

---

# Curso de Programación en Bash Shell

Marco Antonio Toscano

# Marco Antonio Toscano

---

- Quito - Ecuador
- Ingeniero en Sistemas  
Escuela Politécnica Nacional
- Experto en tecnologías Java, Linux, Open Source



[www.youtube.com/user/matoosfe](http://www.youtube.com/user/matoosfe)



[@martosfre](https://twitter.com/martosfre)



[www.marcotoscano.org](http://www.marcotoscano.org)

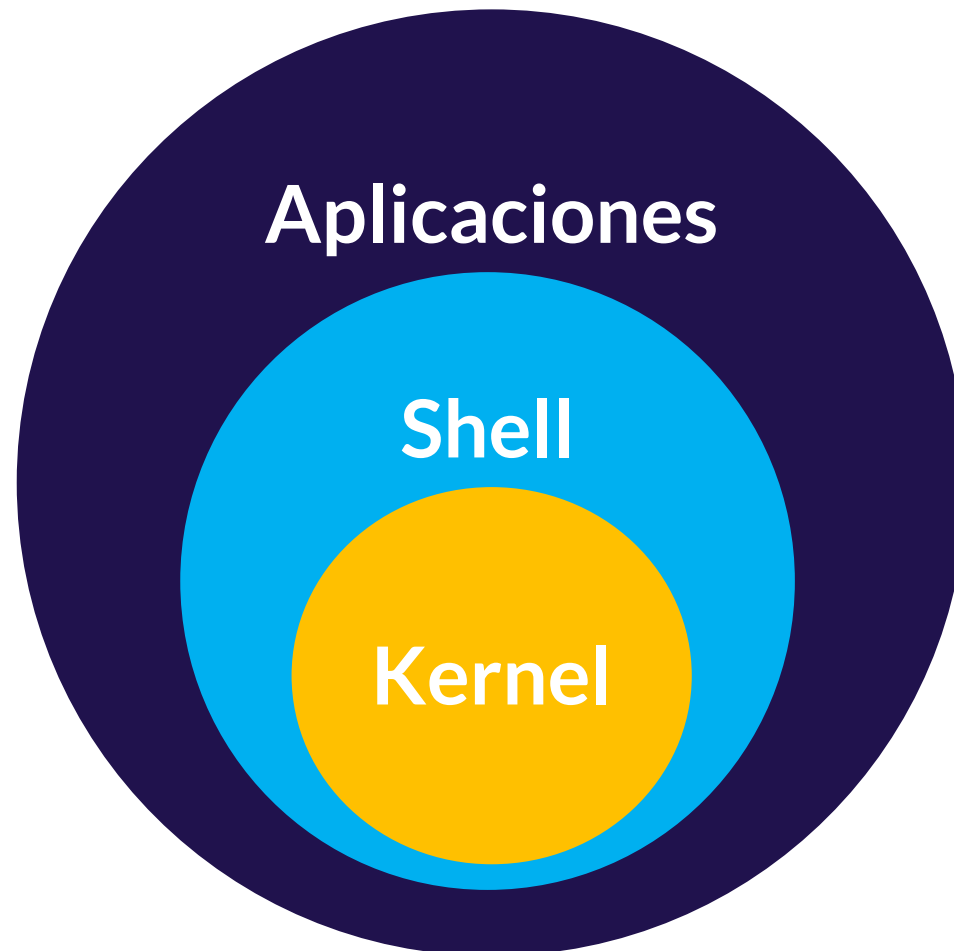
---

# ¿Qué es la programación Shell?

---

# Linux

Linux consta de algunas partes principales:



---

# Tipos de Shells (Formato Lectura)

SH

BASH

KSH

CSH

---

# ¿Qué es el bash scripting?

La idea básica de bash scripting o programación en shell es poder **ejecutar múltiples comandos de forma secuencial** para automatizar una tarea en específico.

Estos comandos son colocados en un archivo de texto simple y ejecutados en un terminal por el usuario.



---

# ¿Cómo preparar el editor de texto?

- Para crear un programa necesitarás utilizar un editor de texto ya sea en terminal o modo gráfico. Nosotros utilizaremos **vim** basado en terminal.
- Comandos Principales.

# **Configuraciones VIM**

- set showmode
- set autoindent
- set tabstop=4
- set expandtab
- syntax on



# Configuraciones VIM



A screenshot of a VIM editor window. The title bar at the top shows a home icon, the text "martosfre — vim .vimrc — 63x24", and three window control buttons (red, yellow, green). The editor area has a white background with syntax-highlighted text. The first line is "set showmode" in purple. The second line is "set autoindent" in purple. The third line is "set tabstop=4" in purple. The fourth line is "set expandtab" in purple. The fifth line is "syntax on" in green, with a green cursor at the end. Below this, there are five lines of blue tilde (~) characters, and a partial line at the bottom.

```
set showmode
set autoindent
set tabstop=4
set expandtab
syntax on
~
~
~
~
~
~
```

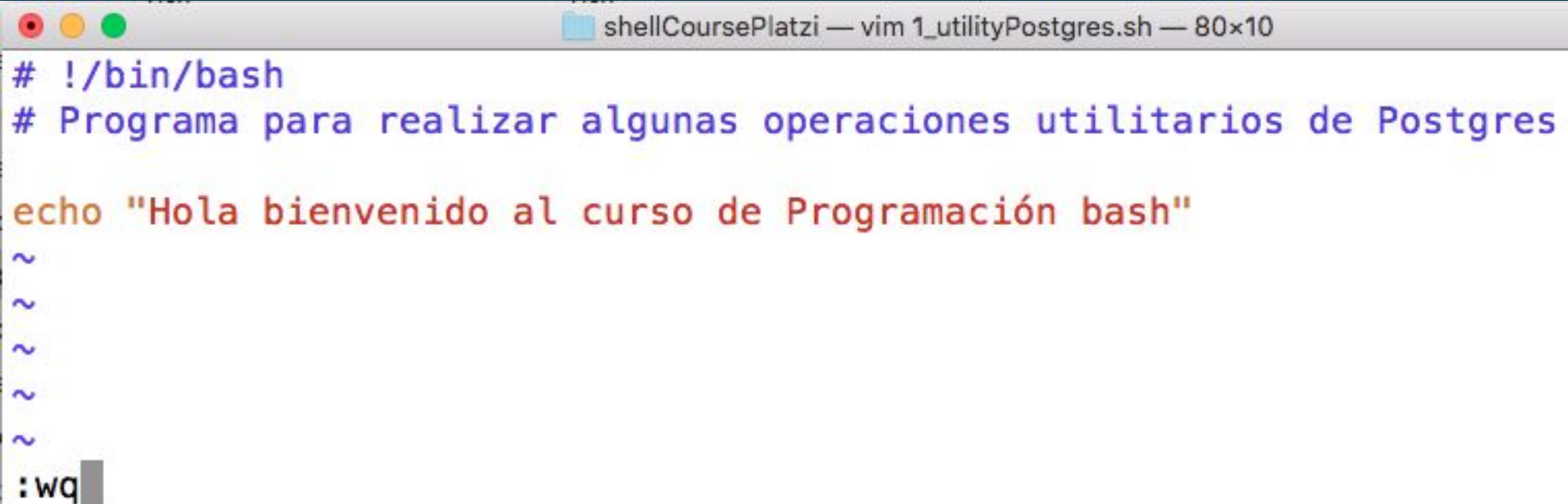
---

# ¿Cómo crear nuestro primer script?

Crear un archivo con la extensión sh, para lo cual nos ubicamos en el directorio **\$HOME** y ejecutamos el siguiente comando **vim utilityPostgres.sh**

```
MacBook-Pro-de-Marco:shellCoursePlatzi martosfre$ pwd  
/Users/martosfre/shellCoursePlatzi  
MacBook-Pro-de-Marco:shellCoursePlatzi martosfre$ vim 1_utilityPostgres.sh
```

