SIT 111 COMPUTER SYSTEMS:

TASK 2.2 BASIC LINUX

# Shell

The Linux shell is a smart command-line interpreter. It takes user commands (in the form of texts), executes them, and returns the result. It is smart in the sense that it provides more utilities than just blindly executing the commands.

By default, the **Terminal** application comes with the shell

A shell is a program that runs other programs. Shells can find programs and files quickly, remember what you’ve done before, and store information that you use over and over.

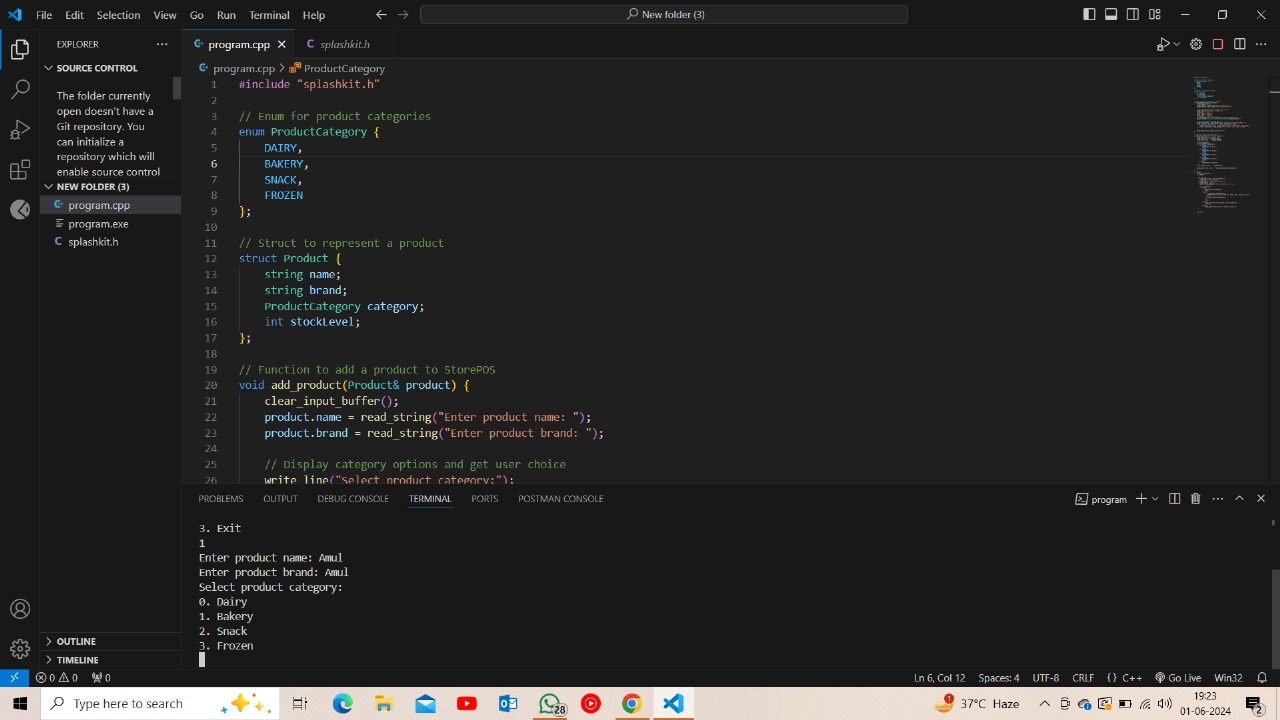
A ‘shell’ can be thought of as an interface between the user and the computer’s operating system, or a ‘wrapper’ around the operating system. It lets you use the computer without needing to understand the low-level details that programmers do.

### **Why use a command line interface?**

In the age of VR/AR, it may seem odd to have to learn a command-line tool. But the fact is that the Command Line Interface (CLI) is the most universal interface there is. It gives users total control over applications.

### **Simple commands**

On the prompt, you can type in some text and press Enter. The shell interprets that as you requesting it to execute the given command.



Following is a screenshot from vs code an app for programming, which comes with an in built terminal and processes the commands after we write the code.

