

Practica 3



[Siguiete](#)
[Anterior](#)



- 1) ¿Qué es el Shell Scripting? ¿A qué tipos de tareas están orientados los script?
- 2) Investigar la funcionalidad de los comandos echo y read
- 3) Crear dentro del directorio personal del usuario logueado un directorio
- 4) Parametrización
- 5) ¿Cual es la funcionalidad de comando exit?
- 6) El comando expr permite la evaluación de expresiones
- 7) El comando "test expresión" permite evaluar expresiones
- 8) Estructuras de control
- 9) ¿Qué acciones realizan las sentencias break y continue dentro de un bucle?
- 10) ¿Qué tipo de variables existen? ¿Es shell script fuertemente tipado?

- 11) ¿Pueden definirse funciones dentro de un script? ¿Cómo?
- 12) Evaluación de expresiones
- 13) Uso de las estructuras de control
- 14) Renombrando Archivos: haga un script que renombre solo archivos
- 15) Comando cut
- 16) Realizar un script que reciba como parámetro una extensión
- 17) Escribir un script que al ejecutarse imprima en pantalla los nombre
- 18) Crear un script que verifique cada 10 segundos si un usuario se ha
- 19) Escribir un Programa de "Menu de Comandos Amigable con el Usuario"
- 20) Realice un script que simule el comportamiento de una estructura de PILA
- 21) Dentro del mismo script y utilizando las funciones implementadas
- 22) Dada la siguiente declaración al comienzo de un script
- 23) Implemente un script que recorra un arreglo compuesto por números
- 24) Dada la definición de 2 vectores del mismo tamaño y cuyas
- 25) Realice un script que agregue en un arreglo todos los nombres
- 26) Escriba un script que reciba una cantidad desconocida de parámetros
- 27) Realice un script que implemente a través de la utilización
- 28) Realice un script que reciba como parámetro el nombre de un directorio
- 29) Implemente un script que agregue a un arreglo todos los archivos
- 30) Realice un script que mueva todos los programas del directorio actual



1) Ejercicio

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

La Shell provee estructuras de control que permiten programar shell scripts.

El Shell Scripting es la técnica (habilidad / destreza) de diseñar y crear Script (archivo de automatización de tareas) mediante un Shell (preferiblemente) de un Sistema Operativo, o un

Editor de Texto (Gráfico o Terminal). Este es un tipo de lenguaje de programación que generalmente es interpretado.

Está orientado a diferentes tareas:

- Automatización de tareas
- Aplicaciones interactivas
- Aplicaciones con interfaz gráfica (con el comando zenity, por ejemplo).



2) Ejercicio

Investigar la funcionalidad de los comandos echo y read

- Echo: Es un comando para la impresión de un texto en pantalla. El comando para imprimir el contenido de la variable es:
- echo \$nombreVar

- Read: El comando read lee su entrada estándar y asigna las palabras leídas en la(s) variable(s) cuyo nombre se pasa como argumento. Se trata del input desde teclado.

(a) ¿Cómo se indican los comentarios dentro de un script?

Los comentarios tienen que comenzar con el símbolo #

(b) ¿Cómo se declaran y se hace referencia a variables dentro de un script?

No necesita declarar una variable. Se creará sólo con asignarle un valor a su referencia. Luego se referencia con el símbolo \$ Ejemplo

Para hacer un script:

- Paso 1 en la consola usar el comando: `touch script.sh` para crear el directorio.
- Paso 2 Ver si se creó el directorio con el comando `ls`
- Paso 3 Ingresar al archivo `script.sh` con el editor de texto vi de la siguiente manera
 - `vim script.sh`
 - DENTRO DEL EDITOR VIM:
 - Ingresar la letra i para estar en modo insertar y Escribir

```
#!/bin/bash
CAD="¡Hola Mundo!"
echo $CAD
```

- ESC (para salir del modo editor)
 - `:wq` (para guardar los cambios)
 - Salimos del editor vi e ingresamos a la consola
- Paso 4: Damos permiso de ejecución con el comando:
 - `chmod u+x script.sh`
 - Si ponemos el comando `ls` vamos a ver que el directorio `.sh` cambió de color.
- Paso 5: Con el comando `./script.sh` vamos a ver el contenido de ese script en consola.
- Paso 6: Si queremos borrar ese directorio con script solo es necesario poner en consola el comando `rm script.sh`

