

- 1) Shell Scripting se refiere a la práctica de escribir secuencias de comandos (scripts) en un lenguaje de shell (como Bash en sistemas Unix/Linux) para automatizar tareas y realizar operaciones en un sistema operativo. Estos scripts están compuestos por una serie de comandos y sentencias que se ejecutan secuencialmente, y no es necesario compilarlos como en lenguajes de programación de alto nivel como C o Java. Aquí te proporciono más detalles sobre el Shell Scripting.

Los scripts están orientados a automatización de tareas, aplicaciones interactivas, aplicaciones con interfaz gráfica.

No se Compilan: A diferencia de los lenguajes de programación compilados como C o C++, los scripts de shell no se compilan. Se escriben en texto plano y se ejecutan directamente por el intérprete de shell (como Bash). Esto hace que sean fáciles de crear, modificar y ejecutar sin necesidad de un proceso de compilación.

- 2) El echo te permite ingresar un determinado mensaje la cual se mostrará en el intérprete. El comando read en Linux Debian se utiliza para leer una línea de texto desde la entrada estándar (generalmente el teclado) y asignar el valor ingresado a una variable en un script de Shell. Este comando es especialmente útil cuando se desea interactuar con el usuario y capturar su entrada.

- a) Los comentarios se indican con "#".
- b) nombre="Juan" # Declaración y asignación a una variable llamada "nombre"
echo "Hola, \$nombre" # Referencia de la variable "nombre"

- 3) a) chmod 766 mostrar.sh

- b) y c)

```
luciano@Debian:~/practica-shell-script$ vi mostarr.sh
luciano@Debian:~/practica-shell-script$ bash mostarr.sh
Introduzca su nombre y apellido:
pepe wagner
Fecha y hora actual:
Sun Sep 10 09:18:55 PM -03 2023
Su apellido y nombre es:
wagner pepe
Su usuario es: 'whoami'
Su directorio actual es:
luciano@Debian:~/practica-shell-script$
```

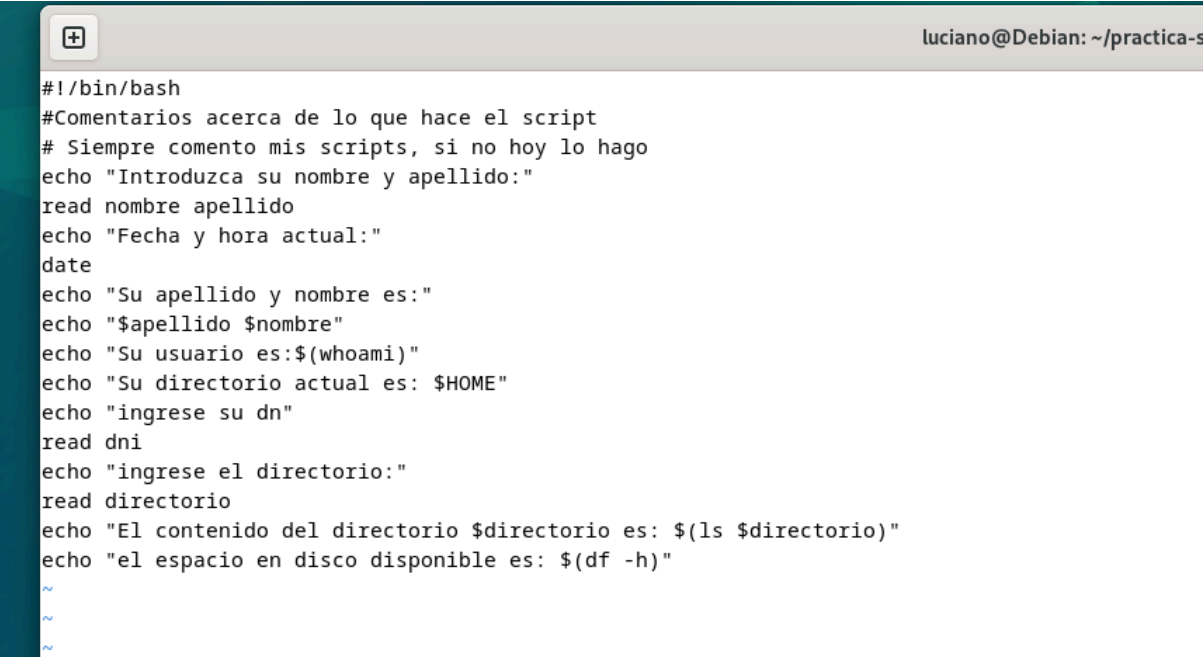
(ya le cambie los permisos dsp de esa screen y lo hice con ./ y funca igual. (antes sin los permisos no)

- d) El reemplazo de comandos permite utilizar la salida de un comando como si fuese una cadena de texto normal. (el 'whoami' con esas comillas simples si te fijas me hacia q se me imprima "whoami" y no mi usuario por ello lo cambie por \$(whoami) y anduvo, no se pq no funcaba el otro, lo abré puesto mal, debería funcionar, igual ambas formas son lo mismo).

e) en el que me pide que le diga el contenido de un directorio le tengo que pasar si o si el directorio completo no? se asume que se va a hacer asi no? pq si le mando cualquier cosa no lo hace jaja. Me dijo que en este punto no pasa nada, pero la idea es siempre verificar que si te pide mandar un directorio, que ese directorio sea valido, que si te pide ingresar por parametros x valores que esos x valores sean enviados, que si vas a trabajar con valores numericos, verifiques a los mismos (tengo un codigo en la notebook en un script de como hacerlo sino buscalo por internet)
asi podes verificar que es un numero:

```
expresion="^[0-9]+$"
```

```
if [[ $1 =~ $expresion ]]; then
    echo "verdadero"
else
    echo "falso"
fi
```



```
#!/bin/bash
#Comentarios acerca de lo que hace el script
# Siempre comento mis scripts, si no hoy lo hago
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: $HOME"
echo "ingrese su dn"
read dni
echo "ingrese el directorio:"
read directorio
echo "El contenido del directorio $directorio es: $(ls $directorio)"
echo "el espacio en disco disponible es: $(df -h)"
~
~
~
```

```

luciano@Debian:~/practica-shell-script$ ./mostarr.sh
Introduzca su nombre y apellido:
wagner luciano
Fecha y hora actual:
Sun Sep 10 10:01:19 PM -03 2023
Su apellido y nombre es:
luciano wagner
Su usuario es:luciano
Su directorio actual es: /home/luciano
ingrese su dn
45571936
ingrese el directorio:
/home/luciano/pruebadirec
El contenido del directorio /home/luciano/pruebadirec es: procesos
el espacio en disco disponible es: Filesystem      Size  Used Avail Use% Mounted on
udev                1.9G   0    1.9G   0% /dev
tmpfs               392M   1.2M  391M   1% /run
/dev/sda1           24G    5.6G   17G   25% /
tmpfs               2.0G   0    2.0G   0% /dev/shm
tmpfs               5.0M   8.0K   5.0M   1% /run/lock
tmpfs               392M  160K   392M   1% /run/user/1000
luciano@Debian:~/practica-shell-script$

```

4) con los \$1, \$2, \$3.....

\$HOME contiene el directorio de inicio del usuario en el que estas usando en ese momento.

\$\$ Contiene la cantidad de argumentos recibidos.

^ \$* contiene la lista de todos los argumentos.

^ \$? contiene en todo momento el valor de retorno del último comando ejecutado. Lo usas en los retornos de una función o justo luego de ejecutar un archivo.sh. (lo que retorna el exit y el return va a estar ahí. El exit lo podes no poner, pero nos dijeron que lo uses igual).

5) El exit se usa para terminar un script y puede devolver un valor entre 0 y 255. El 0 indica que el script se ejecutó de forma exitosa. Luego un valor distinto de 0 indica un código de error. El valor del exit se consulta con \$?.

En un exit la forma es: **exit n (siendo n un numero entre el 0 y el 255).**

chat gpt:

El comando `exit` en Linux es utilizado para finalizar la ejecución de un script o de una terminal. Puede recibir un valor como parámetro que indica el estado de salida del proceso. Aquí te explico su funcionalidad y significado:

- Sin parámetro: Cuando se utiliza `exit` sin ningún parámetro, simplemente finaliza la ejecución del script o de la terminal en la que se encuentra. Por ejemplo, si ejecutas `exit` en una terminal, esta se cerrará.
- Con parámetro: Puedes proporcionar un valor numérico como argumento al comando `exit`, como `exit 0` o `exit 1`. Este valor numérico se utiliza para indicar el estado de salida del proceso. En general, un valor de 0 se utiliza para indicar una finalización exitosa, mientras que otros valores (diferentes de 0) se utilizan para indicar errores o finalizaciones anormales. Los valores de estado pueden ser utilizados por otros programas o scripts para tomar decisiones basadas en el resultado de la ejecución anterior.

6)

El comando ``expr`` en Linux Debian permite realizar diversas operaciones de evaluación de expresiones. Puedes utilizar ``expr`` para realizar las siguientes operaciones básicas:

1. ****Suma:**** Puedes sumar dos números utilizando el operador ``+``. Por ejemplo:

```
``bash
expr 5 + 3
``
```

2. ****Resta:**** Puedes restar dos números utilizando el operador ``-``. Por ejemplo:

```
``bash
expr 10 - 4
``
```

3. ****Multiplicación:**** Puedes multiplicar dos números utilizando el operador ``*``. Por ejemplo:

```
``bash
expr 6 \* 7
``
```

Nota: El operador de multiplicación ``*`` debe escaparse con una barra invertida (``\``) para evitar que sea interpretado como un carácter comodín por la shell.

