Shell Scripting

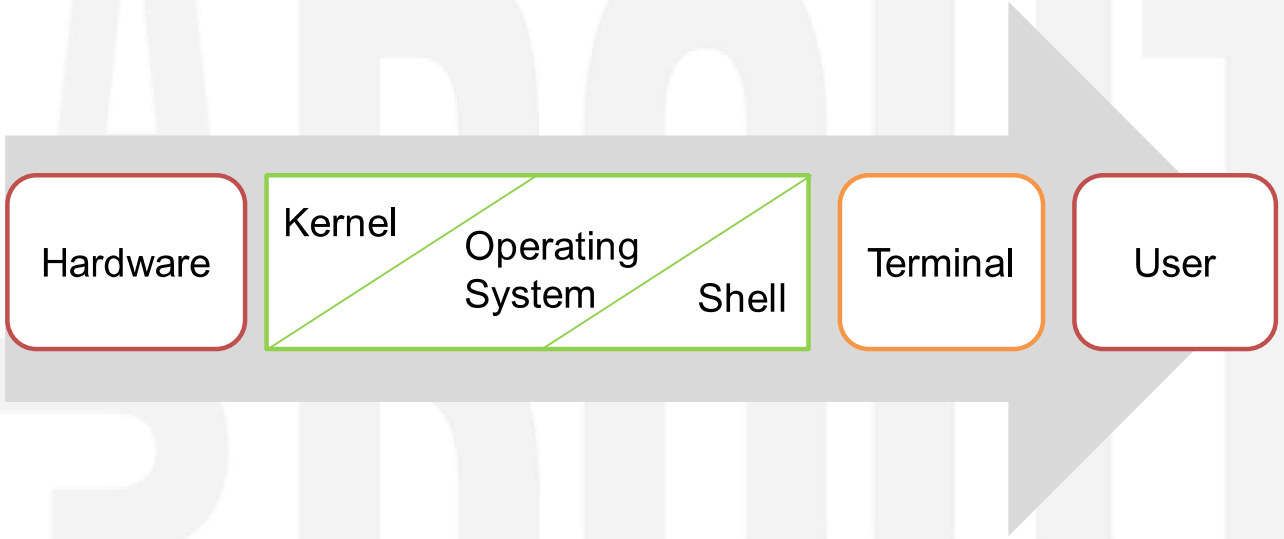
#FullStackFuture

Outcome

After going through this video, you will be able to understand and demonstrate:

- Shell
- Use of Shells
- Types of Shells
- How Shell Works?
- Shell Scripting
- How To Create Your First Bash Script?

What is Shell?



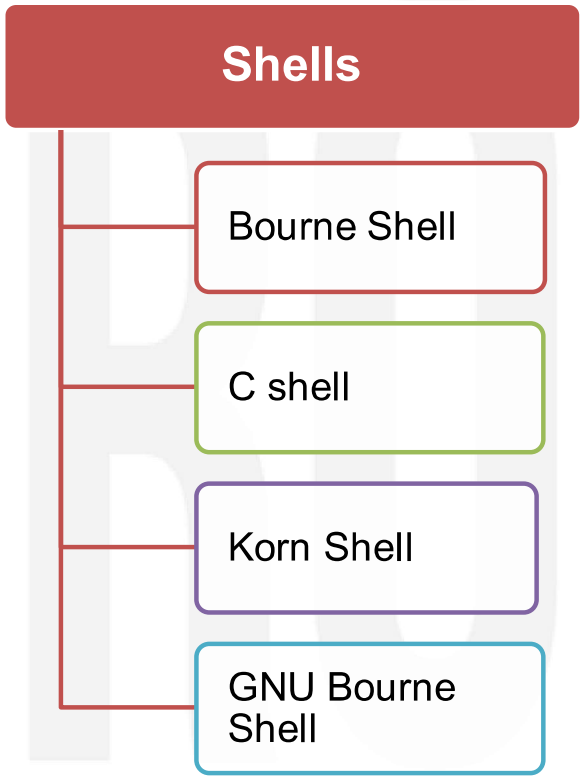
Why Use Shells?

A **Shell** provides you with an interface to the Unix system.

