# Práctica 3

1. ¿Qué es shell scripting?¿ A qué tipos de tareas están orientados los scripts?¿Los scripts deben compilarse?¿Por qué?

   Shell scripting es la escritura de script que se ejecutan en la línea de comando de un SO Unix. Está orientado a automatización de tareas repetitivas, administración de archivos y directorios, procesamiento de datos y de red.

   Los scripts no necesitan ser compilados porque son interpretados directamente por el intérprete de comandos. Cuando se ejecuta un script de shell, el intérprete de comandos de la shell lee el archivo de texto que contiene los comandos de shell y los ejecuta en secuencia.

2. Investigar la funcionalidad de los comandos echo y read

   a. Comando **echo:** imprime texto en la salida estándar

   ```
   echo "Hola, mundo!"
   ```

   b. Comando **read:** lee la entrada de usuario desde teclado y la almacena en una variable

   ```
   read nombre
   ```

   c. ¿Cómo se indican los comentarios dentro de un script?

   Hay varias maneras de indicarlos

   ```
   # Ejemplo de comentario de una sola línea
   : '
   Comentario de varias líneas
   '
   ```

   d. ¿Cómo se declaran y se hace referencia a variables dentro de un script?

   **Declaración:** se hace sin necesidad de especificar un tipo de datos y puede contener números, cadenas de texto, arreglos, etc.

   ```
   nombre_de_la_variable= valor
   ```

   **Referencia:** se usa el signo $ seguido del nombre de la variable. Por ejemplo, para imprimir el valor de la variable mensaje, se usaría el siguiente comando.

   ```
   echo $mensaje
   ```

   Hay que tener en cuenta que los nombres de las variables son case sensitive.

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:

   ```
   #!/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

   ```
   chmod u+rwx,g+rx,o+rx mostrar.sh
   ```

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

   c. ¿Qué resultado visualiza?

   d. Los backquotes entre el comando whoami ilustran el uso de la instrucción de comandos. ¿Qué significa esto?

   Los backquotes se usan para indicar que la salida del comando whoami debe ser interpretada y usada como parte de la cadena de texto que se pasará al comando echo.

   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.

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?

   **Envío de parámetros:** se acceden usando variables especiales que tienen nombres predefinidos y contienen información sobre los parámetros que se pasan al script.

| Variable especial | Función |
|---|---|

| | |
|---|---|
| $# | Contiene el número total de parámetros pasados al script. |
| $* | Contiene todos los parámetros pasados al script como una sola cadena de texto. |
| $@ | Contiene todos los parámetros pasados al script como una lista separada por espacios |
| $1, $2 .. | Contiene los valores de los parámetros pasados al script en orden. |
| $? | Obtiene el código de salida (exit status) del último comando ejecutado. El valor de esta variable es un número que indica si el comando se ejecutó correctamente o si hubo algún error. Por convención 0 indica que el comando se ejecutó correctamente mientras que algún otro valor indica que se produjo error. |
| $HOME | Usado para hacer referencia a archivos o directorios dentro del directorio principal del usuario. |

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

   El comando **exit** provoca la eliminación inmediata del proceso correspondiente al script que está leyendo el shell.

   - 0 indica que el script se ejecutó de forma exitosa.
   - Un valor distinto indica un código de error.

   Puedo consultar el exit status imprimiendo la variable $?.

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 qué tipo de operaciones se pueden utilizar.

| operación | Función |
|---|---|
| -eq | Igualdad |
| -ne | Desigualdad |
| -gt | Mayor que |
| -ge | Mayor o igual |
| -lt | Menor |
| -le | Menor o igual |
| -b fichero | Cierto si el fichero existe y es un dispositivo de bloques |
| -c fichero | Cierto si fichero existe y es un dispositivo de caracteres |
| -d fichero | Cierto si fichero existe y es un directorio |
| -e fichero | Cierto si fichero existe |
| -f fichero | Cierto si fichero existe y es un fichero normal |
| -g fichero | Cierto si fichero existe y tiene el bit de grupo activado |
| -s fichero | Cierto si fichero existe y su tamaño es mayor que 0 |
| -w fichero | Cierto si fichero existe y es escribible |
| -x fichero | Cierto si fichero existe y es ejecutable |
| fichero1 -nt fichero2 | Cierto si fichero1 es más reciente que fichero2 |
| fichero1 -ef fichero2 | Cierto si fichero1 es más viejo que fichero2 |
| -z cadena | Cierto si la longitud de cadena es cero |
| -n cadena | Cierto si la longitud de la cadena no es 0 |
| cadena1 = cadena2 | Cierto si cadena1 es igual a cadena2 |
| cadena1 ! = cadena2 | Cierto si las cadenas no son iguales |
| ! expr | Cierto si expr es falsa |
| expr1 -a expr2 | Cierto si expr1 y expr2 son ciertas |

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 [expresion]. 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.

   El comando text permite efectuar una serie de pruebas sobre los archivos, cadenas de caracteres, valores aritméticos y el entorno del usuario. Posee dos sintaxis

