

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

Un Shell Script es un programa que está creado con instrucciones que son ejecutadas por un Shell (CLI o intérprete de comandos) de Unix o Linux. El código no es compilado ni precompilado, se va ejecutando línea por línea efectuando lo que cada instrucción le indica. Necesita un programa que entienda los comandos y estructuras que contiene y esto se suele poner en la primera línea del programa. Por ejemplo `#!/bin/bash` significa que le pasaremos al BASH las líneas del fichero de Script. Dado que el BASH es el intérprete de comandos más famoso de Linux, los Script que se crean para este entorno también se pueden llamar Bash Script.

Principalmente sirve para automatizar tareas y para realizar procesos más complejos de los que un solo comando puede efectuar. Aunque los comandos se pueden enlazar mediante tuberías o XARGS, a veces necesitamos tomar decisiones condicionales o recorrer elementos mediante bucles. Aquí es donde necesitamos organizarlo todo en un Shell Script.

El propio sistema de Linux tiene programadas multitud de tareas con sus Scripts del sistema, desde la rotación de logs, actualización del arranque, gestión de servidores, niveles de ejecución etc.

## 2. Investigar la funcionalidad de los comandos echo y read

echo: imprime el texto.

read: lee una línea desde entrada estándar.

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

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

- Para crear una variable:

```
NOMBRE="pepe" # SIN espacios alrededor del =
```

- Para accederla se usa \$:

```
echo $NOMBRE
```

- Para evitar ambigüedades se pueden usar llaves:

```
# Esto no accede a $NOMBRE
echo $NOMBREesto_no_es_parte_de_la_variable
# Esto sí
echo ${NOMBRE}esto_no_es_parte_de_la_variable
```

## 3. Crear dentro del directorio personal del usuario logueado un directorio llamado practicashell-script y dentro de él un archivo llamado mostrar.sh cuyo contenido sea el siguiente:

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

```
# Comentarios acerca de lo que hace el script
```

```
# Siempre comento mis scripts, si no hoy lo hago
```

# y mañana ya no me acuerdo de lo que quise hacer

echo "Introduzca su nombre y apellido:"

read nombre apellido

echo "Fecha y hora actual:"

date

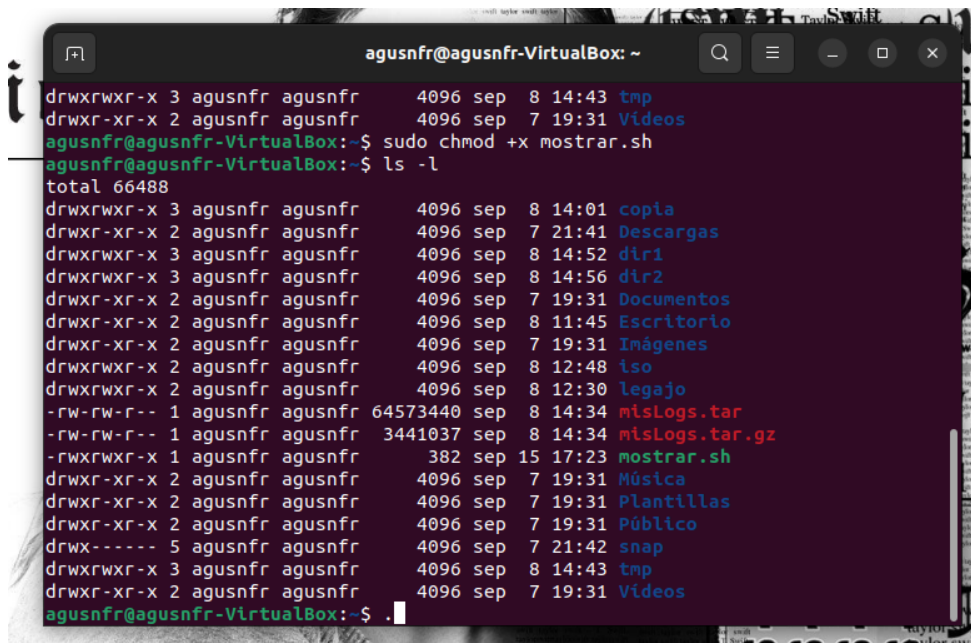
echo "Su apellido y nombre es: " → falta una comilla aca

echo "\$apellido \$nombre"

echo "Su usuario es: `whoami`"

echo "Su directorio actual es:"

