جامعة بيرزيت

BIRZEIT UNIVERSITY

Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS3130

Linux Laboratory

Second Semester - 2023/2024

Python Project

Student Name: Asmaa Abed Al-Rahman Fares

Student ID:1210084

Section: 2

Student Name: Sarah Hassouneh

Student ID:1210068

Section: 1

Date: 10 June 2024

Abstract

This project focuses on the development of a Python-based automation tool designed to facilitate the execution of predefined file and directory management tasks. The tool implements a series of commands such as moving files, categorizing files by size, counting files, deleting files, renaming files, listing directory contents, and sorting files based on various criteria. Configuration flexibility is achieved through a JSON file, allowing users to customize parameters like threshold size and output format. The tool employs the argparse module for command-line interface support and the logging module for detailed operation logging.

Table of Contents

Procedure & Code Idea

Class Design

In this project we implemented our project with the following schema:

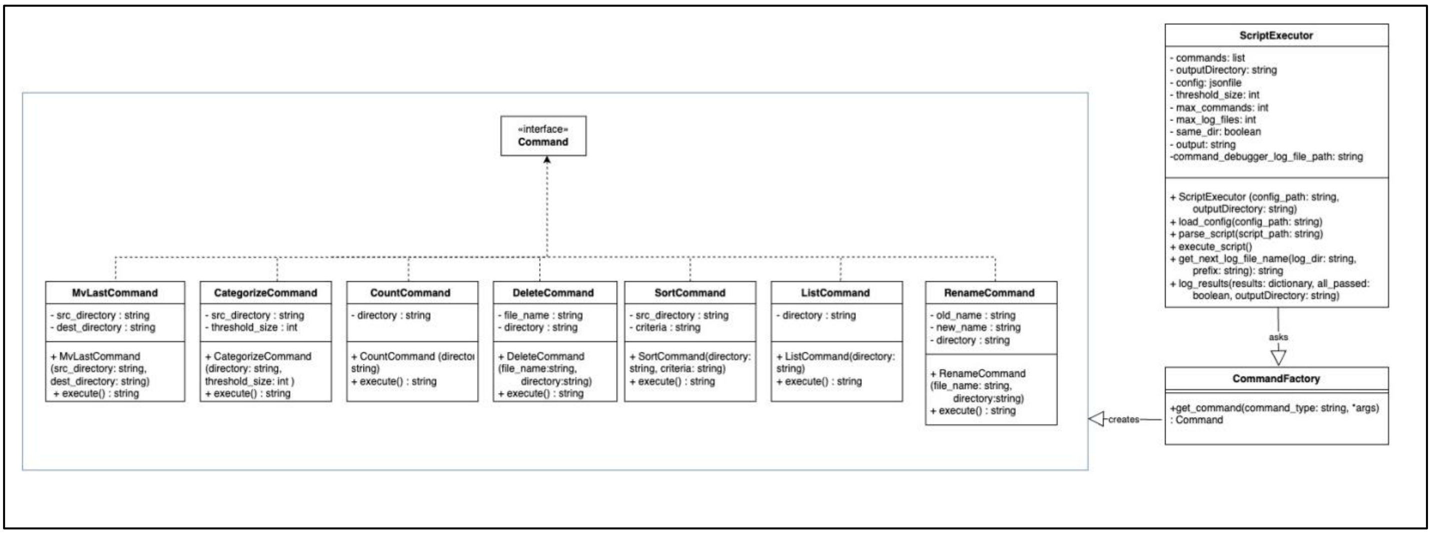

«interface»
Command
MvLastCommand
- src_directory : string
- dest_directory : string
+ MvLastCommand (src_directory: string, dest_directory: string)
+ execute() : string
CategorizeCommand
- src_directory : string
- threshold_size : int
+ CategorizeCommand (directory: string, threshold_size: int)
+ execute() : string
CountCommand
- directory : string
+ CountCommand (directoı string)
+ execute() : string
DeleteCommand
- file_name : string
- directory : string
+ DeleteCommand (file_name:string, directory:string)
+ execute() : string
SortCommand
- src_directory : string
- criteria : string
+ SortCommand(directory: string, criteria: string)
+ execute() : string
ListCommand
- directory : string
+ ListCommand(directory: string)
+ execute() : string
RenameCommand
- old_name : string
- new_name : string
- directory : string
+ RenameCommand (file_name: string, directory:string)
+ execute() : string
ScriptExecutor
- commands: list
- outputDirectory: string
- config: jsonfile
- threshold_size: int
- max_commands: int
- max_log_files: int
- same_dir: boolean
- output: string
-command_debugger_log_file_path: string
+ ScriptExecutor (config_path: string, outputDirectory: string)
+ load_config(config_path: string)
+ parse_script(script_path: string)
+ execute_script()
+ get_next_log_file_name(log_dir: string, prefix: string): string
+ log_results(results: dictionary, all_passed: boolean, outputDirectory: string)
asks
CommandFactory
+get_command(command_type: string, *args) : Command
creates

Figure 1: UML Project Diagram

To use the concept of object oriented , we first created an **(Command) interface class.** This interface has one main abstract method **(execute)** to be implemented by each of its subclasses. We then created 7 subclasses each implement this interface as, each subclass represent a type of command: **MvLastCommand**, **CategorizeCommand**, **CountCommand**, **DeleteCommand, SortCommand**, **ListCommand**, **RenameCommand.**

To use the concept of command factory which is a creational pattern we created a **(Command Factory)** class, and we created the objects of type commands using the (get_command) method in the factory. In Factory pattern, we create an object without exposing the creation logic and refer to newly created object using a common interface.

Starting the program

To start our program, we used the command line, writing a command like this :

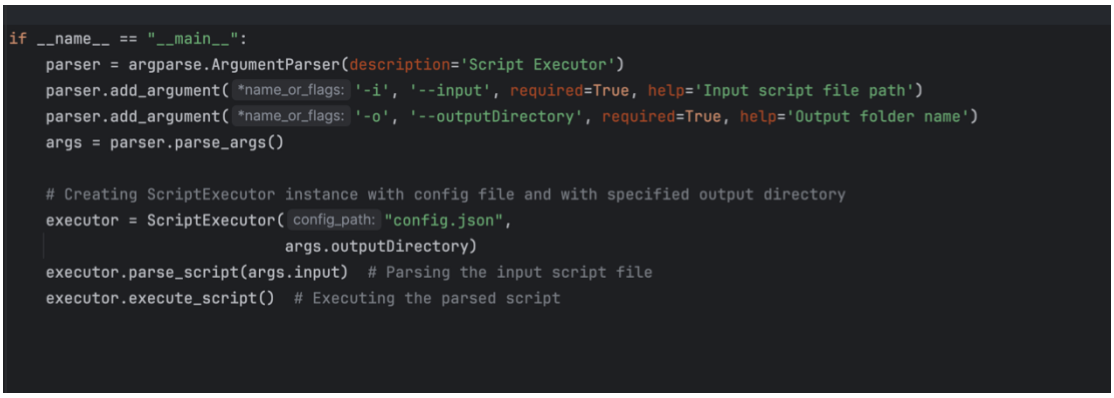
[1]`python3 executorFinal.py -i script3.txt -o outputFile`

In our main we used the argument parser to execute this as shown in (Figure 2: main codeFigure 2), we created a new argument parser and add two argument :

- (-i) argument for the input directory which is the path to the file that has the commands to be executed.
- (-o) argument, which is the name of the output file, which is the file that the log file or csv files will be added to.

After extracting arguments using the argument parser, we created an instance from the Script Executer providing it with the configuration file path (which is a Json file in our program) and also output directory.

We then called the function (parse_script) in the Script Executer object to extract the command and finally called the (execute_script) function to perform the commands.



```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Script Executor')
    parser.add_argument( *name_or_flags: '-i', '--input', required=True, help='Input script file path')
    parser.add_argument( *name_or_flags: '-o', '--outputDirectory', required=True, help='Output folder name')
    args = parser.parse_args()

    # Creating ScriptExecutor instance with config file and with specified output directory
    executor = ScriptExecutor( config_path: "config.json",
                              args.outputDirectory)
    executor.parse_script(args.input)  # Parsing the input script file
    executor.execute_script()  # Executing the parsed script
```

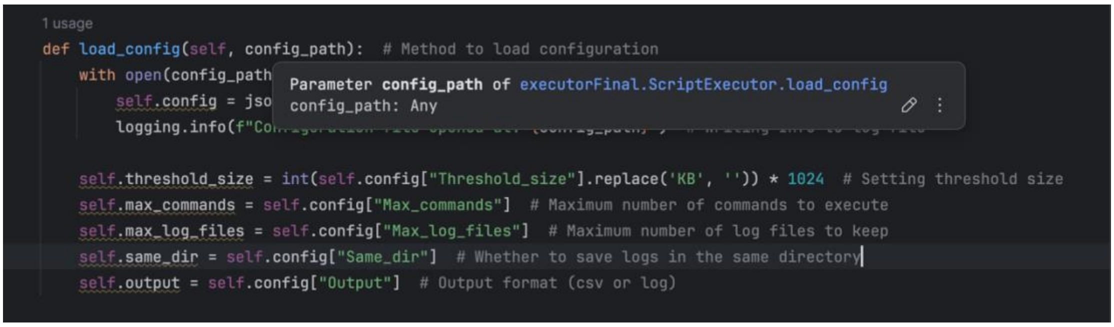
Figure 2: main code

[1] Note : it is assumed that this is run from the same directory that has the script and configuration file, that is why relative path was used (which is just the name), however it will run outside the directory, providing full-path.

