CFGS DESARROLLO DE APLICACIONES WEB

MÓDULO:

Sistemas Informáticos

Sistema operativo Linux

BASH

INDICE DE CONTENIDOS

1. INTRODUCCIÓN 3
2. ESTRUCTURA DE UN SCRIPT 3
3. COMENTARIOS 4
4. SALIR DEL SCRIPT 4
5. VARIABLES 4
5.1 Usando variables de entorno 5
5.2 Asignado resultados de comandos a variables 6
5.3 Usando caracteres especiales en variables 6
5.4 Variables numéricas 7
6. ARGUMENTOS DE LOS SCRIPTS 7
6.1 El comando "shift" 8
7. FUNCIONES 8
8. CONTROL DE FLUJO 9
8.1 Condicionales con números 9
8.2 Condicionales con cadenas de texto 10
8.3 Condicionales con archivos 11
8.4 Condicionales con puertas lógicas 12
8.5 if/else 12
8.6 for 14
8.7 while 15
8.8 until 15
8.9 case 16
8.10 select 17
9. MANIPULACIÓN DE CADENAS DE TEXTO 17
9.1 Extraer subcadena 17
9.2 Borrar subcadena 18
9.3 Reemplazar subcadena 18
10. OPERACIONES ARITMÉTICAS 18

1. INTRODUCCIÓN

Al shell Bash, el intérprete de comandos estándar en Linux, le podemos pasar comandos desde una terminal o desde un script, que es un archivo de texto que contiene un pequeño programa.

2. ESTRUCTURA DE UN SCRIPT

Para empezar, debemos contar con un editor de texto. Los archivos que guardamos con extensión .sh podrán ser ejecutados (o interpretados) por la consola, siempre y cuando la primera línea sea la siguiente:

#!/bin/bash

Esto le dice al sistema que deberá usar la consola para ejecutar el archivo. Además, el carácter # permite escribir comentarios.

El script más sencillo es Hola mundo. Veamos cómo crearlo:

Crearemos el script con un editor de texto. En Linux disponemos de numerosos editores avanzados orientados a programación, con resaltado de sintaxis, como Gedit, Kwrite, Kate, Scite, etc.

Llamaremos a nuestro script hello.sh. A los scripts del shell se les suele poner la extensión .sh para identificarlos más fácilmente.

En un script podemos incluir cualquier comando que se pueda ejecutar en el intérprete de comandos. En nuestro caso, el contenido de hello.sh será el siguiente:

```
#!/bin/bash

echo "Hola, mundo."
```

la primera línea indica la aplicación que interpretará los comandos del script, en este caso Bash, cuyo ejecutable es /bin/bash.

le daremos al script permisos de ejecución con el comando chmod:

```
$ chmod a+x hello.sh
```

para ejecutar el script añadiremos al nombre del script la ruta del directorio actual, "./", ya que el directorio donde se encuentra (nuestra carpeta personal) no está en el PATH:

$ ./hello.sh

También podemos ejecutar el script llamando a bash:

$ bash hello.sh

En ambos casos obtendremos la salida por la salida estándar (la pantalla).

3. COMENTARIOS

Las líneas que comienzan por el carácter "#" (excepto la primera), se consideran comentarios y son ignoradas por el intérprete de comandos.

4. SALIR DEL SCRIPT

El comando para terminar el script es exit. Este comando no es útil al final del script, pero sí para terminar el programa en algún punto intermedio, por ejemplo "si se da tal condición, entonces terminar". Además, el comando exit permite establecer el valor de retorno, el valor devuelto por el script.

5. VARIABLES

Como cualquier otro lenguaje de programación, necesitamos variables que nos servirán para guardar datos en la memoria del ordenador hasta el momento que los necesitemos. Podemos pensar en una variable como una caja en la que podemos guardar un elemento (ej, un número, una cadena de texto, la dirección de un archivo…).

Para asignar el valor a una variable simplemente debemos usar el signo =.

Es importante no dejar espacios ni antes ni después del =.

En BASH, las variables se asignan simplemente dando el nombre de la variable y su valor:

#!/bin/bash

#Asignando variables

hola=1

Como podemos leer aquí, este código asigna el valor 1 a la variable "hola". En BASH las variables son "CASE SENSITIVE" (sensibles), es decir, la variable Hola no es lo mismo que HOLA ni que holA.

