El objetivo de esta práctica es que el alumno desarrolle habilidades concernientes a Shell Scripting.

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

> 1- Los Scripts son interpretados, estos principalmente se utilizan para manejo de archivos, ya que en sí Bash tiene buena eficiencia para la manipulación de muchos archivos rapido y facil. Estos programas son relativamente sencillos pero tienen mucho potencial.

2. Investigar la funcionalidad de los comandos echo y read

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

A- Para indicar un comentario dentro de un script usamos el # para indicar que es un comentario.

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

B- Las variables se declaran indicando el NOMBRE=VALOR) sin dejar ningún espacio, y para hacer uso de esas variables indicamos siempre el símbolo $ antes del nombre de la variable.

3. Crear dentro del directorio personal del usuario logueado un directorio llamado práctica shell-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:"
```

(a) Asignar al archivo creado los permisos necesarios de manera que pueda ejecutarlo.

A- se le asigna permisos al archivo con chmod +x file.txt

(b) Ejecutar el archivo creado de la siguiente manera: ./mostrar
Ejecutado

(c) ¿Qué resultado visualiza?
Veo la fecha, el nombre y apellido introducidos.

(d) Las backquotes (`) entre el comando whoami ilustran el uso de la sustitución de comandos. ¿Qué significa esto?
Permite que la salida estándar de un programa se utilice como parte de la línea que se va a interpretar

(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.
Hecho He

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 ?
$HOME se utiliza para variables globales
$* para las variable pasadas como parámetros
$# Es el numero de argumentos que se le pasa a un script.
$? Este retorna 0 se utiliza para operaciones booleanas “equals”

5. ¿Cual es la funcionalidad de comando exit? ¿Qué valores recibe como parámetro y cuál es su significado?
5- Sirve para detener cualquier ejecucion dentro del SO, si lo volvemos a ejecutar nuevamente, nos saca de la terminal.

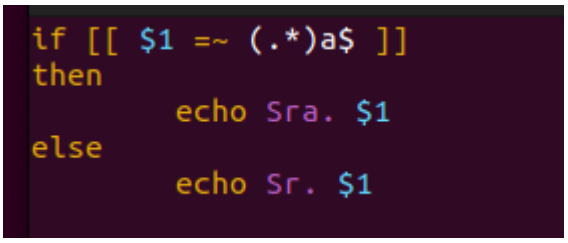
6. El comando `expr` permite la evaluación de expresiones. Su sintaxis es: `expr arg1 op arg2`,
donde `arg1` y `arg2` representan argumentos y `op` la operación de la expresión.
Investigar que tipo de operaciones se pueden utilizar. se utiliza en toda operación lógicas y aritméticas

7. 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 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.

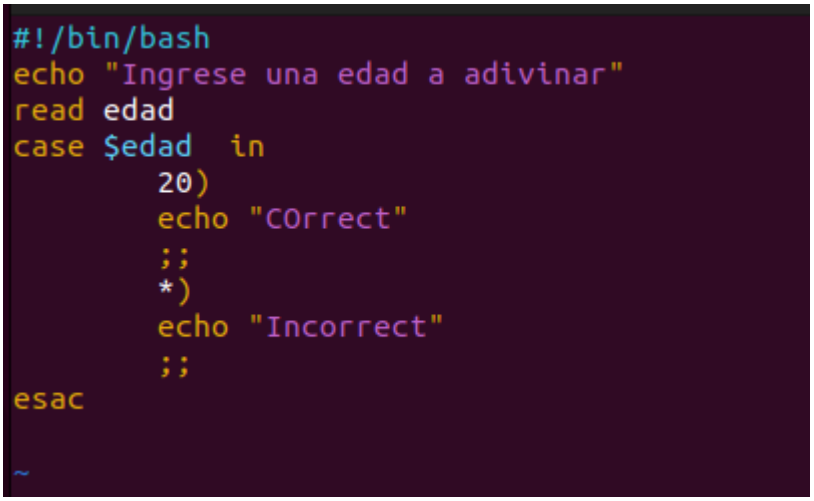
8. Estructuras de control. Investigue la sintaxis de las siguientes estructuras de control incluidas en shell scripting:

if:



```
if [[ $1 =~ (.*)a$ ]]
then
        echo Sra. $1
else
        echo Sr. $1
```

case :