# Nuestro Primer Script



The image shows a terminal window with a light gray title bar. The title bar contains three colored window control buttons (red, yellow, green) on the left and a text label 'shellCoursePlatzi — vim 1\_utilityPostgres.sh — 80x10' on the right. The terminal area has a white background with text in different colors: purple for comments, red for the echo command, and blue for tilde characters. The text is as follows:

```
# !/bin/bash
# Programa para realizar algunas operaciones utilitarios de Postgres

echo "Hola bienvenido al curso de Programación bash"

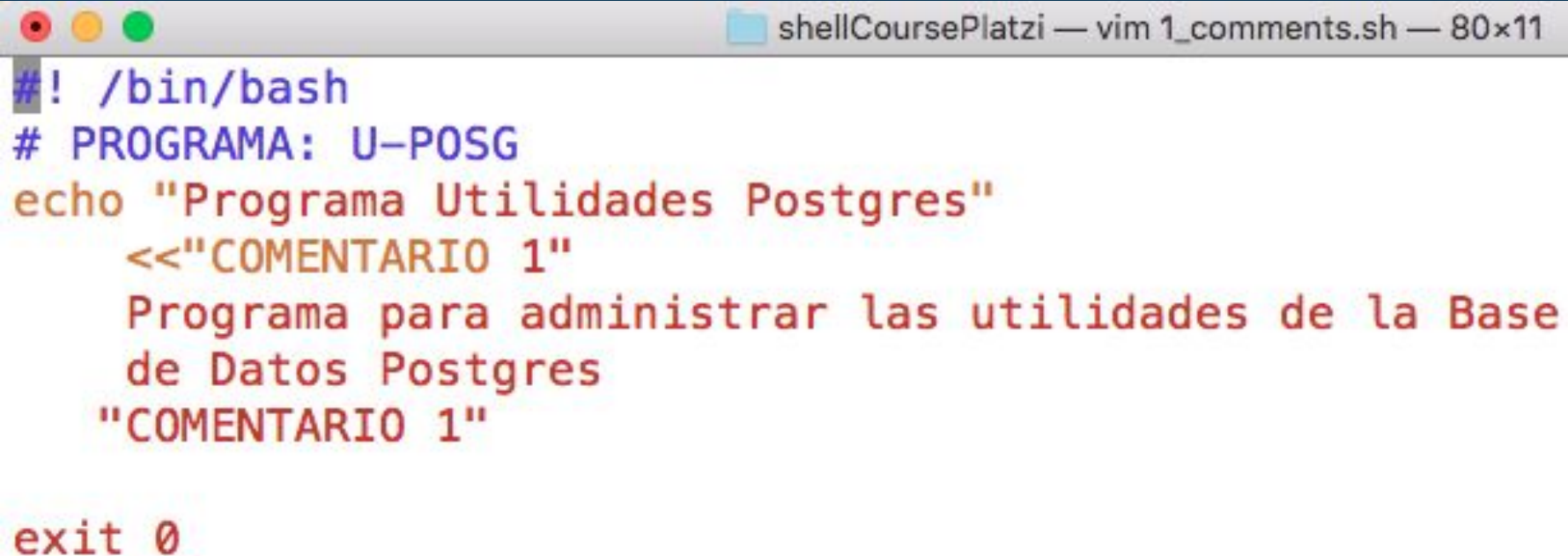
~
~
~
~
~

:wq
```

# **Tipos Comentarios VIM**

- Simple
- Multilínea

# Comentarios Script



A terminal window titled "shellCoursePlatzi — vim 1\_comments.sh — 80x11" displays a shell script. The script starts with a shebang line, followed by a comment line, an echo statement, a here-document containing a comment and a description, and finally an exit statement.

```
#!/bin/bash
# PROGRAMA: U-POSG
echo "Programa Utilidades Postgres"
<<"COMENTARIO 1"
Programa para administrar las utilidades de la Base
de Datos Postgres
"COMENTARIO 1"

exit 0
```

# **Ejecutar Script**

---

- Utilizando comando Bash
- Modo Standalone

# Ejecutar Script

~/shellCourse — -bash

```
[MacBook-Pro-de-Marco:shellCourse martosfre$ bash 1_utilityPostgres.sh
```

~/shellCourse — -bash

```
[MacBook-Pro-de-Marco:shellCourse martosfre$ ls -l 1_utilityPostgres.sh
```

```
-rw-r--r--  1 martosfre  staff  75 Dec  3 12:09 1_utilityPostgres.sh
```

```
[MacBook-Pro-de-Marco:shellCourse martosfre$ chmod +x 1_utilityPostgres.sh
```

```
[MacBook-Pro-de-Marco:shellCourse martosfre$ ls -l 1_utilityPostgres.sh
```

```
-rwxr-xr-x  1 martosfre  staff  75 Dec  3 12:09 1_utilityPostgres.sh
```

```
MacBook-Pro-de-Marco:shellCourse martosfre$
```

---

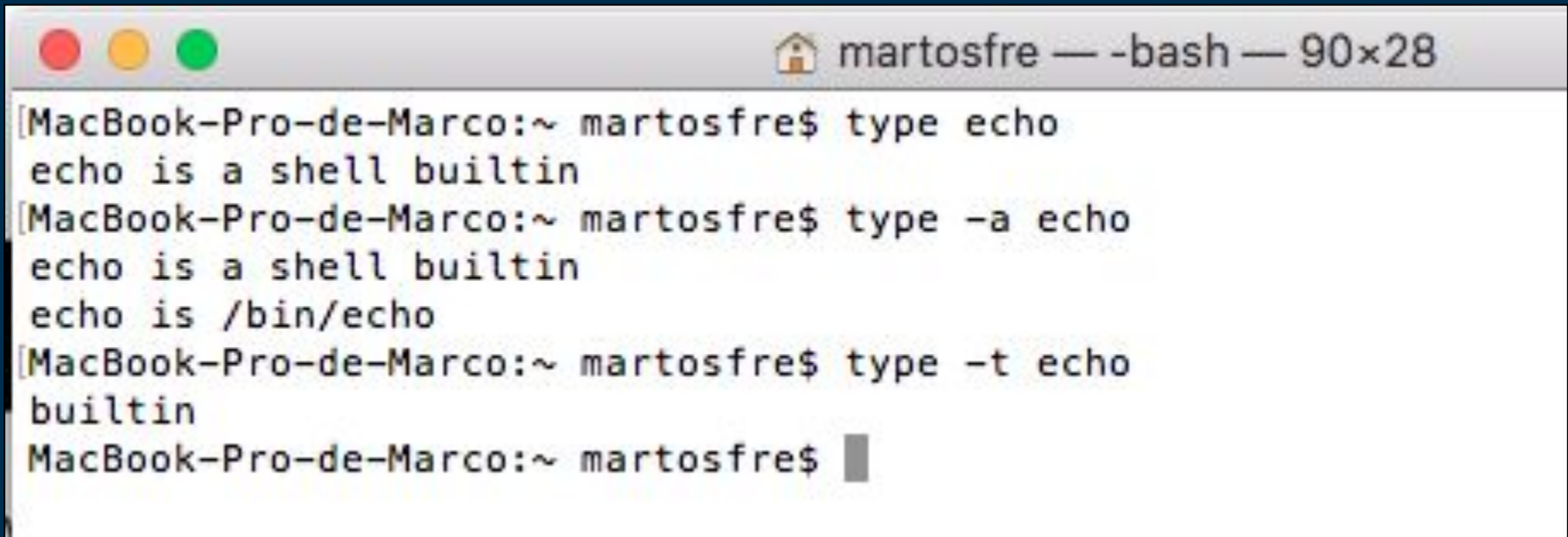
# ¿Cómo asegurar un nombre único?

Se puede verificar a través del comando **type** con las siguientes opciones:

- **type archivo.** Determina el tipo y la ubicación.
- **type -a archivo.** Imprime todos los archivos encontrados si el nombre no es único.
- **type -t archivo.** Imprime el tipo del archivo.



