Bir Zeit University Linux Lab ENCS313 Python Project

Prepared by:

Name: Abdelrhman Abed

ID:119393

Instructor:

DR. Aziz Qaroush

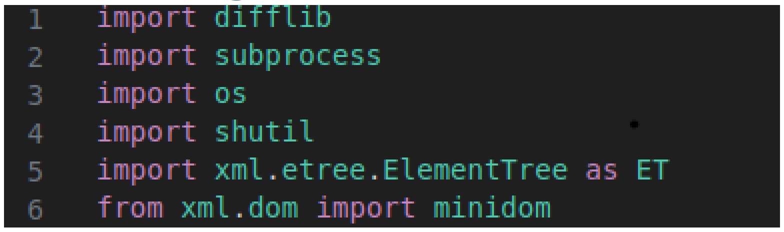
TA:

Eng. Ahed Mafarjeh

Section:4

DATE:29/1/2024

Describing The Code:



```python
import difflib
import subprocess
import os
import shutil
import xml.etree.ElementTree as ET
from xml.dom import minidom
```

The first five lines of the code is used to import functional library by command :

-difflib: Gives functions to compare sequences, including unified diffs, HTML, and context.

-subprocess: This function is used to launch and return codes for new processes and establish connections to the errors, input, and output pipes.

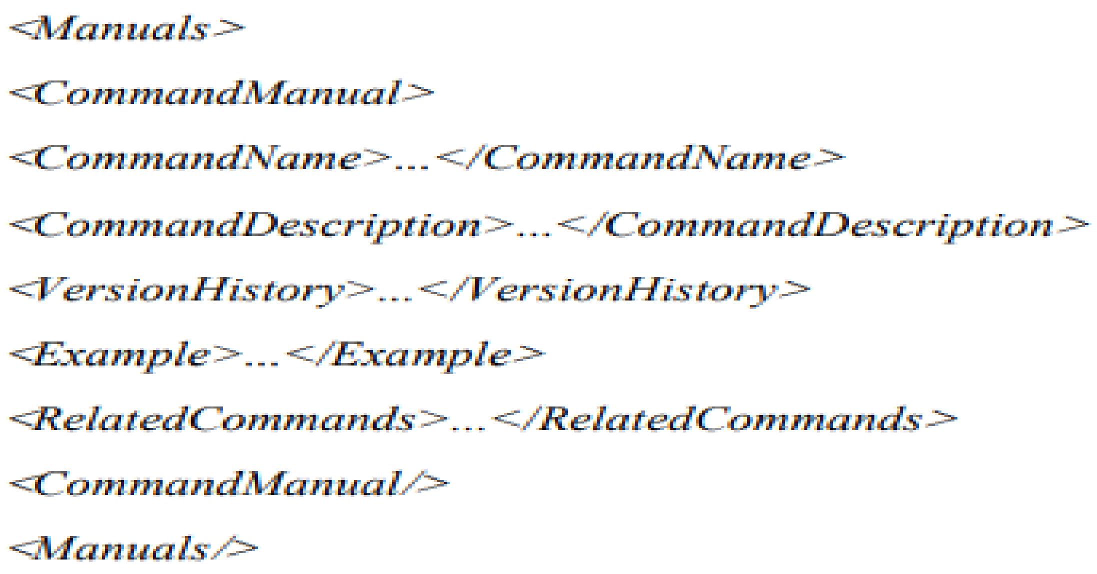
-os: gives a method of utilizing features that are specific to an operating system, such as reading from or writing to a file system.

-Shutil: gives advanced file operations like erasure and duplication.

-xml.etree.ElementTree as ET:A direct and fast library for processing and generating XML data is called xml.etree.ElementTree .

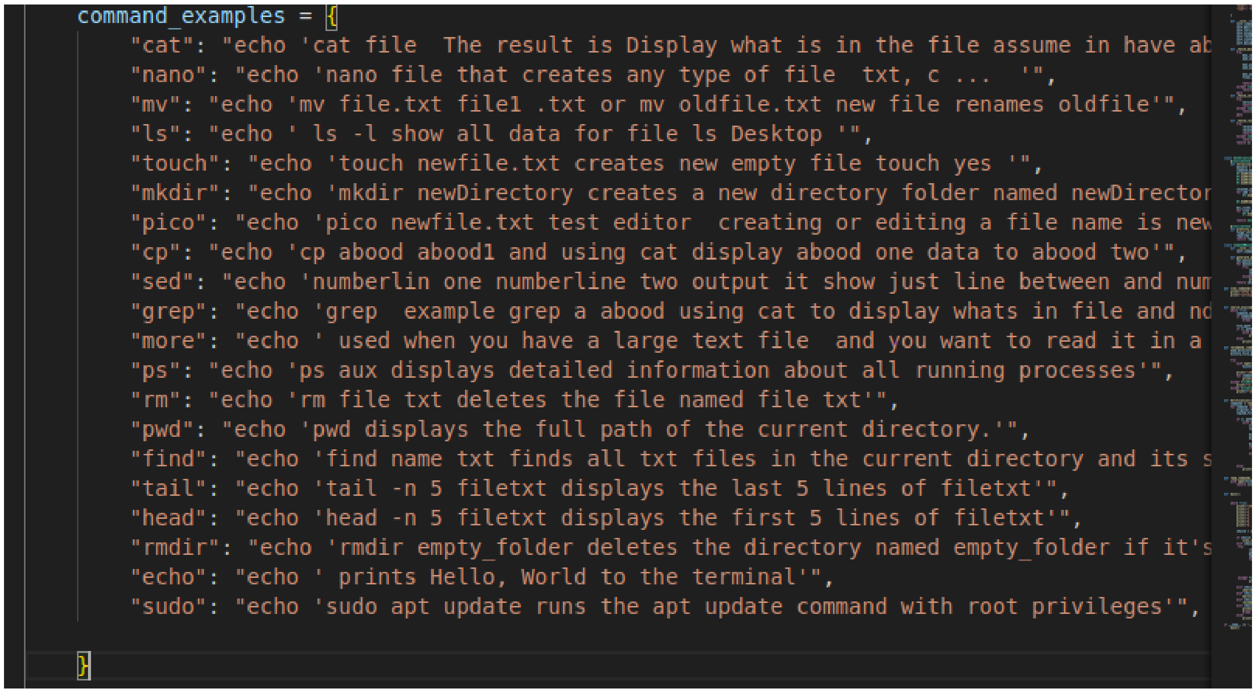
-xml.dom.minidom: gives resources for XML parsing via a DOM API.

