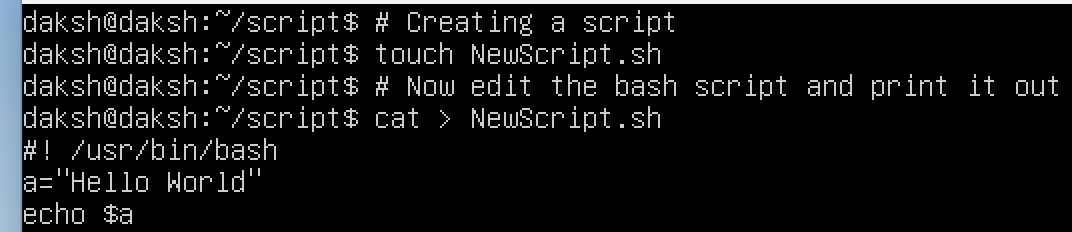
**Creating a bash script: -**

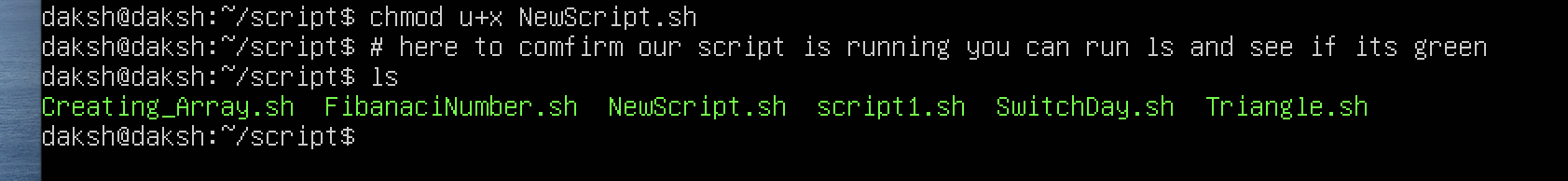
In this unit we will learn how to create a bash shell script. Basically we write a script for the work that we do again and again. So to save ourself from the hassle we create a script for that work which we have repeat again and again in our daily life. Script help us to complete these tasks easily.

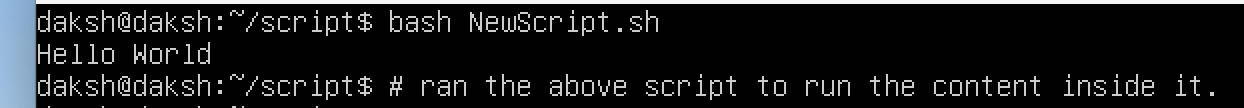
**Steps to create a shell script: -**

1. **We go to our terminal and create a file name like `Script\_name.sh`**
2. **Now your script file is ready and the extension `.sh` is for the script.**
3. **You can now open this script file with your editors here we are using the `vi` editor to edit all our bash scripts.**
4. **Once we are done with all this we have to open the bash script and write #! /usr/bin/bash/ here `#!` means shebang in bash and the path here is where your bash is located if your script is not working then this means your bash is on some other path to check the path we can simply use the command `which bash` and that will return the path of your bash.**
5. **After this open your file and start editing it once you are done then just save it.**
6. **After the creation and the editing of file you have to give your `user` the permission to edit and run the bash script which we will give by running the command `chmod u+x`.**
7. **Once you are done with all the above steps you can run your bash script by running the command `bash File\_name.sh`.**

**Screenshots: -**

****

****

****

Note: - If you have the root access then there’s no need for the chmod command to run the bash scripts