También puedes ejecutar el script con el comando `bash script.sh`



3) Ejercicio

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

```
#!/bin/bash
# Comentarios acerca de lo que hace el script
# Siempre comento mis scripts, si no hoy lo hago
# y mañana ya no me acuerdo de lo que quise hacer
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:"
```

```
fa@FaboFacultad:~/Desktop/Practica3-2023/practicashell-script$ bash mostrar.sh  
Introduzca su nombre y apellido:  
Fabian Martinez Rincon  
Fecha y hora actual:  
Sun 16 Apr 2023 11:16:32 PM EDT  
Su apellido y nombre es:  
Martinez Rincon Fabian  
Su usuario es: fa  
Su directorio actual es:  
fa@FaboFacultad:~/Desktop/Practica3-2023/practicashell-script$
```

- (a) Asignar al archivo creado los permisos necesarios de manera que pueda ejecutarlo
- (b) Ejecutar el archivo creado de la siguiente manera: ./mostrar
- (c) ¿Qué resultado visualiza?

```
fa@FaboFacultad:~/Desktop/Practica3-2023/practicashell-script$ chmod u+x mostrar.sh  
fa@FaboFacultad:~/Desktop/Practica3-2023/practicashell-script$ ./mostrar.sh  
Introduzca su nombre y apellido:  
Fabian Martinez  
Fecha y hora actual:  
Sun 16 Apr 2023 11:20:54 PM EDT  
Su apellido y nombre es:  
Martinez Fabian  
Su usuario es: fa  
Su directorio actual es:  
fa@FaboFacultad:~/Desktop/Practica3-2023/practicashell-script$
```

- (d) Las backquotes (`) entre el comando whoami ilustran el uso de la sustitución de comandos. ¿Qué significa esto?

Son para poder ejecutar comando cuando se encuentran dentro de un string

- (e) Realizar modificaciones al script anteriormente creado de manera de poder mostrar distintos resultados (cuál es su directorio personal, el contenido de un directorio en particular, el espacio libre en disco, etc.). Pida que se introduzcan por teclado (entrada estándar) otros datos.

```

3e.sh
1 #!/bin/bash
2 echo "Nombre y apellido"
3 read nombre apellido
4 echo "Fecha y hora actual: `date`"
5 echo $nombre $apellido
6 echo "Su usuario es: `whoami`"
7 echo "su directorio actual es :`pwd`"
8 echo "su directorio personal es : `$HOME`"
9 echo "Nombre de un directorio:"
10 echo contenido
11 echo "Contenido de $contenido es `ls $contenido`"
12 echo "Espacio libre en el disco `df`"

```

```
fa@FaboFacultad:~/Desktop/Practica3-2023$ ./3e.sh
```

```

Nombre y apellido
Fabian Martinez
Fecha y hora actual: Sun 16 Apr 2023 11:39:30 PM EDT
Fabian Martinez
Su usuario es: fa
su directorio actual es :/home/fa/Desktop/Practica3-2023
./3e.sh: line 8: /home/fa: Is a directory
su directorio personal es :
Nombre de un directorio:
contenido
Contenido de  es 1.sh
3e.sh

```

```
practicashell-script
```

Espacio libre en el disco	Filesystem	1K-blocks	Used	Available	Use%	Mounted on
udev	1496092 0	1496092	0%			/dev
tmpfs	303944 1188	302756	1%			/run
/dev/sda1	14389128 8759044	4877344	65%			/
tmpfs	1519708 0	1519708	0%			/dev/shm
tmpfs	5120 4	5116	1%			/run/lock
/dev/loop2	74752 74752	0	100%			/snap/core22/607
/dev/loop3	74752 74752	0	100%			/snap/core22/583
/dev/loop4	56960 56960	0	100%			/snap/core18/2714
/dev/loop5	56960 56960	0	100%			/snap/core18/2721
/dev/loop1	21248 21248	0	100%			/snap/nvim/2797
/dev/loop0	51072 51072	0	100%			/snap/snapd/18596
/dev/loop6	19968 19968	0	100%			/snap/nvim/2789
tmpfs	303940 144	303796	1%			/run/user/1000

```
fa@FaboFacultad:~/Desktop/Practica3-2023$
```



4) Ejercicio

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 invocación. Para accederlos, se utilizan variables especiales:
- `$0` contiene la invocación al script. (nombre del script)
- `$1`, `$2`, `$3`, ... contienen cada uno de los argumentos.
- `$#` contiene la cantidad de argumentos recibidos.
- `$*` contiene la lista de todos los argumentos separados por espacios.