```
#!/bin/bash
echo "Ingrese una edad a adivinar"
read edad
case $edad  in
        20)
        echo "COrrect"
        ;;
        *)
        echo "Incorrect"
        ;;
esac

~
```

while:

```
#!/bin/bash
contador=0
termina=10

while [ $termina -ge $contador ]
do
        echo $contador
        let contador=contador+1
done
~
~
~
```

for:

```
for i in 5 10
do
        echo "Hello $i" #imprime -> Hello 5 y dsp Hello 10
done
```

select:

```
PS3="ENter a number:"
select character in SHeldon LEonard PEnny HOward Raj
do
        echo "Selected character: $character"
        echo "Selected character: $REPLY"
done
```

El select suele ser usado con un if o case, porque sino te queda en un loop infinito.

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

Break: funciona como un corte de tu proceso/o lo que se aque se este ejecutando, puede ser un select tambien.

Continue: sirve para indicar que justamente continue con el bucle for

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

Pueden ser de dos tipos: **variables de entorno o variables de shell**. Las variables de entorno son variables definidas para el shell actual y heredadas por cualquier shell o proceso secundario. Las variables de entorno se utilizan para transmitir información a procesos que se producen desde el shell.

Los arreglos pueden ser declarados, como array=(valor1 valor2 valor3) o array={1..10}.

11. ¿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 estas se hacen con nombre(){comandos}.

```
suma(){
        var1=1
        var2=1
        var3= expr $var1 + $var2
        echo $var3
}

suma #-> imprime 2
```

12. 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.

```
echo "ingrese dos numeros:"
read num1 num2
echo "la multiplicacion es: $(expr $num1 \* $num2)"

#expr $num1 \* $num2

echo "la sumatoria es: $(expr $num1 + $num2)"
#expr $num1 + $num2

echo "la diferencia es: $(expr $num1 - $num2)"
#expr $num1 - $num2

if [[ $num1 -gt $num2 ]]
then
        echo "el primer numero es mayor"
else
        echo "el segundo numero es mayor"
fi
```

(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.

```
re='^[0-9]+$'
if [[ ! -z "$num1" ]] & [[ $num1 =~ $re ]] && [[ ! -z "$num2" ]] & [[ $num2 =~ $re ]]
then

        echo "suma (expr $num1 + $num2)"
        echo "resta: $(expr $num1 - $num2)"
        echo "multiplicacion: $(expr $num1 \* $num2)"
        echo "Division: $(expr $num1 / $num2)"
        if [[ $num1 -gt  $num2 ]]
        then
                echo "num 1 es mayor"
        else
                echo "num 2 es mayor"
        fi
else
        echo "Error debe ingresar los 2 parametros ENTEROS para su respectiva suma,resta multiplicacion y division"
        exit 1
fi
```

Para que funcione la entrada por parametros, en vez de darle el nombre de $num1 y $num2 se tienen que usar de $1 hasta $N de la cantida que necesites.

```
echo "$#"
if [ "$#" -ne "2" ]
then
        echo "no se pasaron exactamente dos parametros"
else
        echo "los dos numeros ingresados son: $1 y $2 "
        echo "la sumatoria de estos es: $( expr $1 + $2 )"
        echo "la multiplicacion de estos es: $( expr $1 \* $2 )"
        echo "la diferencia de estos es: $( expr $1 - $2 )"
        if [ $1 -gt $2 ]
        then
                echo "$1 es mayor que $2"
        else
                echo "$2 es mayor que $1"
        fi
fi
```

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

```
#!/bin/bash

if [[ ! -z "$1" ]] & [[ ! -z "$2" ]] && [[ ! -z "$3" ]]
then

        case $2 in
                "+")
                        echo "Resultado: $(expr $1 + $3)"
                        ;;
                "-")
                        echo "resta: $(expr $1 - $3)"
                        ;;
                "*")
                        echo "Resultado mul: $(expr $1 * $3)"
                        ;;
                "/")
                        echo "Resultado Division: $(expr $1 / $3)"
                        ;;
                *)
                        echo "Operando invalido"
                        exit 1

                esac
                exit 0
else
        echo "ERROR debe ingresar primer el numero, operador y els egundo numero!"
        exit 1
fi
```

Es mejor usar:

```
# $1 y $3 se esperan numeros y $2 se espera una operacion aritmetica
if [ "$#" -ne "3" ]
then
        echo "no se ingresaron los 3 parametros necesarios (numero, operacion numero)"
else
        echo " la operacion ($1 $2 $3) es: $( expr $1 "$2" $3 )"
fi
```

13. Uso de las estructuras de control:

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