# Nombre Único Archivo



```
martosfre — -bash — 90x28
[MacBook-Pro-de-Marco:~ martosfre$ type echo
echo is a shell builtin
[MacBook-Pro-de-Marco:~ martosfre$ type -a echo
echo is a shell builtin
echo is /bin/echo
[MacBook-Pro-de-Marco:~ martosfre$ type -t echo
builtin
MacBook-Pro-de-Marco:~ martosfre$
```

---

# Programación Shell Básica



---

# ¿Cómo se realiza la declaración de variables?

Las variables permiten almacenar información de algún tipo como numérica, cadena, boolean, etc. Existen dos tipos de variables:

- Entorno
- Usuario

---

# ¿Qué alcance tienen las variables ?

El alcance que tienen las variables definidas en un script está **limitada al proceso que lo creó**; es decir no pueden usarse en otro script a menos que sea visible a nivel del sistema utilizando el comando **EXPORT**.

# Declaración y Alcance Variables

```
shellCoursePlatzi — vim 2_variables.sh — 80x17
# !/bin/bash
# Programa para revisar la declaración de variables
# Autor: Marco Toscano Freire - @martosfre

opcion=0
nombre=Marco

echo "Opción: $opcion y Nombre: $nombre"

# Exportar la variable nombre para que este disponible a los demás procesos
export nombre

# Llamar al siguiente script para recuperar la variable
./2_variables_2.sh
```

```
shellCoursePlatzi — vim 2_variables_2.sh — 80x17
# !/bin/bash
# Programa para revisar la declaración de variables
# Autor: Marco Toscano Freire - @martosfre

echo "Opción nombre pasada del script anterior: $nombre"
```

# **Tipos de Operadores**

- Aritméticos
- Relacionales
- Lógicos
- Asignación
- Bitwise

# Operadores

```
shellCoursePlatzi — vim 3_tipoOperadores.sh — 80x24
# ! /bin/bash
# Programa para revisar los tipos de operadores
# Autor: Marco Toscano - @martosfre

numA=10
numB=4

echo "Operadores Aritméticos"
echo "Números: A=$numA y B=$numB"
echo "Sumar A + B =" $((numA + numB))
echo "Restar A - B =" $((numA - numB))
echo "Multiplicar A * B =" $((numA * numB))
echo "Dividir A / B =" $((numA / numB))
echo "Residuo A % B =" $((numA % numB))

echo -e "\nOperadores Relaciones"
echo "Números: A=$numA y B=$numB"
echo "A > B =" $((numA > numB))
echo "A < B =" $((numA < numB))
echo "A >= B =" $((numA >= numB))
echo "A <= B =" $((numA <= numB))
echo "A == B =" $((numA == numB))
```

---

# ¿Cómo ejecutar un script con argumentos?

Identificador	Descripción
\$0	El nombre del script
\$1 al <b>\${10}</b>	El número de argumento, si son más de un dígito se utiliza las llaves
\$ #	Contador de argumentos
\$ *	Refiere a todos los argumentos



# Scripts con Argumentos

```
shellCoursePlatzi — vim 4_argumentos.sh — 68x24
# ! /bin/bash
# Programa para ejemplificar el paso de argumentos
# Autor: Marco Toscano Freire - @martosfre
nombreCurso=$1
horarioCurso=$2

echo "El nombre del curso es: $nombreCurso dictado en el horario de
$horarioCurso"
echo "El número de parámetros enviados es: $#"
```

```
echo "Los parámetros enviados son: $*"
~
```

```
shellCoursePlatzi — -bash — 67x24
MacBook-Pro-de-Marco:shellCoursePlatzi martosfre$ ./4_argumentos.sh
"Programación Bash" "18:00 a 20:00"
El nombre del curso es: Programación Bash dictado en el horario de
18:00 a 20:00
El número de parámetros enviados es: 2
Los parámetros enviados son: Programación Bash 18:00 a 20:00
MacBook-Pro-de-Marco:shellCoursePlatzi martosfre$
```

---

# ¿Cómo realizar la sustitución de comandos en variables?

La idea de la sustitución de comandos en variables es almacenar la salida de una ejecución de un comando en una variable. Se puede realizar:

- Usando el backtick character. ( ` )
- Usando el signo de dólar con el formato **\$(comando)**

# Command Substitution

```
shellCoursePlatzi — vim 5_subtitucionComand.sh — 67x24
# ! /bin/bash
# Programa para revisar como ejecutar comandos dentro de un programa
# y almacenar en una variable para su posterior utilización
# Autor: Marco Toscano Freire - @martosfre

ubicacionActual=`pwd`
infoKernel=$(uname -a)

echo "La ubicación actual es la siguiente: $ubicacionActual"
echo "Información del Kernel: $infoKernel"
```

```
shellCoursePlatzi — -bash — 67x24
MacBook-Pro-de-Marco:shellCoursePlatzi martosfre$ ./5_subtitucionComand.sh
La ubicación actual es la siguiente: /Users/martosfre/shellCoursePlatzi
Información del Kernel: Darwin MacBook-Pro-de-Marco.local 17.7.0 Darwin Kernel Version 17.7.0: Wed Oct 10 23:06:14 PDT 2018; root:xnu-4570.71.13~1/RELEASE_X86_64 x86_64
MacBook-Pro-de-Marco:shellCoursePlatzi martosfre$
```

---

# ¿Cómo realizar el debug de un script?

Hay dos opciones para realizar el debug utilizando el comando **bash**.

- -v .- Utilizado para ver el resultado detallado de nuestro script, evaluado línea por línea.
- -x .- Utilizado para ver desplegar la información de los comandos que son utilizados, capturando el comando y su salida.



# Debugging

```
shellCoursePlatzi — -bash — 71x24
MacBook-Pro-de-Marco:shellCoursePlatzi martosfre$ bash -x 5_subtitucion
Comand.sh
++ pwd
+ ubicacionActual=/Users/martosfre/shellCoursePlatzi
++ uname -a
+ infoKernel='Darwin MacBook-Pro-de-Marco.local 17.7.0 Darwin Kernel Ve
rsion 17.7.0: Wed Oct 10 23:06:14 PDT 2018; root:xnu-4570.71.13~1/RELEA
SE_X86_64 x86_64'
+ echo 'La ubicación actual es la siguiente: /Users/martosfre/shellCour
sePlatzi'
La ubicación actual es la siguiente: /Users/martosfre/shellCoursePlatzi
+ echo 'Información del Kernel: Darwin MacBook-Pro-de-Marco.local 17.7.
0 Darwin Kernel Version 17.7.0: Wed Oct 10 23:06:14 PDT 2018; root:xnu-
4570.71.13~1/RELEASE_X86_64 x86_64'
Información del Kernel: Darwin MacBook-Pro-de-Marco.local 17.7.0 Darwin
Kernel Version 17.7.0: Wed Oct 10 23:06:14 PDT 2018; root:xnu-4570.71.
13~1/RELEASE_X86_64 x86_64
MacBook-Pro-de-Marco:shellCoursePlatzi martosfre$
```

---

# Creación scripts interactivos

The bottom left corner of the slide features decorative geometric shapes. It includes a dark blue triangle pointing towards the center and a semi-circle of the same color, partially overlapping the triangle.

---

# ¿Cómo capturar la información del usuario?

Se utiliza el comando **read** de dos maneras:

- Utilizando en conjunto con el comando echo.
- Utilizando directamente el comando read.