- \$? contiene en todo momento el valor de retorno del ultimo comando ejecutado.(si el valor es 0 se ejecuto correctamente, en caso de que haya habido algún error dará un valor entre 1 y 255).

```
+ 4.sh
1 #!/bin/bash
2
3
4 echo "$0"
5 echo "$1 $3"
6 echo "$#"
7 echo "$*"
8 echo "$HOME"
```

```
fa@FaboFacultad:~/Desktop/Practica3-2023$ bash 4.sh 10 20 30
4.sh
10 30
3
10 20 30
/home/fa
fa@FaboFacultad:~/Desktop/Practica3-2023$
```



5) Ejercicio

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

El comando exit finaliza un script. Incluso, puede arrojar un valor de terminación, el cual estará disponible para su proceso padre.

Todo comando posee un estado de terminación (exit status). Un comando que se ejecutó satisfactoriamente arrojará un valor de 0, mientras que un comando que no se procesó de manera correcta, finalizará con un número diferente de 0.

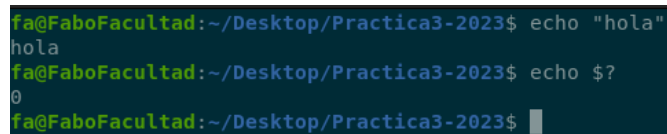
Al igual que un comando, un script posee su propio estado de terminación, el cual será determinado por la última instrucción ejecutada por el script.

Dentro de un script, el comando exit puede arrojar un número dentro de un rango de 0 a 255.

Se puede consultar el exit status imprimiendo la variable \$? de la siguiente manera:

- Ingresamos al editor VIM con el archivo mostrar.sh
- Una vez hecho el bin bash, como última línea y dentro de insertar ingresamos el siguiente comando: echo \$?
- ESC
- :exit

En la consola se mostrara un número de 0 a 255.



```
fa@FaboFacultad:~/Desktop/Practica3-2023$ echo "hola"
hola
fa@FaboFacultad:~/Desktop/Practica3-2023$ echo $?
0
fa@FaboFacultad:~/Desktop/Practica3-2023$
```



6) Ejercicio

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

- Se puede utilizar
 - suma
 - resta
 - multiplicación (/*)
 - división (/).
- Y devuelve solo la parte entera.:

- `expr 10 + 10 -> 20`, `expr 14 % 9 -> 5`.

También se puede utilizar para:

- Cálculo de la longitud de la cadena:
 - `expr length "esto es un test" >`
 - devuelve la cantidad de caracteres: 15
- Cadena de rastreo:
 - `expr substr "esto es un test" 3 5 >`
 - te devuelve el texto desde la pos 3 y 5 lugares: to es
- La cadena numérica primera posición de rastreo personaje que aparece:
 - `expr index "esto es un test" s`
 - te devuelve el índice de la letra pasado por parámetro en la cadena (la primera aparición).

```
6.sh
1 #!/bin/bash/
2
3 SUMA=`expr 10 + 5`
4 RESTA=$(expr 10 - 5)
5 MULTIPLICACION=$(expr 10 \* 5)
6 DIVISION=$(expr 10 / 5)
7 MODULO=$(expr 10 % 5)
8 LONGITUD="Fabian" `$(expr length "Fabian")`
9 SUBSTR="Fabian 1 6: " `$(expr substr "Fabian" 1 6)`
10 INDICE="Fabian b: " `$(expr index "Fabian" b)`
11
12
13 echo "var1 = 10, var2 = 5"
14 echo "SUMA $SUMA"
15 echo "RESTA $RESTA"
16 echo "MULTIPLICACION $MULTIPLICACION"
17 echo "DIVISION $DIVISION"
18 echo "MODULO $MODULO"
19 echo "LONGITUD $LONGITUD"
20 echo "SUBSTR $SUBSTR"
21 echo "INDICE $INDICE"
```

```
var1 = 10, var2 = 5
SUMA 15
RESTA 5
MULTIPLICACION 50
DIVISION 2
MODULO 0
LONGITUD Fabian 6
SUBSTR Fabian 1 6: Fabian
INDICE Fabian b: 3
```



7) Ejercicio

El comando “**test expresión**” 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 que 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.