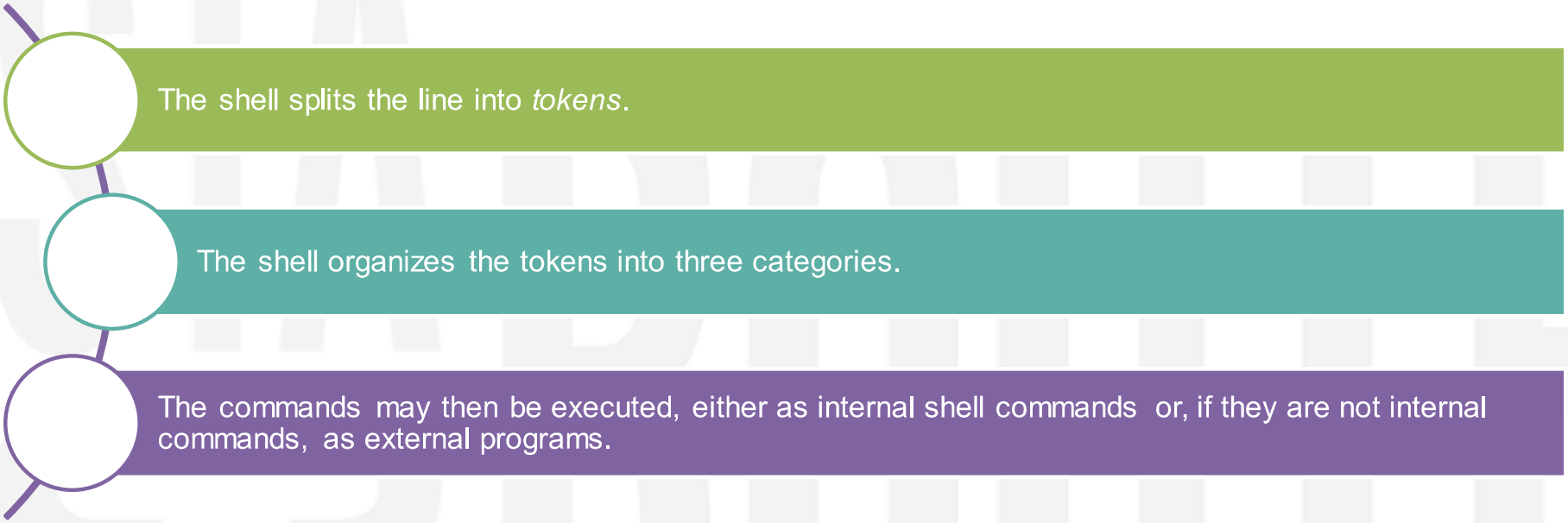
It is common on nearly every OS because they are efficient and easily updatable.

They start quickly and they’re easy to debug.

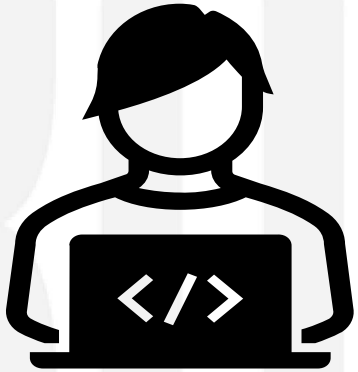
Types of Shells



How Shell Works?



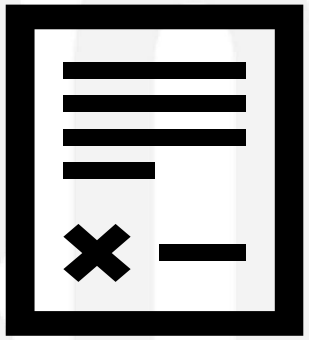
What is Shell Scripting?



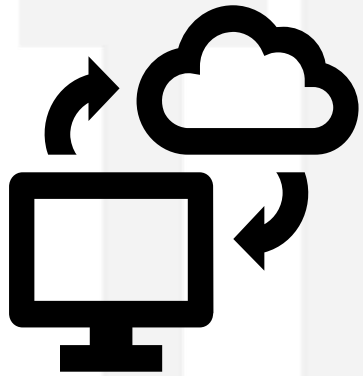
Scripting is a program designed to run in the UNIX shell