# Como capturar la información del usuario

```
shellCoursePlatzi — vim 7_read.sh — 70x24

# ! /bin/bash
# Programa para ejemplificar como capturar la información del usuario
# utilizando el comando read
# Autor: Marco Toscano Freire - @martosfre

option=0
backupName=""

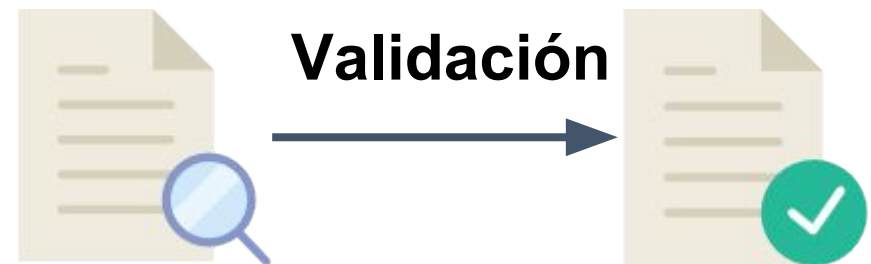
echo "Programa Utilidades Postgres"
read -p "Ingresar una opción:" option
read -p "Ingresar el nombre del archivo del backup:" backupName
echo "Opción:$option , backupName:$backupName"
```



---

# ¿Cómo validar tamaños y tipo de dato en el ingreso de la información?

- Para validar tamaños se utiliza el comando el siguiente comando: `read -n<numero_caracteres>`.
- Para validar el tipo de datos se utilizan expresiones regulares.



# Validar tamaño información

```
shellCoursePlatzi — vim 8_readValidate.sh — 70x24
# ! /bin/bash
# Programa para ejemplificar como capturar la información del usuario
# y validarla
# Autor: Marco Toscano Freire - @martosfre

option=0
backupName=""
clave=""

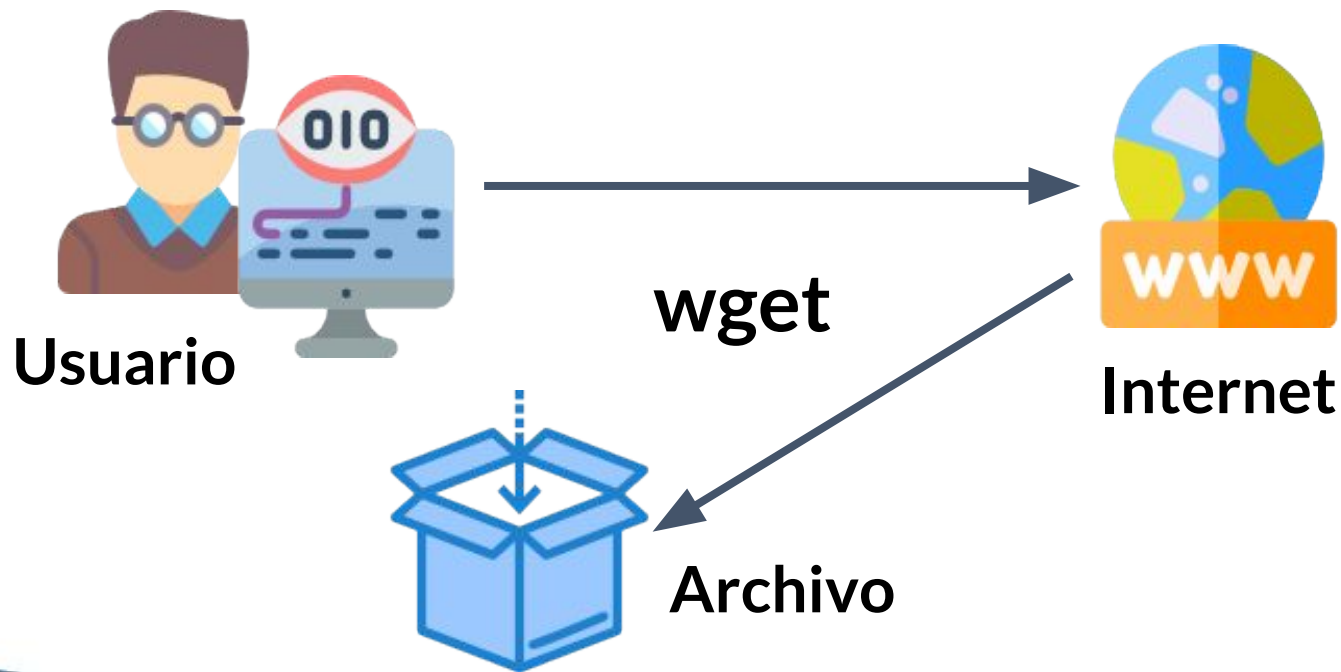
echo "Programa Utilidades Postgres"
# Acepta el ingreso de información de solo un caracter
read -n1 -p "Ingresar una opción:" option
echo -e "\n"
read -n10 -p "Ingresar el nombre del archivo del backup:" backupName
echo -e "\n"
echo "Opción:$option , backupName:$backupName"
read -s -p "Clave:" clave
echo "Clave: $clave"
```

# Envío Opciones / Parámetros

---

- Opciones vs Parámetros
- Envío Independiente
- Envío Complementario
- Leer los valores

# ¿Cómo conectarse a Internet y descargarse un archivo?



# Descargar archivo



```
shellCoursePlatzi — vim 10_download.sh — 71x24
# !/bin/bash
# Programa para ejemplificar el uso de la descarga de información desde
internet utilizando el comando wget
# Autor: Marco Toscano Freire - @martosfre
echo "Descargar información de internet"
wget https://www-us.apache.org/dist/tomcat/tomcat-8/v8.5.35/bin/apache-
tomcat-8.5.35.zip
```

---

# Condicionales



---

# ¿Cómo utilizar las sentencias if, else if, else?

```
if [ condition ]; then  
    statement 1  
elif [ condition ]; then  
    statement 2  
else  
    statement 3  
fi
```

Condition utiliza

- Operadores Lógicos
- Operadores Condicionales

# Sentencia If-Else

```
shellCourse — vim 11_ifElse.sh — 71x24
# !/bin/bash
# Programa para ejemplificar el uso de la sentencia de decisión if, else
# Autor: Marco Toscano Freire - @martosfre

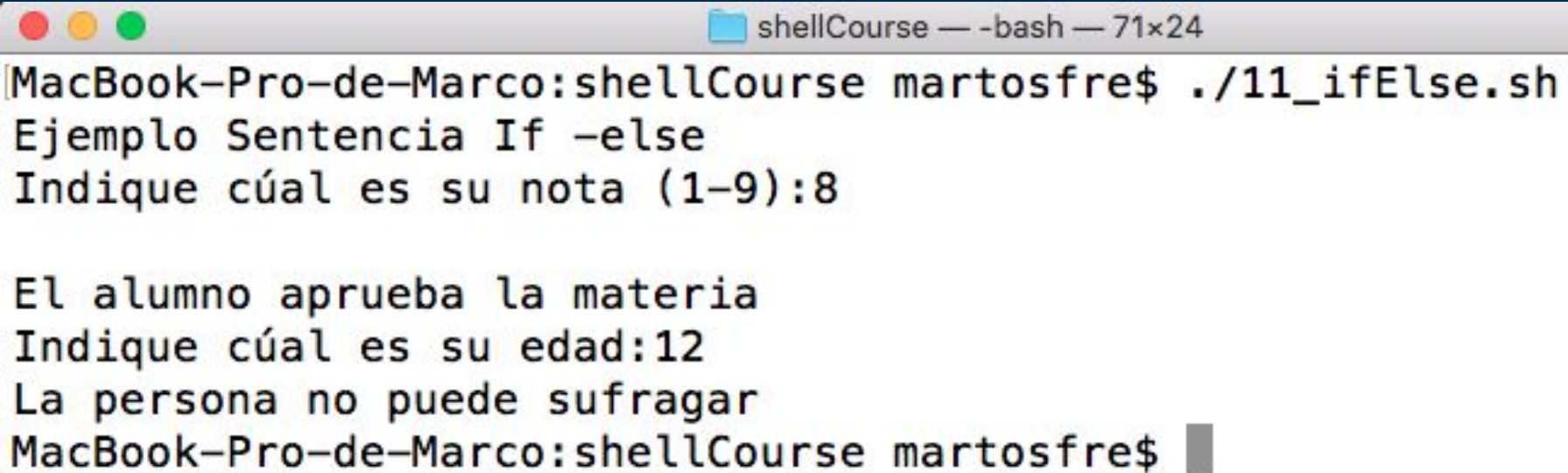
notaClase=0
edad=0

echo "Ejemplo Sentencia If -else"
read -n1 -p "Indique cuál es su nota (1-9):" notaClase
echo -e "\n"
if (( $notaClase >= 7 )); then
    echo "El alumno aprueba la materia"
else
    echo "El alumno reprueba la materia"
fi

read -p "Indique cuál es su edad:" edad
if [ $edad -le 18 ]; then
    echo "La persona no puede sufragar"
else
    echo "La persona puede sufragar"
fi
```