Este comando tiene un código de retorno igual a 0 cuando el test es positivo, y diferente de cero en caso contrario

El comando test posee dos sintaxis: **test expresión** y **[expresión]**, donde **expresión** representa el test que se debe efectuar

```
7.sh
1  !/bin/bash
2  var1=10
3  var2=5
4
5  if [ -e 7.sh ]; then echo "existe"; else echo "no existe"; fi
6
7  if (( $var1 % $var2 == 0 )); then echo "Es Par"; else echo "Es Impar"; fi
8
9  if (( $var1 > $var2 )); then echo "Es Mayor"; else echo "Es Menor"; fi
10
11 if [ -r 7.sh -a -w 7.sh -a -x 7.sh ]; then echo "Todos los permisos"
12 else echo "no tiene todos los permisos"; fi
13
14 if [ -r 7.sh -o -w 7.sh -o -x 7.sh ]; then echo "algun permiso"
15 else echo "no tiene todos los permisos"; fi
fa@FaboFacultad:~/Desktop/Practica3-2023$ nvim 7.sh
fa@FaboFacultad:~/Desktop/Practica3-2023$ bash 7.sh
existe
Es Par
Es Mayor
no tiene todos los permisos
algun permiso
fa@FaboFacultad:~/Desktop/Practica3-2023$
```

Valores a tener en cuenta

Parametro	Valor
-r	Si es de Lectura
-w	Si es de escritura
-x	Si es de ejecución
-a	and
-o	or
-e	Si existe
-d	Si es un directorio
-f	Si es un archivo
-n	No es

	Operador	Con Strings	Con números	Operaciones
cadena vacia	Igualdad	[\$nombre = "Maria"]	[\$edad -eq 20]	\$((2 == 2))
	Desigualdad	[\$nombre != "Maria"]	[\$edad -ne 20]	\$((3 != 4))
	Mayor	[A > Z]	[5 -gt 20]	\$((3 > 1))
	Mayor o Igual	[A >= Z]	[5 -ge 20]	\$(2 >= 1)
	Menor	[A < Z]	[5 -lt 20]	\$((1 < 10))
	Menor o Igual	[A <= Z]	[5 -le 20]	\$((3 <= 5))



8) Ejercicio

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

if	Resultado
<pre>20 if [3 -eq 3]; then 21 echo "Son iguales" 22 else 23 echo "Son distintos" 24 fi</pre>	<pre>Son iguales</pre>

case	Resultado
<pre>8.sh 1 #!/bin/bash 2 3 echo "Ingrese un caracter" 4 read letra 5 6 case \$letra in 7 [a-z]) echo "Es una minuscula";; 8 [A-Z]) echo "Es una mayuscula";; 9 [0-9]) echo "Es un nro";; 10 *) echo "Ingreso un caracter invalida";; 11 esa</pre>	<pre>Ingrese un caracter F Es una mayuscula</pre>
while	Resultado
<pre>13 var1=0 14 while [\$var1 -le 5] 15 do 16 echo "Hola mundo" 17 var1=\$((var1+1)) 18 done</pre>	<pre>Hola mundo Hola mundo Hola mundo Hola mundo Hola mundo Hola mundo fa@FaboFacultad:~/Desktop/Practica3-2023\$</pre>
for	Resultado
<pre>26 for var in 1 2 3 4 5 27 do 28 echo \$var 29 done 30 31 echo "-----" 32 33 for var2 in {1..5} 34 do 35 echo \$var2 36 done 37 38 echo "-----" 39 40 for ((i=1; i<=5; i++)) 41 do 42 echo \$i 43 done</pre>	<pre>1 2 3 4 5 ----- 1 2 3 4 5 ----- 1 2 3 4 5 fa@FaboFacultad:~/Desktop/Practica3-2023\$</pre>
select	resultado

```
45 select opcion in opcion1 opcion2
46 do
47
48     case $opcion in
49         opcion1) echo "Opcion1" ;;
50         opcion2) echo "Opcion2";;
51         *) exit;;
52     esac
53 done
```

```
1) opcion1
2) opcion2
#? 1
Opcion1
#? 2
Opcion2
#? e
```

9) Ejercicio

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

- **break** [n] Corta la ejecución de n niveles de loops.
- **continue** [n] salta a la siguiente iteración del enésimo loop que contiene esta instrucción

```
9.sh
1 #!/bin/bash/
2
3 contador=0
4 while true
5 do
6     let contador++
7     if [ $contador -eq 10 ]; then
8         break
9     elif [ $contador -eq 5 ]; then
10        continue
11    fi
12    echo $contador
13 done
```

```
fa@FaboFacultad:~/Desktop/Practica3-2023$ bash 9.sh
1
2
3
4
5
6
7
8
9
fa@FaboFacultad:~/Desktop/Practica3-2023$
```



10) Ejercicio

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

bash soporta strings y arrays

- Los nombres son case sensitive
- Para crear una variable:
 - `NOMBRE="pepe" # SIN espacios alrededor del =`
- Para accederla se usa \$:
 - `echo $NOMBRE`

Para evitar ambigüedades se pueden usar llaves:

- Esto no accede a \$NOMBRE
- echo \$NOMBREesto_no_es_parte_de_la_variable
- Esto sí
- echo \${NOMBRE}esto_no_es_parte_de_la_variable Los nombres de las variables pueden contener mayúsculas, minúsculas, números y el símbolo (underscore), pero no pueden empezar con un número.

