Operating System Lab

Basics of bash scripting and learn how to write a bash script

Operating system- An operating system (OS) is a software that manages a computer's hardware and software resources, acting as an intermediary between the user and the computer's hardware.

Example- Microsoft Windows, Apple's macOS, and Linux.

The operating system is the core software that controls computer hardware, runs applications, and keeps all operations coordinated. It manages both hardware and software resources, allowing users to interact effectively with their devices. Serving as a bridge between the user and the machine, the OS ensures smooth performance by providing a platform for program execution and managing access to system resources.

Without an operating system, a computer could not perform any tasks, and its hardware would remain idle. It serves as the foundation of all devices, including smartphones, laptops, and servers.

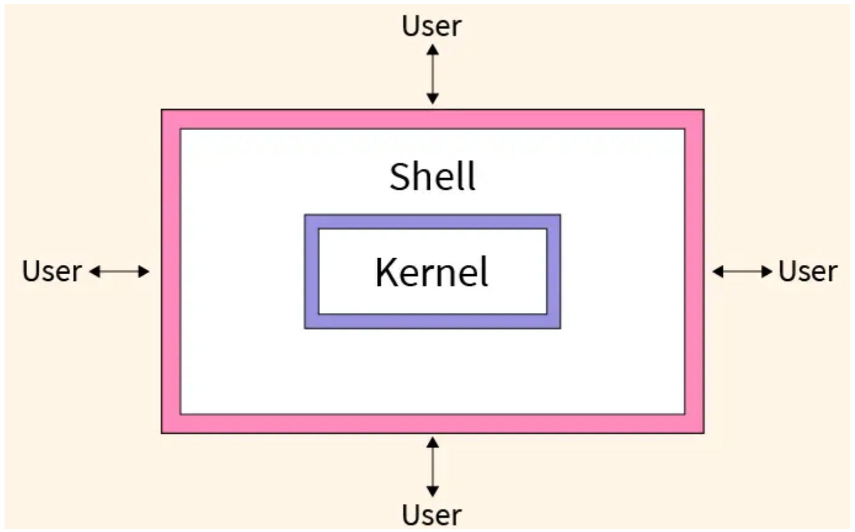
- The **kernel** is the heart of the operating system, responsible for direct communication with the hardware. It manages critical functions such as controlling the CPU, handling memory, and ensuring that multiple processes run efficiently without interfering with one another.
- Surrounding the kernel is the **shell**, which serves as the user interface for interacting with the system. This interface can be a **command-line shell**, such as Bash on Linux or PowerShell on Windows, or a **graphical user interface** like Windows Explorer or macOS Finder.
- While the kernel performs the core operations behind the scenes, the shell provides users with a convenient way to issue commands and control the system.

Shell

- A shell is a program that provides an interface between a user and an operating system (OS) kernel.

- An OS starts a shell for each user when the user logs in or opens a terminal or console window.

- Shell is a UNIX term for the interactive user interface with an operating system. In some systems, the shell is called a command line interpreter.



Shell Script

- A shell script is a program composed of a series of operating system commands that are executed by the command-line interpreter(CLI) or the shell in sequence.

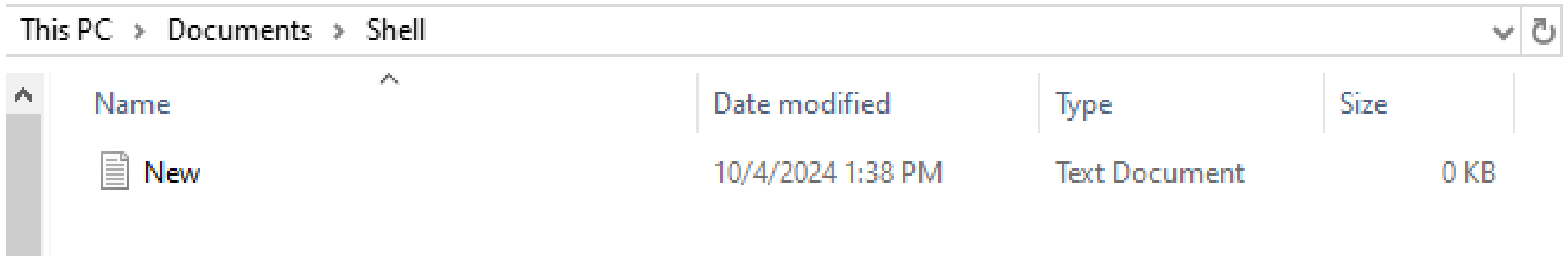
- It is called a shell script because the individual commands are combined to form a "script" that the shell follows and executes, very much like how an actor/actress follows the script written for him/her.
- There are several different shells available to Unix and Linux users. For the remainder, we will focus specifically on the **Bourne Again SHell (bash)**, which is an enhanced version of the **Bourne shell** and is one of the most common command line interfaces for Unix-based operating systems (including Linux).