4. ****División:**** Puedes dividir dos números utilizando el operador ``/``. Por ejemplo:

```
``bash
expr 20 / 4
``
```

5. ****Módulo:**** Puedes calcular el módulo (resto) de la división de dos números utilizando el operador ``%``. Por ejemplo:

```
``bash
expr 15 % 4
``
```

6. ****Comparaciones:**** Puedes realizar comparaciones numéricas utilizando operadores de comparación como ``>``, ``<``, ``>=``, ``<=``, ``=`` y ``!=``. Por ejemplo:

```
``bash
expr 5 \> 3
``
```

Este comando evaluará si 5 es mayor que 3 y devolverá 1 si es verdadero o 0 si es falso.

7. ****Longitud de cadenas:**** Puedes obtener la longitud de una cadena utilizando el operador ``length``. Por ejemplo:

```
```bash
expr length "Hola"
```
```

Esto devolverá la longitud de la cadena "Hola", que es 4.

8. ****Índice de caracteres:**** Puedes buscar un carácter o una subcadena dentro de una cadena utilizando el operador ``index``. Por ejemplo:

```
```bash
expr index "Linux" "nux"
```
```

Esto buscará la subcadena "nux" dentro de "Linux" y devolverá la posición de inicio de la subcadena (en este caso, 2).

Estas son algunas de las operaciones básicas que puedes realizar con el comando ``expr``. Puedes combinar estas operaciones y utilizarlas en scripts de shell para realizar cálculos y evaluaciones numéricas.

7) En general, con el "test expresión" puedes evaluar expresiones vinculadas a archivos, caracteres y números devolviendo true (0) o false (1). (por ahí más cosas pueden evaluar, pero esto es lo que dimos).

El comando ``test`` o su equivalente ``[]`` en Linux Debian se utiliza para evaluar expresiones y generar un valor de retorno que generalmente es ``true`` (0) o ``false`` (1) basado en el resultado de la evaluación de la expresión. Las expresiones que se pueden utilizar con el comando ``test`` o ``[]`` pueden ser de varios tipos, incluyendo evaluaciones de archivos, evaluaciones de cadenas de caracteres y evaluaciones numéricas. A continuación, se describen algunas de las expresiones más comunes:

****Evaluación de archivos:****

Aca siempre le tenes que pasar la ruta, ya que por ejemplo en el caso del `-e` te dice si ese archivo existe en la RUTA especificada ó si vos solo le indicas el nombre **se buscará ese archivo en el directorio actual desde donde se está ejecutando el script o el comando. Si el archivo no se encuentra en el directorio actual, la verificación fallará y devolverá un valor falso.**

- ``-e archivo``: Verifica si el archivo existe.
- ``-f archivo``: Verifica si el archivo existe y es un archivo regular (no un directorio u otro tipo de archivo especial).
- ``-d directorio``: Verifica si el directorio existe.
- ``-s archivo``: Verifica si el archivo existe y tiene un tamaño mayor que cero (no está vacío).
- ``-r archivo``: Verifica si el archivo existe y tiene permisos de lectura. **(con respecto al usuario en donde me encuentro ejecutando el script por ej)**
- ``-w archivo``: Verifica si el archivo existe y tiene permisos de escritura.
- ``-x archivo``: Verifica si el archivo existe y tiene permisos de ejecución.

Ejemplos de evaluación de archivos:

```
```bash
Verifica si el archivo "miarchivo.txt" existe
if [-e "miarchivo.txt"]; then
 echo "El archivo existe."
fi

Verifica si el directorio "miproyecto" existe
if [-d "miproyecto"]; then
 echo "El directorio existe."
fi
```
```

****Evaluación de cadenas de caracteres:****

- `cadena1 = cadena2`: Verifica si las dos cadenas son iguales.
- `cadena1 != cadena2`: Verifica si las dos cadenas son diferentes.
- `-n cadena`: Verifica si la cadena no está vacía (su longitud es mayor que cero).
- `-z cadena`: Verifica si la cadena está vacía (su longitud es cero).

Ejemplos de evaluación de cadenas de caracteres:

```
```bash
Verifica si las cadenas son iguales
if ["hola" = "hola"]; then
 echo "Las cadenas son iguales."
fi

Verifica si una cadena no está vacía
if [-n "texto"]; then
 echo "La cadena no está vacía."
fi
```
```

****Evaluaciones numéricas:****

- `n1 -eq n2`: Verifica si los números n1 y n2 son iguales.
- `n1 -ne n2`: Verifica si los números n1 y n2 son diferentes.
- `n1 -lt n2`: Verifica si n1 es menor que n2.
- `n1 -le n2`: Verifica si n1 es menor o igual que n2.
- `n1 -gt n2`: Verifica si n1 es mayor que n2.
- `n1 -ge n2`: Verifica si n1 es mayor o igual que n2.

Ejemplos de evaluaciones numéricas:

```
```bash
Verifica si un número es igual a otro
```

```
if [5 -eq 5]; then
 echo "Los números son iguales."
fi
```

```
Verifica si un número es menor que otro
if [3 -lt 6]; then
 echo "El primer número es menor que el segundo."
fi
...
```

8)

Las estructuras de control en el scripting de shell (bash) permiten controlar el flujo de un programa o script. A continuación, se presenta la sintaxis de las estructuras de control más comunes en Bash:

**\*\*Estructura de Control `if`\*\***

La estructura `if` se utiliza para tomar decisiones basadas en una condición. Su sintaxis básica es la siguiente:

```
```bash
if [ condición ]; then
    # Código a ejecutar si la condición es verdadera
elif [ otra_condición ]; then
    # Código a ejecutar si la segunda condición es verdadera (opcional)
else
    # Código a ejecutar si ninguna de las condiciones anteriores es verdadera (opcional)
fi
...
```
```

Ejemplo:

```
```bash
if [ $edad -ge 18 ]; then
    echo "Eres mayor de edad."
else
    echo "Eres menor de edad."
fi
...
```
```

**\*\*Estructura de Control `case`\*\***

La estructura `case` se utiliza para realizar comparaciones múltiples y ejecutar código basado en coincidencias. Su sintaxis es la siguiente:

```
```bash
case variable in
    valor1)
```

```

        # Código a ejecutar si variable coincide con valor1
        ;;
valor2)
        # Código a ejecutar si variable coincide con valor2
        ;;
*)
        # Código a ejecutar si no hay coincidencias anteriores (opcional)
        ;;
esac
```

```

Ejemplo:

```

```bash
case $opcion in
    "a")
        echo "Seleccionaste la opción A."
        ;;
    "b")
        echo "Seleccionaste la opción B."
        ;;
    *)
        echo "Opción no válida."
        ;;
esac
```

```

**\*\*Estructura de Control `while`\*\***

La estructura `while` se utiliza para crear bucles que se ejecutan mientras una condición sea verdadera. Su sintaxis es la siguiente:

```

```bash
while [ condición ]; do
    # Código a ejecutar mientras la condición sea verdadera
done
```

```

Ejemplo:

```

```bash
contador=1
while [ $contador -le 5 ]; do
    echo "Número: $contador"
    contador=$((contador+1))
done
```

```



## **\*\*Estructura de Control `for`\*\***

La estructura `for` se utiliza para crear bucles que recorren elementos de una lista o un rango de números. Su sintaxis es la siguiente:

```
``bash
for variable in elemento1 elemento2 ...; do
 # Código a ejecutar para cada elemento en la lista
done
``
```

Ejemplo:

```
``bash
for fruta in manzana pera naranja; do
 echo "Me gusta la $fruta."
done
``
```

## **\*\*Estructura de Control `select`\*\***

La estructura `select` se utiliza para crear un menú interactivo en el que el usuario puede seleccionar una opción. Su sintaxis es la siguiente:

```
``bash
select variable in opcion1 opcion2 ...; do
 case $variable in
 opcion1)
 # Código a ejecutar para la opción 1
 ;;
 opcion2)
 # Código a ejecutar para la opción 2
 ;;
 *)
 # Código a ejecutar si la opción no es válida
 ;;
 esac
done
``
```

Ejemplo:

```
``bash
select comida in pizza hamburguesa tacos; do
 case $comida in
 "pizza")
 echo "Elegiste pizza."
 ;;
 esac
done
``
```

```

 "hamburguesa")
 echo "Elegiste hamburguesa."
 ;;
 "tacos")
 echo "Elegiste tacos."
 ;;
 *)
 echo "Opción no válida."
 ;;
esac
done
```

```

Estas son las estructuras de control más comunes en el scripting de shell (bash) utilizadas para tomar decisiones, crear bucles y crear menús interactivos en scripts de shell.

9) break:

- Acción: La sentencia break se utiliza para salir inmediatamente de un bucle, terminando prematuramente la ejecución del bucle.

continue:

- Acción: La sentencia continue se utiliza para omitir el resto del código dentro de un ciclo actual y avanzar a la siguiente iteración del bucle.

^ break [n] corta la ejecución de n niveles de loops.

^ continue [n] salta a la siguiente iteración del n-ésimo loop que contiene esta instrucción.