```
#!/bin/bash

for i in {0..100}
do
        echo " numero $i"
        echo " su cuadrado es: $(expr $i \* $i)"
done

~
```

(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 directorio actual.

DondeEstoy: muestra el directorio donde me encuentro ubicado.

QuienEsta: muestra los usuarios conectados al sistema.

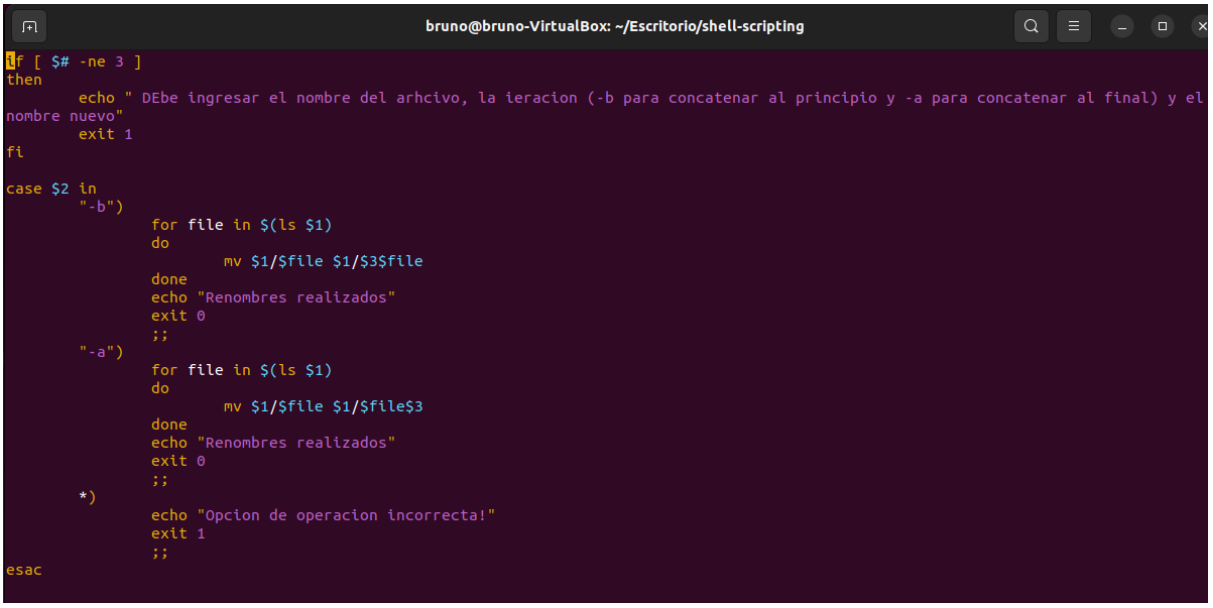
```
#!/bin/bash

if [[ ! -z "$1" ]]
then
        case $1 in
                "Listar")
                        echo "$(ls)"
                        ;;
                "DondeEstoy")
                        echo "$(pwd)"
                        ;;
                "QuienEsta")
                        echo "$(who)"
                        ;;
                *)
                        echo "parametro incorrectoooo"
                        exit 1
        esac
        exit 0
else
        echo "NO se ingreso un parametro correcto"
        exit 1
fi
```

(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.

```
if [ $# -ne 1 ]
then
        echo "Debe indicar nombre del archivo"
        exit 1
fi

if [ ! -e $1 ]
then
        echo " NO se encontro el archivo/directorio!, creando un dir en $(pwd)"
        mkdir $1
        exit 0
elif [ -d $1 ]
then
        echo "$1 es un directorio"
else
        echo "$1 es un archivo"