# Sentencia If-Else



A terminal window titled "shellCourse — -bash — 71x24" displays the execution of a script. The prompt is "MacBook-Pro-de-Marco:shellCourse martosfre\$". The script runs and outputs "Ejemplo Sentencia If -else" and "Indique cuál es su nota (1-9):". The user enters "8", and the script outputs "El alumno aprueba la materia" and "Indique cuál es su edad:". The user enters "12", and the script outputs "La persona no puede sufragar". The prompt returns to "MacBook-Pro-de-Marco:shellCourse martosfre\$".

```
MacBook-Pro-de-Marco:shellCourse martosfre$ ./11_ifElse.sh
Ejemplo Sentencia If -else
Indique cuál es su nota (1-9):8

El alumno aprueba la materia
Indique cuál es su edad:12
La persona no puede sufragar
MacBook-Pro-de-Marco:shellCourse martosfre$
```


# Sentencia if-else if - else

```
shellCourse — vim 11_ifElseIfElse.sh — 71x24
# !/bin/bash
# Programa para ejemplificar el uso de la sentencia de decisión if, else
# Autor: Marco Toscano Freire - @martosfre

edad=0

echo "Ejemplo Sentencia If -else"
read -p "Indique cuál es su edad:" edad
if [ $edad -le 18 ]; then
    echo "La persona es adolescente"
elif [ $edad -ge 19 ] && [ $edad -le 64 ]; then
    echo "La persona es adulta"
else
    echo "La persona es adulto mayor"
fi
```

# Sentencia if-else if - else



```
MacBook-Pro-de-Marco:shellCourse martosfre$ ./11_ifElseIfElse.sh
Ejemplo Sentencia If -else
Indique cuál es su edad:24
La persona es adulta
MacBook-Pro-de-Marco:shellCourse martosfre$ ./11_ifElseIfElse.sh
Ejemplo Sentencia If -else
Indique cuál es su edad:18
La persona es adolescente
MacBook-Pro-de-Marco:shellCourse martosfre$ ./11_ifElseIfElse.sh
Ejemplo Sentencia If -else
Indique cuál es su edad:78
La persona es adulto mayor
MacBook-Pro-de-Marco:shellCourse martosfre$
```

---

# ¿Cómo utilizar sentencias if anidadas?

```
if [condition]; then
    if [ condition ]; then
        statement 1
    else
        statement 2
else
    statement 3
fi
```

Condition utiliza

- Operadores Lógicos
- Operadores Condicionales

# If Anidados

```
shellCourse — vim 12_ifAnidados.sh — 80x25
#!/bin/bash
# Programa para ejemplificar el uso de los ifs anidados
# Autor: Marco Toscano Freire - @martosfre

notaClase=0
continua=""

echo "Ejemplo Sentencia If -else"
read -n1 -p "Indique cuál es su nota (1-9):" notaClase
echo -e "\n"
if [ $notaClase -ge 7 ]; then
    echo "El alumno aprueba la materia"
    read -p "Si va continuar estudiando en el siguiente nivel (s/n):" continua
    if [ $continua = "s" ]; then
        echo "Bienvenido al siguiente nivel"
    else
        echo "Gracias por trabajar con nosotros, mucha suerte !!!"
    fi
else
    echo "El alumno reprueba la materia"
fi
```

---

# ¿Cómo construir expresiones condicionales?

- Utilizada en decisión, Iteración.
- Formada por una o más condiciones.
- Condiciones con tipos de datos diferentes:
- Utiliza los operadores relacionales y condicionales.



```
# !/bin/bash
# Programa para ejemplificar el uso de expresiones condicionales
# Autor: Marco Toscano Freire - @martosfre

edad=0
pais=""
pathArchivo=""

read -p "Ingrese su edad:" edad
read -p "Ingrese su país:" pais
read -p "Ingrese el path de su archivo:" pathArchivo

echo -e "\nExpresiones Condicionales con números"
if [ $edad -lt 10 ]; then
    echo "La persona es un niño o niña"
elif [ $edad -ge 10 ] && [ $edad -le 17 ]; then
    echo "La persona se trata de un adolescente"
else
    echo "La persona es mayor de edad"
fi

echo -e "\nExpresiones Condicionales con cadenas"
if [ $pais = "EEUU" ]; then
    echo "La persona es Americana"
elif [ $pais = "Ecuador" ] || [ $pais = "Colombia" ]; then
    echo "La persona es del Sur de América"
else
    echo "Se desconoce la nacionalidad"
fi
```

---

# ¿ Cómo utilizar las sentencia case?

```
case expression in  
opcion1)  
    statements 1;;  
opcion2)  
    statement 2;;  
....  
esac
```

- Mecanismo para evaluar una simple expresión sea entero o cadena.
- Es muy similar a la sentencia switch.
- Puede evaluar rango de caracteres.



# Sentencia Case

```
shellCourse — vim 14_case.sh — 80x31
#!/bin/bash
# Programa para ejemplificar el uso de la sentencia case
# Autor: Marco Toscano Freire - @martosfre

opcion=""

echo "Ejemplo Sentencia Case"
read -p "Ingrese una opción de la A - Z:" opcion
echo -e "\n"

case $opcion in
    "A") echo -e "\nOperación Guardar Archivo";;
    "B") echo "Operación Eliminar Archivo";;
    [C-E]) echo "No esta implementada la operación";;
    "*") "Opción Incorrecta"
esac
```

---

# Sentencias de Iteración

---

# ¿Cómo utilizar arreglos?

- Una variable con varios elementos.
- Para crear un arreglo se debe colocar los elementos dentro de brackets **nombreArreglo = (valor1, valor2...valorN)** o usar **rangos**.
- El índice de un arreglo comienza en cero.
- Para remover los elementos de un arreglo se utiliza el comando **unset nombreArreglo[pos]**

```
# ! /bin/bash
# Programa para ejemplificar el uso de los arreglos
# Autor: Marco Toscano Freire - @martosfre

arregloNumeros=(1 2 3 4 5 6)
arregloCadenas=(Marco, Antonio, Pedro, Susana)
arregloRangos=({A..Z} {10..20})

#Imprimir todos los valores
echo "Arreglo de Números:${arregloNumeros[*]}"
echo "Arreglo de Cadenas:${arregloCadenas[*]}"
echo "Arreglo de Números:${arregloRangos[*]}"

#Imprimir los tamaños de los arreglos
echo "Tamaño Arreglo de Números:${#arregloNumeros[*]}"
echo "Tamaño Arreglo de Cadenas:${#arregloCadenas[*]}"
echo "Tamaño Arreglo de Números:${#arregloRangos[*]}"

#Imprimir la posición 3 del arreglo de números, cadenas de rango
echo "Posición 3 Arreglo de Números:${arregloNumeros[3]}"
echo "Posición 3 Arreglo de Cadenas:${arregloCadenas[3]}"
echo "Posición 3 Arreglo de Rangos:${arregloRangos[3]}"

#Añadir y eliminar valores en un arreglo
arregloNumeros[7]=20
unset arregloNumeros[0]
echo "Arreglo de Números:${arregloNumeros[*]}"
echo "Tamaño arreglo de Números:${#arregloNumeros[*]}"
```

---

# ¿Cómo utilizar la sentencia for loop?

```
for var in item1 item2... itemN
do
    statement1
    statement2
    ....
    statementN
done
```

Se puede iterar **lista de valores** de: números, cadenas, nombre de archivos, argumentos de línea de comandos.

