Assignment: Bash Shell Basics

NAME: VEDANTH

REG: 20BCN7115

CAMPUS: VITAP

Task 1: File and Directory Manipulation

File and directory manipulation refers to the process of creating, accessing, modifying, and organizing files and directories (also known as folders) on a computer's file system. It involves performing operations such as creating new files and directories, deleting existing ones, renaming them, moving them to different locations, and retrieving information about them.

1. Create a directory called "my_directory".
2. Navigate into the "my_directory".
3. Create an empty file called "my_file.txt".
4. List all the files and directories in the current directory.
5. Rename "my_file.txt" to "new_file.txt".
6. Display the content of "new_file.txt" using a pager tool of your choice.
7. Append the text "Hello, World!" to "new_file.txt".
8. Create a new directory called "backup" within "my_directory".
9. Move "new_file.txt" to the "backup" directory.
10. Verify that "new_file.txt" is now located in the "backup" directory.
11. Delete the "backup" directory and all its contents.

```
csi@csi:~$ mkdir abc
csi@csi:~$ cd abc
csi@csi:~/abc$ touch my_file.txt
csi@csi:~/abc$ ls
my_file.txt
csi@csi:~/abc$ mv my_file.txt new_file.txt
csi@csi:~/abc$ less new_file.txt
```

```
file_name=”new_file.txt”
text_to_append=”hello,World!”

with open(file_name,’a’) as file:
        file.write(text_to_append)
```

```
csi@csi:~$ cd Documents
csi@csi:~/Documents$ python3 code.py
```

```
csi@csi:~/Documents/abc$ mkdir backup
csi@csi:~/Documents/abc$ mv new_file.txt abc/backup
```

Task 2: Permissions and Scripting

Permissions and scripting are two concepts that are often used in the context of computer systems and programming. Let's take a closer look at each of them:

Permissions: Permissions refer to the rights and privileges granted to users or processes on a computer system. They determine what actions a user or process can perform on files, directories, and other system resources. Permissions are commonly used in operating systems like Unix/Linux and Windows to control access to sensitive information and ensure the security and integrity of the system.

Scripting: Scripting refers to the process of writing and executing scripts, which are sequences of instructions or commands that automate tasks or perform specific actions. Scripts are commonly used in programming and system administration to automate repetitive tasks, configure system settings, or interact with various applications and services.

- Create a new file called "my_script.sh".
- Edit "my_script.sh" using a text editor of your choice and add the following lines:

bash

#!/bin/bash

echo "Welcome to my script!"

echo "Today's date is $(date)." Save and exit the file.

- Make "my_script.sh" executable.
- Run "my_script.sh" and verify that the output matches the expected result.

```
csi@csi:~$ cd ~
csi@csi:~$ nano my_script.sh
csi@csi:~$
```

```
  GNU nano 6.2                                    my_script.sh *
#!/bin/bash
echo "Welcome to my script!"
echo "Today's date is $(date)."
```

```
csi@csi:~$ chmod +x my_script.sh
```

```
csi@csi:~$ ls -l my_script.sh
-rwxrwxr-x 1 csi csi 73 May 28 11:09 my_script.sh
```

```
csi@csi:~$ ./my_script.sh
Welcome to my script!
Today's date is Sun May 28 11:09:54 MDT 2023.
```

Task 3: Command Execution and Pipelines

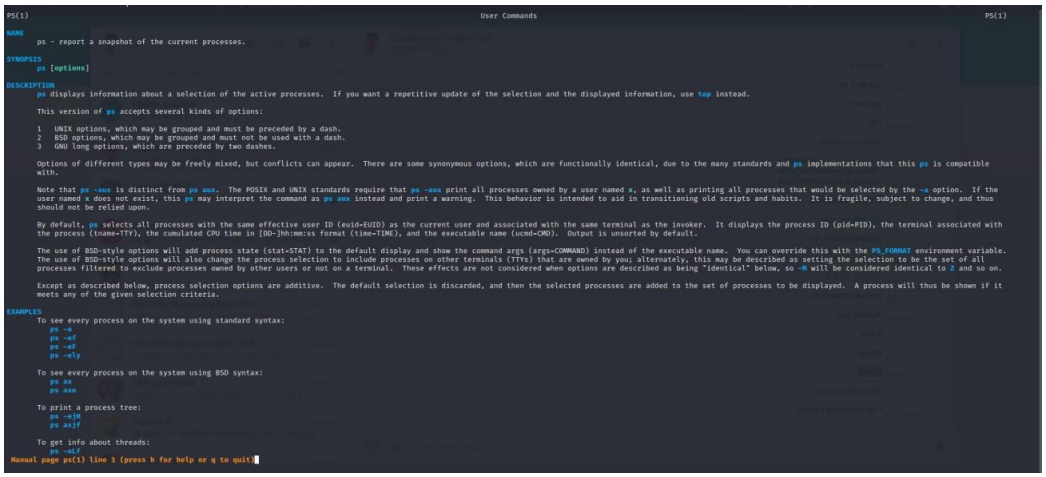
Command execution and pipelines are concepts commonly used in command-line interfaces and scripting environments. They allow you to execute multiple commands sequentially or in parallel, enabling powerful and flexible data processing and manipulation.

In a command-line interface, a command is a specific instruction or task that you provide to the system to perform. Commands can be executed individually, but pipelines allow you to connect multiple commands together, using the output of one command as the input for another. This allows you to create complex workflows and perform more advanced data processing.

- List all the processes running on your system using the "ps" command.
- Use the "grep" command to filter the processes list and display only the processes with "bash" in their name.
- Use the "wc" command to count the number of lines in the filtered output.

```
csi@csi:~$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.2 166552 12032 ?        Ss   09:06   0:02 /sbin/init
root           2  0.0  0.0      0     0 ?        S    09:06   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   09:06   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   09:06   0:00 [rcu_par_gp]
root           5  0.0  0.0      0     0 ?        I<   09:06   0:00 [netns]
root           7  0.0  0.0      0     0 ?        I<   09:06   0:00 [kworker/0:0H-events_highpri]
root          10  0.0  0.0      0     0 ?        I<   09:06   0:00 [mm_percpu_wq]
root          11  0.0  0.0      0     0 ?        S    09:06   0:00 [rcu_tasks_rude_]
root          12  0.0  0.0      0     0 ?        S    09:06   0:00 [rcu_tasks_trace]
root          13  0.0  0.0      0     0 ?        S    09:06   0:00 [ksoftirqd/0]
root          14  0.0  0.0      0     0 ?        I    09:06   0:01 [rcu_sched]
root          15  0.0  0.0      0     0 ?        S    09:06   0:00 [migration/0]
root          16  0.0  0.0      0     0 ?        S    09:06   0:00 [idle_inject/0]
root          17  0.0  0.0      0     0 ?        S    09:06   0:00 [cpuhp/0]
```

```
csi@csi:~$ man ps
```




```
csi@csi:~$ ps aux | grep bash
csi         5092  0.0  0.1   6076  5036 pts/0    Ss+  09:07   0:00 /bin/bash
csi       214939  0.0  0.1   6080  5088 pts/1    Ss+  11:01   0:00 /bin/bash
csi       219384  0.0  0.1   6080  5108 pts/2    Ss   11:04   0:00 /bin/bash
csi       229095  0.0  0.0   4020  2080 pts/2    S+   11:11   0:00 grep --color=auto bash
```

```
csi@csi:~$ ps aux | grep bash | wc -l
4
```