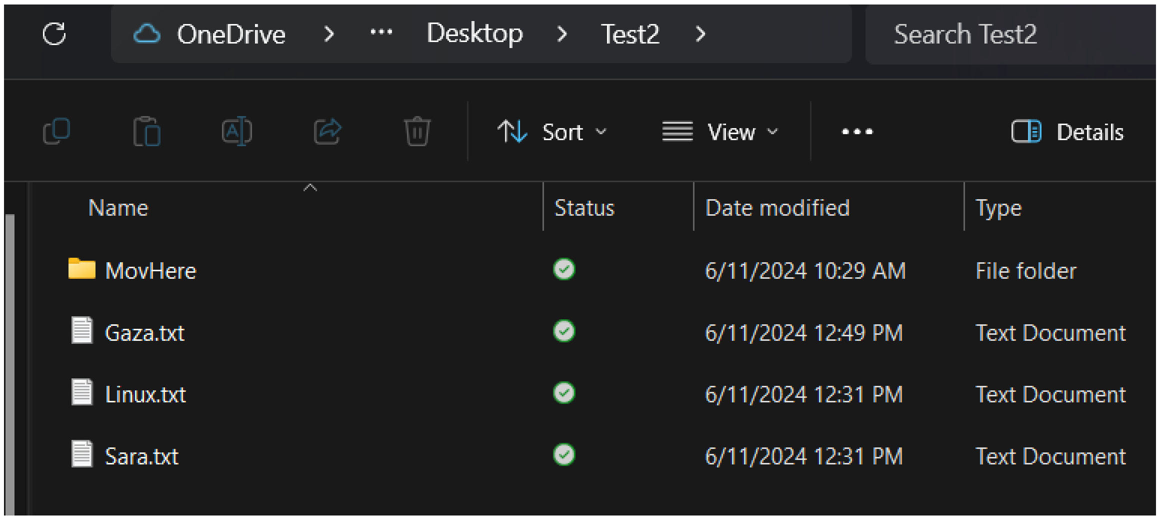
Log Entry: "Sorted files in C:\Users\Asus\OneDrive\Desktop\Test2 by name: ['Linux.txt', 'Sara.txt']"

3.1Script Execution Outcome

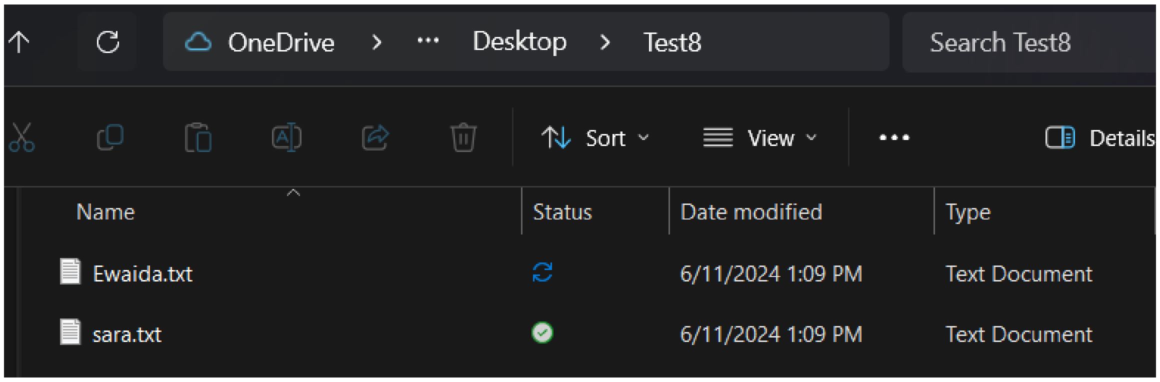


```
Delete newest.txt C:\Users\Asus\OneDrive\Desktop\Test2\MovHere
Count C:\Users\Asus\OneDrive\Desktop\Test8
Rename free.txt free_palestine.txt C:\Users\Asus\OneDrive\Desktop\Test3\rename
List C:\Users\Asus\OneDrive\Desktop\Test8
Sort C:\Users\Asus\OneDrive\Desktop\Test2 date
```

