BIRZEIT UNIVERSITY

Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

Second Semester 2022/2023

ENCS313- Linux Laboratory

Linux Project No.2 _PYTHON_

Student Name : Noura Manassra

Student ID : 1212359

Section : 5

Instructor : Dr. Abdel Salam Sayyad

Date : Jan. 31. 2024

1. Abstract

This Python project runs in a Linux environment, executing commands directly from Python code. Its purpose is to create a user-friendly manual for selected commands. The manual includes essential details like what each command does, its version, usage examples, related commands, and recommended ones. Users can easily navigate the manual and search for specific commands. If a command is found, the project displays relevant information, making it simple for users to understand and use various commands in the Linux environment, catering to both beginners and experienced users.

Table of Figures

Table of Contents

2. Theory

2.1 Linux Environment Background

Linux is a powerful operating system widely used for its reliability and flexibility. It's packed with various commands that users can execute to perform specific tasks. However, understanding these commands can be a bit challenging, especially for beginners. Our project aims to bridge this gap by providing a straightforward manual for selected Linux commands.

2.2 Python Background

Python is a popular programming language known for its simplicity and readability. It acts as the engine behind our project, allowing us to seamlessly interact with the Linux environment. Python's ease of use makes it an ideal choice for this project, enabling us to execute commands effortlessly and create a user-friendly interface for our manual.

2.3 Why Python?

Simplicity and Readability:

Python is easy to understand, even for those new to programming. Its clean syntax and readability make it a great tool for developing projects where simplicity is key. This simplicity is crucial for creating a user-friendly manual that anyone can navigate and comprehend.

Versatility:

Python's versatility allows us to integrate it with the Linux environment seamlessly. Whether it's executing commands or processing data, Python adapts well to various tasks, making it a reliable choice for our project.

Community Support:

Python boasts a vast and supportive community. This is advantageous for our project as it ensures a wealth of resources, libraries, and tools that enhance our ability to develop a robust and efficient command guide.

3. Procedure

First of all, I defined the important libraries to make python interact with the operating system environment, and for some other built in functions in Python to find the matching words for example or to find the matching lines which are used to find the recommendation commands.

Secondly, I defined the command_recommendation, which will be used to find the recommendation for each command.

The first class defined at this project is CommandManualGenerator. And in it, I defined the first function, which is the constructed function which takes the file name, (the file which contains the commands). And inside it, I defined some properties to find the recommendation for each command correctly. Also I defined [creat_manuals_directory] , which create a directory for manuals which will create the directory if it does not exist, and if it exists it won’t make an error so I made it in exception handling. Besides that, I created the [read_commands_fron_file]

function which will check if the file exists, and if yes will read the commands. And if not, it will exit the program. Furthermore, I used [generate_manual_for_command], this function will get the description, version, example, related commands, and the command recommendation. And it will print it to the user, the information. But for sure the description, example, related commands and the recommendation are called from another function. Moreover, I created [get_command_description] function which will find the description for each command. Also, I created [get_command_example], which will get the example for each command, I used directories here, so when it's the time to excute a command, it will run the values for this dictionary. And I created [get_related_commands], which will find the related using the shell commands. Furthermore, I used [write_manual] which will print the data as the standard output given to us in the report. Besides that I used [generate_manuals] function, here I will create a directory for the command, and make sure that all commands are read correctly from the file, so if all of then generated correctly, it will print to the user that all of them are generated correctly.

Now for the verification part I created function called [verify_manual], this function is also used at the same class with other functions mentioned above. First, I created a file path for a command manual based on the name of the command, and all of them are stored in CommandsManualds. And it checks if the manual file for the specific command exists, if it does, it proceeds to open the file and read its content into the variable xml_content, else it exits the program. I initialized a variable which will checks if the verification succeed or not, and compare the content including the description, examples, related to get the verification for the command. And for this function I used [verify_manuals], so each command will be verified alone.

Furthermore, I used the [search_command_manual] function, here were is the program in main is excuted, and it will look for the commands created using the .xml, and for the menu will be showed up here is just to show the information for each command, this will happen only if the the command substring exists between the xml files. Also I used the function [display_part], at this part it reads the xml file, and searches for the specified command, and it displays the content between the <start_tag> and the <end_tag> if the file exists, otherwise it prints the error message. Also I used [get_recommendations], as I mentioned before I created a lists for the recommendations part, all of them has this style commandName_recommendation, so here it will call the list depending on it's name. And now for the last function at this class which is [recommend_commands], which will get the last search for the user, and will call the get_recommendations function to get the recommended function for the last search by the user, and will print it if it exists. Otherwise it will print that the recommendation not found.

