**Shell Scripting**

A **shell script** is a computer program designed to be run by the Unix shell, a command-line interpreter. Typical operations performed by shell scripts include file manipulation and program execution, and this is precisely why it is relevant to learn the basics as a data scientist.

# Why shell scripting?

Shell scripts enable you to automate a collection of command-line operations in a single script, executing them line by line. Your project is ready for these automation steps, and it is handy to learn shell sooner than later.

A few examples where I used it in my work as a data scientist: virtual environment management, instantiation of PySpark tables, execution of Python linters, updating and building static documentation websites, execution of production pipelines with specific arguments, etc.

# Why do we use shell scripting? / Why do we need shell scripts

Shell scripting is meant to be simple and efficient. It uses the same syntax in the script as it would on the shell command line, removing any interpretation issues. Writing code for a shell script is also faster and requires less of learning curve than other programming languages.

There are many reasons to write shell scripts –

* To avoid repetitive work and automation
* System admins use shell scripting for routine backups
* System monitoring
* Adding new functionality to the shell etc.

# What can you do with shell scripting?

Shell scripting can be used to **automate several daily tasks, repetitive tasks**, etc. If we want to execute the same command multiple times then we can use shell script built-in functions like for loop, while loop etc.

# How does a shell work?

The shell is your interface to the operating system. It acts as a command interpreter; it takes each command and passes it to the operating system. It then displays the results of this operation on your screen. There are several shells in widespread use.

# Is shell a programming language?

A Unix shell is both a command interpreter and a programming language. As a command interpreter, the shell provides the user interface to the rich set of GNU utilities. The programming language features allow these utilities to be combined. Files containing commands can be created, and become commands themselves.

# What is the difference between script and shell?

In computer programming, a script is defined as a sequence of instructions that is executed by another program. A shell is a command-line interpreter of Linux which provides an interface between the user and the kernel system and executes a sequence of instructions called commands

# Advantages of shell scripts

* The command and syntax are exactly the same as those directly entered in command line, so programmer do not need to switch to entirely different syntax
* Writing shell scripts are much quicker
* Quick start
* Interactive debugging etc.

# Disadvantages of shell scripts

* Prone to costly errors, a single mistake can change the command which might be harmful
* Slow execution speed
* Design flaws within the language syntax or implementation
* Not well suited for large and complex task
* Provide minimal data structure unlike other scripting languages. Etc

***Shell Scripting***

Shell Scripting is an open-source operating system.

Our Shell Scripting includes all topics of Scripting executing scripting, loops, scripting parameters, shift through parameters, sourcing, getopts, case, eval, let etc. There is also given Shell Scripting interview questions to help you better understand the Shell Scripting operating system.

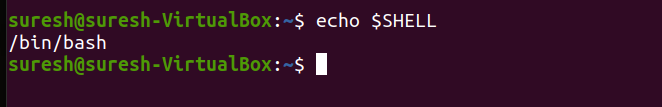
**Shell Scripting**

# How to determine Shell

* You can get the name of your shell prompt, with following command:

**Syntax:**

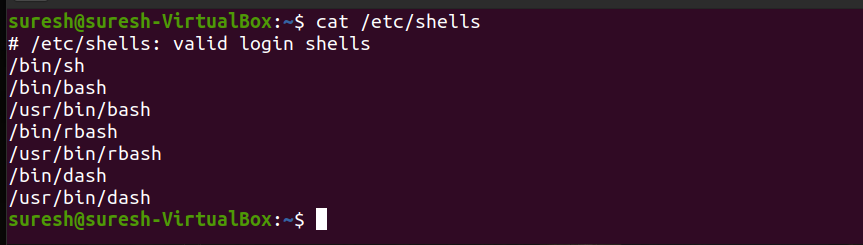
1. **echo $SHELL**



Look at the above snapshot, with the help of above command we got the name of our shell which is **'bash'.**

The $ sign stands for a shell variable; echo will return the text whatever you typed in.