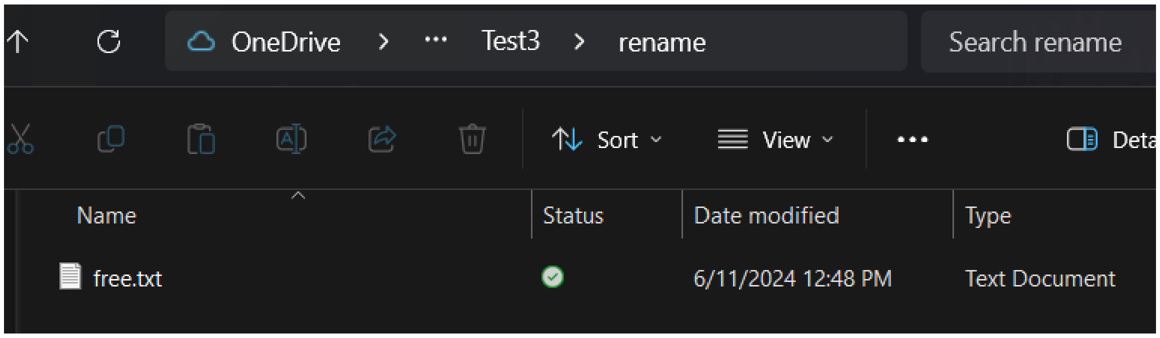
3.2 Before



Name	Status	Date modified	Type
MovHere	✓	6/11/2024 10:29 AM	File folder
Gaza.txt	✓	6/11/2024 12:49 PM	Text Document
Linux.txt	✓	6/11/2024 12:31 PM	Text Document
Sara.txt	✓	6/11/2024 12:31 PM	Text Document

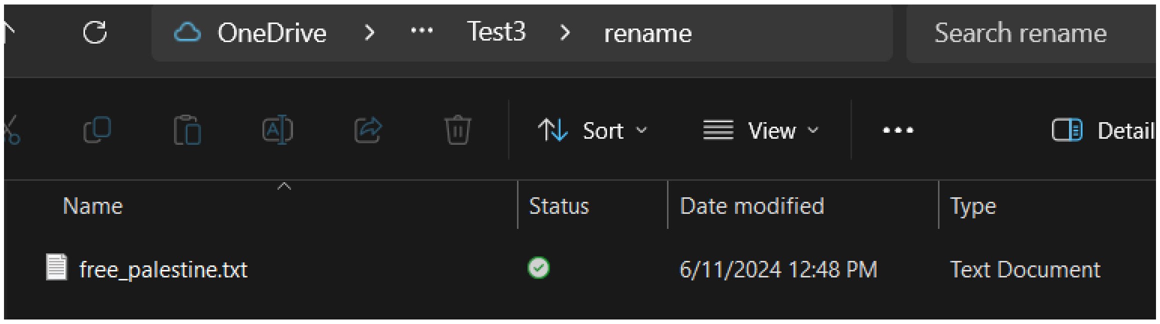


Name	Status	Date modified	Type
Ewaida.txt	⟳	6/11/2024 1:09 PM	Text Document
sara.txt	✓	6/11/2024 1:09 PM	Text Document



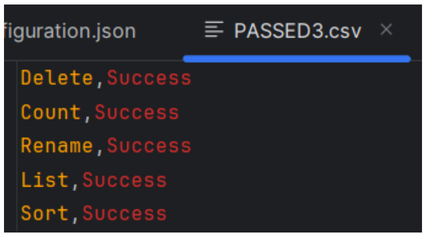
Name	Status	Date modified	Type
free.txt	✓	6/11/2024 12:48 PM	Text Document