**Arithmetic Operators:-**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Use** | **Example** |
| **+** | **Addition** | **It adds two operands** | **Results= a+b** |
| **-** | **Subtraction** | **It subtracts second operand from the first one** | **Result= a-b** |
| **\*** | **Multiplication** | **Multiply two operands** | **Result= a\*b** |
| **/** | **Division** | **Return the quotient after diving the number** | **Result= a/b** |
| **%** | **Modulo** | **Returns the remainder of the number once it is divided** | **Result= a%b** |
| **+=** | **Increment by constant** | **Returns the result with the addition of the constant that is given** | **Result=**  **x=13**  **x+=3; Now x=16** |
| **-=** | **Decrement by constant** | **Returns the result with the subtraction by the given constant** | **Result=**  **x=13**  **x-=3; Now x=10** |
| **/=** | **Divide by constant** | **Returns the result by diving the number with the given constant** | **Result=**  **x=15**  **x/=5; Now x=3** |
| **%=** | **Remainder by diving the constant** | **Return the result with given constant by giving the remainder by diving the num with constant** | **Result=**  **x=31**  **x%=3; Now x=1** |
| **\*\*** | **Exponential** | **Return the number by the power of constant** | **Result=**  **3\*\*2=9** |
|  |  |  |  |

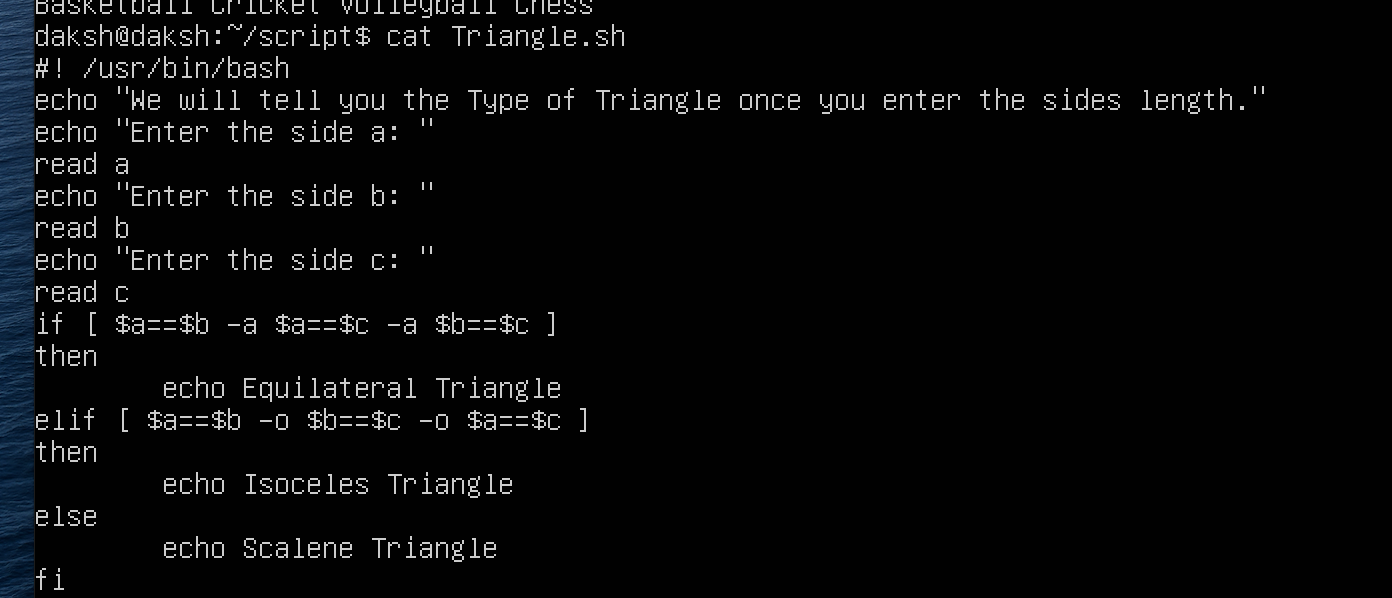
**Screenshot: -**

**Comparison Operators: -**

|  |  |  |
| --- | --- | --- |
| **OPERATIONS** | **SYNTAX** | **EXPLANATION** |
| Equality | Num1 -eq Num2 | Is num1 equal to num2 |
| Less than equal to | Num1 -le Num2 | Is num1 less than equal to num2 |
| Less than | Num1 -lt Num2 | Is num1 less than num2 |
| Greater than equal to | Num1 -ge Num2 | Is num1 greater than equal to num2 |
| Greater than | Num1 -gt Num2 | Is num1 greater than num2 |
| Not equal to | Num1 -ne Num2 | Is num1 not equal to num2 |