Examples of Shell Scripting

Monitor systems

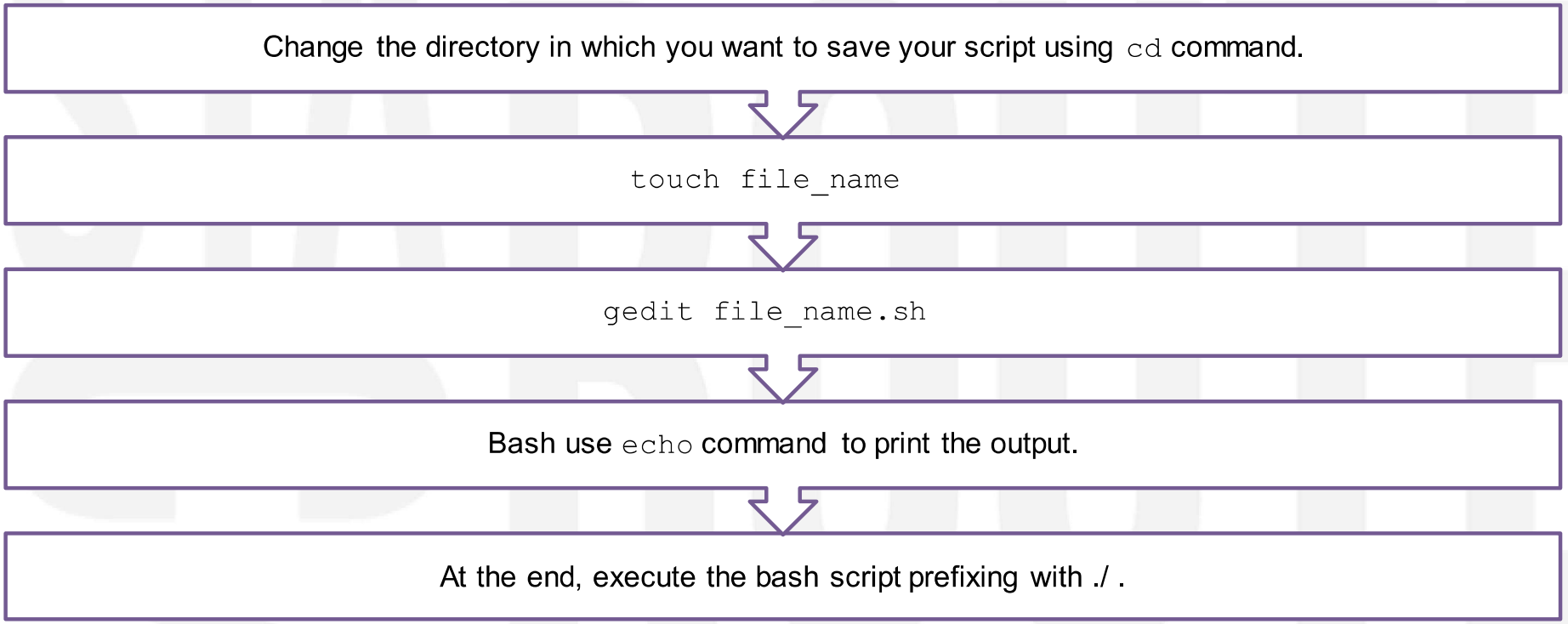


Delete duplicate files



Routine backups

How To Create Your First Bash Script



How To Define Variables

Syntax

```
variable_name = value
```

Arithmetic Expressions

Operator	Usage
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
%	Modulus

How To Read User Input

Syntax

```
read <variable_name>
```

Conditional Statements

- `if...then...fi` statements
- `if...then...else...fi` statements
- `if..elif..else..fi` statements
- `if..then..else..if..then..fi..fi..` (Nested Conditionals)

Example Syntax:

```
if [[ condition ]]
then
    statement
elif [[ condition ]]; then
    statement
else
    do this by default
fi
```

Numeric Comparison Logical Operators

OPERATION	SYNTAX	EXPLANATION
Equality	`num1 -eq num2`	num1 equal to num2
Greater than equal to	`num1 -ge num2`	num1 greater than equal to num2
Greater than	`num1 -gt num2`	num1 greater than num2
Less than equal to	`num1 -le num2`	num1 less than equal to num2
Less than	`num1 -lt num2`	num1 less than num2
Not Equal to	`num1 -ne num2`	num1 not equal to num2