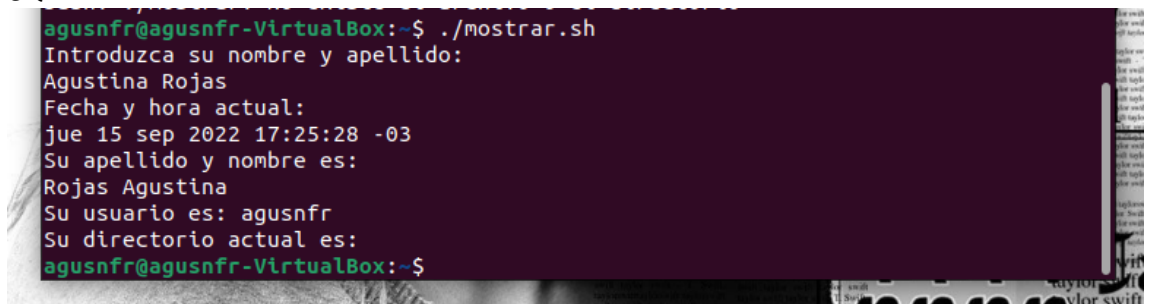
(a) Asignar al archivo creado los permisos necesarios de manera que pueda ejecutarlo



```
agusunfr@agusunfr-VirtualBox: ~  
drwxrwxr-x 3 agusunfr agusunfr 4096 sep 8 14:43 tmp  
drwxr-xr-x 2 agusunfr agusunfr 4096 sep 7 19:31 Videos  
agusunfr@agusunfr-VirtualBox:~$ sudo chmod +x mostrar.sh  
agusunfr@agusunfr-VirtualBox:~$ ls -l  
total 66488  
drwxrwxr-x 3 agusunfr agusunfr 4096 sep 8 14:01 copia  
drwxr-xr-x 2 agusunfr agusunfr 4096 sep 7 21:41 Descargas  
drwxrwxr-x 3 agusunfr agusunfr 4096 sep 8 14:52 dir1  
drwxrwxr-x 3 agusunfr agusunfr 4096 sep 8 14:56 dir2  
drwxr-xr-x 2 agusunfr agusunfr 4096 sep 7 19:31 Documentos  
drwxr-xr-x 2 agusunfr agusunfr 4096 sep 8 11:45 Escritorio  
drwxr-xr-x 2 agusunfr agusunfr 4096 sep 7 19:31 Imágenes  
drwxrwxr-x 2 agusunfr agusunfr 4096 sep 8 12:48 iso  
drwxrwxr-x 2 agusunfr agusunfr 4096 sep 8 12:30 legajo  
-rw-rw-r-- 1 agusunfr agusunfr 64573440 sep 8 14:34 misLogs.tar  
-rw-rw-r-- 1 agusunfr agusunfr 3441037 sep 8 14:34 misLogs.tar.gz  
-rw-rw-r-x 1 agusunfr agusunfr 382 sep 15 17:23 mostrar.sh  
drwxr-xr-x 2 agusunfr agusunfr 4096 sep 7 19:31 Música  
drwxr-xr-x 2 agusunfr agusunfr 4096 sep 7 19:31 Plantillas  
drwxr-xr-x 2 agusunfr agusunfr 4096 sep 7 19:31 Público  
drwx----- 5 agusunfr agusunfr 4096 sep 7 21:42 snap  
drwxrwxr-x 3 agusunfr agusunfr 4096 sep 8 14:43 tmp  
drwxr-xr-x 2 agusunfr agusunfr 4096 sep 7 19:31 Videos  
agusunfr@agusunfr-VirtualBox:~$ .
```

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

(c) ¿Qué resultado visualiza?



```
agusunfr@agusunfr-VirtualBox:~$ ./mostrar.sh  
Introduzca su nombre y apellido:  
Agustina Rojas  
Fecha y hora actual:  
jue 15 sep 2022 17:25:28 -03  
Su apellido y nombre es:  
Rojas Agustina  
Su usuario es: agusunfr  
Su directorio actual es:  
agusunfr@agusunfr-VirtualBox:~$
```

(d) Las backquotes (`) entre el comando whoami ilustran el uso de la sustitución de comandos. ¿Qué significa esto?

