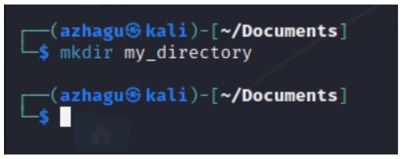
Assignment 2: Bash Shell Basics

Name: Alagappan
Date: 30-05-2023
Regs_No: 20BCE7211
Campus: VIT-AP

Task 1: File and Directory Manipulation

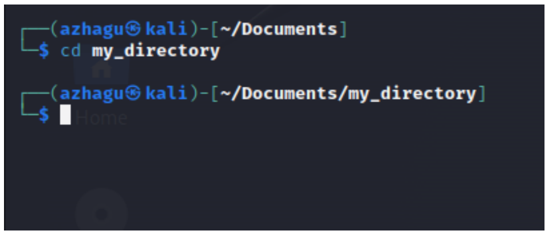
1. Create a directory called "my_directory".



```
┌──(azhagu㉿kali)-[~/Documents]
└─$ mkdir my_directory

┌──(azhagu㉿kali)-[~/Documents]
└─$ █
```

This command creates a new directory named "my_directory" in the current working directory.

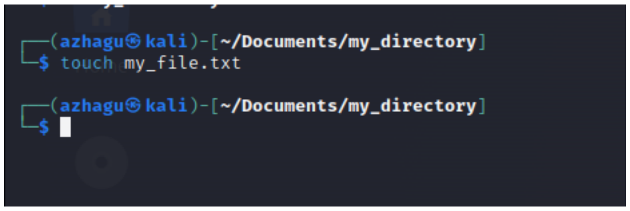
2. Navigate into the "my_directory".



```
┌──(azhagu㉿kali)-[~/Documents]
└─$ cd my_directory

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ █
```

This command changes the current working directory to "my_directory".

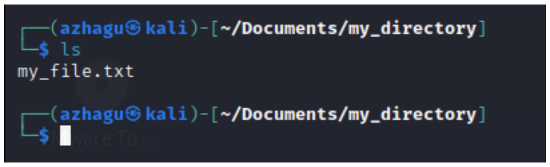
3. Create an empty file called "my_file.txt".



```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ touch my_file.txt

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ █
```

The touch command is used to create an empty file. In this case, it creates a file named "my_file.txt" in the current directory.

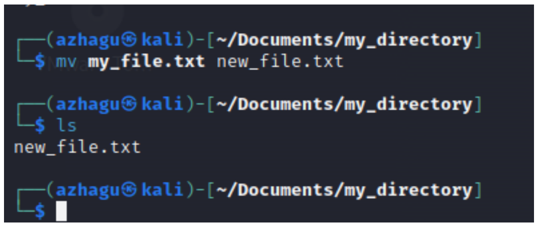
4. List all the files and directories in the current directory.



```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ ls
my_file.txt

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ █
```

The ls command lists the files and directories in the current directory.

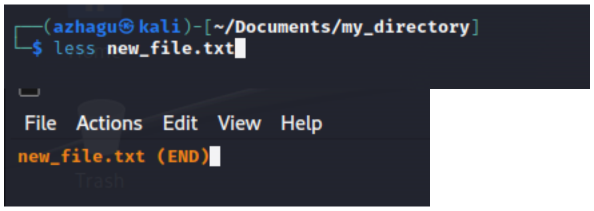
5. Rename "my_file.txt" to "new_file.txt".



```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ mv my_file.txt new_file.txt

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ ls
new_file.txt

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ 
```

The mv command is used to move or rename files. In this case, it renames the file "my_file.txt" to "new_file.txt"

6. Display the content of "new_file.txt" using a pager tool of your choice

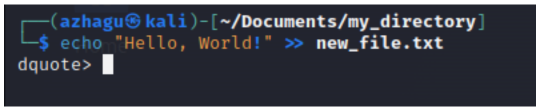


```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ less new_file.txt
```

```
new_file.txt (END)
```

The less command is a pager tool that allows you to view the content of a file page by page. In this case, it displays the content of the file "new_file.txt". You can scroll through the content using the arrow keys and press "q" to exit.