I using this to make shown below:

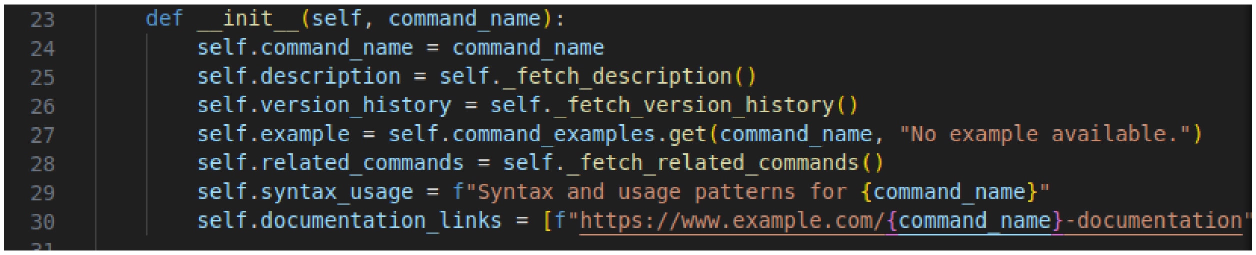


```
<Manuals>
<CommandManual>
<CommandName>...</CommandName>
<CommandDescription>...</CommandDescription>
<VersionHistory>...</VersionHistory>
<Example>...</Example>
<RelatedCommands>...</RelatedCommands>
<CommandManual/>
<Manuals/>
```

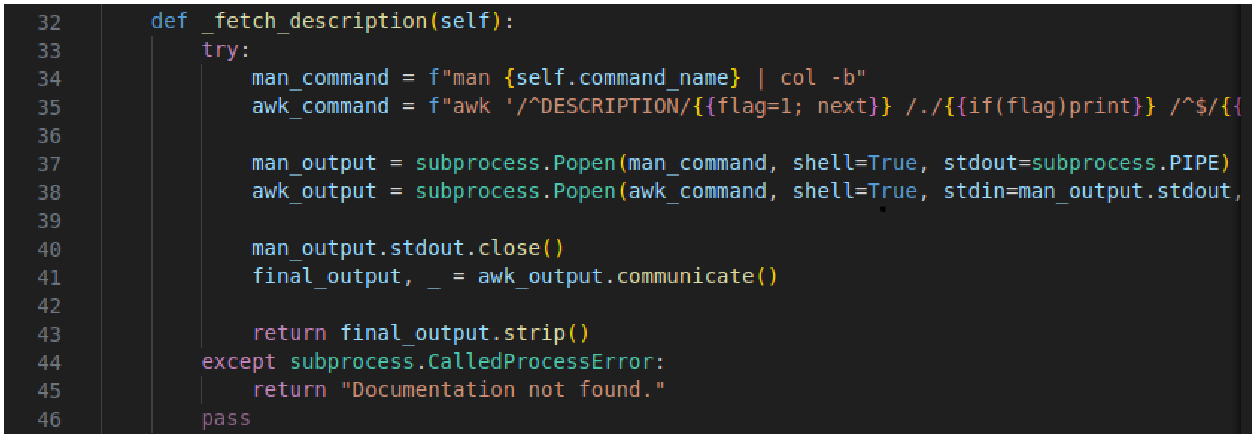
Class Command Manual:



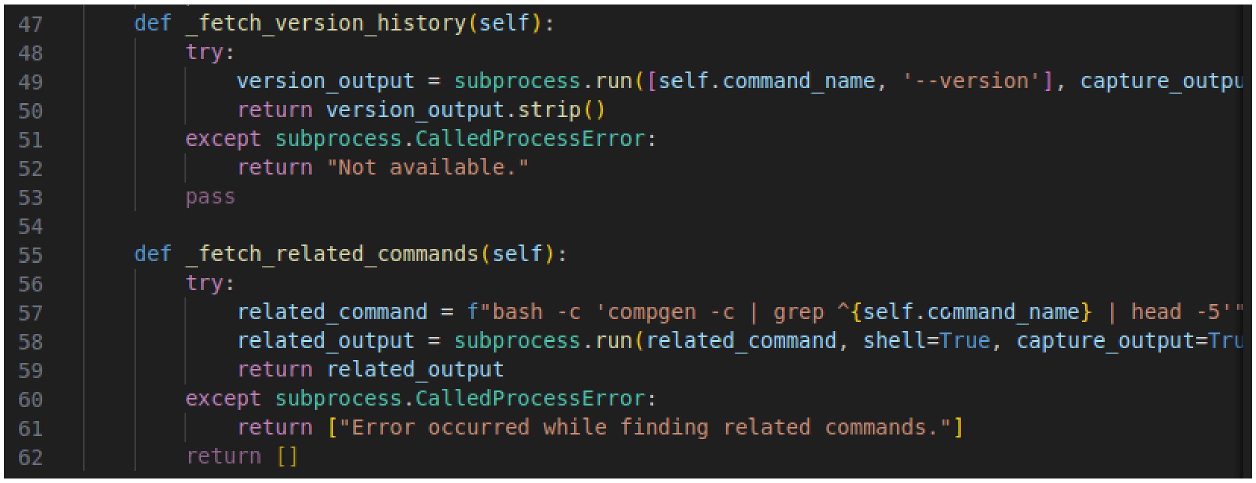
```
command_examples = {
    "cat": "echo 'cat file  The result is Display what is in the file assume in have ab
    "nano": "echo 'nano file that creates any type of file  txt, c ...  '",
    "mv": "echo 'mv file.txt file1 .txt or mv oldfile.txt new file renames oldfile'",
    "ls": "echo ' ls -l show all data for file ls Desktop '",
    "touch": "echo 'touch newfile.txt creates new empty file touch yes '",
    "mkdir": "echo 'mkdir newDirectory creates a new directory folder named newDirector
    "pico": "echo 'pico newfile.txt test editor  creating or editing a file name is new
    "cp": "echo 'cp abood abood1 and using cat display abood one data to abood two'",
    "sed": "echo 'numberlin one numberline two output it show just line between and num
    "grep": "echo 'grep  example grep a abood using cat to display whats in file and no
    "more": "echo ' used when you have a large text file  and you want to read it in a
    "ps": "echo 'ps aux displays detailed information about all running processes'",
    "rm": "echo 'rm file txt deletes the file named file txt'",
    "pwd": "echo 'pwd displays the full path of the current directory.'",
    "find": "echo 'find name txt finds all txt files in the current directory and its s
    "tail": "echo 'tail -n 5 filetxt displays the last 5 lines of filetxt'",
    "head": "echo 'head -n 5 filetxt displays the first 5 lines of filetxt'",
    "rmdir": "echo 'rmdir empty_folder deletes the directory named empty_folder if it's
    "echo": "echo ' prints Hello, World to the terminal'",
    "sudo": "echo 'sudo apt update runs the apt update command with root privileges'",

}
```



```
    def __init__(self, command_name):
        self.command_name = command_name
        self.description = self._fetch_description()
        self.version_history = self._fetch_version_history()
        self.example = self.command_examples.get(command_name, "No example available.")
        self.related_commands = self._fetch_related_commands()
        self.syntax_usage = f"Syntax and usage patterns for {command_name}"
        self.documentation_links = [f"https://www.example.com/{command_name}-documentation"
```



```
    def _fetch_description(self):
        try:
            man_command = f"man {self.command_name} | col -b"
            awk_command = f"awk '/^DESCRIPTION/{{flag=1; next}} /./{{if(flag)print}} /^$/{{

            man_output = subprocess.Popen(man_command, shell=True, stdout=subprocess.PIPE)
            awk_output = subprocess.Popen(awk_command, shell=True, stdin=man_output.stdout,

            man_output.stdout.close()
            final_output, _ = awk_output.communicate()

            return final_output.strip()
        except subprocess.CalledProcessError:
            return "Documentation not found."
        pass
```



```
    def _fetch_version_history(self):
        try:
            version_output = subprocess.run([self.command_name, '--version'], capture_outpu
            return version_output.strip()
        except subprocess.CalledProcessError:
            return "Not available."
        pass

    def _fetch_related_commands(self):
        try:
            related_command = f"bash -c 'compgen -c | grep ^{self.command_name} | head -5'"
            related_output = subprocess.run(related_command, shell=True, capture_output=Tru
            return related_output
        except subprocess.CalledProcessError:
            return ["Error occurred while finding related commands."]
        return []
```