**Types of shells**

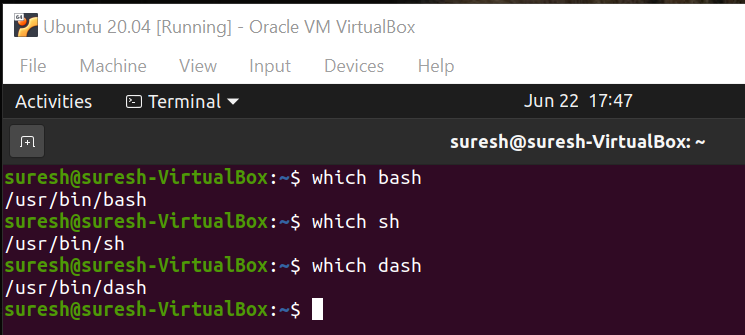
****

* etc – this can be treated as nerve centre (central location of files) of your Linux/Unix.

These are the different kind of shells which your system can support.

* Sh-stands for Bourne shell. which is the original shell still used on unix system or unix-like environment.
* Bash-stands for Bourne-Again Shell. which is the improved version of sh(shell) . (it can used most of the UNIX OS or Linux based OS and including mac OS and nowadays we are used in windows 10)
* RBash-stands for **Restricted Shell is a Linux Shell** that restrict some of the features of bash shell, and is very clear from the name. The restriction is well implemented for the command as well as script running in restricted shell. It provides an additional layer for security to bash shell in Linux.
* Dash shell is a simplistic modern POSIX-compliant version of the Bourne shell.

**How to located shells?**

****

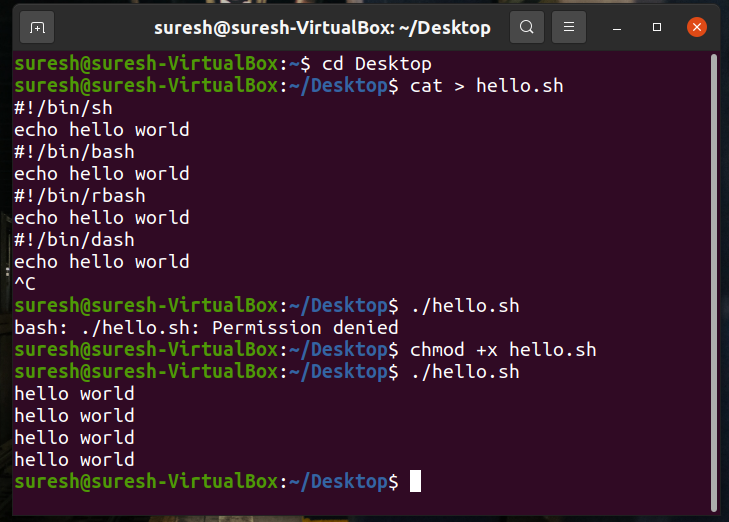
# Shell Scripting She-bang

The sign #**!** is called she-bang and is written at top of the script. It passes instruction to program **/bin/sh.**

To run your script in a certain shell (shell should be supported by your system), start your script with #! followed by the shell name.

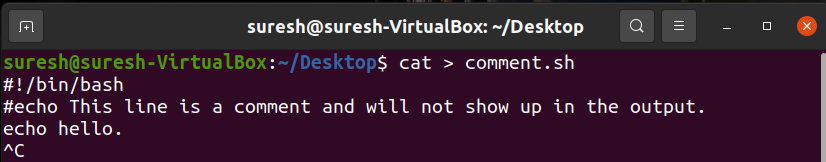
**Example:**

1. #!/bin/sh
2. echo Hello World
3. #!/bin/bash
4. echo Hello World
5. #!/bin/rbash
6. echo Hello World
7. #!/bin/dash
8. echo Hello World

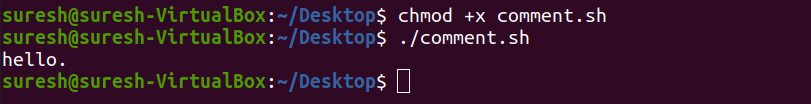


# Shell Scripting Comments

Any line starting with a hash (#) becomes comment. Comment means, that line will not take part in script execution. It will not show up in the output.