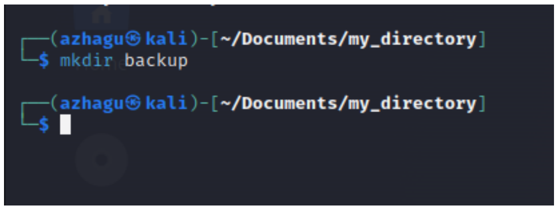
7. Append the text "Hello, World!" to "new_file.txt".



```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ echo "Hello, World!" >> new_file.txt
dquote> 
```

The echo command is used to print text. The >> operator is used to append the output to a file. In this case, it appends the text "Hello, World!" to the file "new_file.txt"

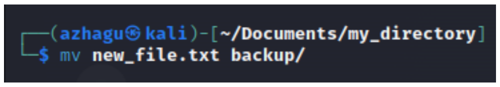
8. Create a new directory called "backup" within "my_directory".



```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ mkdir backup

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ 
```

This command creates a new directory named "backup" within the "my_directory" directory.

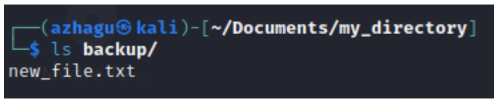
9. Move "new_file.txt" to the "backup" directory.



```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ mv new_file.txt backup/
```

This command moves the file "new_file.txt" to the "backup" directory.

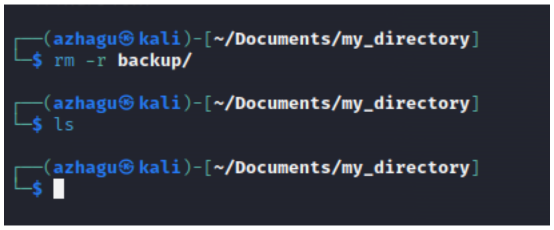
10. Verify that "new_file.txt" is now located in the "backup" directory.



```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ ls backup/
new_file.txt
```

This command lists the contents of the "backup" directory to verify that "new_file.txt" is present there.

11. Delete the "backup" directory and all its contents.

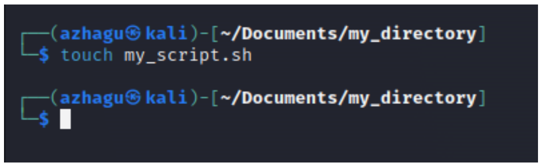


```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ rm -r backup/

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ ls

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ 
```

The rm command is used to remove files and directories. The -r option is used to recursively remove directories and their contents. In this case, it deletes the "backup" directory and all its contents.

Task 2: Permissions and Scripting

1. Create a new file called "my_script.sh".

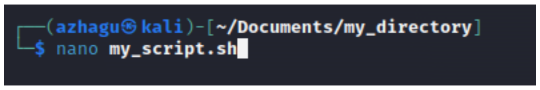


```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ touch my_script.sh

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ 
```

This command creates a new file named "my_script.sh" in the current directory.

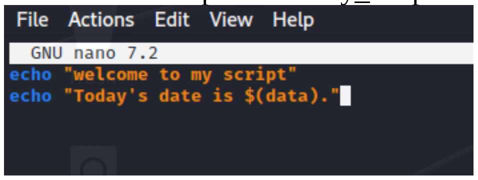
2. Edit "my_script.sh" using a text editor of your choice and add the following lines:
bash

#!/bin/bash
echo "Welcome to my script!"
echo "Today's date is $(date)."
Save and exit the file.

