Unix-like Shell in Python

1. Introduction

This project implements a Unix-like shell using Python, which supports various commands for file management, system information, and network operations. The shell allows users to interact with the system through a command prompt.

2. Libraries Used

2.1. OS Library

- **Overview**: The os module in Python provides a way of using operating system-dependent functionality like reading or writing to the file system.
- **Key Functions Used**:
  - os.getcwd(): Returns the current working directory as a string.
  - os.listdir(): Returns a list of entries in a specified directory.
  - os.path.isfile(): Checks if a given path is a file.
  - os.path.isdir(): Checks if a given path is a directory.
  - os.remove(): Deletes a specified file.
  - os.system(): Executes a system command (e.g., for clearing the screen).

2.2 shutil Library

- **Overview**: The shutil module offers a higher-level interface for file operations and is particularly useful for copying files and directories.
- **Key Functions Used**:
  - shutil.copy(src, dst): Copies a file from the source path to the destination path.

2.3 socket Library

- **Overview**: The socket module provides a way to create network connections using sockets.
- **Key Functions Used**:
  - socket.socket(): Creates a new socket object.
  - s.connect(address): Establishes a connection to the specified address (used to determine the local IP address).
  - s.getsockname(): Retrieves the local endpoint address of the socket, which can be used to obtain the machine's IP address.

3. Modules Overview

The project consists of three primary modules:

3.1 main.py

- The main.py file serves as the entry point for the shell application.
- Its primary role is to initialize the shell environment and start the command execution loop.

Functionality of main.py:

- When the user runs main.py, the shell prompt (pythonshell>) appears, allowing users to input commands interactively.
- The run() method of the Shell class handles all command processing.

3.2 shell.py

- The shell.py file contains the main logic for the shell. It defines a Shell class that manages user commands.
- When started, the shell prompts the user for input and processes commands like listing files, showing the date, or displaying the current directory.
- It uses a loop to continuously accept commands until the user types exit. The shell also includes error handling to provide feedback if a command is invalid.

run() Method

1. **Command Loop**: The method starts an infinite loop that keeps the shell active, prompting the user for commands until they type exit.
2. **User Input Handling**: It collects user input, removes whitespaces, and splits it into a list to separate the command from its arguments.
3. **Command Execution**: Based on the input command, it uses if-elif-else statements to call the corresponding methods for file management or network management for IP.
4. **Error Management**: The run() method performs error handling to catch issues like invalid commands.
5. **Exit Functionality**: If the user enters exit, the loop breaks, and the shell session ends gracefully.

3.3 file_manager.py

- **Purpose**: Manages file operations within the shell.

- **Methods Overview**:

1. list_files():

- This method returns a list of all files in the current directory by using `os.listdir()` to get entries and filtering them with `os.path.isfile()`.

2. list_dirs():

- This method returns a list of all directories in the current directory, filtering with `os.path.isdir()`.

3. cat_file(filename):

- Opens the specified file in read mode and returns its entire content as a string. If the file cannot be accessed (e.g., it doesn't exist), it returns an error message.

4. head_file(filename, lines=5):

- Reads the first five lines of the specified file and returns them as a single string. It handles errors to ensure a user-friendly response if the file cannot be read.

5. tail_file(filename, lines=5):

- Reads the last five lines of the specified file and returns them as a string. It also includes error handling.

6. copy_file(src, dest):

- Uses `shutil.copy()` to copy a file from the source path to the destination path. It returns a success message in case of successful copy or an error message if the operation fails.

7. remove_file(filename):

- Deletes the specified file using `os.remove(). If the file is successfully deleted, it returns a confirmation message; otherwise, it returns an error message.

8. empty_file(filename):

- Opens the specified file in write mode and erases the content of the file. It returns a message indicating the file has been cleared, or an error if the operation fails.

3.4 network_manager.py

- **Purpose**: Handles network-related functionalities.
- **Methods Overview**:
  1. get_ip_address():
     - This method retrieves the system's local IP address. It creates a socket using socket.socket() and connects to a public address (like Google DNS at 8.8.8.8) to determine the local endpoint. The local IP address is then extracted using s.getsockname()[0]. If any error occurs during this process (e.g., network issues), it returns an error message.