Script Executor

To get started with the script executer, in addition to the constructor, two main functions are used :

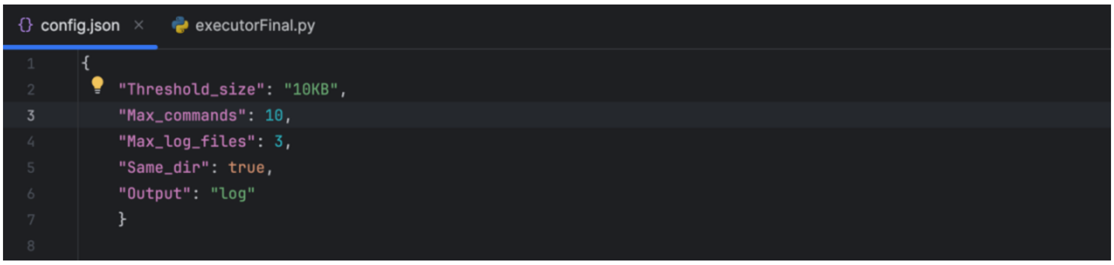
- (load_config) : used to set attributes(threshold_size, max_commands, max_log_files, same_dir, output) from the provided Json file (Figure 4) . The code is shown as follows (Figure 3) :



```python
def load_config(self, config_path):  # Method to load configuration
    with open(config_path
        self.config = jso
        logging.info(f"Co

    self.threshold_size = int(self.config["Threshold_size"].replace('KB', '')) * 1024  # Setting threshold size
    self.max_commands = self.config["Max_commands"]  # Maximum number of commands to execute
    self.max_log_files = self.config["Max_log_files"]  # Maximum number of log files to keep
    self.same_dir = self.config["Same_dir"]  # Whether to save logs in the same directory
    self.output = self.config["Output"]  # Output format (csv or log)
```

```
Parameter config_path of executorFinal.ScriptExecutor.load_config
config_path: Any
```

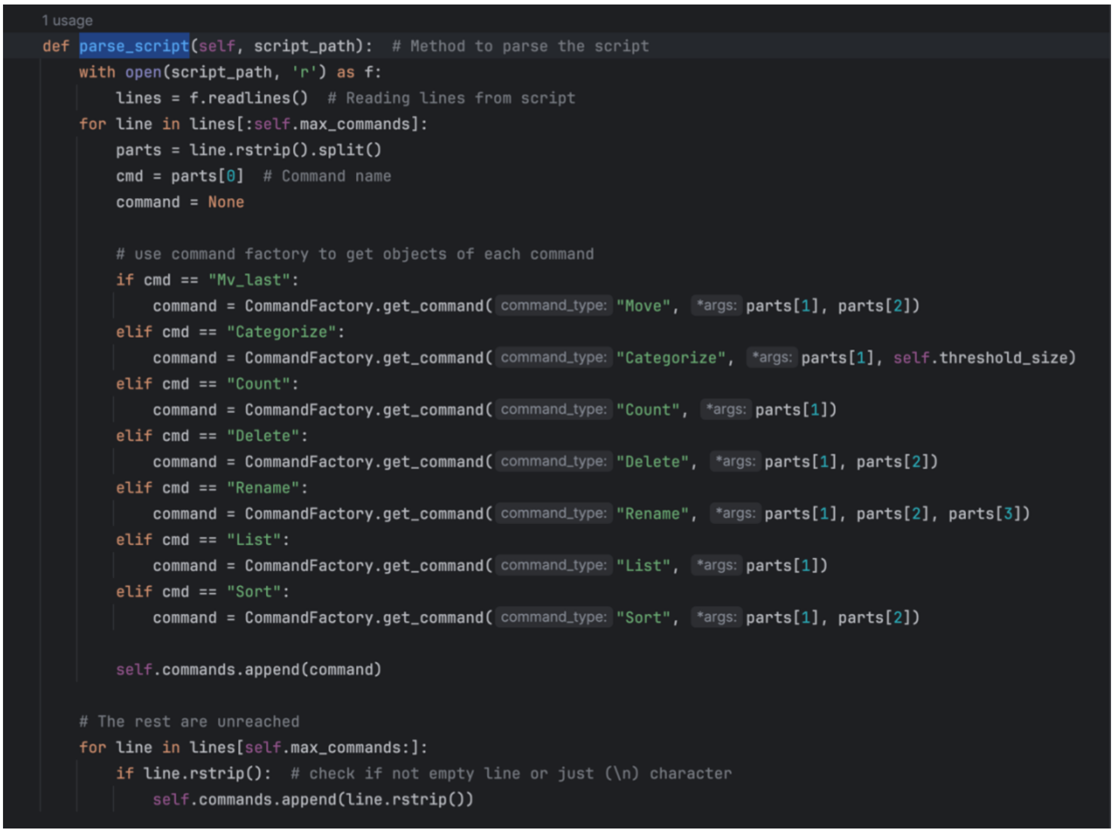
Figure 3: load_config code



```json
{
    "Threshold_size": "10KB",
    "Max_commands": 10,
    "Max_log_files": 3,
    "Same_dir": true,
    "Output": "log"
}
```

Figure 4:JSON file

- (parse_script) : used to parse the script and extract the commands, this function use the command factory to generate the commands and saves them. It starts by reading line by line , it extracts the command name, creates the appropriate command and adds it to command list . It does that until max commands is reached then the rest are added as unreachable lines in the list. The code is shown in (Figure 5):

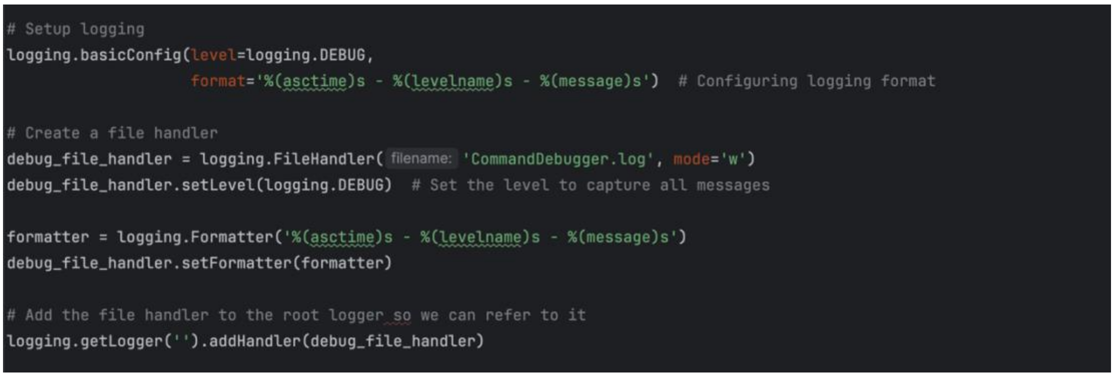


```
1 usage
def parse_script(self, script_path):  # Method to parse the script
    with open(script_path, 'r') as f:
        lines = f.readlines()  # Reading lines from script
    for line in lines[:self.max_commands]:
        parts = line.rstrip().split()
        cmd = parts[0]  # Command name
        command = None

        # use command factory to get objects of each command
        if cmd == "Mv_last":
            command = CommandFactory.get_command( command_type: "Move",  *args: parts[1], parts[2])
        elif cmd == "Categorize":
            command = CommandFactory.get_command( command_type: "Categorize",  *args: parts[1], self.threshold_size)
        elif cmd == "Count":
            command = CommandFactory.get_command( command_type: "Count",  *args: parts[1])
        elif cmd == "Delete":
            command = CommandFactory.get_command( command_type: "Delete",  *args: parts[1], parts[2])
        elif cmd == "Rename":
            command = CommandFactory.get_command( command_type: "Rename",  *args: parts[1], parts[2], parts[3])
        elif cmd == "List":
            command = CommandFactory.get_command( command_type: "List",  *args: parts[1])
        elif cmd == "Sort":
            command = CommandFactory.get_command( command_type: "Sort",  *args: parts[1], parts[2])

        self.commands.append(command)

    # The rest are unreached
    for line in lines[self.max_commands:]:
        if line.rstrip():  # check if not empty line or just (\n) character
            self.commands.append(line.rstrip())
```

Figure 5: parse_script code

Setting up Logging

To make logging easier and more coherent, we set up the logging from the start as follows:



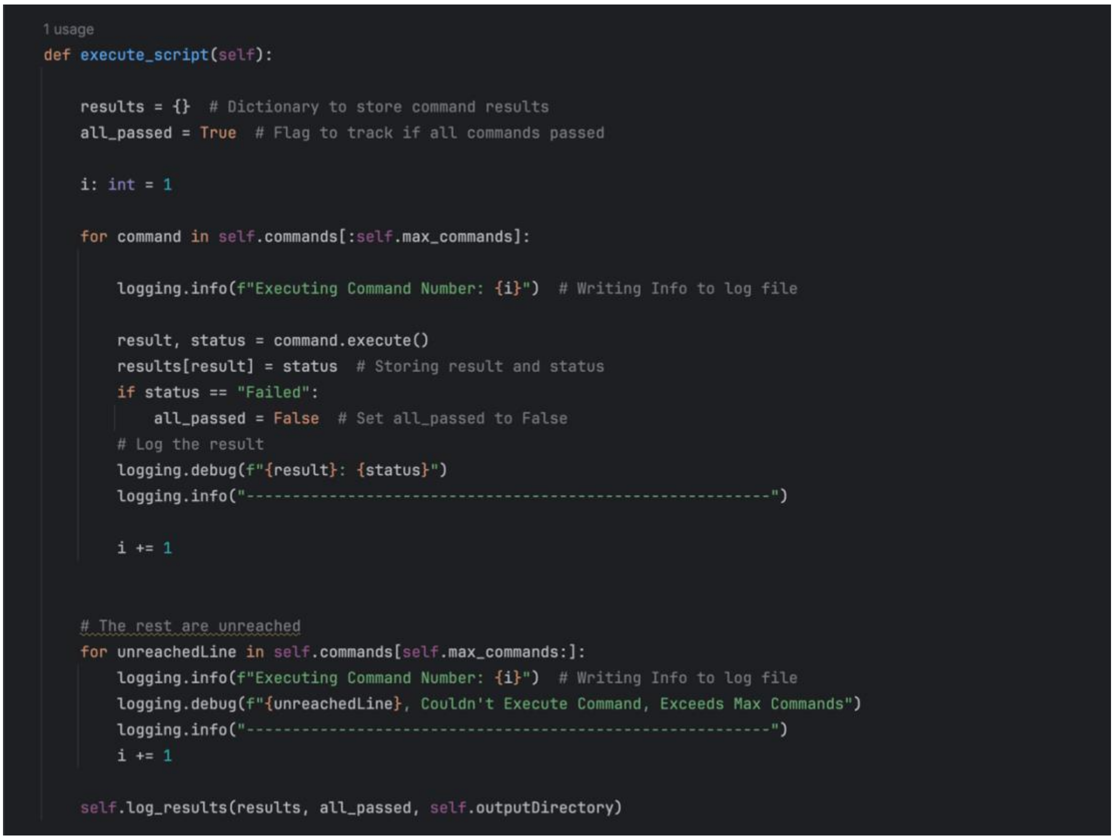
```
# Setup logging
logging.basicConfig(level=logging.DEBUG,
                    format='%(asctime)s - %(levelname)s - %(message)s')  # Configuring logging format

# Create a file handler
debug_file_handler = logging.FileHandler( filename: 'CommandDebugger.log', mode='w')
debug_file_handler.setLevel(logging.DEBUG)  # Set the level to capture all messages

formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
debug_file_handler.setFormatter(formatter)

# Add the file handler to the root logger so we can refer to it
logging.getLogger('').addHandler(debug_file_handler)
```

Figure 6: logging code

We set up a default log file called (commandDebugger.log) that will register information as the program runs. If the output is decided to be log and not csv the contents of this file are used for the output log file.

