

UNIX LIKE SHELL USING PYTHON

Overview

This project implements a Unix like shell in Python that allows users to perform various file operations, system commands, and retrieve date and time information.

1. main.py

1.1 Purpose

Created to serve as the entry point for the shell application.

1.2 Process

1. Import the UnixShell class from shell.py.
2. Create an object of the UnixShell.
3. Call the run() method to start the shell interface.

2. shell.py

2.1 Purpose

Created to define the UnixShell class that handles command parsing, validation, and execution.

2.2 Methods

- **__init__()**: Create a dictionary and linking command names to their corresponding methods in other classes.
- **run()**: Accepts user input and executes the corresponding command. It keeps asking for new commands, checks if the commands are correct, and calls the right function. If there's an error or the command isn't valid, it handles that too.

2.3 Process

1. Initializes command names to methods using dictionary in `__init__()`.
2. Enters a loop in `run()`, asking continuously the user for input until exit
3. Processes input commands, validates them, and calls the respective methods.
4. Prints exceptions for invalid commands.

3. `time_operations.py`

3.1 Purpose

Created to handle commands related to date and time.

3.2 Libraries Used

- `datetime`: For retrieving and formatting date and time.

3.3 Static Methods

- **`display_date()`**: Prints the current date in dd-month-yyyy format, retrieving and formatting the current date.
- **`display_time()`**: Prints the current time in HH:MM:SS format, retrieving and formatting the current time.
- **`display_time_hours()`**: Prints the current hour.
- **`display_time_minutes()`**: Prints the current minute.
- **`display_time_seconds()`**: Prints the current second, with each method retrieving the respective time component.

3.4 Process

1. Each method uses `datetime` to retrieve the current date and time.
2. Formats the date and time as specified.
3. When calling any of the methods to display the date or time, if there's an unexpected issue like system time not being accessible, an exception will be raised and caught.

4. system_operations.py

4.1 Purpose

Created to manage system-related commands.

4.2 Libraries Used

- `os`: For interacting with the operating system.
- `netifaces`: For retrieving network interface information.

4.3 Static Methods

- **`display_ipconfig()`**: Displays the system's IP address by iterating through network interfaces and retrieving IPv4 addresses.
- **`display_pwd()`**: Prints the current working directory using `os.getcwd()` to get the current directory.
- **`clear_screen()`**: Clears the terminal screen using system commands to clear the terminal.

4.4 Process

1. Each method uses the relevant libraries to perform tasks.
2. Catches exceptions related to system calls and network issues.
3. Prints error messages when applicable
4. An exception may occur if there are issues accessing network configurations or if the `netifaces` library fails to retrieve information or if there's an error in accessing its address data

5. file_operations.py

5.1 Purpose

Created to handle commands related to file operations.

5.2 Libraries Used

- `os`: For interacting with the file system.
- `shutil`: For file copying operations.

5.3 Static Methods

- **`list_files()`**: Lists files in the current directory.
- **`list_dirs()`**: Lists directories in the current directory.
- **`cat_file(filename)`**: Displays the contents of a specified file.
- **`head_file(filename)`**: Displays the top 5 lines of a specified file.
- **`tail_file(filename)`**: Displays the last 5 lines of a specified file.
- **`copy_file(src, dest)`**: Copies a file from source to destination.
- **`remove_file(filename)`**: Deletes a specified file.
- **`empty_file(filename)`**: Truncates the contents of a specified file.

5.4 Process

1. Each method performs file operations using `os` to interact with file system and `shutil` to copy the file.
2. Validates the presence of files before performing operations.
3. Executes exceptions such as file not found and prints relevant error messages.