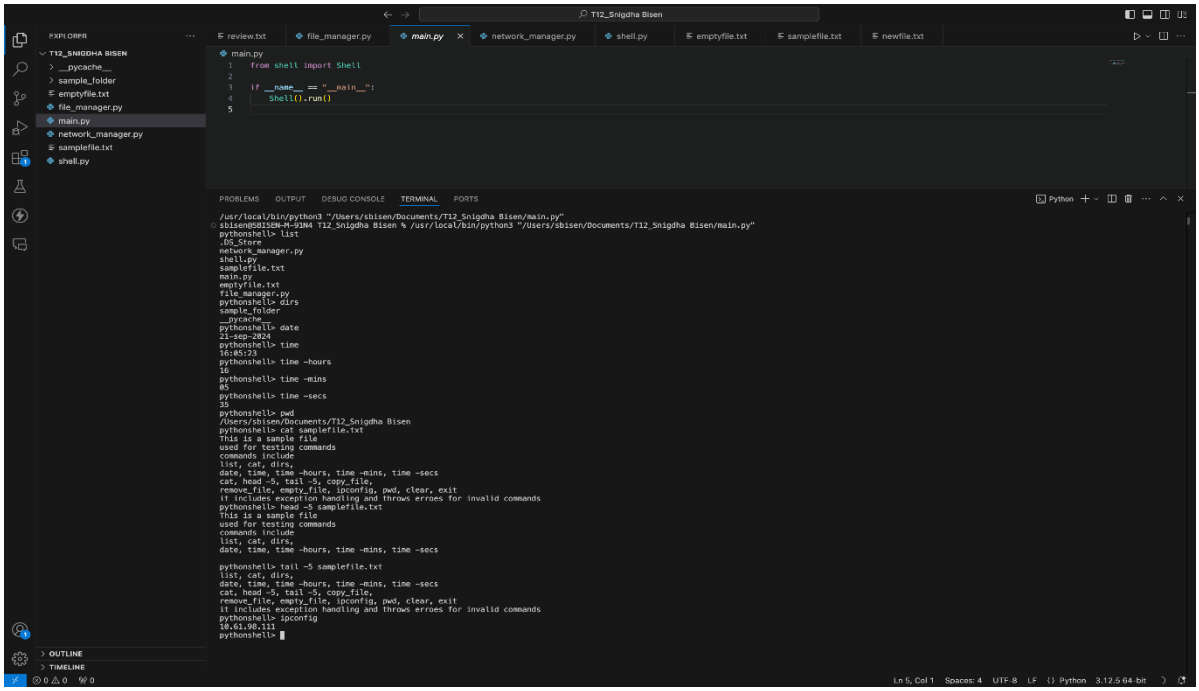
4. Workflow of the Code

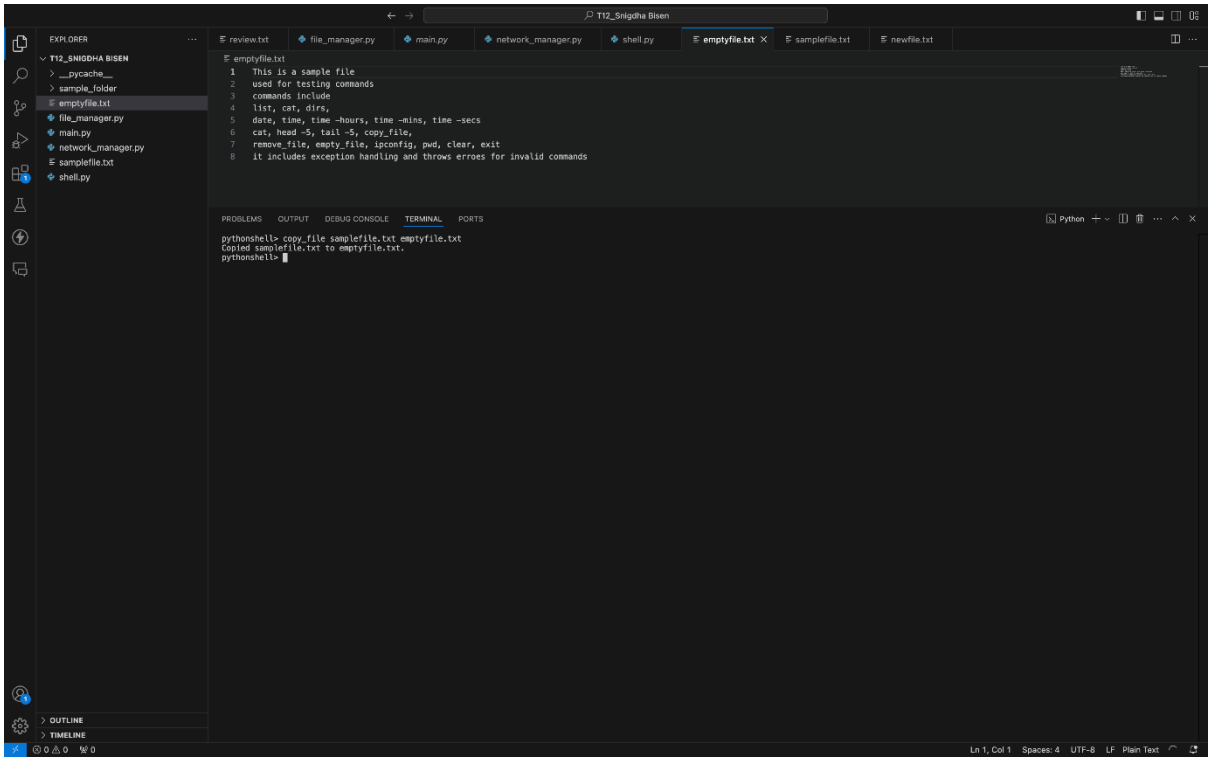
1. **Initialization**: The main.py file is executed, which creates an instance of the Shell class and invokes its run() method.
2. **Command Prompt**: Inside the run() method, a continuous loop prompts the user to enter commands. The command input is split into a list for further processing.
3. **Command Processing**:
   - The first element of the input list is identified as the command.
   - Based on the command entered, the corresponding method from respective class is called.
   - Valid commands include:
     - list: Lists files from current directory.
     - dirs: Lists directories.
     - date: Displays the current date.
     - time: Displays the current time or specific units (hours/minutes/seconds).
     - cat <filename>: Displays file contents.
     - head -5 <filename>: Displays the top five lines of a file.
     - tail -5 <filename>: Displays the last five lines of a file.
     - copy_file <src> <dest>: Copies content from src file to dest file.