- **What is a Bash Script?**
- Bash is a Unix command line interface responsible for interacting with a computer's operating system. Similarly to how movie scripts inform actors of what actions to take, a bash script tells the bash shell what to do. Thus, **a bash script is a useful way to group commands to create a program**.
- Any command you could execute directly from the command line can be put into a bash script, and you could expect it to do the same actions as it would from the command line.
- One of the main differences is a bash script is a plain text file that is stored with *filename.sh*

- **How to Write a Bash Script**
- **Task 1:** We are going to build a simple “Hello World” bash script example to get you started with bash scripting.

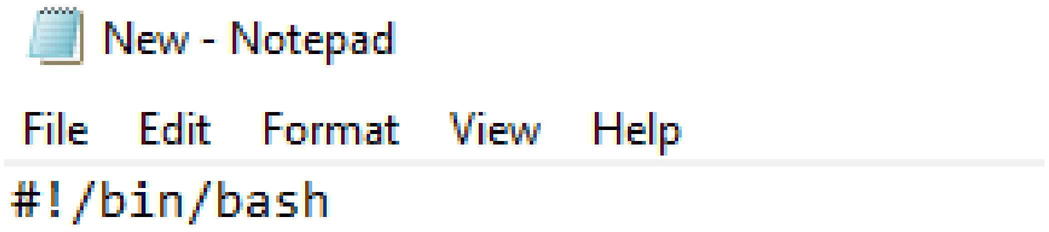
Step 1: Create a new plain text file

- The first step is to create a new plain text file. Here, I create a folder **shell** on **Documents** destination.


This PC › Documents › Shell

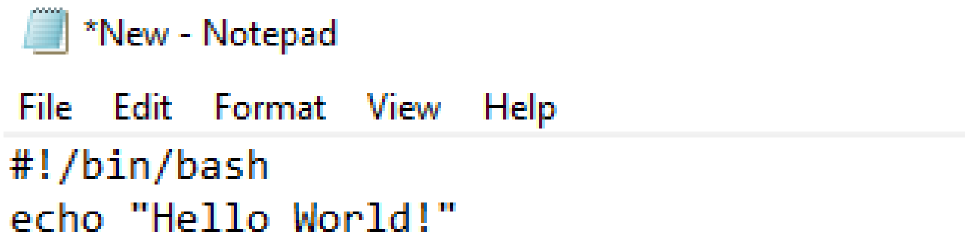
Name	Date modified	Type	Size
New	10/4/2024 1:38 PM	Text Document	0 KB

- **Step 2: Specifying the interpreter**
- On the first line of our script, we must specify which interpreter we would like to use to parse our script. In this scenario, it is Bash. Thus, we must put the *shebang* in the first line of our script.
- the shebang is an absolute path to the bash interpreter made up of a combination of bash “#” and bang “!” followed by the bash shell path. It is used to tell the Linux operating system which interpreter to use to parse the file. The #! characters at the beginning of the line indicate to the operating system that this is a shebang line, and the path to the interpreter (/bin/bash) follows.



