3️⃣

Practica 3

1. ¿Qué es el Shell Scripting? ¿A qué tipos de tareas están orientados los script? ¿Los scripts deben compilarse? ¿Por qué?

El shell scripting hace referencia a la programación de scripts utilizando estructuras de control proporcionadas por las shells (interprete de comandos)

- **Script:** serie de comandos escritos en un archivo
- El lenguaje utilizado (el lenguaje es la sintaxis de Bash) incluye estructuras de control, redirecciones, pipes y variables

Los scripts estan orientados a la automatización de tareas, pero otras de sus funciones serian

- Creacion de aplicaciones interactivas
- Creacion de aplicaciones con interfaz grafica, por ejeplo usando el comando zenity
- Manejar archivos
- Manejar las salidas de procesos

Un script no debe compilarse, debido a que la shell actua como interprete, osea que se ejecutan directamente procesando las las instrucciones linea a linea

2. Investigar la funcionalidad de los comandos echo y read.
a. ¿Cómo se indican los comentarios dentro de un script?
b. ¿Cómo se declaran y se hace referencia a variables dentro de un script?

Comando	Funcionalidad	Ejemplo

echo	Imprime texto en la salida estandar (stdout)	echo “hola mundo”
read	Lee una lnea desde la entrada estandar (stdin) y gurda la info en una variable	read nombre
Ambos juntos	——	#!/bin/bash echo "ingresa tu nombre:” read nombre echo "Hola, $nombre”

Los comentarios dentro de un script se ponen con un #, aunque la primera linea tiene un # este indica la ruta del shell (#!/bin/bash)

Para declarar variables se usa el tag de la variable seguido de un =, tipo NOMBRE=”Juan”

- No se usan espacio entre el = porque se pudre todo

Para acceder al contenido se usa $, tipo echo $NOMBRE

Se pueden usar llaves para meter variables en texto, tipo asi

- echo ${NOMBRE}esto_no_es_parte_de_la_variable

3. Crear dentro del directorio personal del usuario logueado un directorio llamado practica-shell-script y dentro de él un archivo llamado mostrar.sh cuyo contenido sea el siguiente:

- Para darle permisos es `chmod +x mostrar.sh`
- Para ejecutarlo es `./mostrar.sh`

```
#!/bin/bash
echo "Introduzca su nombre y apellido:"
read nombre apellido
echo "Fecha y hora actual:"
date
echo "Su apellido y nombre es:
echo "$apellido $nombre"
echo "Su usuario es: whoami"
echo "Su directorio actual es:"
```

4. Parametrización: ¿Cómo se acceden a los parámetros enviados al script al momento de su invocación? ¿Qué información contienen las variables $#, $*, $? y $HOME dentro de un script?

Los scripts pueden recibir argumentos en su ejecucion

- $0 Contiene el nombre o la ruta donde se ejecuto el script
- $1, $2, $3 contiene los argumentos que le pasaron al script en ese orden
    - Ejemplo: ./mostrar.sh alvaro aufmuth, echo “Nombre: $1, Apellido: $2”

Variable	Funcionalidad
$#	Contiene la cantidad de argumentos recibidos, se usa para saber si se pasaron los args necesarios
$*	Es la lista de todos los argumentos que se pasaron
$?	Contiene en todo momento el valor de retorno del utlimo comando ejecutado o funcion
$HOME	Es una variable de entorno que ahce ref al directorio home del usuario

5. ¿Cual es la funcionalidad de comando exit? ¿Qué valores recibe como parámetro y cuál es su significado?

El `exit` se usa para terminar un script, devuelve un valor que va entre 0 y 255

- Retorna 0: El script termino de forma exitosa
- Retorna entre 1 y 255: El script termino mal y el numero es el codigo de error

Despues que un script termina, se puede usar #? para imprimir o usar el valor de retorno del exit

Ejemplo de uso

```
if [ $# -ne 2 ]; then
  exit 1 # Error
else
```

```
  # ... Bloque de código si funciona ...
fi
exit 0 # Funcionó correctamente
```

Esto hace que si se recibieron los parametros que esperaba

6. El comando expr permite la evaluación de expresiones. Su sintaxis es:
`expr arg1 op arg2` ,
donde arg1 y arg2 representan argumentos y op la operación de la expresión.
Investigar que tipo de operaciones se pueden utilizar.