### **Navigating in the shell**

The file path in Linux specifies a location in the file system. The path uses / as the separator. The / directory is the root directory, i.e. top of the filesystem hierarchy.

* Any path starts with / is called an *absolute path*.
* Else, it is called a *relative path*. Relative means in relation to the **current working directory**.

To see what the current directory is, uses pwd. To see the content of the directory, ls. To change to another directory, use cd

In each directory, there are two special directories. . refers to the current directory, and .. refers to the parent directory.

### **Environment Variables**

How to find a specific file before executing it ? The shell does it by consulting **environment variables**. We Think of the shell as a Python program being executed, then environment variables are global variables accessible anywhere inside the program.

### **Connecting Applications**

One distinguishing feature of the Linux shell is its support for stitching up multiple applications together, to build a workflow. It does it by connecting the inputs and outputs of different applications

**MANUAL:**

The man command in Unix-like operating systems is used to display the manual pages for various commands. For example, to see the manual page for the ls command, you would type man ls in the terminal.

The manual page provides detailed information about the command, including its syntax, options, usage examples, and often additional details about its functionality. By using the man command, users can quickly access documentation for commands and learn about their available options and how to use them effectively.

**REGULAR EXPRESSION:**

A regular expression is a string **pattern** that describes a set of strings. It mixes normal ASCII characters with special, metacharacters. These metacharacters have unique matching capabilities:

* . matches any single character. So a.c matches with any string starts with a and ends with c.
* [abc] matches any one of the three characters. For example, [cs]at matches with cat and with sat. Another example [0-9] matches with any digit.
* (RX1|RX2) matches any string that either matches RX1 or RX2.
* + means one or more of the **previous matches**. For example, a+ matches strings with at least one a.
* ? means zero or one appearance of the **previous match**. For example, colou?r matches with both colour and color.
* \* means zero or more appearances of the **previous match**. For example, ba\* will match `b, ba, baaa, baaaa, etc.
* ^ matches the start of the line. For example, ^d matches any line of string that starts with d.
* [^a-z] matches any character **NOT** in the set a-z.
* $ matches the end of the line.

**GREP**

grep is a popular search tool that comes with most Linux distributions. Try to read the manual of this command to understand more. It supports **basic regular expression** by default, meaning that most of the metacharacters, e.g. (,),|,+,$,{,} needs to be preceded by a backslash \

Example:

* grep Linux <file> finds occurrences of the words Linux in file
* ls | grep ^[0-9] finds all files starting with a letter
* ls | grep -v "\.py$" finds all files not ending with .py

**DATA WRANGLING TOOLS:**

One typical example of data wrangling (or data processing) that is very suitable for Linux shells is log processing. Your computer often produces many log messages, from the OS as well as from the applications. The logs are often structured, but they are very long to be able to process by hand or by eye.

**SED**

sed is a **s**tream **ed**itor, whose most popular use is to perform **search & replace** on the file content. The common syntax is below:

sed 's/<SEARCH PATTERN>/<REPLACE PATTERN>/g' <file>

It scans through the file, finding all matches for <SEARCH PATTERN>, which is a regular expression, and replacing them with <REPLACE PATTERN>.

Example:

echo "Hello SIT111" | sed 's/SIT/SIT\_/g'

echo "Hello world SIT111" | sed 's/world//g'

### awk

While sed excels at text transformation, especially at search and replace, awk is another popular tool whose main use is for extracting data.

awk reads the file line by line, automatically splitting the line into tokens, denoted as $1, $2,.... The basic syntax is as:

awk '<CONDITION> {COMMAND}' <file>

Whenever it reads a line satisfying <CONDITION>, it performs COMMAND.

## Introduction to Bash

Bash, or the Bourne Again SHell, is the default command-line interpreter for Linux systems. It provides a powerful interface for interacting with the system, running programs, and managing files and processes. Bash combines features from the original Bourne Shell (sh) with additional enhancements, making it a standard for shell scripting.