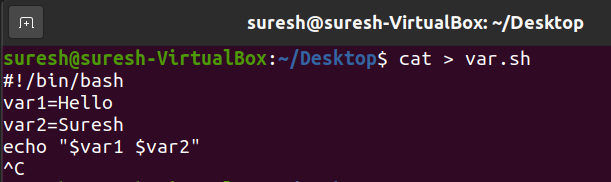
Look at the above snapshot, lines after the # are commented.



Look at the above snapshot, commented lines are not displayed in the output.

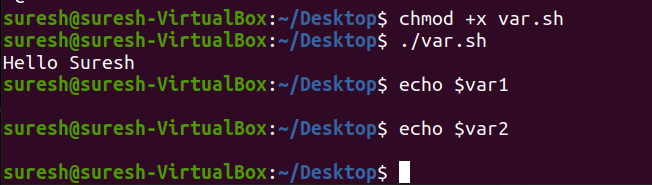
# Shell Scripting Variables

Scripts can contain variables inside the script.



Look at the above snapshot, two variables are assigned to the script **$var1** and **$var2.**

As scripts run in their own shell, hence variables do not survive the end of the script.



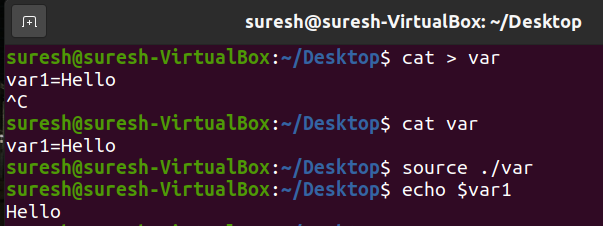
Look at the above snapshot, **var1** and **var2** do not run outside the script.

* **Shell Scripting Sourcing a file**

A file is sourced in two ways. One is either writting as **source** <**fileName**> or other is writing as . ./<**filename>** in the command line. When a file is sourced, the code lines are executed as if they were printed on the command line.

The difference between sourcing and executing a script is that, while executing a script it runs in a new shell whereas while sourcing a script, file will be read and executed in the same shell.

In sourcing, script content is displayed in the same shell but while executing script run in a different shell.



Look at the above snapshot, we have sourced the file **var** with one of the method.

# Troubleshooting a shell script

There is one more way other than script execution to run a script in a different shell. Type bash with the name of the script as parameter.

**Syntax:**

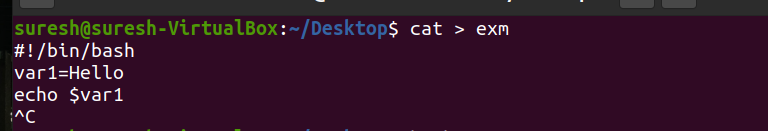
1. bash **<fileName>**

**Example:**

bash exm

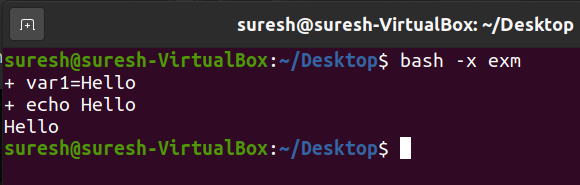


Look at the above snapshot, it displays the **exm** script content with bash command.



Look at the above snapshot, this is the exm script we have written.

By expanding **bash** command with -**x**, shell allows us to see commands that the shell is executing.



Look at the above snapshot, with command **bash -x**, we can see the shell expansion.

**Executing Script**

# Steps to write and execute a script

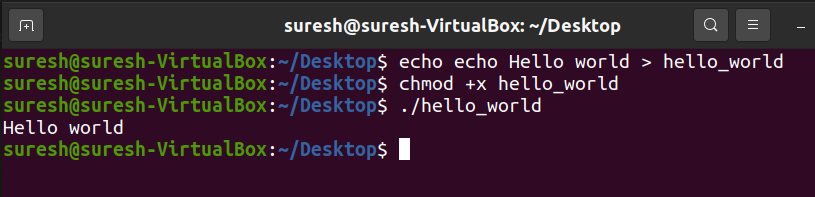
* Open the terminal. Go to the directory where you want to create your script.
* Create a file with **.sh** extension.
* Write the script in the file using an editor.
* Make the script executable with command **chmod +x** <**fileName**>.
* Run the script using ./<**fileName**>.

