Faculty of Engineering and Technology
Electrical and Computer Engineering Department
LINUX LABORATORY
ENCS3130

Student's Name: Faten Sultan **Student's Number: 1202750**

Instructor's Name: Dr. Amr Slimi

Assistance's Name: Eng.Tareq Zidan

Project No.2

Section: 1

January, 31 2024

Contents

Introduction

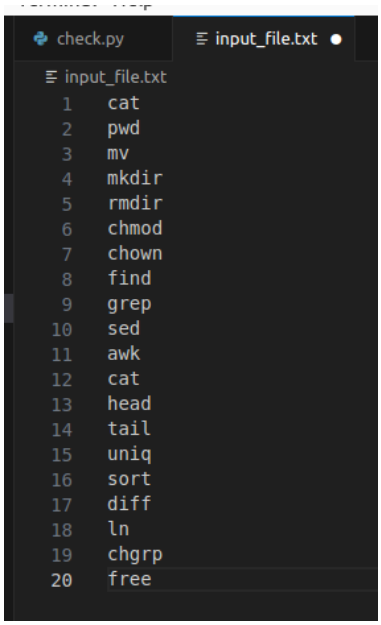
This project aims to create a manual information for Linux shell command using python language with the use of OOP.

In general system manual for shell commands display the name, description, examples, version in addition to any related commands, errors, options, synopsis , author. However in this project we will generate the manual command in xml file using the description of the command, version, related commands as well as real examples for each command only.

Our project will cover 20 Linux shell command

Commands

These are the commands that there manual will be generated:



Theory Procedure

PARTI

Command Manual Generator

This class is used to generate the commands manual. It has the following:

- Instance of the class where it takes command_examples_file as an argument. This file contain the path of the commands file as shown above
- Static method reads commands from a file.
- It calls several methods (extract ..) that are all defined in the commands manual
- It opens the file, reads each line, removes whitespace, and any empty lines.
- It then generates a filename for the output based on the command name and saves the serialized XML content of the command manual to a file. The serialization is done

using XmlSerializer.serialize, which is not defined in the given code but is assumed to be available elsewhere.

- The last method, it takes content and filename as arguments, opens the file in write mode with UTF-8 encoding, and writes the content to the file.

```python
#-----------------------------------------------------------------CommandManualGenerator class -----------------------------------------
class CommandManualGenerator:
    def __init__(self, command_examples_file):
        self.command_manuals = CommandManualGenerator.load_commands(command_examples_file)

    def load_commands(filename):
        # Read the commands from the file, each command on a line
        with open(filename, 'r') as file:
            commands = [line.strip() for line in file if line.strip()]
        return [CommandManual(command, []) for command in commands]


    def generate_manuals(self):
        for command_manual in self.command_manuals:
            command_manual.extract_description()
            command_manual.extract_version()
            command_manual.extract_related_commands()
            command_manual.run_example_commands()
            output_filename = f"{command_manual.command} manual.xml"
            xml_content = XmlSerializer.serialize(command_manual)
            self.save_to_file(content=xml_content, filename=output_filename)

    def save_to_file(self, content, filename):
        with open(filename, "w", encoding='utf-8') as xml_file:
            xml_file.write(content)
```

Command Manual

This class responsible for extracting all the detailed information about the commands We use a module called subprocess, The recommended approach of invoking subprocesses is to use the run() function for all use cases it can handle.

This class involve the following:

- Initial information in case there is no information is available for any command.
- Method run_command: this method attempts to execute a command using the library subprocess. If it returns ‘0’ mean the execution is done successfully and the output will be captured. If the command fails (non-zero return code) or an exception is raised, it returns ‘None’.
- Method (extract_*): This method constructs a manual page command to get the description, version and related commands of all command using man. It captures the output of the man command and uses a regular expression to search for the required information section. If it finds a match, it extracts the data and strips any unwanted whitespace.
- The last method run_example_commands: it tries to append all the data with the self.example either it is append with the output of the command or with an error message if there was a failing during execution

```python
#------------------------------------------------- Command Manual Class ----------------------------
class CommandManual:
    def __init__(self, command, example_inputs=[]):
        self.command = command.strip()
        self.description = "No description available"
        self.version = "No version information available"
        self.related_commands = "No related commands information available"
        self.example_inputs = example_inputs
        self.examples = []

    def run_command(self, command, shell=False):
        try:
            result = subprocess.run(command, capture_output=True, text=True, shell=shell)
            if result.returncode != 0:
                return None
            return result
        except subprocess.CalledProcessError:
            return None

    def extract_description(self):
        man_command = f"man {self.command}"
        result = self.run_command(man_command, shell=True)
        if result and result.stdout:
            match = re.search(r'DESCRIPTION(.+?)(?=\n\n)', result.stdout, re.DOTALL)
            if match:
                self.description = match.group(1).strip()

    def extract_version(self):
        version_command = f"{self.command} --version"
        result = self.run_command(version_command.split())
        if result and result.stdout:
            self.version = result.stdout.split('\n')[0]

    def extract_related_commands(self):
        man_command = f"man {self.command}"
        result = self.run_command(man_command, shell=True)
        if result and result.stdout:
            match = re.search(r'SEE ALSO(.+?)(?=\n\n)', result.stdout, re.DOTALL)
            if match:
                self.related_commands = match.group(1).strip()
```

```python
    def extract_version(self):
        version_command = f"{self.command} --version"
        result = self.run_command(version_command.split())
        if result and result.stdout:
            self.version = result.stdout.split('\n')[0]

    def extract_related_commands(self):
        man_command = f"man {self.command}"
        result = self.run_command(man_command, shell=True)
        if result and result.stdout:
            match = re.search(r"SEE ALSO(.+?)(?=\n\n)", result.stdout, re.DOTALL)
            if match:
                self.related_commands = match.group(1).strip()

    def run_example_commands(self):
        for example_input in self.example_inputs:
            try:
                command_string = ' '.join(example_input)
                result = subprocess.run(command_string, capture_output=True, text=True, shell=True)
                example_output = result.stdout if result.stdout else "No output or error"
                self.examples.append({"input": ' '.join(example_input), "output": example_output})
            except Exception as e:
                self.examples.append({"input": ' '.join(example_input), "output": f"Error: {e}"})
```

XML Serializer

This class Is responsible for the design of the xml file

```python
#-------------------------------------------------XML SERIALIZER ----------------------------------------
# XmlSerializer class
class XmlSerializer:
    def serialize(command_manual):
        xml_content = "<Manual>\n"
        xml_content += "<Manual Command>\n"
        xml_content += f"  <name>{command_manual.command}</name>\n"
        xml_content += f"  <description>{command_manual.description}</description>\n"
        xml_content += f"  <version>{command_manual.version}</version>\n"
        xml_content += f"  <related>{command_manual.related_commands}</related>\n"
        xml_content += "  <examples>\n"
        for example in command_manual.examples:
            xml_content += f"    <example>\n"
            xml_content += f"      <input>{example['input']}</input>\n"
            xml_content += f"      <output>{example['output']}</output>\n"
            xml_content += f"    </example>\n"
        xml_content += "  </examples>\n"
        xml_content += "</Manual Command>\n"
        xml_content += "</Manual>"
        return xml_content
```

These are all the classes that I have used in order to generate a manual for 20 commands using python language

And here is the result

```
check.py    sort_manual.xml ×
sort_manual.xml
1  <Manual>
2  <Manual Command>
3    <name>sort</name>
4    <description>Write sorted concatenation of all FILE(s) to standard output.</description>
5    <version>sort (GNU coreutils) 8.32</version>
6    <related>shuf(1), uniq(1)</related>
7    <examples>
8      <example>
9        <input>sort -n remover.txt</input>
10       <output>5
11 5
12 5
13 7
14 14
15 15
16 22
17 47
18 51
19 447
20 548
21 </output>
22     </example>
23   </examples>
24 </Manual Command>
25 </Command>
```

```
check.py    grep_manual.xml ×
grep_manual.xml
1  <Manual>
2  <Manual Command>
3    <name>grep</name>
4    <description>grep  searches  for  PATTERNS in each FILE.  PATTERNS is one or more patterns separated by
5          newline characters, and grep prints each line that matches a pattern.  Typically  PATTERNS
6          should be quoted when grep is used in a shell command.</description>
7    <version>grep (GNU grep) 3.7</version>
8    <related>Regular Manual Pages
9          awk(1),  cmp(1),  diff(1),  find(1), perl(1), sed(1), sort(1), xargs(1), read(2), pcre(3),
10         pcresyntax(3), pcrepattern(3), terminfo(5), glob(7), regex(7)</related>
11   <examples>
12     <example>
13       <input>grep good file.txt</input>
14       <output>good
15 </output>
16     </example>
17   </examples>
18 </Manual Command>
19 </Command>
```

PARTII

Verification

Our project do more than just generating manual commands, it is also build to verify if the information in the manual.xml is correct or it has been manipulated.