### Script Basics

* **Scripts**: Text files containing a series of commands executed by the shell.
* **Creating a Script**: Use any text editor to write Bash commands, then save the file with a .sh extension. In Ubuntu, typically there are a few default text editors such as: vi/vim, nano, and gedit. Alternatively you can also use other text editors such as Emacs or integrated development editors (IDEs) such as Eclipse. This is a personal choice and you can use any editor of your preference.
* **Running a Script**: Make the script executable with chmod +x script.sh and run it using ./script.sh.

**CONTROL STRUCTURES:**

Control structures are fundamental elements in scripting and programming that allow you to **control** the **flow of execution** based on conditions or by repeating actions. They are essential for making decisions within your scripts and handling different inputs or situations.

REFLECTIONS:

How do you know you achieved the learning goals?

I know I've achieved the learning goals by successfully understanding and applying the concepts discussed. For instance, if I can navigate the Linux shell effectively, utilize regular expressions for pattern matching, and comprehend the usage of tools like grep, sed, and awk, I can confidently say I've achieved the learning goals related to shell scripting and data wrangling.

What is the most important things you learned from these and why?

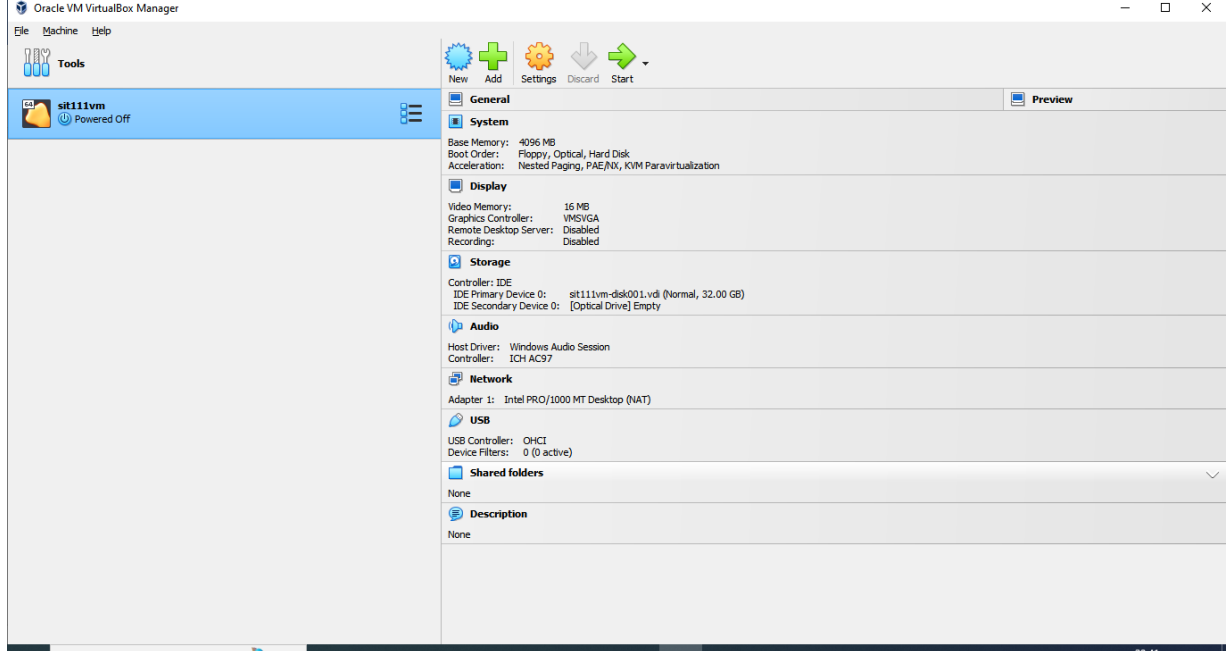
The most crucial takeaway is understanding the versatility and power of the Linux shell. Learning to navigate directories, manipulate files, and automate tasks using shell scripting tools like sed and awk opens up a world of possibilities in terms of efficiency and productivity. Additionally, mastering regular expressions enhances my ability to search for and manipulate text patterns efficiently, a skill applicable in various contexts beyond shell scripting.