Otro de los puntos importantes de las variables en BASH es que no tenemos que asignarles un tipo, es decir, podemos darles a las variables cualquier valor y lo aceptará sin tener que decirle a BASH si es numérico o si son letras.

Para recuperar el valor de dicha variable sólo hay que anteponer el símbolo de dolar $ antes del nombre de la variable:

$nombre_variable

Las variables pueden tomar prácticamente cualquier nombre, sin embargo, existen algunas restricciones:

- Sólo puede contener caracteres alfanuméricos y guiones bajos.
- El primer carácter debe ser una letra del alfabeto o "_" (este último caso se suele reservar para casos especiales).
- No pueden contener espacios.
- Las mayúsculas y las minúsculas importan, "a" es distinto de "A".
- Algunos nombres son usado como variables de entorno y no los debemos utilizar para evitar sobrescribirlas (e.g.,PATH).

De manera general, y para evitar problemas con las variables de entorno que siempre están escritas en mayúscula, deberemos escribir el nombre de las variables en minúscula.

Además, aunque esto no es una regla que deba obedecerse obligatoriamente, es conveniente que demos a las variables nombres que más tarde podamos recordar. Si abrimos un script tres meses después de haberlo escrito y nos encontramos con la expresión "m=3.5" nos será difícil entender que hace el programa. Habría sido mucho más claro nombrar la variable como "media=3.5".

5.1 Usando variables de entorno

```
#!/bin/bash

echo "El usuario '$USERNAME' ha ejecutado el script $0, en el ordenador
'$HOSTNAME'. "
```

También se pueden usar en nuestro script, variables que no hemos definido nosotros. Estas variables son las llamadas 'variables de entorno' del sistema.

Algunas de estas variables son:

```
HOME         El directorio principal de usuario
```

```
PATH          Una lista de directorios, separados por comas, en los cuales
              el intérprete de comandos busca por comandos
BASH          La ruta de acceso completa usada para ejecutar la instancia
              actual de bash
BASH_VERSION El número de versión de bash usada
EUID          El identificador numérico de usuario del usuario actual
HOSTNAME      El nombre de maquina actual
PWD           Directorio actual definido por el comando 'cd'
UID           El valor numérico real del usuario actual
```

En nuestro ejemplo hemos utilizado $USERNAME y $HOSTNAME para obtener el nombre de usuario y del ordenador. La variable $0 contiene el nombre del script, más adelante explicaremos esto.

5.2 Asignado resultados de comandos a variables

```
#!/bin/bash

ATRIBUTOS_SCRIPT=`/bin/ls -l $0`

echo "El usuario '$USERNAME' ha ejecutado el script $0, en el ordenador '$HOSTNAME'. "

echo "Los atributos del script son: "

echo $ATRIBUTOS_SCRIPT
```

También podemos asignar la salida que producen los comandos del sistema a una variable. En nuestro ejemplo hemos asignado la salida del comando 'ls -l a una variable llamada ATRIBUTOS_SCRIPT. Esto nos será muy útil en nuestros scripts para obtener información del sistema que podremos utilizar en nuestros scripts.

5.3 Usando caracteres especiales en variables

Existen una serie de caracteres que tienen un significado especial en Bash, por ejemplo $ y ". Si queremos usar literalmente estos caracteres en el valor de una variable tendremos que usar el símbolo '\' delante de ellos.

```
#!/bin/bash

MENSAJE="\"Hola Mundo ...\""

echo "El valor de la variable \$MENSAJE es $MENSAJE"
```

5.4 Variables numéricas

Si queremos definir variables numéricas para su utilización en scripts Bash podemos utilizar el comando let.

```
#!/bin/bash
let  A=100
let  B=200
let C=$A+$B
echo "A: $A | B: $B | C: $C"
```

6. ARGUMENTOS DE LOS SCRIPTS

Es posible pasar a un script, en la línea de comando, los argumentos que necesita para su ejecución. Estos argumentos son llamados "parámetros".

Estos son pasados al script cuando un script es invocado.

Estos son almacenados en las variables reservadas 1,2,3,...9,10,11,... y pueden ser llamados con las expresiones $1,$2...${10},${11}...

El número de argumentos que se le pasa al script está contenido en la variable $#.

El conjunto de todos los argumentos está almacenado en la variable $*.

