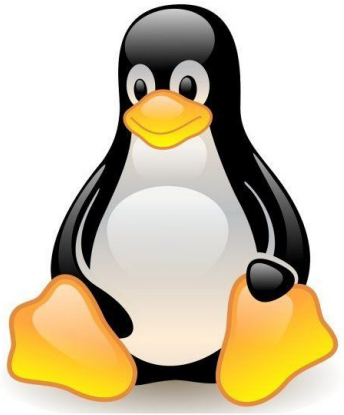
# Linux Plus

for

# AWS and DevOps

Session - 5



CLARUSWAY

WAY TO REINVENT YOURSELF

# Shell Scripting

BASH

THE BOURNE-AGAIN SHELL

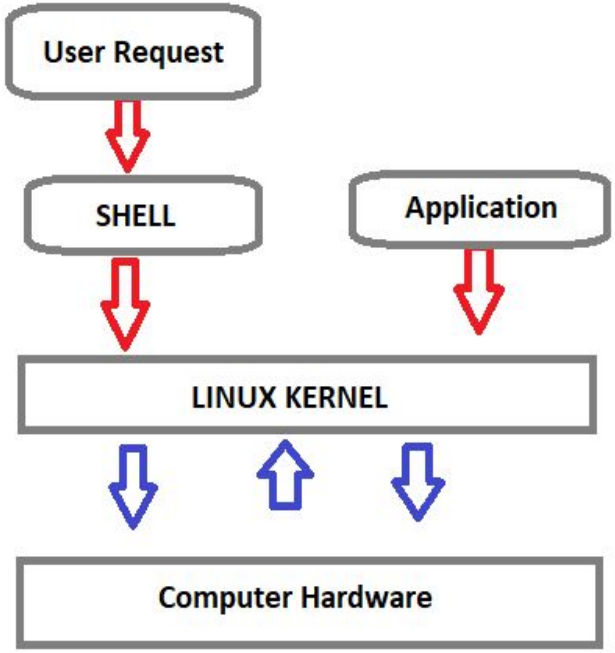
# Table of Contents



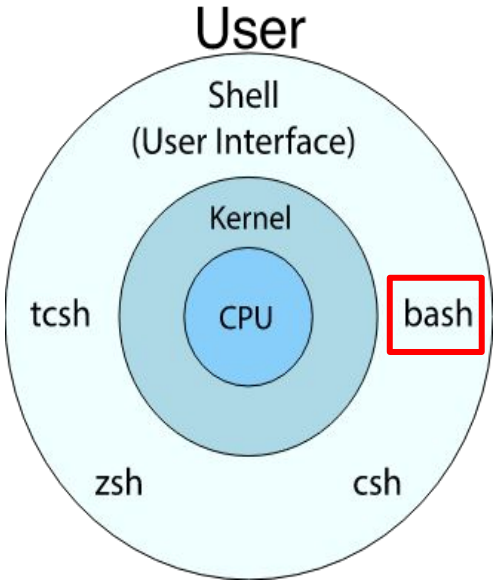
# What is SHELL?

Shell is **a program** that **receives** the user's **commands** and **gives** them to the **operating system to process** and displays the output.

The standard Linux shell is both a **command-line interpreter** and a **programming language.**



# Linux Shells




**SH**

**Bourne-Again SHell**


**Overview of Bash shell and command line interface**

The terms "shell" and "bash" are used interchangeably. But there is a subtle difference between the two.

The term "shell" refers to a program that provides a command-line interface for interacting with an operating system. Bash (Bourne-Again SHell) is one of the most commonly used Unix/Linux shells and is the default shell in many Linux distributions.

# 3 Shell Scripts

# Shell Scripts

**What is Shell Scripting?**

Shell Scripting is an open-source computer program designed to be run by the Unix/Linux shell which could be one of the following:

- **Bourne Shell** - Developed at AT&T Bell Labs by Steve Bourne, the Bourne shell is regarded as the first UNIX shell ever. It's denoted as sh.
- **GNU Bourne-Again Shell(BASH)**- Designed to be compatible with the Bourne shell. It incorporates useful features from different types of shells in Linux such as Korn shell and C shell. Most popular shell, default on most Linux systems. Installed on all Linux systems.
- **C Shell** - The C shell was created at the University of California by Bill Joy. It is denoted as csh. It was developed to include useful programming features like in-built support for arithmetic operations and a syntax similar to the C programming language.
- **Korn Shell** - The Korn shell was developed at AT&T Bell Labs by David Korn, to improve the Bourne shell. It's denoted as ksh. The Korn shell is essentially a superset of the Bourne shell.