Permite comparar expresiones, tiene 3 partes

- Operaciones aritmeticas: solo numeros enteros
  - Suma: expr 5 + 3
  - Resta: expr 10 - 2
  - Multiplicacion: expr 4 * 5
  - Division: expr 10 / 3
  - Modulo: expr 10 % 3
- Comaparciones logicas: devuelven 1 si es verdadero y 0 si es falso
  - Igual: expr 5 = 5
  - Distinto: expr 5 ! = 6
  - Mayor/Menor: expr 5 \> 3
  - AND: expr 1 \& 0
  - OR: expr 1 | 0
- Manipulaciones de cadenas: expresiones de texto
  - length: expr length “hola” (devuelve 4)
  - Extraer: expr substr “abcd” 2 2 (queda “bc”, empieza en la pos 2 y extrae 2 caraceteres)
  - Indice: expr index “banana” “a” (devuelve 2)

- Comparacion: expr “hola” : “ho” (devuelve 2)

7. El comando test expresion permite evaluar expresiones y generar un valor de retorno, true o false. Este comando puede ser reemplazado por el uso de corchetes de la siguiente manera [expresión]. Investigar qué tipo de expresiones pueden ser usadas con el comando test. Tenga en cuenta operaciones para: evaluación de archivos, evaluación de cadenas de caracteres y evaluaciones numéricas.

Operaciones de strings

Operador	Ejemplo
=	“$nombre” = “Maria”
! =	“$nombre” ! = “Maria”
> o ≥	A > Z
< o ≤	A ≤ Z

Operaciones numericas

Operador	Significado	Ejemplo
`-eq`	Igualdad	$edad -eq 20
`-ne`	Desigualdad	$edad -ne 20
`-gt`	Mayor	5 -gt 20
`-ge`	Mayor o igual	5 -ge 20
`-lt`	Menor	5 -lt 20
`-le`	Menor o igual	5 -le 20

8. Estructuras de control. Investigue la sintaxis de las siguientes estructuras de control incluidas en shell scripting:

IF

Sintaxis:

```
if [ condicion ]
then
  bloque
elif [ condicion ]
then
  bloque
else
  bloque
fi
```

CASE

Sintaxis:

```
case $variable in
"valor 1")
  bloque
  ;;
"valor 2")
  bloque
  ;;
*) # default
  bloque
  ;;
esac
```

WHILE

Sintaxis:

```
while [ condicion ]
do
  bloque
done
```

FOR

Sintaxis del normal:

```
for ((i=0; i < 10; i++))
do
  bloque
done
```

Sintaxis del foreach:

```
for i in valor1 valor2 valor3 valorN
do
  bloque
done
```

SELECT

Se usa para hacer como un menu de opciones, la sintaxis es:

```
select variable in opcion1 opcion2 opcion3
do
  bloque
done
```

9. ¿Qué acciones realizan las sentencias break y continue dentro de un bucle? ¿Qué parámetros reciben?

La sentencia break se usa apra cortar un loop y sigue la ejecucion con el resto de cosas fuera del loop

- Puede recibir un parametro n que indica de cuantos loops hay que salir, si tenes un while adentro de un for y pones `break 2` se va a romper el while y el for

La sentencia continue omite el resto del codigo del loop para pasar a la siguiente iteracion directamente

- Tambien se le puede pasar un parametro n que hace que se salte a la siguiente instrccion del loop que pongas, como arriba, si pones `continue 2` en el mismo contexto, saltaria a la siguiente iteracion del for

10. ¿Qué tipo de variables existen? ¿Es shell script fuertemente tipado? ¿Se pueden definir arreglos? ¿Cómo?

En bash los tipos de variables que hay son arreglos y strings, y bueno los numeros se pueden interpretar, por eso se usa -eq para comparar y todo eso

Es fuertemente tipado pero con la excpecion de que los numeros son interpretaciones del mismo shell, entonces se puede hacer nombre=alvaro y despues hacer nombre=5 por la propia naturaleza de como se hace para usar numeros

Los arreglos puede definirse vacios o ya inicializados, asi:

```
arreglo_vacio=()

arreglo_inicializado=(1 21 3 43 12 13)
```

Despues se puede acceder a una posicion especifica para imprimir o asignar

```
arreglo[2]=spam #Se asigna spam en la tercera pos pq arranca de 0
echo ${arreglo[2]}
echo ${arreglo[@]} # Esto imprime todos los elementos
echo ${arreglo[*]} # Esto tmb
```

Y despues hay otras operaciones

```
echo ${#arreglo[@]} # Devuelve la cant de elementos
unset arreglo[2] # Borra la 3ra posicion, no se reacomoda sino que queda l
a pos 2 vacia,
                # aunque el tamaño se disminuye
```

11. ¿Pueden definirse funciones dentro de un script? ¿Cómo? ¿Cómo se maneja el pasaje de parámetros de una función a la otra?

Formas de definir una funcion

- usando la palabra function
- usando parentesis ( )

Las funciones retornan un valor usando la sentencia return

- El valor de retorno debe estar entre 0 y 255
- El valor de retorno puede chequearse usando la variable $?

Pasaje de parametros

- Los parametros se pasan a la funcion y se acceden como los argumentos, usando $1, $2, $3
- La variable $* contiene la lista de todos los argumentos

Ejemplo

```
mayor() {
    echo "Los parametros q pasaste son: $*"
    if [$1 -gt $2]; then
        echo "$1 es mas grande B)"
        return 1
    fi
    echo "$2 es mas grande"
    return 0
}
```