Soporta también el For loop three expression.

# Sentencia Iteración For Loop

```
shellCourse — vim 16_forLoop.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar el uso de la sentencia de iteración for
# Autor: Marco Toscano Freire - @martosfre

arregloNumeros=(1 2 3 4 5 6)

echo "Iterar en la Lista de Números"
for num in ${arregloNumeros[*]}
do
    echo "Número:$num"
done

echo "Iterar en la lista de Cadenas"
for nom in "Marco" "Pedro" "Luis" "Daniela"
do
    echo "Nombres:$nom"
done

echo "Iterar en Archivos"
for fil in *
do
    echo "Nombre archivo:$fil"
done

echo "Iterar utilizando un comando"
for fil in $(ls)
do
    echo "Nombre archivo:$fil"
done
```

---

# ¿Cómo utilizar la sentencia while loop?

```
while [ condition ]  
do  
    statement1  
    statement2  
    ....  
    statementN  
done
```

Itera lista de valores basada en una condición lógica que debe ser evaluada a verdad.



# Sentencia Iteración While Loop

```
shellCourse — vim 17_whileLoop.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar el uso de la sentencia de iteración while
# Autor: Marco Toscano Freire - @martosfre

numero=1

while [ $numero -ne 10 ]
do
    echo "Imprimiendo $numero veces"
    numero=$(( numero + 1 ))
done
```

```
shellCourse — -bash — 80x31
MacBook-Pro-de-Marco:shellCourse martosfre$ ./17_whileLoop.sh
Imprimiendo 1 veces
Imprimiendo 2 veces
Imprimiendo 3 veces
Imprimiendo 4 veces
Imprimiendo 5 veces
Imprimiendo 6 veces
Imprimiendo 7 veces
Imprimiendo 8 veces
Imprimiendo 9 veces
MacBook-Pro-de-Marco:shellCourse martosfre$
```



---

# ¿ Cómo realizar loops anidados?

```
for var in item1 item2... itemN
do
    for var2 in [A..Z]
    do
        statement1
        ....
        statementN
    done
done
```

# Loop Anidados

```
shellCourse — vim 18_loopsAnidados.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar el uso de los loops anidados
# Autor: Marco Toscano Freire - @martosfre

echo "Loops Anidados"
for fil in $(ls)
do
    for nombre in {1..4}
    do
        echo "Nombre archivo:$fil _ $nombre"
    done
done
```

---

# ¿ Cómo utilizar las sentencia **break** y **continue**?

- Utilizar la sentencia **break** para salir de la ejecución de los loops for, while; es decir parar la iteración.
- Utilizar la sentencia **continue** para continuar con la siguiente iteración.

# Sentencias break y continue

```
shellCourse — vim 19_breakContinue.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar el uso de break y continue
# Autor: Marco Toscano Freire - @martosfre

echo "Sentencias break y continue"
for fil in $(ls)
do
    for nombre in {1..4}
    do
        if [ $fil = "10_download.sh" ]; then
            break;
        elif [[ $fil == 4* ]]; then
            continue;
        else
            echo "Nombre archivo:$fil _ $nombre"
        fi
    done
done
```

# Cómo generar un menú de opciones

```
shellCourse — vim 20_menuOpciones.sh — 80x31
#!/bin/bash
# Programa que permite manejar las utilidades de Postres
# Autor: Marco Toscano Freire - @martosfre

opcion=0

while :
do
    #Limpiar la pantalla
    clear
    #Desplegar el menú de opciones
    echo "_____""
    echo "PGUTIL – Programa de Utilidad de Postgres"
    echo "_____""
    echo "                MENÚ PRINCIPAL                "
    echo "_____""
    echo "1. Instalar Postgres"
    echo "2. Desinstalar Postgres"
    echo "3. Sacar un respaldo"
    echo "4. Restar respaldo"
    echo "5. Salir"

    #Leer los datos del usuario – capturar información
    read -n1 -p "Ingrese una opción [1-5]:" opcion

    #Validar la opción ingresada
    case $opcion in
        1)
            echo -e "\nInstalar postgres....."
            sleep 3
```

---

# Archivos

The bottom-left corner of the slide features a series of overlapping geometric shapes. It includes a dark blue triangle, a medium blue square, and a large dark blue semi-circle, all partially visible and layered to create a modern, abstract design.

---

# ¿Cómo se realiza de creación de directorios / archivos

- Para crear directorios se utiliza el comando **mkdir** seguidor del nombre del directorio.
  - mkdir directorioBackup
- Para crear archivos se utiliza el comando **touch** seguido del nombre del archivo.
  - touch respaldo20181207.sql

# ¿Cómo crear directorios/archivos?

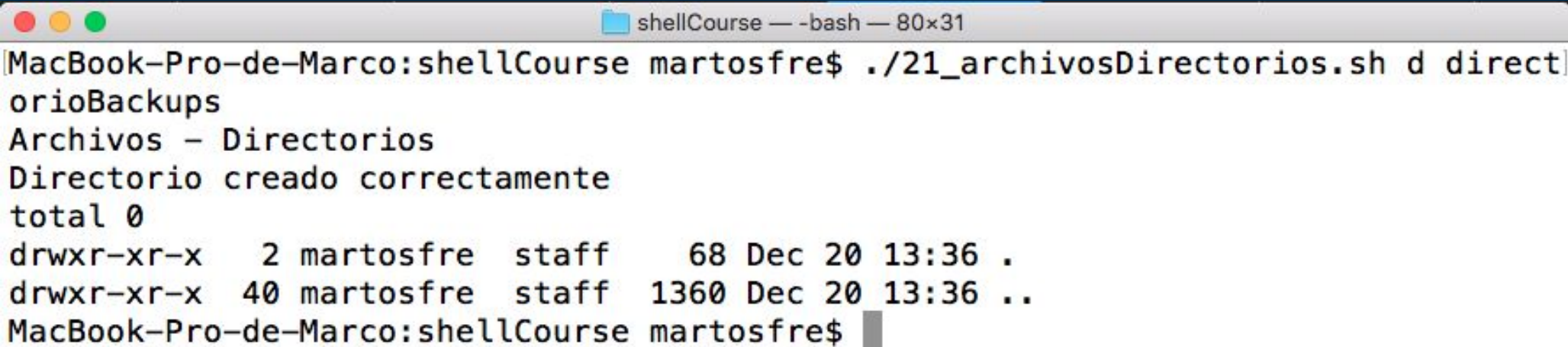
```
shellCourse — vim 21_archivosDirectorios.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar la creación de archivos y directorios
# Autor: Marco Toscano Freire - @martosfre

echo "Archivos - Directorios"

if [ $1 = "d" ]; then
    mkdir -m 755 $2
    echo "Directorio creado correctamente"
    ls -la $2
elif [ $1 == "f" ]; then
    touch $2
    echo "Archivo creado correctamente"
    ls -la $2
else
    echo "No existe esa opción: $1"
fi
```



# ¿Cómo crear directorios/archivos?

A terminal window titled 'shellCourse — -bash — 80x31' on a Mac. The user 'martosfre' runs the command './21\_archivosDirectorios.sh d directorioBackups'. The script outputs 'Archivos - Directorios' and 'Directorio creado correctamente'. It then shows the directory's contents with 'total 0' and a listing of permissions, owner, group, size, date, and names for '.' and '..'.

```
MacBook-Pro-de-Marco:shellCourse martosfre$ ./21_archivosDirectorios.sh d directorioBackups
Archivos - Directorios
Directorio creado correctamente
total 0
drwxr-xr-x  2 martosfre  staff   68 Dec 20 13:36 .
drwxr-xr-x 40 martosfre  staff 1360 Dec 20 13:36 ..
MacBook-Pro-de-Marco:shellCourse martosfre$
```

---

# ¿Cómo se escribe dentro de un archivo?

- Para escribir dentro de un archivo que se encuentra creado o para crear uno con contenido se utiliza tanto el comando **echo** como el comando **cat**.
- Con el comando **cat** se pueden realizar más funcionalidades como por ejemplo crear un archivo a partir de la unión de varios archivos.

