* **Escriba un script que reciba dos nombres de directorios como parámetros. Deberá validar que se reciben 2 y además que el primer parámetro sea un directorio existente. El script deberá copiar todos los archivos del primer directorio en el directorio destino recibido como segundo parámetro, para los cuales el usuario tenga permisos de lectura (no deberá validar subdirectorios). En el caso de que el directorio destino (segundo parámetro) no exista, deberá crearlo.**
* **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) 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.**
* **Realice un script que simule el comportamiento de una estructura de cola 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

Vector=()

Index=0

Push() {

Vector[index]=$1

Let index++

}

Pop() {

Unset vector[index]

Let index--

}

Function length {

Return ‘expr $index + 1’

}

Print() {

For elem in ${vector[\*]}; do

echo $elem

done

}

* **Suponga que es administrador de un servidor de correo y se le ha encargado un listado con todos los nombres de usuarios que poseen cuenta de mail. Se sabe que un usuario que tiene cuenta de mail posee un directorio llamado mailDir en su directorio personal. Además de tener una cuanta de mail los usuarios del listado deben tener definido a bash como su Shell por defecto. Para el listado solicitado se debe generar un archivo llamado listado.txt en el /var.**
* **Se desea saber el uso del consumo de recursos del sistema por parte de los usuarios. Desarrolle un script que se correrá ingresando como argumento los nombres de los usuarios de los cuales se necesita obtener dicha información, no se sabe cuántos son con anticipación.**

**Para la realización del script debe definir las siguientes funciones:**

**Esta\_logueado(): devuelve true si el usuario pasado como argumento esta logueado**

**Cant\_procesos(): devuelve la cantidad de procesos que está corriendo un usuario**

**Uso\_de\_procesos(): devuelve true si el usuario pasado como argumento, usando la función cant\_procesos, está corriendo más de 100 procesos.**

**Utilizando la funcionalidad anterior, el script debe guardar en u archivo de log, en una ubicación apropiada del file system, el nombre de los usuarios que estén corriendo más de 100 procesos junto con la fecha de ejecución del script.**

* **Escribir un script que reciba como parámetro el string “riRRsroersgerRnrRiarl” y lo convierta a “isoesgenial”. El string convertido deberá ser guardado en el archivo frase.txt ubicado en el path personal del usuario que ejecuta el script. Debe verificar que el string llegue efectivamente como parámetro.**
* **Realice un script que verifique cada 5 segundos si el proceso apache se está ejecutando en el sistema. El script debe correr indefinidamente hasta que verifique que en 10 ocasiones ha encontrado el proceso apache corriendo, lo cual debe causar la terminación del proceso, informar dicha acción en pantalla y posteriormente causar la terminación del script con condición de retorno 50.**

#!/bin/bash

Contador=0

While [ $contador –lt 10 ]; do

Ver=$(ps -C apache | wc –l )

If [[ $ver –gt 1 ]]; then

Let “contador++”

Fi

Sleep 5

Done

Killall apache

Echo “El proceso apache fue terminado”

Exit 50

* **Realice un script que agregue a un arreglo los nombres de los archivos del directorio /etc cuya terminación sea .conf e implemente las siguientes funciones:**

**Cantidad: imprime la cantidad de archivos del /etc con terminación .conf**

**verArchivos: imprime los nombres de todos los archivos /etc con terminación .conf**

**existe: recibe como primer parámetro el nombre de un archivo con terminación .conf e imprime en pantalla si existe en el /etc o no**

**eliminar: recibe como primer parámetro el nombre de un archivo con terminación .conf y como segundo parámetro la cadena lógico o físico. Si el segundo parámetro es lógico, solo borra la entrada del arreglo, si es físico borra la entrada del arreglo y en el FileSystem informando en cada caso la acción realizada**

* **Escriba un script que reciba una cantidad desconocida de parámetros al momento de su invocación (debe validar que al menos se recibe 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, 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.**
* **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.**
* **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 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 en una posición valida.**

**Longitud: imprime la longitud del arreglo en pantalla.**

**Imprimir: imprime todos los valores del arreglo en pantalla.**

**inicializarConValores <parametro1> <parametro2>: crea un arreglo con longitud <parametro1> y en todas las posiciones asigna el valor <parametro2>.**

#!/bin/bash

Inicializar() {

array=()

índice=0

}

Agregar\_elem() {

Array[índice]=$1

Let índice++

}

Eliminar\_elem() {

If [$1 -le ${#array[\*]}]; then

Unset array[pos]

Let índice—

Else

Echo “la posición no existe dentro del arreglo”

fi

}

Longitud() {

Echo ${#arreglo[\*]}

}

Imprimir() {

For elem in ${#arreglo[\*]}; do

Echo elem

Done

}

inicializarConValores() {

array=()

for indice in $1; do

array[índice]=$2

done

}

* **Se desea implementar un reproductor multimedia en bash para archivos de audios y videos, que debe funcionar de la siguiente manera:**

**El script recibe como parámetros un flag que indica el tipo de los contenidos a reproducir (-m o -v, para música o video respec), y una lista de rutas a los archivos que se quieren reproducir.**