3.3 After Running

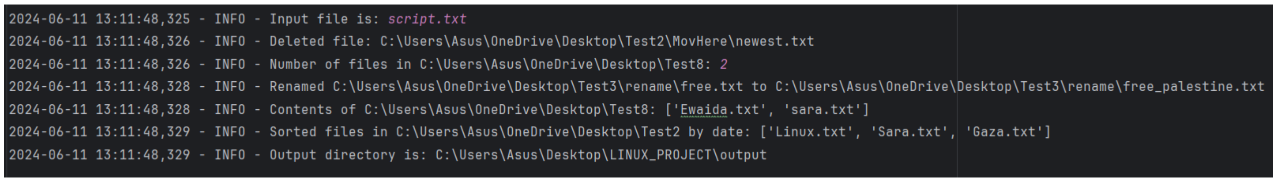


Name	Status	Date modified	Type
free_palestine.txt	✓	6/11/2024 12:48 PM	Text Document

3.4 Test Output

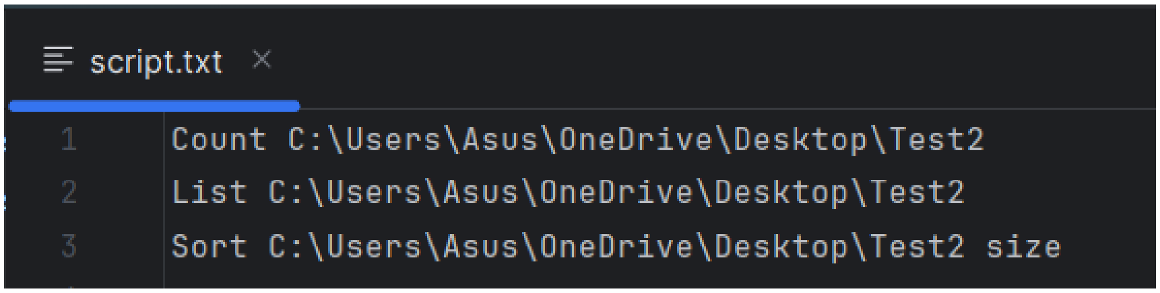


```
Delete,Success
Count,Success
Rename,Success
List,Success
Sort,Success
```



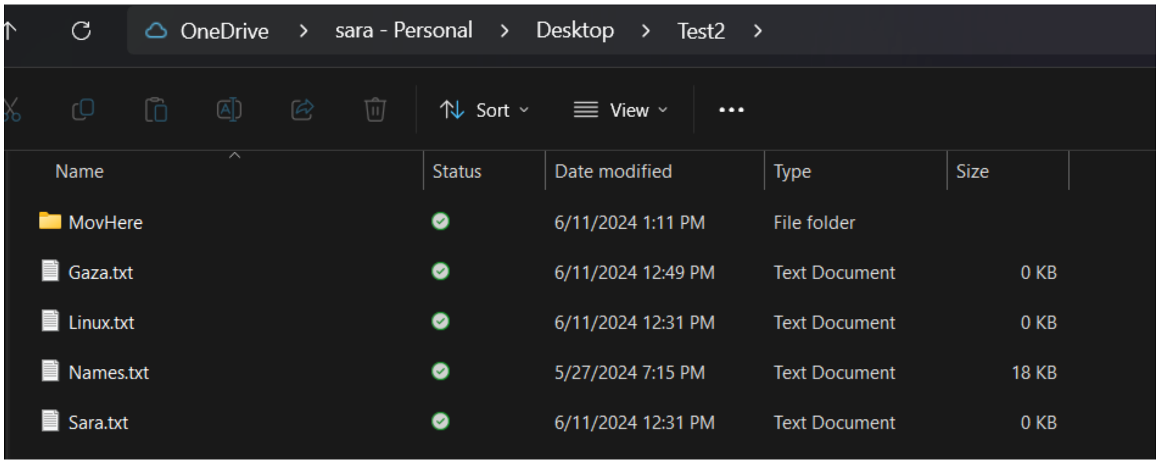
```
2024-06-11 13:11:48,325 - INFO - Input file is: script.txt
2024-06-11 13:11:48,326 - INFO - Deleted file: C:\Users\Asus\OneDrive\Desktop\Test2\MovHere\newest.txt
2024-06-11 13:11:48,326 - INFO - Number of files in C:\Users\Asus\OneDrive\Desktop\Test8: 2
2024-06-11 13:11:48,328 - INFO - Renamed C:\Users\Asus\OneDrive\Desktop\Test3\rename\free.txt to C:\Users\Asus\OneDrive\Desktop\Test3\rename\free_palestine.txt
2024-06-11 13:11:48,328 - INFO - Contents of C:\Users\Asus\OneDrive\Desktop\Test8: ['Ewaida.txt', 'sara.txt']
2024-06-11 13:11:48,329 - INFO - Sorted files in C:\Users\Asus\OneDrive\Desktop\Test2 by date: ['Linux.txt', 'Sara.txt', 'Gaza.txt']
2024-06-11 13:11:48,329 - INFO - Output directory is: C:\Users\Asus\Desktop\LINUX_PROJECT\output
```