* definition of a class name (CommandManual)

* Class Variable: command_example ---> using “command” In order to specify a file, we want to enter the print statement We entered the sentences manually

-Constructor Method:

*def __init__ : it is called when a new instance of commandManual is created , it takes self and command namd

-Instance Varibles:

*slef.command_name : stores name of command

*slef.descrption: method to get the command’s description

*slef.version_history: calls _fetch_version_history

* self.example:get command_name “no example available retrieves an example for the command is not found

* self.related_commands: method to get related commands and stores them.

* self.syntax_usage: Sets a string describing the syntax and usage pattern for the command

* self.documentation_links: Creates a list containing a URL to the command's documentation

---

def _fetch_description(self): used to fetch the description of the command. It uses subprocesses to execute shell commands and capture the output using man to open the data of command and using sed to make cut of description from the data and ended after 2 line empty

man_output = subprocess.Popen(man_command, shell=True, stdout=subprocess.PIPE)

awk_output = subprocess.Popen(awk_command, shell=True, stdin=man_output.stdout, stdout=subprocess.PIPE, text=True)

using subprocess.pop to execute man_command . it runs in the shell true and the standard output is given

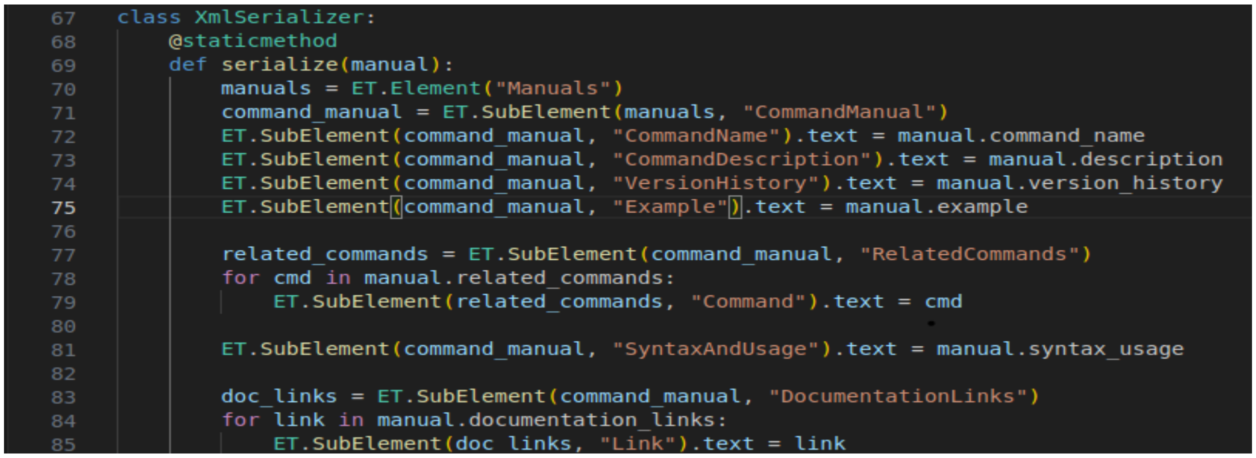
man_output.stdout.close() → This line closes the standard output stream

final_output, _ = awk_output.communicate() → stores the extracted DESCRIPTION section from the manual page   _ to ignore error ouput