**Screenshot: -**

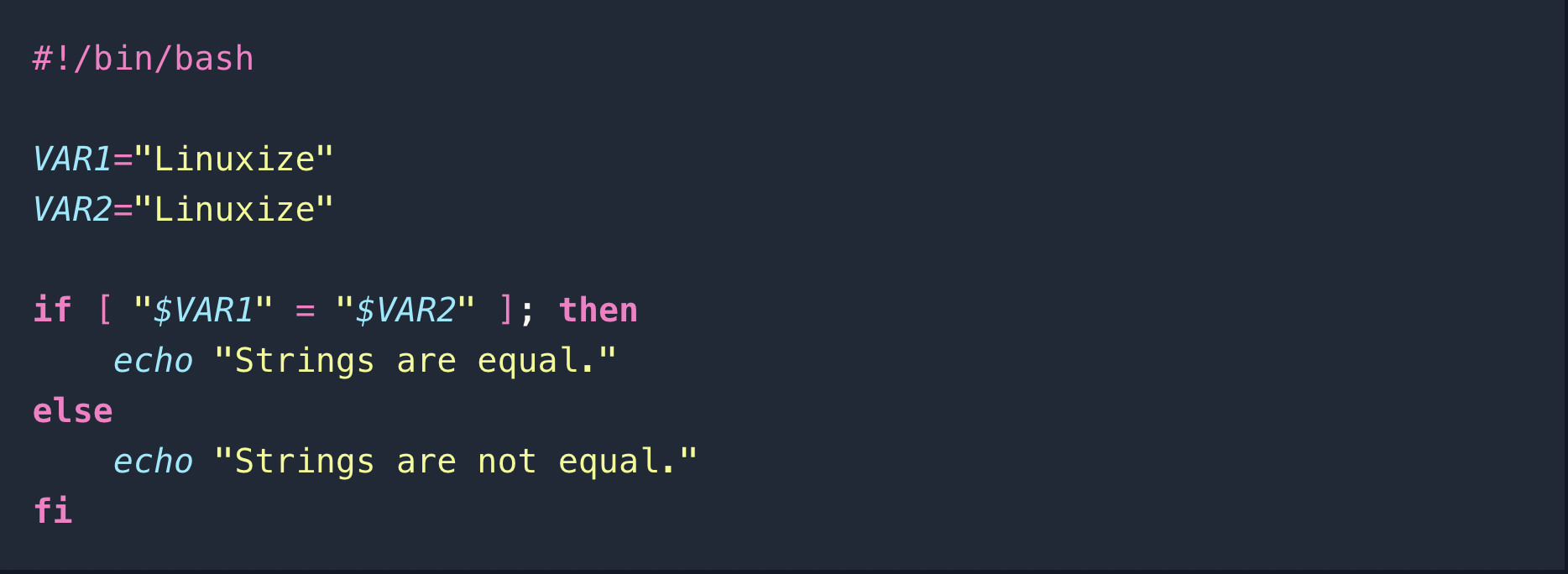
**Using all the comparison operators**

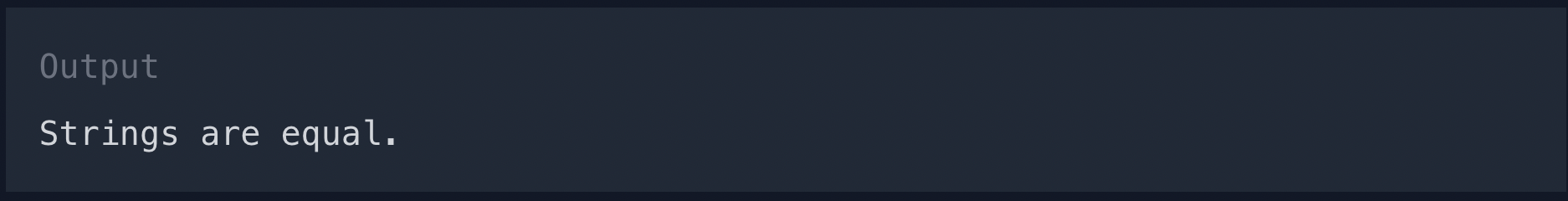