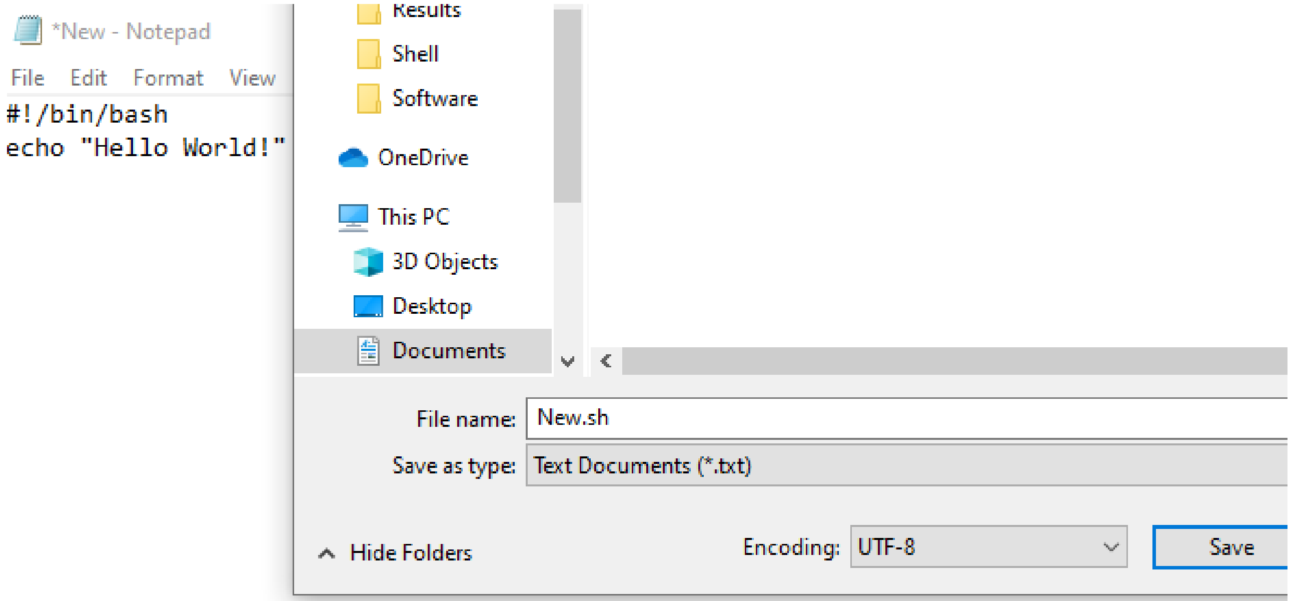
```
#!/bin/bash
```

- **Step 3: Implement commands**
- The purpose of our bash script is to print “Hello World!” To perform this task, move to a new line and use the echo command followed by the string we would like to print.



```
#!/bin/bash
echo "Hello World!"
```

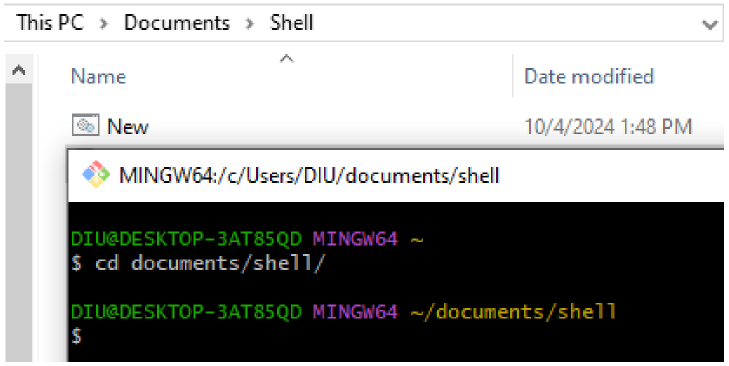
- Once complete, save the file using *.sh* extensions!



```
#!/bin/bash
echo "Hello World!"
```

File name: New.sh
Save as type: Text Documents (*.txt)
Encoding: UTF-8

- **Note**: You can download Git Bash from Git downloads. Simply select the version of Git that corresponds with your operating system.
- Now, open Git Bash and select correct directory.