```
test expresion
[expresion] #expresion representa el test a efectuar
```

   - -b fichero: cierto si el fichero existe y es un dispositivo de bloques.
   - c fichero: cierto si el fichero existe y es un dispositivo de caracteres.
   - -d fichero: cierto si el fichero existe y es un directorio.
   - -e fichero: cierto si el fichero existe.
   - -f fichero: cierto si el fichero existe y es un fichero normal.
   - -g fichero: cierto si el fichero existe y tiene el bit de grupo activado.
   - -k fichero: cierto si el fichero tiene el bit de sticky activado.
   - -L fichero: cierto si el fichero existe y es un enlace simbólico.

- -p fichero: cierto si el fichero existe y es una tubería nombrada.
- -r fichero: cierto si el fichero existe y es legible.
- -s fichero: cierto si el fichero existe y su tamaño es mayor que cero.
- -S fichero: cierto si el fichero existe y es un socket.
- -t [df]: cierto si df está abierto en un terminal. Si fd es omitido, se toma 1 (salida estándar) por defecto.
- -u fichero: cierto si el fichero existe y tiene el bit de usuario activo.
- -w fichero: cierto si el fichero existe y es escribible.
- -x fichero: cierto si el fichero existe y es ejecutable.
- -O fichero: cierto si el fichero existe y es propiedad del identificador efectivo del usuario.
- -G fichero: cierto si el fichero existe y es propiedad del identificador efectivo del grupo.
- fichero1 -nt fichero2: cierto si fichero1 es más reciente (en base a la fecha de modificación) que fichero2.
- fichero1 -ot fichero2: cierto si fichero1 es más antiguo que fichero2.
- fichero1 -ef fichero2: cierto si fichero1 y fichero2 tienen el mismo número de dispositivo y de nodo-i.
- -z cadena: cierto si la longitud de cadena es cero.
- -n cadena: cierto si la longitud de cadena no es cero.
- cadena1 = cadena2: cierto si las cadenas son iguales.
- cadena1 != cadena2: cierto si las cadenas no son iguales.
- ! expr: cierto si expr es falsa.
- expr1 -a expr2: cierto si expr1 y expr2 son ciertas.
- expr1 -o expr2: si expr1 o expr2 son ciertas.
- arg1 OP arg2: OP es uno de los siguientes valores: -eq, -ne, -lt, -le, -gt, o -ge.

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

- if

```
if [condition]
then
  block
fi
```

- case

```
case $variable in
"valor 1")
  block
;;
"valor 2")
  block
;;
*)
  # booque usado en caso de que no haya entrado a ningún block
;;
esac
```

- while

```
while [condition]
do
  block
done
```

- for

```
## opción 1
for ((i=0;i<10;i++)) #C style
do
  block
done

## opción 2
for i in value1 value2 valueN;
do
  block
done
```

- select

```
select variable in opcion1 opcion2 opcion3
do
  # en $variable está el valor elegido
```

