# CAPÍTULO 4: PROGRAMACIÓN ESTRUCTURADA

## 4.1. Cree un script que pida tres números y los muestre ordenados.

```
#!/bin/bash
# Pide tres numeros y los devuelve ordenados

echo -n " Introduce el primer valor: "
read n1
echo -n " Introduce el segundo valor: "
read n2
echo -n " Introduce el tercer valor: "
read n3

if [ $n1 -lt $n2 ]; then
     if [ $n2 -lt $n3 ]; then
        if [ $n1 -lt $n3 ]; then
           echo "$n1 $n2 $n3"
        else
           echo "$n1 $n3 $n2"
        fi
     else
        if [ $n1 -lt $n3 ]; then
                echo "$n1 $n3 $n2"
          else
                echo "$n3 $n1 $n2"
        fi

     fi
else
     if [ $n2 -lt $n3 ]; then
        if [ $n1 -lt $n3 ]; then
           echo "$n2 $n1 $n3"
        else
           echo "$n2 $n3 $n1"
        fi
     else
        if [ $n1 -lt $n3 ]; then
                echo "$n1 $n2 $n3"
          else
                echo "$n3 $n2 $n1"
        fi

     fi
fi
```

Esto mismo se puede conseguir de un modo mucho más sencillo haciendo uso de comandos del sistema, en este caso del comando *sort.*

```
#!/bin/bash
# Pide tres números y los devuelve ordenados
```

```
echo -n " Introduce el primer valor: "
read n1
echo -n " Introduce el segundo valor: "
read n2
echo -n " Introduce el tercer valor: "
read n3

echo Valores ordenados:
# Ahora hacemos un 'echo' con cada uno de los números en
una
# línea diferente y aplicamos el comando 'sort' con un
pipe así:
echo -e "${n1}\n${n2}\n${n3}" | sort -n
# la opción '-n' es para que ordene números, aunque sin
ella también lo hace.
```

4.2. **Cree un script que pida cinco números y muestre los que sean mayores que la media.**

```
#!/bin/bash
#Pide cinco valores numéricos numéricos (validados) hasta
que se introduzca 0
#Tras esto calcula la suma y media de los valores
introducidos

num=0 #número de valores introducidos
suma=0 #suma de los valores introducidos
#vector para almacenar los numeros
declare -a numeros


while [ $num -lt 5  ]; do
      read -p "Deme un número: " n
      if [ $n -eq $n 2> /dev/null ] ; then #numérico
            numeros[$num]=$n
            suma=`expr $suma + $n`
            num=`expr $num + 1`
      fi
done

#se calcula la media
media=$(($suma/$num))
echo
echo Media=$media
echo
echo Valores introducidos mayores a la media:
#se recorre el vector
for i in "${numeros[@]}" ; do
      if [ $i -gt $media ] ; then
            echo $i
      fi
done
```

4.3. **Cree una función que determine si el usuario que ejecuta el script es**

***root*. En caso contrario, finaliza la ejecución del script mostrando un mensaje que informa de tal hecho (modo alternativo de hacerlo al visto en el Ejemplo 4.4). Nota: utilizar el comando *id –un*.**

```
function chkroot()
{
      ROOTUSER_NAME=root
username=`id -nu`
if ["$username" != "$ROOTUSER_NAME" ] ;  then
            echo "Debe ser root para ejecutar el script"
            exit 1
fi
}
```

4.4. **Utilice el valor devuelto por el comando *grep* para comprobar si el usuario actual es controlado localmente.**

```
grep $USER /etc/passwd
if [ $? –ne 0] ; then     #captura de la salida de los
comandos anteriores
echo "Usuario remoto no controlado localmente"
fi
```

4.5. **Utilice una estructura anidada *if…then… elif* para calcular si el año en curso es bisiesto. Nota: Un año es bisiesto si es divisible por 400 o bien es divisible por 4 y no es divisible por 100.**

```
#!/bin/bash
# Comprueba si el año en curso es bisiesto

anio=`date +%Y`

cond1=$(($anio%400))
cond2=$(($anio%4))
cond3=$(($anio%100))


if [ $cond1 -eq "0" ] ; then
      echo Este año es bisiesto
elif [ $cond2 -eq "0" -a $cond3 -ne "0" ] ; then
      echo Este año es bisiesto
else
      echo Este año no es bisiesto
fi
```

4.6. **Repita el ejercicio anterior, pero en este caso utilice una estructura *if…then…else* junto a operadores lógicos *AND* y *OR* para llevar a**