Veamos por ejemplo el siguiente script, llamado args.sh:

```
#!/bin/bash
echo  "\$0  contiene  $0"
echo  "\$1  contiene  $1"
echo "\$2 contiene $2"
echo "En total hay $# parámetros"
```

Ejecutamos este script sin pasarle argumentos:

```
$ ./args.sh
$0 contiene ./args.sh
$1 contiene
$2 contiene
```

En total hay 0 parámetros

Ahora ejecutamos el script pasándole dos argumentos:

$ ./args.sh buenos días

$0 contiene ./args.sh

$1 contiene buenos

$2 contiene días

En total hay 2 parámetros

ARGUMENTO DE ENTRADA INTERACTIVO

La función **read** nos permitirá escribir un valor, que bash asignará a una variable.

```
echo "Tu nombre: "
read nombre
read -p "Apellido: " apellido
```

6.1 El comando shift

El comando interno **shift** permite desplazar los parámetros.

El valor del 1er parámetro ($1) es reemplazado por el valor del 2do parámetro ($2), el del 2do parámetro ($2) por el del 3er parámetro ($3), …

Es posible indicar como argumento (shift n) el número de posiciones que hayque desplazar los parámetros.

7. FUNCIONES

En Bash se pueden definir funciones. En Bash las funciones se pueden definir dela siguiente manera:

```
function nombre_de_funcion(){
  comandos_del_shell
}
```

Un ejemplo de función en un script:

```
#!/bin/bash
let A=100
let B=200
function suma(){
```

```
 let C=$A+$B echo

 "Suma: $C"

}
function resta(){

 let C=$A-$B echo

 "Resta: $C"

}
#Llamamos a las funciones
suma
resta
```

8. CONTROL DE FLUJO

Los scripts se ejecutan línea a línea hasta llegar al final, sin embargo, muchas veces nos interesará modificar ese comportamiento de manera que el programa pueda responder de un modo u otro dependiendo de las circunstancias o pueda repetir trozos de código.

En Bash existen estas construcciones para controlar el flujo de ejecución de un script:

- for: Ejecuta una serie de comandos un número determinado de veces.
- if/else: Ejecuta una serie de comandos dependiendo si una cierta condición se cumple o no.
- while: Ejecuta una seria de comandos mientras que una determinada condición sea cumpla.
- until: Ejecuta una serie de comandos hasta que una determinada condición se cumpla.
- case: Ejecuta una o varias listas de comandos dependiendo del valor de una variable.
- select: Permite seleccionar al usuario una opción de una lista de opciones en un menú.

La mayoría de condiciones utilizadas con estas construcciones son comparaciones de cadenas alfanuméricas o numéricas, valores de terminación de comandos y comprobaciones de atributos de ficheros. Antes de seguir viendo como estas construcciones se pueden utilizar, vamos a ver como las condiciones se pueden definir.

8.1 Condicionales con números

Al comparar números podemos realizar las siguientes operaciones:

operador	significado
-lt	menor que (<)
-gt	mayor que (>)
-le	menor o igual que (<=)
-ge	mayor o igual que (>=)
-eq	igual (==)
-ne	no igual (!=)

EJEMPLO:

```
#!/bin/bash
# la variable $1 toma el primer valor que le pasamos al script
# la variable $2 toma el segundo valor que le pasamos al script
if [ $1 -gt $2 ];
then
  echo $1 es mayor que $2
else
  echo $2 es mayor que $1
fi
```

8.2 Condicionales con cadenas de texto

A la hora de comparar cadenas de texto:

operador	significado
=	igual, las dos cadenas de texto son exactamente idénticas
!=	no igual, las cadenas de texto no son exactamente idénticas
<	es menor que (en orden alfabético ASCII)
>	es mayor que (en orden alfabético ASCII)
-n	la cadena no está vacía
-z	la cadena está vacía

EJEMPLO:

```
#!/bin/bash
string1='reo'
string2='teo'
if [[ $string1 > $string2 ]];
then
  echo Eso es verdad
else
  echo Eso es mentira
fi
```

8.3 Condicionales con archivos

operador	Devuelve *true* si
-e name	*name* existe
-f name	*name* es un archivo normal (no es un directorio)
-s name	*name* NO tiene tamaño cero
-d name	*name* es un directorio
-r name	*name* tiene permiso de lectura para el user que corre el script
-w name	*name* tiene permiso de escritura para el user que corre el script
-x name	*name* tiene permiso de ejecución para el user que corre el script

