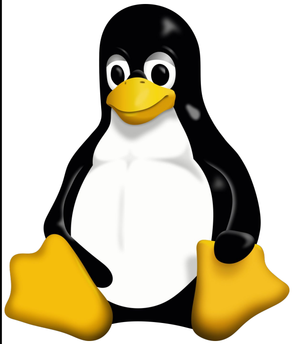


The Linux Week

Introduction to Linux

The Tech Stuff by Maya Mnaizel

Welcome to Day 4

Day 4

- ★ Introduction to Bash Scripting
  - What is Bash?
  - Adding the shebang line
  - Command review
  - Why use Bash?
- ★ Variables and User Input
- ★ Conditional Statements
- ★ Loops in Bash
  - For Loop
  - While Loop
- ★ Functions and Script Organization

Bash

Understanding Bash

01

What is Bash?

Bash stands for Bourne Again Shell. It is a command-line interpreter, or shell, for the GNU operating system. Bash is widely used in various Unix-like operating systems, including Linux and macOS.

It serves as both a command language and a scripting language, enabling users to interact with the system and automate tasks.

Basic Components of Bash

Shebang

The shebang line at the beginning of a script specifies the interpreter to be used. For Bash scripts, it is typically #!/bin/bash.

Commands

Bash supports a wide range of built-in commands (e.g., cd, ls, echo) and allows the execution of external programs.

Basic Components of Bash

Variables

Variables store data that can be used and manipulated within a script. They are defined using the syntax x=value.

Control Structure

Bash supports various control structures, such as if statements, for and while loops.

Commands

Review

1	Echo	Prints text to the terminal.
2	Read	Reads user input and stores it in a variable
3	pwd	Print working directory.
4	ls	List directory contents.

Example

```
#!/bin/bash

# A simple Bash script to greet the user

echo "What is your name?"

read name

echo "Hello, $name! Welcome to Bash scripting."
```

Why Use Bash?

Efficiency

Automation of tasks, reducing the need for repetitive manual operations.

Simplicity

Relatively easy to write and understand

Power

Handle complex tasks and integrate with other tools and languages

Portability

Can run on various Unix-like systems with little to no modification

Variables

And User Input

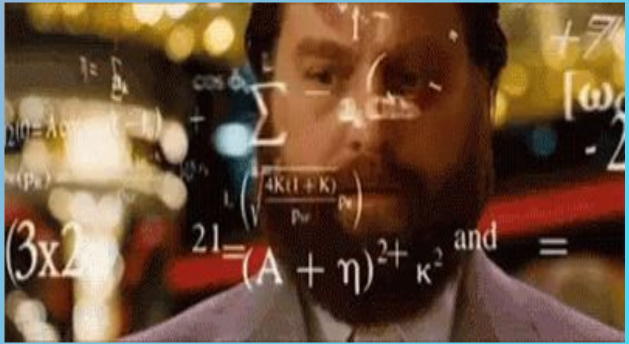
02

Variables

Declaration

the syntax `VARIABLE_NAME=value`

```
name="John"
age=25
```



Accessing

To access the value of a variable -> a dollar sign ($).

```
echo "Name: $name"
echo "Age: $age"
```

Example

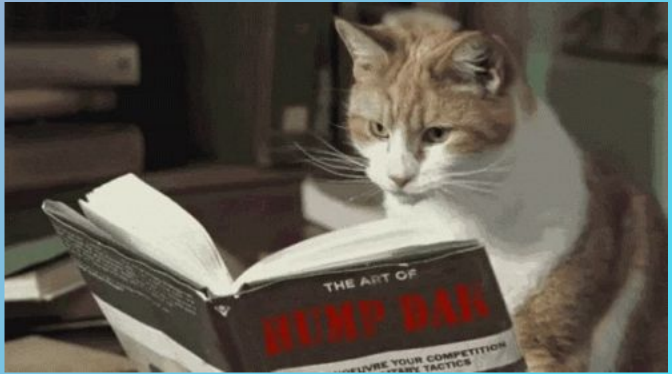
```
#!/bin/bash

# Declaring variables
name="Alice"
age=30

# Accessing and printing variables
echo "Name: $name"
echo "Age: $age"

# Using command substitution
current_date=$(date)
echo "Current Date and Time: $current_date"
```