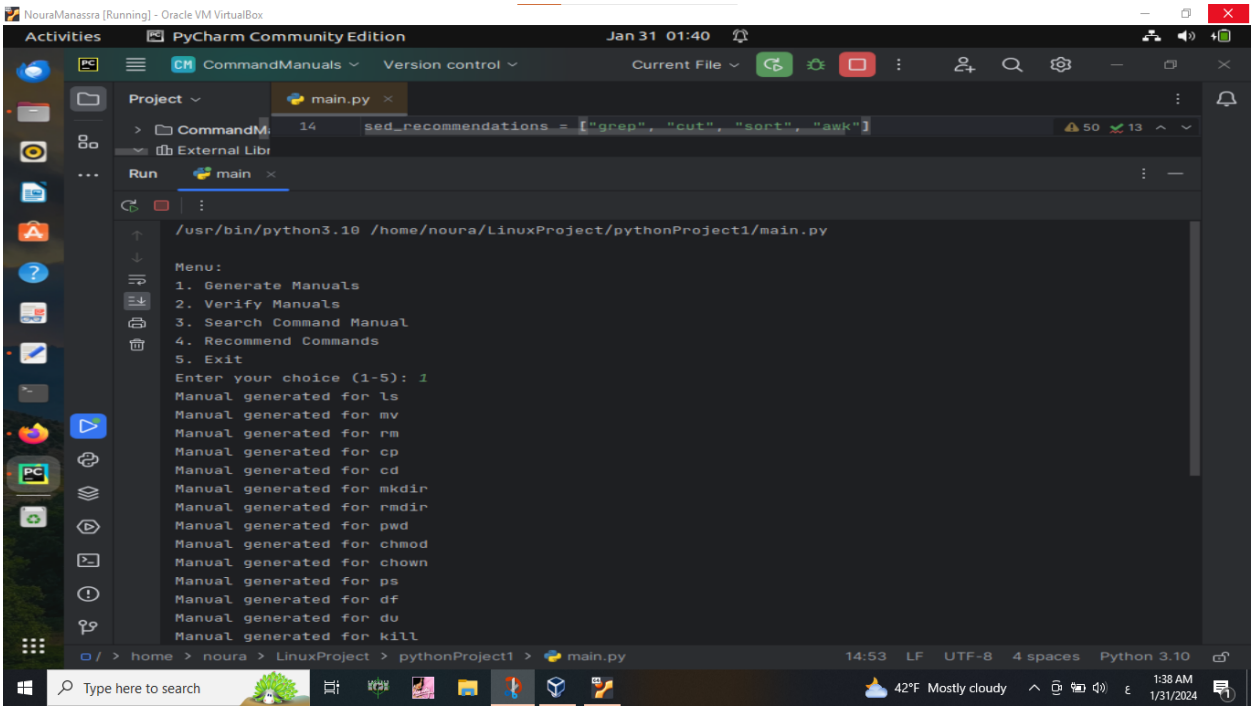
So Overall, the previous class finds each item at the manual for each commend. And here is the second class, which is [commandManual] , this class will bring the description, version, example, excute_example and the related commands for each command. So later an objects from this class will be used in the XMLSerializer.

The last class used is the [XMLSerializer], here it uses the [commandManual] object and convert it into XML strings, and note here that I used the tree technique, so each node will present the command name, and all it's children will act as its attributes including the description, example, version and related commands.

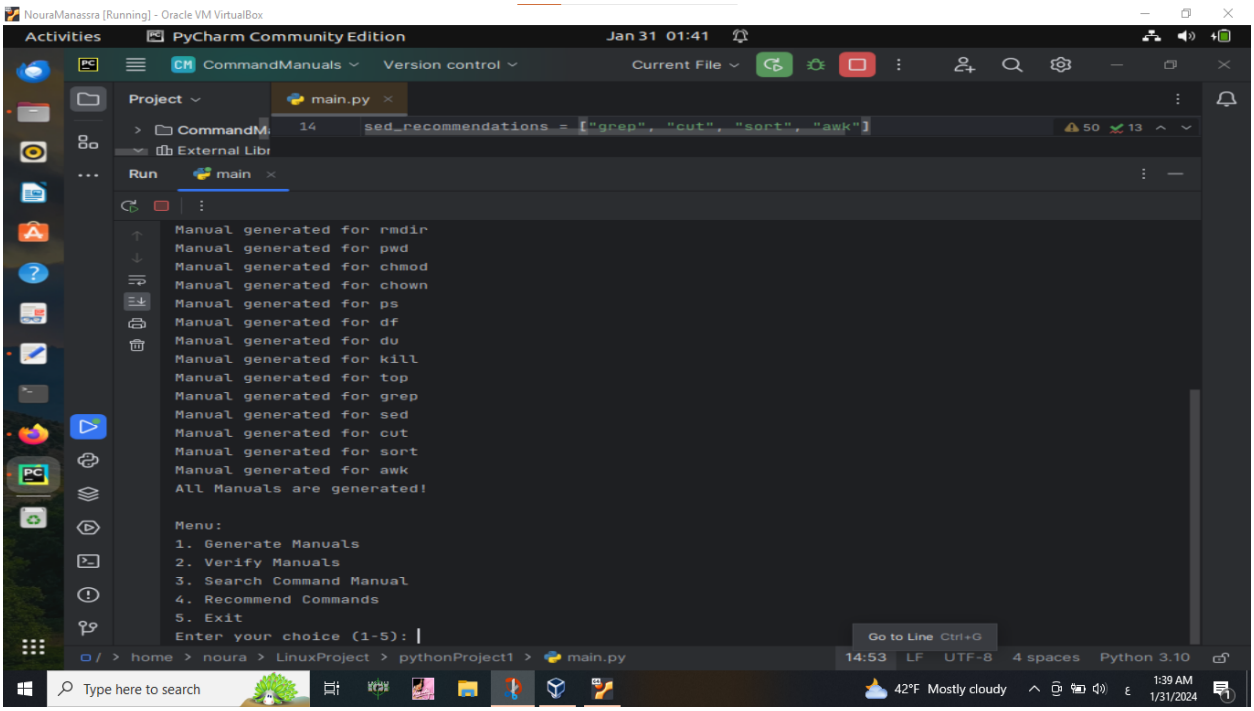
The last thing, which will excute the program is the main function, and the script here creates an instance of [commandManualGenerator], and here exactly were the project works. Furthermore, I put the main list which contains the options for the user to choose what he wants.