No es fuertemente tipado, y no requiere una declaración explícita de tipo variables.

Todo sobre los arreglos

Bash	Resultado
arreglo=()	Definir Arreglo vacío
arreglo[0]="valor1"	Asignación
arreglo=("valor1", "valor2")	Definir con valores
read -a arreglo	leer desde teclado tiene que estar definido arriba
\${arreglo[0]}	retorna "valor1"
\${arreglo[*]}	Retorna todos los elementos del array
\${arreglo[@]}	Retorna todos los elementos del array
\${#arreglo[@]}	Retorna 2, si contiene valor1 y valor2

```
10.sh
1 #!/bin/bash
2
3 array=()
4 read -a array
5 echo "Array pelado" $array
6 echo "index 3" ${array[3]}
7 echo "Todos los elementos" ${array[*]}
8 echo "Cantidad de elementos" ${#array[*]}
```

```
14 23 123 5
Array pelado 14
index 3 5
Todos los elementos 14 23 123 5
Cantidad de elementos 4
```




11) Ejercicio

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

Si, se pueden definir funciones dentro de un script.

- Las funciones permiten modularizar el comportamiento de los scripts.
- Se pueden declarar de 2 formas:

```
function nombre { block }  
nombre() { block }
```

- Con la sentencia return se retorna un valor entre 0 y 255
- El valor de retorno se puede evaluar mediante la variable \$?
- Reciben argumentos en las variables \$1, \$2, etc

Para definir variables locales podemos usar la palabra clave local var1

```
11.sh
1 #!/bin/bash
2
3 sumar(){
4     return $(( $1 + $2 ))
5
6 }
7 cadena(){
8     echo "Hola $1, como estas"
9 }
10
11 sumar 2 3
12 echo $?
13
14 mensaje=$(cadena "Juan")
15 echo $mensaje
```

```
5
Hola Juan, como estas
```



12) Ejercicio

Evaluación de expresiones

(a) Realizar un script que le solicite al usuario 2 números, los lea de la entrada Standard e imprima la multiplicación, suma, resta y cual es el mayor de los números leídos.

—

```

1 #!/bin/bash
2 |
3 echo "Ingrese dos nros"
4 read nro1 nro2
5 echo "MULTIPLICACION: $(( $nro1 * $nro2 ))"
6 echo "SUMA: $(( $nro1 + $nro2 ))"
7 echo "RESTA: $(( $nro1 - $nro2 ))"
8 if [ $nro1 -gt $nro2 ]; then
9     echo "es mayor el $nro1"
10 else
11     echo "es mayor el $nro2"
12 fi

```

```

Ingrese dos nros
3 6
MULTIPLICACION: 18
SUMA: 9
RESTA: -3
es mayor el 6

```

(b) Modificar el script creado en el inciso anterior para que los números sean recibidos como parámetros. El script debe controlar que los dos parámetros sean enviados.

```

12b.sh
1 |
2 #!/bin/bash
3 if (( $# != 2 )); then exit 0; fi
4
5 echo "MULTIPLICACION: $(( $1 * $2 ))"
6 echo "SUMA: $(( $1 + $2 ))"
7 echo "RESTA: $(( $1 - $2 ))"
8 if [ $1 -gt $2 ]; then
9     echo "es mayor el $1"
10 else
11     echo "es mayor el $2"
12 fi

```

```

fa@FaboFacultad:~/Desktop/Practica3-2023$ bash 12b.sh 2 5
MULTIPLICACION: 10
SUMA: 7
RESTA: -3
es mayor el 5
fa@FaboFacultad:~/Desktop/Practica3-2023$

```

(c) Realizar una calculadora que ejecute las 4 operaciones básicas: +, -, *, %. Esta calculadora debe funcionar recibiendo la operación y los números como parámetros

```

12c.sh
1 #!/bin/bash
2
3 select operacion in Multiplicacion Suma Resta CualEsMayor Cerrar
4 do
5     case $operacion in
6         Multiplicacion) echo "MULTIPLICACION: $(( $1 * $2 ))" ;;
7         Suma) echo "SUMA: $(( $1 + $2 ))" ;;
8         Resta) echo "RESTA: $(( $1 - $2 ))" ;;
9         CualEsMayor) if [ $1 -gt $2 ]; then echo "es mayor el $1";
10             else echo "es mayor el $2"; fi;;
11         Cerrar|*) exit ;;
12     esac
13 done

