ÍNDICE

INICIACIÓN A SCRIPS....2
VARIABLES....2
1. Concatenación de variables....3
2. Operaciones matemáticas....3
3. Almacenar en variables datos introducidos por el usuario....4
4. Almacenar el resultado de un comando en una variable....5
ESTRUCTURA CONDICIONALES....5
1. If....6
a) Comparar cadenas alfanuméricas....6
b) Comparación con números....7
2. If…. else....7
3. if...elif...else....8
4. Operadores condicionales para ficheros y directorios....8
ESTRUCTURA REPETITIVAS....9
1. WHILE Y UNTIL....9
2. LEER FICHERO CON WHILE....11
a) Leer fichero con un solo campo....11
b) Leer fichero con varios campos separados por espacios....12
c) Leer fichero con varios campos separados por un carácter especial....12
3. ESTRUCTURA FOR....13
a) Estructura for ((inicializador; condición; contador))....13
b) Estructura for variable in “dato1” “dato2” ….....13
c) Pasar como parámetro la ejecución de un comando....13
4. ESTRUCTURA CASE....14
PASAR PARÁMETROS....15
1. Comando shift – desplazamiento de parámetros....15
FUNCIONES....16
1. Incluir ficheros de funciones en un scripts....17

INICIACIÓN A SCRIPS

Un shell script es un archivo con comandos de una shell(en nuestro caso bash) que se ejecutan secuencialmente, se utilizan para automatizar tareas.

Todo script Bash comienza con la cabecera:

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

Por tanto si edito un archivo, lo guardo como miprimerscript.sh y dentro escribo:

```
#!/bin/bash
echo "Hola mundo"
```

Y para ejecutarlo debemos de darle permisos

```
chmod +x miprimerscript.sh
```

Y para ejecutarlo

```
./miprimerscript.sh
```

El resultado seria, mostrar por pantalla: Hola mundo.

Vamos a crea un escript que se mueva con el comando cd a un directorio y después mueva un fichero de ese directorio a otro.

```
#!/bin/bash
echo "Este es mi segundo scripts"
cd /home/rubenb/PracticaLinux/Sección1
mv fichero1.txt /home/rubenb/PracticaLinux/Sección2
```

VARIABLES

Las variables en bash son muy simples, ya que no tienen tipo definido ni necesitan ser declaradas antes de poder ser usadas. Para introducir valor en una variable simplemente se usa su nombre, y para obtener el valor de una variable se le antepone un símbolo dólar.

```
#!/bin/bash
DECIR=”Hola Mundo”
echo $DECIR
```

Este script realiza exactamente la misma función que el anterior, pero usando una variable.

1. Concatenación de variables

Para concatenar el contenido de dos variables solo hay que poner las dos variables con $ una a continuación de otra:

```
 #!/bin/bash
DECIR1=HOLA
DECIR2=MUNDO
DECIR3=${DECIR1}${DECIR2}
echo ${DECIR3}
```

El resultado de ese script sería:

```
HOLAMUNDO
```

Si queremos un espacio entre HOLA Y MUNDO DECIR3 se tendría que definir asi:

```
DECIR3=”${DECIR1}    ${DECIR2}”
```

2. Operaciones matemáticas

Como hemos dicho antes cualquier valor introducido en una variable se considera alfanumérico, por tanto si introducimos un número en una variable con esta variable no se pueden realizar operaciones matemáticas.