4. Results

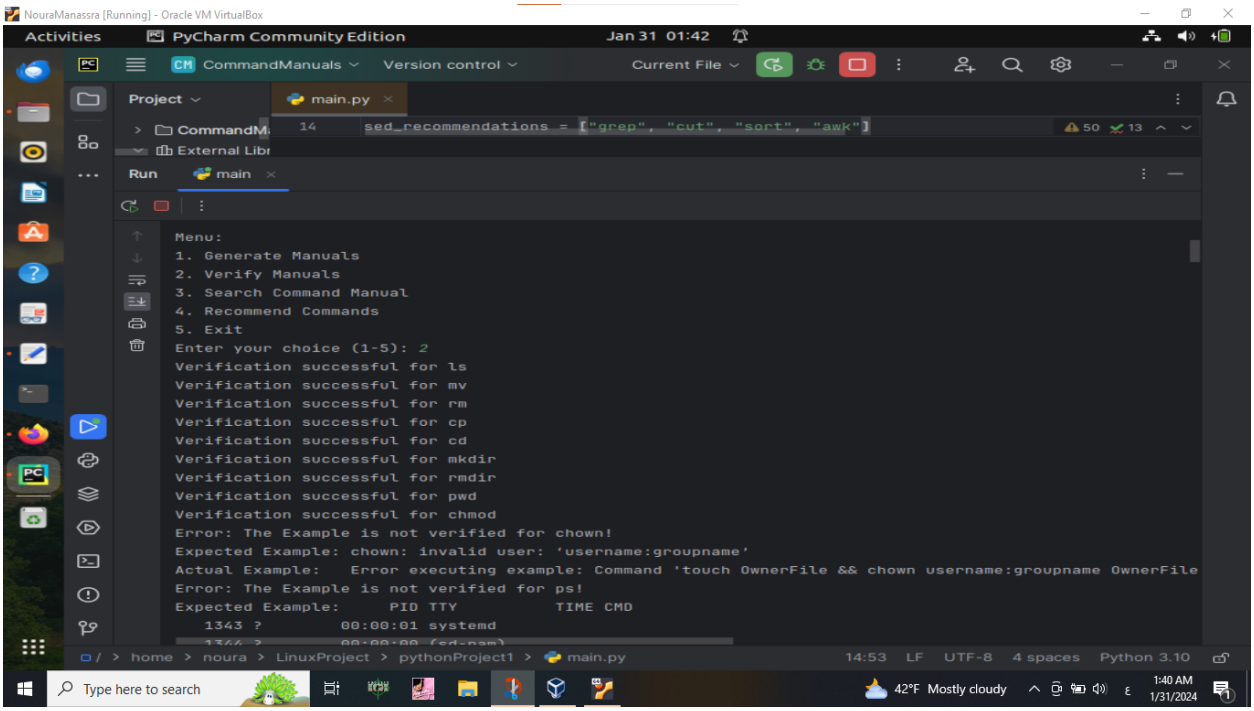
Here are the results of execution the python code.



