

PROGRAMACIÓN DE SHELL SCRIPTS EN LINUX

El shell es un intérprete de órdenes, pero el shell no es solamente eso; los intérpretes de órdenes de Linux son auténticos lenguajes de programación. Como tales, incorporan sentencias de control de flujo, sentencias de asignación, funciones, etc.

Los programas de shell no necesitan ser compilados como ocurre en otros lenguajes. En este caso, el propio shell los ejecuta línea a línea. A estos programas se les conoce con el nombre de shell scripts y son los equivalentes a los archivos por lotes de otros sistemas operativos.

PASOS A SEGUIR:

- Crear un archivo de texto con un editor (vi, emacs, etc.). Este archivo contendrá las órdenes que el shell va a ir interpretando y ejecutando.
- Asignar permisos de ejecución al archivo creado, utilizando la orden chmod.
- Ejecutar el script generado pasándole los parámetros necesarios
./shell_script param1 param2 ...

RECOMENDACIONES:

- La primera línea de cada script debe contener:
#!/bin/bash
- Una línea de comentarios debe comenzar con el carácter # . Estos comentarios no se verán en pantalla cuando se ejecute el script.
- Para mostrar comentarios que luego veremos por pantalla, se utilizará el comando echo

PASO DE PARÁMETROS A UN PROGRAMA DE SHELL

A menudo queremos que nuestros programas de shell reciban parámetros desde la línea de órdenes para hacerlos más versátiles. Estos parámetros son lo que se conoce como parámetros de posición. Los parámetros de posición se pueden usar dentro de un programa de shell como cualquier otra variable de shell; es decir, para saber su valor utilizaremos el símbolo \$. Los parámetros dentro del shell script son accesibles utilizando las variables:

\$0	Representa al parámetro cero o nombre del programa
\$1	Representa al parámetro uno
\$2	Representa al parámetro dos
...	
\$9	Representa al parámetro nueve
\${10}	Representa al parámetro diez
\${11}	Representa al parámetro once
...	

Ejemplo1:

```
#!/bin/bash
echo El nombre del programa es $0
echo El primer parámetro recibido es $1
echo El segundo parámetro recibido es $2
echo El tercer parámetro recibido es $3
echo El cuarto parámetro recibido es $4
```

ALGUNAS VARIABLES ESPECIALES DEL SHELL

\$#	Número de parámetros que han pasado a la shell.
-----	---

- \$* Un argumento que contiene todos los parámetros que se han pasado (\$1, \$2...) menos el \$0.
- \$? Número donde se almacena el código de error del último comando que se ha ejecutado.
- \$\$ Número de proceso actual (PID)
- #! Último número de proceso ejecutado.

Ejemplo2:

```
#!/bin/bash
echo El nombre del programa es $0
echo El número total de parámetros es $#
echo Todos los parámetros recibidos son $*
echo El primer parámetro recibido es $1
echo El segundo parámetro recibido es $2
echo El tercer parámetro recibido es $3
echo El cuarto parámetro recibido es $4
```

shift

Sintaxis: shift n

Esta orden se utiliza para desplazar los argumentos, de manera que \$2 pasa a ser \$1, \$3 pasa a ser \$2, y así sucesivamente (esto si el desplazamiento n es igual a 1). Es muy utilizada dentro de los bucles.

Ejemplo3:

```
#!/bin/bash
#Este script se llama ej_shift2

echo El nombre del programa es: $0
echo El número total de parámetros es: $#
echo Todos los parámetros recibidos son: $*
echo El primer parámetro recibido es: $1
echo El segundo parámetro recibido es: $2
echo El tercer parámetro recibido es: $3
echo El cuarto parámetro recibido es: $4
shift 2
echo Ahora el parámetro \1 vale: $1
echo Ahora el parámetro \2 vale: $2
echo Ahora el parámetro \3 vale: $3
echo Ahora el parámetro \4 vale: $4
echo El número total de parámetros es: $#
echo Todos los parámetros recibidos son: $*
```

Mostrará el siguiente resultado:

```
$/ej_shift2 uno dos tres cuatro cinco seis
El nombre del programa es: ./ej_shift2
El número total de parámetros es: 6
```