4.1Script Execution Outcome



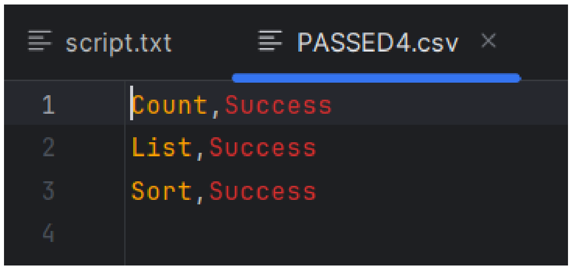
```
Count C:\Users\Asus\OneDrive\Desktop\Test2
List C:\Users\Asus\OneDrive\Desktop\Test2
Sort C:\Users\Asus\OneDrive\Desktop\Test2 size
```

4.2 Before

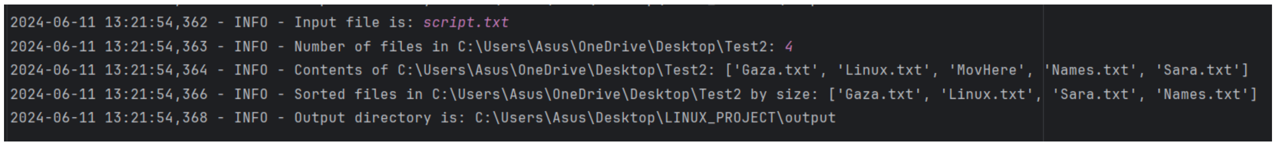


Name	Status	Date modified	Type	Size
MovHere	✓	6/11/2024 1:11 PM	File folder	
Gaza.txt	✓	6/11/2024 12:49 PM	Text Document	0 KB
Linux.txt	✓	6/11/2024 12:31 PM	Text Document	0 KB
Names.txt	✓	5/27/2024 7:15 PM	Text Document	18 KB
Sara.txt	✓	6/11/2024 12:31 PM	Text Document	0 KB

4.3 Test Output



```
Count,Success
List,Success
Sort,Success
```



```
2024-06-11 13:21:54,362 - INFO - Input file is: script.txt
2024-06-11 13:21:54,363 - INFO - Number of files in C:\Users\Asus\OneDrive\Desktop\Test2: 4
2024-06-11 13:21:54,364 - INFO - Contents of C:\Users\Asus\OneDrive\Desktop\Test2: ['Gaza.txt', 'Linux.txt', 'MovHere', 'Names.txt', 'Sara.txt']
2024-06-11 13:21:54,366 - INFO - Sorted files in C:\Users\Asus\OneDrive\Desktop\Test2 by size: ['Gaza.txt', 'Linux.txt', 'Sara.txt', 'Names.txt']
2024-06-11 13:21:54,368 - INFO - Output directory is: C:\Users\Asus\Desktop\LINUX_PROJECT\output
```