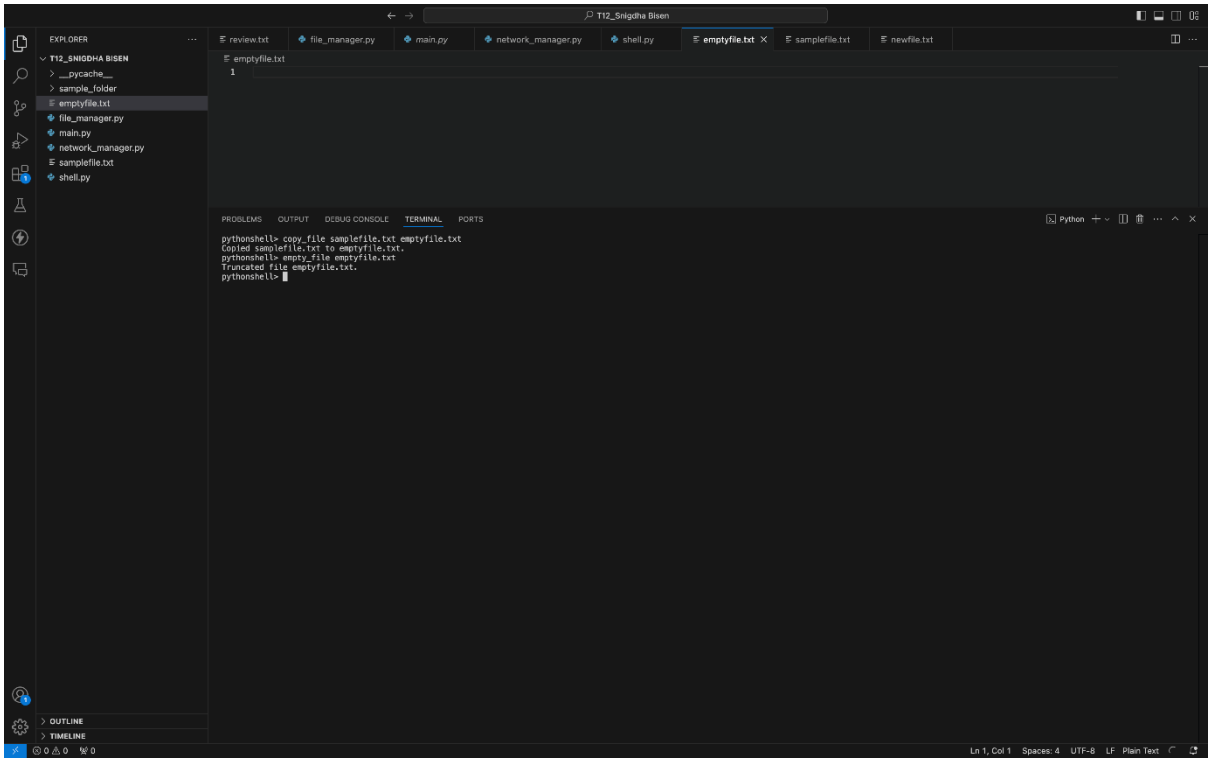
- remove_file <filename>: Deletes a file.
- empty_file <filename>: Empties a file.
- ipconfig: Displays the IP address.
- pwd: Displays the current working directory.
- clear: Clears the console.
- exit: Exits the shell.