User Input

Read

The `read` command reads a line of input from the terminal and stores it in a variable.

```
echo "Enter your name:"
read user_name
```



Prompting

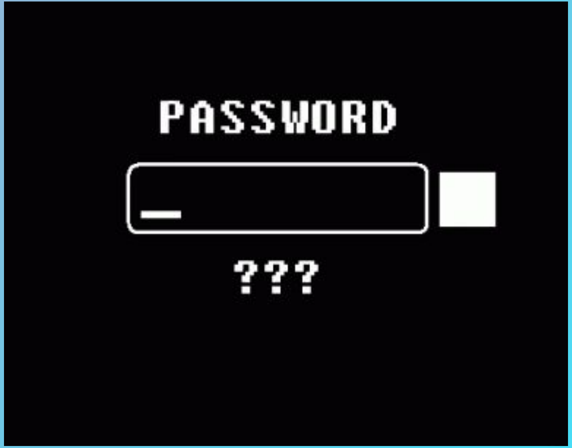
You can prompt the user for input by using the `-p` option with the `read` command

```
read -p "Enter your age: " user_age
echo "You are $user_age years old."
```

Silent Input

For sensitive information, such as passwords, you can use the `-s` option to hide the input.

```
read -s -p "Enter your password: " user_password
echo
echo "Password entered."
```



Example

```
#!/bin/bash

# Prompting the user for their name
echo "Enter your name:"
read user_name
echo "Hello, $user_name!"

# Prompting the user for their age
read -p "Enter your age: " user_age
echo "You are $user_age years old."

# Reading sensitive input silently
read -s -p "Enter your password: " user_password
echo
echo "Password entered."
```

Conditions

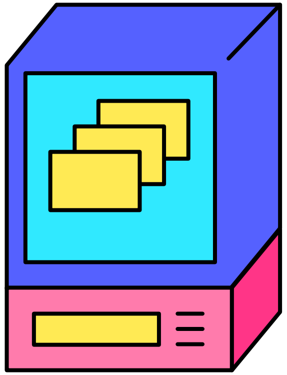
Conditionals statement

03

IF Statements

- The basic `if` statement checks if a condition is true and executes the commands within the block if it is.

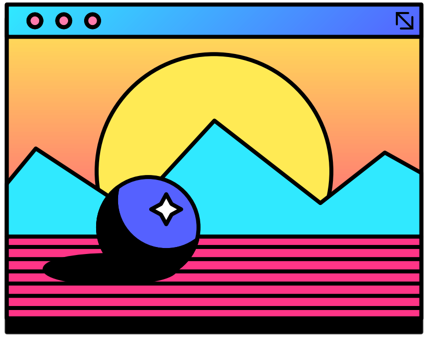
```
if [ condition ]; then
    # Commands to execute if the condition is true
fi
```



IF Statements

- The `if-elif-else` statement allows for multiple conditions to be checked in sequence.

```
if [ condition1 ]; then
    # Commands to execute if condition1 is true
elif [ condition2 ]; then
    # Commands to execute if condition2 is true
else
    # Commands to execute if none of the conditions are true
fi
```



Comparison Operators

Numeric Operator

-eq: Equal to
-ne: Not equal to
-lt: Less than
-le: Less than or equal to
-gt: Greater than
-ge: Greater than or equal to

String Operator

=: Equal to
!=: Not equal to
-z: String is null (zero length)
-n: String is not null (non-zero length)