Fig[1] : Generating Manual

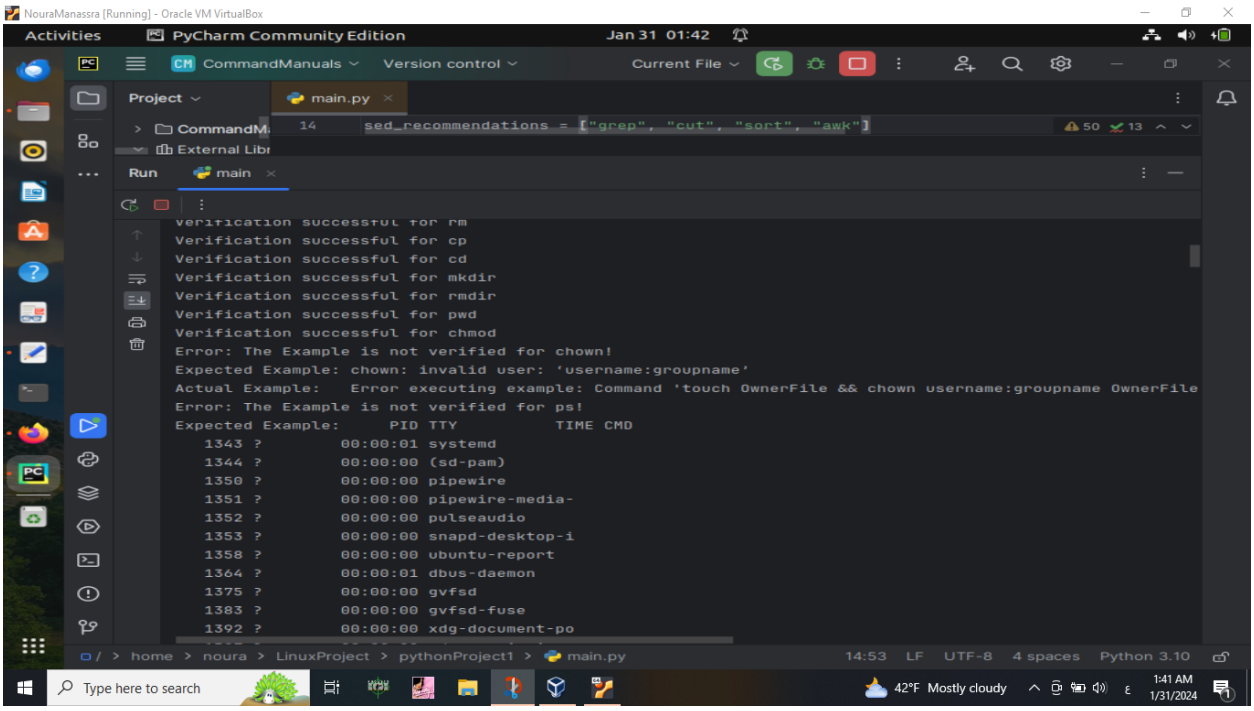


Fig[2] : DONE Generating Manual

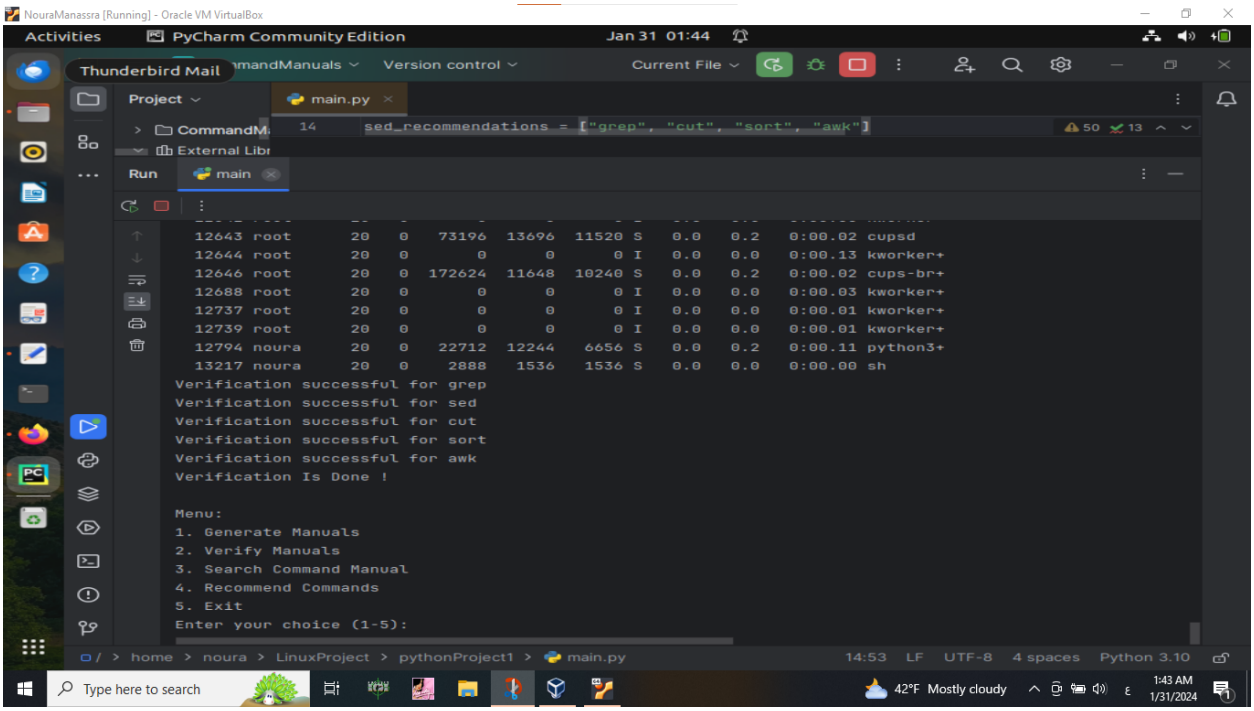


Fig[3] : Verfication

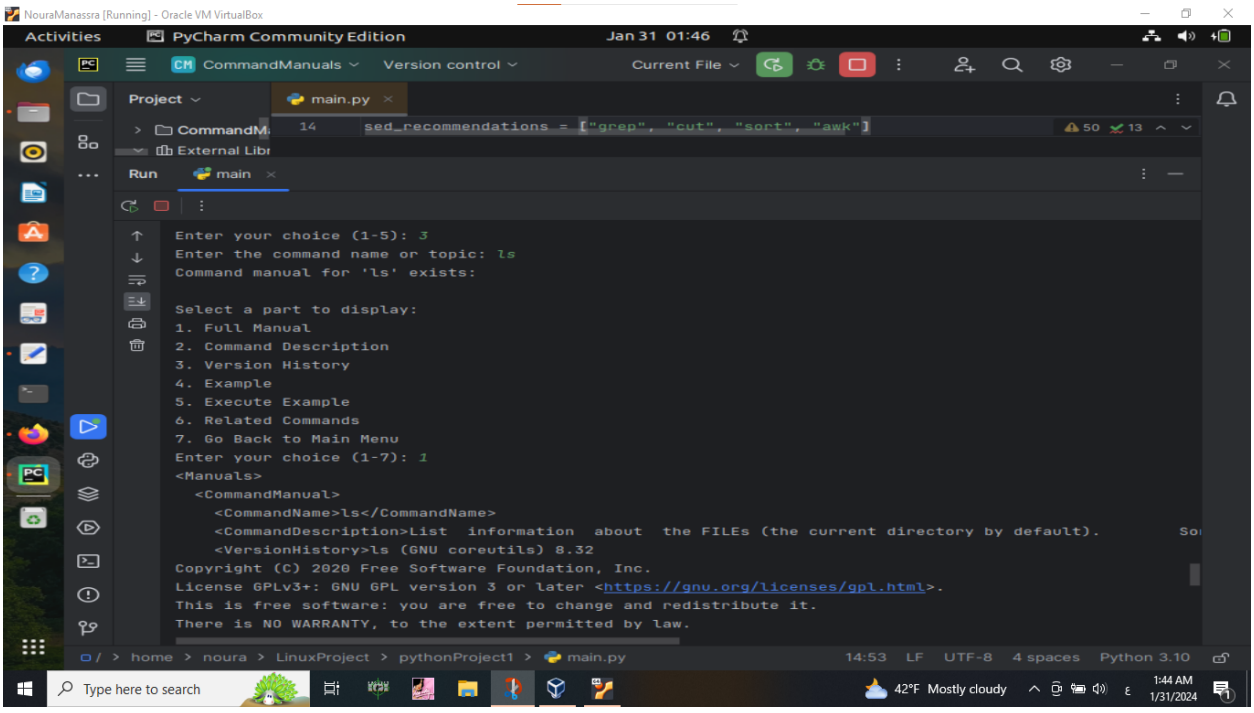
Note for the unverified commands, it will print the expected and the real results.