# ¿Cómo se escribe en un archivo?

```
shellCourse — vim 22_writeFile.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar como se escribe en un archivo
# Autor: Marco Toscano Freire — @martosfre

echo "Escribir en un archivo"

echo "Valores escritos con el comando echo" >> $1

#Adición multilínea
cat <<EOM >>$1
$2
EOM
```

```
shellCourse — -bash — 80x31
MacBook-Pro-de-Marco:shellCourse martosfre$ ./22_writeFile.sh prueba.txt algo
Escribir en un archivo
MacBook-Pro-de-Marco:shellCourse martosfre$ cat prueba.txt
Valores escritos con el comando echo
algo
MacBook-Pro-de-Marco:shellCourse martosfre$
```

---

# ¿Cómo se lee el contenido de un archivo?

- Para leer el contenido de un archivo de manera general; es decir, todo el contenido se pueden utilizar los comandos **cat**.
- En el caso de que requiere procesar el contenido línea por línea se deberá utilizar una sentencia de iteración como el **while**.

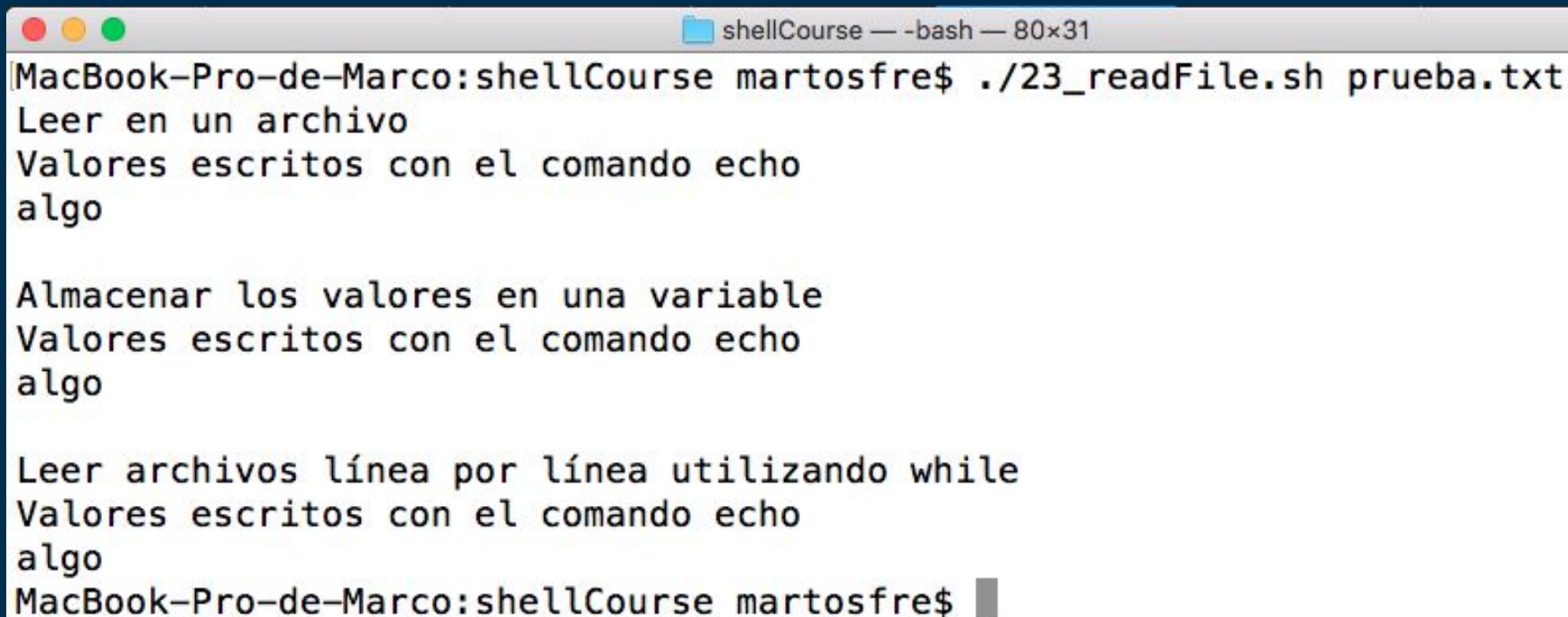
# ¿Cómo se lee el contenido en un archivo?

```
shellCourse — vim 23_readFile.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar como se lee en un archivo
# Autor: Marco Toscano Freire - @martosfre

echo "Leer en un archivo"
cat $1
echo -e "\nAlmacenar los valores en una variable"
valorCat=`cat $1`
echo "$valorCat"

# Se utiliza la variable IFS (Internal Field Separator) para evitar que los espacios en blanco al inicio al final se recortan
echo -e "\nLeer archivos línea por línea utilizando while"
while IFS= read linea
do
    echo "$linea"
done < $1
```

# ¿Cómo se lee el contenido en un archivo?

A terminal window titled 'shellCourse — -bash — 80x31' is shown. It displays the execution of a script './23\_readFile.sh prueba.txt'. The script's output is divided into three sections, each starting with a header line: 'Leer en un archivo', 'Almacenar los valores en una variable', and 'Leer archivos línea por línea utilizando while'. Each section contains two lines of text: 'Valores escritos con el comando echo' and 'algo'. The terminal ends with the prompt 'MacBook-Pro-de-Marco:shellCourse martosfre\$' and a cursor.

```
MacBook-Pro-de-Marco:shellCourse martosfre$ ./23_readFile.sh prueba.txt
Leer en un archivo
Valores escritos con el comando echo
algo

Almacenar los valores en una variable
Valores escritos con el comando echo
algo

Leer archivos línea por línea utilizando while
Valores escritos con el comando echo
algo
MacBook-Pro-de-Marco:shellCourse martosfre$
```

---

# ¿Cómo se realizan las operaciones de copiar, mover y eliminar archivos?

- Una vez que se tenga creado el archivo, se pueden realizar varias operaciones en torno a él como son copiar (**cp**), mover(**mv**) y eliminar archivos (**rm**).
- Generalmente varias de estas operaciones funcionan de manera complementaria.



# Operaciones Archivos

```
shellCourse — vim 24_fileOperations.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar las operaciones de un archivo
# Autor: Marco Toscano Freire - @martosfre

echo "Operaciones de un archivo"
mkdir -m 755 backupScripts

echo -e "\nCopiar los scripts del directorio actual al nuevo directorio backupScripts"
cp *.* backupScripts/
ls -la backupScripts/

echo -e "\nMover el directorio backupScripts a otra ubicación: $HOME"
mv backupScripts $HOME

echo -e "\nEliminar los archivos .txt"
rm *.txt
```



---

# Empaquetamiento

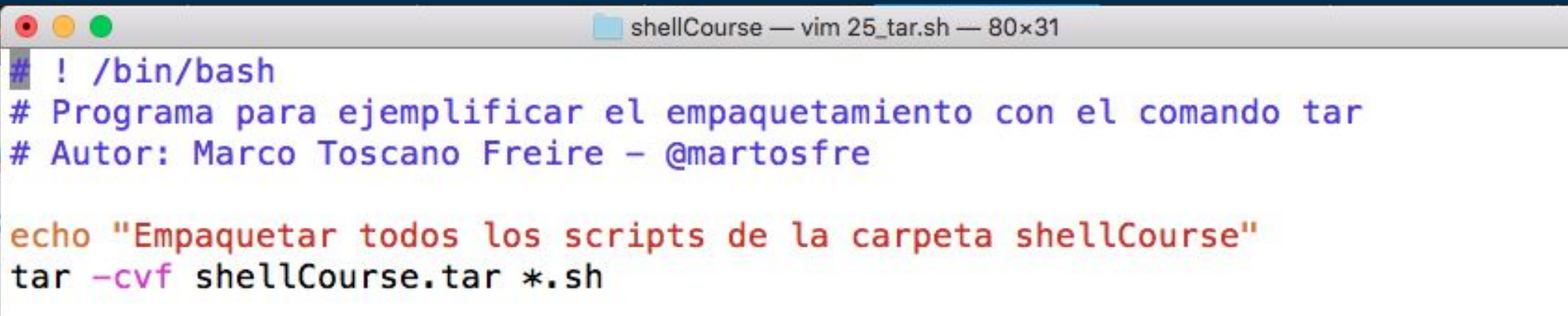
The bottom-left corner of the slide features a decorative design consisting of a dark blue triangle and a semi-circle, both in a darker shade of blue than the background.