```
  block
done
```

9. ¿Qué acciones realizan las sentencias break y continue dentro de un bloque?¿Qué parámetros reciben?
   - **break:** usado para terminar la ejecución del loop entero después de completar la ejecución de todas las líneas de código anteriores a la sentencia break, una vez llegada a la sentencia break, saldrá del loop y ejecutará el código siguiente fuera del loop.

     Si indico break n donde n es un número, n es la cantidad de bucles de los que saldré, por default es 1.
   - **continue:** similar a la sentencia break pero saldrá de la iteración actual en vez de salir del loop entero.

     Si indico continue n donde n es un número, n es la cantidad de loops de los que saldré, por default es 1.
10. ¿Qué tipo de variables existen?¿Es shell script fuertemente tipado?¿Se pueden definir arreglos?¿Cómo?

    Shell scripting tiene varios tipos de variables
    - Variables de entorno: variables predefinidas por el SO. Por ejemplo $HOME, $?.
    - Variables de posición: variables predefinidas que contienen información sobre los argumentos que se pasan al script. Por ejemplo $1 es el primer argumento.
    - Variables de cadena: variables que contienen cadenas de texto.
    - Variables numéricas: variables que contienen valores numéricos.
    - Variables de arreglo: variables que contienen una lista de valores.
    - Variables de lectura: variables usadas para leer la entrada del usuario.

    Shell scripting no es fuertemente tipado porque por ejemplo no se necesita declarar un tipo de variable antes de usarla.

    ```
    mi_arreglo=(elemento1 elemento2 elemento3 elemento4) #para declarar arreglos
    ```
11. ¿Pueden definirse funciones dentro de un script?¿Cómo?¿Cómo se maneja el pasaje de parámetros de una función a la otra?

    ```
    nombre_de_la_funcion() {
      # Cuerpo de la función
    }

    ## llamada a la función
    saludar
    ```

    Se puede pasar parámetros de una función a otra de igual forma que se pasan los parámetros a una función.

    ```
    funcion1() {
      parametro="$1"
      funcion2 "$parametro"
    }

    funcion2() {
      parametro="$1"
      echo "El parámetro recibido por la función2 es: $parametro"
    }
    ```
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.

    ```
    #!/bin/bash
    echo "Ingrese el primer número"
    read nro1
    echo "Ingrese el segundo número"
    read nro2

    #operaciones
    suma=$(($nro1 + $nro2)) #si no pongo $ por fuera de [] no se asigna
    resta=$(($nro1 - $nro2))
    multiplicacion=$(($nro1 * $nro2))
    division=$(($nro1/$nro2))

    echo "La suma es: $suma"
    echo "La resta es: $resta"
    echo "La multiplicacion es: $multiplicacion"
    echo "La division es: $division"
    if [$nro1 -gt $nro2];then
      echo $nro1; #en la última sentencia antes de else lleva punto y coma para indicar final del bloque
    else
      echo $nro2;
    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.

    ```
    #!/bin/bash

    ## control de que haya dos parámetros
    if [$# -ne 2];then
      exit 1
    ```

```
fi

echo "Ingrese el primer número"
read nro1
echo "Ingrese el segundo número"
read nro2

#operaciones
suma=$(($nro1 + $nro2)) #si no pongo $ por fuera de [] no se asigna
resta=$(($nro1 - $nro2))
multiplicacion=$(($nro1 * $nro2))
division=$(($nro1/$nro2))

echo "La suma es: $suma"
echo "La resta es: $resta"
echo "La multiplicacion es: $multiplicacion"
echo "La division es: $division"
if [$nro1 -gt $nro2];then
  echo $nro1; #en la última sentencia antes de else lleva punto y coma para indicar final del bloque
else
  echo $nro2;
fi
```

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

13. Uso de estructuras de control:

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

```
#!/bin/bash

echo "Se imprimirá los números del 1 al 100 junto con sus cuadrados"

for ((i=1; i<=100;i++))
do
  cuadrado=$((i*i))
  echo "$i, su cuadrado es $cuadrado"
done
```

b. Crear un script que muestre 3 opciones al usuario: Listar, DondeEstoy y QuienEsta. Según la opción elegida 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

listar(){
  ls
}
dondeEstoy(){
  pwd
}
quienEsta(){
  who
}
echo "1: Listar contenido del directorio actual"
echo "2: Mostrar directorio donde estoy ubicado"
echo "3: Informar quiénes están conectados al sistema"
read opcion
case $opcion in
  "1")
    listar
  ;;
  "2")
    dondeEstoy
  ;;
  "3")
    quienEsta
  ;;
  *)
    echo "$opcion no es una opcion válida"
```

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.

```
#!/bin/bash

buscarArchivo(){
  if [ $# -ne 1 ];then
    echo "Debe proporcionarse un nombre de archivo"
  else
    if [ -f $1 ];then
    echo "$1 existe y es un fichero"
    else
      if [ -d $1 ];then
        echo "$1 existe y es un directorio"
      else
        echo "$1 no existe en el sistema"
        mkdir $1
      fi
  fi
}
buscarArchivo "notas.txt"
```

14. Renombrando archivos: haga un script que renombre solo archivos de un directorio pasado como parámetro agregándole 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.