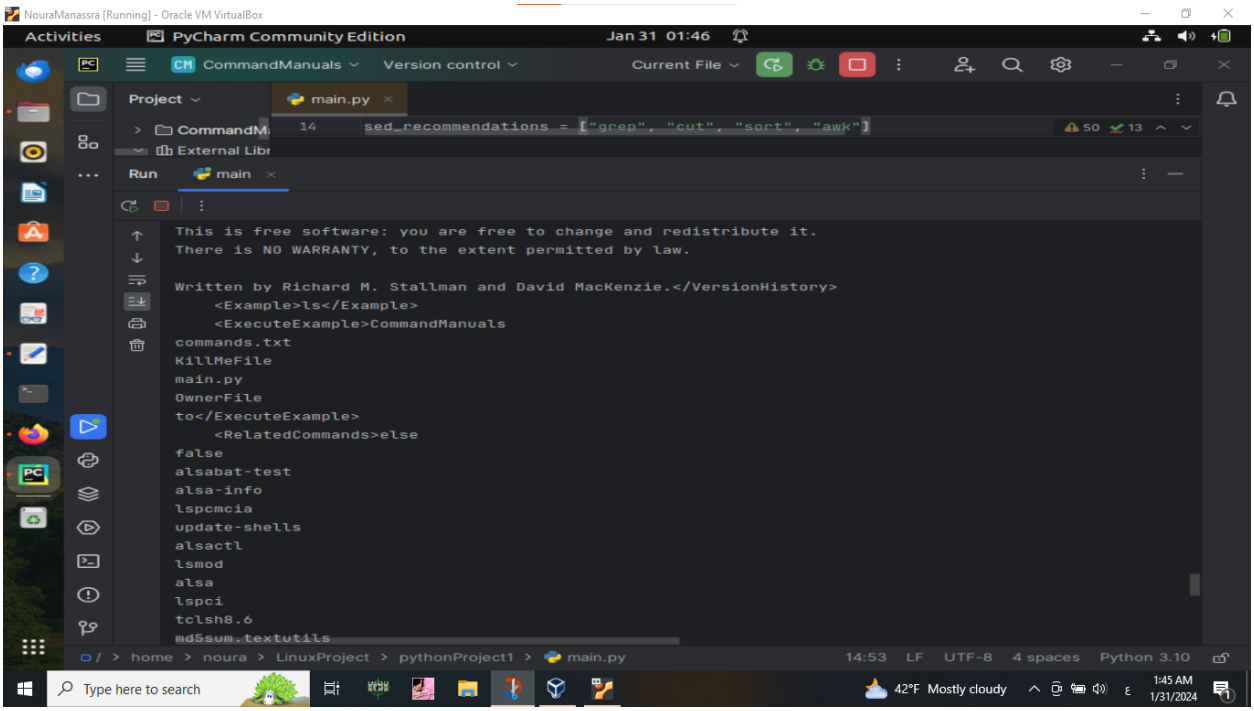
Fig[4] : Unsuccessful verification example