# Shell Scripts

**What is Shell Scripting?**

- Typical activities that can be done in a shell, such as file manipulation, program execution, and printing text, can also be done with the shell script.

- Lengthy and repetitive sequences of commands can be combined into a single script that can be stored and executed anytime.

# Shell Scripts

```
clarus-linux@professor: ~
clarus-linux@professor:~$ vim class.sh
clarus-linux@professor:~$ chmod +x class.sh
clarus-linux@professor:~$ ./class.sh
Hello World!
clarus-linux@professor:~$
```

```
clarus-linux@professor: ~
#!/bin/bash

echo "Hello World!"

~
~
~
~
"class.sh" 5L, 35C
```

**Shebang (#!)**

- Before we add anything else to our script, we need to alert the system that a shell script is being started.
- This is done specifying `#!/bin/bash` on the first line, meaning that the script should always be run with bash, rather than another shell.
- It's called a `shebang` because the `#` symbol is called a hash, and the `!` symbol is called a bang.

# Shell Scripts

```
clarus-linux@professor: ~
clarus-linux@professor:~$ vim class.sh
clarus-linux@professor:~$ chmod +x class.sh
clarus-linux@professor:~$ ./class.sh
Hello World!
clarus-linux@professor:~$
```

```
clarus-linux@professor: ~
#!/bin/bash

echo "Hello World!"

~
~
~
~
"class.sh" 5L, 35C
```