**Primero se deberá validar que se indique el tipo de los archivos (por simplicidad serán o bien todos de audio, o todos de video, pero no mezclados), y que se reciba al menos un archivo a reproducir (el número de archivos a reproducir es desconocido de antemano).**

**Luego se comenzará la reproducción de los archivos en el orden en que se los recibió, validando antes de reproducir cada archivo si este existe en el file system. En caso de no existir, se debe guardar el error en el archivo /var/log/reproductor.error consignando la fecha del error y la ruta al archivo que no existe. Si el archivo existe, se lo debe reproducir con el comando vic, pasándole como parámetros la ruta al archivo y el tipo de reproducción (-m o -v). Por ejemplo: vic mitema.mp3 -m**

* **Realice un script que agregue todos los nombres de usuario del sistema a un arreglo e implemente las siguientes funciones:**

**Existe <parametro1>: imprime el índice dentro del arreglo donde está el usuario <parametro1>, solo si existe en el arreglo. Adicionalmente, debe retornar 0 si el usuario existe en el arreglo o 1 en caso contrario.**

**Eliminar\_usuario <parametro1>: si el usuario pasado como parámetro existe en el arreglo, lo elimina del mismo. Caso contrario retorna código de error 2.**

**Usuarios\_con\_patron<parametro1>: recorrer el arreglo e imprime en pantalla los nombres de los usuarios cuyos caracteres aparece el patrón <parametro1>.**

**Cantidad: imprime la cantidad total de usuarios en el arreglo.**

**Usuarios: imprime todos los nombres de los usuarios que están en el arreglo.**

**Luego de definir las funciones, deberá proveer un breve cuerpo del script que los utilice, a modo de ejemplo.**

#!/bin/bash

Cant=0

While read line

Do let “cant++”

User=$(echo $line | cut –d\: -f1)

Usuarios[$cant]=$user

Done < “/etc/passwd”

Function existe {

Esta=1

For i in “${!usuarios[@]}”; do

If [[ “${usuarios[$i]}” = “${1}” ]]; then

Echo “El índice es ${i}”;

Esta=0

Fi

Done

Echo $esta

}

Function eliminar\_usuario {

Esta=1

For i in “${!usuarios[@]}”; do

If [[ “${usuarios[$i]}” = “${1}” ]]; then

Usuarios=( “${usuarios[@]/$usuarios[$i]}” )

Esta=0

Fi

Done

If [ $esta –eq 1 ]; then

Echo “Codigo de error 2”

Fi

}

Function usuarios\_con\_patron{

For i in “${!usuarios[@]}”; do

If [[ “${usuarios[$1]}” == \*{1}\* ]]; then

Echo “El usuario ${usuarios[i]} contiene la cadena ingresada”

Fi

Done

}

Function cantidad{

Echo “Hay ${#usuarios[@]}” usuarios”

}

Function usuarios{

For i in “${usuarios[@]}”; do

Echo “Usuario numero $i: ${usuarios[$i]}”

Done

}

Echo “Elige una opción: “

Select opción in existe eliminar\_usuario usuarios\_con\_patron cantidad usuarios salir; Do

Case $opcion in

Existe)

Read –p “Ingrese un usuario: “ dato

Existe $dato ;;

Eliminar\_usuario)

Read –p “Ingrese un usuario: “ dato

Eliminar\_usuario $dato ;;

Usuarios\_con\_patron)

Read –p “Ingrese un usuario: “ dato

Usuarios\_con\_patron $dato ;;

Cantidad)

Cantidad ;;

Usuarios)

Usuarios ;;

Salir)

Break ;;

Esac

done

* **Escribir un script que reciba como primer parámetro un directorio, luego uno o más nombres de archivos. Debemos validar que el directorio indicado exista y que sea un directorio. Luego por cada nombre de archivo ingresado debemos validar que el mismo exista, y en caso de existe, si es un archivo, informar si poseemos permiso de ejecución sobre el mismo y si es un directorio, informar si poseemos permiso de escritura sobre el mismo.**

#!/bin/bash

If [ -d $1]; then

For 1 in $\*; do

If [ -e $1 ]; then

If [ -f $1 ]; then …

If [ -x $1 ]; then

Echo “el archivo tiene permiso de ejecución”

Else

Echo “el archivo no tiene permiso de ejecución”

Fi

elif [ -d $1 ]; then

if [ -w $1 ]; then

Echo “la carpeta tiene permiso de escritura”

Else

Echo “la carpeta no tiene permiso de escritura”

Fi

fi

Else

Echo “no existe el directorio o archivo”

Fi

Done

Else

Echo “El parámetro no existe o no es un directorio”

fi

* **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

if [ $# -ne 1 ]; then

echo "La Cantidad de parámetros es incorrecta"

exit 1

fi

while true; do

seLogueo=`who | grep $1 | wc -l`

if [ $seLogueo != 0 ]; then

echo "usuario $1 logueado"

exit 0

fi

sleep 10

done