Un ejemplo del uso de estos parametros son cuando tienes un for anidado (ponerle que tienes un triple for, entonces si pones en el for de adentro de todo continue [2] va a ir para arriba, y si pones break [2] tambien va dos para arriba pero te corta la ejecución de los otros for). (me dijo el profe).

10)

Las variables no son fuertemente tipadas, lo que significa que no están asociadas a un tipo de datos específico y su tipo puede cambiar durante la ejecución del programa.

En Bash y en muchos otros lenguajes de programación, es común clasificar las variables en tres categorías principales: variables globales, variables locales y variables de entorno.

1. ****Variables Globales:****

- Las variables globales son aquellas que se declaran fuera de cualquier función o bloque de código.
- Tienen un alcance que abarca todo el script o programa, lo que significa que están disponibles para todos los comandos y funciones en el script.
- Si se modifican dentro de una función, la modificación se reflejará en todo el script.
- Se declaran sin la palabra clave `local``.

Ejemplo de variable global en Bash:

```
```bash
```

```
mi_variable="Hola, soy global"
```

```
'''
```

## 2. **\*\*Variables Locales:\*\***

- Las variables locales se declaran dentro de una función o bloque de código.
- Tienen un alcance limitado al bloque en el que se declaran. No están disponibles fuera de ese bloque.
- Su valor no afecta a variables con el mismo nombre en otros ámbitos.
- Se pueden declarar como locales utilizando la palabra clave ``local``.

Ejemplo de variable local en Bash:

```
'''bash
funcion_ejemplo() {
 local mi_variable="Hola, soy local"
 echo $mi_variable
}
'''
```

## 3. **\*\*Variables de Entorno:\*\***

**-Video que vi de explicacion: dice que son variables especiales globales que son heredables por procesos hijos de la shell donde se esta ejecutando.**

**Si quieres ver todas las variables de entorno pone el comando `export`.**

**Si quieres crear una hace: `export VARIABLE_GLOBAL="Mi var global"` .**

- Las variables de entorno son variables especiales que están disponibles para todos los procesos en un sistema Unix o Linux.
- Se utilizan para configurar el entorno de usuario y el comportamiento de programas.
- Pueden ser accedidas por cualquier programa en ejecución, y suelen contener información como rutas de búsqueda, configuraciones de idioma, información del usuario actual, etc.
- Se definen en mayúsculas y, por convención, se escriben en mayúsculas.

Ejemplo de variable de entorno en Bash:

```
'''bash
echo $HOME # Muestra el directorio de inicio del usuario actual (variable de entorno)
'''
```

Si se pueden definir arreglos.

```
arreglos=(1 2 3 4 5 6 7).
```

11) Si se puede, con la siguiente sintaxis:

```
^ function nombre { block }
```

ó

```
^ nombre() { block }
```

Por medio de variables (`$1`, `$2`, `$3...`) (al igual que cuando ejecutas scripts, le puedes mandar parámetros que se van a contener en estas variables y vas a poder usarlas dentro del script (por ej en un script `mi_script.sh` haces: `./mi_script.sh arg1 arg2 arg3`).

12)

a)

```
Activities Terminal Sep 13 10:10
luciano@Debian: ~/practica3

#!/bin/bash
echo "ingrese un numero:"
read num1
echo "ingrese el otro numero:"
read num2
echo "el resultado de la multiplicacion es: $((num1 * num2))"
echo "el resultado de la suma es: $((num1 + num2))"
echo "el resultado de la resta es: $((num1 - num2))"
if [$num1 -gt $num2]; then
 echo "el numero $num1 es mayor que $num2"
elif [$num1 -eq $num2]; then
 echo "los valores son iguales"
else
 echo "el numero $num2 es mayor que $num1"
fi
~
~
~
~
~
~
```

b)

```
Activities Terminal Sep 13 11:51
luciano@Debian: ~/practica3

#!/bin/bash
if [$# -ne 2]; then #verifico que se hayan proporcionado dos parametros
 echo "no ingresaste los dos numeros"
 exit 1
fi
echo "el resultado de la multiplicacion es: $((1 * 2))"
echo "el resultado de la suma es: $((1 + 2))"
echo "el resultado de la resta es: $((1 - 2))"
if [$1 -gt $2]; then
 echo "el numero $1 es mayor que $2"
elif [$1 -eq $2]; then
 echo "los valores son iguales"
else
 echo "el numero $2 es mayor que $1"
fi
exit 0
~
~
~
~
~
~
```

- c) En este inciso si vos pones "\*" cuando ejecutas el script (como parámetro agregas el \* seguido de 2 numeros) no se porque no te lo toma como un parámetro, osea en este caso imprime " no se ingresaron los parámetros correspondientes" (mensaje en el primer if), pero si lo pones como "\"\*\" funca todo perfecto (junto con los otros numeros claro). (porque?) porque el case tiene el \* para indicar el caso de error, por eso tenes que poner el /\*.

Además si yo imprimo en el primer if el comando "\$#" me dice que hay 6 parametros:

```
luciano@Debian:~/practica3$ bash script12c.sh * 4,2 4,7
no se ingresaron los parametros correspondientesd 6
```

```
luciano@Debian:~/practica3$
```

CONSULTAR SI ESTA BIEN COMO LO HICE Y SI SE SUPONE QUE SE DEBE INGRESAR EN EL PRIMER PARAMETRO UNA OPERACION Y LUEGO SE DEBE INGRESAR SOLO 2 NUMEROS MAS. SI, IGUAL ME DIJO QUE HAY QUE VERIFICAR QUE LOS PARAMETROS Q TENES QUE MANDAR VALORES NUMERICOS SEAN NUMERICOS:

así puedes verificar que es un numero:

```
expresion="^[0-9]+$"
```

```
if [[$1 =~ $expresion]]; then
 echo "verdadero"
else
 echo "falso"
fi
```

#### **CODIGO CORREGIDO:**

```
#!/bin/bash
determino si se ingresaron todos los parametros
if [$# -ne 3]; then
 echo "no se ingresaron los parametros correspondientesd $#"
```

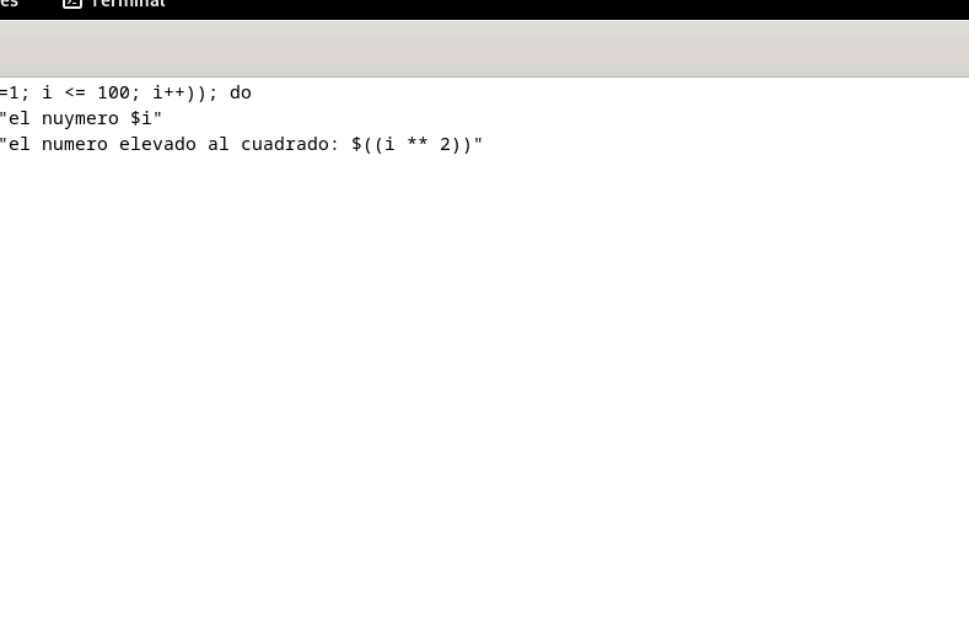
```
 exit 1
fi
#evaluo que se haya ingresado valores numericos:
expresion="^[0-9]+$"
```

```
if [[$2 =~ $expresion]] && [[$3 =~ $expresion]]; then
 num1=$2
 num2=$3
else
 echo "no ingresaste valores numericos"
 exit 1
fi
```

```
operacion="$1"
```

```
case $operacion in
 "+")
 echo "el resultado de la suma es: $((num1 + num2))"
```

13a)



The screenshot shows a terminal window titled "Debian [Corriendo] - Oracle VM VirtualBox". The window has a menu bar with "Archivo", "Máquina", "Ver", "Entrada", "Dispositivos", and "Ayuda". Below the menu bar is a dark bar with "Activities" and "Terminal" icons. The terminal itself has a light gray background and a dark gray prompt character "^". The script being executed is:

```
for ((i=1; i <= 100; i++)); do
 echo "el nuymero $i"
 echo "el numero elevado al cuadrado: $((i ** 2))"
done
exit 0
```

The output of the script is a series of 100 lines, each containing two lines of text: "el nuymero \$i" and "el numero elevado al cuadrado: \$((i \*\* 2))". The prompt character "^" is visible at the start of each line of output. The terminal window also shows a status bar at the bottom with the text "Sep 13" and "luciano@Debi".

b) SE SALE CREANDO UNA SECCIÓN SALIR CON UN BREAK O SINO EN LA LÍNEA DE COMANDOS APRETAS CTRL+C Y SE TERMINA.



luciano@Debian