Todos los parámetros recibidos son: uno dos tres cuatro cinco seis
El primer parámetro recibido es: uno
El segundo parámetro recibido es: dos
El tercer parámetro recibido es: tres
El cuarto parámetro recibido es: cuatro
Ahora el parámetro \$1 vale: tres
Ahora el parámetro \$2 vale: cuatro
Ahora el parámetro \$3 vale: cinco
Ahora el parámetro \$4 vale: seis
El número total de parámetros es: 4
Todos los parámetros recibidos son: tres cuatro cinco seis

La orden shift desplaza todas las cadenas en * a la izquierda n posiciones y decrementa # en n. Si a shift no se le indica el valor de n, por defecto tomará el valor 1. La orden shift no afecta al parámetro de posición 0 o nombre del programa.

read

Sintaxis: read variable (s)

La orden read se usa para leer información escrita en el terminal de forma interactiva. Si hay más variables en la orden read que palabras escritas, las variables que sobran por la derecha se asignan a NULL. Si se introducen más palabras que variables haya, todos los datos que sobran por la derecha se asignan a la última variable de la lista.

Ejemplo4:

```
#!/bin/bash
#script ej_read
#La opción -n se emplea para evitar el retorno de carro
echo -n "Introduce una variable: "
read var
echo La variable introducida es: $var
```

Ejemplo5:

```
#!/bin/bash
#script ej_read_var que lee varias variables con read
echo -n "Introduce las variables: "
read var1 var2 var3
echo Las variables introducidas son:
echo var1 = $var1
echo var2 = $var2
echo var3 = $var3
```

Si ejecutamos el Ejemplo5 sólo con dos parámetros podemos observar que la variable var3 queda sin asignar, puesto que solo hemos introducido dos valores:

```
$/ej_read_var
Introduce las variables: uno dos
Las variables introducidas son:
var1 = uno
```

```
var2 = dos
var3 =
$
```

A continuación ejecutamos el Ejemplo5 con cuatro parámetros y podemos observar que a la variable var3 se le asignan todas las variables a partir de la dos.

```
$/ej_read_var
```

Introduce las variables: uno dos tres cuatro

Las variables introducidas son:

```
var1 = uno
var2 = dos
var3 = tres cuatro
$
```

expr

Sintaxis: `expr arg1 op arg2 [op arg3 ...]`

Los argumentos de la orden `expr` se toman como expresiones y deben ir separados por blancos. La orden `expr` evalúa sus argumentos y escribe el resultado en la salida estándar. El uso más común de esta orden es para efectuar operaciones de aritmética simple y, en menor medida, para manipular cadenas.

Operadores aritméticos

+	Suma arg2 a arg1
-	Resta arg2 a arg1
*	Multiplica los argumentos
/	Divide arg1 entre arg2 (división entera)
%	Resto de la división entera entre arg1 y arg2

- En el caso de utilizar varios operadores, las operaciones de suma y resta se evalúan en último lugar, a no se que vayan entre paréntesis.
- No hay que olvidar que los símbolos `*`, `(` y `)` tienen un significado especial para el shell y deben ser precedidos por el símbolo backslash (`\`) o encerrados entre comillas simples.

Ejemplo6:

```
#!/bin/bash
#script que multiplica dos variables leídas desde teclado
echo "Multiplicación de dos variables"
echo "-----"
echo -n "Introduce la primera variable:"
read var1
echo -n "Introduce la segunda variable:"
read var2
resultado=`expr $var1 \* $var2`
echo El resultado de la multiplicación es =$resultado
```

El resultado de ejecutar el programa anterior es el producto de las dos variables leídas desde el teclado.

Operadores relacionales

Estos operadores se utilizan para comparar dos argumentos. Los argumentos pueden ser también palabras. Si el resultado de la comparación es cierto, el resultado es uno (1); si es falso, el resultado es cero (0). Estos operadores se utilizan mucho para comparar operandos y tomar decisiones en función de los resultados de la comparación. Los distintos tipos de operadores relacionales son:

=	¿Son los argumentos iguales?
!=	¿Son los argumentos distintos?
>	¿Es arg1 mayor que arg2?
>=	¿Es arg1 mayor o igual que arg2?
<	¿Es arg1 menor que arg2?
<=	¿Es arg1 menor o igual que arg2?

Los símbolos > y < tienen significado especial para el shell, por lo que deben ser entrecomillados.

Ejemplo7:

```
#!/bin/bash
#El script se llama ejemplo7
#script que determina si dos variables leídas desde teclado son iguales o no
echo "¿Son iguales las variables?"
echo "-----"
echo -n "Introduce la primera variable: "
read var1
echo -n "Introduce la segunda variable: "
read var2
resultado=`expr $var1 = $var2`
echo Resultado=$resultado
```

El programa anterior devolverá 0 si las dos variables introducidas son distintas y 1 si son iguales.