Permite utilizar la salida de un comando como si fuese una cadena de texto normal, es equivalente a hacer \$(whoami)

- (e) Realizar modificaciones al script anteriormente creado de manera de poder mostrar distintos resultados (cuál es su directorio personal, el contenido de un directorio en particular, el espacio libre en disco, etc.). Pida que se introduzcan por teclado (entrada estándar) otros datos.



```
#!/bin/bash
# Comentarios acerca de lo que se hace en el script
# Siempre comento mis scripts, si no hoy lo hago
# y mañana ya no me acuerdo que es lo que quise hacer
echo "Introduzca su nombre y apellido:"
read nombre apellido
echo "Fecha y hora actual:"
date
echo "Su apellido y nombre es: $apellido $nombre"
echo "Su usuario es: `whoami`"
echo "Su directorio actual es: $(pwd)"
echo "Su directorio personal es: $HOME"
echo "El contenido de $HOME es: $(ls $HOME)"
echo "Ingrese su numero de legajo: "
read legajo
echo "Su numero de legajo es: $legajo"
echo "El espacio libre en el disco es: $(df -h)"
~
~
~
~
~
"mostrar.sh" 17L, 604B 17,48 Todo
```

4. Parametrización: ¿Cómo se acceden a los parámetros enviados al script al momento de su invocación? ¿Qué información contienen las variables \$#, \$\*, \$? Y \$HOME dentro de un script?

- Los *scripts* pueden recibir argumentos en su invocación.
- Para accederlos, se utilizan variables especiales:
  - \$0 contiene la invocación al script.
  - \$1, \$2, \$3, ... contienen cada uno de los argumentos.
  - \$# 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.

\$HOME es el directorio de trabajo por defecto del usuario.

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

Para terminar un *script* usualmente se utiliza la función `exit`:

- Causa la terminación de un *script*
- Puede devolver cualquier valor entre 0 y 255:
  - El valor 0 indica que el script se ejecutó de forma exitosa
  - Un valor distinto indica un código de error
  - Se puede consultar el *exit status* imprimiendo la variable `$?`

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

Este comando devuelve en su salida estándar el resultado de las expresiones aritméticas pasadas como argumentos. Su sintaxis es `expr expresión`.

Todos los elementos de la expresión deben ir separados por al menos un espacio, y ciertos operadores aritméticos llevan como prefijo una barra invertida para evitar toda confusión con los caracteres especiales del shell.

Operaciones básicas: suma, resta, multiplicación, división y módulo en números enteros.

Otras: evaluación de expresiones regulares, operaciones de cadena como subcadena, longitud de cadenas, etc.

Los operadores aritméticos son:

`+`: suma

`-`: resta

`\*`: multiplicación

`/`: división entera

`%`: resto de la división entera o módulo

`\(` y `\)`: paréntesis

Se utiliza generalmente una sustitución de comandos para asignar el resultado del comando `expr` a una variable. Se obtiene por ejemplo:

```
[agusnfr]$ expr 2 + 3
```

```
5
```

```
[agusnfr]$ expr 2 - 3
```

```
-1
```

```
[agusnfr]$ expr 2 + 3 \* 4
```

```
14
```

```
[agusnfr]$ expr \( 2 + 3 \) \* 4
```

20

```
[agusnfr]$ resultado=$(expr 9 / 2)
```

```
[agusnfr]$ echo $resultado
```

4

```
[agusnfr]$ expr $resultado % 3
```

1

<https://linuxhint.com/linux-expr-command/> mas info ahí

**7. El comando “test expresión” permite evaluar expresiones y generar un valor de retorno, true o false. Este comando puede ser reemplazado por el uso de corchetes de la siguiente manera [ expresión ]. Investigar que tipo de expresiones pueden ser usadas con el comando test. Tenga en cuenta operaciones para: evaluación de archivos, evaluación de cadenas de caracteres y evaluaciones numéricas.**

Comando test

El comando test permite efectuar una serie de pruebas sobre los archivos, las cadenas de caracteres, los valores aritméticos y el entorno de usuario.