Predefined commands

`list` - list only the files from current directory

`dirs` - list only the directories from current directory

`date` - display system date in the format dd-month-yyyy

`time` - display system time in the format HH:MM:SS

`time -hours` - display only system times number of hours

`time -mins` - display only system times number of minutes

`time -secs` - display only system times number of seconds

cat <filename> - display the contents of text file
head -5 <filename> - display the top 5 lines of text file
tail -5 <filename> - display the last 5 lines of text file
copy_file <src> <dest> - copy source file to destination
remove_file <filename> - delete the file
empty_file <filename> - truncate the file contents - file size set 0
ipconfig - display the system IP ADDRESS
pwd - Present working directory
clear - clear the screen
exit - Shell exits

Overall Workflow

1. Initialization:

- The shell application starts by executing the main.py file.
- An object of the UnixShell class is created, which initializes the commands to respective methods

2. Command Loop:

- The run() method of the UnixShell class is called, entering an infinite loop to continuously prompt the user for input.
- The prompt appears as shell> .

3. User Input:

- The user enters a command, which is taken as a string and split into command and arguments.
- spaces are removed from the input.

4. Command Execution:

- The first element of the input is treated as the command e.g., list, date.
- The remaining elements are treated as arguments.
- The command is checked against a predefined list of valid commands:
 - **Valid Commands:** If the command matches one of the predefined commands, the corresponding method from the appropriate class is called.
 - **Special Cases:** Certain commands like cat, head, and tail require specific argument formats and are handled separately.
 - **Invalid Commands:** If the command does not match any valid options, an error message is displayed.

5. Method Execution:

- Each method executed like listing files, displaying date and time performs its respective operation:
 - **File Operations:** Interacts with the file system to list, read, copy, remove, or empty files.
 - **System Operations:** Retrieves system information like current working directory and network details.
 - **Time Operations:** Gets and formats current date and time.
- Each operation has exception handling to manage potential errors like file not found, permission issues.

6. Error and Exception Handling:

- If an invalid command is entered:
 - The shell prints an error message indicating the command is not recognized.
- If an exception occurs during command execution such as:

- File not found (e.g., for cat, head, tail, remove_file, copy_file)
- Permission issues
- Invalid arguments for commands
- The shell catches the exception and prints an appropriate error message . This ensures a smooth user experience even when unexpected situations arise.

6. Output:

- Results of the executed commands are printed .
- Error messages are displayed if exceptions occur during command execution.

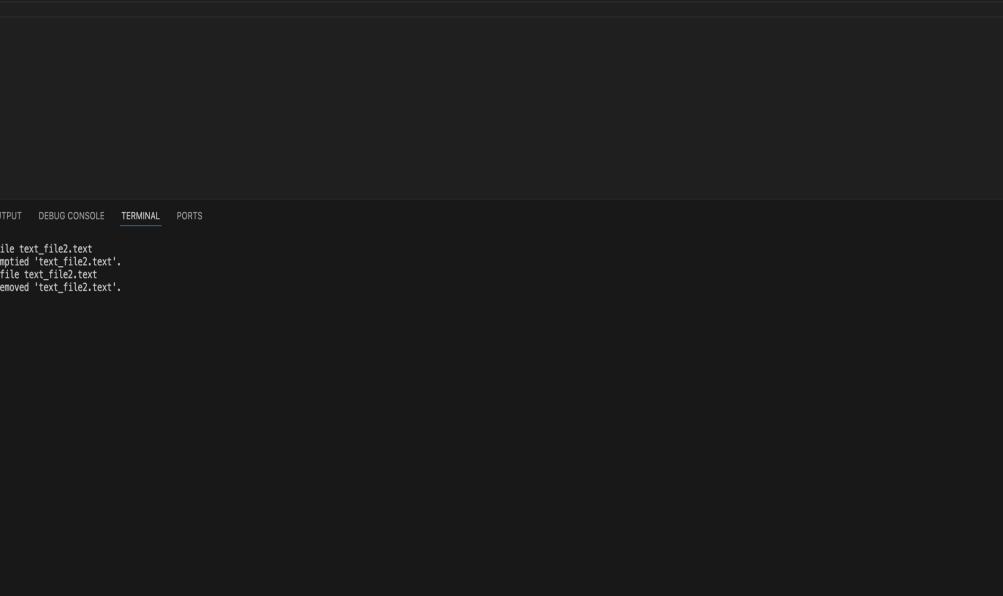
7. Repeat or Exit:

- The loop continues, prompting the user for a new command.
- If the exit command is entered, the loop terminates, and the program ends.