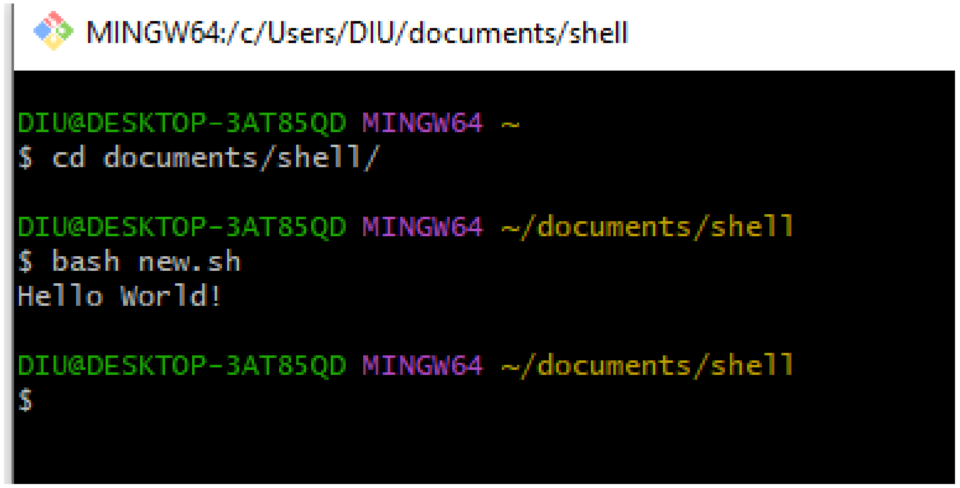
This PC › Documents › Shell

Name	Date modified
New	10/4/2024 1:48 PM

MINGW64:/c/Users/DIU/documents/shell

```
DIU@DESKTOP-3AT85QD MINGW64 ~
$ cd documents/shell/

DIU@DESKTOP-3AT85QD MINGW64 ~/documents/shell
$
```

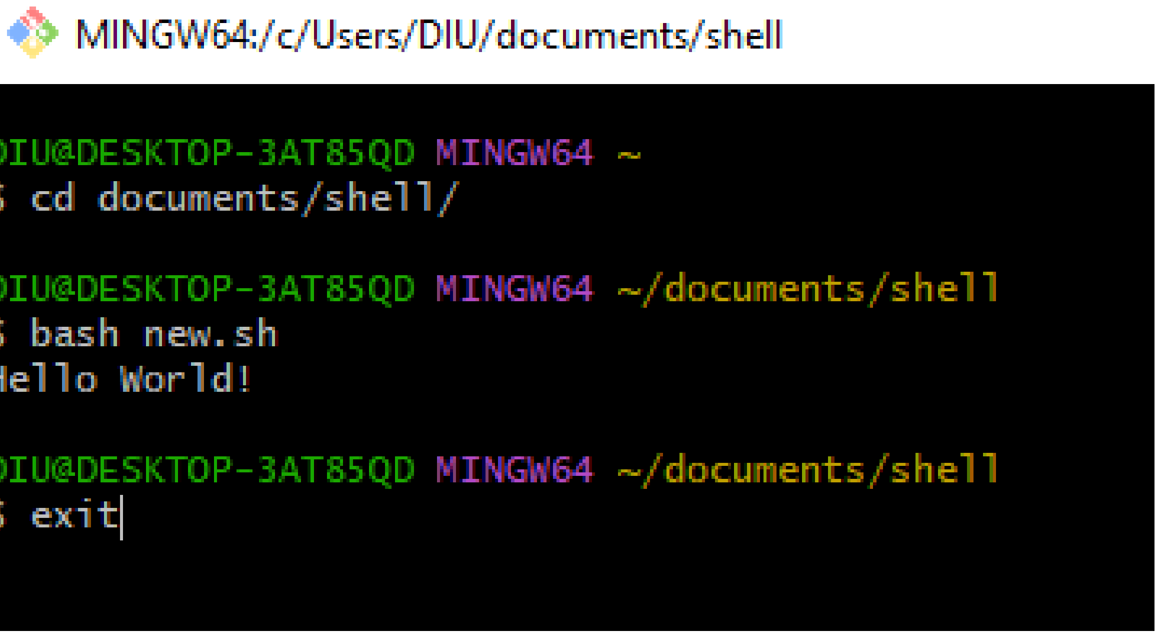
- Next, run the following command using bash word and filename:



MINGW64:/c/Users/DIU/documents/shell

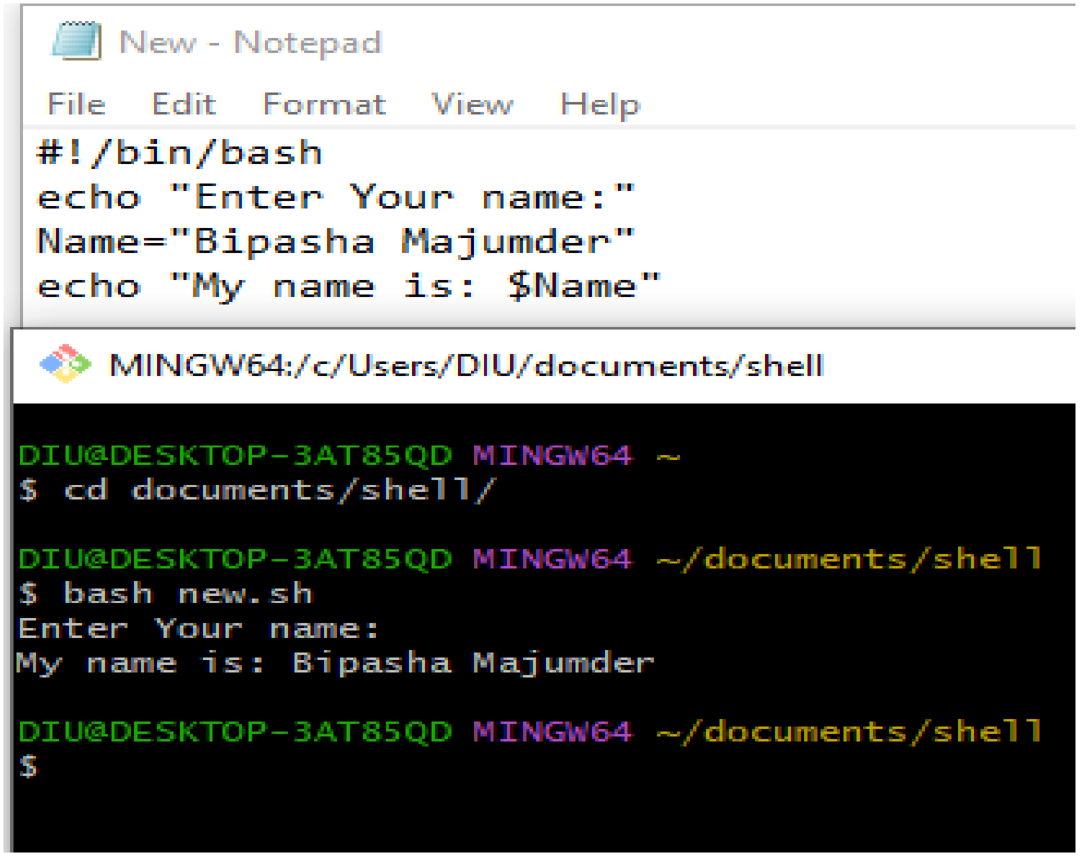
```
DIU@DESKTOP-3AT85QD MINGW64 ~
$ cd documents/shell/

DIU@DESKTOP-3AT85QD MINGW64 ~/documents/shell
$ bash new.sh
Hello World!

DIU@DESKTOP-3AT85QD MINGW64 ~/documents/shell
$
```

- **Note:** To exit from bash, just use exit command.


MINGW64:/c/Users/DIU/documents/shell

```
DIU@DESKTOP-3AT85QD MINGW64 ~
$ cd documents/shell/

DIU@DESKTOP-3AT85QD MINGW64 ~/documents/shell
$ bash new.sh
Hello World!

DIU@DESKTOP-3AT85QD MINGW64 ~/documents/shell
$ exit
```

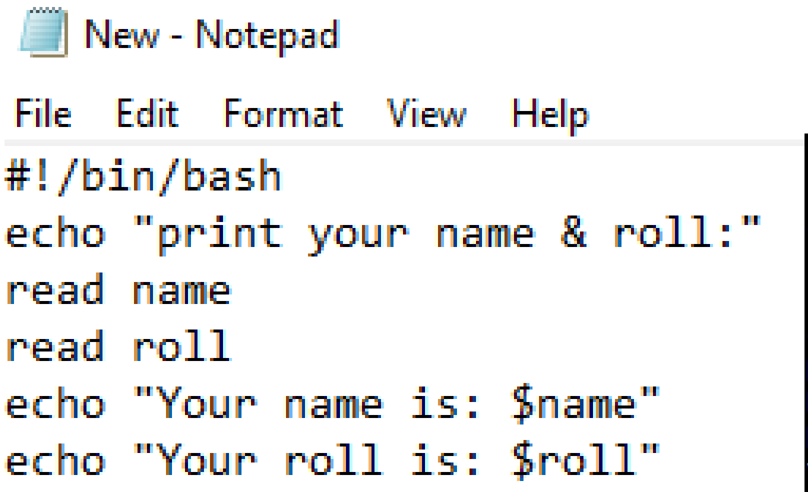
- **Task 2:** how to create and use a variable in a shell script
- **Note:** variables are named values that can be used to store data and manipulate it within scripts or commands. Variables can be assigned using the = operator, and their values can be accessed using the $ notation followed by the name of the variable.