Este comando tiene un código de retorno igual a cero cuando el test es positivo, y diferente de cero en caso contrario; esto permite utilizarlo en encadenamientos de comandos con ejecución condicional (&& y ||) o en las estructuras de control que veremos más adelante.

El comando test posee dos sintaxis: test expresión y [ expresión ], donde "expresión" representa el test que se debe efectuar.

Los espacios detrás del corchete de apertura y antes del corchete de cierre son obligatorios en la sintaxis [ expresión ]. En general, todos los elementos de sintaxis del comando test deben ir separados por al menos un espacio.

El resto de la sección presenta los principales operadores que componen las expresiones de test del comando.

Operadores para condition:

Operador	Con strings	Con números
Igualdad	"\$nombre" = "Maria"	\$edad -eq 20
Desigualdad	"\$nombre" != "Maria"	\$edad -ne 20
Mayor	A > Z	5 -gt 20
Mayor o igual	A >= Z	5 -ge 20
Menor	A < Z	5 -lt 20
Menor o igual	A <= Z	5 -le 20

<https://francisconi.org/linux/comandos/test>

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

- if
- case
- while
- for
- select

## IF

Decisión:

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

## CASE

Selección:

```
case $variable in
    "valor 1")
        block
    ;;
    "valor 2")
        block
    ;;
    *)
        block
    ;;
esac
```

## SELECT

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

<https://noviello.it/es/como-usar-select-en-bash-en-linux/>

### Menú de opciones:

#### Ejemplo:

```
select action in New Exit
do
case $action in
    "New")
        echo "
        Selected
        option
        is NEW"
        ;;
    "Exit")
        exit 0
        ;;
esac
done
```

Imprime:

y espera el número de opción  
por teclado

## FOR

- *C-style*:

```
for ((i=0; i < 10; i++))
do
    block
done
```

- Con lista de valores (*foreach*):

```
for i in value1 value2 value3 valueN;
do
    block
done
```

## WHILE

### while

```
while [ condition ] #Mientras se cumpla la condición
do
    block
done
```

## UNTIL

### until

```
until [ condition ] #Mientras NO se cumpla la condición
do
    block
done
```

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

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

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

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

Las variables son de tipo string o arreglos.

El shell es un lenguaje débilmente tipado (las variables en un idioma están fuertemente tipadas si tienen tipos explícitos; las variables sin tipos débiles son lenguajes débilmente tipados) y son diferentes de C. La definición de variables en la programación de shell no requiere un tipo, y no existe un concepto de tipo.

Si se puede definir arreglos.

- Creación:

```
arreglo_a=() # Se crea vacío
arreglo_b=(1 2 3 5 8 13 21) # Inicializado
```

- Asignación de un valor en una posición concreta:

```
arreglo_b[2]=spam
```

- Acceso a un valor del arreglo (En este caso las llaves no son opcionales):

```
echo ${arreglo_b[2]}
copia=${arreglo_b[2]}
```

- Acceso a todos los valores del arreglo:

```
echo ${arreglo[@]} # o bien ${arreglo[*]}
```

- Tamaño del arreglo:

```
${#arreglo[@]} # o bien ${#arreglo[*]}
```

- Borrado de un elemento (reduce el tamaño del arreglo pero no elimina la posición, solamente la deja vacía):

```
unset arreglo[2]
```

- Los índices en los arreglos comienzan en 0



11. ¿Pueden definirse funciones dentro de un script? ¿Cómo? ¿Cómo se maneja el pasaje de parámetros de una función a la otra?