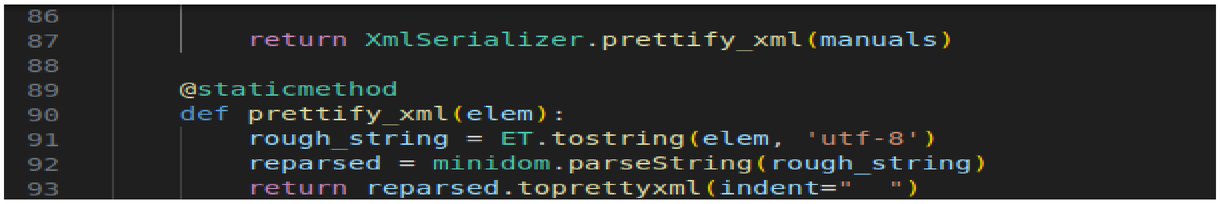
* def _fetch_version_history(self): fetching the version history of the command. It attempts to run the command with the --version flag and captures the output

* def _fetch_related_commands(self): This method fetches related commands. It uses a shell command to get a list of commands starting with the same name and returns the first five

Class XmlSerializer:



```python
class XmlSerializer:
    @staticmethod
    def serialize(manual):
        manuals = ET.Element("Manuals")
        command_manual = ET.SubElement(manuals, "CommandManual")
        ET.SubElement(command_manual, "CommandName").text = manual.command_name
        ET.SubElement(command_manual, "CommandDescription").text = manual.description
        ET.SubElement(command_manual, "VersionHistory").text = manual.version_history
        ET.SubElement(command_manual, "Example").text = manual.example

        related_commands = ET.SubElement(command_manual, "RelatedCommands")
        for cmd in manual.related_commands:
            ET.SubElement(related_commands, "Command").text = cmd

        ET.SubElement(command_manual, "SyntaxAndUsage").text = manual.syntax_usage

        doc_links = ET.SubElement(command_manual, "DocumentationLinks")
        for link in manual.documentation_links:
            ET.SubElement(doc_links, "Link").text = link
```



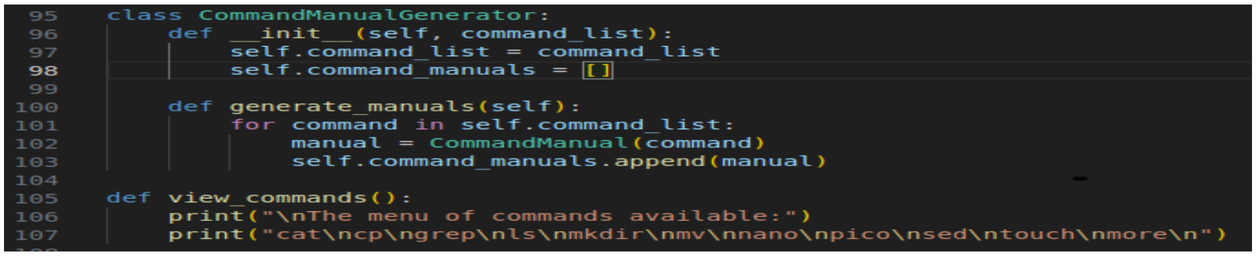
```python

        return XmlSerializer.prettify_xml(manuals)

    @staticmethod
    def prettify_xml(elem):
        rough_string = ET.tostring(elem, 'utf-8')
        reparsed = minidom.parseString(rough_string)
        return reparsed.toprettyxml(indent="  ")
```

*This class is used to serialize an object of some manual data structure into an XML format and then prettify the resulting XML string

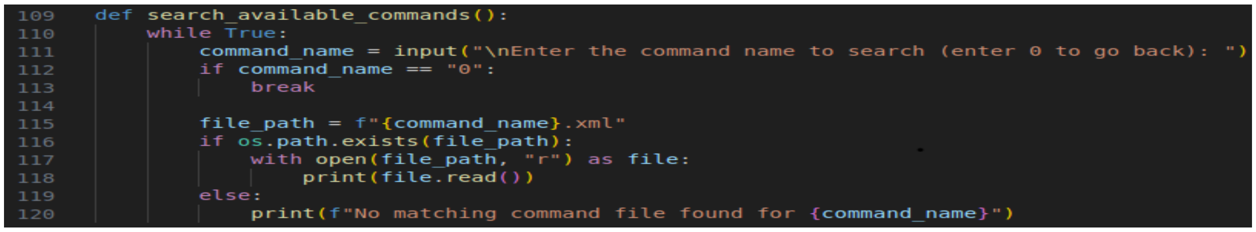
*def serialize(manual) : It takes an argument manual and It creates an XML structure using the xml.etree.ElementTree library, The data from the manual object is populated into these XML elements , it calls the prettify_xml method to convert the XML into a nicely formatted string and returns that string

* def prettify_xml: It takes an XML element elem as an argument , It converts the given XML element into a UTF-8 encoded string using ET.tostring , It then parses the string and reformats it into a pretty-printed XML format using minidom.parseString with an indentation of two spaces.

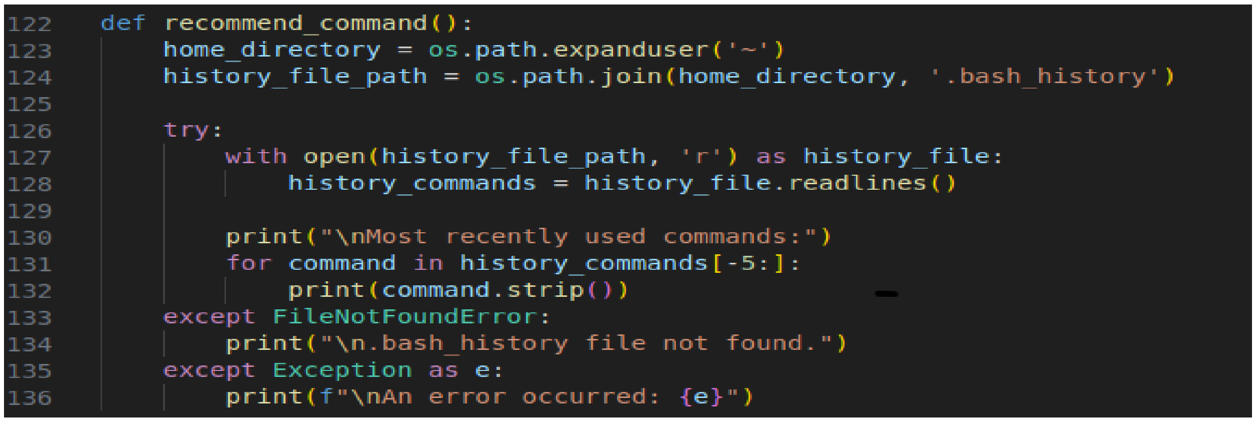
Class CommandManualGenerator:



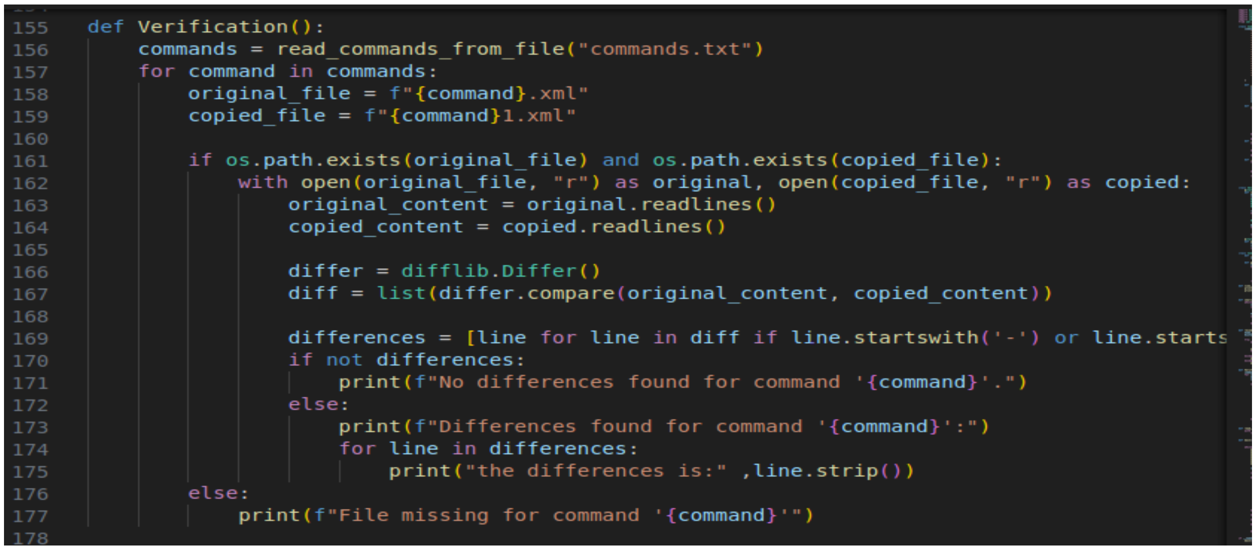
```python
class CommandManualGenerator:
    def __init__(self, command_list):
        self.command_list = command_list
        self.command_manuals = []

    def generate_manuals(self):
        for command in self.command_list:
            manual = CommandManual(command)
            self.command_manuals.append(manual)

def view_commands():
    print("\nThe menu of commands available:")
    print("cat\ncp\ngrep\nls\nmkdir\nmv\nnano\npico\nsed\ntouch\nmore\n")
```



```python
def search_available_commands():
    while True:
        command_name = input("\nEnter the command name to search (enter 0 to go back): ")
        if command_name == "0":
            break

        file_path = f"{command_name}.xml"
        if os.path.exists(file_path):
            with open(file_path, "r") as file:
                print(file.read())
        else:
            print(f"No matching command file found for {command_name}")
```



```python
def recommend_command():
    home_directory = os.path.expanduser('~')
    history_file_path = os.path.join(home_directory, '.bash_history')

    try:
        with open(history_file_path, 'r') as history_file:
            history_commands = history_file.readlines()

        print("\nMost recently used commands:")
        for command in history_commands[-5:]:
            print(command.strip())
    except FileNotFoundError:
        print("\n.bash_history file not found.")
    except Exception as e:
        print(f"\nAn error occurred: {e}")
```



```python
def Verification():
    commands = read_commands_from_file("commands.txt")
    for command in commands:
        original_file = f"{command}.xml"
        copied_file = f"{command}1.xml"

        if os.path.exists(original_file) and os.path.exists(copied_file):
            with open(original_file, "r") as original, open(copied_file, "r") as copied:
                original_content = original.readlines()
                copied_content = copied.readlines()

                differ = difflib.Differ()
                diff = list(differ.compare(original_content, copied_content))

                differences = [line for line in diff if line.startswith('-') or line.starts
                if not differences:
                    print(f"No differences found for command '{command}'.")
                else:
                    print(f"Differences found for command '{command}':")
                    for line in differences:
                        print("the differences is:" ,line.strip())
        else:
            print(f"File missing for command '{command}'")
```

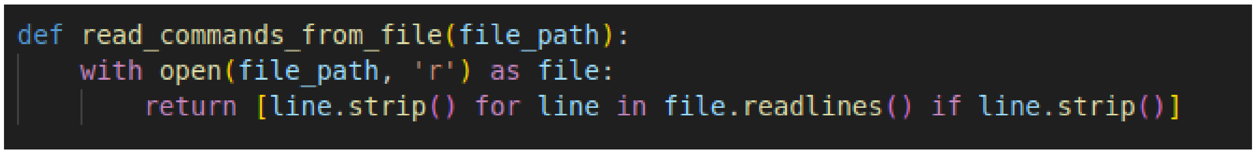
*is responsible for generating command manuals. It takes a list of commands as input and creates command manuals for each command in the list and Using an iterative process, this technique builds a CommandManual object for every command in the list and adds it to the command_manuals list.