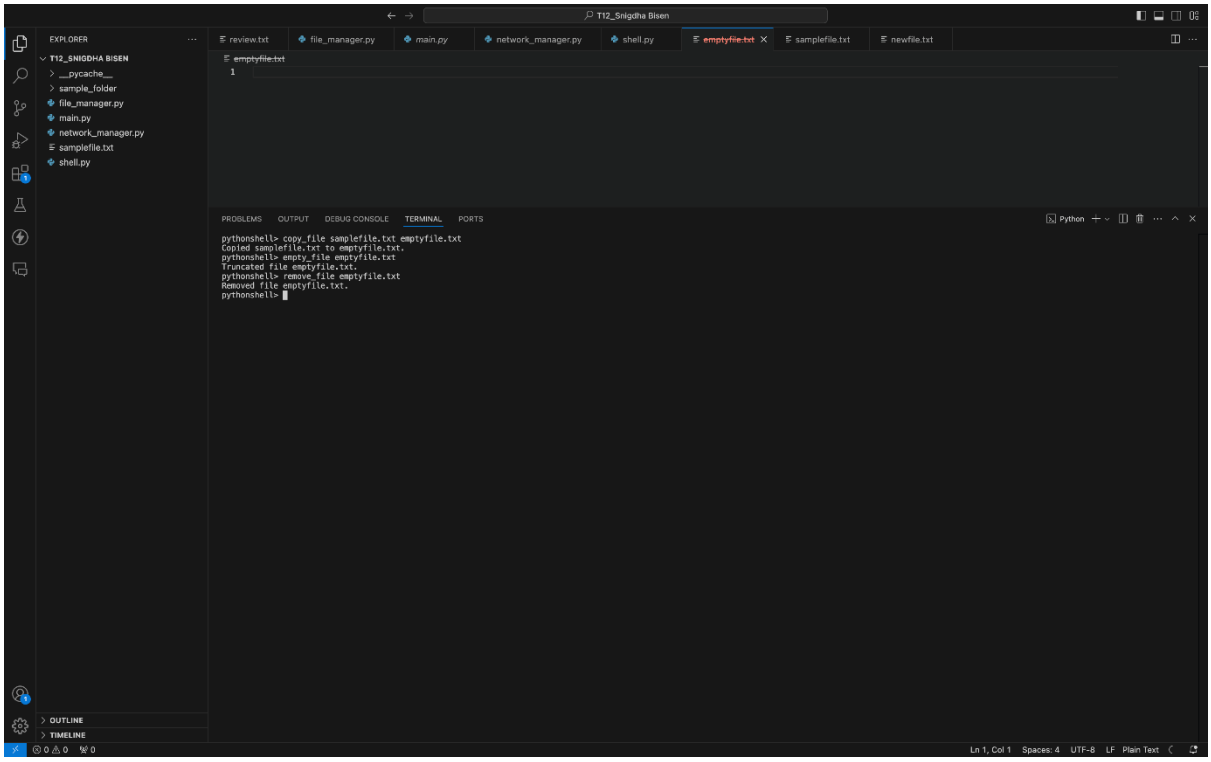
4. **Error Handling**: The shell is designed to be robust, using exception handling to manage errors gracefully, such as invalid commands or file operations.
5. **Exit Mechanism**: The loop continues until the user inputs the exit command, after which the shell terminates.