**cabo las comprobaciones. Nota: para llevar a cabo las operaciones aritméticas puede hacer uso del modificador let o de los dobles paréntesis *( ( ) )*.**

```
#!/bin/bash
# Comprueba si el año en curso es bisiesto

anio=`date +%Y`

if (( ($anio % 400) == "0" )) || (( ($anio % 4 == "0" ) &&
($anio % 100 != "0" ) )) ; then
      echo Este año es bisiesto
else
      echo Este año no es bisiesto
fi
```

4.7. **Cree un script que muestre por pantalla un menú con cuatro opciones: *1) Mostrar la fecha*, *2) Mostrar los usuarios conectados*, *3) Mostrar el directorio de trabajo* y *4) Listar el contenido del directorio de trabajo*. Implemente la funcionalidad de cada uno de los apartados.**

```
#!/bin/bash
# menu Implementa un menu usando case

echo -e "\n  MENÚ DE COMANDOS\n"
echo " 1. Mostrar la fecha del sistema"
echo " 2. Mostrar los usuarios conectados"
echo " 3. Mostrar el nombre del directorio de trabajo"
echo " 4. Listar el contenido del directorio de trabajo"

read -p "> " opcion

echo

case "$opcion" in
      1)    date ;;
      2)    who  ;;
      3)    pwd  ;;
      4)    ls   ;;
      *)    echo "Opción incorrecta" ;;
esac
```

4.8. **Cree un script que liste todos los ficheros ejecutables que se encuentren en los directorios de la variable de entorno *PATH*. En caso de que encuentre un directorio que no exista, debe informar de tal hecho.**

```
#!/bin/bash
# listaPath Lista los ficheros ejecutables que se
```

```
encuentren en los directorios
# de la variable de entorno PATH

#Función que lista los ejecutables de un directorio
function listarEjecutables
{
	IFS='
'
	ficheros=$(ls -1 $1)
	for fichero in $ficheros
	do
		pathFichero="$1/$fichero"
		if [ -x $pathFichero ] ; then
			echo $pathFichero
		fi
	done
	IFS=':'
}

IFS=':'


for dir in $PATH
do
	if [ -z "$dir" ] ; then
		echo "Directorio vacío encontrado"
		exit 1
	elif ! [ -d "$dir" ] ; then
		echo "$dir no es un directorio válido"
		exit 1
	else
		listarEjecutables $dir
	fi
done
```

Ideas básicas:

1. El script comienza cambiando la variable IFS por ':' para que el *for* pueda recorrer todos los elementos del PATH
2. En cada iteración del bucle principal se comprueba que el directorio exista y, en ese caso, se llama a la función para listar los ejecutables.
3. Para obtener los ficheros de un directorio en una lista se utiliza el modificador *-1* del comando *ls*, que hace que cada fichero se escriba en una línea distinta. Para ello se debe cambiar el valor de *IFS* por un retorno de carro

## 4.9. Crea un script que dado un directorio, indicado como argumento, muestre el tamaño ocupado por el directorio en bytes, KB y MB.

```
#!/bin/bash
# ocupa.sh Informa del tamaño del directorio que se le
indica

if [ -z $1 ] ; then
       echo "Uso: ocupa Directorio"
       exit 1
fi

#obtiene los directorios y su tamaño en una lista
lista=$(du -k $1)
IFS='
'

for fila in $lista
do
       dir=$(echo $fila|cut -f 2)
       let kb=$(echo $fila|cut -f 1)
       let b=1024*kb
       let mb=kb/1024
       echo "$dir: $mb MB, $kb KB, $b B"
done
```

## 4.10. Implemente un script que calcule la frecuencia de repetición de las palabras de un fichero de texto dado.

```
#!/bin/bash
#Name: frecuenciaPalabra.sh
#Descripción: Calcula la frecuencia de cada palabra en un
fichero de texto

if [ $# -ne 1 ] ; then
       echo "Uso: $0 fichero";
       exit -1
fi

fichero=$1
egrep -o "\b[[:alpha:]]+\b" $fichero | \
awk '{ count[$0]++ }
END{ printf("%-14s%s\n","Palabra","Apariciones") ;
for(ind in count)
{ printf("%-14s%d\n",ind,count[ind]); }
}'
```

**Nota.** Utilice un array asociativo, de modo que la palabra sea el índice del array. De este modo se evita el tener que buscar cada palabra dentro del array.