Por ejemplo, podemos hacer un script que nos informe sobre el tipo de contenido de un directorio:

```
#!/bin/bash for
file in $(ls);do
  if [[ -d $file ]];
  then
    echo directorio: $file
  else
    if [[ -x $file ]];
    then
```

```
    echo archivo ejecutable: $file
  else
    echo archivo no ejecutable: $file
  fi
 fi
done
```

8.4 Condicionales con puertas lógicas (operadores lógicos)

&& (-a)	AND (Y)
|| (-o)	OR (O)

Se suelen usar en expresiones condicionales como por ejemplo " if ", bucles y en la consecuencia de ejecución de comandos.
A nivel sintáctico podemos declararlo de varias maneras:
ej:

Si se cumplen las dos condiciones:

if [$condition1] && [$condition2]
Por ejemplo: if [$número1=3] && [$número2=3]
if [$condition1 -a $condition2] Expresión también válida
if [[$condition1 && $condition2]] Expresión también valida

Si se cumple alguna de las dos condiciones:

if [$condition1] || [$condition2]
Por ejemplo: if [$número1=3] || [$número2=3]
if [$condition1 -o $condition2] Expresión también válida
if [[$condition1 || $condition2]] Expresión también válida

8.5 if/else

La sintaxis de esta construcción es la siguiente:

Las estructuras condicionales if permiten que nuestros scripts se comporten deuna forma u otra según la condición que especifiquemos. La sintaxis es:

```
if condicion1; then
    Bloque1
elif condicion2; then
    Bloque2
else
    Bloque3
fi
```

En el siguiente ejemplo pedimos al usuario un número que se almacena en la variable $input. Dependiendo de si el número es menor, igual o mayor que 5, el script se comporta de una forma u otra. Si el usuario no introduce nada o introduce una cadena de texto, se ejecutará lo que hay tras else, pues ninguna de las condiciones anteriores se cumple.

```
#!/bin/bash
echo "Introduzca un numero: "
read input
if [ $input -lt 5 ]; then
    echo "El número era menor que 5"
elif [ $input -eq 5 ]; then
    echo "El número era 5"
elif [ $input -gt 5 ]; then
    echo "El número era mayor que 5"
else
    echo "No introdujo un número"
fi
```

En el siguiente ejemplo, primero comprobamos que nos han pasado dos argumentos (el archivo donde buscar y la cadena de búsqueda): si no es así, le indicamos al usuario cómo debe ejecutar el script y salimos. A continuación, comprobamos que el archivo existe: si no es así, advertimos y salimos. Una vez verificadas estas dos condiciones, mediante cat y el pipe, grep cuenta el número de veces que el valor de $BUSQUEDA está en el archivo, que queda guardado en

$NUM_VECES.

```
#!/bin/bash
if [ $# -ne 2 ]; then
    echo "Necesito dos argumentos, el primero"
    echo "es el archivo donde debo buscar y"
    echo "el segundo es lo que quieres que busque."
    echo " "
    echo "Uso: $0 <archivo> <patrón_búsqueda>"
    echo " "
    exit
fi
ARCHIVO=$1
BUSQUEDA=$2
if [[ ! -e $ARCHIVO ]]; then
    echo "El archivo no existe"
    exit
fi
NUM_VECES=`cat "$ARCHIVO" | grep --count "$BUSQUEDA"`
if [ $NUM_VECES -eq 0 ];
then
    echo "El patrón de búsqueda "$BUSQUEDA" no fue encontrado"
    echo "en el archivo $ARCHIVO "
else
    echo "El patrón de búsqueda "$BUSQUEDA" fue encontrado"
    echo "en el archivo $ARCHIVO $NUM_VECES veces"
fi
```

8.6 for

La sintaxis de esta construcción es la siguiente:

```
for nombre [in lista]
do
   comandos que pueden utilizar $nombre
done
```

Un ejemplo nos aclarara las cosas. Vamos a listar información en el DNS de unalista de direcciones web:

```
#!/bin/bash

for HOST in www.google.com www.altavista.com www.yahoo.com
do
  echo "---------------------- "
  echo $HOST
  echo "---------------------- "
  /usr/bin/host $HOST
  echo "---------------------- "
done
```