chmod

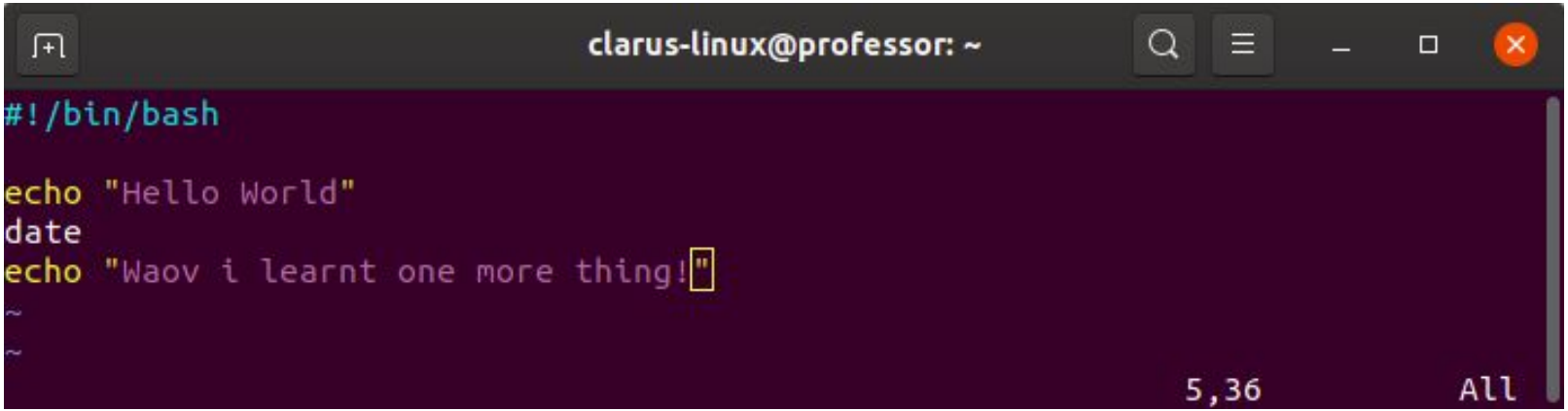
# Shell Scripts

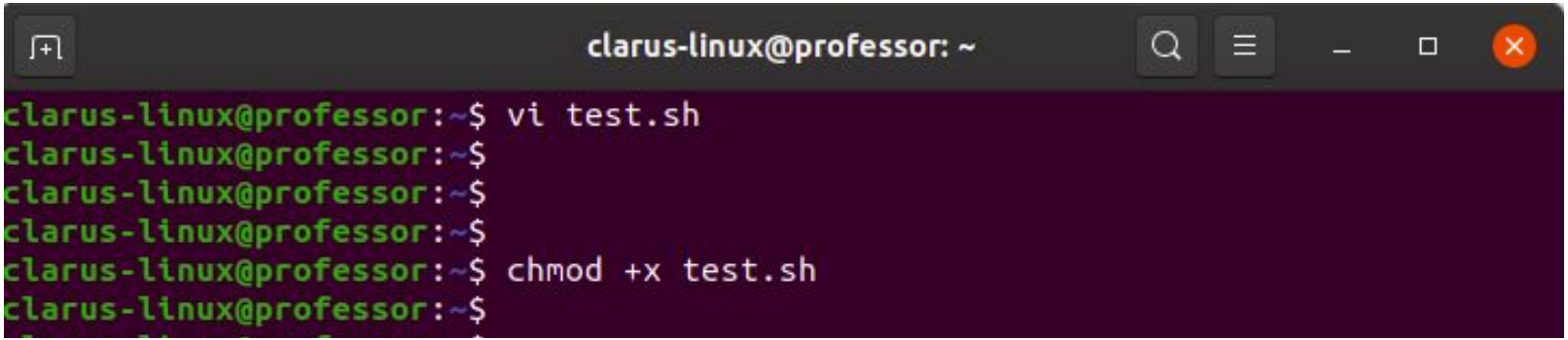
```
clarus-linux@professor: ~
clarus-linux@professor:~$ vim class.sh
clarus-linux@professor:~$ chmod +x class.sh
clarus-linux@professor:~$ ./class.sh
Hello World!
clarus-linux@professor:~$
```

```
clarus-linux@professor: ~
#!/bin/bash

echo "Hello World!"

~
~
~
~
"class.sh" 5L, 35C
```

"./"

# Shell Scripts



# HEREDOC Syntax

- Bash is a popular shell scripting language that allows us to execute commands and manipulate data. One of the features of Bash is the ability to use here-documents to provide multi-line input to a command or variable.
- A heredoc consists of the << (redirection operator), followed by a delimiter token.
- A delimiter is one or more characters that separate text strings.
- After the delimiter token, lines of string can be defined to form the content.
- Finally, the delimiter token is placed at the end to serve as the termination.
- The delimiter token can be any value as long as it is unique enough that it won't appear within the content.
- EOF means End of File

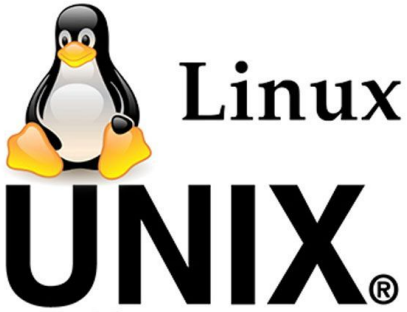
```
#!/bin/bash
echo "hello"
# date
pwd # This is an inline comment
# ls

cat << EOF
Welcome to the Linux Lessons.
This lesson is about the shell scripting
EOF

<< multiline-comment
pwd
ls
Everything inside the
HereDoc body is
a multiline comment
multiline-comment
```

```
[ec2-user@ip-172-31-37-149 ~]$ cat << EOF
Welcome to the Linux Lessons.
This lesson is about the shell scripting
EOF
```

# HomeWork 1

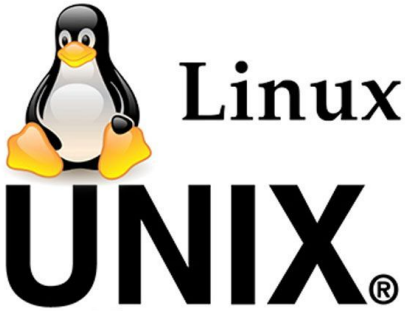
1. Create a script named: “**my-first-script.sh**”

   It should print: “**This is my first script.**”
2. Make the script executable.
3. Execute the script.



# Homework 2

Create an environment that you don’t need to provide “./” before your scripts while executing them.