```
select seleccion in listar dondeestoy quienesta salir; do
 case $seleccion in
 "listar")
 pwd | ls
 ;;
 "dondeestoy")
 pwd
 ;;
 "quienesta")
 who
 ;;
 "salir")
 echo "saliendo..."
 break
 ;;
 *)
 echo "seleccion no valida"
 esac
done
~
~
~
~
```

c) DUDA, como en linux todo es un archivo, el primer if hace referencia a si existe ese archivo (que puede ser un directorio, archivo regular, o otro tipo de archivo), luego el segundo if se fija si es directorio, el otro si es archivo regular y sino es un archivo existente de otro tipo no?. Además la búsqueda de si existe el archivo la realiza sobre el directorio en donde te encontras nada mas no? no busca en todo el file system. (NO, es una ruta relativa, busca en donde esta parado el archivo que mandaste por parametro)



luciano@Debian: ~/practica3

```
#!/bin/bash
if [$# -ne 1]; then
 echo "no ingresaste nada pibe"
 exit 1
fi
if [-e $1]; then
 echo "el archivo existe"
 if [-d $1]; then
 echo "es un directorio"
 elif [-f $1]; then
 echo "es un archivo regular"
 else
 echo "el archivo existe pero no es un archivo regular ni directorio"
 fi
else
 mkdir $1
fi
exit 0
~
~
~
~
~
```

14) Consultar resolucion. (me funcó) Esta bien pero tenes que cambiar que reciba los 3 parametros me dijo. (la idea del for de este ejercicio es la misma que la del 16):

```
#!/bin/bash
verifico que se haya ingresado algo como parametro
if [$# -ne 3]; then
 echo "no ingresaste lo que se te pedia"
 exit 1
fi

#verifico que sea directorio
if [! -d $directorio]; then
 echo "lo que mandaste no es un directorio"
 exit 1
fi

#verifico que sea una opcion valida
if [$3 = "a"] || [$3 = "b"]; then
 opcion=$3
else
 echo "ingresaste una opcion no valida"
 exit 1
fi

cadena=$2

cargo el parametro recibido en una variable
directorio=$1

cd $directorio
for archivo in *; do #recorro los archivos pueden ser de cualquier tipo.
 if [-f $archivo]; then
 if [$opcion = "a"]; then
 nuevo_nombre="${archivo}_${cadena}"
 else
 nuevo_nombre="${cadena}_${archivo}"
 fi
 echo "renombrando...."
 mv $archivo $nuevo_nombre
 echo "listo...."
 fi
done
exit 0
```

15)

El comando `cut` en Linux se utiliza para extraer secciones o campos específicos de líneas de texto de archivos o de la entrada estándar (stdin). Puede ser especialmente útil cuando



se trabaja con datos estructurados o archivos delimitados por caracteres (como archivos CSV o TSV).

La sintaxis básica del comando `cut` es la siguiente:

```
```bash
cut OPCIONES ARCHIVO
```
```

Donde `OPCIONES` son los parámetros que indican qué secciones o campos del archivo se deben extraer. A continuación, se describen algunas de las opciones más comunes que puede recibir el comando `cut`:

- **\*\* -c CARACTERES \*\***: Esta opción se utiliza para especificar los caracteres o rangos de caracteres que se deben extraer. Los caracteres se cuentan desde 1 y se pueden especificar individualmente o en rangos utilizando la notación `N-M`.

Ejemplo: Supongamos que tenemos un archivo llamado `datos.txt` con el siguiente contenido:

```
```
Hola Mundo
OpenAI es genial
```
```

Para extraer el primer carácter de cada línea:

```
```bash
cut -c 1 datos.txt
```
```

Esto producirá la salida:

```
```
H
O
```
```

- **\*\* -f CAMPOS \*\***: Esta opción se utiliza para especificar los campos que se deben extraer cuando se trabaja con archivos delimitados por caracteres (por ejemplo, archivos CSV o TSV). Los campos se identifican por números, donde el primer campo es el número 1.

Ejemplo: Supongamos que tenemos un archivo CSV llamado `empleados.csv` con el siguiente contenido:

```
```
Juan,Perez,30
Maria,Gonzalez,25
```
```

```
'''
```

Para extraer el segundo campo de cada línea (apellidos):

```
'''bash
cut -d ',' -f 2 empleados.csv
'''
```

Esto producirá la salida:

```
'''
```

```
Perez
Gonzalez
'''
```

- **\*\*-d DELIMITADOR\*\***: Esta opción se utiliza para especificar el delimitador que se utiliza en archivos delimitados por caracteres. El delimitador es el carácter que separa los campos.

Ejemplo: Supongamos que tenemos un archivo TSV (valores separados por tabulaciones) llamado `datos.tsv` con el siguiente contenido:

```
'''
```

| Nombre | Edad | Ciudad       |
|--------|------|--------------|
| Juan   | 30   | Buenos Aires |
| Maria  | 25   | Madrid       |

```
'''
```

Para extraer la segunda columna (Edad):

```
'''bash
cut -d$'\t' -f 2 datos.tsv
'''
```

Esto producirá la salida:

```
'''
```

```
Edad
30
25
'''
```

Estos son solo algunos ejemplos de cómo se puede utilizar el comando `cut`. Puedes combinar múltiples opciones y utilizarlas de acuerdo con tus necesidades específicas para extraer información de archivos de texto de manera efectiva. El comando `man cut` en tu terminal proporcionará información detallada sobre todas las opciones disponibles y su funcionamiento.

```

Mandatory arguments to long options are mandatory for short options too.
-b, --bytes=LIST select only these bytes
-c, --characters=LIST select only these characters
-d, --delimiter=DELIM use DELIM instead of TAB for field delimiter
-f, --fields=LIST select only these fields; also print any line
 that contains no delimiter character, unless
 the -s option is specified
-n (ignored)
 --complement complement the set of selected bytes, characters
 or fields
-s, --only-delimited do not print lines not containing delimiters
 --output-delimiter=STRING use STRING as the output delimiter
 the default is to use the input delimiter
-z, --zero-terminated line delimiter is NUL, not newline
 --help display this help and exit
 --version output version information and exit

```

**16) Consultar resolución, osea funca, pero para ver si lo que pedía el enunciado era esto (yo deduzco de que cada vez que ejecute el script se me guardara el usuario y cantidad de archivos (regulares asumi yo) (que están en el directorio /home/luciano/practica3 (¿tiene que ser del directorio de inicio del usuario y a su vez de los subdirectorios del mismo (recursivo) la cantidad de archivos q tengo q guardar q coincidan con esa extension ?) en reporte.txt, en este caso de /home/luciano/practica3. Me dijo que esta bien, pero la idea es que guarde toda la info de los usuarios, entonces una forma es que te fijas en el /home donde estan todos los usuarios que existen o la otra forma que me dijo que es mejor es fijarte en el /etc/passwd. CAMBIAR.**

**forma con el /etc/passwd (a probar):**

directorios= cut -d: -f6 te quedas con los directorios que hay. Deberia quedarme con todos los que tienen /home. (con un find capaz). find type-d (directorio) -name "\*" /home" lo cargas a un archivo y dsp lo pasas a una variable.

lo recorres en un for esa variable (recorrerías cada directorio /home.

para cada uno te posicionas ahí, calculas la cantidad de archivos y dsp con el tr te guardas el nombre.

cargas en el archivo.

**forma recorriendo del /home: (mejor es la del /etc/passwd, HACERLO)**

```
#!/bin/bash
```

```
#Verificar si se proporcionó la extensión como parámetro
```

```
if [$# -ne 1]; then
```

```
 echo "Debe proporcionar una única extensión como parámetro."
```

```
 exit 1
```

```
fi
```

```
extension="$1"
```

```
#Obtener el nombre de usuario
```

```

cd /home
usuarios=$(ls)
echo "$usuarios"
for usuario in $usuarios; do
 echo "$usuario"
 cd /home/$usuario
 #Buscar archivos con la extensión en el directorio actual y subdirectorios

 cantidad_archivos=$(find -type f -name ".*$extension" | wc -l)

 #Crear el reporte en un archivo llamado reporte.txt

 echo "$usuario $cantidad_archivos" >> /home/luciano/practica3/reporte.txt
done
exit 0

```

### FORMA QUE SE QUIERE CON EL /ETC/PASSWD:

```

#!/bin/bash
#Verificar si se proporcionó la extensión como parámetro

if [$# -ne 1]; then
 echo "Debe proporcionar una única extensión como parámetro."
 exit 1
fi

extension="$1"

echo -n > reporte2.txt #esto para que cuando lo ejecutes este script, si ese archivo tenia
algo q se borre (aca le asignas vacio de manera
#restrictiva borrando todo lo q habia antes. #el -n es para q no te haga salto de linea