**Note:** In the last step you have to mention the path of the script if your script is in other directory.

## Hello World script

Here we'll write a simple programme for Hello World.

First of all, create a simple script in any editor or with echo. Then we'll make it executable with **chmod +x** command. To find the script you have to type the script path for the shell.



Look at the above snapshot, script **echo Hello World** is created with echo command as **hello\_world.** Now command **chmod +x hello\_world** is passed to make it executable. We have given the command **./hello\_world** to mention the hello\_world path. And output is displayed.

**Shell Parameters**

* **Shell Script Parameters**

A bash shell script have parameters. These parameters start from **$1** to **$9.**

When we pass arguments into the command line interface, a positional parameter is assigned to these arguments through the shell.

The first argument is assigned as $1, second argument is assigned as $2 and so on...

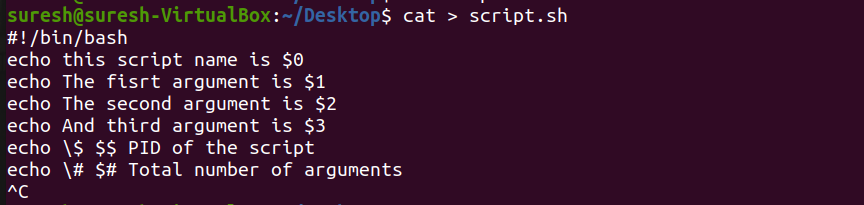
If there are more than 9 arguments, then **tenth** or onwards arguments can't be assigned as $10 or $11.

You have to either process or save the $1 parameter, then with the help of **shift** command drop parameter 1 and move all other arguments down by one. It will make $10 as $9, $9 as $8 and so on.

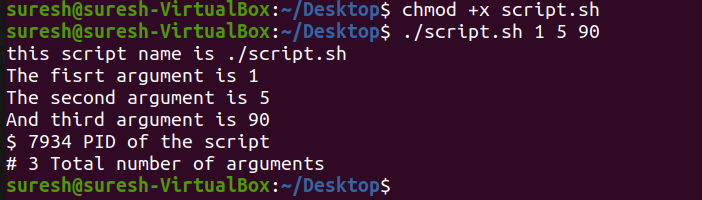
**Shell Parameters**

|  |  |
| --- | --- |
| **Parameters** | **Function** |
| $1-$9 | Represent positional parameters for arguments one to nine |
| ${10}-${n} | Represent positional parameters for arguments after nine |
| $0 | Represent name of the script |
| $∗ | Represent all the arguments as a single string |
| $@ | Same as $∗, but differ when enclosed in (") |
| $# | Represent total number of arguments |
| $$ | PID of the script |
| $? | Represent last return code |

**Example:**



Look at the above snapshot, this is the script we have written to show the different parameters.



Look at the above snapshot, we have passed arguments **1, 5, 90**. All the parameters show their value when script is run.

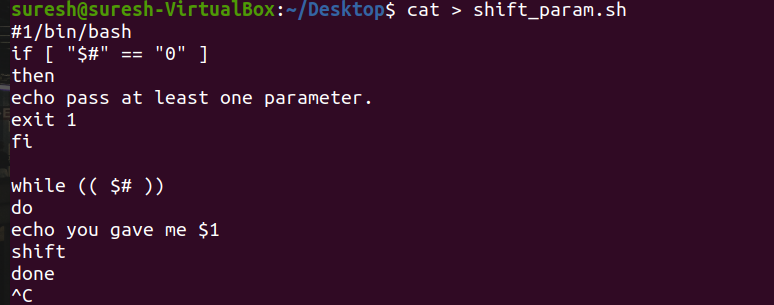
* **Shell Scripting Shift Through Parameters**

Shift command is a built-in command. Command takes number as argument. Arguments shift down by this number.