New - Notepad
File Edit Format View Help
MINGW64:/c/Users/DIU/documents/shell

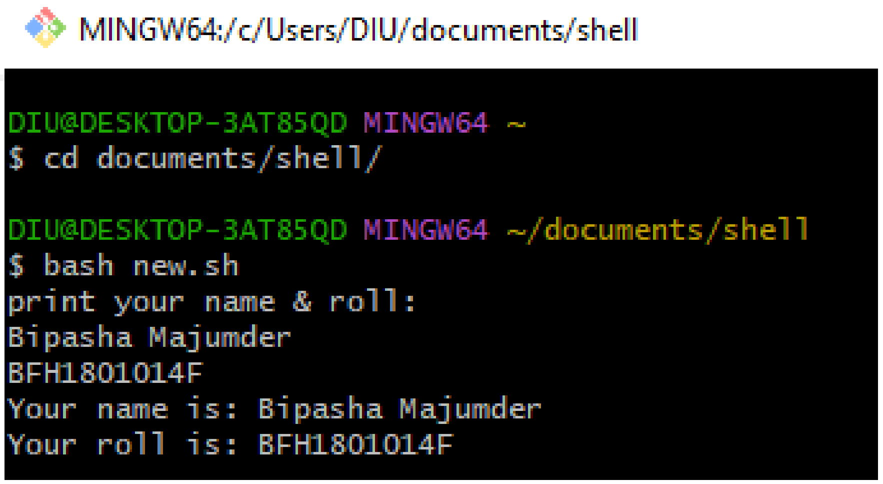
```
#!/bin/bash
echo "Enter Your name:"
Name="Bipasha Majumder"
echo "My name is: $Name"
```

```
DIU@DESKTOP-3AT85QD MINGW64 ~
$ cd documents/shell/

DIU@DESKTOP-3AT85QD MINGW64 ~/documents/shell
$ bash new.sh
Enter Your name:
My name is: Bipasha Majumder

DIU@DESKTOP-3AT85QD MINGW64 ~/documents/shell
$
```

- **Task 3:** How do I read user input into a variable in Bash
- **Note:** To read user input in a bash script, use the **read** command. It allows you to prompt the user for input and store their response in a variable. Here's a basic example:

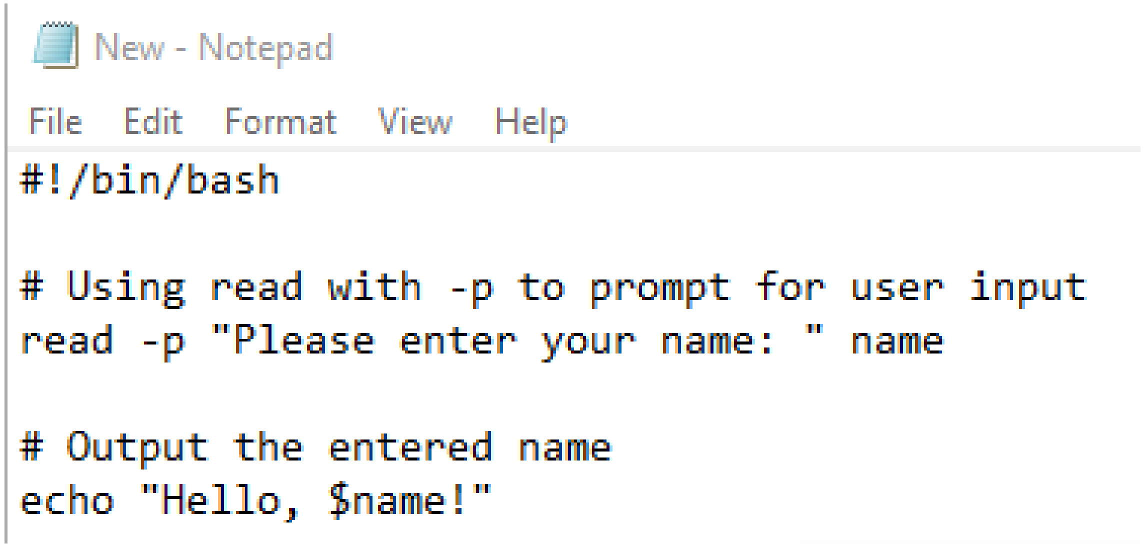


```
#!/bin/bash
echo "print your name & roll:"
read name
read roll
echo "Your name is: $name"
echo "Your roll is: $roll"
```