This is done using method

- First we extract/ fetch the part that will be verify using a method called fetch_manual_info) as shown below, we used it to extract version, description and see also (the comments are written and are enough to explain the code)
- The example has not been verify in my code , because the output of the example will be different each time I do a run to the project. However if I should extract them to be verified then they will be treated in the same way as the description, ver.. etc.
- We used another function that verify the data in the file with the real data (the real data has been generated in the same way as we generated above to fill the xml files)

```
1  ###---------------------------------- Verification ----------------------------------
2
3  # Function to fetch manual page information
4  def fetch_manual_info(command):
5      # Fetch the manual page description and related commands
6      man_page = subprocess.run(f"man {command}", shell=True, capture_output=True, text=True)
7      description = version = related = None
8      if man_page.returncode == 0 and man_page.stdout:
9          # Extract description
10         description_match = re.search(r"DESCRIPTION(.+?)(?=\n\n)", man_page.stdout, re.DOTALL)
11         if description_match:
12             description = description_match.group(1).strip()
13
14         # Extract related commands
15         related_match = re.search(r"SEE ALSO(.+?)(?=\n\n)", man_page.stdout, re.DOTALL)
16         if related_match:
17             related = related_match.group(1).strip()
18
19     # Fetch the version information
20     version_info = subprocess.run(f"{command} --version", shell=True, capture_output=True, text=True)
21     if version_info.returncode == 0 and version_info.stdout:
22         version = version_info.stdout.split('\n')[0]
23
24     return description, version, related
```

```
65 def verify_xml(xml_file, command):
66     try:
67         # Parse the generated XML file
68         tree = ET.parse(xml_file)
69         root = tree.getroot()
70     except ET.ParseError as e:
71         return [f"XML parsing error in file: {xml_file} - {e}"]
72
73     # Fetch the original manual information
74     original_description, original_version, original_related = fetch_manual_info(command)
75
76     # Compare each element
77     xml_description = root.find('description').text
78     xml_version = root.find('version').text
79     xml_related = root.find('related').text
80
81     discrepancies = []
82     if xml_description != original_description:
83         discrepancies.append(f"Description mismatch for {command}")
84     if xml_version and original_version and xml_version.strip() != original_version.strip():
85         discrepancies.append(f"Version mismatch for {command}")
86     if xml_related and original_related and xml_related.strip() != original_related.strip():
87         discrepancies.append(f"Related commands mismatch for {command}")
88
89     return discrepancies
```

```python
def verify_all_commands(input_file):
    with open(input_file, 'r') as file:
        commands = [line.strip() for line in file.readlines()]

    for command in commands:
        xml_file = f"{command}_manual.xml"
        if not os.path.exists(xml_file):  # Check if the XML file exists
            print(f"Warning: No XML file found for {command}. Skipping verification
            continue
        discrepancies = verify_xml(xml_file, command)
        if discrepancies:
            for discrepancy in discrepancies:
                print(discrepancy)
        else:
            print(f"No discrepancies found for {command}.")

```

Search Functionality

A search functionality allow the users to find information quickly.

In this function we used the “os” module to interact with the file system particularly to list and access files in a directory. This is a common use case in Python scripts that need to perform file operations like reading, writing, or searching through files.

As a result we used it to make it easy to search on the specific word the user will enter.

```python
#-----------------------------------------------------Searching-----------------------------------
def run_research(word):
    word = word.lower()
    files = os.listdir('.')
    found = False
    for filename in files:
        if filename.endswith('_manual.xml'):
            with open(filename, 'r') as file:
                if word in file.read().lower():
                    print(f"Found in {filename}")
                    found = True

    if not found:
        print("No commands related to this word were found.")