exit 0
fi
```

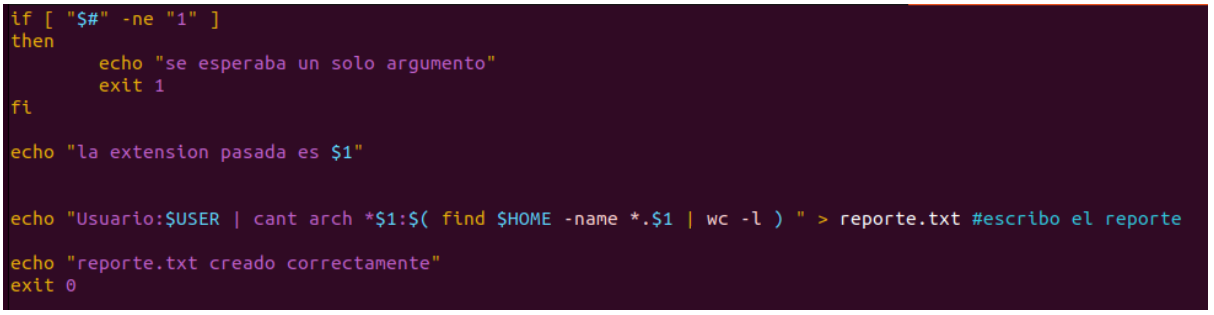
14. Renombrando Archivos: haga un script que renombre solo archivos de un directorio pasado como parametro agregandole una CADENA, contemplando las opciones
“-a CADENA”: renombra el fichero concatenando CADENA al final del nombre del archivo
“-b CADENA”: renombra el fichero concatenado CADENA al principio 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
```



15. 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.pedimos que nos retorne todos los string que estén en la columna 4

16. 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. 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`

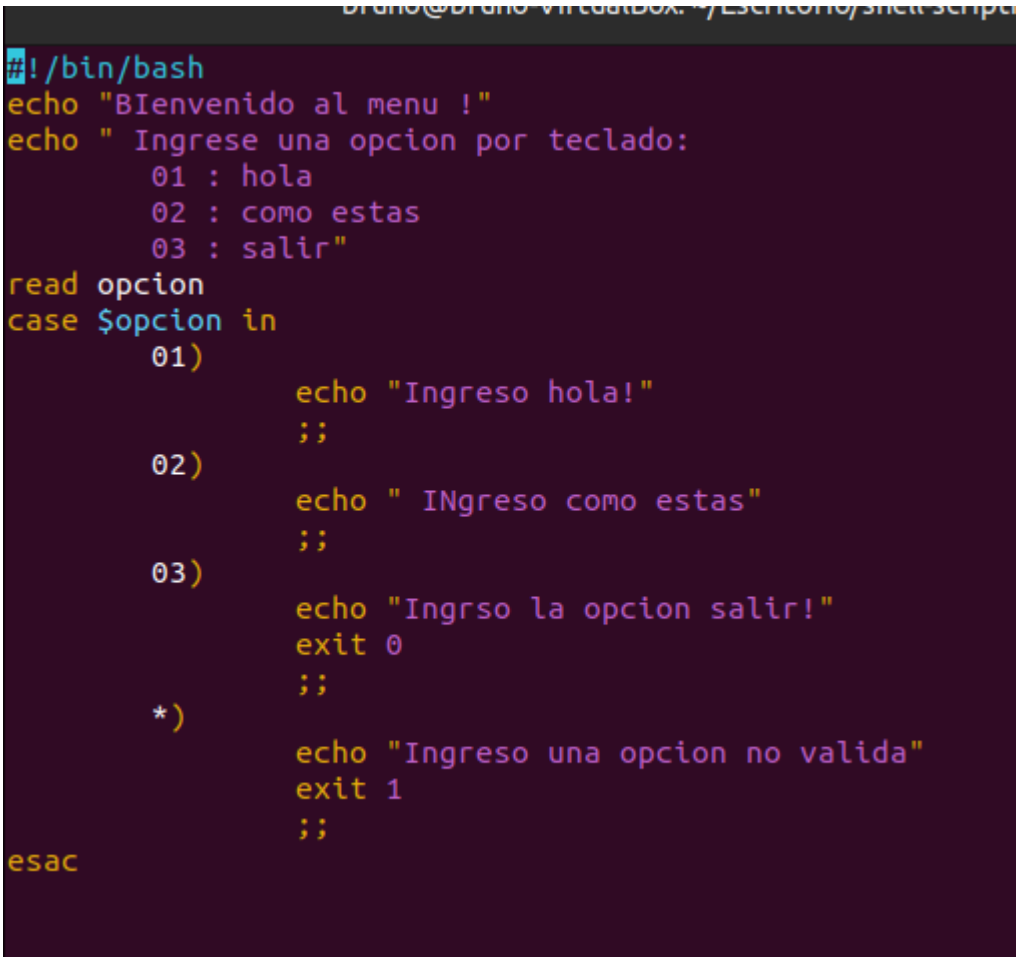

```
echo $(ls) | tr -d ‘a’ | tr -d ‘A’ | tr -t ‘a-zA-Z’ ‘A-Za-z’
```