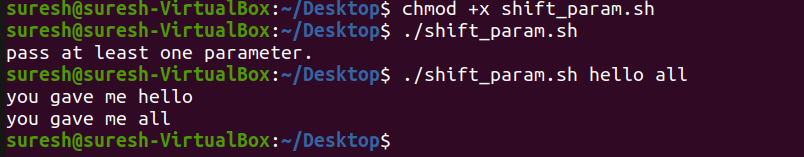
For example, if number is 5, then $5 become $1, $6 become $2 and so on.

**Example:**

The shift command is mostly used when arguments are unknown. Arguments are processed in a while loop with a condition of **(( $# ))**. this condition holds true as long as arguments are not zero. Number of arguments are reduced each time as the shift command executes.



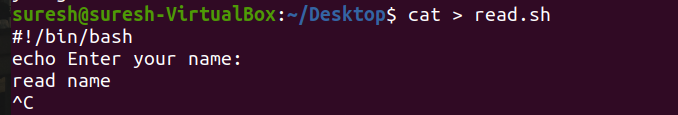
Look at the above snapshot, this is our script.



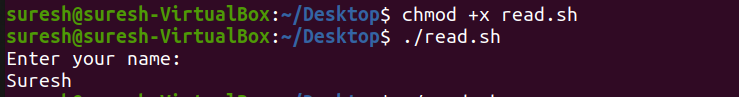
Look at the above snapshot, this is the output of the above script.

**read command**

The read command allows a user to provide the runtime input.



Look at the above snapshot, this is our script using read command.



Look at the above snapshot, a user can enter the name in the shell.

**Shell Sourcing**

* **Shell Scripting Sourcing a config file**

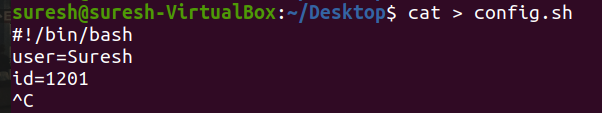
Many programs use external configuration files. Use of external configuration files prevents a user from making changes to a script. Config file is added with the help of **source** command.

If a script is shared in many users and every user need a different configuration file, then instead of changing the script each time simply include the config files.

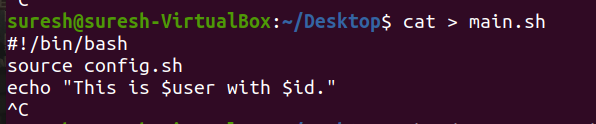
**Example:**

We have two files, one is parent file **(main.sh)** and other is configuration file **(config.sh)**. We have to source this configuration file into our parent file.

**Script for config.sh**



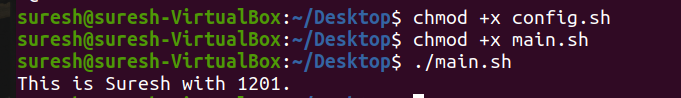
**Script for main.sh**



Look at the above snapshot, we've included config.sh file with source command.

**Note:** We can also use **( . config.sh )** command instead of **( source config.sh )** .

Now on running main.sh file, config.sh file is included.



Look at the above snapshot, on running main.sh file, content of config.sh file is imported via source command.

**Shell Loops**

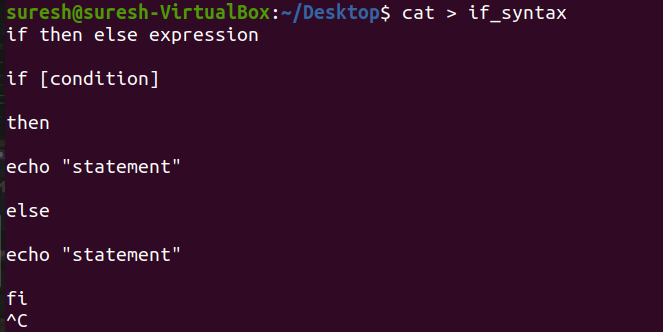
# Shell Scripting if then else

The if then else condition loop states that **if** condition meets, output goes to if part otherwise it goes to **else** part.

The word **fi** represents if loop termination .