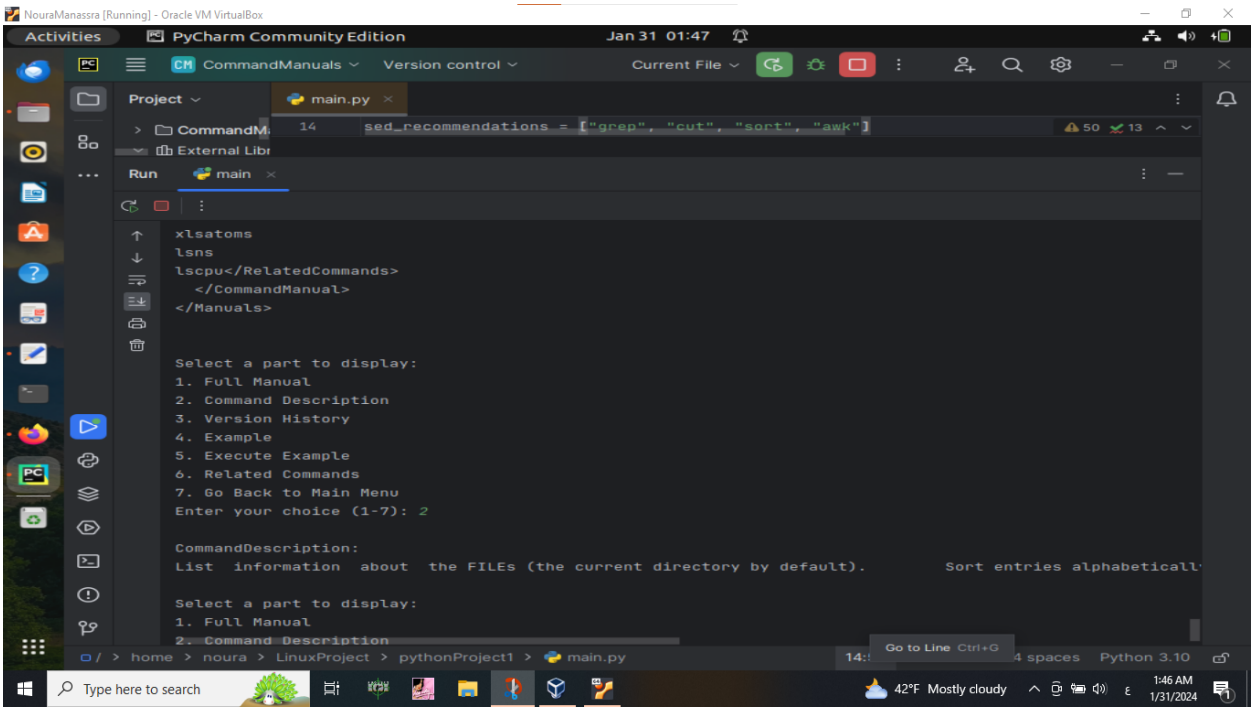
Fig[5] : Done vefication



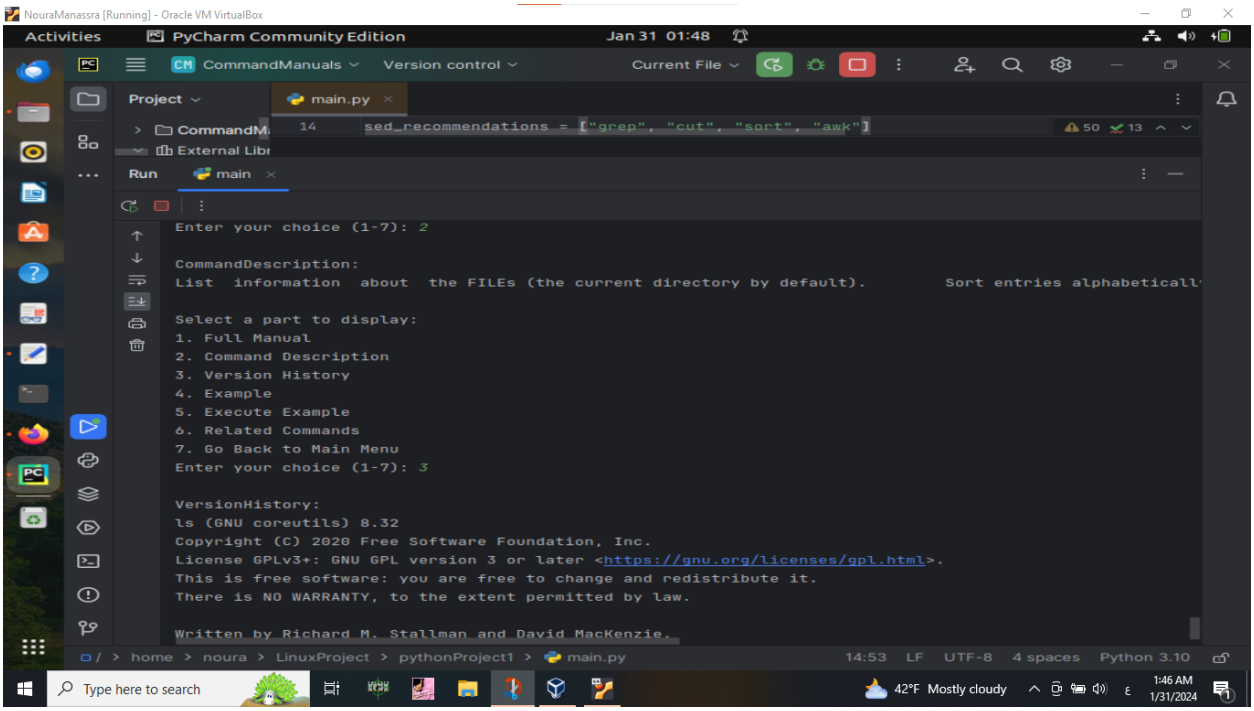
Fig[6] : Search for a command



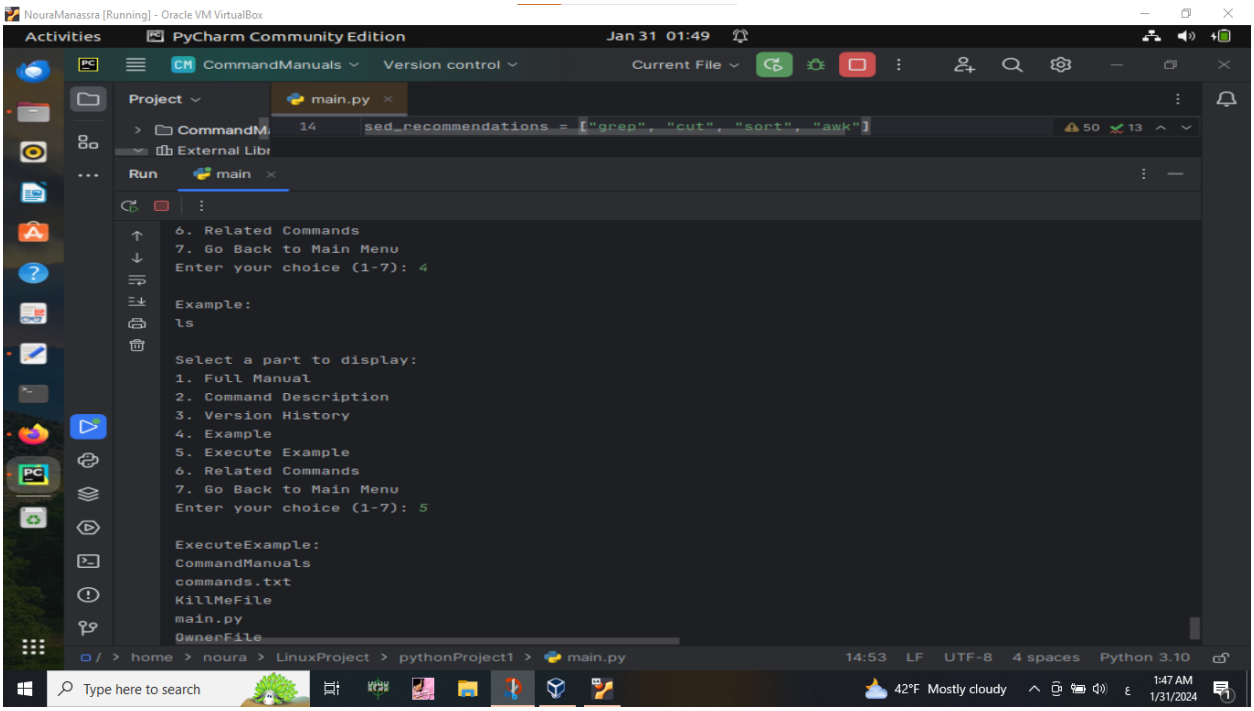