## 4.11. Implemente un script que reciba un directorio y descomprima todos los ficheros *.zip* del mismo en nuevos subdirectorios con

**nombre igual al del fichero comprimido.**

```
#! /bin/bash
# unzipDir Descomprime todos los archivos zip de un
directorio dado

if [ -z $1 ] ; then
       echo "Uso: unzipDir Directorio"
       exit 1
fi

# Se buscan los ficheros con extensión .zip
for fichero in `find $1 -name "*.zip*" -type f`
do
       # Se elimina la extensión .zip
       directorio=`echo ${fichero} | awk -F '.' '{print
$1}'`
       echo "Directorio: $directorio"

       # Se crea el directorio
       mkdir $directorio

       # Copiado del archivo zip al nuevo directorio y
acceso al mismo
       cp ${fichero} ${directorio}
       cd $direccion

       # Descompresión del archivo en el nuevo directorio
       unzip ${directorio}/$(echo ${fichero##/*/})
done
```

## 4.12. Implemente un juego que genere un número aleatorio entre 0 y 99 y que se lo pregunte al usuario hasta que éste lo acierte o falle 10 veces.

```
#!/bin/bash
# adivinaNumero.sh Juego para adivinar un número
comprendido entre 0 y 99

let intentos=10
let solucion=$RANDOM%100

while ((intentos-->0))
do
       read -p "Indique un número: " numero
       if ((numero==solucion)) ; then
              echo "Enhorabuena!, el número era $solucion"
              exit
       elif ((numero<solucion)) ; then
              echo "El número buscado es mayor"
       else
              echo "El número buscado es menor"
       fi
```

```
done

echo "Lo siento. Ha superado los 10 intentos"
```

Ideas básicas:

1. Al estar trabajando con expresiones aritméticas la comparación de igualdad en el if utiliza == y no =
2. La expresión intentos-->0 comprueba primero que la variable intentos sea mayor de 0 y posteriormente decrementa el valor de dicha variable con el operador --

## 4.13. Implemente un script que acepte valores enteros desde teclado en el rango [0-100] hasta que se introduzca el carácter *q*. Tras ello calcula la media de los valores introducidos.

```
#! /bin/bash
# media Calcula la media aritmética de la serie de números
introducidos

NUMERO="0"
MEDIA="0"
SUMA="0"
CONTADOR="0"

while true ; do
      echo -n "Introduzca un número en el rango [0 - 100]
('q' para quitar): "
      read NUMERO

      if (("$NUMERO" < "0")) || (("$NUMERO" > "100")) ;
then
            echo "Debe introducir un valor en el rango
[0-100]"
      elif [ "$NUMERO" == "q" ] ; then
            break
      else
            SUMA=$[$SUMA + $NUMERO]
            CONTADOR=$[$CONTADOR + 1]
            MEDIA=$[$SUMA / $CONTADOR]
      fi
done

echo "La media es $MEDIA"
```

**Nota.** Recuerde que los operadores aritméticos de Bash trabajan con números enteros. Por lo tanto, la media anterior es truncada al entero inferior.

4.14. **Implemente un script que solicite una ruta absoluta para un fichero y que almacene en el mismo todo lo introducido por la entrada estándar hasta que se pulse Ctrl+C.**

```
#! /bin/bash
# escribirFichero.sh Lee datos desde la entrada estándar y
los almacena en un fichero
# cuya dirección absoluta se le indica

echo -e "Indique la ruta absoluta del fichero a crear:"
read fichero

echo -e "Introduzca el contenido (Ctrl+C para terminar):"

while read linea ; do
      echo $linea >> $fichero
done
```

4.15. **Implemente un script que solicite la ruta absoluta de un fichero y muestre el contenido del mismo por la salida estándar. Utilice para ello un bucle *while* y una redirección mediante el comando *exec*.**

```
#! /bin/bash
# leerFichero.sh Muestra por la salida estándar el
contenido del fichero que
# se le indica

echo -e "Indique la ruta absoluta del fichero a mostrar:"

read fichero
exec <$fichero # redirección de la entrada estándar a
fichero

while read linea ; do
      echo $linea
done
```

4.16. **Cree un script que monitorice cada 5 segundos el tamaño del fichero de log (*/tmp/logfile)* y una vez que alcance un tamaño de 2000 bytes, realice una copia del mismo que contenga la fecha en que fue realizada. Para llevar a cabo la temporización puede utilizar el comando *sleep*, que detiene la ejecución del proceso durante el periodo (en segundos) que se le indique.**

```
#! /bin/bash
# monitorLog.sh Monitoriza el tamaño del fichero de log y
realiza una copia
# del mismo cuando su tamaño supera los 2000 bytes

fichero=/tmp/logfile

until [ $(ls -l $fichero | awk '{print $5}') -gt 2000 ]
        do
            sleep 5
        done
date=`date +%s`
cp $fichero "$fichero-"$date.bak
```