Command Execution

To execute the script that was parsed, we need to call the (excute_script) function, shown in (Figure 7). The function creates a dictionary that saves each command, with its status. It traverses the command list and calls the (execute) function for each command. It also set a flag (all_passed) to help in logging and creating the appropriate log file later.The unreachable lines are logged as unreached and are not executed therefore their status are not taken into consideration.
After this (log_results) function is being called to create the appropriate log file.



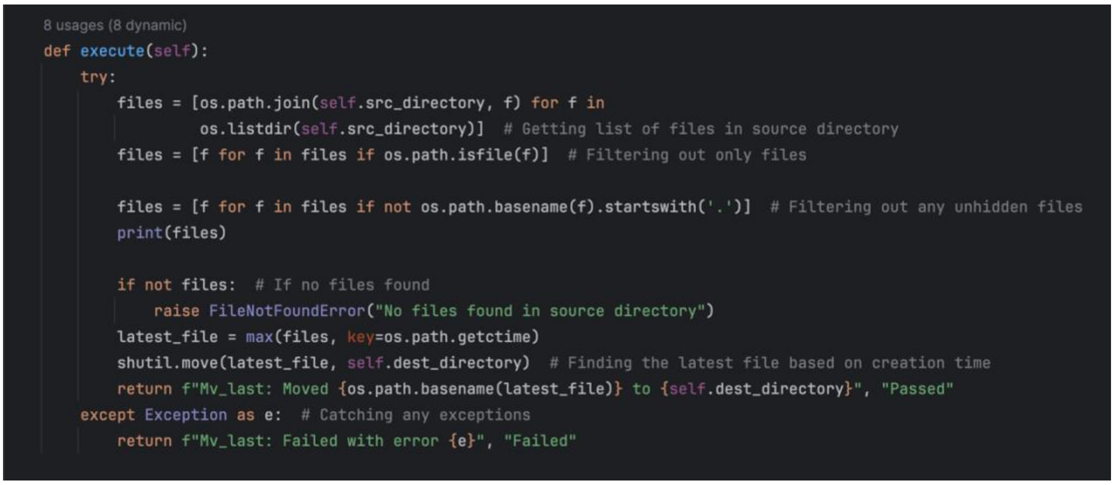
```
1 usage
def execute_script(self):

    results = {}  # Dictionary to store command results
    all_passed = True  # Flag to track if all commands passed

    i: int = 1

    for command in self.commands[:self.max_commands]:

        logging.info(f"Executing Command Number: {i}")  # Writing Info to log file

        result, status = command.execute()
        results[result] = status  # Storing result and status
        if status == "Failed":
            all_passed = False  # Set all_passed to False
        # Log the result
        logging.debug(f"{result}: {status}")
        logging.info("---------------------------------------------------------")

        i += 1


    # The rest are unreached
    for unreachedLine in self.commands[self.max_commands:]:
        logging.info(f"Executing Command Number: {i}")  # Writing Info to log file
        logging.debug(f"{unreachedLine}, Couldn't Execute Command, Exceeds Max Commands")
        logging.info("---------------------------------------------------------")
        i += 1

    self.log_results(results, all_passed, self.outputDirectory)
```

Figure 7: execute_script code

The actual command execution is implemented in the execute function that is written in each of the subclasses. The OS library was used to execute the commands. Below is a brief of how each command was executed :

1) Mv Last Command

To move the last file, we first started by getting the list of files in the source directory, we removed any hidden files (because some errors occurred while we were testing ad there was some hidden files). Then we extracted the last file using (os.path.getctime) and moved this file to the destination file. If all good the status is passed and if any exception happens then it would return a failed as a status. Code shown below.

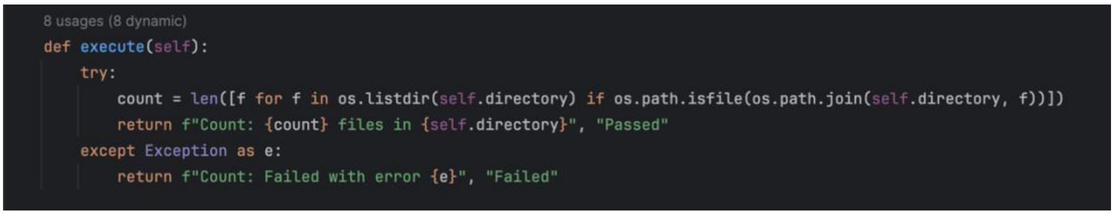


```
8 usages (8 dynamic)
def execute(self):
    try:
        files = [os.path.join(self.src_directory, f) for f in
                 os.listdir(self.src_directory)]  # Getting list of files in source directory
        files = [f for f in files if os.path.isfile(f)]  # Filtering out only files

        files = [f for f in files if not os.path.basename(f).startswith('.')]  # Filtering out any unhidden files
        print(files)

        if not files:  # If no files found
            raise FileNotFoundError("No files found in source directory")
        latest_file = max(files, key=os.path.getctime)
        shutil.move(latest_file, self.dest_directory)  # Finding the latest file based on creation time
        return f"Mv_last: Moved {os.path.basename(latest_file)} to {self.dest_directory}", "Passed"
    except Exception as e:  # Catching any exceptions
        return f"Mv_last: Failed with error {e}", "Failed"
```

Figure 8: Mv_last code

2) Count Command

To count command in a specific path, we first get only the files in the directory and save them in a list and return the length of that list as the count. Code shown below.

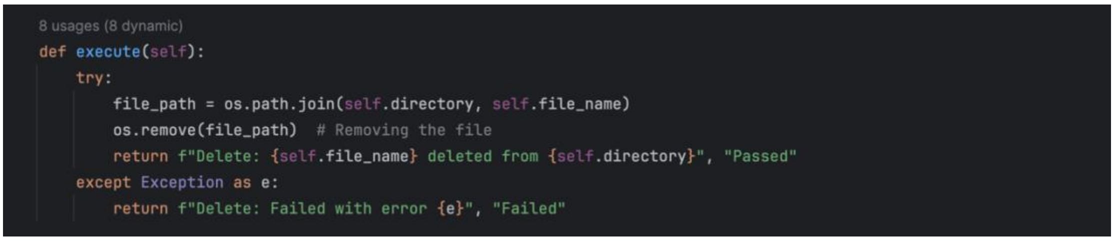


```
8 usages (8 dynamic)
def execute(self):
    try:
        count = len([f for f in os.listdir(self.directory) if os.path.isfile(os.path.join(self.directory, f))])
        return f"Count: {count} files in {self.directory}", "Passed"
    except Exception as e:
        return f"Count: Failed with error {e}", "Failed"
```

Figure 9: Count code

3) Delete Command

To delete a file, we first get the appropriate file name and then use (os.remove)to delete the file. If any exception happens then failed status is returned. Code shown below.

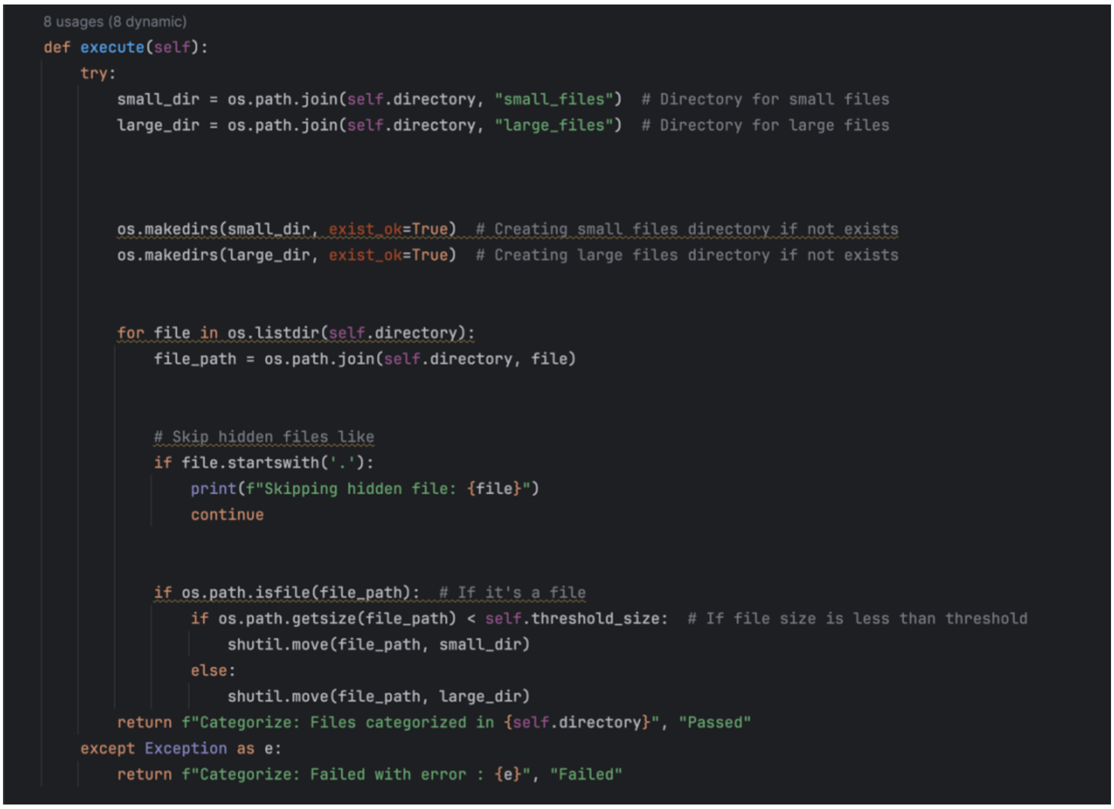


```
8 usages (8 dynamic)
def execute(self):
    try:
        file_path = os.path.join(self.directory, self.file_name)
        os.remove(file_path)  # Removing the file
        return f"Delete: {self.file_name} deleted from {self.directory}", "Passed"
    except Exception as e:
        return f"Delete: Failed with error {e}", "Failed"
```

Figure 10: Delete code

4) Categorize Command

To categorize the files we first started by creating two directories (small and large) in the provided directory if they don’t already exist. We the looped over each file , got the size using (os.path.getsize()) and compared with the threshold and moved to the appropriate category. If any exception happens then failed status is returned.