Fig[7] : Full manual for a command



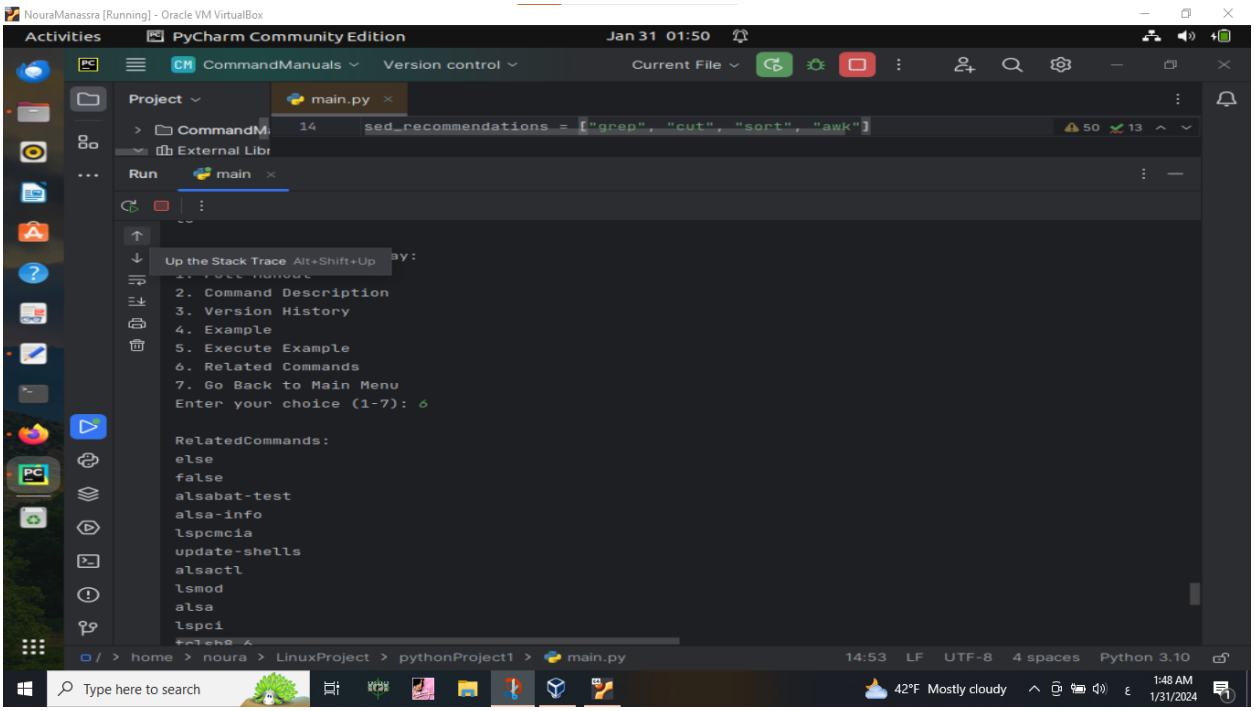
Fig[8] : description for a command



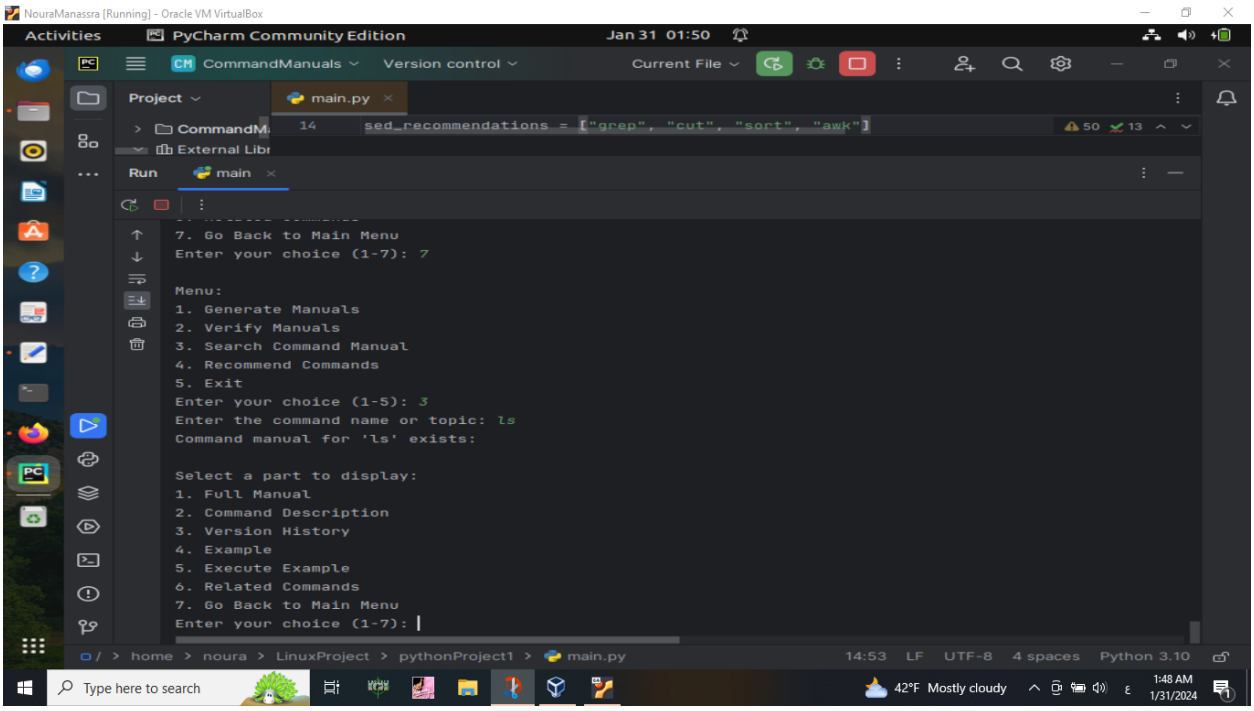
Fig[8] : version



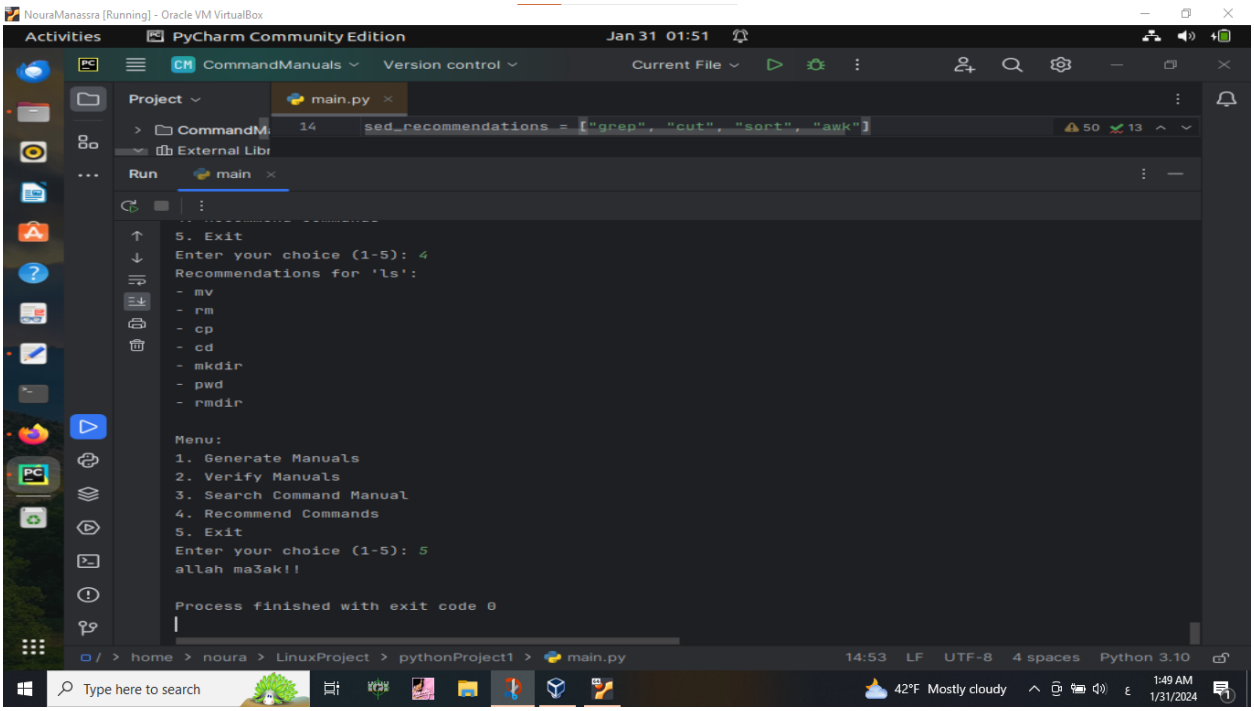
Fig[9] : example and it’s execution for a command



Fig[10] related commands for a command



Fig[11] : search for a command



Fig[12] : Recommended commands for a command and exiting the program