4.17. **Realice un script que monitorice la disponibilidad de una máquina red, cuya dirección IP se solicite. Utilice para ello un bucle *until* que finalice cuando la máquina responda a *ping*. El test a la máquina se debe llevar a cabo cada 60 segundos.**

```
#! /bin/bash
# testIP.sh Monitoriza la disponibilidad de una máquina.
# El bucle acaba cuando la máquina responda a ping

read -p "Introduzca la dirección IP: " ipadd

until ping -c 1 $ipadd
do
        sleep 60
done
```

Este script puede ser de gran utilidad, por ejemplo, para realizar una conexión remota mediante *ssh* en el momento en que la máquina remota responda.

4.18. **Implemente el comando de las distribuciones BSD lock. Al principio solicita una clave secreta y utiliza una estructura de control until para bloquear el terminal hasta que se vuelva a introducir dicha clave. Este script permite abandonar el terminal abierto durante cortos periodos de tiempo de una forma segura.**

```
#! /bin/bash
# locktty Bloquea el terminal hasta que se introduzca la
clave secreta

trap '' 1 2 3 18
stty -echo
echo -n "Introduzca la palabra secreta: "
read key_1
```

```
echo
echo -n "Repita la palabra secreta: "
read key_2
echo
key_3=
if [ "$key_1" = "$key_2" ] ; then
       tput clear
       until [ "$key_3" = "$key_2" ]
       do
              read key_3
       done
else
       echo "locktty: Las palabras dadas no coinciden"
1>&2
fi
stty echo
```

Si olvida la clave necesita iniciar sesión en otra terminal y eliminar el proceso *locktty* mediante el comando *kill*.

## 4.19. Cree un script que muestre un menú por pantalla hasta que se pulse la opción de salir. Las opciones de dicho menú serán las siguientes:

1. **Mostrar la fecha del sistema.**
2. **Mostrar información sobre qué usuarios han iniciado sesión y qué están haciendo. Para ello utilice el comando w.**
3. **Mostrar los 10 procesos que consumen más memoria. Para ello utilice el comando ps.**
4. **Mostrar los 10 procesos que consumen más CPU. Para ello utilice el comando ps.**
5. **Mostrar el estado de la red. Para ello utilice el comando netstat.**
6. **Salir del menú.**

**Además, debe evitar su finalización de otro modo que no sea mediante la opción del menú.**

```
#!/bin/bash
# menu.sh Muestra varios mensajes por pantalla.
# captura CTRL+C, CTRL+Z y quit usando el comando trap
trap '' SIGINT
trap ''  SIGQUIT
trap '' SIGTSTP

# Muestra un mensaje y pausa la ejecución
pause(){
       local m="$@"
       echo "$m"
       read -p "Presione [Enter] para continuar..." key
}

# bucle infinito
while :
do
       # Muestra el menú
       clear
       echo "--------------------------------"
       echo "      MENU DE OPCIONES"
       echo "--------------------------------"
       echo "1. Mostrar la fecha del sistema"
       echo "2. Mostrar usuarios y uso"
       echo "3. Top 10 de consumo de memoria"
       echo "4. Top 10 de consumo de CPU"
       echo "5. Mostrar estadísticas de red"
       echo "6. Salir"
       echo "--------------------------------"
       read -r -p "Introduzca su opción [1-6] : " c
       # take action
       case $c in
             1) pause "$(date)";;
             2) w| less;;
             3) echo '*** Top 10 de consumo de memoria:';
ps -auxf | sort -nr -k 4 | head -10;
                echo;  pause;;
             4) echo; echo '*** Top 10 de consumo de
CPU:';ps -auxf | sort -nr -k 3 | head -10;
                echo;  pause;;
             5) netstat -s | less;;
             6) break;;
             *) Pause "Opción incorrecta. Recuerde [1-6]"
       esac
done
```

## 4.20. Cree un script que realice la ordenación de un vector de elementos mediante el algoritmo de la burbuja.

```
#!/bin/bash
# Ordenación por burbuja

function burbuja {
       lista=$1
       tam=${#lista[@]}
```

```
      for i in $(seq 1 $[$tam-1]); do
            for j in $(seq 0 $[$tam - $i - 1]); do
            if [ ${lista[$j]} -gt ${lista[$j+1]} ] ; then
                        k=${lista[$[$j+1]]}
                        lista[$j+1]=${lista[$j]}
                        lista[$j]=$k
                  fi
            done
      done
}

lista=(5 4 3 2 1)

burbuja $lista

for i in ${lista[@]};do
echo $i
done
```