**Syntax:**

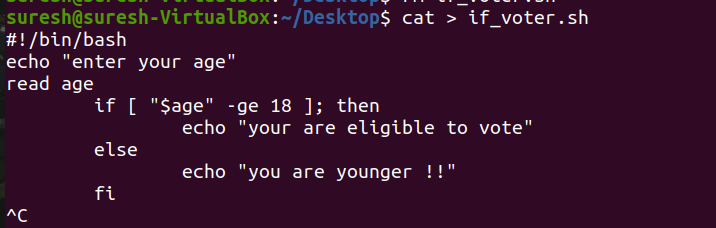
Syntax of if then else is shown in the snapshot below,



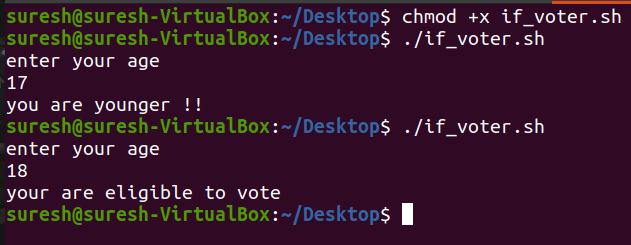
**Example if then else:**

We have shown the example of voting. If user's age will be greater than 18 then he or she will be eligible to vote, otherwise not.

1. if condition:
2. if [ "$age" -ge 18 ];



Look at the above snapshot, we have shown the script of file voter.



Look at the above snapshot, with age **17** it displays the message **"you are younger !!"** and with age **18** it displays the message **"you are eligible to vote".**

# Shell Scripting if then elif

A new if can be nested inside an elif.

**Syntax:**

Syntax of if then elif is shown in the snapshot below,

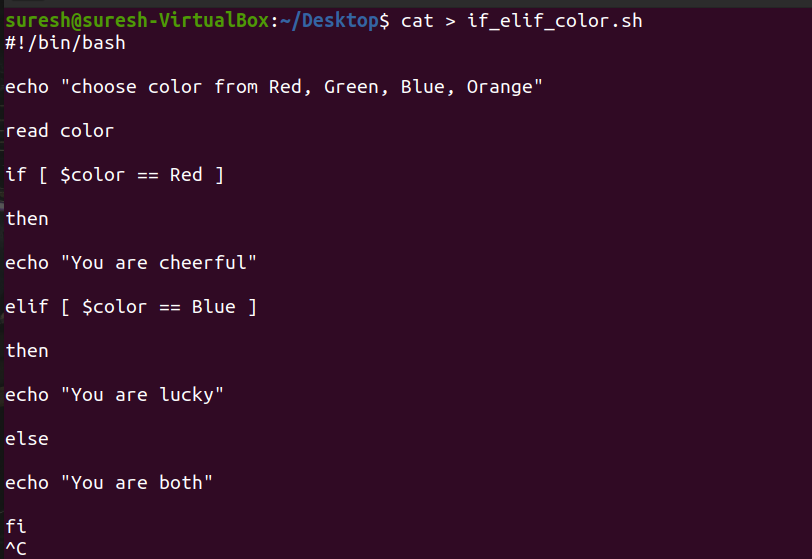


## Example if then elif:

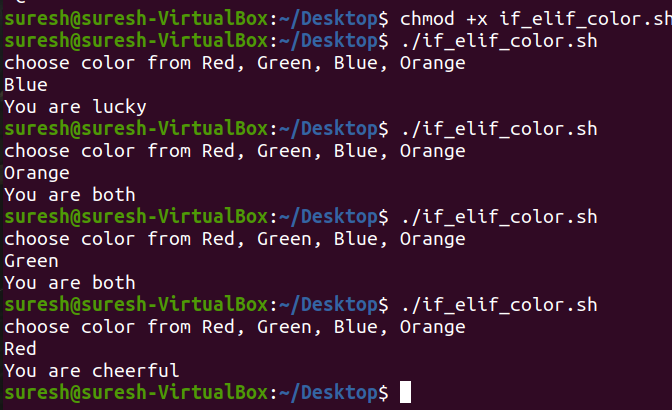
We have shown the example of choosing **color.**

**Condition:**

1. if [ $color == Red ]
2. elif [ $color == Blue ]



Look at the above snapshot, we have shown the script.



Look at the above snapshot, on **Red** color it goes to **if** part, on **Blue** color it goes to **elif** part and on other colors it goes to **else** part.