---

# ¿Cómo se realiza el empaquetamiento de archivos utilizando el comando tar?

- El comando tar puede ser usado para empaquetar archivos, originalmente diseñado para almacenar datos de archivos de cintas.
- Permite almacenar archivos y directorios en un simple archivo, reteniendo todos los atributos del archivo como propietario, permiso.

# Empaquetamiento con tar



```
shellCourse — vim 25_tar.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar el empaquetamiento con el comando tar
# Autor: Marco Toscano Freire - @martosfre

echo "Empaquetar todos los scripts de la carpeta shellCourse"
tar -cvf shellCourse.tar *.sh
```

---

# ¿ Cómo se realiza el empaquetamiento de archivos utilizando el comando gzip?

- El comando gzip puede ser aplicado solamente para comprimir un archivo simple o flujo de datos; es decir, no puede archivar directorios o múltiples archivos.

---

# ¿ Cómo se realiza el empaquetamiento de archivos utilizando el comando gzip?

- Además permite configurar el ratio de compresión de 1 a 9; siendo 1 la compresión más baja pero más rápida y 9 la compresión mejor pero más lenta.

# Empaquetamiento con gzip

```
shellCourse — vim 26_gzip.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar el empaquetamiento con el comando tar y gzip
# Autor: Marco Toscano Freire - @martosfre

echo "Empaquetar todos los scripts de la carpeta shellCourse"
tar -cvf shellCourse.tar *.sh

# Cuando se empaqueta con gzip el empaquetamiento anterior se elimina
gzip shellCourse.tar

echo "Empaquetar un solo archivo, con un ratio 9"
gzip -9 9_options.sh
```

---

# ¿ Cómo se realiza el empaquetamiento de archivos utilizando el comando pbzip2?

- El comando pbzip2 está diseñado para correr en procesadores multicore, lo cual permite optimizar el tiempo de empaquetamiento de los archivos.

---

# ¿ Cómo se realiza el empaquetamiento de archivos utilizando el comando pbzip2?

- pbzip2 generalmente no viene instalada en la mayoría de distros linux, se tiene que **instalar**.



---

# ¿ Cómo se realiza el empaquetamiento de archivos utilizando el comando pbzip2?

- pbzip2 trabaja solo sobre un simple archivo, para comprimir múltiples archivos o directorios trabaja conjuntamente con tar.

# Empaquetamiento con bzip2

```
shellCourse — vim 27_pbzip2.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar el empaquetamiento con el comando pbzip
# Autor: Marco Toscano Freire - @martosfre

echo "Empaquetar todos los scripts de la carpeta shellCourse"
tar -cvf shellCourse.tar *.sh
pbzip2 -f shellCourse.tar

echo "Empaquetar un directorio con tar y pbzip2"
tar -cf *.sh -c > shellCourseDos.tar.bz2
```

---

# ¿Cómo se realiza un respaldo de información empaquetado y con clave de acceso?

- Utilizaremos la herramienta zip.
- zip generalmente no viene instalada en la mayoría de distros linux, se tiene que **instalarla**.
- Existen otras alternativas como GnuPG, bcrypt, 7zip entre otras.

# Empaquetamiento con zip y con clave



The image shows a terminal window with a title bar that reads "shellCourse — vim 28\_zipPassword.sh — 80x31". The terminal content is as follows:

```
# ! /bin/bash
# Programa para ejemplificar el empaquetamiento con clave utilizando zip
# Autor: Marco Toscano Freire - @martosfre

echo "Empaquetar todos los scripts de la carpeta shellCourse con zip y asignarle
una clave de seguridad"
zip -e shellCourse.zip *.sh
~
```

---

# ¿Cómo transferir información por la red empaquetada?

- Utilizaremos rsync, es un comando que puede ser usado para sincronizar archivos y directorios de una localización a otra mientras minimiza la transferencia de datos.
- Soporta características de compresión, encriptación entre otras.

# ¿Cómo transferir información por la red empaquetada?

```
shellCourse — vim 29_packageSSH.sh — 80x31
# ! /bin/bash
# Programa para ejemplificar la forma de como transferir por la red utilizando el comando rsync, utilizando las opciones de empaquetamiento para optimizar la velocidad de transferencia
# Autor: Marco Toscano Freire - @martosfre

echo "Empaquetar todos los scripts de la carpeta shellCourse y transferirlos por la red a otro equipo utilizando el comando rsync"

host=""
usuario=""

read -p "Ingresar el host:" host
read -p "Ingresar el usuario:" usuario
echo -e "\nEn este momento se procederá a empaquetar la carpeta y transferirla según los datos ingresados"
rsync -avz $(pwd) $usuario@$host:/Users/martosfre/Downloads/platzi
```

---

# Funciones



---

# ¿Qué es una función y cuál es su estructura?

```
function-name () {  
    <codeToExecute>  
}
```

Son bloques de código con funcionalidad específica que existen en memoria y que ayudan a organizar el código en un programa



# Funciones

```
shellCourse — vim 30_functionsArgs.sh — 80x31
# ! /bin/bash
# Programa que permite manejar las utilidades de Postres
# Autor: Marco Toscano Freire - @martosfre

opcion=0

# Función para instalar postgres
instalar_postgres () {
    echo "Instalar postgres..."
}

# Función para desinstalar postgres
desinstalar_postgres () {
    echo "Desinstalar postres..."
}
```

# ¿Cómo se llama a una función?

```
#Validar la opción ingresada
case $opcion in
  1)
    instalar_postgres
    sleep 3
    ;;
  2)
    desinstalar_postgres
    sleep 3
    ;;
```

---

# ¿Cómo realizar el paso de argumentos a una función?

- El paso de argumentos a una función se lo realiza de igual manera que cuando se envía parámetros para ejecutar un script; es decir, en la llamada de la función se envía los n argumentos y luego dentro de la función se los lee a través de su posición.

# ¿Cómo se realiza el paso de argumentos a una función?

```
# Función para sacar un respaldo
sacar_respaldo () {
    echo "Sacar respaldo..."
    echo "Directorio backup: $1"
}

# Función para restaurar un respaldo
restaurar_respaldo () {
    echo "Restaurar respaldo..."
    echo "Directorio respaldo: $1"
}
```

```
3)
    read -p "Directorio Backup:" directorioBackup
    sacar_respaldo $directorioBackup
    sleep 3
    ;;

4)
    read -p "Directorio de Respaldos:" directorioRespaldos
    restaurar_respaldo $directorioRespaldos
    sleep 3
    ;;
```

---

# ¿Cómo ejecutar una función en segundo plano?

- Para ejecutar una función en segundo plano se utiliza el operador **&** al final del comando. Lo cual pone al comando a ejecutarse en background y libera al terminal.
- Es importante conocer cuál es el proceso asociado a la ejecución de la función, en el caso de que se requiera terminarlo.

# Gracias