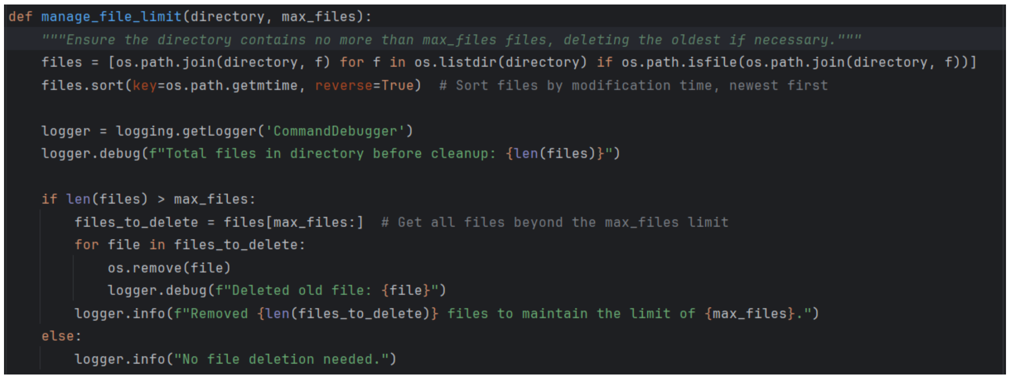
Managing File Limits

The primary objective of implementing file limit management is to maintain system efficiency and manage storage by preventing directories from accumulating an excessive number of log and output files. This is crucial in long-running applications or those generating a substantial number of output files, as it ensures that the system does not run out of storage space and that file navigation remains manageable.

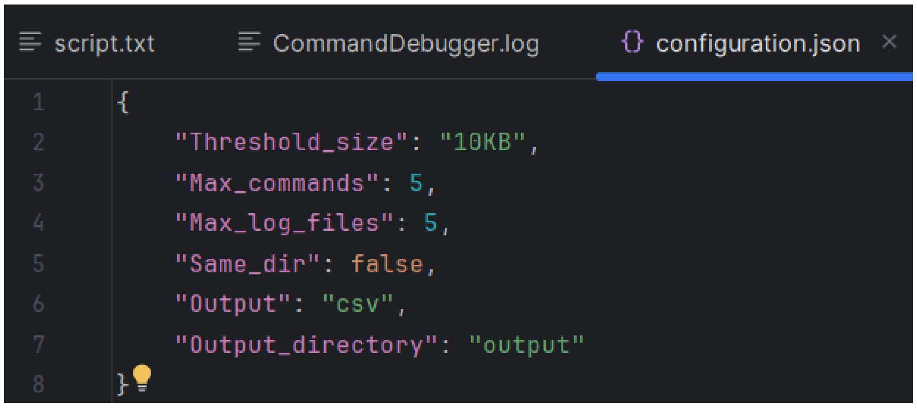
The manage_file_limit function enforces a cap on the number of files stored in specific directories (Passed and Failed outputs). When the number of files exceeds the defined maximum, the function removes the oldest files, retaining only the most recent files up to the specified limit.

Configuration

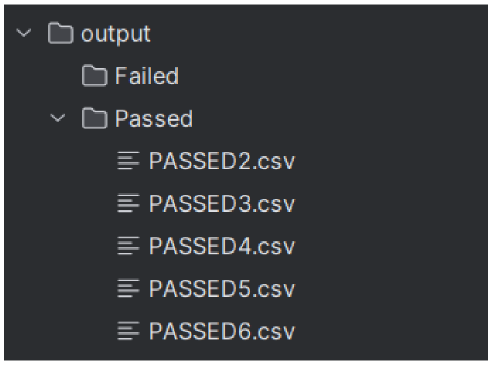
The limits for file retention are configurable through a JSON configuration file (configuration.json), which allows for easy adjustment of parameters such as maximum file counts without altering the script's source code.