## 4.21. Cree un script que realice la ordenación de un vector de elementos mediante el algoritmo Quicksort.

```
#!/bin/bash
# SCRIPT : quicksort.sh Ordena una lista de números
mediante el Algoritmo Quicksort
# USO : quicksort.sh


imprimirNumeros()
{
   echo ${ARRAY[*]}
}

quicksort()
{

    local array=( `echo "$@"` )
    local -a l
    local -a g
    local -a e
    local x=

    if [ ${#array[@]} -lt 2 ]; then
        echo -n ${array[@]}
    else
        local pivote=${array[0]}

        for x in ${array[@]}
        do

          if [ $x -lt $pivote ]
          then
              l=( ${l[@]} $x )
          elif [ $x -gt $pivote ]
```

```
          then
              g=( ${g[@]} $x )
          else
              e=(${e[@]} $x)
          fi

        done

      echo "`quicksort "${l[@]}"` ${e[@]} `quicksort
"${g[@]}"`"

    fi
}

################################################################
############
#                       DECLARACIÓN DE VARIABLES
#
################################################################
############

clear

echo "Introduzca los números a ordenar : "

read -a ARRAY

count=${#ARRAY[@]}

################################################################
############
#                       Llamadas a funciones
#
################################################################
############

echo "-----------------------------------------------------
----------"

echo "Números antes de la ordenación:"

imprimirNumeros


echo "Números tras la ordenación: "

quicksort "${ARRAY[@]}"

echo "-----------------------------------------------------
----------"
```

4.22. **Cree una función recursiva denominada *makepath* que, dado un *pathname,* cree todos los componentes del mismo como directorios. Por ejemplo, el comando *makepath a/b/c/d* creará los directorios *a,***

***a/b, a/b/c* y *a/b/c/d*. Nota: La orden *mkdir –p* crea los directorios usando este procedimiento.**

```
#!/bin/bash
# makepath.sh Crea la estructura de directorios indicada

function makepath()
{
      if [[ ${#1} -eq 0 || -d "$1" ]] ; then
            return 0 # salir
      fi

      if [[ "${1%/*}" = "$1" ]] ; then
            mkdir $1
            return $?
      fi

      makepath ${1%/*} || return 1
      mkdir $1
      return $?
}
```

El algoritmo recursivo empleado es el siguiente:

- Se examinan los argumentos del *path*. Si la cadena es nula o se refiere a un directorio existente, no se realiza acción alguna.
- Si el *path* se refiere a un único componente, lo crea mediante la orden *mkdir* y sale.
- En otro caso, se llama a la función *makepath* usando el prefijo del argumento original. Este paso crea todos los directorios hasta el último componente, que se puede crear con *mkdir*.

## 4.23. Cree un script para la búsqueda de elementos en un vector mediante el algoritmo de Búsqueda Binaria.

**Nota: Recuerde que para poder llevar a cabo la búsqueda binaria los elementos del vector deben estar ordenados. Puede utilizar el algoritmo Quicksort anterior para lograrlo.**

```
#! /bin/bash
# busquedaBinaria.sh
# Algoritmo de Búsqueda binaria

BusquedaBinaria()
{
    estado=-1
    i=1
    array=($(echo "$@"))
    indiceInferior=0
```

```
    indiceSuperior=$((${#array[@]}-1))

    while [ $indiceInferior -le $indiceSuperior ] ; do

        indiceMedio=$(($indiceInferior+($indiceSuperior-
$indiceInferior)/2))
           elementoMedio=${array[$indiceMedio]}

        if [ $elementoMedio -eq $elementoBuscado ] ; then
            estado=0
            busquedas=$i
            return
        elif [ $elementoBuscado -lt $elementoMedio ] ;
then
            indiceSuperior=$(($indiceMedio-1))
        else
            indiceInferior=$(($indiceMedio+1))
        fi

        let i++

    done
}

#Se crea un vector de elementos
elementos=( `seq 0 100` )

#totalElementos=${#a[@]}
elementoBuscado=51

BusquedaBinaria "${elementos[@]}"

if [ $estado -eq 0 ] ; then
         echo "Elemento $elementoBuscado encontrado tras
$busquedas iteraciones"
else
         echo "$elementoBuscado no encontrado en el
vector"
fi
```