8. End of Program:

- The shell exits when exit command is executed completing the session.

Output

[illegible]

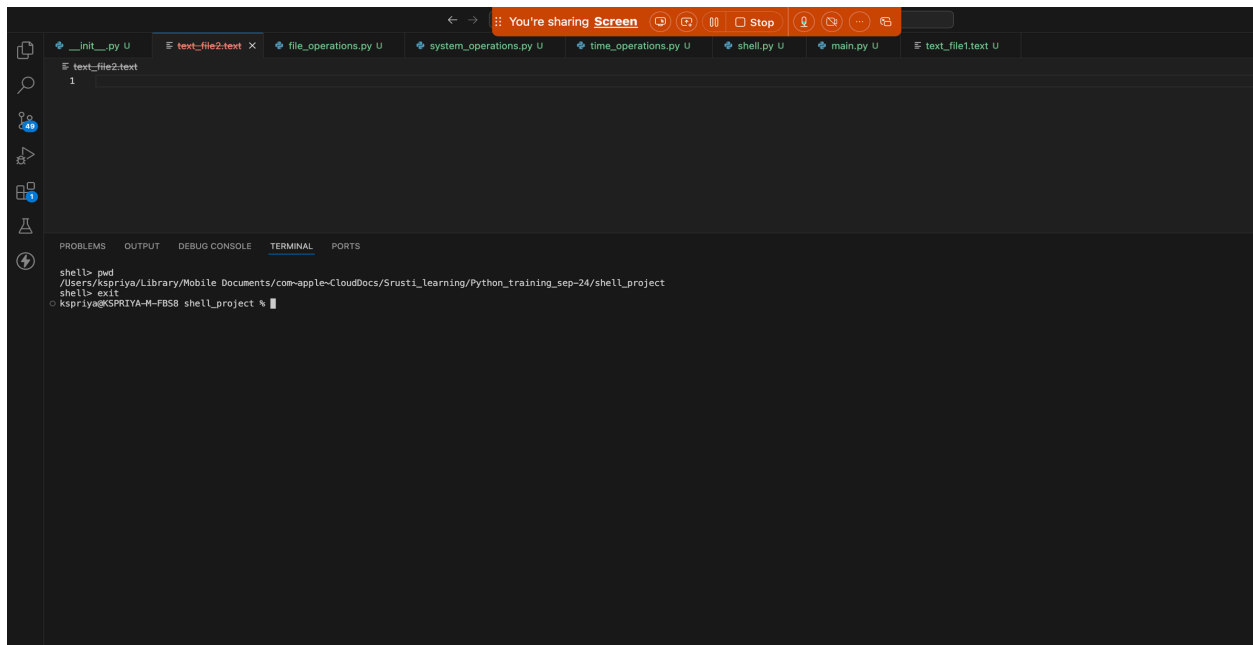
The screenshot shows a VS Code editor with a terminal window open at the bottom. The terminal displays the following commands and output:

```

shell$ empty_file text_file2.text
Successfully emptied 'text_file2.text'.
shell$ remove_file text_file2.text
Successfully removed 'text_file2.text'.
shell$ clear

```

The terminal window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (selected), and PORTS. On the right side, there is a sidebar with a search icon and a list of open files, including `__init__.py U`, `text_file2.text X`, `file_operations.py U`, `system_operations.py U`, `time_operations.py U`, `shell.py U`, `main.py U`, and `text_file1.text U`.



Conclusion

The Unix like shell python we created allows users to run common commands like listing files, checking the date, and viewing file contents .We added error handling to ensure the shell works smoothly, even if users make mistakes. Overall,it satisfies all the given commands.