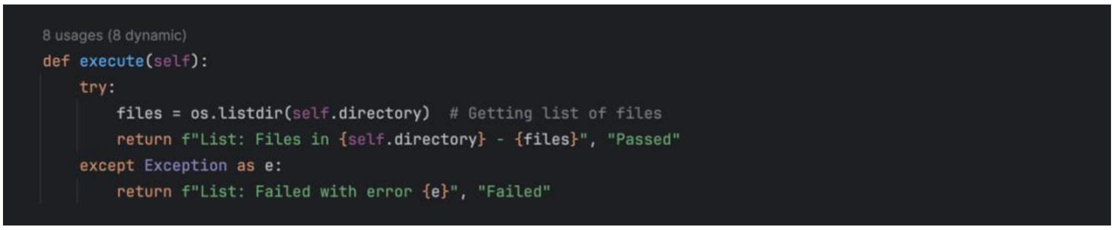


```
8 usages (8 dynamic)
def execute(self):
    try:
        small_dir = os.path.join(self.directory, "small_files")  # Directory for small files
        large_dir = os.path.join(self.directory, "large_files")  # Directory for large files



        os.makedirs(small_dir, exist_ok=True)  # Creating small files directory if not exists
        os.makedirs(large_dir, exist_ok=True)  # Creating large files directory if not exists


        for file in os.listdir(self.directory):
            file_path = os.path.join(self.directory, file)


            # Skip hidden files like
            if file.startswith('.'):
                print(f"Skipping hidden file: {file}")
                continue


            if os.path.isfile(file_path):  # If it's a file
                if os.path.getsize(file_path) < self.threshold_size:  # If file size is less than threshold
                    shutil.move(file_path, small_dir)
                else:
                    shutil.move(file_path, large_dir)
        return f"Categorize: Files categorized in {self.directory}", "Passed"
    except Exception as e:
        return f"Categorize: Failed with error : {e}", "Failed"
```

Figure 11: Categorize code

5) List Command

The list command use (os.listdir()) to list all the files in the directory.

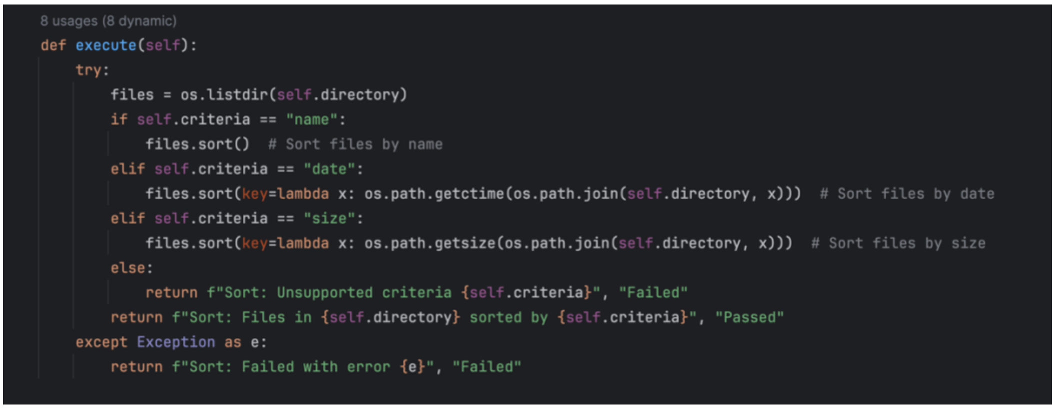


```
8 usages (8 dynamic)
def execute(self):
    try:
        files = os.listdir(self.directory)  # Getting list of files
        return f"List: Files in {self.directory} - {files}", "Passed"
    except Exception as e:
        return f"List: Failed with error {e}", "Failed"
```

Figure 12: list code

6) Sort Command

The sort of command has 3 options, sort be name, date or size. To start the sorting we first list the files using (os.listdir()) and then we sort them using sort method.

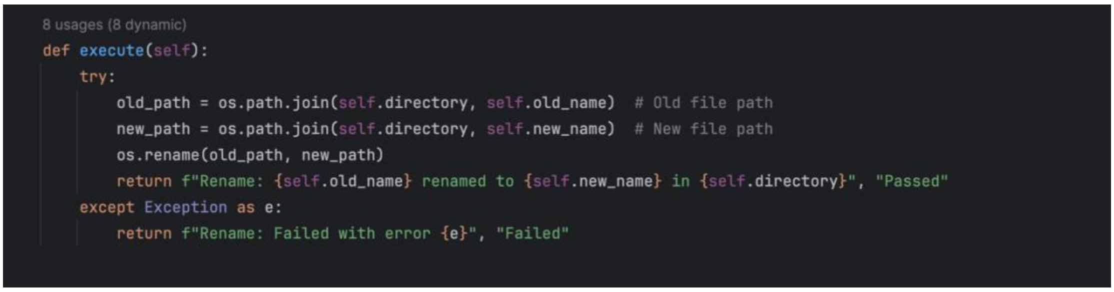


```
8 usages (8 dynamic)
def execute(self):
    try:
        files = os.listdir(self.directory)
        if self.criteria == "name":
            files.sort()  # Sort files by name
        elif self.criteria == "date":
            files.sort(key=lambda x: os.path.getctime(os.path.join(self.directory, x)))  # Sort files by date
        elif self.criteria == "size":
            files.sort(key=lambda x: os.path.getsize(os.path.join(self.directory, x)))  # Sort files by size
        else:
            return f"Sort: Unsupported criteria {self.criteria}", "Failed"
        return f"Sort: Files in {self.directory} sorted by {self.criteria}", "Passed"
    except Exception as e:
        return f"Sort: Failed with error {e}", "Failed"
```

Figure 13: Sort code

7) Rename Command.

The rename starts by joining the appropriate names for the old path and the new path and then uses (os.rename()) to rename the file.



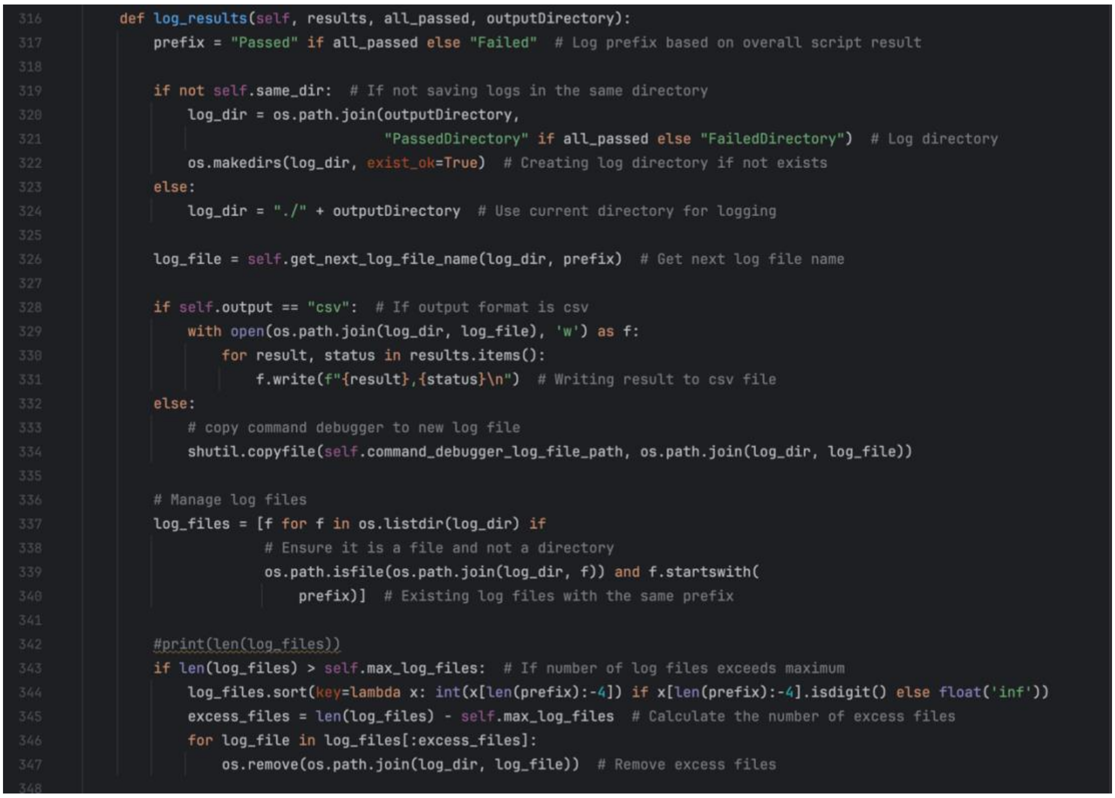
```
8 usages (8 dynamic)
def execute(self):
    try:
        old_path = os.path.join(self.directory, self.old_name)  # Old file path
        new_path = os.path.join(self.directory, self.new_name)  # New file path
        os.rename(old_path, new_path)
        return f"Rename: {self.old_name} renamed to {self.new_name} in {self.directory}", "Passed"
    except Exception as e:
        return f"Rename: Failed with error {e}", "Failed"
```

Figure 14: rename code

Logging results

To log results we use the function log_results, which accepts the dictionary of the results that was created in the (excute_script), it all accepts all_passed flag and the output directory to write to. The file name has the following format [Prefix, number, extension]:

- The (all_passed) will present the prefix (the first part of the file name).
- Using the pre-defined configuration, the path will either be the same as the output directory or a subdirectory inside it.
- The file that will be created will either have a (.csv) extension or (.log) based on the configuration. If the csv file is chosen it will write the command followed by the pass/fail. If log file it will copy contents of the main CommandDebugger.log file that was created at the start.
- The number of the file is determined by the (get_next_log_file_name) function.

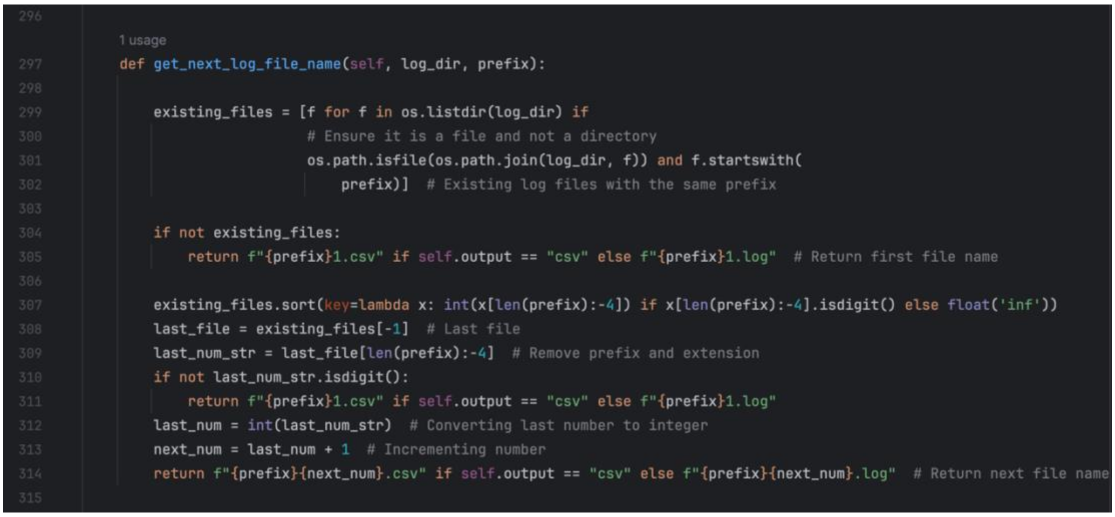


```
    def log_results(self, results, all_passed, outputDirectory):
        prefix = "Passed" if all_passed else "Failed"  # Log prefix based on overall script result

        if not self.same_dir:  # If not saving logs in the same directory
            log_dir = os.path.join(outputDirectory,
                                   "PassedDirectory" if all_passed else "FailedDirectory")  # Log directory
            os.makedirs(log_dir, exist_ok=True)  # Creating log directory if not exists
        else:
            log_dir = "./" + outputDirectory  # Use current directory for logging

        log_file = self.get_next_log_file_name(log_dir, prefix)  # Get next log file name

        if self.output == "csv":  # If output format is csv
            with open(os.path.join(log_dir, log_file), 'w') as f:
                for result, status in results.items():
                    f.write(f"{result},{status}\n")  # Writing result to csv file
        else:
            # copy command debugger to new log file
            shutil.copyfile(self.command_debugger_log_file_path, os.path.join(log_dir, log_file))

        # Manage log files
        log_files = [f for f in os.listdir(log_dir) if
                     # Ensure it is a file and not a directory
                     os.path.isfile(os.path.join(log_dir, f)) and f.startswith(
                         prefix)]  # Existing log files with the same prefix

        #print(len(log_files))
        if len(log_files) > self.max_log_files:  # If number of log files exceeds maximum
            log_files.sort(key=lambda x: int(x[len(prefix):-4]) if x[len(prefix):-4].isdigit() else float('inf'))
            excess_files = len(log_files) - self.max_log_files  # Calculate the number of excess files
            for log_file in log_files[:excess_files]:
                os.remove(os.path.join(log_dir, log_file))  # Remove excess files
```