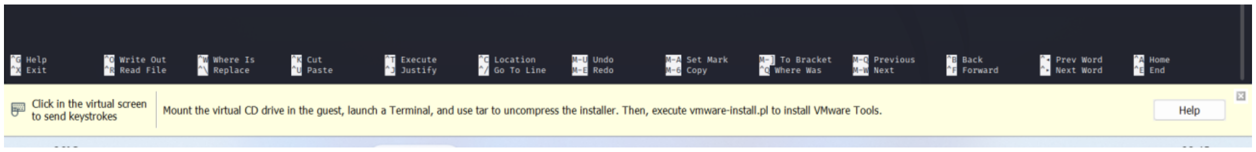


```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ nano my_script.sh
```

This command opens the "my_script.sh" file in the nano text editor, allowing you to edit the file



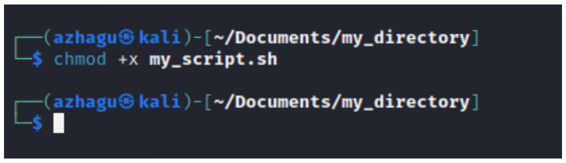
```
  GNU nano 7.2
echo "welcome to my script"
echo "Today's date is $(data)."
```



Mount the virtual CD drive in the guest, launch a Terminal, and use tar to uncompress the installer. Then, execute vmware-install.pl to install VMware Tools.

These lines are added to the "my_script.sh" file. The first line specifies the interpreter (#!/bin/bash), and the subsequent lines use the echo command to print text.

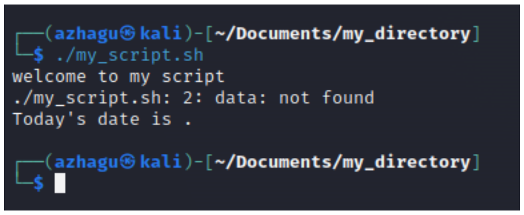
3. Make "my_script.sh" executable



```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ chmod +x my_script.sh

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ 
```

The chmod command is used to change the permissions of a file. The +x option makes the file executable, allowing it to be run as a script.

4. Run "my_script.sh" and verify that the output matches the expected result.

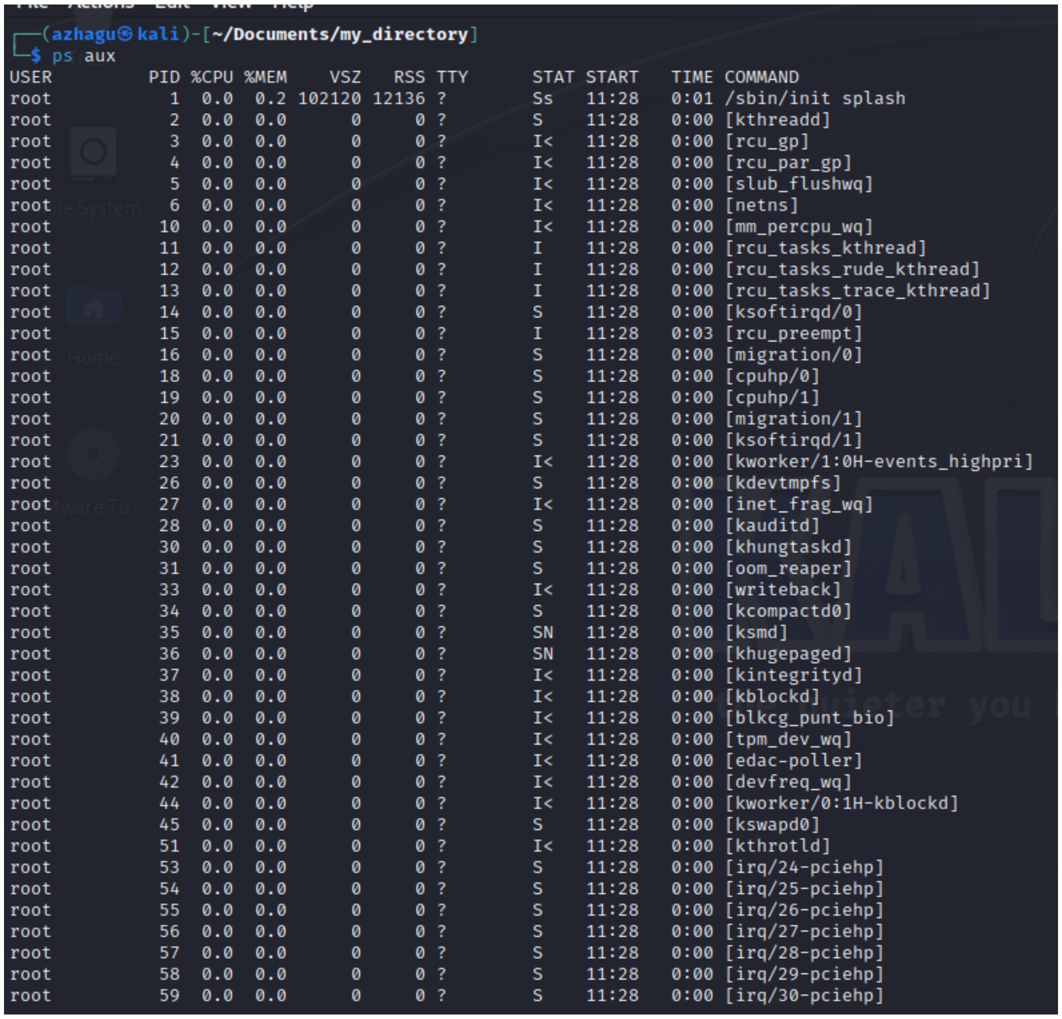


```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ ./my_script.sh
welcome to my script
./my_script.sh: 2: data: not found
Today's date is .

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ 
```