#Obtener el nombre de usuario
for usuario in $(cat "/etc/passwd"); do
 echo "$usuario"
 nombre=$(echo "$usuario" | cut -d":" -f1)
 ruta=$(echo "$usuario" | cut -d":" -f6)
 echo "$ruta"
 #Buscar archivos con la extensión en el directorio actual y subdirectorios
 if [-d $ruta]; then #lo hacia en el video contabilizas usuarios con directorio personal
 basicamente
 cantidad_archivos=$(find $ruta -type f -name ".*$extension" | wc -l)
 #Crear el reporte en un archivo llamado reporte.txt
 echo "$nombre $cantidad_archivos" >> /home/luciano/practica3/reporte2.txt
 fi
done

```

## 17) Explicación del uso del TR.

**El comando tr en sí mismo no modifica directamente el contenido de un archivo. En lugar de eso, puedes redirigir la salida del comando tr a un nuevo archivo. EL TR SI BIEN SIRVE O SE APLICA PARA ARCHIVOS DE TEXTO Y ENTRADA STANDAR, NO MODIFICAS EL ARCHIVO, TIPO NO PODES HACER “TR -S “aA” PRUEBA.TXT.**

El comando `tr` en Linux es una utilidad de traducción de caracteres que se utiliza para realizar transformaciones de caracteres en un flujo de texto. Puede usarse para eliminar, reemplazar o transliterar caracteres en un archivo de texto o en la entrada estándar (stdin) y producir la salida resultante en la salida estándar (stdout). El comando `tr` es muy útil para realizar operaciones simples de manipulación de texto.

La sintaxis básica del comando `tr` es la siguiente:

bash

Copy code

```
tr [OPCIONES] SET1 [SET2]
```

Donde:

- `SET1` es el conjunto de caracteres que se busca reemplazar o eliminar.
- `SET2` es el conjunto de caracteres con los que `SET1` se reemplazará (opcional).

A continuación, se explican algunas de las opciones más comunes y se proporcionan ejemplos de uso con archivos y texto:

### Opciones comunes de `tr`:

- `-d SET1`: Elimina todos los caracteres en `SET1` del flujo de texto de entrada.

Ejemplo:

- bash
- Copy code

```
echo "Hola Mundo" | tr -d "o"
```

- Salida: Hla Mund
- `-s`: Sustituye cada secuencia repetida de caracteres en `SET1` con un solo carácter.

Opción `-s` (squeeze):

La opción `-s` se utiliza para eliminar duplicados de caracteres especificados en un flujo de datos. Por ejemplo, si tienes un archivo llamado "archivo.txt" con el siguiente contenido:

mathematica

Copy code

- ¡¡Hola!! ¿Cómo estás?

Puedes eliminar los signos de exclamación duplicados utilizando `tr -s "!"` de la siguiente manera:

bash

Copy code

- `cat archivo.txt | tr -s "!"`

Esto eliminará los signos de exclamación duplicados y mostrará el resultado en la pantalla:

mathematica

Copy code

¡Hola! ¿Cómo estás?

ACLARO YO QUE SI EL ARCHIVO TIENE MAS LINEAS, TAMBIEN TE IMPRIME ESAS OTRAS LINEAS, LO ACABO DE PROBAR, OSEA TE IMPRIME TODO PERO BORRANDO ESA REPETICION.

-c: Complementa `SET1`, es decir, reemplaza todos los caracteres que no están en `SET1`.

Ejemplo:

- bash

- Copy code

```
echo "abcdef" tr -c "a-c" "X"
```

- Salida: abcXXX

### EJERCICIO:

```
#!/bin/bash
```

```
Obtener la lista de archivos en el directorio actual
```

```
archivos=$(ls)
```

```
Recorrer cada archivo
```

```
for archivo in $archivos; do
```

```
 # Cambiar minúsculas por mayúsculas y eliminar la letra "a"
```

```
 nuevo_nombre=$(echo "$archivo" | tr -d "aA" | tr "a-z" "A-Z")
```

```
 # Imprimir el nuevo nombre en pantalla
```

```
 echo "$nuevo_nombre"
```

```
done
```

```
exit 0
```

### Mejor forma:

```
#!/bin/bash
```

```
echo "$(ls | tr -d "aA" | tr "a-z" "A-Z")"
```

**18)** (hay audio wtsp explicando algo importante)

```
#!/bin/bash
```

```
if [$# -ne 1]; then
```

```
 echo "no ingresaste los parametros que se te solicitan"
```

```
 exit 1
```

```
fi
```

```
while true; do
```

```
 if who | grep -q $1; then #el -q lo que te permite es que no se muestre nada en la pantalla al momento de efect>
```

```
 echo "Usuario $1 logueado en el sistema"
```

```
 exit 0
```

```
 fi
```

```
 #en caso contrario:
```

```
 sleep 10
```

```
done
```

El comando `grep -q` se utiliza comúnmente en secuencias de comandos de shell o scripts cuando solo se necesita verificar la existencia de un patrón en un archivo y no se requiere mostrar el resultado en la pantalla. El valor de salida de `grep -q` es 0 si se encuentra el patrón y 1 si no se encuentra, lo que permite a los scripts tomar decisiones basadas en si se encontró o no el patrón.

19) (la prueba lo hice con el 14 pq le mando parametros y eso, dsp con el resto me dio paja.

```
#!/bin/bash
```

```
echo "MENU DE COMANDOS:"
```

```
select elegido in 12 13 14 15 salir; do
```

```
 case $elegido in
```

```
 "12")
```

```
 bash /home/luciano/practica3/script12.sh
```

```
 ;;
```

```
 "13")
```

```
 bash /home/luciano/practica3/script13.sh
```

```
 ;;
```

```
 "14")
```

```
 echo "dame un directorio para pasarle como parametro:"
```

```
 read directorio
```

```
 echo "dame una cadena:"
```

```
 read cadena
```

```
 echo "dame una opcion:"
```

```
 read opcion
```

```
 bash /home/luciano/practica3/script14.sh $directorio $cadena $opcion
```

```
 ;;
```

```
 "15")
```

```
 bash /home/luciano/practica3/script15.sh
```

```
 ;;
```

```
 "salir")
```

```
 echo "cerrando..."
```

```
 exit 0
```

```
 ;;
```

```
 *)
```

```
 echo "hubo un problema"
```

```
 exit 1
```

```
 ;;
```

```
 esac
```

```
done
```

**CONSULTAR SI TENES TIEMPO.**

**20) y 21) DUDA EN LA FUNCION PUSH TENGO QUE VERIFICAR QUE SE RECIBE ALGO COMO PARAMETRO? Y EN EL INICIO DEL SCRIPT, TENGO QUE VERIFICAR QUE NO SE INGRESEN PARAMETROS? FORMA CORRECTA CON FUNCIONES:**



```

#!/bin/bash
if [$# -ne 0]; then
 echo " no ingreses parametros"
 exit 1
fi

Inicializar una pila vacía como un arreglo en Bash
pila=()

Función para agregar un elemento a la pila (push)
push() {
 pila+=($1)
}

Función para sacar un elemento de la pila (pop)
pop() {
 if [${#pila[@]} -eq 0]; then
 echo "La pila está vacía. No se puede realizar pop."
 else
 indice=$((${#pila[*]} - 1))
 echo "el indice es: $indice"
 elemento=${pila[$indice]}
 unset pila[$indice] # Eliminar el último elemento de la pila
 echo "Se sacó el elemento: $elemento"
 fi
}

Función para obtener la longitud de la pila (length)
length() {
 echo "La longitud de la pila es: ${#pila[@]}"
}

Función para imprimir todos los elementos de la pila (print)
print() {
 echo "contenido de la: ${pila[@]}"
}

for ((i=1; i<=10; i++)); do
 echo "ingrese un parametro a agregar en la pila"
 read valor
 push $valor
done
for ((i=1; i<=3; i++)); do
 echo "sacando elemento"
 pop
done
length
print

```

22)

```
#!/bin/bash
```

```
vector=(10 3 5 7 9 3 5 4)
```

```
productoria() {
```

```
 local resultado=1 #inicializas
```

```
 local arreglo=($@) #eso es una variable que representa todos los argumentos pasados a un
 script/funcion en forma de lista de palabras separadas.
```

```
 echo "valores: ${arreglo[*]}"
```

```
 for v in ${arreglo[*]}; do
```

```
 echo "valor tomado: $v"
```

```
 resultado=$((resultado * v))
```

```
 done
```

```
 echo "$resultado" #retornas el resultado usando echo ya que el return no te sirve (acá
se te van a retornar TODOS los echos que hagas en la funcion, ojo, la idea es que
solo retorne el valor de la multiplicacion".
```

```
}
```

```
resultado=$(productoria ${vector[*]}) #tenes que poner el $() para que te retorne lo de
echo de la funcion. Se te guarda un string en la variable resultado.
```

```
echo "el resultado de la multiplicacion es: $resultado"
```

23)

```
#!/bin/bash
```

```
vector=(1 2 3 4 5 6 7 8)
```

```
impares=0
```

```
for v in ${vector[*]}; do
```

```
 if [$((v % 2)) -eq 0]; then
```

```
 echo "el valor $v es par"
```

```
 else
```

```
 impares=$((impares + 1))
```

```
 fi
```

```
done
```

```
echo "la cantidad de numeros impares es de: $impares"
```

24)

```
#!/bin/bash
```

```
vector1=(1 2 3 4 5 6)
```

```
vector2=(7 8 9 10 11 12)
```

```
for ((i=0; i<${#vector1[*]}; i++)); do
 echo "La suma de los elementos de la posicion $i de los vectores es $((vector1[$i] +
vector2[$i]))"
done
```

### Otra forma:

```
#!/bin/bash
```

```
vector1=(1 2 3 4)
```

```
vector2=(1 2 3 4)
```

```
for indice in ${!vector1[*]}; do
 echo "La suma de los elementos de la posicion $indice de los vectores es
$((${vector1[$indice]} + ${vector2[$indice]}))"
done
```

25) **Consultar este punto, me funco y todo, pero igual consultar.** Ademas tengo una duda, cuando uso el -z en el if [ -z "\${arreglo[\$2]}" ], si el arreglo es de valores numericos igual funciona, esto es porque como en linux no tenes que definirle a un valor si es intiger o no, en esa condicion (suponete que me retorna el valor 4 al acceder a esa pos del arreglo) me lo toma como una cadena por el -z, pero dsp si yo quiero a ese valor de la pos del arreglo sumarlo con por ej 20 ahi me lo toma como un integer no?. en los arreglos, sus contenidos se definen como strings de manera predeterminada? osea me puso esto chatgpt: **los valores en las posiciones de un arreglo se tratan como cadenas de caracteres (strings)**

```
#!/bin/bash
```

```
if [$# -ne 1] && [$# -ne 2]; then
 echo "ingresaste mal los parametros"
 exit 1
fi
```

```
arreglo=$(users))
```

```
#verifico si el primer parametro es un string
```

```
if [[-n "$1" && ! "$1" =~ [0-9]]]; then #se usa doble corchete ya que asi se evaluan
expresiones condicionales, si la expresion es V devuelve 0 y sino otro valor
```

```
 echo "$1 es una cadena de caracteres."
```

```
else
```

```
 echo "$1 no es una cadena de caracteres."
```

```
 exit 1
```

```
fi
```

```
if [-z $2]; then #si el segundo parametro es vacio entonces por descarte puede ser el -l o -i
 case $1 in
```

```

 "-l")
 echo "la longitud del arreglo es:${#arreglo[*]}"
 exit 0
 ;;
 "-i")
 echo "${arreglo[*]}"
 exit 0
 ;;
 *)
 echo " no era ninguno de los anteriores, chau"
 exit 1
 ;;
 esac
fi
#analizoo que el segundo parametro sea un valor numerico
if [[$2 =~ ^[0-9]+$]]; then
 echo "\$2 es un valor numérico."
 if [$1 = "-b"]; then
 if [-z "${arreglo[$2]}"]; then #usas " para asegurarte que el contenido del
 elemento del arreglo se interprete bien, especialmente si contiene espacios o >
 echo "error, la posicion en la que quieres obtener el usuario esta vacia"
 else
 echo "el valor en la posicion $2 es: ${arreglo[$2]}"
 fi
 fi
 else
 echo "\$2 no es un valor numérico o está vacío."
 exit 1
 fi
fi

```

### MODIFICADO Y ANDA JOYA:

```

#!/bin/bash

if [$# -ne 1] && [$# -ne 2]; then
 echo "ingresaste mal los parametros"
 exit 1
fi

arreglo=($(users))

if [-z $2]; then #si el segundo parametro es vacio entonces por descarte puede ser el -l o -i
 case $1 in
 "-l")
 echo "la longitud del arreglo es:${#arreglo[*]}"
 exit 0
 ;;
 "-i")
 echo "${arreglo[*]}"
 ;;
 esac
fi

```

```

 exit 0
 ;;
 *)
 echo " no era ninguno de los anteriores, chau"
 exit 1
 ;;
esac
fi
#analizoo que el segundo parametro sea un valor numerico
if [[$2 =~ ^[0-9]+$]]; then
 echo "$2 es un valor numérico."
 if [$1 = "-b"]; then
 if [-z "${arreglo[$2]}"]; then #usas " para asegurarte que el contenido del
elemento del arreglo se interprete bien, especialmente si contiene espacios o caracteres
especiales
 echo "error, la posicion en la que quieres obtener el usuario esta vacia"
 else
 echo "el valor en la posicion $2 es: ${arreglo[$2]}"
 fi
 else
 echo "ingresaste como primer parametro algo incorrecto"
 exit 1
 fi
else
 echo "$2 no es un valor numérico."
 exit 1
fi

```

26) **consultar resolucion (funca todo) y si cuando me pide que informe los que no se pudieron encontrar, si esta bien como lo hago con respecto a incrementar mi iterador solamente en los casos impares donde ese archivo no existe. ESTA BIEN**

```

#!/bin/bash
Verifica que se haya proporcionado al menos un parámetro
if [$# -lt 1]; then
 echo "Debe proporcionar al menos un parámetro."
 exit 1
fi

Inicializa un contador para el número de archivos o directorios inexistentes
inexistentes=0

Itera por los parámetros recibidos en posiciones impares
for ((i = 1; i <= $#; i += 2)); do
 ruta="{i}" # Obtiene la ruta del parámetro actual, toma todos los parametros impares, $1,
$3, etc
 # Verifica si el archivo o directorio existe

```

```

if [-e "$ruta"]; then
 # Determina si es un archivo o un directorio
 if [-f "$ruta"]; then
 echo "$ruta es un archivo."
 elif [-d "$ruta"]; then
 echo "$ruta es un directorio."
 fi
else
 echo "$ruta no existe en el sistema."
 inexistentes=$((inexistentes + 1))
fi
done

Informa la cantidad de archivos o directorios inexistentes
if [$inexistentes -eq 0]; then
 echo "No se encontraron archivos o directorios inexistentes."
else
 echo "Se encontraron $inexistentes archivos o directorios inexistentes."
fi

```

#### **otra forma:**

```

#!/bin/bash
if [$# -eq 0]; then
 echo "ingresaste mal los parametros"
 exit 1
fi

arreglo=($@)

echo "info del arreglo: ${arreglo[*]}"

for ((i=1; i<${#arreglo[*]}; i+=2)); do
 echo "ruta tomada: ${arreglo[$i]}"
 if [-e ${arreglo[$i]}]; then
 if [-d ${arreglo[$i]}]; then
 echo "es directorio el directorio ${arreglo[$i]}"
 fi
 if [-f ${arreglo[$i]}]; then
 echo "es archivo regular el archivo ${arreglo[$i]}"
 fi
 else
 cant=$((cant + 1))
 fi
done

echo "cantidad de archivos inexistentes es de $cant"

```

27) **MODIFICADO MEJOR EN LA PC EN EL LINUX, LO HICE MAL ESO, FIJATE Q ME FALTO LOS [\*]** ///consultar si esta bien el eliminar\_elem y luego PORQUE EN EL AGREGAR\_ELEM CUANDO VOY AGREGANDO ATRAS CON LA POS \${#ARREGLO\_VACIO} (arreglo\_vacio[\${#arreglo\_vacio}]= \$1) Y YO PONIA PARA AGREGAR EL POR EJ 3 5 7, ME AGREGABA EL 3 Y 7 NOMAS Y CUANDO YO IMPRIMIA COMO IBA VARIANDO EL VALOR DE LA LONGITUD ME PONIA ESTO:

```
luciano@Debian:~/practica3$ bash script27.sh
longitud del areglo antes 0
longitud del areglo ahora 2
longitud del areglo antes 2
longitud del areglo ahora 2
longitud del areglo antes 2
longitud del areglo ahora 2
55 22
```

Dsp cuando lo arregle de la manera que dice justo aca abajo tambien el tamaño de la longitud lo imprime de la misma manera, por ende cual seria la logica ahi?

#### **TENER EN CUENTA:**

**La expresión ``array_vacio+=("$1")`` en Bash se utiliza para agregar un elemento al final de un arreglo. Aquí está el significado de cada parte de esta expresión:**

- ``array_vacio``: es el nombre del arreglo al que deseas agregar un elemento.
- ``+=``: es el operador de concatenación para arreglos en Bash.
- ``"$1"``: es el valor que deseas agregar al arreglo. En este caso, ``$1`` se refiere al primer argumento que se pasa a la función.

Entonces, cuando ejecutas ``array_vacio+=("$1")``, estás tomando el valor de ``$1`` (el primer argumento pasado a la función) y lo estás agregando al final del arreglo ``array_vacio``. Después de ejecutar esta línea de código, ``array_vacio`` tendrá un nuevo elemento que es el valor de ``$1``. Esto es útil para construir arreglos dinámicamente, agregando elementos a medida que sea necesario.

```
#!/bin/bash
```

```
inicializar(){
 array_vacio=()
}
```

```
agregar_elem() {
 if [-n "$1"]; then
 echo "longitud del areglo antes ${#array_vacio}"
 array_vacio+=($1)
 echo "longitud del areglo ahora ${#array_vacio}"
 else
```

```

 echo "no se pudo agregar porque no enviaste nada como parametro"
 fi
}

eliminar_elem() {
 # probe que si no le mandas nada como parametro, el if que verifica si es
 # numérico, te da que no lo es, entonces el -n seria al pedo.
 if [-n "$1"]; then #analizo si no esta vacio, deja los " de $1 pq sino cuando no le
 mandes parametro no te tira el mensaje de variable no tiene nada
 echo "valor $1"
 if [[$1 =~ ^[0-9]+$]] && [$1 -ge 0] && [$1 -lt ${#array_vacio[*]}]; then #analizo
 que sea un valor numerico mayor o igual a 0 y menor a la longitud del array
 unset array_vacio[$1]
 else
 echo "algo paso"
 fi
 else
 echo "la variable no tiene nada"
 fi
}

longitud() {
 echo " la longitud del arreglo es: ${#array_vacio[*]}"
}

imprimir() {
 echo "${array_vacio[*]}"
}

inicializar_Con_Valores() {
 #analizo que el primer parametro sea un valor numerico si o si, el otro no me importa
 arreglo=()
 if [[$1 =~ ^[0-9]+$]]; then
 for ((i=1; i<=$1; i++)); do
 arreglo+=("$2")
 done
 echo "info del nuevo arreglo: ${arreglo[*]}"
 else
 echo "no era un valor numerico el primer parametro"
 fi
}

prueba() {
 arreglooo=(1 2 3 4)
 arreglooo[7]=3
 echo "elemento en la posicion 4 es :${arreglooo[4]} y dsp en la 7: ${arreglooo[7]}"
}

```