    Por 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

renombrar(){
  if [ $# -ne 3];then
      echo "Debe ingresar dos parámetros"
      return 1
  fi
  case $1 in
    "-a")
      for i in "ls $3";
      do
          mv $i ${i}${2}
      done
    ;;
    "-b")
      for i in "ls $3";
      do
          mv $i ${2}${i}
      done
    ;;
    *)
      echo "Se ingresó un parámetro incorrecto"
}
echo "-a: contatena la cadena al final del nombre del archivo"
echo "-b: concatena la cadena al principio del nombre del archivo"
echo "Ingrese el pámetro"
read opcion
echo "Ingrese la cadena a concatenar"
read cadena
echo "Ingrese el directorio el cual tiene los archivos a modificar"
read directorio
renombrar $opcion $cadena $directorio
```

15. Comando cut: 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.

    **CUT** herramienta de procesamiento de texto para cortar secciones de líneas de texto. Algunos parámetros son

| Parámetro | Función |
|---|---|
| -c | Permite cortar caracteres específicos. Se especifica el rango de caracteres separados por comas. Por ejemplo, el comando “cut -c 1-5 archivo.txt” corta los primeros cinco caracteres de cada línea en el archivo “archivo.txt”. |
| -f | Permite cortar campos o columnas de texto. Se especifica el número de campo, separado por comas si se quieren seleccionar múltiples campos. Además, se puede especificar el delimitador de campos con el parámetro -d. Por ejemplo, el comando “cut -f 2 -d ‘,’ archivo.csv” corta el segundo campo de un archivo CSV usando la coma como deliminador. |
| -n | Ignora líneas que no contienen delimitadores de campo. Por ejemplo, el comando “cut -f 2 -d ‘,’ -n archivo.csv” corta el segundo campo de un archivo CSV, pero ignora las líneas que no tienen una coma. |
| -d | Especifica el delimitador de campos |

```
# Ejemplos con comando CUT

cut -c 1-3 archivo.txt # corta los primeros tres caracteres de cada línea en archivo.txt
cut -f 1,3 -d ':' archivo.txt #corta el primer y tercer
                #campo de cada línea en el archivo archivo.txt, usando ':' como delimitador
cut -f 2 -d ',' -n archivo.csv #corta el segundo campo de cada línea
               #ignorando las que no tienen ',' como delimitador
```

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.

```
#!/bin/bash

if [ $# -ne 1 ]; then
    echo "Debe proporcionar una extensión de archivo como argumento"
    exit 1
fi

EXTENSION=$1
REPORT="reporte.txt"

# Obtener una lista de todos los archivos con la extensión dada y usar cut para obtener el nombre de usuario asociado
# con cada archivo. Usamos sort y uniq para contar cuántos archivos posee cada usuario con la extensión dada.
```

```
find . -type f -name "*.$EXTENSION" -printf "%u\n" | cut -d "/" -f 1 | sort | uniq -c | awk '{print $2, $1}' > $REPORT

echo "Reporte generado y guardado en $REPORT"


#Este script primero verifica si se proporcionó un argumento en la línea de comandos. Luego, almacena la extensión de archivo en una variable llamada EXTENSION y e
# del archivo de informe en una variable llamada REPORT. El script utiliza el comando find para buscar todos los archivos en el directorio actual con la extensión d
#el comando cut para extraer el nombre de usuario asociado con cada archivo. Después de eso, usa los comandos sort y uniq para contar cuántos archivos posee cada us
#extensión dada y guarda los resultados en un archivo de informe llamado reporte.txt. Por último, muestra un mensaje indicando que el reporte fue generado y guardad
```

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 comando tr

#!/bin/bash
for i in `ls -1`
do
  echo $i | tr '[:upper:][:lower:]' '[:lower:][:upper:]' | tr -d '[aA]'
done
```

