**Chapter 1**

Q1. What Is Linux?

Ans. Linux® is an [open source](https://www.redhat.com/en/topics/open-source/what-is-open-source) operating system (OS). An [operating system](https://www.redhat.com/en/technologies/linux-platforms/old-enterprise-linux) is the software that directly manages a system’s hardware and resources, like CPU, memory, and [storage](https://www.redhat.com/en/topics/data-storage/software-defined-storage). The OS sits between applications and hardware and makes the connections between all of your software and the physical resources that do the work.

Q2. What is Linux Kernel?

The Linux® kernel is the main component of a [Linux operating system (OS)](https://www.redhat.com/en/topics/linux/what-is-linux) and is the core interface between a computer’s hardware and its processes. It communicates between the 2, managing resources as efficiently as possible.

The kernel is so named because—like a seed inside a hard shell—it exists within the OS and controls all the major functions of the hardware, whether it’s a phone, laptop, server, or any other kind of computer.

The kernel has 4 jobs:

1. **Memory management:** Keep track of how much memory is used to store what, and where
2. **Process management:** Determine which processes can use the central processing unit (CPU), when, and for how long
3. **Device drivers:** Act as mediator/interpreter between the hardware and processes
4. **System calls and security:** Receive requests for service from the processes

Q3. What is Linux Shell?

The shell can be defined as a command interpreter within an operating system like Linux/GNU or Unix. It is a program that runs other programs. The shell facilitates every user of the computer as an interface to the Unix/GNU Linux system. Hence, the user can execute different tools/utilities or commands with a few input data.

Q4. Unix philosophy?

The Unix philosophy emphasizes building simple, compact, clear, modular, and [extensible](https://en.wikipedia.org/wiki/Extensibility) code that can be easily maintained and repurposed by developers other than its creators. The Unix philosophy favors [composability](https://en.wikipedia.org/wiki/Composability) as opposed to [monolithic design](https://en.wikipedia.org/wiki/Monolithic_application).

Q5. The role of shells in the Linux environment

Q6. Shebang

This sequence of characters (#!) is called **shebang** and is used to tell the operating system which interpreter to use to parse the rest of the file.

If a shebang is not specified and the user running the Bash script is using another Shell the script will be parsed by whatever the default interpreter is used by that Shell. For example, the default interpreter for bash is bash and for zsh is sh. To ensure that your script will always be interpreted with Bash you’ll need to specify the executable path using shebang.

Q7. Setting up permissions on a script & Execute, Debug?

To check permissions - $ls -al

Text, calendar

Description automatically generated with medium confidence

Allowing everyone

$ chmod +x <filename> or chmod 0755 <filename>

Only allow owner to execute the script :-

Chmod u+x <filename> or chmod 0700 <filename>

For permission for read and execute only :-

Chmod ug=rx <scriptname>

Remove read and execute permission for the group:-

Chmod ug= script.sh



<https://bash.cyberciti.biz/guide/Setting_up_permissions_on_a_script>

<https://www.cyberciti.biz/tips/debugging-shell-script.html>

Q8. Display the value of shell variables?

<https://www.digitalocean.com/community/tutorials/how-to-read-and-set-environmental-and-shell-variables-on-linux>

$printenv



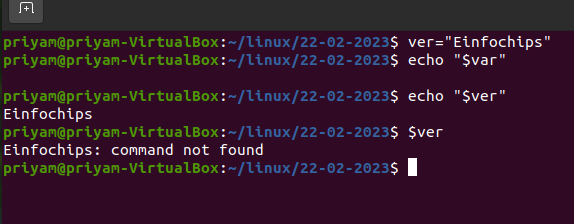
$set

Text

Description automatically generated with medium confidence

$set | less

$(set -o posix; set)



Q9. The export statement



Q10. Unset shell and environment variables

To unset

Text

Description automatically generated

Q11. Getting User Input Via Keyboard

<https://www.masteringunixshell.net/qa24/bash-how-to-read-from-keyboard.html#:~:text=To%20read%20input%20from%20the,variable%20use%20the%20read%20command.&text=If%20you%20don't%20give,while%20reading%20from%20a%20keyboard>.