18. 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.

```bash
#!/bin/bash

if [ $# -ne 1 ]
then
        echo "Ingrese el nombre del usuario por parametro
        exit 1
fi




while [ $(w -s -h $1 | wc -l) -ne 1 ]
do
        sleep 10s
        echo "El usuario $1 no esta logueado!"
done

echo "El usuario $1 esta logueado!"
exit 0
```

19. 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
```



```
#!/bin/bash
echo "BIenvenido al menu !"
echo " Ingrese una opcion por teclado:
        01 : hola
        02 : como estas
        03 : salir"
read opcion
case $opcion in
        01)
                echo "Ingreso hola!"
                ;;
        02)
                echo " INgreso como estas"
                ;;
        03)
                echo "Ingrso la opcion salir!"
                exit 0
                ;;
        *)
                echo "Ingreso una opcion no valida"
                exit 1
                ;;
esac
```

este jercicio tendria q ejecutar ejercicios anteriores, claramente no los tengo guaraddos, so aqui tienes un ejemplo de menu >:d

20. Realice un script 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
`length`: Devuelve la longitud de la pila `print`: Imprime todos elementos de la pila

```
#!/bin/bash
echo "Bienvenido al menu de la pila"
echo "Las distintas opciones son: "
echo "'push <parametro>' -> mete el parametro en la pila"
echo "'pop' -> saca el primer elemento de la pila"
echo "'lenght' -> devuelve la longitud de la lista"
echo "'print' -> imprime los elementos de la lista"
echo "'exit' -> sale del menu y termina el script"

declare -a pila

while true
do
        echo "ingrese su opcion:"
        read opcion
        case $opcion in
                "push")
                        echo "ingrese un valor para apilar"
                        read valor
                        pila=("${pila[@]}" $valor)
                ;;
                "pop")
                        echo "desapilando la pila"
                        unset pila[-1]
                ;;

                "lenght")
                        echo "longitud: ${#pila[@]}"
                ;;
                "print")
                        echo "Elementos: ${pila[@]}"
                ;;
                "exit")
                        echo "saliendo del menu.."
                        exit 0
                ;;
                *)
                        echo "opcion no valida, intente otra"
                ;;

        esac
done
```

21. Dentro del mismo script y utilizando las funciones implementadas:
Agregue 10 elementos a la pilaSaque 3 de ellos
Imprima la longitud de la cola Luego imprima la totalidad de los elementos que en ella se encuentran. Hecho

22. 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

```
#!/bin/bash
num=( 10 2 1 1 1 1 1 1)
OP=${num[0]}
productoria(){
        for i in "${num[@]}"
        do
                resultado=$(expr $OP \* $i )
                OP=$resultado
                echo $resultado
        done
        echo "La suma de los valores del vector es $resultado )"
}
productoria
```

23. 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.

```
#!/bin/bash

pares=()
impares=()
arr=(1 32 10 20 5 7 8 9)
for i in ${arr[@]}
do
        if [ $(expr $i % 2) -eq 0 ]
        then
                impares=("${impares[@]}" $i)
        else
                pares=("${pares[@]}" $i)
        fi
done
echo "pares: ${pares[@]}"
echo "impares: ${impares[@]}"
```

24. 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
```

```bash
#!/bin/bash
vector1=( 1 80 65 35 2 )
vector2=( 5 98 3 41 8 )
fila=0
for i in "${vector1[@]}"
do
        echo "La suma de los elementos en la posicion $fila: $(expr ${vector1[$fila]} + ${vector2[$fila]})"
        let fila++
done
```

25. 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

```bash
#!/bin/bash
if [ $# -gt 2 ]; then
        echo "Numero de parametros invalido"
        exit 1
fi

#agrego al arreglo
arreglo=( $(cat /etc/group | grep users | cut -d: -f4| tr ',' ' '))

#dependiendo parametro


case $1 in
        "-b")
                if [ -z $2 ]; then
                        echo "Ingrese un valor de n"
                elif [ $2 -lt 0 ] || [ $2 -ge ${#arreglo[*]} ]; then
                        echo "Valor de n invalido"
                else
                        echo "${arreglo[$2]}"
                fi
        ;;
        "-l")
                if ! [ -z $2 ]; then
                        echo "Numero de parametros invalido"
                else
                        echo "${#arreglo[*]}"
                fi
        ;;
        "-i")
                if ! [ -z $2]; then
                        echo "Numero de parametros invalido"
                else
                        echo "${arreglo[*]}"
                fi
        ;;
        *)
                echo "Ingrese un parametro valido"
        ;;
esac
exit 0
```