Por tanto si hacemos un scrips como el siguiente

```
#!/bin/bash
N1=5
N2=4
echo $N1 + $N2
```

El resultado es mostrar en pantalla

```
5 + 4
```

Para que las variables sean tratados como números se tiene meter entre $[]

```
#!/bin/bash
N1=5
N2=4
echo $[$N1 + $N2]
```

El resultado es mostrar en pantalla

```
9
```

Otra forma sería con la expresión expr

```
#!/bin/bash
N1=5
N2=4
RESULTADO=`expr $N1 + $N2` #Acuérdate de los espacios
echo $RESULTADO
```

La ejecución sería

```
9
```

Para el resto de operaciones sería

* Multiplicar

/ Dividir

- Restar

3. Almacenar en variables datos introducidos por el usuario

El scripts en un momento dado puede preguntar al usuario datos y almacenarlos en variables. La forma de pedir información es de la siguiente manera:

```
#!/bin/bash
#Muestra el mensaje en pantalla
echo “Introduce tu nombre:”
# Almacena en la variable nombre el dato introducido
```

```
read nombre
echo “Hola $nombre”
```

Al ejecutar lo anterior

```
$ ./vari.sh
“Introduce tu nombre:”
Ruben # Dato introducido por el usuario
“Hola Ruben”
```

También se puede hacer de la siguiente manera:

```
#!/bin/bash
#Muestra el mensaje en pantalla y almacena en la variable
nombre el dato introducido
read -p “Introduce tu nombre:” nombre
echo “Hola $nombre”
```

El resultado es el mismo

4. Almacenar el resultado de un comando en una variable

Para almacenar el resultado que devuelve la ejecución de un comando se hace como se ve en el siguiente ejemplo:

```
#!/bin/bash
resultado=`ps -ef`
echo $resultado
```

Al ejecutar lo anterior se mostraría en pantalla los procesos que están corriendo.

Con esto podemos almacenar en una variable objetos como el pid de un proceso para que después lo podamos matar:

```
#!/bin/bash
pid_proceso= `ps -ef |grep shutter |grep -v grep |tr -s " "
|cut -d " " -f2`
kill -9 ${pid_proceso}
```

ESTRUCTURA CONDICIONALES

Son estructuras que permiten ejecutar una parte del código en función de si se cumple o no una codición.

1. If

La principal estructura condicional de los scripts en shell es el if (Sí en inglés):

```
var=azul
if [ $var = “azul” ]
then
    echo “La variable var contiene azul”
fi
```

Después del if, entre corchetes tiene que haber una expresión lógica que produzca un resultado verdadero o falso.

En nuestro ejemplo comparamos el contenido de la variable var con la palabra azul, como la variable contiene la palabra azul, entonces esa comparación **es verdadera** y por tanto se ejecuta lo que hay dentro del if.

Si la variable hubiera tenido otra contenido, por ejemplo “rojo”, la comparación **no hubiera sido cierta** porque rojo no es igual a azul y por tanto no se ejecuta lo que hay dentro del if

Las comparaciones se hacen de forma diferente en función de si comparamos cadenas o numeros

a) Comparar cadenas alfanuméricas

Si estamos operando con cadenas alfanuméricas, los operadores que podemos utilizar son los siguientes:

Comparación de cadenas alfanuméricas	
Cadena1 = Cadena2	Verdadero si Cadena1 es IGUAL a Cadena2
Cadena1 != Cadena2	Verdadero si Cadena1 NO es IGUAL a Cadena2
Cadena1 \< Cadena2	Verdadero si Cadena1 es MENOR a Cadena2

Cadena1 \> Cadena2	Verdadero si Cadena1 es MAYOR que Cadena2

b) Comparación con números

Si estamos operando con números, los operadores que podemos utilizar son los siguientes:

Operadores de comparación de valores numéricos	
Numero1 -eq Numero2	Verdadero si Numero1 es IGUAL a Numero2 (equal)
Numero1 -ne Numero2	Verdadero si Numero1 NO es IGUAL a Numero2. (not equal)
Numero1 -lt Numero2	Verdadero si Numero1 es MENOR a Numero2. (less that)
Numero1 -gt Numero2	Verdadero si Numero1 es MAYOR que Numero2. (greater that)
Numero1 -le Numero2	Verdadero si Numero1 es MENOR O IGUAL que Numero2. (less or equal).

2. If..... else

En todo if se puede añadir un else para ejecutar el código en caso de que no se cumpla la condición del if.