Veamos la ejecución de Ejemplo7:

```
$/ejemplo7
¿Son iguales las variables?
-----
Introduce la primera variable: dato
Introduce la segunda variable: dato
Resultado=1
$
```

Si las variables fuesen distintas, el resultado sería:

```
$/ejemplo7
¿Son iguales las variables?
-----
Introduce la primera variable: 123
Introduce la segunda variable: 45
Resultado=0
$
```

Operadores lógicos

Estos operadores se utilizan para comparar dos argumentos. Dependiendo de los valores, el resultado puede ser arg1 (o alguna parte de él), arg2 o cero.

Como operadores lógicos tenemos los siguientes:

	Or lógico. Si el valor de arg1 es distinto de cero, el resultado es arg1; si no es así, el resultado es arg2.
&	And lógico. Si arg1 y arg2 son distintos de cero, el resultado es arg1; si no es así, el resultado es arg2.
:	El arg2 es el patrón buscado en arg1. Si el patrón arg2 está encerrado dentro de paréntesis \(\), el resultado es la parte de arg1 que coincide con arg2. Si no es así, el resultado es simplemente el número de caracteres que coinciden.

No debemos olvidar que los símbolos | y & deben ser entrecomillados o precedidos del símbolo \, por tener un significado especial para el shell.

Ejemplo8: En este ejemplo incrementamos en una unidad el valor de la variable a.

```
$a=5
$a=`expr $a + 1`
$echo $a
6
$
```

Ejemplo9: En este ejemplo calculamos el número de caracteres de la cadena a.

```
$a=palabra
$b=`expr $a : "."`
$echo $b
7
$
```

Ejemplo10: En este ejemplo determinamos cuáles son los caracteres comprendidos entre la a y la z minúsculas en la cadena a.

```
$a=Febrero_1998
$b=`expr $a : "[a-z]*"`
$echo $b
ebrero
$
```

test (evaluación de archivos)

Sintaxis: test –opcion archivo
 test [expresión]

La orden test se usa para evaluar expresiones y genera un valor de retorno; este valor no se escribe en la salida estándar, pero asigna 0 al código de retorno si la expresión se evalúa como true, y le asigna 1 si la expresión se evalúa como false.

Cuando se invoque la orden test mediante la segunda sintaxis, hay que dejar un espacio en blanco entre los corchetes.

La orden test puede evaluar tres tipos de elementos: archivos, cadenas y números.

Opciones:

- f Devuelve verdadero (0) si el archivo existe y es un archivo regular (no es un directorio ni un archivo de dispositivo).
- s Devuelve verdadero (0) si el archivo existe y tiene un tamaño mayor que cero.
- r Devuelve verdadero (0) si el archivo existe y tiene un permiso de lectura.
- w Devuelve verdadero (0) si el archivo existe y tiene un permiso de escritura.
- x Devuelve verdadero (0) si el archivo existe y tiene un permiso de ejecución.
- d Devuelve verdadero (0) si el archivo existe y es un directorio.

Ejemplo11:

```
$test -f archivo32
$echo $?
1        (El archivo archivo32 no existe)
$
$ test -f /etc/passwd
$echo $?
0        (El archivo /etc/passwd sí existe)
$
```

test (evaluación de cadenas)

Sintaxis: [cadena1 = cadena2]
 [cadena1 != cadena2]

Ejemplo12:

```
$a=palabra1
$[ "$a" = "palabra2" ]
$ echo $?
1
$[ "$a" = "palabra1" ]
$ echo $?
0
$
```

De esta manera, test evalúa si las cadenas son iguales o distintas.

test (evaluaciones numéricas)

Sintaxis: número relación número

En evaluaciones numéricas, esta orden es sólo válida con números enteros. Los operadores usados para comparar números son diferentes de los usados para comparar cadenas. Estos operadores son:

-lt	Menor que
-le	Menor o igual que
-gt	Mayor que
-ge	Mayor o igual que
-eq	Igual a
-ne	No igual a

Hay unos cuantos operadores que son válidos en una expresión de la orden test a la hora de evaluar tanto archivos como cadenas o números. Estos operadores son llamados conectores y son:

-o	OR
-a	AND
!	NOT