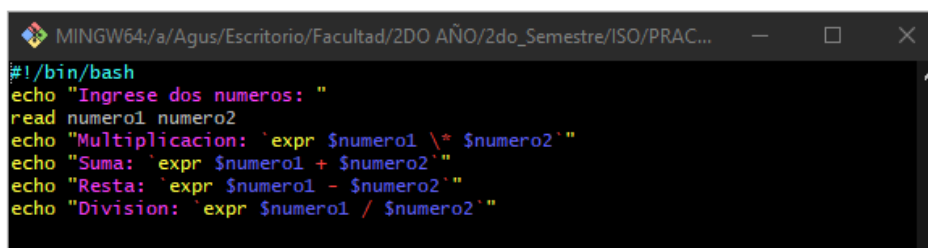
Las funciones permiten modularizar el comportamiento de los *scripts*.

- Se pueden declarar de 2 formas:
  - `function nombre { block }`
  - `nombre() { block }`
- Con la sentencia `return` se retorna un valor entre 0 y 255
- El valor de retorno se puede evaluar mediante la variable `$?`
- Reciben argumentos en las variables `$1`, `$2`, etc.

```
# Recibe 2 argumentos y devuelve:
# 1 si el primero es el mayor
# 0 en caso contrario
mayor()
{
    echo "Se van a comparar los valores $*"
    if [ $1 -gt $2 ]; then
        echo "$1 es el mayor"
        return 1
    fi
    echo "$2 es el mayor"
    return 0
}
mayor 5 6 # Invocación
echo $?   # Imprime el exit Status de la funcion
```

12. Evaluación de expresiones:

(a) Realizar un script que le solicite al usuario 2 números, los lea de la entrada Standard e imprima la multiplicación, suma, resta y cual es el mayor de los números leídos.



```
MINGW64:/a/Agus/Escritorio/Facultad/2DO AÑO/2do_Semestre/ISO/PRAC...
#!/bin/bash
echo "Ingrese dos numeros: "
read numero1 numero2
echo "Multiplicacion: `expr $numero1 \* $numero2`"
echo "Suma: `expr $numero1 + $numero2`"
echo "Resta: `expr $numero1 - $numero2`"
echo "Division: `expr $numero1 / $numero2`"
```

(b) Modificar el script creado en el inciso anterior para que los números sean recibidos como parámetros. El script debe controlar que los dos parámetros sean enviados.

```
#!/bin/bash
if [ $# -ne 2 ]; then
    exit 1
else
    echo "Multiplicacion: `expr $1 \* $2`"
    echo "Suma: `expr $1 + $2`"
    echo "Resta: `expr $1 - $2`"
    echo "Division: `expr $1 / $2`"
fi
exit 0
```

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

```
#!/bin/bash
if [ $# -ne 3 ]; then
    exit 1
else
    echo "Resultado: $(expr $2 $1 $3)"
fi
exit 0
```

### 13. Uso de las estructuras de control:

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

```
#!/bin/bash
if [ $# -ne 0 ]; then
    exit 1
else
    num=0
    while [ $num -ne 100 ]; do
        num=$((expr $num + 1))
        echo "Numero: $num"
        echo "Cuadrado: $(expr $num \* $num)"
    done
fi
exit 0
```

(b) Crear un script que muestre 3 opciones al usuario: Listar, DondeEstoy y QuienEsta. Según la opción elegida se le debe mostrar:

Listar: lista el contenido del directorio actual.

DondeEstoy: muestra el directorio donde me encuentro ubicado.

QuienEsta: muestra los usuarios conectados al sistema.

```
#!/bin/bash
if [ $# -ne 0 ]; then
    exit 1
else
    select var in Listar DondeEstoy QuienEsta
    do
        case $var in
            "Listar")
                echo "$(ls)"
                ;;
            "DondeEstoy")
                echo "$(pwd)"
                ;;
            "QuienEsta")
                echo "$(who)"
                ;;
        esac
    done
fi
exit 0
```