```

Recommendation

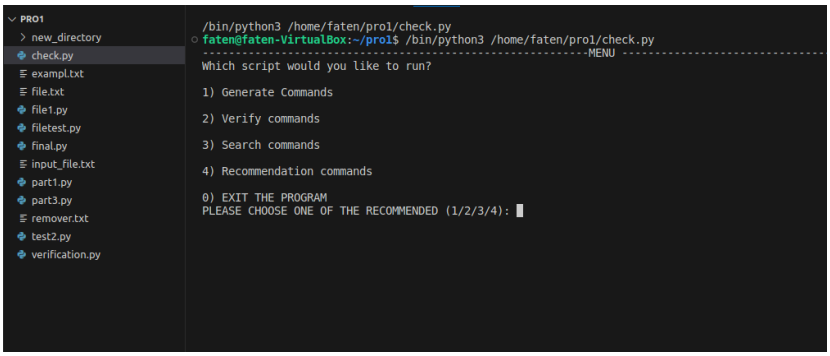
The re module is used in this function to accurately identify and extract the 'SEE ALSO' section from a command's manual page using a regular expression. This is a common approach when you need to parse and extract specific parts from structured or semi-structured text data.

Recommendation function is always used after the user search on a particular word

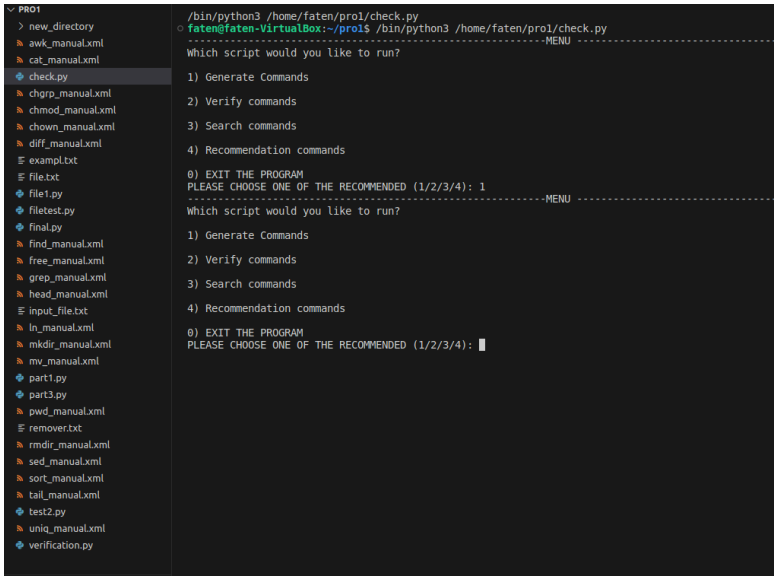
```python
#-----------------------------------------------Recommendation ----------------------------------------
def get_see_also_section(command):
    try:
        result = subprocess.run(['man', command], stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
        man_page = result.stdout

        # Use regex to find the 'SEE ALSO' section
        match = re.search(r'SEE ALSO(.*?)\n\n', man_page, re.S)

        if match:
            return match.group(1).strip()
        else:
            return "No 'SEE ALSO' section found."

    except Exception as e:
        return f"Error occurred: {e}"
```

Results

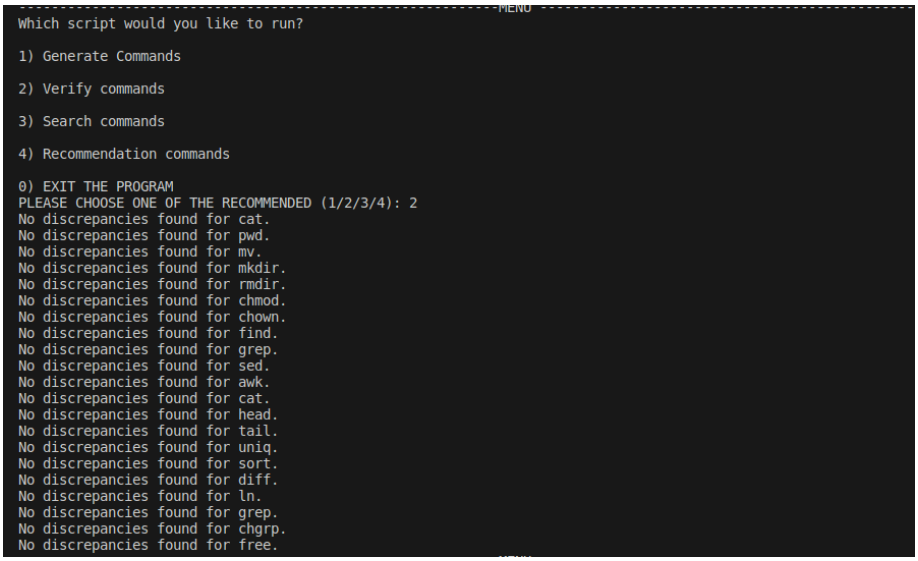
Here is the output of the full project:



Generating all the Commands_manual.xml



Verify that the manual information is the same



Search

```
-------------------------------------------------------MENU -------------------------------------------------------
Which script would you like to run?

1) Generate Commands

2) Verify commands

3) Search commands

4) Recommendation commands

0) EXIT THE PROGRAM
PLEASE CHOOSE ONE OF THE RECOMMENDED (1/2/3/4): 3
Please Enter The Word You Are Looking For: search
Found in grep_manual.xml
Found in find_manual.xml
1) Do You Want To Find the Recommendation commands? By entering a yes or no
yes
please chose a  command :
grep
Command: grep
See Also:
Regular Manual Pages
      awk(1), cmp(1), diff(1), find(1), perl(1), sed(1), sort(1), xargs(1), read(2), pcre(3), pcresyntax(3), pcrepattern(3), terminfo(5), glob(7), regex(7)
```

Recommendation Commands

```
-------------------------------------------------------MENU -------------------------------------------------------
Which script would you like to run?

1) Generate Commands

2) Verify commands

3) Search commands

4) Recommendation commands

0) EXIT THE PROGRAM
PLEASE CHOOSE ONE OF THE RECOMMENDED (1/2/3/4): 4
please enter what command you are lokking at: tail
Command: tail
See Also:
head(1)
```

Testing

We test our verification function by editing on one of the commands file and then see if it will print there is a mismatch or not

We do the test in sed command

As you can see it prints that there is a mismatch in the comparing section

```
-------------------------------------------------------MENU -------------------------------------------------------
Which script would you like to run?

1) Generate Commands

2) Verify commands

3) Search commands

4) Recommendation commands

0) EXIT THE PROGRAM
PLEASE CHOOSE ONE OF THE RECOMMENDED (1/2/3/4): 2
No discrepancies found for cat.
No discrepancies found for pwd.
No discrepancies found for mv.
No discrepancies found for mkdir.
No discrepancies found for rmdir.
No discrepancies found for chmod.
No discrepancies found for chown.
No discrepancies found for find.
No discrepancies found for grep.
No discrepancies found for sed.
No discrepancies found for awk.
No discrepancies found for cat.
No discrepancies found for head.
No discrepancies found for tail.
No discrepancies found for uniq.
Description mismatch for sort
No discrepancies found for diff.
No discrepancies found for ln.
No discrepancies found for grep.
No discrepancies found for chgrp.
No discrepancies found for free.
```

Conclusion

As the project comes to an end, I contemplate the remarkable progress I have achieved in the domain of automated text processing and data retrieval through the utilization of the Python language. Through my proficient creation and execution of Python scripts, I have successfully automated the extraction of information from manual pages for Linux shell commands, thereby highlighting the adaptability and potency of this programming language.

This project shows as an illustration of how Python can be employed to develop tools that are not only robust and efficient but also user-friendly. The knowledge and experiences acquired from this endeavor will undoubtedly shape and motivate future projects, emphasizing our unwavering dedication to excellence in the realm of software development.

References

https://docs.python.org/3/library/subprocess.html
https://www.geeksforgeeks.org/os-module-python-examples/
https://docs.python.org/3/library/os.html
https://docs.python.org/3/library/re.html
https://developers.google.com/edu/python/regular-expressions#:~:text=The%20Python%20%22re%22%20module%20provides%20regular%20expression%20support.&text=str%20%3D%20'an%20example%20word%3Acat!!'&text=The%20code%20match%20%3D%20re.search,e.g.%20'word%3Acat').