After creating the appropriate log file, we managed log files . If the number of log files that starts with the prefix is larger than max files then earlier files will be deleted. The files were sorted based on the number after the prefix and not alphabetically (it might have error if numbers go to 10).

The get_next_log_file_name, returns the file name based on the given constraints. It counts the number of files that starts with the same prefix, it sorts them based on number then it gets the last file and extracts its number. After extracting the number it returns the appropriate name.

If at any time there are no existing it return the name with number 1.

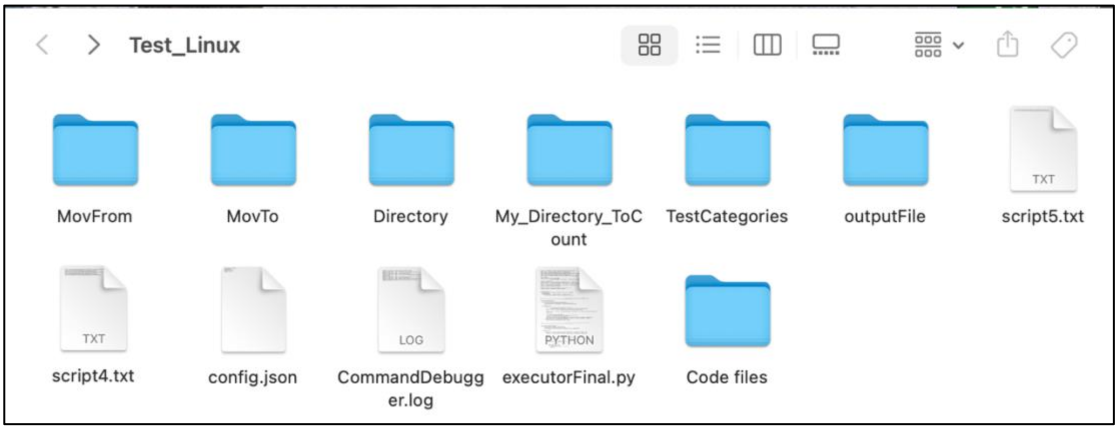


```python
def get_next_log_file_name(self, log_dir, prefix):

    existing_files = [f for f in os.listdir(log_dir) if
                      # Ensure it is a file and not a directory
                      os.path.isfile(os.path.join(log_dir, f)) and f.startswith(
                          prefix)]  # Existing log files with the same prefix

    if not existing_files:
        return f"{prefix}1.csv" if self.output == "csv" else f"{prefix}1.log"  # Return first file name

    existing_files.sort(key=lambda x: int(x[len(prefix):-4]) if x[len(prefix):-4].isdigit() else float('inf'))
    last_file = existing_files[-1]  # Last file
    last_num_str = last_file[len(prefix):-4]  # Remove prefix and extension
    if not last_num_str.isdigit():
        return f"{prefix}1.csv" if self.output == "csv" else f"{prefix}1.log"
    last_num = int(last_num_str)  # Converting last number to integer
    next_num = last_num + 1  # Incrementing number
    return f"{prefix}{next_num}.csv" if self.output == "csv" else f"{prefix}{next_num}.log"  # Return next file name
```

Test cases and running examples

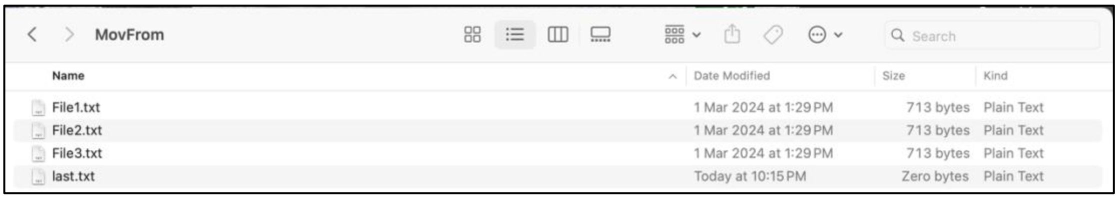
Step 1 :Before Running :

For this program this is the current working directory, with a look inside the sub directories:



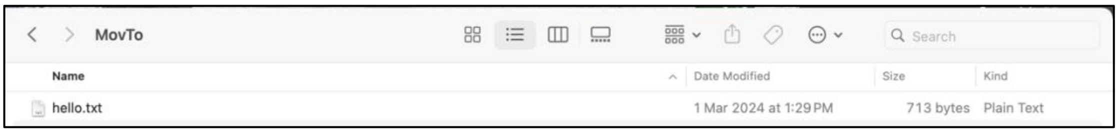
Test_Linux

MovFrom, MovTo, Directory, My_Directory_ToCount, TestCategories, outputFile, script5.txt, script4.txt, config.json, CommandDebugger.log, executorFinal.py, Code files



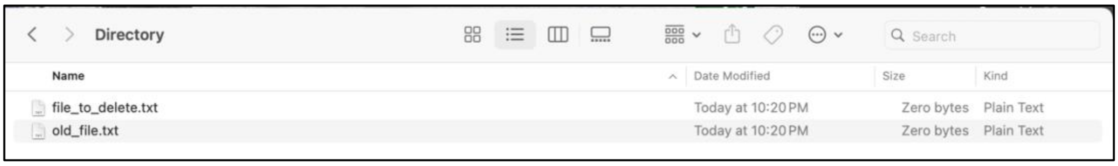
MovFrom

Name	Date Modified	Size	Kind
File1.txt	1 Mar 2024 at 1:29 PM	713 bytes	Plain Text
File2.txt	1 Mar 2024 at 1:29 PM	713 bytes	Plain Text
File3.txt	1 Mar 2024 at 1:29 PM	713 bytes	Plain Text
last.txt	Today at 10:15 PM	Zero bytes	Plain Text



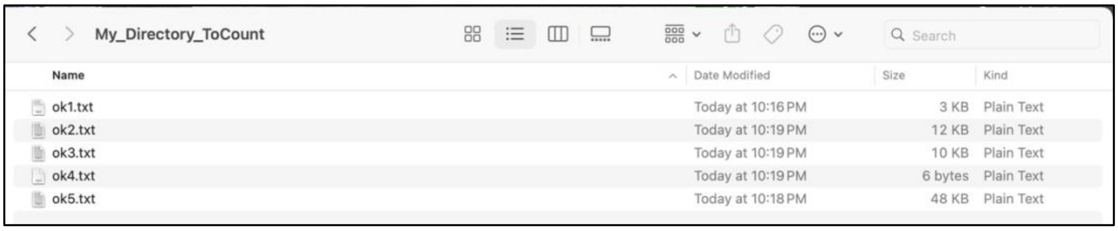
MovTo

Name	Date Modified	Size	Kind
hello.txt	1 Mar 2024 at 1:29 PM	713 bytes	Plain Text



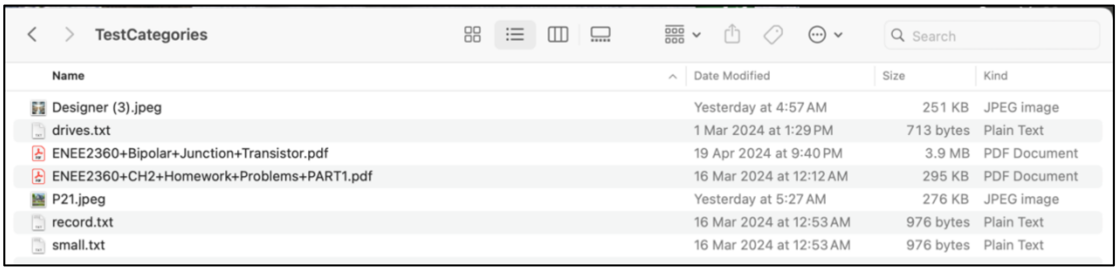
Directory

Name	Date Modified	Size	Kind
file_to_delete.txt	Today at 10:20 PM	Zero bytes	Plain Text
old_file.txt	Today at 10:20 PM	Zero bytes	Plain Text



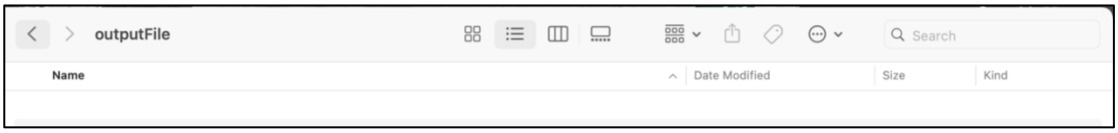
My_Directory_ToCount

Name	Date Modified	Size	Kind
ok1.txt	Today at 10:16 PM	3 KB	Plain Text
ok2.txt	Today at 10:19 PM	12 KB	Plain Text
ok3.txt	Today at 10:19 PM	10 KB	Plain Text
ok4.txt	Today at 10:19 PM	6 bytes	Plain Text
ok5.txt	Today at 10:18 PM	48 KB	Plain Text