```

```

fa@FaboFacultad:~/Desktop/Practica3-2023$ bash 12c.sh 2 3
1) Multiplicacion 3) Resta 5) Cerrar
2) Suma 4) CualEsMayor
#? 1
MULTIPLICACION: 6
#? 2
SUMA: 5
#? 3
RESTA: -1
#? 4
es mayor el 3
#? 6
fa@FaboFacultad:~/Desktop/Practica3-2023$

```

13) Ejercicio

Uso de las estructuras de control

(a) Realizar un script que visualice por pantalla los números del 1 al 100 así como sus cuadrados.

```
13.sh
1 #!/bin/bash
2 for i in {1..100}
3 do
4     echo "$i La potencia es: $((i * i))"
5 done
```

```
1 La potencia es: 1
2 La potencia es: 4
3 La potencia es: 9
4 La potencia es: 16
5 La potencia es: 25
6 La potencia es: 36
7 La potencia es: 49
8 La potencia es: 64
9 La potencia es: 81
10 La potencia es: 100
```

(b) Crear un script que muestre 3 opciones al usuario: Listar, DondeEstoy y QuienEsta. Según la opción elegida se le debe mostrar:

- Listar: lista el contenido del directoria actual.
- DondeEstoy: muestra el directorio donde me encuentro ubicado.
- QuienEsta: muestra los usuarios conectados al sistema.

```
13b.sh
1 #!/bin/bash
2
3 select opcion in Listar DondeEstoy QuienEsta
4 do
5     case $opcion in
6         Listar) echo $(ls) ;;
7         DondeEstoy) echo $(pwd);;
8         QuienEsta) echo $(who);;
9         *) exit;;
10    esac
11 done
```

```
1) Listar
2) DondeEstoy
3) QuienEsta
#? 1
= 10.sh 11.sh 12a.sh 12b.sh 12c.sh
acticashell-script
#? 2
/home/fa/Desktop/Practica3-2023
#? 3
fa tty2 2023-04-17 14:25 (tty2)
#? █
```

(c) Crear un script que reciba como parámetro el nombre de un archivo e informe si el mismo existe o no, y en caso afirmativo indique si es un directorio o un archivo. En caso de que no exista el archivo/directorio cree un directorio con el nombre recibido como parámetro

```

13c.sh
1 #!/bin/bash
2 if (( $# != 1 )); then exit; fi
3
4 if [ -e $1 ]; then
5     echo "El archivo $1 existe"
6     if [ -f $1 ]; then echo "Y es un archivo"; fi
7     if [ -d $1 ]; then echo "Y es un directorio"; fi
8     exit
9 fi
10 mkdir $1

```

fa@FaboFacultad:~/Desktop/Practica3-2023\$ nvim 13c.sh

fa@FaboFacultad:~/Desktop/Practica3-2023\$ ls

'='	11.sh	12b.sh	13b.sh	13.sh	3e.sh	6.sh	7.shw	9.sh
	10.sh	12a.sh	12c.sh	13c.sh	1.sh	4.sh	7.sh	8.sh
								practicashell-script

fa@FaboFacultad:~/Desktop/Practica3-2023\$ bash 13c.sh 1.sh

El archivo 1.sh existe

Y es un archivo

fa@FaboFacultad:~/Desktop/Practica3-2023\$ bash 13c.sh practicashell-script/

El archivo practicashell-script/ existe

Y es un directorio

fa@FaboFacultad:~/Desktop/Practica3-2023\$ bash 13c.sh prueba

fa@FaboFacultad:~/Desktop/Practica3-2023\$ ls

'='	12a.sh	13b.sh	1.sh	6.sh	8.sh			prueba
	10.sh	12b.sh	13c.sh	3e.sh	7.sh	9.sh		
	11.sh	12c.sh	13.sh	4.sh	7.shw			practicashell-script

fa@FaboFacultad:~/Desktop/Practica3-2023\$



14) Ejercicio

Renombrando Archivos: haga un script que renombre solo archivos de un directorio pasado como parametro agregandole una CADENA, contemplando las opciones:

- **"-a CADENA"**: renombra el fichero concatenando CADENA al final del nombre del archivo
- **"-b CADENA"**: renombra el fichero concatenando CADENA al principio del nombre del archivo

Ejemplo:

- Si tengo los siguientes archivos: /tmp/a /tmp/b
- Al ejecutar: ./renombra /tmp/ -a EJ
- Obtendré como resultado: /tmp/aEJ /tmp/bEJ
- Y si ejecuto: ./renombra /tmp/ -b EJ
- El resultado será: /tmp/EJa /tmp/EJb

Comandos ejecutados

- bash 14.sh prueba -a Ej
- Despues
- bash 14.sh prueba -b Ej

```
14.sh
1 #!/bin/bash
2
3 if [ $# -ne 3 ]; then
4     echo -e "
5 Parametros: \n
6 Path \n
7 -a/-b \n
8 sufijo/prefijo
9 "; exit;fi
10
11 if ! [ -e $1 ]; then
12     echo "El path ingresado no existe"
13     exit
14 fi
15
16 cd $1
17 archivos="$(ls)"
18 ls -l
19
20 if [ $2 = "-a" ];then
21     for i in ${archivos[*]};do mv $i $i$3;done
22 elif [ $2 = "-b" ];then
23     for i in ${archivos[*]};do mv $i $3$i;done
24 else
25     echo "Tiene que ingresar -a o -b"
26 fi
27 echo "Despues de todo:"
28 ls -l
```

```
fa@FaboFacultad:~/Desktop/Practica3-2023$ bash 14.sh prueba -a Ej
archivo1.txt
archivo2.txt
archivo3.txt
archivo4.txt
archivo5.txt
crear.sh
Despues de todo:
archivo1.txtEj
archivo2.txtEj
archivo3.txtEj
archivo4.txtEj
archivo5.txtEj
crear.shEj
fa@FaboFacultad:~/Desktop/Practica3-2023$ bash 14.sh prueba -b Ej
archivo1.txtEj
archivo2.txtEj
archivo3.txtEj
archivo4.txtEj
archivo5.txtEj
crear.shEj
Despues de todo:
Ejarchivo1.txtEj
Ejarchivo2.txtEj
Ejarchivo3.txtEj
Ejarchivo4.txtEj
Ejarchivo5.txtEj
Ejcrear.shEj
```



15) Ejercicio

Comando cut. El comando cut nos permite procesar la líneas de la entrada que reciba (archivo, entrada estándar, resultado de otro comando, etc) y cortar columnas o campos, siendo posible indicar cual es el delimitador de las mismas. Investigue los parámetros que puede recibir este comando y cite ejemplos de uso.



16) Ejercicio

Realizar un script que reciba como parámetro una extensión y haga un reporte con 2 columnas, el nombre de usuario y la cantidad de archivos que posee con esa extensión. Se debe guardar el resultado en un archivo llamado **reporte.txt**



17) Ejercicio

Escribir un script que al ejecutarse imprima en pantalla los nombre de los archivos que se encuentran en el directorio actual, intercambiando minúsculas por mayúsculas, además de eliminar la letra a (mayúscula o minúscula). Ejemplo, directorio actual:

```
IsO  
pepE  
Maria  
Si ejecuto: ./ejercicio17
```

Obtendré como resultado

iSo
PEPe
mRI

Ayuda: Investigar el comando **tr**



18) Ejercicio

Crear un script que verifique cada 10 segundos si un usuario se ha logueado en el sistema (el nombre del usuario será pasado por parámetro). Cuando el usuario finalmente se loguee, el

programa deberá mostrar el mensaje "Usuario XXX logueado en el sistema" y salir.



19) Ejercicio

Escribir un Programa de "Menu de Comandos Amigable con el Usuario" llamado menu, el cual, al ser invocado, mostrará un menú con la selección para cada uno de los scripts creados en esta práctica. Las instrucciones de como proceder deben mostrarse junto con el menú. El menú deberá iniciarse y permanecer activo hasta que se seleccione Salir. Por ejemplo:

```
MENU DE COMANDOS
03. Ejercicio 3
12. Evaluar Expresiones
```

13. Probar estructuras de control
...
Ingrese la opción a ejecutar: 03



20) Ejercicio

Realice un scrip que simule el comportamiento de una estructura de PILA e implemente las siguientes funciones aplicables sobre una estructura global definida en el script:

- **push:** Recibe un parámetro y lo agrega en la pila
- **pop:** Saca un elemento de la pila
- **lenght:** Devuelve la longitud de la pila

- **print:** Imprime todos elementos de la pila



21) Ejercicio

- Agregue 10 elementos a la pila.
- Saque 3 de ellos.
- Imprima la longitud de la cola.
- Luego imprima la totalidad de los elementos que en ella se encuentran.