**TR** usado para hacer traducción o eliminación de caracteres en un texto. El comando toma un conjunto de caracteres de entrada y los transforma en un conjunto de caracteres de salida usando un conjunto de reglas de traducción definidas por el usuario

| Parámetro | Función |
|---|---|
| -d | Elimina los caracteres especificados en el conjunto de entrada en lugar de traducirlos a caracteres de salida. |
| -s | Comprime cualquier ocurrencia consecutiva de los caracteres especificados en el conjunto de entrada a un solo carácter en la salida. |
| -c | Realiza la traducción de cualquier carácter que no esté en el conjunto de entrada en lugar de los que sí están |

```
# EJEMPLOS
# eliminar todos los caracteres en mayúscula de una cadena de texto
echo "Hola Mundo" | tr -d '[:upper:]'
# salida= ola undo

# convertir todos los caractreres en minúscula
echo "Hola mundo" | tr '[:upper:]' '[:lower:]'
# salida= hola mundo

# comprimir ocurrencias de un caracter
echo "¡¡¡Hola!!!" | tr -s '!'
# salida ¡Hola!

# eliminar todos los dígitos de una cadena
 echo "Año 2021" | tr -c '[:digit:]'
# salida Año
```

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.

```
#!/bin/bash/
verificarLogueo(){
  if [ $# -ne 1 ];  then
      return 1
  else
      while true; do
        logueado=$(who| grep "$1")
        if [$logueado];
        then
            echo "Usuario $1 loguado en el sistema"
            return 0
        fi
        sleep 10
      done
  fi

}


echo "Ingrese nombre de usuario"
read nombre
verificarLogueo $nombre
```

19. Escribir un programa de “Menú de comandos amigable con el usuario” llamado menú, el cual, al ser invocado, mostrará un menú con la selección para cada uno de los scripts creados en esta práctica. Las instrucciones de cómo 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. 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 los elementos de la pila.

```
#!/bin/bash/
arreglo=()

push(){
  if [ $# -ne 1];
  then
    echo "No se indicó ningún valor para agregar"
    return 1
  fi
  arreglo+=("$1")
  echo "$1 agregado a la pila"
  return 0
}

pop(){
  if [ ${#arreglo[@]} -eq 0]; then
    echo "La pila está vacía"
    return 1
  fi
  unset arreglo[-1]
  echo "Se sacó un elemento de la pila"
  return 0
}
lenght(){
  echo "La pila tiene ${#arreglo[@]} elementos"
  return 0
}
print(){
  if [ ${#arreglo[@]} -eq 0]; then
    echo "La pila está vacía"
    return 1
  fi
  for i in "${arreglo[@]}";
  do
    echo i
  done
  return 0
}

# main
push 10
pop
lenght
print
```

21. Dentro del mismo script y usando las funciones implementadas:
    - 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.

```
#!/bin/bash
arreglo=()

push(){
  if [ $# -ne 1];
  then
    echo "No se indicó ningún valor para agregar"
    return 1
  fi
  arreglo+=("$1")
  echo "$1 agregado a la pila"
  return 0
}

pop(){
  if [ ${#arreglo[@]} -eq 0]; then
    echo "La pila está vacía"
    return 1
  fi
  for ((i=0;i<3;i++))
  do
    unset arreglo[-1]
    echo "Se sacó un elemento de la pila"
  done
  return 0
}
lenght(){
```

```
  echo "La pila tiene ${#arreglo[@]} elementos"
  return 0
}
print(){
  if [ ${#arreglo[@]} -eq 0]; then
    echo "La pila está vacía"
    return 1
  fi
  for i in "${arreglo[@]}";
  do
    echo i
  done
  return 0
}

# main
## agregar 10 elementos
for ((i=0;i<10;i++));do
  echo "Ingrese un valor para agregar a la pila: "
  read valor
  push $valor
done
# sacar 3 elementos
pop
lenght
print
```

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 3 5 7 9 3 5 4)
productoria(){
  total=0
  for i in "${num[@]}";do
    total=$(($total*$i))
  done
  echo $total
  return 0
}

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 final del recorrido.

```
#!/bin/bash
nros=(10 8 4 9 3 1 2)
cant_impares=0
resto=0
for i in "${nros[@]}";do
  resto=$(($i % 2))
  if [ $resto -eq 0]; then
    echo $i
  else
    ((cant_impares++))
  fi
done
  echo "Hay $cant_impares números impares"
```