* view_commands function: This function displays a menu of available commands by printing a list of command names

* search_available_commands function: This function allows the user to search for command documentation by entering a command name. It checks if an XML file exists for the entered command and displays its content if found

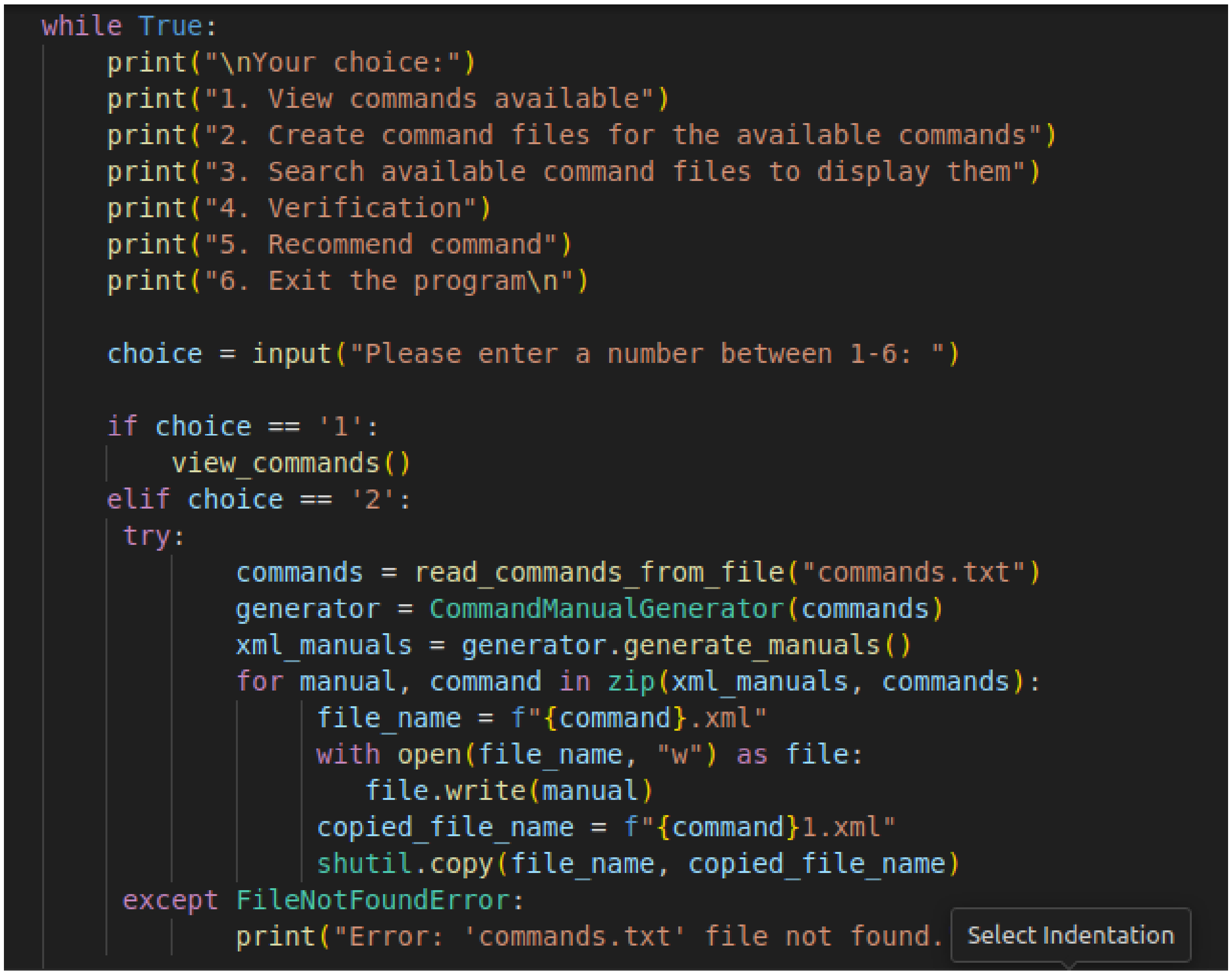
* recommend_command function: This function retrieves and displays the most recently used commands from the user's .bash_history file.

* Verification function: This function verifies the accuracy of copied command documentation by comparing the content of the original and copied XML files for each command. It uses the difflib library to highlight differences



```python
def read_commands_from_file(file_path):
    with open(file_path, 'r') as file:
        return [line.strip() for line in file.readlines() if line.strip()]
```

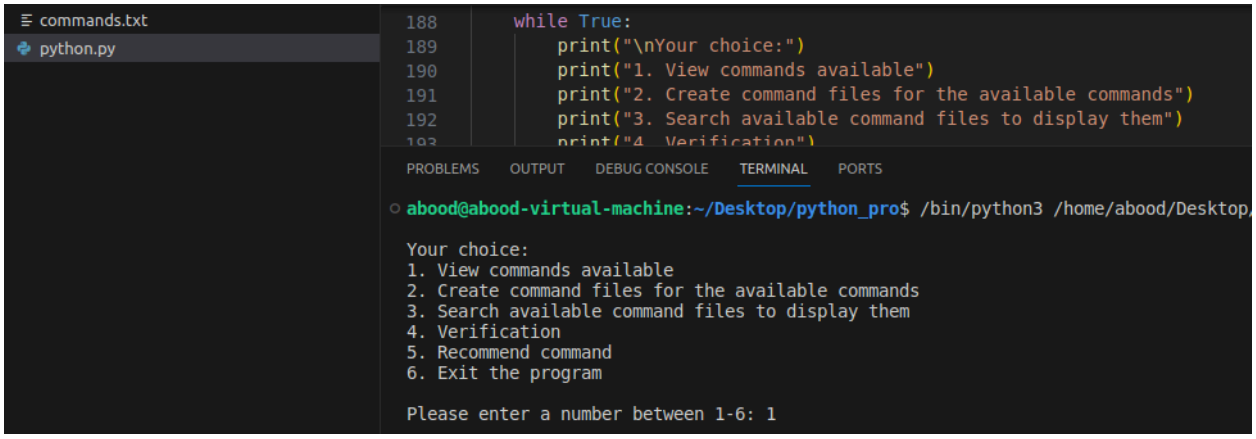
*this for reading form file I put the name of file commands.txt



```python
while True:
    print("\nYour choice:")
    print("1. View commands available")
    print("2. Create command files for the available commands")
    print("3. Search available command files to display them")
    print("4. Verification")
    print("5. Recommend command")
    print("6. Exit the program\n")

    choice = input("Please enter a number between 1-6: ")

    if choice == '1':
        view_commands()
    elif choice == '2':
     try:
            commands = read_commands_from_file("commands.txt")
            generator = CommandManualGenerator(commands)
            xml_manuals = generator.generate_manuals()
            for manual, command in zip(xml_manuals, commands):
                file_name = f"{command}.xml"
                with open(file_name, "w") as file:
                   file.write(manual)
                copied_file_name = f"{command}1.xml"
                shutil.copy(file_name, copied_file_name)
     except FileNotFoundError:
            print("Error: 'commands.txt' file not found.
```