Example

```
# Numeric comparison
if [ $a -eq $b ]; then
   echo "a is equal to b"
fi

# String comparison
if [ "$str1" = "$str2" ]; then
   echo "str1 is equal to str2"
fi
```

Loops

For and While Loops

04

For Loop

```
for variable in list
do
    # Commands to execute
done
```

For Loop Example

```
#!/bin/bash

for fruit in apple banana orange
do
   echo "I like $fruit"
done
```

While Loop

```
while [ condition ]
do
   # Commands to execute
done
```

While Loop Example

```
#!/bin/bash

counter=1

while [ $counter -le 5 ]
do
   echo "Counter: $counter"
   ((counter++))
done
```

Loop Control Commands

Break

```
#!/bin/bash

for num in 1 2 3 4 5
do
   if [ $num -eq 3 ]; then
      break
   fi
   echo "Number: $num"
done
```

Continue

```
#!/bin/bash

for num in 1 2 3 4 5
do
   if [ $num -eq 3 ]; then
      continue
   fi
   echo "Number: $num"
done
```

Nested Loops

```
#!/bin/bash

for i in 1 2 3
do
   for j in a b c
   do
      echo "i: $i, j: $j"
   done
done
```

Functions

05

And Script Organization

Functions in Bash

Functions in Bash allow you to group commands into reusable blocks

Defining Functions (1)

```
function_name() {
  # Commands
}
```

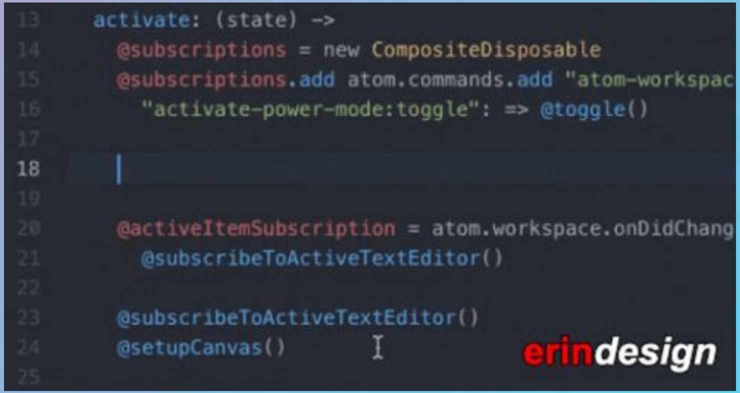
Defining Functions (2)

```
function function_name {
  # Commands
}
```

Functions in Bash

```
#!/bin/bash

add_numbers() {
   local sum=$(( $1 + $2 ))
   echo "Sum: $sum"
}

add_numbers 5 7
```

Script Organization



```
  activate: (state) ->
    @subscriptions = new CompositeDisposable
    @subscriptions.add atom.commands.add "atom-workspac
      "activate-power-mode:toggle": => @toggle()



    @activeItemSubscription = atom.workspace.onDidChang
      @subscribeToActiveTextEditor()

    @subscribeToActiveTextEditor()
    @setupCanvas()
```

Script Organization

Using Shebang

The shebang line at the beginning of the script specifies the interpreter to be used.

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

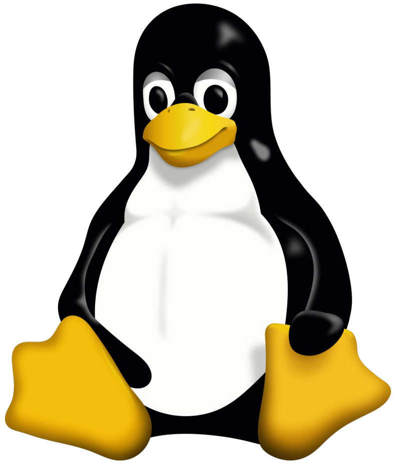
Comments

Use comments to explain the purpose of the script

```
#!/bin/bash
# This is a comment
```

Q/A Session

Thank you !



End of Day 4!

By Maya Mnaizel