24. Dada la definición de 2 vectores del mismo tamaño y cuyas longitudes no se conocen: vector1=(1..N) y vector2=(7..N), complete este script de manera tal de implementar la suma de elemento a elemento entre ambos vectores y que la misma sea impresa en pantalla de la siguiente manera:

```
#!/bin/bash

# Definición de los vectores
vector1=(1 2 3 4 5)
vector2=(7 8 9 10 11)

# Longitud de los vectores
n=${#vector1[@]}

# Vector resultado de la suma
vector_suma=()

# Suma de elemento a elemento
for ((i=0; i<n; i++)); do
  suma=$(( ${vector1[i]} + ${vector2[i]} ))
  vector_suma+=($suma)
done

# Impresión del resultado
echo "Vector 1: ${vector1[@]}"
echo "Vector 2: ${vector2[@]}"
echo "Vector suma: ${vector_suma[@]}"
```

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 parámetro:
    - “-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.

```
#!/bin/bash
usuarios=()

for i in `getent group users | cut -d: -f4 | tr',' '\n'`;do
  usuarios+=$i
done
echo "-b: retorna el elemento de una posición"
echo "-l: longitud del arreglo"
echo "-i: mostrar todos los usuarios"
echo "Ingrese una opción"
read opcion
case $opcion in
"-b")
  echo "Ingrese una posicion"
  read pos
  if [ $pos -le ${#usuarios[@]}];then
    echo ${usuarios[$pos]}
  else
      echo "Posición inválida"
  fi
;;
"-l")
  echo ${#usuarios[@]}
;;
"-i")
  for i in ${usuarios[@]};do
    echo $i
  done
;;
esac
```

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

```
#!/bin/bash
arreglo=() ##hacer de cuenta que tiene elementos
informar(){
  if [ @{#arreglo[@]} -lt 1];then
    echo "El arreglo está vacío"
    return 1
  fi
  for ((i=1;i<${#arreglo[@]};i+=2)) do
      if [ -d ${arreglo[i]}];then
        echo "El elemento $i es un directorio"
      elif [ -f ${arreglo[i]}];then
        echo "El elemento $i es un fichero"
      fi

done

## informar cantidad de archivos o directorios en el sistema
archivos=$(find / -type f | wc -l)
directorios=$(find / -type d | xc -l)
```

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>.

```
#!/bin/bash

inicializar(){
  arreglo=()
}
agregar_elem(){
  if [ $# -ne 1];then
    echo "No se ingresó ningún valor como parámetro"
    return 1
  fi
  arreglo+=($1)
  echo "Se agregó el elemetno $1"
}

eliminar_elem(){
  if [ $1 -ge ${#arreglo[@]}];then
    echo "Posición inválida"
    return 1
  fi
  unset arreglo[$1]
  echo "Se eliminó el elemento de la posición $1"
}

longitud(){
  echo ${#arreglo[@]}
}
```

```
imprimir(){
  for i in ${arreglo[@]};do
    echo $1
  done
}
inicializar_con_valores(){
  if [ $# -ne 2];then
    echo "Cantidad de parámetros inválido"
    return 1
  fi
  nuevo=()
  for ((i=0; i<$1;i++));do
    nuevo[$1]=$1
  done
}
```

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.

```
#!/bin/bash

contarArchivos(){
  archivosCumple=0
  for i in $1/*;do
    if [ -r $i] && [ -w $i] && [ -d $i]; then
        ((archivosCumple++))
        echo $1
    fi
  done

}

buscar(){
  buscar(){
  if [ $# -ne 1];then
    echo "No se indicó ningún nombre de directorio"
    return 1
  fi
  existe=$(find / -name $1)
  if [ -n $existe]; then
    contarArchivos $existe
  else
    echo "No existe ese directorio"
    return 4
  fi
}
```

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 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 validad 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.

```
#!/bin/bash

verArchivo(){
  if [ $# -ne 1];then
    echo "No se especificó un nombre de un archivo"
    return 1
  fi
  for i in ${arreglo[@]};do
    if [ $i = $1]; then
      echo "Archivo encontrado"
      return 0
    fi
  done
  echo "Archivo no encontrado"
  return 5
}


archivos=()
for i in "$(find /home -name "*.doc"` -type f -maxdepth)";;do #separa columnas con / y se queda con la segunda que es el nombre
  nombreArch= $(basename $1)
  archivos+=$nombreArch
done
verArchivo precios.doc
```

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.

```

```