****

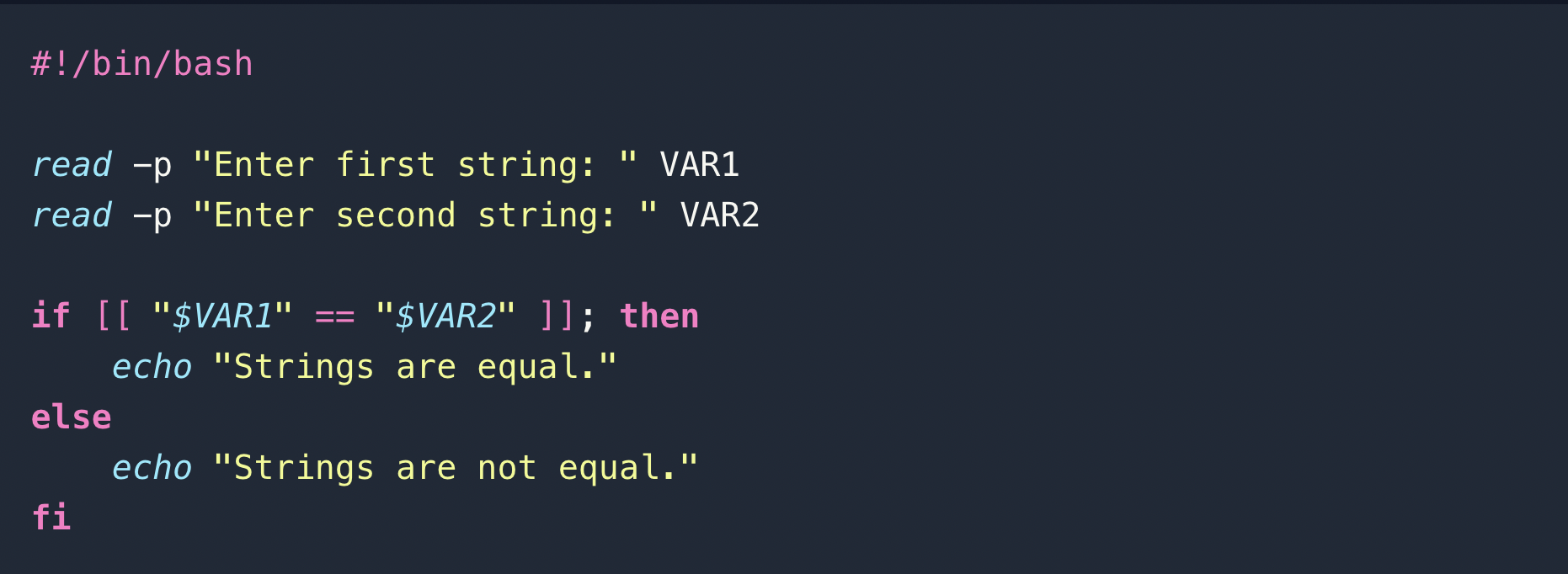
**Equality(-eq)**: -

Here we will learn how to use the comparison operator `-eq` in bash shell scripting.