How does the content or skills learned here relate to the things you already know?

The content and skills learned here build upon my existing knowledge of command-line interfaces and programming concepts. For instance, my familiarity with programming logic and syntax aids in understanding and writing shell scripts effectively. Similarly, understanding regular expressions in the context of shell scripting extends my knowledge of pattern matching from programming languages like Python or JavaScript. Overall, the content reinforces and expands upon the foundational knowledge I already possess.

ACTIVITIES:

\*INSTALLATION OF VIRTUAL BOX:

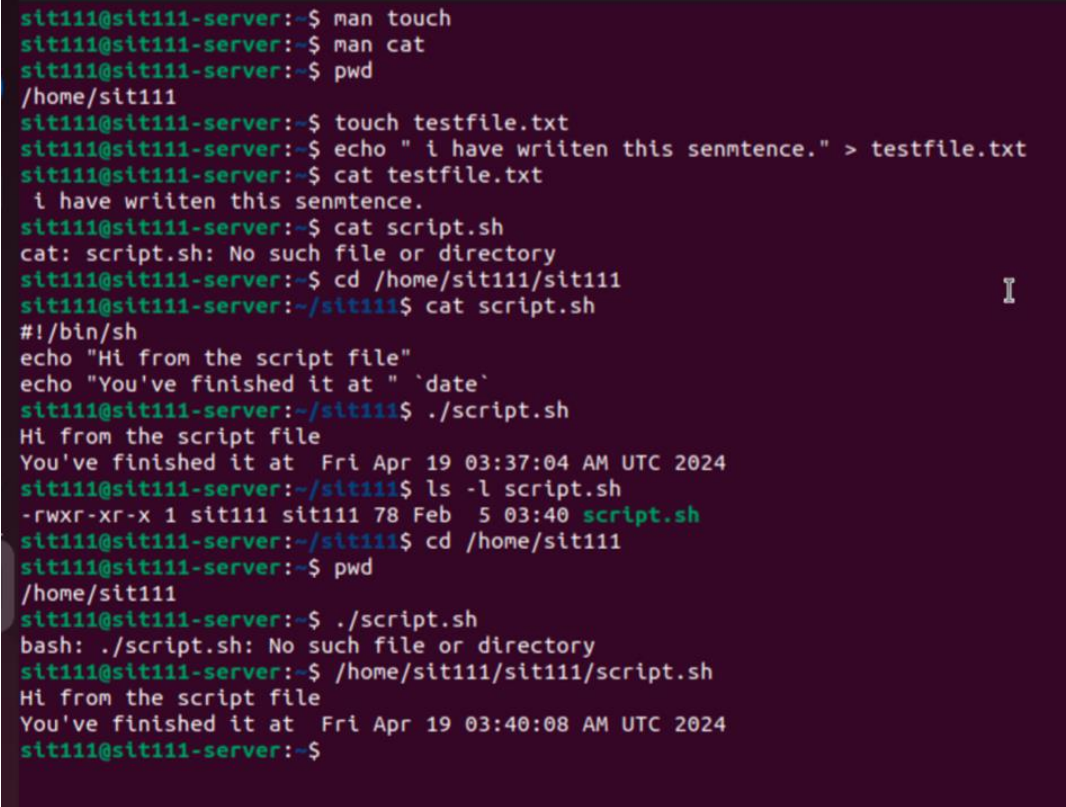
* 

EXERCISE 1

QUESTION 1, 2, 3

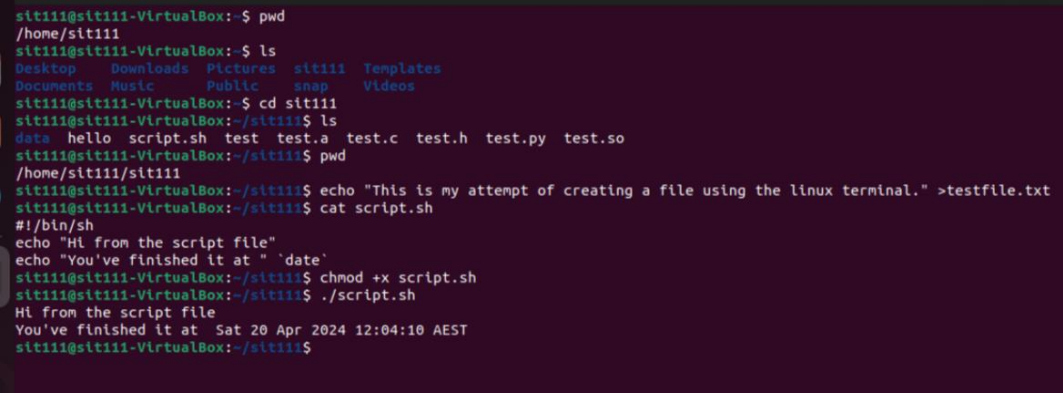


FVFFCRR



QUESTION 4:

SCREENSHOT FROM THE TERMINAL:



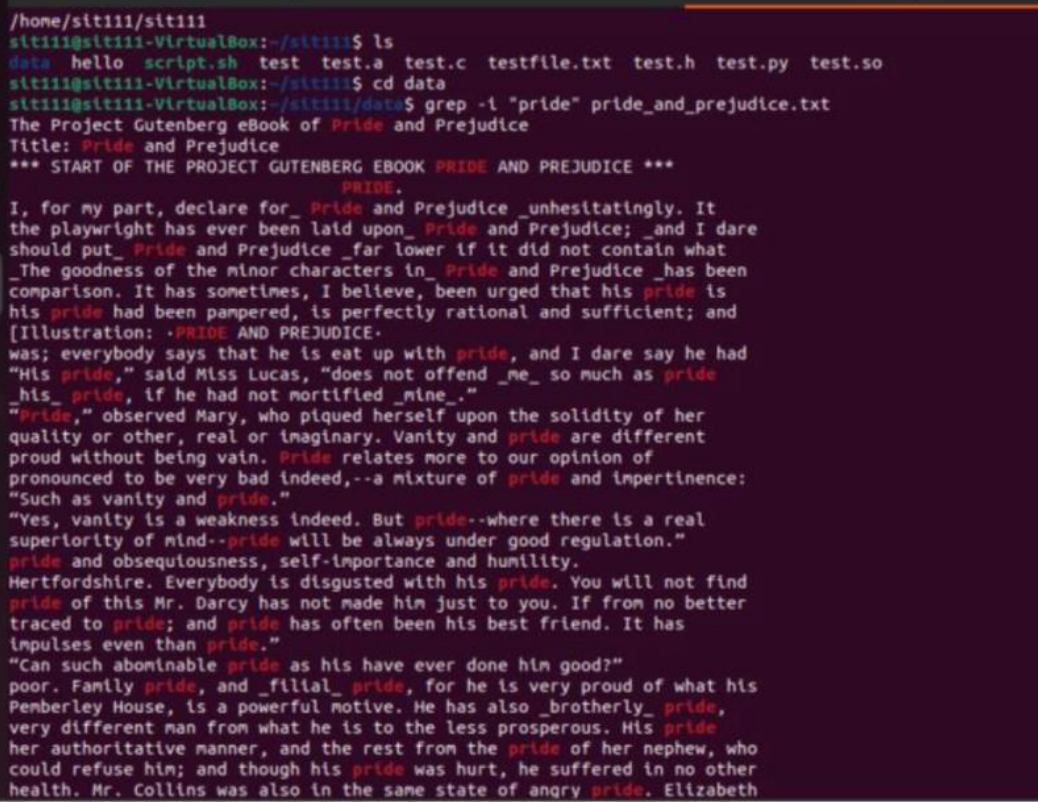
QUESTION 5:

SCREENSHOT FROM THE TERMINAL:

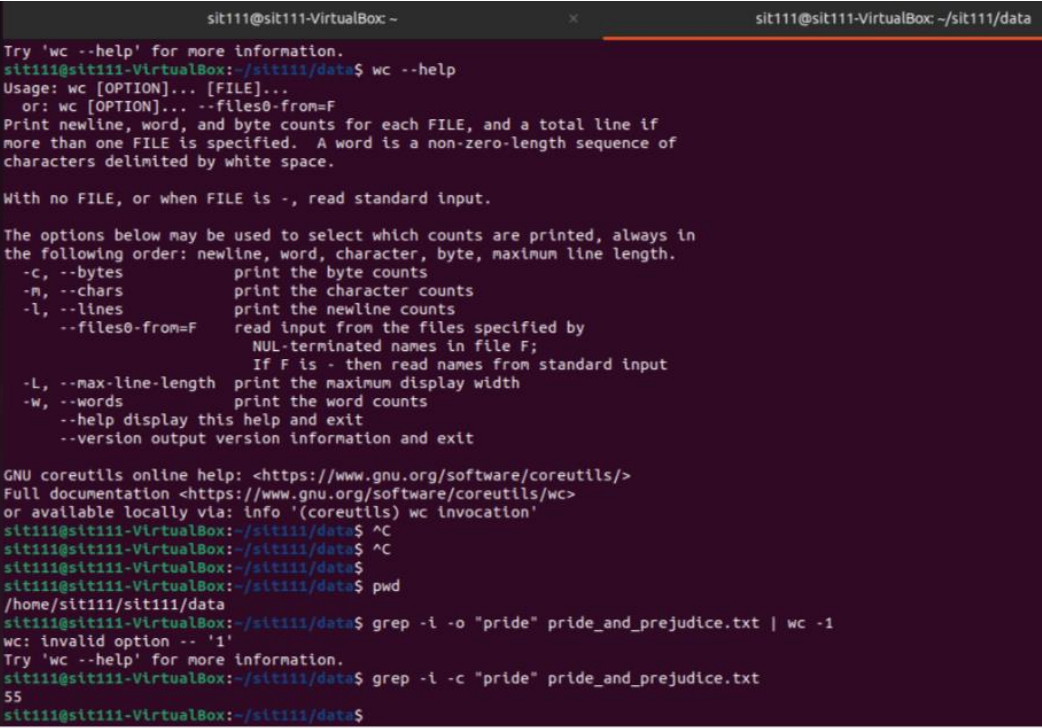


EXERCISE 2 :

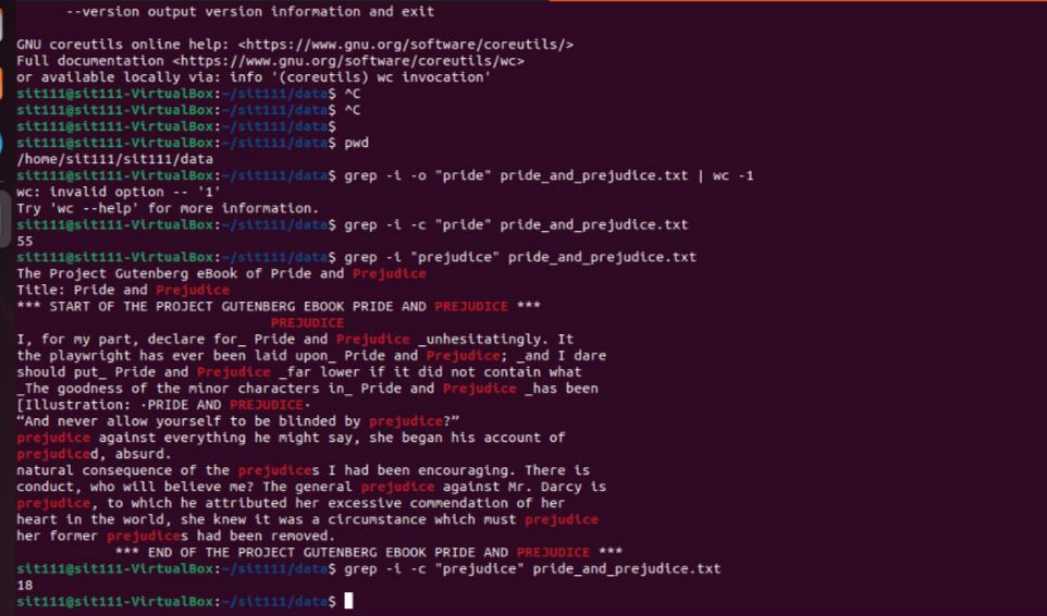
QUESTION 1:



QUESTION 2:

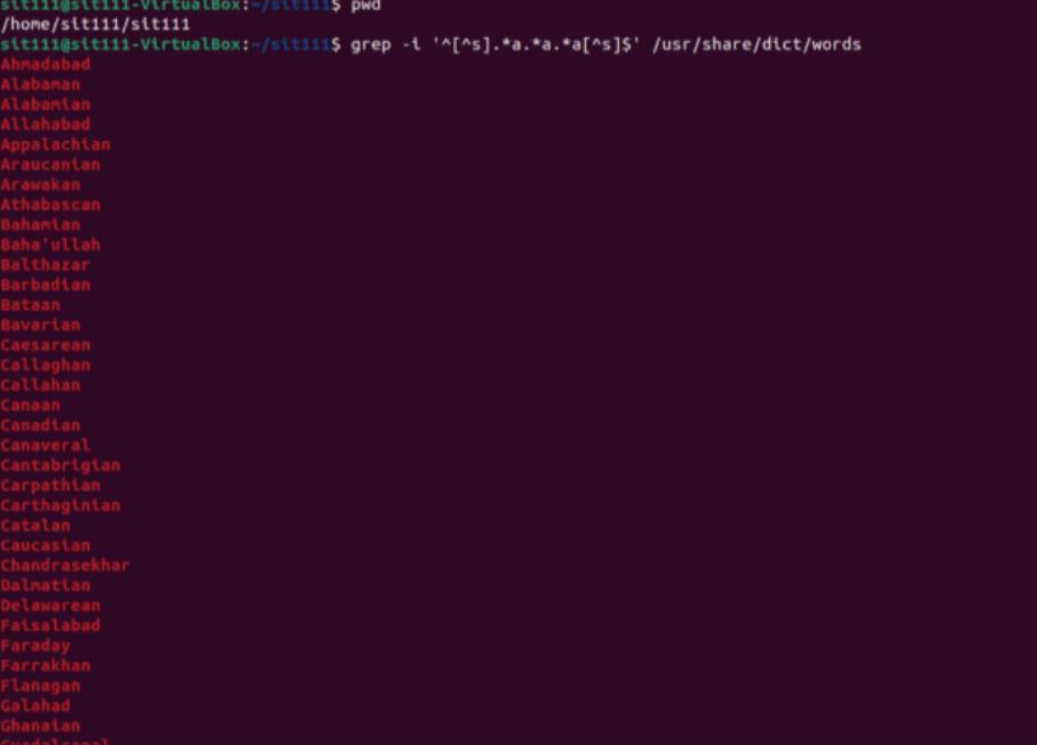


QUESTION 3:



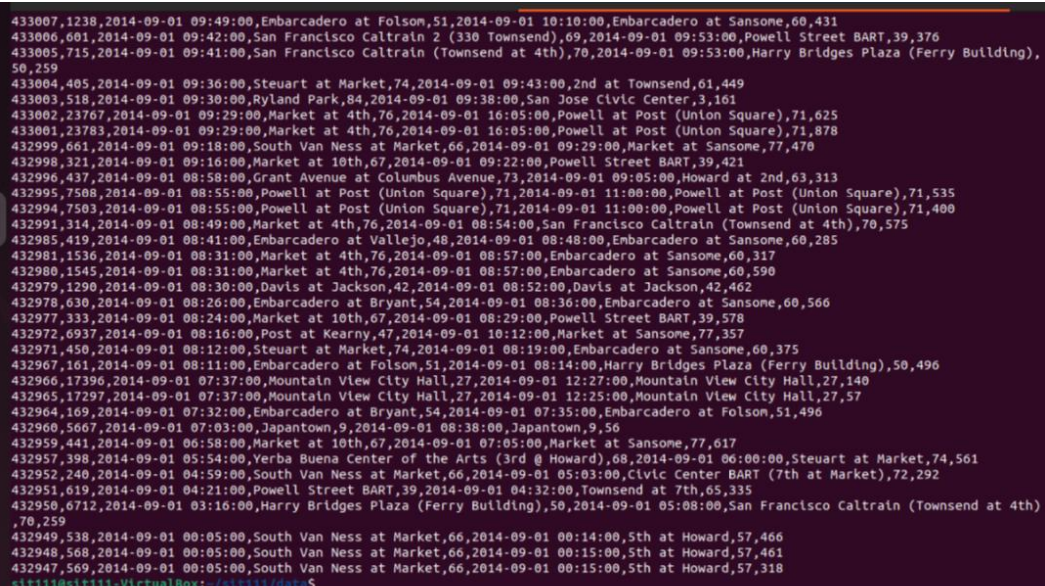
QUESTION 4:

SCREENSHOT FROM THE TERMINAL:



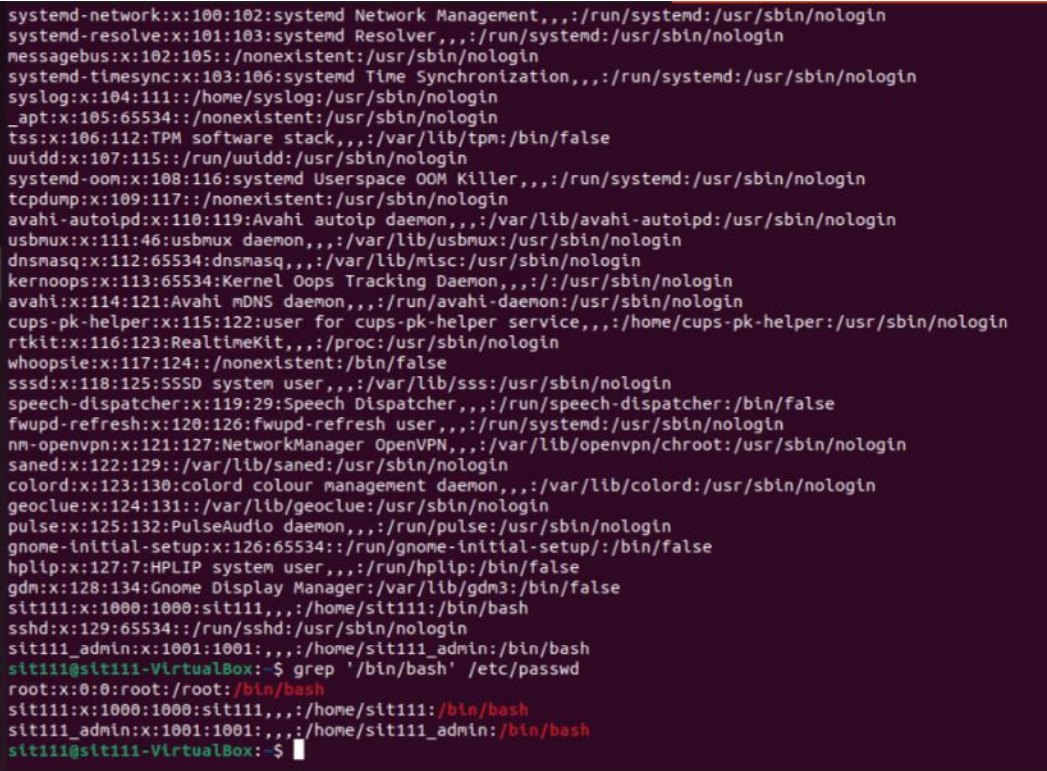
EXERCISE 3:

QUESTION 1:



EXERCISE 4 :

QUESTION 1:



QUESTION2:

ANSWER:

I can understand the contents of these files because they are essentially symbolic links, indicating that they are linked to another directory.