Case Statements

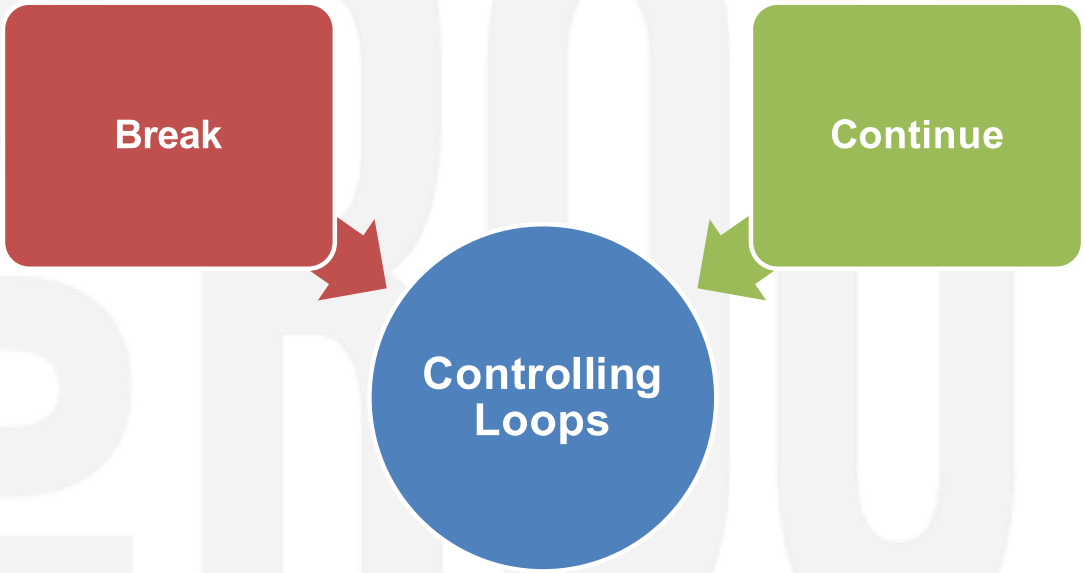
Syntax

```
case <variable> in
     <pattern 1>)
     <commands>
          ;;
      <pattern 2>)
    <other commands>
          ;;
        esac
```

Loops

for loop	• for var in <list> do <commands> done
until loop	• until [<some test>] do <commands> done
while loop	• while [<some test>] do <commands> done

Controlling Loops: Break and Continue



Functions

Syntax

```
function_name () {
    <commands>
}
```

or

```
function function_name {
    <commands>
}
```

Reading and Writing Files

Command	Operation
`-d file`	True if the file is a directory
`-e file`	True if the file exists (note that this is not particularly portable, thus -f is generally used)
`-f file`	True if the provided string is a file
`-g file`	True if the group id is set on a file
`-r file`	True if the file is readable
`-s file`	True if the file has a non-zero size
`-u file`	True if the user id is set on a file
`-w file`	True if the file is writable
`-x file`	True if the file is an executable

Just a Minute

1. **The shell script is a _____.**

   a) File containing a series of commands
   b) Group of commands
   c) File containing special symbols
   d) None of the above

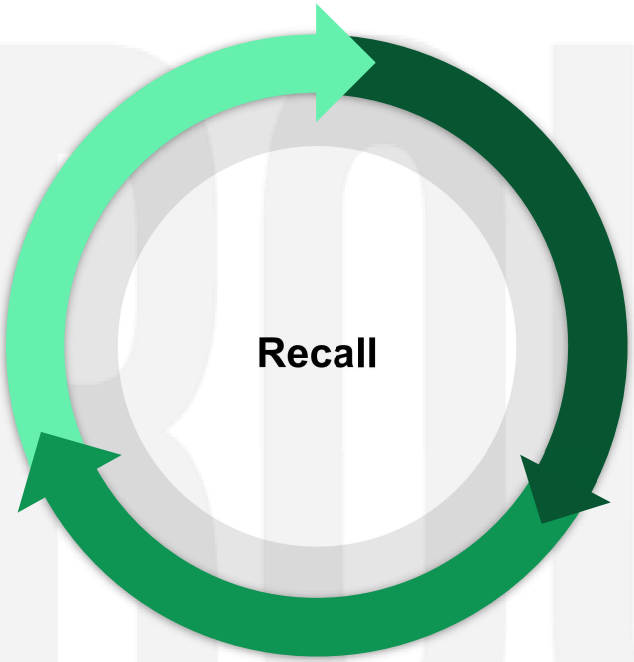
   Correct Answer: (a)

2. **In order to run a script, we have to make it executable first by using the command _____.**

   a) `chmod +w`
   b) `chmod +x`
   c) `chmod +r`
   d) None of the above

   Correct Answer: (b)

Recall



Thank You

#FullStackFuture