Su estructura es:

```
var=azul
```

```
if [ $var = “azul” ]
then
    echo “La variable var contiene azul”
else
    echo “La variable var NO contiene el color azul”
fi
```

3. if...elif...else

Si se necesita añadir mas comprobaciones se puede utilizar la siguiente sentencia

```
var=azul
if [ $var = “rojo” ]
then
    echo “La variable var contiene rojo”
elif [ $var = “verde” ]
then
     echo “La variable var contiene verde”
elif [ $var = “azul” ]
then
 echo “La variable var contiene azul”
else
 echo “La variable var no contiene ni rojo, ni verde ni
azul”
fi
```

Esta estructura de control va comparando de arriba a abajo hasta que una de ellas se cumpla, en ese caso ejecuta el código de la condición que se cumple y no sigue comprobando mas.

4. Operadores condicionales para ficheros y directorios

Un uso muy común de shell scripts es la manipulación de ficheros(copiar, mover, buscar información dentro del fichero…), por tanto existen unos operadores que me permiten hacer comprobaciones sobre ficheros:

Operaciones condicionales usando test	
-a fichero	Verdadero si fichero existe
-d fichero	Verdadero si fichero existe, y es un fichero de tipo directorio

-f fichero	Verdadero si fichero existe, y es un fichero regular.
-s fichero	Verdadero si existe y tiene un tamaño mayor que cero

```
#!/bin/bash
# Almacenamos en una variable la fecha actual en formato
dd-mm-aa
fecha=`date "+%d-%m-%Y"`
fecha_menos_tres_dias=`date –date “-3 days”  "+%d-%m-%Y"`
cd /home/almacen
if[ -f datos.txt ]
then
  echo el fichero existe. Lo copio a mi directorio
 cp almacen/datos.txt /home/rubenb/datos/${fecha}_datos.txt
else
    echo El fichero datos.txt no existe.
fi
```

ESTRUCTURA REPETITIVAS

Son estructuras que me van a permitir repetir repetir sentencias hasta que se deje de cumplir una condición

1. WHILE Y UNTIL

La estructura del while es la siguiente:

```
while [ expresión ]
do
   echo “Hola mundo”
done
```

La estructura del until es la siguiente:

```
La estructura del until es la siguiente:
until [ expresión ]
do
   echo “Hola mundo”
done
```

Estas dos estructuras permiten ejecutar la sentencias “echo “hola mundo”, hasta que la expresión que hay entre [ ] deja de ser verdadera.

Veamos un ejemplo:

```
#!/bin/bash
#El primer paso es meter dentro de la variable $numero el valor 0
numero=0
#Comprueba si el contenido de numero es menor que 10 y si eso es cierto se mete dentro del while y ejecuta las sentencias que hay dentro.
while [ $numero -le 10 ]
do
   #Muestra por pantalla el siguiente mensaje
   echo “El contenido de numero es $numero”
   #Aumenta el valor de numero en uno
   numero=$[ $numero + 1 ]
done
```

Este scripts va a ejecutar las sentencias que están dentro del while, mientras el contenido de la variable numero sea menor que 10.

Como dentro del while cada vez que se ejecuta aumentamos el valor de numero en uno llegará un momento que el contenido de numero será mayor que 10 y por tanto dejará de ejecutarse.

Otro ejemplo

```
#!/bin/bash
#Metemos dentro de var el valor true(verdadero)
var=true
#Comprueba si el contenido de var es verdadero, si es verdadero ejecuta las sentencias que hay dentro.
while [ $var ]
do
   #Muestra por pantalla el siguiente mensaje
```

```
    echo “¿Quieres seguir ejecutando el while?(S|N)”
     read respuesta
    #Si la respuesta es N entra en el if y pone var a false
    #y por tanto hace que se termine el while
    if [ $respuesta = “N” ]
    then
       var=false
       echo “Me salgo del while”
    fi
done
```