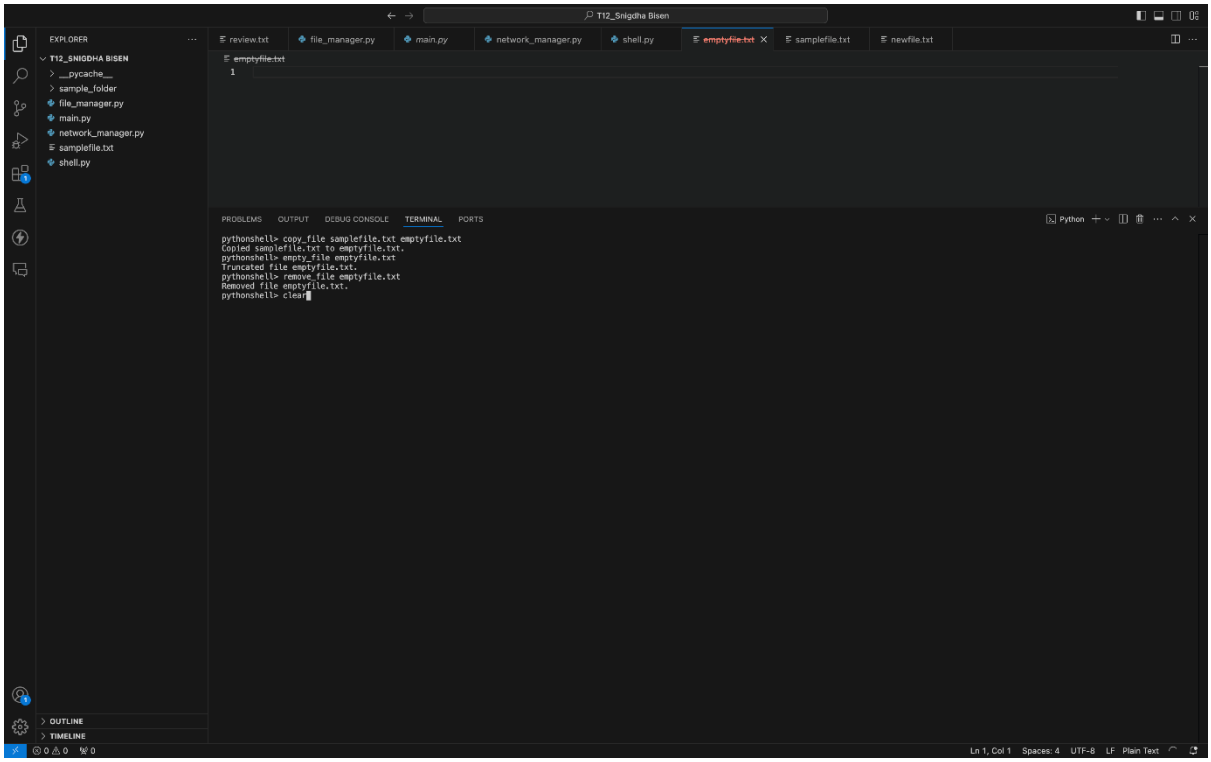
5.Output

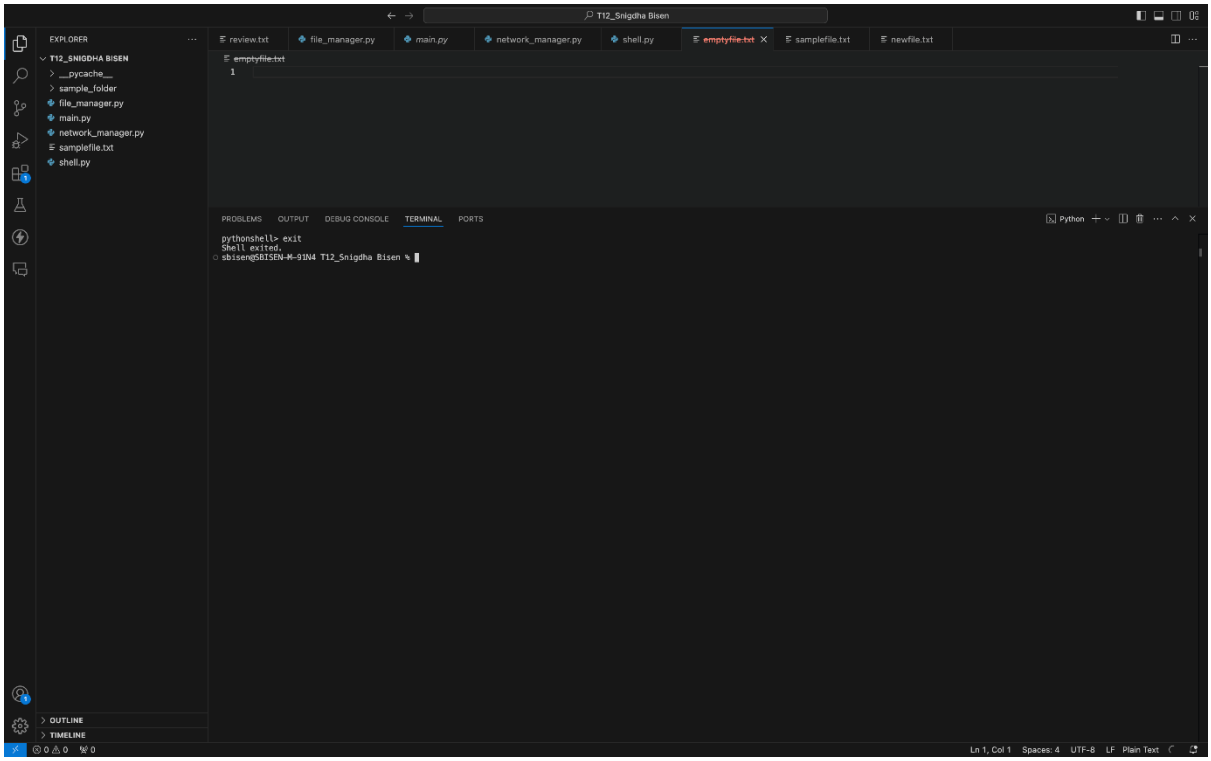



T12_Snigdha Bisen
EXPLORER
T12_SNIGDHA BISEN
__pycache__
sample_folder
emptyfile.txt
file_manager.py
main.py
network_manager.py
samplefile.txt
shell.py
review.txt
file_manager.py
main.py
network_manager.py
shell.py
emptyfile.txt
samplefile.txt
newfile.txt
emptyfile.txt
This is a sample file
used for testing commands
commands include
list, cat, dirs,
date, time, time -hours, time -mins, time -secs
cat, head -5, tail -5, copy_file,
remove_file, empty_file, ipconfig, pwd, clear, exit
it includes exception handling and throws erroes for invalid commands
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
pythonshell> copy_file samplefile.txt emptyfile.txt
Copied samplefile.txt to emptyfile.txt.
pythonshell>
OUTLINE
TIMELINE
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Plain Text


T12_Snigdha Bisen
EXPLORER
T12_SNIGDHA BISEN
__pycache__
sample_folder
emptyfile.txt
file_manager.py
main.py
network_manager.py
samplefile.txt
shell.py
review.txt
file_manager.py
main.py
network_manager.py
shell.py
emptyfile.txt
samplefile.txt
newfile.txt
emptyfile.txt
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
pythonshell> copy_file samplefile.txt emptyfile.txt
Copied samplefile.txt to emptyfile.txt.
pythonshell> empty_file emptyfile.txt
Truncated file emptyfile.txt.
pythonshell>
OUTLINE
TIMELINE
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Plain Text


T12_Snigdha Bisen
EXPLORER
T12_SNIGDHA BISEN
__pycache__
sample_folder
file_manager.py
main.py
network_manager.py
samplefile.txt
shell.py