**Code for comparison operator**: -



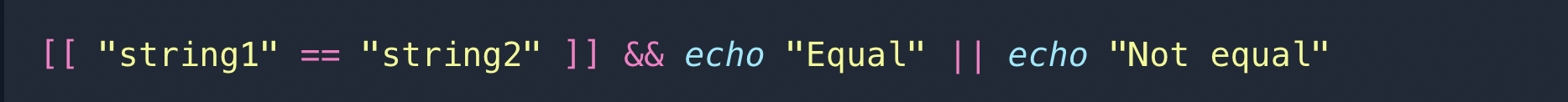


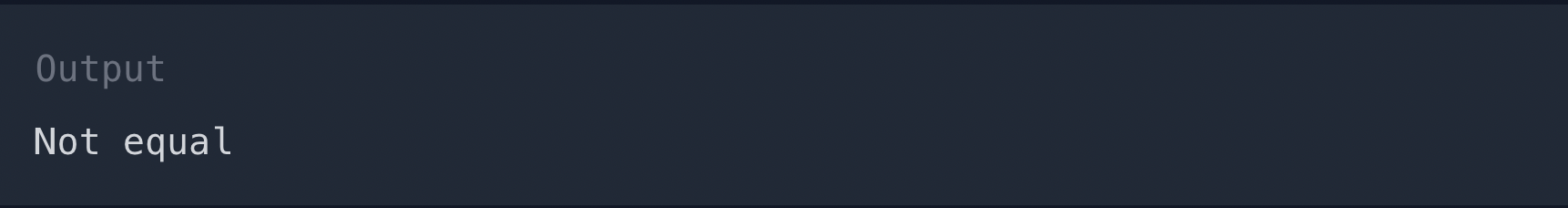


A picture containing shape

Description automatically generated

You can also use the logical and && and or || to compare strings:





**Printing Fibonacci Series using `for` loop**: -

So here we will learn to print out the fibonacci series using the for loop that we learned in the previous classes.

Code: -

```bash

#! /usr/bin/bash

echo “Enter the number of terms: “

read num

echo “Now we will print the terms below: -“

for(i=0;i<n;i++)

do

echo -n “$a “ //Here we kept a space so the terms which will be printed later won’t get messy and will have space in between them

a=0

b=1

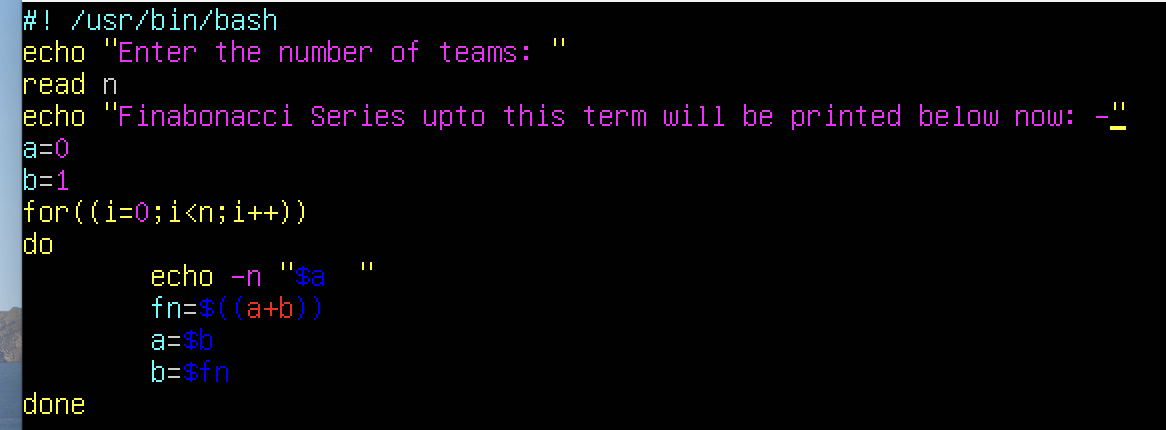
fn=$((a+b))

b=$fn

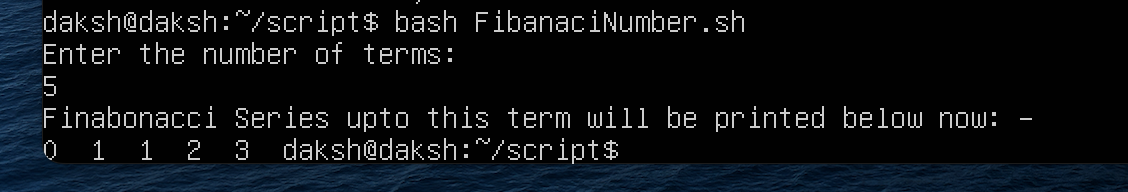
done

```

**Screenshot: -**

****

**Output: -**

****

**Switch statement in bash scripting**: -

So here we will learn to write a basic program in bash script using the `switch statement`. The code and the syntax will be mentioned below.