Otra forma de hacer el código anterior:

```
#!/bin/bash
#Metemos dentro de var el valor true(verdadero)
var=1
#Comprueba si el contenido de var es verdadero, si es
verdadero ejecuta las sentencias que hay dentro.
while [ $var -eq 1 ]
do
    #Muestra por pantalla el siguiente mensaje
    echo “¿Quieres seguir ejecutando el while?(S|N)”
     read respuesta
    #Si la respuesta es N entra en el if y pone var a false
    #y por tanto hace que se termine el while
    if [ $respuesta = “N” ]
    then
       var=0
       echo “Me salgo del while”
    fi
done
```

2. LEER FICHERO CON WHILE

a) Leer fichero con un solo campo

La estructura while también se utiliza para leer el contenido de un fichero linea por linea.

Veamos como leemos el siguiente fichero que se llama fichero.txt

```
usuario1
usuario2
usuario3
```

El script es el siguiente:

```
while read variable
do
  echo La linea leída es $variable
done < fichero.txt
```

b) Leer fichero con varios campos separados por espacios

Cuando en un fichero hay varios campos separados por espacios cada campo se almacena en una variable.

Veamos como leemos el siguiente fichero que se llama fichero.txt

```
usuario1 1234 /home/usuario1
usuario2 2442 /home/usuario2
usuario3 4464 /home/usuario3
```

El script es el siguiente:

```
while read usuario contraseña directorio
do
  echo “”
  echo La linea leída es $usuario
  echo La linea leída es $contraseña
  echo La linea leída es $directorio
  echo “”
done < fichero.txt
```

c) Leer fichero con varios campos separados por un carácter especial

Puede ser que los campos en un fichero esten separados por un carácter, como puede ser el siguiente fichero que se llama fichero.txt

```
usuario1#1234#/home/usuario1
usuario2#2442#/home/usuario2
usuario3#4464#/home/usuario3
```

El script sería el mismo que en el punto anterior pero definiendo un separador de campos con la opción IFS

```
IFS=#
while read usuario contraseña directorio
do
  echo “”
  echo La linea leída es $usuario
  echo La linea leída es $contraseña
  echo La linea leída es $directorio
  echo “”
done < fichero.txt
```

3. ESTRUCTURA FOR

Los ciclos for permiten ejecutar una o varias instrucciones de forma iterativa. Son utilizados cuando se conoce el valor inicial y el valor final de las iteraciones, además permite indicar el tamaño del salto entre una iteración y otra.

a) Estructura for ((inicializador; condición; contador))

```
for (( inicializador; condición; contador ))
do
  #Instrucciones
done
```

```
#!/bin/bash
for (( i=10; i<20; i++ ))
do
  echo $i
done
```

La salida de este scripts sería 10 11 … hasta 19

b) Estructura for variable in “dato1” “dato2” …

Permite ejecutar el cuerpo del for por cada uno de los parámetros pasados.

```
#!/bin/bash
for nombre in "Adrés" "Juan"
do
        echo Hola $nombre
done
```

c) Pasar como parámetro la ejecución de un comando

El siguiente código copia todos los archivos de /etc al un directorio que se llame seguridad(este directorio tiene que estar en el directorio donde se ejecute el scripts)

```
#!/bin/bash
for file in `ls /etc`
do
        cp /etc/$file ./seguridad
done
```

4. ESTRUCTURA CASE

La estructura case me permite ejecutar un trozo de código u otro en función de una variable

```
 read x
 case $x in
     [1-2])
        echo "uno o dos"
     ;;
     3)
        echo "Tres"
     ;;
     *)
        echo "no se qué numero es"
     ;;
  esac
```

opciones múltiples

```
#!/bin/bash
#Control de flujo: case

echo "Escribe una frase"
read frase

case $frase in
  a*)
    echo "La frase empieza con a"
  ;;
  c*t)
    echo "La frase empieza con c y termina con t"
  ;;
  *com)
    echo "La frase termina con la cadena com"
  ;;
```

```
  *)
    echo "La frase no cumple con ninguna de las
condiciones"
  ;;
esac
```