TestCategories

Name	Date Modified	Size	Kind
Designer (3).jpeg	Yesterday at 4:57 AM	251 KB	JPEG image
drives.txt	1 Mar 2024 at 1:29 PM	713 bytes	Plain Text
ENEE2360+Bipolar+Junction+Transistor.pdf	19 Apr 2024 at 9:40 PM	3.9 MB	PDF Document
ENEE2360+CH2+Homework+Problems+PART1.pdf	16 Mar 2024 at 12:12 AM	295 KB	PDF Document
P21.jpeg	Yesterday at 5:27 AM	276 KB	JPEG image
record.txt	16 Mar 2024 at 12:53 AM	976 bytes	Plain Text
small.txt	16 Mar 2024 at 12:53 AM	976 bytes	Plain Text

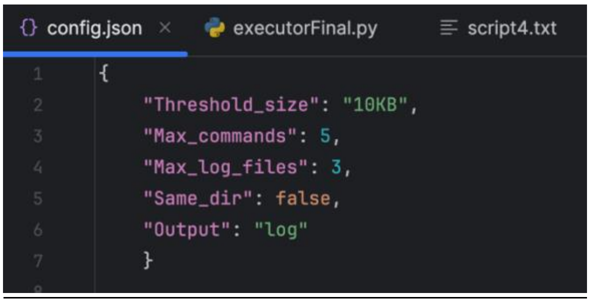


outputFile

Name	Date Modified	Size	Kind

Step 2 : JSON file configuration

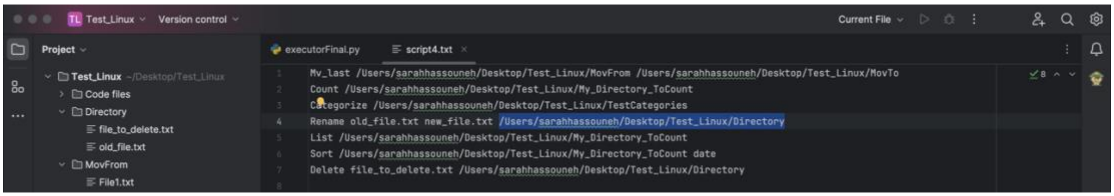
For the first run we started by setting the JSON file configurations as follows:



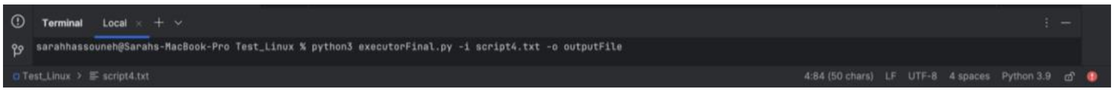
```
{
    "Threshold_size": "10KB",
    "Max_commands": 5,
    "Max_log_files": 3,
    "Same_dir": false,
    "Output": "log"
}
```

Step 3 : Running First Script

The following picture shows the (script4.txt) which we want to run, at the bottoms shows the command used at the terminal and at the side shows the directory (Test Linux) right before running:



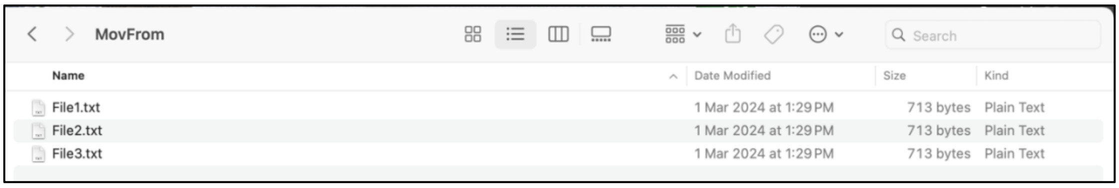
```
Mv_last /Users/sarahhassouneh/Desktop/Test_Linux/MovFrom /Users/sarahhassouneh/Desktop/Test_Linux/MovTo
Count /Users/sarahhassouneh/Desktop/Test_Linux/My_Directory_ToCount
Categorize /Users/sarahhassouneh/Desktop/Test_Linux/TestCategories
Rename old_file.txt new_file.txt /Users/sarahhassouneh/Desktop/Test_Linux/Directory
List /Users/sarahhassouneh/Desktop/Test_Linux/My_Directory_ToCount
Sort /Users/sarahhassouneh/Desktop/Test_Linux/My_Directory_ToCount date
Delete file_to_delete.txt /Users/sarahhassouneh/Desktop/Test_Linux/Directory
```



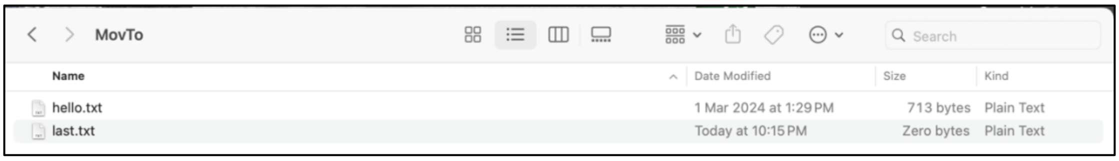
```
sarahhassouneh@Sarahs-MacBook-Pro Test_Linux % python3 executorFinal.py -i script4.txt -o outputFile
```

Now after running the first script we notice these changes on the sub directories of the current directory :

1) The (last.txt) file which is the last file in the (MovFrom) was moved successfully to the (MovTo) directory.

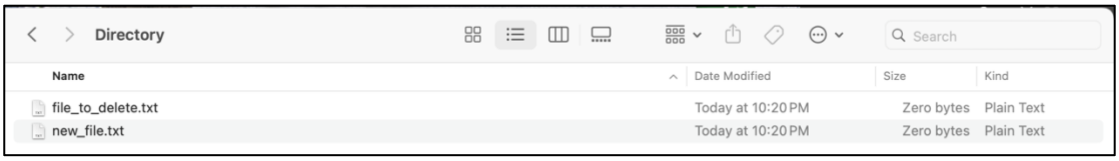


Name	Date Modified	Size	Kind
File1.txt	1 Mar 2024 at 1:29 PM	713 bytes	Plain Text
File2.txt	1 Mar 2024 at 1:29 PM	713 bytes	Plain Text
File3.txt	1 Mar 2024 at 1:29 PM	713 bytes	Plain Text



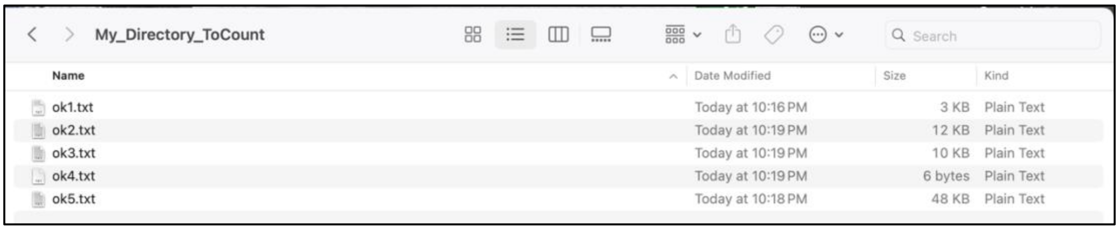
Name	Date Modified	Size	Kind
hello.txt	1 Mar 2024 at 1:29 PM	713 bytes	Plain Text
last.txt	Today at 10:15 PM	Zero bytes	Plain Text

2) The (old_file.txt) file was renamed to (new_file.txt) in the (Directory) Folder.



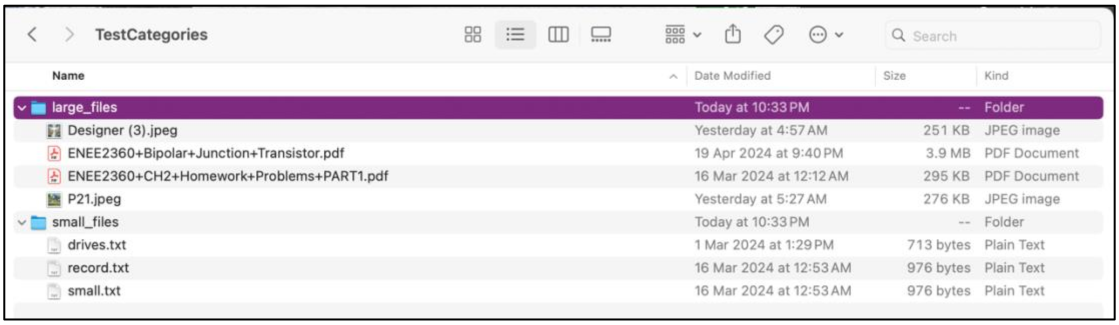
Name	Date Modified	Size	Kind
file_to_delete.txt	Today at 10:20 PM	Zero bytes	Plain Text
new_file.txt	Today at 10:20 PM	Zero bytes	Plain Text

3) This directory was not sorted because max commands is 5 and sort is the 6th.



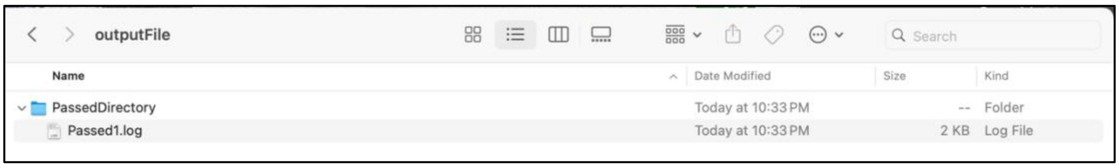
Name	Date Modified	Size	Kind
ok1.txt	Today at 10:16 PM	3 KB	Plain Text
ok2.txt	Today at 10:19 PM	12 KB	Plain Text
ok3.txt	Today at 10:19 PM	10 KB	Plain Text
ok4.txt	Today at 10:19 PM	6 bytes	Plain Text
ok5.txt	Today at 10:18 PM	48 KB	Plain Text

4) This Test Categories folder was successfully categorized to small and large files as follows.