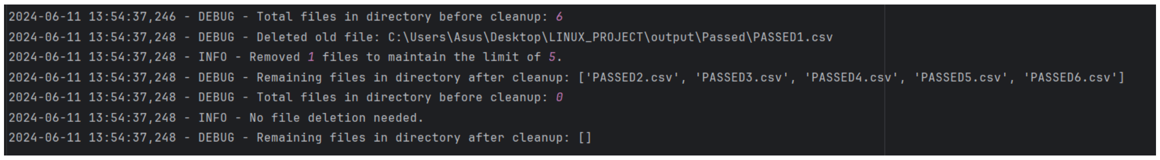
```python
def manage_file_limit(directory, max_files):
    """Ensure the directory contains no more than max_files files, deleting the oldest if necessary."""
    files = [os.path.join(directory, f) for f in os.listdir(directory) if os.path.isfile(os.path.join(directory, f))]
    files.sort(key=os.path.getmtime, reverse=True)  # Sort files by modification time, newest first

    logger = logging.getLogger('CommandDebugger')
    logger.debug(f"Total files in directory before cleanup: {len(files)}")

    if len(files) > max_files:
        files_to_delete = files[max_files:]  # Get all files beyond the max_files limit
        for file in files_to_delete:
            os.remove(file)
            logger.debug(f"Deleted old file: {file}")
        logger.info(f"Removed {len(files_to_delete)} files to maintain the limit of {max_files}.")
    else:
        logger.info("No file deletion needed.")
```



```json
{
    "Threshold_size": "10KB",
    "Max_commands": 5,
    "Max_log_files": 5,
    "Same_dir": false,
    "Output": "csv",
    "Output_directory": "output"
}
```



```
output
    Failed
    Passed
        PASSED2.csv
        PASSED3.csv
        PASSED4.csv
        PASSED5.csv
        PASSED6.csv
```



```
2024-06-11 13:54:37,246 - DEBUG - Total files in directory before cleanup: 6
2024-06-11 13:54:37,248 - DEBUG - Deleted old file: C:\Users\Asus\Desktop\LINUX_PROJECT\output\Passed\PASSED1.csv
2024-06-11 13:54:37,248 - INFO - Removed 1 files to maintain the limit of 5.
2024-06-11 13:54:37,248 - DEBUG - Remaining files in directory after cleanup: ['PASSED2.csv', 'PASSED3.csv', 'PASSED4.csv', 'PASSED5.csv', 'PASSED6.csv']
2024-06-11 13:54:37,248 - DEBUG - Total files in directory before cleanup: 0
2024-06-11 13:54:37,248 - INFO - No file deletion needed.
2024-06-11 13:54:37,248 - DEBUG - Remaining files in directory after cleanup: []
```

Conclusion

In this project, we developed a Python script to process file manipulation commands from an input file, generating logs based on command success or failure. The script was configured to manage log files, ensuring the number of log files did not exceed a specified limit. Key configurations included file size thresholds, maximum commands per run, maximum log files, output directories, and file formats.

Testing confirmed that the script correctly categorized passed and failed logs into separate directories when Same_dir was set to false, and managed all logs in the same directory when set to true. This project demonstrates the script's effectiveness in handling file operations and log management, providing a solid foundation for future enhancements.

References

1. Optparse Module

https://docs.python.org/3/library/optparse.html

2. Logging Module

https://www.youtube.com/watch?v=urrfJgHwIJA

https://docs.python.org/3/library/logging.html

3. JSON Module

https://docs.python.org/3/library/json.html

https://www.youtube.com/watch?v=9N6a-VLBa2I

4. Shutil Module

https://docs.python.org/3/library/shutil.html