Text

Description automatically generated

Q12. Perform arithmetic operations

Text

Description automatically generated

Q13. Create an integer variable

A picture containing text

Description automatically generated

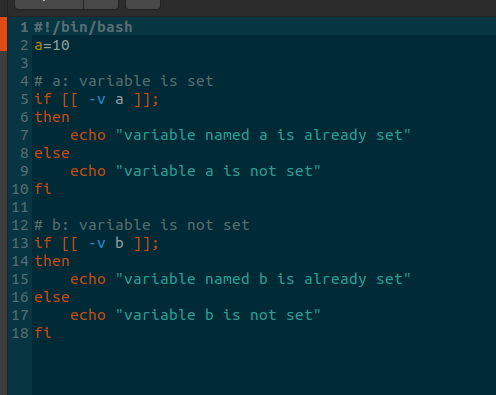
Q14. Create the constants variable

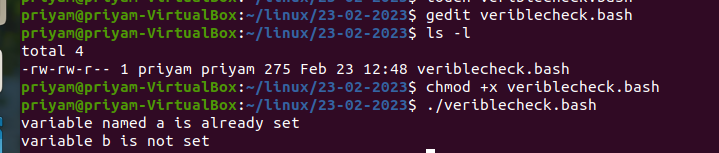


Text

Description automatically generated

Q15. Bash variable existence check





Q16. Recalling command history

Text

Description automatically generated

Q17. Path name expansion ?



Q18. Create and use aliases

Text

Description automatically generated

To remove alias use “unalias”



Q19. The tilde expansion



Q20.Using aliases

Text

Description automatically generated

Q21. Test command

* -f: regular file (returns false for a directory)
* -d: directory
* -b: block (such as /dev/sda1)
* -L or -h: symlink
* -S: socket

( -eq -ne -gt -ge -lt -le )

Text

Description automatically generated with low confidence

https://www.geeksforgeeks.org/shell-scripting-test-command/

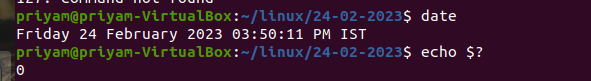
**Chapter 2**

1 . If structures to execute code based on a condition If..else..fi Nested ifs Multilevel if-then-else

Text

Description automatically generated

2. The exit status of a command?



Text

Description automatically generated

3. Conditional execution?

4. Numeric comparison?

Graphical user interface, text

Description automatically generated

1. String comparison.

Text

Description automatically generated

Q6. File attributes comparisons?

Text

Description automatically generated

Q7 . Shell command line parameters

Graphical user interface, application, table

Description automatically generated

Q8 . How to use positional parameters



Graphical user interface, text

Description automatically generated

Q9. Parameters Set by the Shell

Q10. Create usage messages

Q11. Exit command

Q12. The case statement

Text

Description automatically generated

Q13. Dealing with case sensitive pattern.

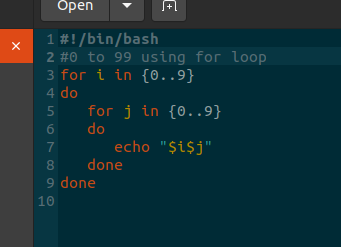
**Chapter 3**

1.The for loop statement

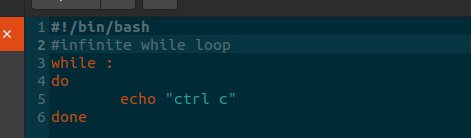
Text

Description automatically generated

2. Nested for loop statement



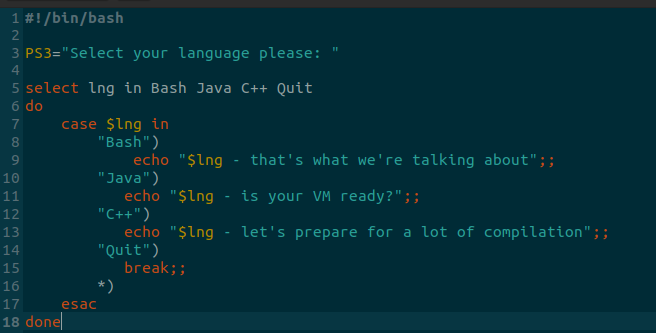
3. The while loop statement. Use of : to set infinite while loop. The until loop statement