8.7 while

La sintaxis de esta construcción es la siguiente:

```
while condición
do
  comandos
done
```

Un ejemplo simple con while en donde escribimos el valor de una variable 10 veces, después de incrementar su valor:

```
#!/bin/bash

NUM=0

while [ $NUM -le 10 ]; do

    echo "\$NUM: $NUM"

    let NUM=$NUM+1

done
```

8.8 until

La sintaxis de esta construcción es la siguiente:

```
until condición; do

   comandos

done
```

Un ejemplo simple con until en donde escribimos el valor de una variable 10 veces, después de incrementar su valor:

```
#!/bin/bash
NUM=0
until [ $NUM -gt 10 ]; do

    echo "\$NUM: $NUM"

    let NUM=$NUM+1

done
```

8.9 case

La sintaxis de esta construcción es la siguiente:

```
case expresión in

     caso_1 )

        comandos;;

     caso_2 )

           comandos;;

     ......

esac
```

Un ejemplo simple con case para aclarar las cosas:

```
#!/bin/bash
for NUM in 0 1 2 3
do
  case $NUM in
      0)
            echo "\$NUM es igual a cero";;
      1)
            echo "\$NUM es igual a uno";;
      2)
            echo "\$NUM es igual a dos";;
      3)
            echo "\$NUM es igual a tres";;
  esac
done
```

8.10 select

La sintaxis de esta construcción es la siguiente:

```
select nombre [in lista]
do
  comandos que pueden utilizar $nombre
done
```

Un ejemplo simple para aclarar las cosas.

```
#!/bin/bash

select OPCION in opcion_1 opcion_2 opcion_3
 do
  if [ $OPCION ]; then
```

```
    echo "Opción elegida: $OPCION"
    break
  else
    echo "Opción no valida"
  fi
done
```

9. MANIPULACIÓN DE CADENAS DE TEXTO

9.1 Extraer subcadena

Mediante ${cadena:posicion:longitud} podemos extraer una subcadena de otra cadena. Si omitimos :longitud, entonces extraerá todos los caracteres hasta el final de cadena. También se puede poner una longitud negativa (se debe poner entre paréntesis o un espacio antes del menos) para que se empiece por el final en vez de empezar por el principio.

Por ejemplo, en la cadena string=abcABC123ABCabc:
echo ${string:0} : abcABC123ABCabc

echo ${string:0:1} : a (primer carácter)

echo ${string:7} : 23ABCabc

echo ${string:7:3} : 23A (3 caracteres desde posición 7)

echo ${string:7:-3} : 23ABC

echo ${string: -4} : Cabc (atención al espacio antes del menos)echo

${string: -4:2} : Ca (atención al espacio antes del menos)

9.2 Borrar subcadena

Hay diferentes formas de borrar subcadenas de una cadena:

${cadena#subcadena} : borra la coincidencia más corta de subcadena desde el principio de cadena

${cadena##subcadena} : borra la coincidencia más larga de subcadena desdeel principio de cadena

Por ejemplo, en la cadena string=abcABC123ABCabc:

echo ${string#a*C} : 123ABCabc

echo ${string##a*C} : abc

9.3 Reemplazar subcadena

También existen diferentes formas de reemplazar subcadenas de una cadena:

${cadena/buscar/reemplazar} : Sustituye la primera coincidencia de buscar con reemplazar

${cadena//buscar/reemplazar} : Sustituye todas las coincidencias de buscarcon reemplazar

Por ejemplo, en la cadena string=abcABC123ABCabc:

echo ${string/abc/xyz} : xyzABC123ABCabc.

echo ${string//abc/xyz} : xyzABC123ABCxyz.

10. OPERACIONES ARITMÉTICAS

Bash también permite las operaciones aritméticas con número enteros:

+ - : suma, resta

~$ num=10

~$ echo $((num + 2))

**** : potencia**

~$ echo $((num ** 2))

*** / % : multiplicación, división, resto (módulo)**

~$ echo $((num * 2))

~$ echo $((num / 2))

~$ echo $((num % 2))

VAR++ VAR– : post-incrementa, post-decrementa

~$ echo $((num++))

~$ echo $num

++VAR –VAR : pre-incrementa, pre-decrementa

~$ echo $((++num))

~$ echo $num