# Variables

- A variable is pointer to the actual data. The shell enables us to create, assign, and delete variables.
- The name of a variable can contain only letters (a to z or A to Z), numbers ( 0 to 9) or the underscore character (_) and beginning with a letter or underscore character.
- The reason you cannot use other characters such as !, *, or - is that these characters have **a special meaning for the shell.**

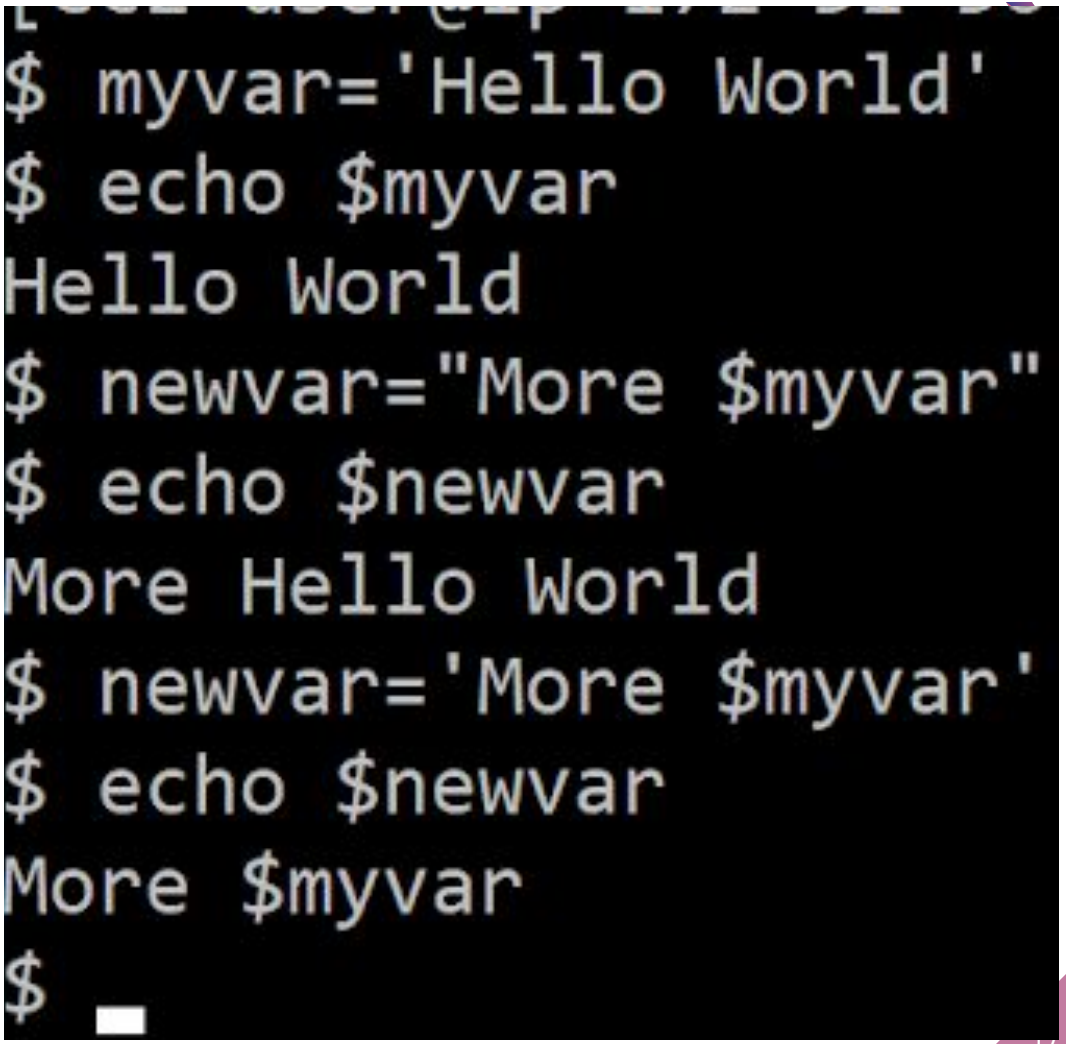
```
$VARIABLE=value
$echo $VARIABLE
value
$
$my_var=my_value
$echo $my_var
my_value
$
$my-var=my-value
my-var=my-value: command not found
$
$myvar?=my-value
myvar?=my-value: command not found
```

# Variables

**variable=value**

This is one of those areas where formatting is important. Note there is no space on either side of the equals ( = ) sign. We also leave off the $ sign from the beginning of the variable name when setting it.

```
sampledir=/etc
ls $sampledir
```