review.txt
file_manager.py
main.py
network_manager.py
shell.py
emptyfile.txt
samplefile.txt
newfile.txt
emptyfile.txt
1
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
pythonshell> copy_file samplefile.txt emptyfile.txt
Copied samplefile.txt to emptyfile.txt.
pythonshell> empty_file emptyfile.txt
Truncated file emptyfile.txt.
pythonshell> remove_file emptyfile.txt
Removed file emptyfile.txt.
pythonshell>
OUTLINE
TIMELINE
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Plain Text


T12_Snigdha Bisen
EXPLORER
T12_SNIGDHA BISEN
__pycache__
sample_folder
file_manager.py
main.py
network_manager.py
samplefile.txt
shell.py
review.txt
file_manager.py
main.py
network_manager.py
shell.py
emptyfile.txt
samplefile.txt
newfile.txt
emptyfile.txt
1
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
pythonshell> copy_file samplefile.txt emptyfile.txt
Copied samplefile.txt to emptyfile.txt.
pythonshell> empty_file emptyfile.txt
Truncated file emptyfile.txt.
pythonshell> remove_file emptyfile.txt
Removed file emptyfile.txt.
pythonshell> clear
OUTLINE
TIMELINE
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Plain Text



6. Conclusion

The project has successfully created a Unix-like shell in Python that meets all requirements in the problem statement. A command prompt appears to accept user input, and recognizes the specified 14 commands, returning errors for any invalid entries. The shell is case sensitive, ensuring accurate command recognition. Additionally, the program is robust, with exception handling in place to manage errors gracefully, providing a smooth user experience.