26. 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 q 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.

```
#!/bin/bash
if [ $# -lt 1 ]; then
        exit 1
fi

cant=0
it=0
for i in $*; do
        let it++
        if [ $(expr $it % 2) -ne 0 ]; then
                if ! [ -e $i ]; then
                        echo "No existe"
                        cant=$(expr $cant + 1)
                else
                        if [ -d $i ]; then
                                echo "Existe es un directorio"
                        else
                                if [ -f $i ]; then
                                        echo "Existe es un archivo"
                                fi
                        fi
                fi
        else
                continue 2
        fi
done

echo "Cantidad de archivos inexistentes: $cant"

exit 0
```

27. 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>

```
function inicializar(){
    arreglo=()
}

function agregar_elem(){
    if [$# -ne 1]
    then
        echo "No se ha ingresado un valor para agregar"
        break
    fi
    else
    if [ ${#arreglo[*]} -eq 0 ]
        then
                arreglo[0]=$1
        else
                arreglo[${#arr[*]}]=$1
        fi
}

function imprimireliminar_elem(){
    if ($# -ne 1)
    then
        echo "No se ha ingresado una posicion para eliminar"
        break
    fi
    else
        if [$1 -lt $(expr ${#arreglo[*]} - 1)]
            unset arreglo[$1]
        fi
}
```

```
function longitud(){
    echo "La longitud del arreglo es: ${#arreglo[*]-1}"
}
function imprimir {
        echo "${arr[*]}"
}

function inicializar_con_valores {
        arreglo=()
        for ((i=0; i<$1; i++))
        do
                arr[$i]=$2
        done
}

##PROGRAMA PRINCIPAL
inicializar
agregar_elem 45
eliminar_elem 1
longitud
imrpmir
inicializar con valores 2 3
```

28. 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.

```bash
#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.


informar_directorio(){
    if [$# -ne 1]
    then
        echo "No se ha ingresado un nombre de directorio"
        break
    fi
    else
        if [ -d "$1" ]
        then
            for i in ls $1;
            do
               if [ -f $i ]&&[ -r $i ]&&[ - w $i ]; then
                    CANT=`expr $CANT + 1`
                fi
            done
            echo $CANT
        else echo "No existe directorio."
        exit 4
        fi
}


##principal
echo "ingrese nombre de directorio"
read directorio
informar_directorio $directorio
```

29. 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 en-
contrado” 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 archi-

vo 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

```
15
16  arr=()
17
18  function cantidadArchivos {
19          echo "Cantidad de archivos: ${#arr[*]}"
20  }
21
22  function verArchivo {
23          existe=0
24          for i in ${arr[*]}
25          do
26                  #Cuando se hace echo $i imprime /home/cacho/prueba/CSOClase7.docx
27                  echo $(echo $i | cut -d '/' -f 5)
28                  if [ "$(echo $i | cut -d '/' -f 5)" = "$1" ]
29                  then
30                          existe=1
31                          break
32                  fi
33          done
34          if [ $existe -eq 1 ]
35          then
36                  cat $i
37          else
38                  echo "No existe el archivo en el arreglo"
39          fi
40  }
41
42  j=0
43  #Deberia ser solo $HOME pero /prueba/ es para debuggear
44  #Deberia ser solo "*.doc" pero la x es por el tipo de extension que use para debuggear
45  for i in $(find $HOME/prueba/ -name "*.docx")
46  do
47          arr[j]=$i
48          let j++
49  done
50
51  verArchivo CSOClase7.docx
52  cantidadArchivos
53
```

30. 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.

```
#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.

#!/bin/bash

#prueba deberia ser bin
if [ -e $HOME/prueba ]
then
        echo existe
        for i in $(find $PWD -name "*.docx")
        do
                mv $i $HOME/prueba
        done
else
        mkdir prueba
        echo lo cree
```

ultimos 5 ejercicios revisar, ya que fueron copipasteados…. saludos!