22) Ejercicio

Dada la siguiente declaración al comienzo de un script: `num=(10 3 5 7 9 3 5 4)` (la cantidad de elementos del arreglo puede variar). Implemente la función productoria dentro de este script, cuya tarea sea multiplicar todos los números del arreglo



23) Ejercicio

Implemente un script que recorra un arreglo compuesto por números e imprima en pantalla sólo los números pares y que cuente sólo los números impares y los informe en pantalla al finalizar el recorrido



24) Ejercicio

Dada la definición de 2 vectores del mismo tamaño y cuyas longitudes no se conocen

```
vector1=( 1 .. N)
```

```
vector2=( 7 .. N)
```

Por ejemplo:

```
vector1=( 1 80 65 35 2 )
```

y

```
vector2=( 5 98 3 41 8 ).
```

Complete este script de manera tal de implementar la suma elemento a elemento entre ambos vectores y que la misma sea impresa en pantalla de la siguiente manera:

```
La suma de los elementos de la posición 0 de los vectores es 6
La suma de los elementos de la posición 1 de los vectores es 178
...
La suma de los elementos de la posición 4 de los vectores es 10
```



25) Ejercicio

Realice un script que agregue en un arreglo todos los nombres de los usuarios del sistema pertenecientes al grupo "users". Adicionalmente el script puede recibir como parametro:

- **"-b n"**: Retorna el elemento de la posición n del arreglo si el mismo existe. Caso contrario, un mensaje de error.
- **"-l"**: Devuelve la longitud del arreglo
- **"-i"**: Imprime todos los elementos del arreglo en pantalla



26) Ejercicio

Escriba un script que reciba una cantidad desconocida de parámetros al momento de su invocación (debe validar que al menos se reciba uno). Cada parámetro representa la ruta absoluta de un archivo

o directorio en el sistema. El script deberá iterar por todos los parámetros recibidos, y solo para aquellos parámetros que se encuentren en posiciones impares (el primero, el tercero, el quinto, etc.) verificar si el archivo o directorio existen en el sistema, imprimiendo en pantalla que tipo de objeto es (archivo o directorio). Además, deberá informar la cantidad de archivos o directorios inexistentes en el sistema.



27) Ejercicio

Realice un script que implemente a través de la utilización de funciones las operaciones básicas sobre arreglos:

- inicializar: Crea un arreglo llamado array vacío

- agregar_elem <parametro1>: Agrega al final del arreglo el parámetro recibido
- eliminar_elem <parametro1>: Elimina del arreglo el elemento que se encuentra en la posición recibida como parámetro. Debe validar que se reciba una posición válida
- longitud: Imprime la longitud del arreglo en pantalla
- imprimir: Imprime todos los elementos del arreglo en pantalla
- inicializar_Con_Valores <parametro1> <parametro2>: Crea un arreglo con longitud <parametro1> y en todas las posiciones asigna el valor <parametro2>



28) Ejercicio

Realice un script que reciba como parámetro el nombre de un directorio. Deberá validar que el mismo exista y de no existir causar la terminación del script con código de error 4. Si el directorio

existe deberá contar por separado la cantidad de archivos que en él se encuentran para los cuales el usuario que ejecuta el script tiene permiso de lectura y escritura, e informar dichos valores en pantalla. En caso de encontrar subdirectorios, no deberán procesarse, y tampoco deberán ser tenidos en cuenta para la suma a informar.



29) Ejercicio

Implemente un script que agregue a un arreglo todos los archivos del directorio /home cuya terminación sea .doc. Adicionalmente, implemente las siguientes funciones que le permitan acceder a la estructura creada:

- verArchivo <nombre_de_archivo>: Imprime el archivo en pantalla si el mismo se encuentra en el arreglo. Caso contrario imprime el mensaje de error "Archivo no encontrado" y devuelve como valor de retorno 5
- cantidadArchivos: Imprime la cantidad de archivos del /home con terminación .doc
- borrarArchivo <nombre_de_archivo>: Consulta al usuario si quiere eliminar el archivo lógicamente. Si el usuario responde Si, elimina el elemento solo del arreglo. Si el usuario responde No, elimina el archivo del arreglo y también del FileSystem. Debe validar que el archivo exista en el arreglo. En caso de no existir, imprime el mensaje de error "Archivo no encontrado" y devuelve como valor de retorno 10



30) Ejercicio

Realice un script que mueva todos los programas del directorio actual (archivos ejecutables) hacia el subdirectorio "bin" del directorio HOME del usuario actualmente logueado. El script debe imprimir en pantalla los nombres de los que mueve, e indicar cuántos ha movido, o que no ha movido ninguno. Si el directorio "bin" no existe, deberá ser creado.