- (c) Crear un script que reciba como parámetro el nombre de un archivo e informe si el mismo existe o no, y en caso afirmativo indique si es un directorio o un archivo. En caso de que no exista el archivo/directorio cree un directorio con el nombre recibido como parámetro.

```
#!/bin/bash
if [ $# -ne 1 ]; then
    exit 1
else
    if [ -f $1 ]; then
        echo "Existe, es un archivo"
    elif [ -d $1 ]; then
        echo "Existe, es un directorio"
    else
        echo "No existe, vamos a crearlo"
        mkdir $HOME/$1
        echo "$(ls $HOME)"
    fi
fi
exit 0
```

14. Renombrando Archivos: haga un script que renombre solo archivos de un directorio pasado como parametro agregandole una CADENA, contemplando las opciones:

“-a CADENA”: renombra el fichero concatenando CADENA al final del nombre del archivo

“-b CADENA”: renombra el fichero concantenado CADENA al principio del nombre del archivo

```
#!/bin/bash
if [ $# -ne 3 ]; then
    echo "No se enviaron los parametros necesarios"
    exit 1
else
    if [ -d $1 ]; then
        case $2 in
            "-a")
                for FILE in $(ls $1); do
                    mv -v $1$FILE $1$FILE$3
                done
                ;;
            "-b")
                for FILE in $(ls $1); do
                    mv -v $1$FILE $1$3$FILE
                done
                ;;
            *)
                echo "No se envio correctamente el parametro"
                exit 1
                ;;
        esac
    else
        echo "No se envio un directorio"
    fi
fi
exit 0
```

15. Comando cut. El comando cut nos permite procesar la líneas de la entrada que reciba (archivo, entrada estándar, resultado de otro comando, etc) y cortar columnas o campos, siendo posible indicar cual es el delimitador de las mismas. Investigue los parámetros que puede recibir este comando y cite ejemplos de uso.

<https://geekland.eu/uso-del-comando-cut-en-linux-y-unix-con-ejemplos/#:~:text=%C2%BFQU%C3%89%20HACE%20EXACTAMENTE%20LA%20UTILIDAD,de%20un%20fichero%20de%20texto.>

<https://www.ochobitshacenunbyte.com/2019/06/09/ejemplos-practicos-del-comando-cut-en-linux/>

16. Realizar un script que reciba como parámetro una extensión y haga un reporte con 2 columnas, el nombre de usuario y la cantidad de archivos que posee con esa extensión. Se debe guardar el resultado en un archivo llamado reporte.txt

```
#!/bin/bash
if [ $# -ne 1 ]; then
    echo "No se pasaron bien los parametros"
    exit 1
else
    for usuarios in $(cat /etc/passwd | cut -d: -f1); do
        var=$(sudo find /home -user ${usuarios} -name "$1" | wc -l)
        echo " USUARIO: $usuarios | CANTIDAD: $var" >> $HOME/reporte.txt
    done
fi
exit 0
```

17. Escribir un script que al ejecutarse imprima en pantalla los nombre de los archivos que se encuentran en el directorio actual, intercambiando minúsculas por mayúsculas, además de eliminar la letra a (mayúscula o minúscula). Ejemplo, directorio actual:

```
#!/bin/bash
if [ $# -ne 0 ]; then
    echo "No se deben recibir parametros"
    exit 1
else
    for file in $(ls); do
        newName=$(echo $file | tr 'A-Z' 'a-z' | tr -d "a")
        mv $file $newName
    done
fi
exit 0
```

No se como hacer que se intercambien.

18. Crear un script que verifique cada 10 segundos si un usuario se ha logueado en el sistema (el nombre del usuario será pasado por parámetro). Cuando el usuario finalmente se loguee, el programa deberá mostrar el mensaje "Usuario XXX logueado en el sistema" y salir.

```
#!/bin/bash
if [ $# -ne 1 ]; then
    exit 1
else
    while [ $1 != "$(who | cut -d" " -f1 | grep $1)" ]; do
        echo "Buscando..."
        sleep 10
    done
    echo "Usuario $1 logeado en el sistema"
fi
exit 0
```

```
agusnfr@agusnfr-VirtualBox:~$ ./ejercicio18.sh ARojas
Buscando...
Buscando...
Buscando...
Usuario ARojas logeado en el sistema
```