```

 echo "${#arreglooo[*]}"
 }
 inicializar
 agregar_elem 55
 agregar_elem 44
 agregar_elem 22
 imprimir
 longitud
 eliminar_elem 3
 eliminar_elem
 eliminar_elem r
 eliminar_elem 0
 imprimir
 longitud
 inicializar_Con_Valores

```

28) estoy teniendo unas complicaciones con dicho punto, mi idea era dado un nombre de un directorio recibido como parámetro determinar si este existe quedándome con su correspondiente ruta con la utilización de un find (en caso de que no exista quería que dicha variable quede vacía para que no entre al primer if), sin embargo no estaría entendiendo porque no se le asigna ninguna ruta a mi variable "ruta", ¿me falta algo más?. ME DIJO QUE LOS Q TIENEN DE LECTURA PARA UN LADO, LOS QUE TIENEN DE ESCRITURA PARA OTRO LADO (OTRO CONTADOR) Y EL NOMBRE DEL DIRECTORIO ES LA RUTA.

TE PIDE Q SI ES SUBDIRECTORIO NO LE DES BOLA, TENES Q ANALIZAR ARCHIVOS REGLARES. ARREGLAR

**resuelto:**

```

GNU nano 7.2 script28.sh
#!/bin/bash

if [$# -ne 1]; then
 echo "no se ingresaron los parametros correspondientes"
 exit 1
fi

if [-d "$1"]; then #si existe el directorio
 cant_lecturas=0
 cant_escrituras=0
 cd $1

```

```

for archivo in *; do
 if [-w "$archivo"]; then
 cant_escrituras=$((cant_escrituras + 1))
 fi
 if [-r "$archivo"]; then
 cant_lecturas=$((cant_lecturas + 1))
 fi
 echo "archivo: $archivo"
done

else
 echo "el directorio no existe"
 exit 4
fi

echo "la cantidad de archivos donde el usuario $whoami tiene permisos de lectura es:
$cant_lecturas"
echo "la cantidad de archivos donde el usuario $whoami tiene permisos de escritura es:
$cant_escrituras"

```

```

GNU nano 5.8
#!/bin/bash

if [$# -ne 1]; then
 echo "Debe pasar como parametro el nombre de un directorio"
 exit 4
fi

cantR=0
cantW=0
if [-d $1]; then
 for archivo in $(ls $1)
 do
 path="$1/$archivo"
 if [-f $path]; then
 echo "$archivo es un file"
 if [-r $path]; then
 ((cantR++))
 fi
 if [-w $path]; then
 ((cantW++))
 fi
 fi
 done
fi

echo "La cantidad de archivos con permisos de escritura es $cantW y de lectura es $cantR"
exit 0

```

29) **Consultar resolucion,** pero funco. HAY OTRA FORMA ABAJO

```
#!/bin/bash
```

```
vector=$(ls /home | grep ".doc") #me guardo LOS NOMBRES de los archivos que
contienen ".DOC" en /home
```

```
verArchivo() {
```

```
 if [-n "$1"]; then
 for archivo in ${vector[*]}; do
 if ["$1" = "$archivo"]; then
 echo "el archivo $1 se encuentra en el arreglo"
 return 0
 fi
 done
 fi
 echo "no se encontro o no pusiste nada como parametro"
 return 5
```

```
}
```

```
cantidadArchivos() {
```

```
 echo "la cantidad de archivos son: ${#vector[*]}"
```

```
}
```

```
borrarArchivo() {
```

```
 if [-n "$1"]; then
 for ((i=0;i<${#vector[*]};i++)); do
 if ["$1" = "${vector[$i]}"]; then
 echo " elija:"
 select eligio in si no terminar; do
 case $eligio in
 "si")
 unset vector[$i]
 ;;
 "no")
```

```
 #sabiendo que en distintos directorios pueden haber archivos
con el mismo nombre, aplico maxdepth
```

```
 ruta=$(find /home -maxdepth 1 -type f -name ${vector[$i]})
 echo "ruta: $ruta"
 rm $ruta
 unset vector[$i]
 ;;
 "terminar")
 exit 0
```

```

 ;;
 *)
 echo "hubo un error"
 exit 1
 ;;
 esac
done
fi
done
echo "archivo no encontrado"
exit 10
fi
echo "${vector[*]}"
}

```

```

verArchivo hola.doc
cantidadArchivos
borrarArchivo hola.doc

```

### OTRA FORMA QUE ME FUNCO:

```
#!/bin/bash
```

```

arreglo=($(ls /home | grep "doc$"))
echo ${arreglo[*]}

```

```

verArchivo () {
 if [$# -ne 1]; then
 echo "no ingresaste los parametros necesarios"
 return 1
 fi
 local i=0
 while [$i -lt ${#arreglo[*]}]; do
 echo "entre"
 if [${arreglo[$i]} == $1]; then
 echo "se lo encontro ${arreglo[$i]}" o imprimo el contenido del arch.
 return 0
 fi
 i=$((i + 1))
 done
 echo "archivo no encontrado"
 return 5
}

```

```
cantidadArchivos () {
```

```

 echo "la cantidad de archivos del /home con terminacion .doc son ${#arreglo[*]}"
 }

 borrarArchivo () {
 if [$# -ne 1]; then
 echo "no ingresaste bien los parametros"
 return 1
 fi
 echo "FORRO"
 echo "indices: ${!arreglo[*]}"
 for i in ${!arreglo[*]}; do
 echo "valor de i: $i "
 if [${arreglo[$i]} == $1]; then
 echo "el archivo fue encontrado en el arreglo y es : ${arreglo[$i]}"
 echo "porfavor, ingrese SI o NO"
 read respuesta
 case $respuesta in
 "SI")
 unset arreglo[$i]
 ;;
 "NO")
 rm /home/${arreglo[$i]}
 unset arreglo[$i]
 ;;
 esac
 fi
 done
 echo "el archivo no encontrado"
 return 10
 }
}

```

```

verArchivo hola.doc
cantidadArchivos
borrarArchivo hola.doc

```

30) funciono, agregar condicion if para q si es un archivo con permisos de ejecucion -x  
#!/bin/bash

```

#verifico que /bin/home exista
if [! -d "$HOME/bin"]; then
 echo "no existe ese directorio, vamos a crearlo..."
 mkdir "$HOME/bin"

```

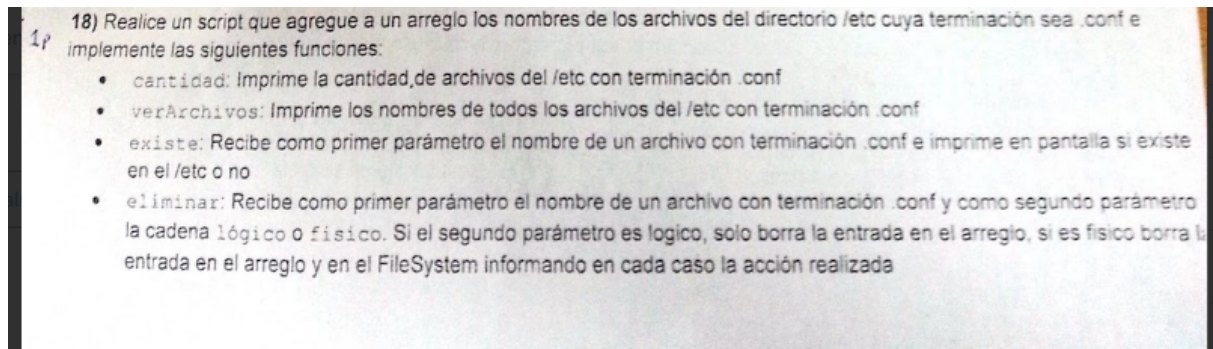
```

fi

#recorro el directorio actual moviendo todo
cant=0
for archivo in *; do
 if [-f $archivo] && [-x $archivo]; then
 mv "$archivo" "$HOME/bin"
 echo "el archivo que movi fue: $archivo"
 cant=$((cant + 1))
 fi
done
if [$cant -eq 0]; then
 echo "No movio ningun archivo"
else
 echo "la cantidad de archivos que movio fue de : $cant"
fi

```

### Parcial: funko todo perfecto



```

#!/bin/bash

arreglo=$(ls /etc | grep ".conf$")

cantidad() {
 echo "la cantidad de archivos con terminacion .conf del directorio /etc son:
 ${#arreglo[*]}"
}

verArchivos() {
 echo "${arreglo[*]}"
}

existe() {
 if [$# -ne 1]; then
 echo "ingresaste mal los parametros"
 return 1
 fi
}

```

```

if echo "$1" | grep -q ".conf$"; then
 echo "apto para ser buscado"
 for archivo in ${arreglo[*]}; do
 if [$archivo = $1]; then
 echo "existe en el /etc"
 return 0
 fi
 done
 echo "no se lo encontro"
 return 1
fi
echo "ingresaste por parametro un nombre que no termina en .conf"
return 1
}

eliminar(){
 if [$# -ne 2]; then
 echo "ingresaste mal los parametros"
 return 1
 fi
 for indice in ${!arreglo[*]}; do
 if [${arreglo[$indice]} = $1]; then
 case $2 in
 "logico")
 echo "borrando logico..."
 unset arreglo[$indice]
 echo "arreglo ahora: ${arreglo[*]}"
 return 0
 ;;
 "fisico")
 echo "borrando fisico..."
 rm /etc/${arreglo[$indice]}
 unset arreglo[$indice]
 return 0
 ;;
 *)
 echo "ingresaste otra cosa amigo"
 return 1
 ;;
 esac
 fi
 done
 echo "nunca lo encontro..."
 return 1
}

```

cantidad

verArchivos  
existe "fuse.conf"  
eliminar fuse.conf logico

17) Escriba un script que reciba una cantidad desconocida de parámetros al momento de su invocación (debe validar que al menos se reciba uno). Cada parámetro representa la ruta absoluta de un archivo o directorio en el sistema. El script deberá iterar por todos los parámetros recibidos, y solo para aquellos parámetros que se encuentren en posiciones impares (el primero, el tercero, el quinto, etc.), verificar si el archivo o directorio existen en el sistema, imprimiendo en pantalla que tipo de objeto es (archivo o directorio). Además, deberá informar la cantidad de archivos o directorios inexistentes en el sistema.