**Syntax**: -

```bash

#!/bin/bash

read -p "Enter a number: " num

case $num in // This will compare the number which you entered

1)

echo "You entered 1."

;;

2)

echo "You entered 2."

;;

3)

echo "You entered 3."

;;

\*)

echo "You entered a different number."

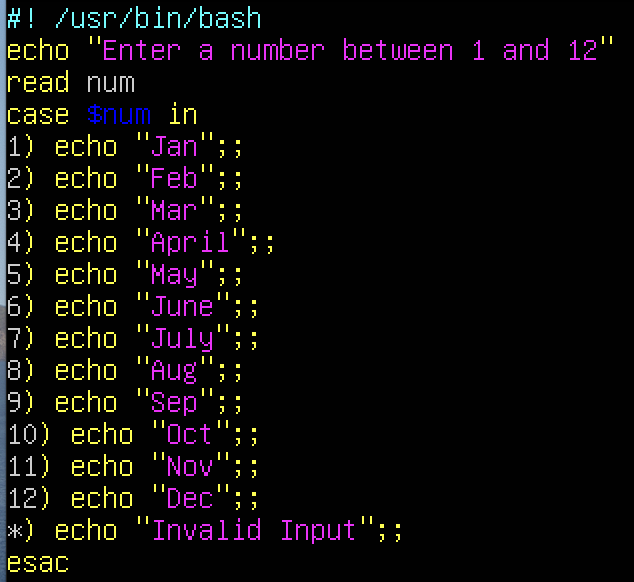
;;

esac

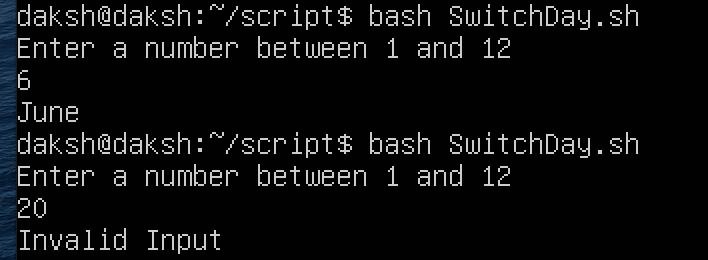
```

Here esac means end of statement which is similar to fi which we used in the if statement.

**Screenshot: -**

****

**Output: -**

****

**Creating an Array**: -

Here we will learn how to create an array in bash shell scripting. The syntax will be mentioned below.

**Syntax**: -

Declare -a array\_name=

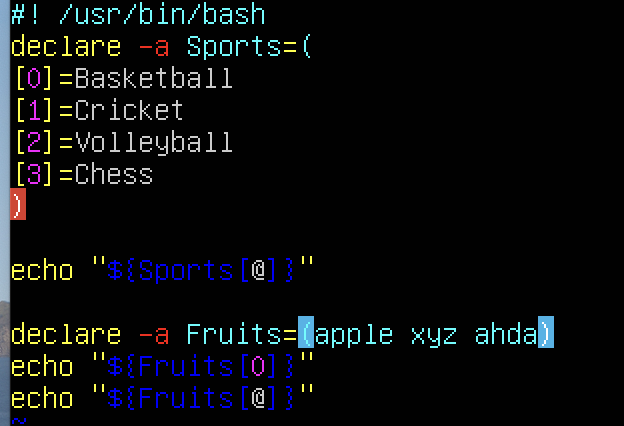
(element1 element2 element3 ...)

Printing the elements: -

echo ${Array\_name[@]} //this will print all the elements inside the array

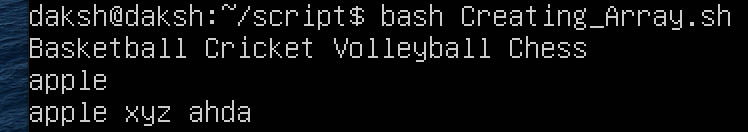
echo ${Array\_name[Index\_Value]} // this will print the exact Index value element from the array

**Screenshot: -**

****

* Here we can write the index values of each element when creating or just add elements like above both the methods are mentioned.

**Output: -**

****

**System Call: -**

A system call is a function that allows a process to communicate with the Linux kernel. It’s just a programmatic way for a computer program to order a facility from the operating system’s kernel. System calls expose the operating system’s resources to user programs through an API (Application Programming Interface). System calls can only access the kernel framework. System calls are needed for all the services that need resources.