PASAR PARÁMETROS

Se pueden pasar parámetros a nuestros scripts, y la forma de recogerlos dentro del scripts estos parámetros es con la siguientes variables:

```
$0 => contiene el nombre nombre de nuestro script
$# => el número de parámetros con los que se ha invocado al scripts
$1 => Primer parámetro
$2 => Segundo parámetro
…….

$$ => el PID de nuestro proceso
$* => todos los parámetros menos $0
```

Ejemplo

```
#!/bin/bash

if [ $# -ne 2 ]
then
    echo número de parámetros incorrecto.
    echo "USO: $0 <numero para sumar 1> <numero para suamar 2>"
    exit 5
fi

num1=$1
num2=$2

echo "La suma de $num1 + num2 es:"`echo $1 + $2 |bc`
```

1. Comando shift – desplazamiento de parámetros

La función shift lo que hace es cada vez que lo ejecutamos es desplazar hacia la izquierda una posición el contenido de las variables $1...${12}

Por ejemplo si ejecutamos el siguiente código:

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

```
if [ $# -ne 3 ]
then
    echo número de parámetros incorrecto. Se tienen que pasar 3 parámetros.
    exit 5
fi



echo El contenido de primer parámetro es $1
echo El segundo parámetro es $2
echo El tercer parámetro es $3

shift
echo

#Una vez ejecutado shift el contenido de $3 pasa a $2, el
contenido de $2 pasa a $1 y el de $1 se pierde.

echo El primer parámetro ahora es  $1
echo El segundo parámetro ahora es $2
echo El tercer parámetro ahora es $3
```

El resultado es

```
rubenb@pci600:~/scripts$ ./pasar_parametro.sh 44 66 77

El contenido de primer parámetro es 44
El segundo parámetro es 66
El tercer parámetro es 77

El primer parámetro ahora es 66
El segundo parámetro ahora es 77
El tercer parámetro ahora es
```

Ejemplo: Recorrer todos los parámetros pasados a un scripts

```
#!/bin/bash

while  [ -n "$1" ]
do
   #Muestra por pantalla el siguiente mensaje
   echo El parámetro de entrada es $1
   shift
done
```

FUNCIONES

El concepto de funciones existe en todos los lenguajes de programación, y no va a permitir la reutilización de código.

La forma de definir una función es:

```
function nombre_funcion()
{
      #CÓDIGO
}
```

Ejemplos:

Función par gestionar los errores

```
#!/bin/bash
function mensaje_error(
{
  echo Se ha producido el error $1
}

mensaje_error “Error al copiar fichero”
```

La forma de pasar parámetros a una función es exactamente igual que como se los pasamos a los scripts

1. Incluir ficheros de funciones en un scripts

Para estructurar los scripts se pueden crear ficheros donde se definan las funciones y las variables de entorno para posteriormente incluirlos en nuestros scripts.

Ejemplos

Suponemos que tenemos el siguiente fichero “funciones_y_entorno.sh” con las siguientes variables y funciones:

```
#!/bin/bash

##### VARIABLES
DIR_TRABAJO=/home/alumno/script
DIR_LOG=/home/alumno/log
DIR_DATOS=/home/alumno/datos
```

```
##### FUNCIONES
function mensaje_error(
{
  echo Se ha producido el error $1
}

function mensaje_suma(
{
  if [ $# -ne 2 ]
  then
      echo “Número de parámetros incorrecto”
      return 1
   fi

   echo "La suma de $num1 + num2 es:"`echo $1 + $2 |bc`
  return 0
}
```

Si queremos utilizar dichas funciones y variables en un scripts tenemos que añadir la ejecución de la siguiente manera

```
#!/bin/bash

. funciones_y_entorno.sh

echo scripts que llama a funciones de otro archivo

suma 34 24
```