This command executes the "my_script.sh" file, and the output should display the text specified in the script, including the current date and time

Task 3: Command Execution and Pipelines

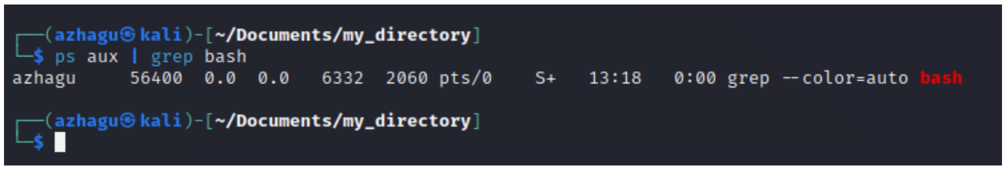
1. List all the processes running on your system using the "ps" command.



```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.2 102120 12136 ?        Ss   11:28   0:01 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    11:28   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   11:28   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   11:28   0:00 [rcu_par_gp]
root           5  0.0  0.0      0     0 ?        I<   11:28   0:00 [slub_flushwq]
root           6  0.0  0.0      0     0 ?        I<   11:28   0:00 [netns]
root          10  0.0  0.0      0     0 ?        I<   11:28   0:00 [mm_percpu_wq]
root          11  0.0  0.0      0     0 ?        I    11:28   0:00 [rcu_tasks_kthread]
root          12  0.0  0.0      0     0 ?        I    11:28   0:00 [rcu_tasks_rude_kthread]
root          13  0.0  0.0      0     0 ?        I    11:28   0:00 [rcu_tasks_trace_kthread]
root          14  0.0  0.0      0     0 ?        S    11:28   0:00 [ksoftirqd/0]
root          15  0.0  0.0      0     0 ?        I    11:28   0:03 [rcu_preempt]
root          16  0.0  0.0      0     0 ?        S    11:28   0:00 [migration/0]
root          18  0.0  0.0      0     0 ?        S    11:28   0:00 [cpuhp/0]
root          19  0.0  0.0      0     0 ?        S    11:28   0:00 [cpuhp/1]
root          20  0.0  0.0      0     0 ?        S    11:28   0:00 [migration/1]
root          21  0.0  0.0      0     0 ?        S    11:28   0:00 [ksoftirqd/1]
root          23  0.0  0.0      0     0 ?        I<   11:28   0:00 [kworker/1:0H-events_highpri]
root          26  0.0  0.0      0     0 ?        S    11:28   0:00 [kdevtmpfs]
root          27  0.0  0.0      0     0 ?        I<   11:28   0:00 [inet_frag_wq]
root          28  0.0  0.0      0     0 ?        S    11:28   0:00 [kauditd]
root          30  0.0  0.0      0     0 ?        S    11:28   0:00 [khungtaskd]
root          31  0.0  0.0      0     0 ?        S    11:28   0:00 [oom_reaper]
root          33  0.0  0.0      0     0 ?        I<   11:28   0:00 [writeback]
root          34  0.0  0.0      0     0 ?        S    11:28   0:00 [kcompactd0]
root          35  0.0  0.0      0     0 ?        SN   11:28   0:00 [ksmd]
root          36  0.0  0.0      0     0 ?        SN   11:28   0:00 [khugepaged]