*Main menu form choice and in case 2 creation form file sure the commands takes form file and name the file commands.txt I put in the text file 20 commands

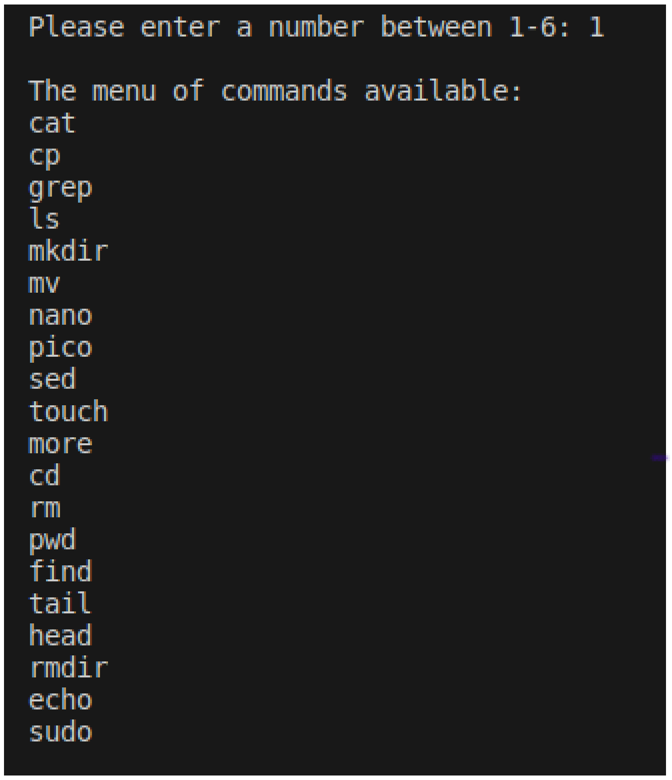
The result when run the code:



```
    while True:
        print("\nYour choice:")
        print("1. View commands available")
        print("2. Create command files for the available commands")
        print("3. Search available command files to display them")
        print("4. Verification")
```

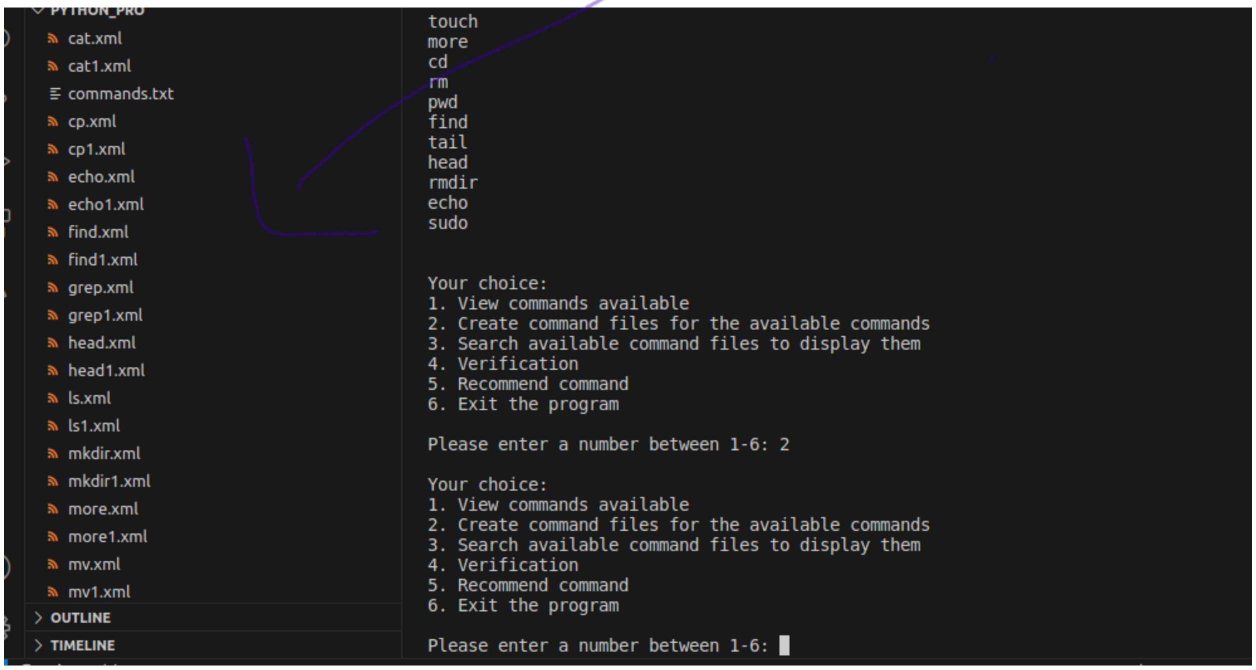
```
abood@abood-virtual-machine:~/Desktop/python_pro$ /bin/python3 /home/abood/Desktop/

Your choice:
1. View commands available
2. Create command files for the available commands
3. Search available command files to display them
4. Verification
5. Recommend command
6. Exit the program

Please enter a number between 1-6: 1
```