Text

Description automatically generated

4. The select loop statement? Exit the select loop statement? Using the break statement?



5. Using the continue statement

Graphical user interface, text

Description automatically generated

**Chapter 4**

Q1. Command substitution?

Graphical user interface, application

Description automatically generated

Q2.Input and Output Standard input Standard output

Text

Description automatically generated

Q3. Standard error?

Q4. Empty file creation?

Touch command

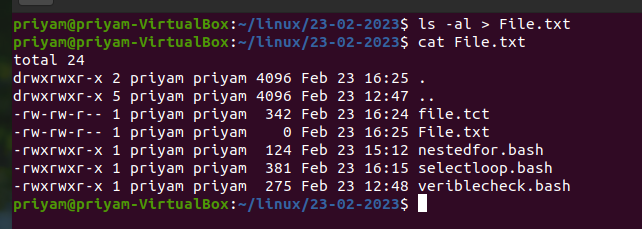


Text

Description automatically generated

Q5. Redirection of standard error

Redirection of standard output



Appending redirected output

Redirection of both standard error and output

Writing output to files Assigns the file descriptor (fd) to file for output

Assigns the file descriptor (fd) to file for input

Closes the file descriptor (fd)

Opening the file descriptors for reading and writing

Reads from the file descriptor (fd)

Executes commands and send output to the file descriptor (fd)

**Chapter 5**

Q1. Linking Commands?

Q2. Multiple commands?

Q3. Putting jobs in background?

Sol.

1. Using & in the end of ./<>.bash, i.e. ./longprogram.bash &
2. Using “” nohup tome < -- > &”” to put job in background

Text

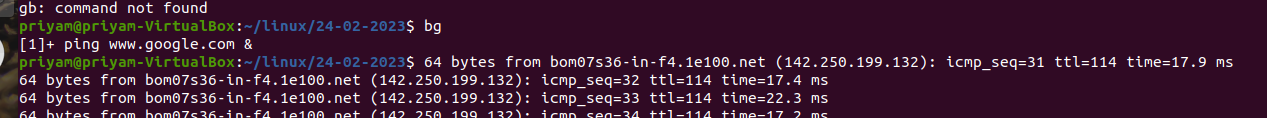
Description automatically generated

“ fg “ to bring again in forgroung

* **Ctrl+C**: Sends SIGINT, signal 2, to the process—if it is accepting input—and tells it to terminate.
* **Ctrl+D**: Sends SISQUIT, signal 3, to the process—if it is accepting input—and tells it to quit.
* **Ctrl+Z**: Sends SIGSTP, signal 20, to the process and tells it to stop (suspend) and become a background process.
* **jobs**: Lists the background jobs and shows their job number.
* **bg***job\_number*: Restarts a background process. If you don’t provide a job number the last process that was turned into a background task is used.
* **fg***job\_number*: brings a background process into the foreground and restarts it. If you don’t provide a job number the last process that was turned into a background task is used.
* *commandline***&**: Adding an ampersand & to the end of a command line executes that command as a background task, that is running.
* **kill %***job\_number*: Sends SIGTERM, signal 15, to the process to terminate it.
* **kill 9** **%***job\_number*: Sends SIGKILL, signal 9, to the process and terminates it abruptly.

Text

Description automatically generated



Text

Description automatically generated

Q4. Pipes?



Text

Description automatically generated

Q5. How to use pipes to connect programs?

Q6. Input redirection in pipes Output redirection in pipes?