```
#!/bin/bash
if [$# -eq 0]; then
 echo "ingresaste mal los parametros"
 exit 1
fi

arreglo=($@)

echo "info del arreglo: ${arreglo[*]}"

for ((i=1; i<${#arreglo[*]}; i+=2)); do
 echo "ruta tomada: ${arreglo[$i]}"
 if [-e ${arreglo[$i]}]; then
 if [-d ${arreglo[$i]}]; then
 echo "es directorio el directorio ${arreglo[$i]}"
 fi
 if [-f ${arreglo[$i]}]; then
 echo "es archivo regular el archivo ${arreglo[$i]}"
 fi
 else
 cant=$((cant + 1))
 fi
done

echo "cantidad de archivos inexistentes es de $cant"
```

14. Escribir un script que verifique cada 5 segundos si un usuario está logeado en el sistema. El nombre del usuario es pasado como parámetro. En caso de que se detecte que el usuario está logeado, se deberá informar: "El usuario X es usuario del sistema", guardar en un archivo llamado usuarios.log el nombre del usuario y la fecha y terminar el script. Tenga en cuenta que la información del archivo NO debe pisarse y que el archivo resultante lo debe almacenar en una ubicación apropiada. Recuerde validar los parámetros del Script

**CONSULTAR AMBAS RESOLUCIONES, ME FUNCIONARON, PERO CONSULTAR.**  
**LO TENES QUE HACER DON DATE LA FECHA, NO SACARLA DE LA COLUMNA ESA.**  
#!/bin/bash



```

if [$# -ne 1]; then
 echo "ingresaste mal los parametros"
 exit 1
fi

while true
do
 if who | grep -q "$1"; then
 usuario=$(who | grep -w "$1" | cut -d" " -f1)
 fecha=$(who | grep -w "$1" | cut -d" " -f12)
 echo "USUARIO $usuario es usuario del sistema"
 echo "$usuario $fecha" >> /var/log/usuarios.log
 exit 0
 fi
 sleep 5
done

```

### OTRA FORMA:

```

#!/bin/bash

if [$# -ne 1]; then
 echo "ingresaste mal los parametros"
 exit 1
fi

while true
do
 for usuario in $(who); do
 user=$(echo $usuario | cut -d" " -f1)
 if ["$user" = "$1"]; then
 fecha=$(who | grep "$1" | cut -d" " -f12)
 echo "USUARIO $user es un usuario del sistema"
 echo "$user $fecha" >> /home/luciano/usuarios.log
 exit 0
 fi
 done
 sleep 5
done

```

15- Realice un script que reciba una cantidad indefinida de parámetros(al menos uno) y los agregue a un arreglo. Además debe implementar las siguientes funciones:

- existe <parametro1>: Imprime el índice dentro del arreglo donde está el parámetro <parametro1>, sólo si este existe en el arreglo. Adicionalmente, debe retornar 0 si el elemento existe en el arreglo ó 1 en caso contrario.
- replace <parametro1> <parametro2>: Reemplazar <parametro1> por <parametro2> en el arreglo(solo si <parametro1> existe )
- eliminar <parametro1>: Si el parámetro pasado como parámetro existe en el arreglo, lo elimina del mismo. Caso contrario retorna código de error 2.
- cantidad: Imprime la cantidad total de elementos en el arreglo.
- todos: Imprime todos los elementos del arreglo

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

## **FUNCO TODO, PERO CONSULTAR IGUAL.**

```
#!/bin/bash
```

```
if [$# -le 0]; then
 echo "ingresaste mal los parametros"
 exit 1
fi
```

```
arreglo=($@)
```

```
echo "arreglo: ${arreglo[*]}"
```

```
existe() {
 if [$# -ne 1]; then
 echo "ingresaste mal los parametros"
 return 1
 fi
 for ((i=0; i<${#arreglo[*]}; i++)); do
 if ["${arreglo[$i]}" = "$1"]; then
 echo "$i"
 return 0
 fi
 done
 return 1
}
```

```
replace() {
 if [$# -ne 2]; then
 echo "ingresaste mal los parametros"
 return 1
 fi
 #si existe, me carga el indice en la variable, sino me carga vacio.
 indice=$(existe $1)
 if [$? -eq 0]; then
 echo "realizando reemplazo"
 arreglo[$indice]="$2"
 return 0
 else
 echo "no se lo encontro asi que no se hizo el reemplazo"
 return 1
 fi
}
```

```
eliminar () {
 if [$# -ne 1]; then
 echo "ingresaste mal los parametros"
 return 1
 fi
```

```

 indice=$(existe $1)
 if [-n "$indice"]; then
 echo "eliminando"
 unset arreglo[$indice]
 return 0
 else
 echo "no se lo encontro para eliminarlo"
 return 2
 fi
}

cantidad () {
 echo "la cantidad de elementos es: ${#arreglo[*]}"
}

todos () {
 echo "${arreglo[*]}"
}

```

existe 5

existe 20

replace 3 90

replace 77 4

todos

eliminar 90

todos

cantidad

16- Escribir un script que reciba como primer parámetro un directorio, luego uno o más nombres de archivos (no sabemos cuántos). 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 existir, si es un archivo e informar por pantalla si poseemos permiso de ejecución sobre el mismo y si es un directorio, informar si poseemos permiso de escritura. Recuerde validar los parámetros del Script

**Consultar si tenes tiempo, funkio.**

```

#!/bin/bash
if [$# -lt 2]; then
 echo "ingresaste mal los parametros"

```

```

 exit 1
 fi

 if [! -d $1]; then
 echo "el directorio pasado como parametro no existe"
 exit 1
 fi

 # tenes que obviar el primer parametro
 arreglo=()
 for ((i=2; i<=$#; i++)); do
 arreglo+=(${!i})
 done

 echo "arreglo: ${arreglo[*]}"

 for archivo in ${arreglo[*]}; do
 echo "archivo: $archivo"
 if [-e $archivo]; then
 if [-d $archivo]; then
 echo "es un directorio, vamos a ver si tiene permisos de lectura..."
 if [-r $archivo]; then
 echo "tenes permisos de lectura"
 else
 echo "no tiene permisos de lectura"
 fi
 fi
 if [-f $archivo]; then
 echo "es un archivo regular, vamos a ver si tiene permisos de ejecucion"
 if [-x $archivo]; then
 echo "tenes permisos de ejecucion"
 else
 echo "no tiene permisos de ejecucion"
 fi
 fi
 fi
 fi

done

```

#### A continuación: Parcial completo(alumnos que adeudan autoevaluaciones)

17- Escriba un script que obtenga los nombres de los usuarios del sistema de la manera mas certera posible. Luego para cada usuario debe verificar si en su directorio personal existe al menos un archivo llamado detect, Si se encuentra al menos uno el script debe terminar exitosamente. Caso contrario debe terminar el script con el código de error 1. Recuerde validar los parámetros del Script

Pregunta alguno tiene el 16 hecho? tengo una duda al preguntar por los parametros

CONSULTAR SI O SI, tengo duda, cuando me pide verificar si en su directorio personal existe al menos un archivo llamado detect, yo al usar grep me devuelve las lineas que contienen detect, puede llamarse el archivo holadetect y no "detect" solo, esta mal?

LO TENGO EN LA NOTEBOOK, PERO SOLO CAMBIE LO Q ESTA EN VERDE.

```
#!/bin/bash
```

```
#me quedo con el usuario y su directorio personal
```

```
arreglo=$(cat /etc/passwd | cut -d":" -f1,6)
```

```
echo ${arreglo[*]}
```

```
for usuario in ${arreglo[*]}; do
```

```
 direc=$(echo "$usuario" | cut -d":" -f2)
```

```
 #anaizas que el direc es un directorio pq puede tener vacio o una x me dijo el profe.
```

```
 if [-f $direc/home]; then (el if de abajo no iria)
```

```
 if [$(ls "$direc" | grep "detect" | wc -l) -gt 0]; then
```

```
 echo "se encontro al menos un archivo llamado detect en $direc"
```

```
 exit 0
```

```
 else
```

```
 echo "no se lo encontro"
```

```
 fi
```

```
done
```

```
echo "no lo encontro"
```

```
exit 1
```