**Screenshot: -**

**Program to read the file**

Text

Description automatically generated

**Compiling the program to read the Sample.txt: -**

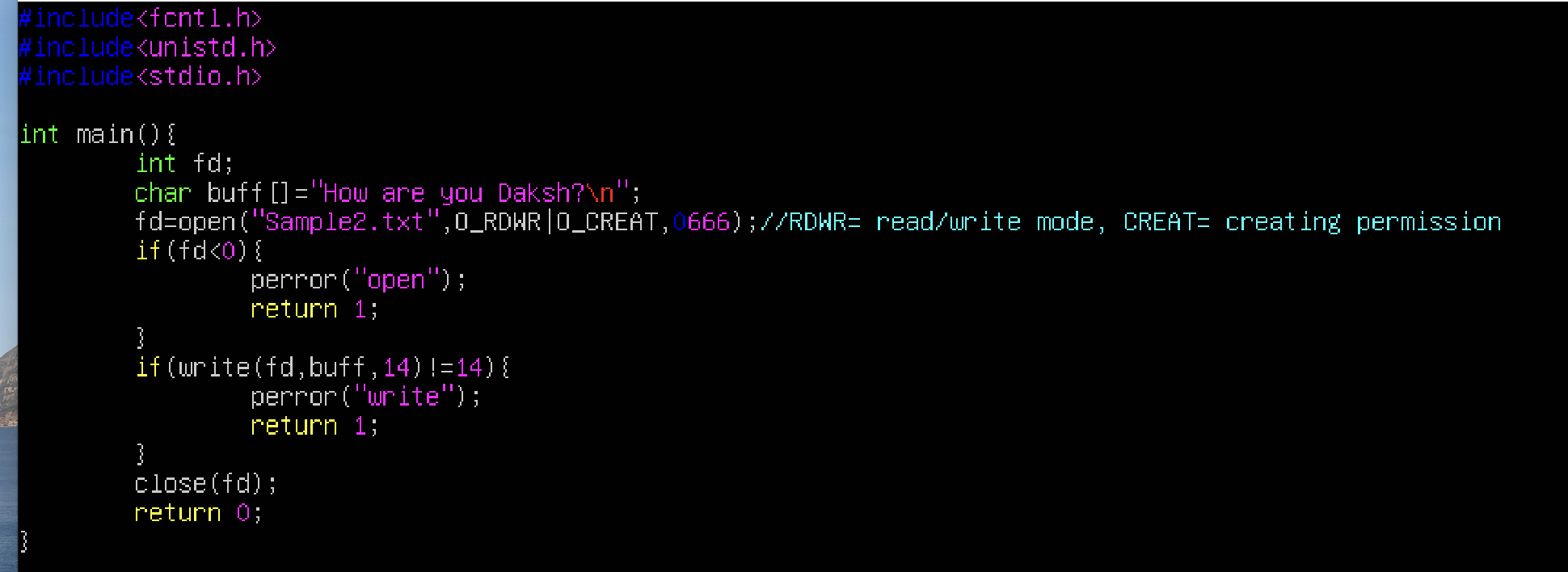
Text

Description automatically generated

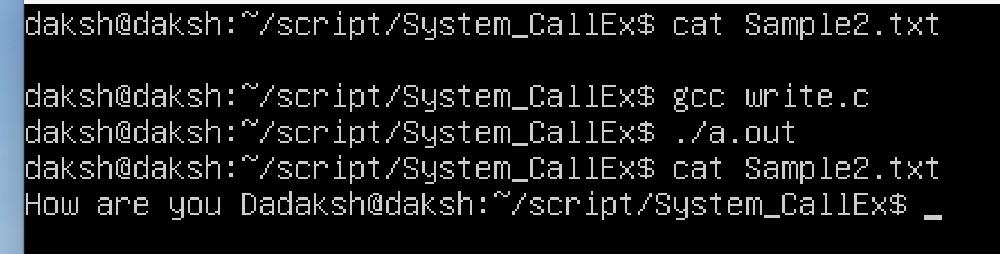
**Program to write in the file**

Now we will write a program to write the Sample file.

**Screenshot: -**

****

**Output**

****