# Console input

**read [variable-name]**

- The Bash `read` command is a powerful built-in utility used to take user input
- The read command reads one line from standard input and assigns the values of each field in the input line to a shell variable using the characters

```
#!/bin/bash

echo "Enter your name: "
read name
echo Hello $name
~
~
```

```
[ec2-user@ip-172-31-36-108 ~]$ ./run.sh
Enter your name:
Raymond
Hello Raymond
[ec2-user@ip-172-31-36-108 ~]$
```

# Console input

**read**

```
#!/bin/bash

read -p "Enter Your Name: "  username
echo "Welcome $username!"
```

```
#!/bin/bash

read -s -p "Enter Password: " pswd
echo $pswd
```

```
#!/bin/bash

read -sp "Enter Password: " pswd
echo $pswd
```

```
#!/bin/bash

echo What cars do you like?

read -p "1." car1
read -p "2." car2
read -p "3." car3

echo Your first car was: $car1
echo Your second car was: $car2
echo Your third car was: $car3
```

# Command Line Arguments

- Command-line arguments are parameters that are passed to a script while executing them in the bash shell.
- The command-line arguments $1, $2, $3, ...$9 are positional parameters, with $0 pointing to the actual command, program, shell script, or function and $1, $2, $3, ...$9 as the arguments to the command.

# Command Line Arguments

**$0** - The name of the Bash script.
**$1 - $9** - The first 9 arguments to the Bash script.
**$#** - How many arguments were passed to the Bash script.
**$@** - All the arguments supplied to the Bash script.
**$?** - The exit status of the most recently run process.
**$$** - The process ID of the current script.
**$USER** - The username of the user running the script.
**$HOSTNAME** - The hostname of the machine the script is running on.
**$SECONDS** - The number of seconds since the script was started.
**$RANDOM** - Returns a different random number each time is it referred to.
**$LINENO** - Returns the current line number in the Bash script.

# Simple Arithmetic

**expr (evaluate expressions)** command **print** the value of expression to **standard output.**

**expr item1 operator item2**
**eg : $expr 12 + 8**

**let** is a builtin function of Bash that helps us to do simple arithmetic. It is similar to **expr** except instead of printing the answer **it saves the result to a variable.**

**let <arithmetic expression>**
**eg: let total=1+100**
**echo $total**

We can also evaluate arithmetic expression with double parentheses.

**$((arithmetic expression))**
**eg: ((total=60+30))**
**echo $total**

# Arithmetic Expressions

**`expr item1 operator item2`**

```
#!/bin/bash
first_number=8
second_number=2

echo "SUM="`expr $first_number + $second_number`
echo "SUB="`expr $first_number - $second_number`
echo "MUL="`expr $first_number \* $second_number`
echo "DIV="`expr $first_number / $second_number`
```

```
$ chmod +x cal.sh
$ ./cal.sh
SUM=10
SUB=6
MUL=16
DIV=4
```

# Arithmetic Expressions

`let [expression]`

```
#!/bin/bash

number1=8
number2=2

let total=number1+number2
let diff=number1-number2
let mult=number1*number2
let div=number1/number2

echo "Total = $total"
echo "Difference = $diff"
echo "Multiplication = $mult"
echo "Division = $div"
```

```
$ ./run.sh
Total = 10
Difference = 6
Multiplication = 16
Division = 4
```

# Arithmetic Expressions

`$((Expression))`

`((Expression))`

```
#!/bin/bash

number1=8
number2=2

echo "Total = $((number1+number2))"

((total=number1+number2))
echo "Total = $total"
~
```

```
[ec2-user@ip-172-31-91-206 ~]$ ./run.sh
Total = 10
Total = 10
[ec2-user@ip-172-31-91-206 ~]$
```

# HomeWork 3

1. Ask user to enter two numbers to variables **num1** and **num2.**
2. Calculate the total of 2 numbers.
3. Print the **total** number and increase it by 1.
4. Print the new value of the **total** number.
5. Subtract **num1** from the **total** number and print result.
6. Change the **num1** and **num2** variables to be passed from the **Command line arguments** instead of receiving them from the user

# HomeWork 4

1. Create a script named **calculate.sh:**

   Create a variable named **base_value** with default value of **5**

   Request another number from user and assign it to **user_input** variable

   Add **user_value** to the **base_value** and assign it to **total** variable

   Print **total** to the screen with the message “**Total value is:** ”

2. Make the script executable.
3. Execute the script.

# THANKS!

## Any questions?

You can find me at:

- @sumod
- sumod@clarusway.com