root          37  0.0  0.0      0     0 ?        I<   11:28   0:00 [kintegrityd]
root          38  0.0  0.0      0     0 ?        I<   11:28   0:00 [kblockd]
root          39  0.0  0.0      0     0 ?        I<   11:28   0:00 [blkcg_punt_bio]
root          40  0.0  0.0      0     0 ?        I<   11:28   0:00 [tpm_dev_wq]
root          41  0.0  0.0      0     0 ?        I<   11:28   0:00 [edac-poller]
root          42  0.0  0.0      0     0 ?        I<   11:28   0:00 [devfreq_wq]
root          44  0.0  0.0      0     0 ?        I<   11:28   0:00 [kworker/0:1H-kblockd]
root          45  0.0  0.0      0     0 ?        S    11:28   0:00 [kswapd0]
root          51  0.0  0.0      0     0 ?        I<   11:28   0:00 [kthrotld]
root          53  0.0  0.0      0     0 ?        S    11:28   0:00 [irq/24-pciehp]
root          54  0.0  0.0      0     0 ?        S    11:28   0:00 [irq/25-pciehp]
root          55  0.0  0.0      0     0 ?        S    11:28   0:00 [irq/26-pciehp]
root          56  0.0  0.0      0     0 ?        S    11:28   0:00 [irq/27-pciehp]
root          57  0.0  0.0      0     0 ?        S    11:28   0:00 [irq/28-pciehp]
root          58  0.0  0.0      0     0 ?        S    11:28   0:00 [irq/29-pciehp]
root          59  0.0  0.0      0     0 ?        S    11:28   0:00 [irq/30-pciehp]
```

The ps command is used to display information about active processes. The aux options provide a detailed list of all processes running on the system.

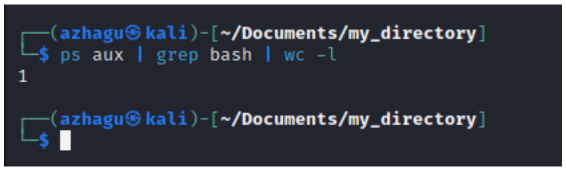
2. Use the "grep" command to filter the processes list and display only the processes with "bash" in their name.



```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ ps aux | grep bash
azhagu     56400  0.0  0.0   6332  2060 pts/0    S+   13:18   0:00 grep --color=auto bash

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ 
```

The grep command is used to search for specific patterns in the input. In this case, it filters the output of the ps aux command to display only the processes that contain the word "bash"

3. Use the "wc" command to count the number of lines in the filtered output.



```
┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ ps aux | grep bash | wc -l
1

┌──(azhagu㉿kali)-[~/Documents/my_directory]
└─$ 
```

The wc command is used to count the number of lines, words, and characters in the input. The -l option tells wc to count only the lines. In this case, it counts the number of lines in the filtered output of the previous command, giving the total number of processes with "bash" in their name.