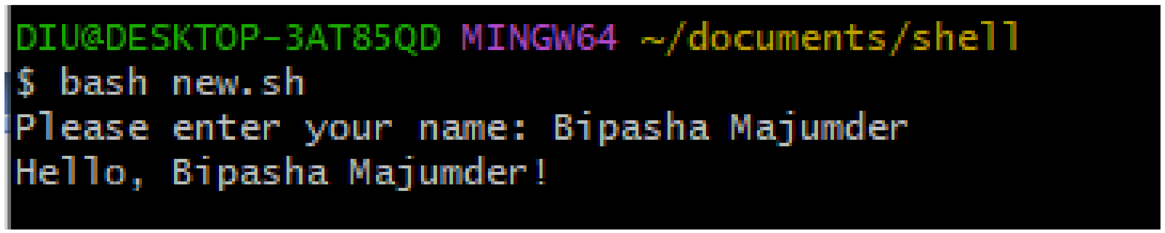


```
DIU@DESKTOP-3AT85QD MINGW64 ~
$ cd documents/shell/

DIU@DESKTOP-3AT85QD MINGW64 ~/documents/shell
$ bash new.sh
print your name & roll:
Bipasha Majumder
BFH1801014F
Your name is: Bipasha Majumder
Your roll is: BFH1801014F
```

- **Task 4:** How do I put print statement and input on same line?
- **Note:** *read -p* "Please enter your name: " name: This line prompts the user to enter their name and stores it in the variable name on same line.

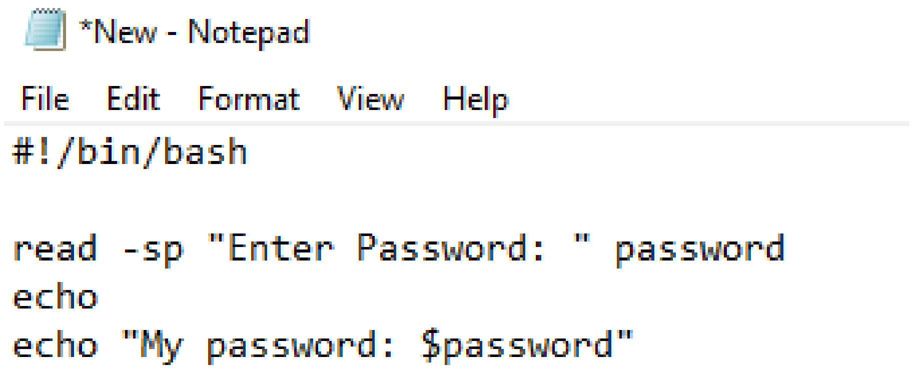


```
#!/bin/bash

# Using read with -p to prompt for user input
read -p "Please enter your name: " name

# Output the entered name
echo "Hello, $name!"
```

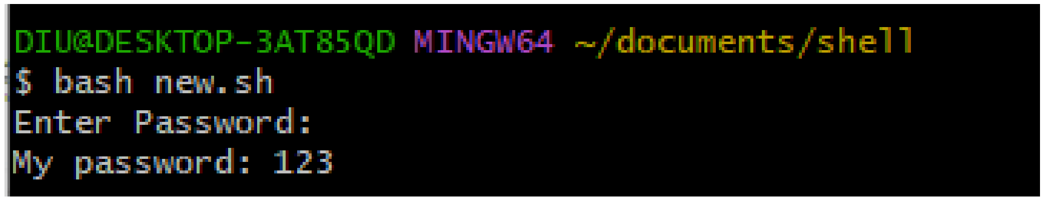


```
DIU@DESKTOP-3AT85QD MINGW64 ~/documents/shell
$ bash new.sh
Please enter your name: Bipasha Majumder
Hello, Bipasha Majumder!
```

- **Task 5:** How do I take silent Input using read Command?
- **Note:** In the context of the *read -sp*
- -s: This option makes the input silent, meaning that the characters typed by the user are not displayed on the screen. This is useful for sensitive information like passwords.
- -p: As previously mentioned, this option allows you to specify a prompt message.
- Note: *echo* use for new line.



```
#!/bin/bash

read -sp "Enter Password: " password
echo
echo "My password: $password"
```



```
DIU@DESKTOP-3AT85QD MINGW64 ~/documents/shell
$ bash new.sh
Enter Password:
My password: 123
```