*When run the code it show the menu for choice



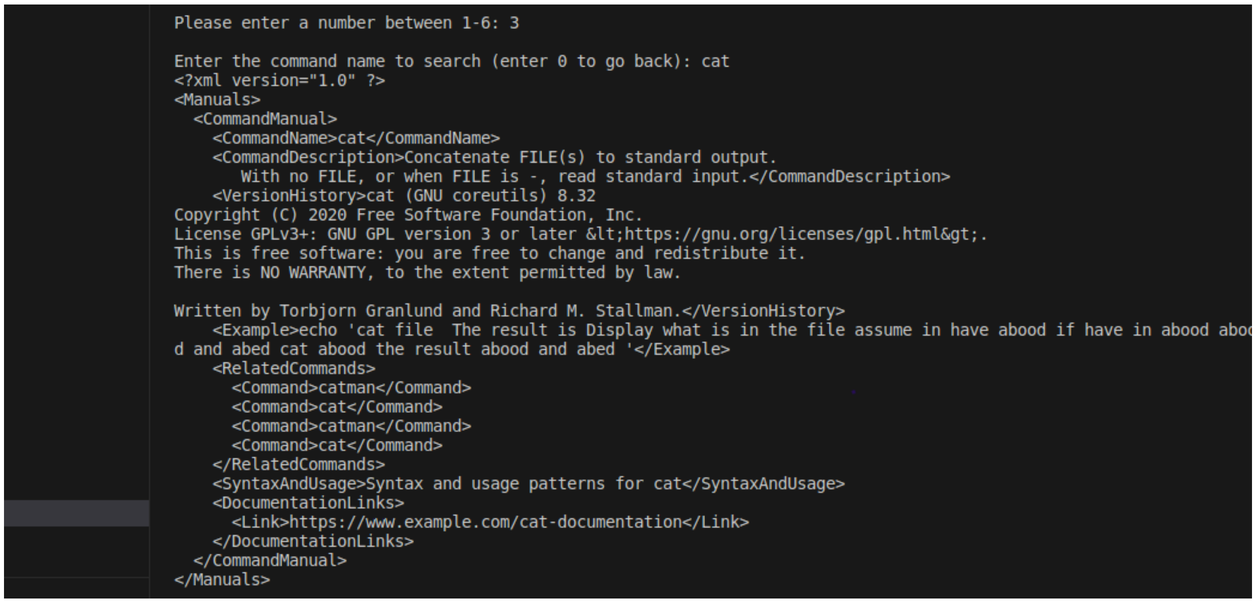
```
Please enter a number between 1-6: 1

The menu of commands available:
cat
cp
grep
ls
mkdir
mv
nano
pico
sed
touch
more
cd
rm
pwd
find
tail
head
rmdir
echo
sudo
```

*when enter the number 1 it show the commands available



```
touch
more
cd
rm
pwd
find
tail
head
rmdir
echo
sudo


Your choice:
1. View commands available
2. Create command files for the available commands
3. Search available command files to display them
4. Verification
5. Recommend command
6. Exit the program

Please enter a number between 1-6: 2

Your choice:
1. View commands available
2. Create command files for the available commands
3. Search available command files to display them
4. Verification
5. Recommend command
6. Exit the program

Please enter a number between 1-6: 
```

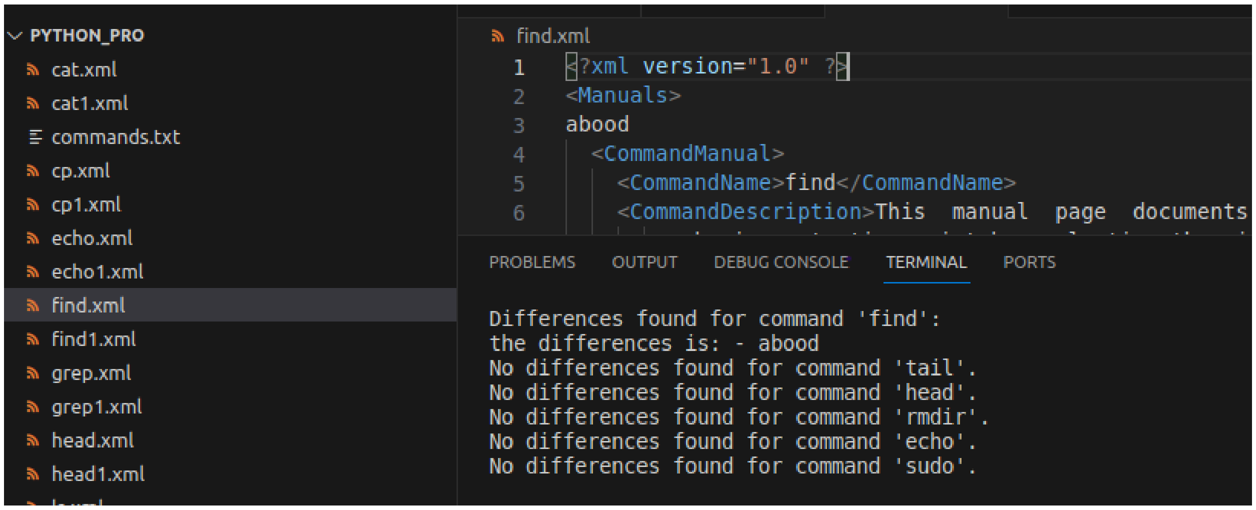
*when enter number 2 it is creation the file.xml have data for commands when creation the file it is make copy form file and insert in onther file to make verification such as cat.xml and cat1.xml



```
Please enter a number between 1-6: 3

Enter the command name to search (enter 0 to go back): cat
<?xml version="1.0" ?>
<Manuals>
  <CommandManual>
    <CommandName>cat</CommandName>
    <CommandDescription>Concatenate FILE(s) to standard output.
       With no FILE, or when FILE is -, read standard input.</CommandDescription>
    <VersionHistory>cat (GNU coreutils) 8.32
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later &lt;https://gnu.org/licenses/gpl.html&gt;.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Torbjorn Granlund and Richard M. Stallman.</VersionHistory>
    <Example>echo 'cat file  The result is Display what is in the file assume in have abood if have in abood abo
d and abed cat abood the result abood and abed '</Example>
    <RelatedCommands>
      <Command>catman</Command>
      <Command>cat</Command>
      <Command>catman</Command>
      <Command>cat</Command>
    </RelatedCommands>
    <SyntaxAndUsage>Syntax and usage patterns for cat</SyntaxAndUsage>
    <DocumentationLinks>
      <Link>https://www.example.com/cat-documentation</Link>
    </DocumentationLinks>
  </CommandManual>
</Manuals>
```

*When enter number 3 we can search for command and display what is in file command we searching when enter the command is not a available it is print meassge not matching command file found for name command

0 to back to the meun



```
<?xml version="1.0" ?>
<Manuals>
abood
  <CommandManual>
    <CommandName>find</CommandName>
    <CommandDescription>This  manual  page  documents
```

```
Differences found for command 'find':
the differences is: - abood
No differences found for command 'tail'.
No differences found for command 'head'.
No differences found for command 'rmdir'.
No differences found for command 'echo'.
No differences found for command 'sudo'.
```

*when change in the file and enter number 4 verification will code show different between file command.xml and file command 1.xml it is make comparison between to files and display the difference