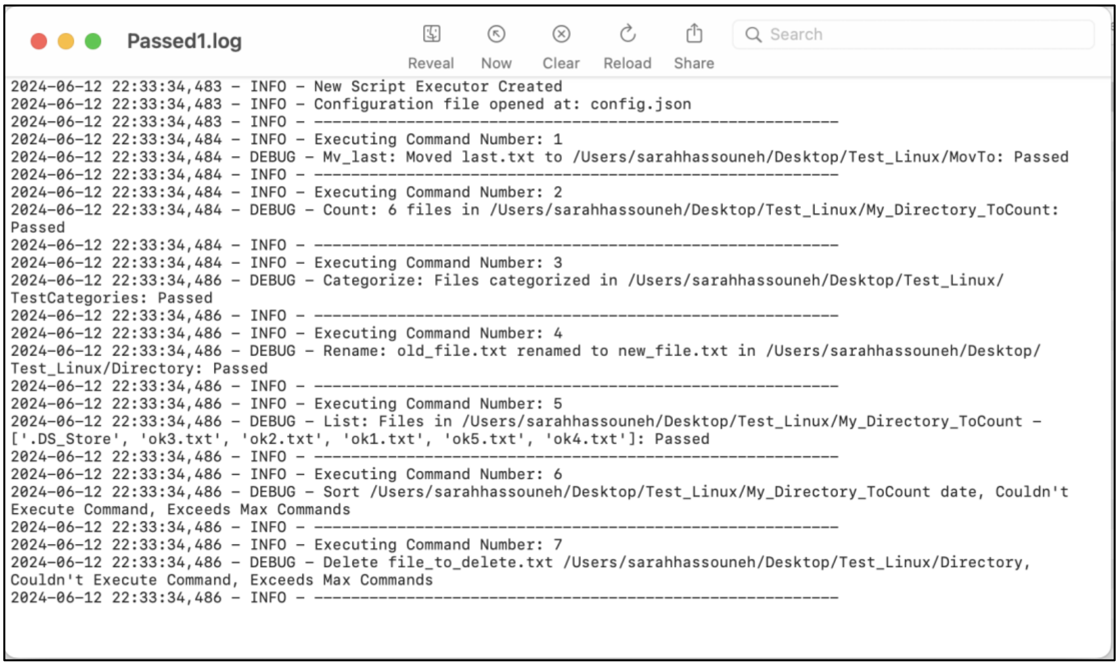
Name	Date Modified	Size	Kind
large_files	Today at 10:33 PM	--	Folder
Designer (3).jpeg	Yesterday at 4:57 AM	251 KB	JPEG image
ENEE2360+Bipolar+Junction+Transistor.pdf	19 Apr 2024 at 9:40 PM	3.9 MB	PDF Document
ENEE2360+CH2+Homework+Problems+PART1.pdf	16 Mar 2024 at 12:12 AM	295 KB	PDF Document
P21.jpeg	Yesterday at 5:27 AM	276 KB	JPEG image
small_files	Today at 10:33 PM	--	Folder
drives.txt	1 Mar 2024 at 1:29 PM	713 bytes	Plain Text
record.txt	16 Mar 2024 at 12:53 AM	976 bytes	Plain Text
small.txt	16 Mar 2024 at 12:53 AM	976 bytes	Plain Text

5) A new (Passed1.log) file was created in a subdirectory (Passed), because configuration is (false) and type of file is log.



Name	Date Modified	Size	Kind
PassedDirectory	Today at 10:33 PM	--	Folder
Passed1.log	Today at 10:33 PM	2 KB	Log File

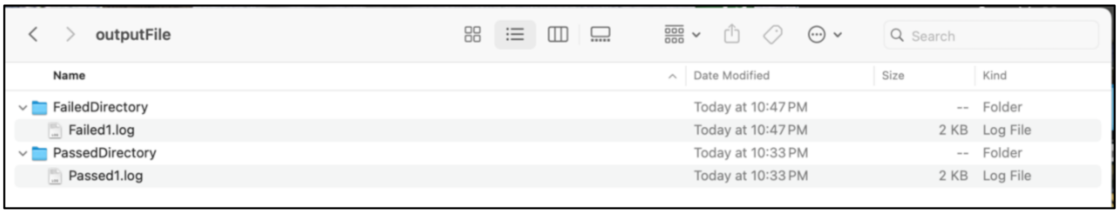
6) The contents of (Passed1.log) is shown here:.



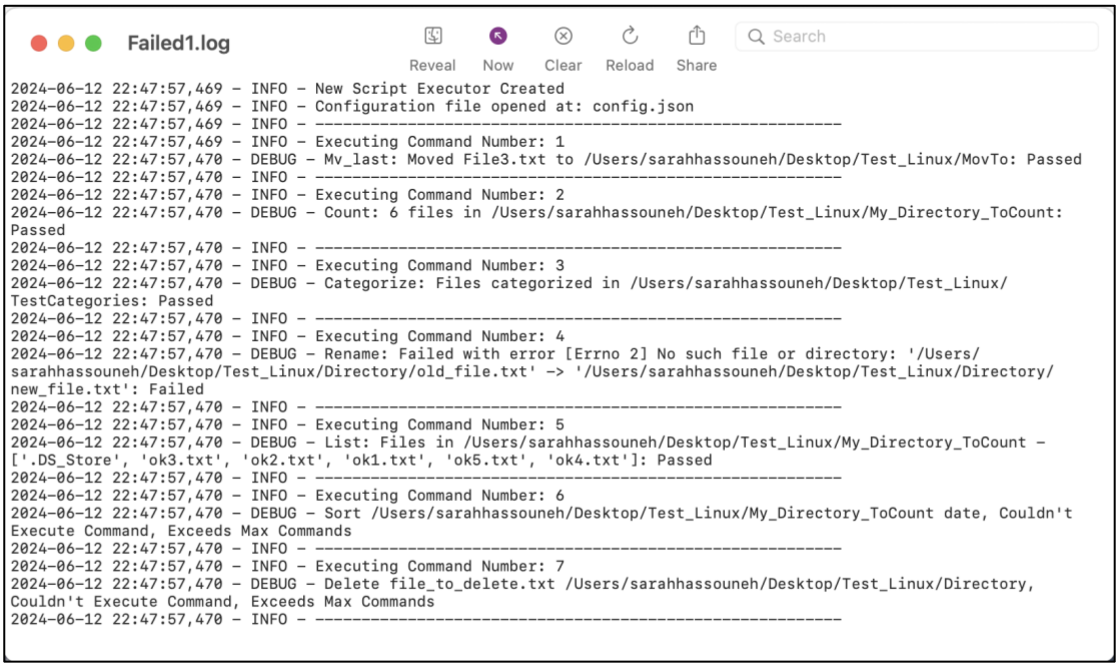
```
2024-06-12 22:33:34,483 - INFO - New Script Executor Created
2024-06-12 22:33:34,483 - INFO - Configuration file opened at: config.json
2024-06-12 22:33:34,483 - INFO - ---------------------------------------------------------
2024-06-12 22:33:34,484 - INFO - Executing Command Number: 1
2024-06-12 22:33:34,484 - DEBUG - Mv_last: Moved last.txt to /Users/sarahhassouneh/Desktop/Test_Linux/MovTo: Passed
2024-06-12 22:33:34,484 - INFO - ---------------------------------------------------------
2024-06-12 22:33:34,484 - INFO - Executing Command Number: 2
2024-06-12 22:33:34,484 - DEBUG - Count: 6 files in /Users/sarahhassouneh/Desktop/Test_Linux/My_Directory_ToCount:
Passed
2024-06-12 22:33:34,484 - INFO - ---------------------------------------------------------
2024-06-12 22:33:34,484 - INFO - Executing Command Number: 3
2024-06-12 22:33:34,486 - DEBUG - Categorize: Files categorized in /Users/sarahhassouneh/Desktop/Test_Linux/
TestCategories: Passed
2024-06-12 22:33:34,486 - INFO - ---------------------------------------------------------
2024-06-12 22:33:34,486 - INFO - Executing Command Number: 4
2024-06-12 22:33:34,486 - DEBUG - Rename: old_file.txt renamed to new_file.txt in /Users/sarahhassouneh/Desktop/
Test_Linux/Directory: Passed
2024-06-12 22:33:34,486 - INFO - ---------------------------------------------------------
2024-06-12 22:33:34,486 - INFO - Executing Command Number: 5
2024-06-12 22:33:34,486 - DEBUG - List: Files in /Users/sarahhassouneh/Desktop/Test_Linux/My_Directory_ToCount -
['.DS_Store', 'ok3.txt', 'ok2.txt', 'ok1.txt', 'ok5.txt', 'ok4.txt']: Passed
2024-06-12 22:33:34,486 - INFO - ---------------------------------------------------------
2024-06-12 22:33:34,486 - INFO - Executing Command Number: 6
2024-06-12 22:33:34,486 - DEBUG - Sort /Users/sarahhassouneh/Desktop/Test_Linux/My_Directory_ToCount date, Couldn't
Execute Command, Exceeds Max Commands
2024-06-12 22:33:34,486 - INFO - ---------------------------------------------------------
2024-06-12 22:33:34,486 - INFO - Executing Command Number: 7
2024-06-12 22:33:34,486 - DEBUG - Delete file_to_delete.txt /Users/sarahhassouneh/Desktop/Test_Linux/Directory,
Couldn't Execute Command, Exceeds Max Commands
2024-06-12 22:33:34,486 - INFO - ---------------------------------------------------------
```

Step 4 : Running First Script Again

If we attempt to run the first script again it will fail(because rename will not find a file to rename), running with the same configuration will result in these main changes:



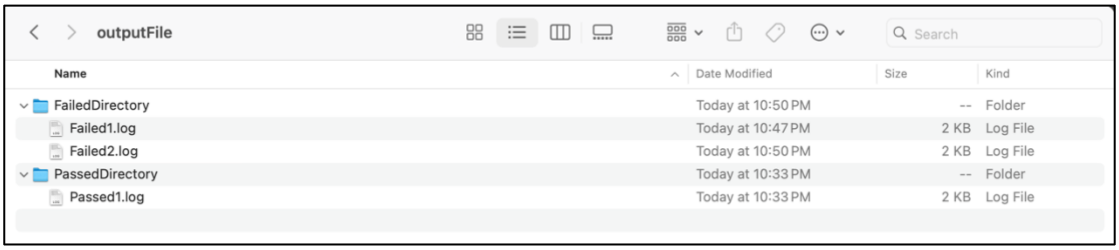
Name	Date Modified	Size	Kind
FailedDirectory	Today at 10:47 PM	--	Folder
Failed1.log	Today at 10:47 PM	2 KB	Log File
PassedDirectory	Today at 10:33 PM	--	Folder
Passed1.log	Today at 10:33 PM	2 KB	Log File



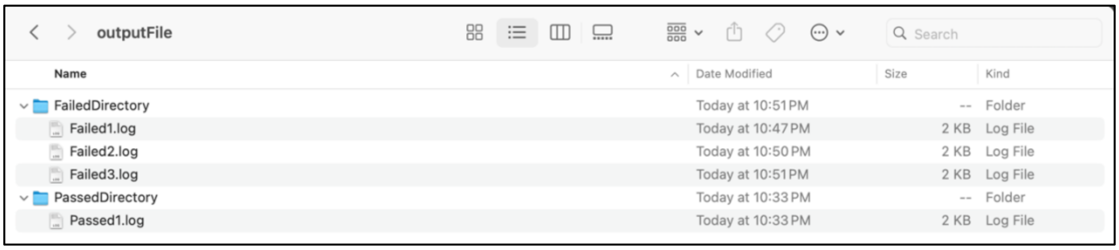
```
2024-06-12 22:47:57,469 - INFO - New Script Executor Created
2024-06-12 22:47:57,469 - INFO - Configuration file opened at: config.json
2024-06-12 22:47:57,469 - INFO - ---------------------------------------------------------
2024-06-12 22:47:57,469 - INFO - Executing Command Number: 1
2024-06-12 22:47:57,470 - DEBUG - Mv_last: Moved File3.txt to /Users/sarahhassouneh/Desktop/Test_Linux/MovTo: Passed
2024-06-12 22:47:57,470 - INFO - ---------------------------------------------------------
2024-06-12 22:47:57,470 - INFO - Executing Command Number: 2
2024-06-12 22:47:57,470 - DEBUG - Count: 6 files in /Users/sarahhassouneh/Desktop/Test_Linux/My_Directory_ToCount:
Passed
2024-06-12 22:47:57,470 - INFO - ---------------------------------------------------------
2024-06-12 22:47:57,470 - INFO - Executing Command Number: 3
2024-06-12 22:47:57,470 - DEBUG - Categorize: Files categorized in /Users/sarahhassouneh/Desktop/Test_Linux/
TestCategories: Passed
2024-06-12 22:47:57,470 - INFO - ---------------------------------------------------------
2024-06-12 22:47:57,470 - INFO - Executing Command Number: 4
2024-06-12 22:47:57,470 - DEBUG - Rename: Failed with error [Errno 2] No such file or directory: '/Users/
sarahhassouneh/Desktop/Test_Linux/Directory/old_file.txt' -> '/Users/sarahhassouneh/Desktop/Test_Linux/Directory/
new_file.txt': Failed
2024-06-12 22:47:57,470 - INFO - ---------------------------------------------------------
2024-06-12 22:47:57,470 - INFO - Executing Command Number: 5
2024-06-12 22:47:57,470 - DEBUG - List: Files in /Users/sarahhassouneh/Desktop/Test_Linux/My_Directory_ToCount -
['.DS_Store', 'ok3.txt', 'ok2.txt', 'ok1.txt', 'ok5.txt', 'ok4.txt']: Passed
2024-06-12 22:47:57,470 - INFO - ---------------------------------------------------------
2024-06-12 22:47:57,470 - INFO - Executing Command Number: 6
2024-06-12 22:47:57,470 - DEBUG - Sort /Users/sarahhassouneh/Desktop/Test_Linux/My_Directory_ToCount date, Couldn't
Execute Command, Exceeds Max Commands
2024-06-12 22:47:57,470 - INFO - ---------------------------------------------------------
2024-06-12 22:47:57,470 - INFO - Executing Command Number: 7
2024-06-12 22:47:57,470 - DEBUG - Delete file_to_delete.txt /Users/sarahhassouneh/Desktop/Test_Linux/Directory,
Couldn't Execute Command, Exceeds Max Commands
2024-06-12 22:47:57,470 - INFO - ---------------------------------------------------------
```

Step 5 : Start decrement of files

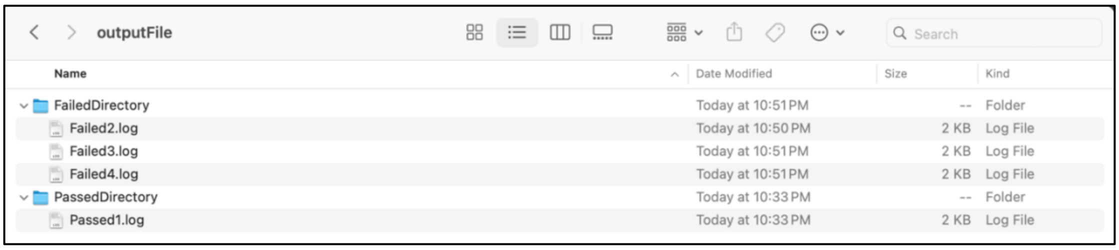
If we attempt to run the first script again it will fail, we can keep executing it until it exceeds max and then the earlier files will start to be deleted as follows. The last picture shows how (Failed1.log) was deleted :



Name	Date Modified	Size	Kind
FailedDirectory	Today at 10:50 PM	--	Folder
Failed1.log	Today at 10:47 PM	2 KB	Log File
Failed2.log	Today at 10:50 PM	2 KB	Log File
PassedDirectory	Today at 10:33 PM	--	Folder
Passed1.log	Today at 10:33 PM	2 KB	Log File



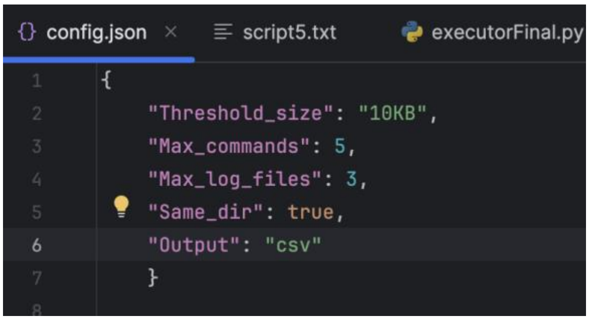
Name	Date Modified	Size	Kind
FailedDirectory	Today at 10:51 PM	--	Folder
Failed1.log	Today at 10:47 PM	2 KB	Log File
Failed2.log	Today at 10:50 PM	2 KB	Log File
Failed3.log	Today at 10:51 PM	2 KB	Log File
PassedDirectory	Today at 10:33 PM	--	Folder
Passed1.log	Today at 10:33 PM	2 KB	Log File



Name	Date Modified	Size	Kind
FailedDirectory	Today at 10:51 PM	--	Folder
Failed2.log	Today at 10:50 PM	2 KB	Log File
Failed3.log	Today at 10:51 PM	2 KB	Log File
Failed4.log	Today at 10:51 PM	2 KB	Log File
PassedDirectory	Today at 10:33 PM	--	Folder
Passed1.log	Today at 10:33 PM	2 KB	Log File

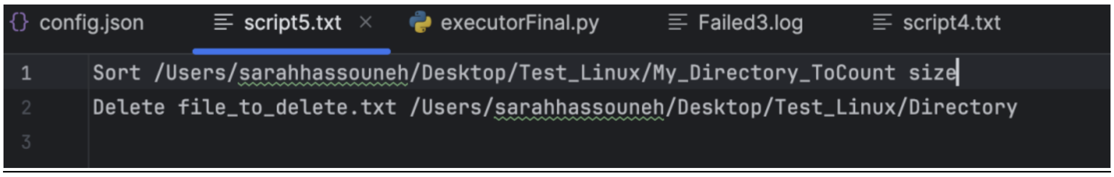
Step 3 : Running Second Script

For the second script we changed the configuration as follows,(csv and true):



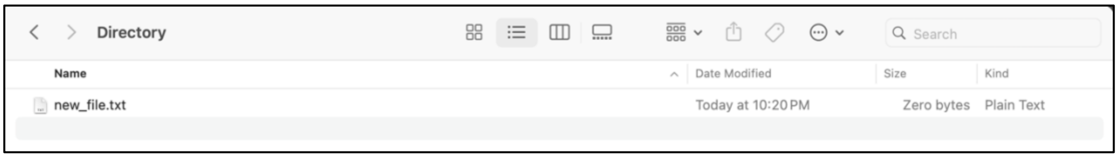
```
{
    "Threshold_size": "10KB",
    "Max_commands": 5,
    "Max_log_files": 3,
    "Same_dir": true,
    "Output": "csv"
    }
```

Now we want to execute this script and notice the changes:



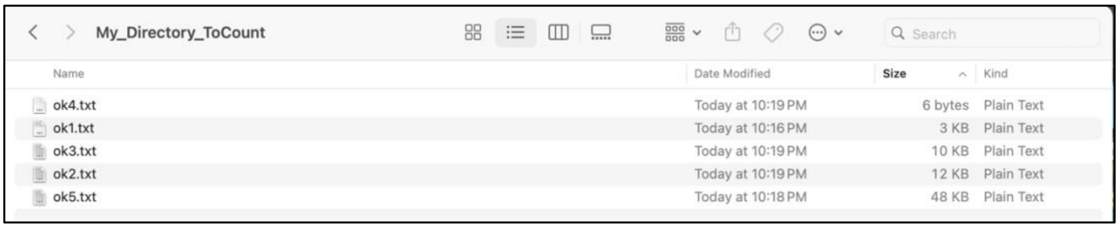
```
Sort /Users/sarahhassouneh/Desktop/Test_Linux/My_Directory_ToCount size
Delete file_to_delete.txt /Users/sarahhassouneh/Desktop/Test_Linux/Directory
```

1) The (file_to_delete.txt) file was successfully removed in the (Directory) Folder.



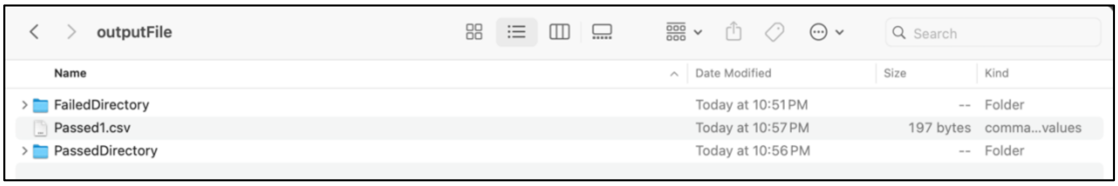
Name	Date Modified	Size	Kind
new_file.txt	Today at 10:20 PM	Zero bytes	Plain Text

2) The (My_directory_ToCount) was sorted based on size from smallest to largest.



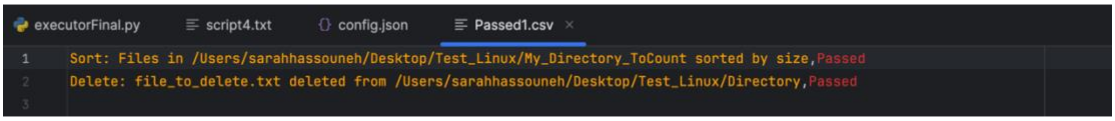
Name	Date Modified	Size	Kind
ok4.txt	Today at 10:19 PM	6 bytes	Plain Text
ok1.txt	Today at 10:16 PM	3 KB	Plain Text
ok3.txt	Today at 10:19 PM	10 KB	Plain Text
ok2.txt	Today at 10:19 PM	12 KB	Plain Text
ok5.txt	Today at 10:18 PM	48 KB	Plain Text

3) A new csv file (Passed1.csv) was created in the same directory.



Name	Date Modified	Size	Kind
FailedDirectory	Today at 10:51 PM	--	Folder
Passed1.csv	Today at 10:57 PM	197 bytes	comma...values
PassedDirectory	Today at 10:56 PM	--	Folder

4) The contents of (Passed1.csv) is shown here:



```
Sort: Files in /Users/sarahhassouneh/Desktop/Test_Linux/My_Directory_ToCount sorted by size,Passed
Delete: file_to_delete.txt deleted from /Users/sarahhassouneh/Desktop/Test_Linux/Directory,Passed
```