Programación shell

Un shell es un programa interactivo que se interpone entre el usuario y el núcleo del sistema permitiendo que aquél envíe órdenes al sistema.

Estudiamos en la unidad anterior la forma de comunicarnos con el sistema de orden en orden o como mucho, agrupándolas con ; o con redireccioamientos.

En esta unidad aprenderemos a crear programas basados en instrucciones del sistema. Para ello las incluiremos en un fichero de texto (denominado script) que haremos ejecutable con la orden chmod (ver la unidad introducción a sesión GNU/Linux) para que puedan ser interpretados por el "intérprete" de comandos..

Estos programas, para que tengan una mayor potencia, constarán también de estructuras de flujo de control como son if, case, while o for que estudiaremos.

Existen varios programas shell que pueden instalarse en nuestrso sistemas GNU/Linux. Nosotros vamos a centrar el estudio en el shell bash.

Variables y entorno

El shell incorpora dos zonas de memoria para almacenar variables: el área de datos locales y el entorno.

Al definir una variable se le asigna memoria de la zona de datos locales, pero no es accesible por ningún otro shell que se inicie con posterioridad. Ningún sub-shell podrá utilizarla.

El entorno, en cambio, contiene variables globales que permanecen activas durante toda la sesión y son accesibles por cualquier sub-shell. Suelen crearse al acceder al sistema y se utilizan para determinar las características de la sesión.

No obstante, pueden incorporarse al entorno variables locales definidas durante la sesión con la orden export:

```
A="hola"
export A
```

Otros comandos relacionados con las variables:

set muestra la lista de variables locales definidas

env muestra la lista de varialbes iniciales del entono.

Variables posicionales

Son variables que adoptan como valores los argumentos introducidos a continuación del nombre del script en la línea de órdenes. Se nombran como $1, $2...$9. Se toma como delimitador cualquier separador admitido por el shell, normalmente el espacio. Ejemplo

```
prog1 hola 77 88
```

En este ejemplo $1 será hola, $2 será 77 y $3 será 88.

Además de éstas el shell genera automáticamente las siguientes variables:

$0	Nombre del guión ejecutado.
$*	Conjunto de todos los argumentos
$#	Número de argumentos introducidos
$?	Código de error generado por la última instrucción ejecutada.

Comandos relacionados:

shift Desplaza hacia la derecha los valores de los parámetros posicionales. Tiene como parámetro un valor entero, por defecto 1.

read Permite la introducción de valores en un guión shell tras solicitarlos al usuario:

ejemplo:

```
echo “Introduce tu edad”
read varA
```

Comando expr

Comando de cálculo aritmético entero. Admite las siguientes operaciones básicas:

Suma	+
Resta	-
Producto	*
División	/
Módulo	%

El resultado de una operación con expr puede asignarse a una variable, pero deben utilizarse los acentos graves para ejecutar la expresión antes de la asignación:

```
$X=`expr 7 + 8`
```

También pueden utilizarse variables en los cálculos:

```
expr $X + 1
```

Una expresión expr puede contener otras expresiones expr pero deben ir delimitadas por acentos graves para poder utilizar su resultado parcial en la expresión general:

expr `expr 7 + 8 ` / `expr $a + $b`

Operadores de comparación

Para comparar *valores numéricos:*

-eq	igual
-ne	no igual
-gt	mayor que
-lt	menor que
-ge	mayor o igual que
-le	menor o igual que

Ejemplo:

[$num1 -eq $num2] ----> será verdadero si num1 y num2 son iguales.
[$num1 -le $num2]-----> será veradero si num1 es mayor o igual que num2

Para comparar *cadenas*:

=	igual
!=	distinto
-z	Cadena de longitud 0
-n	Cadena de longitud mayor que 0

Ejemplo

A=hola
[-z $A]--->es falso pero,
[-n $A]--->es verdadero.

Sobre *nombres de archivo*

-a	archivo existente
-s	archivo existente y no vacío.
-f	archivo ordinario
-d	directorio
-w	archivo con permiso de escritura
-r	archivo con permiso de lectura
-x	archivo con permiso de ejecutable.

Ejemplo:

[-f bienvenido.sh] --> será verdadero si bienvenido.sh es un fichero.

Las comparaciones deben incluirse entre corchetes "[]" debiendo existir un espacio de separación entre el corchete de apertura y el primer caracter de la condición y un espacio entre el último carácter de la condición y el corchete de cierre.

Pueden combinarse condiciones con O e Y lógicos, siendo || el o lógico y && el y lógico.

Ejemplo:

[! $A -lt 7] && [$B -eq 0]

Para obtener el negado de la condición : **!**

if

Ejecuta una acción o bloque de acciones si se verifica una condición:

Sintaxis a:

if [condición]
then
bloque de instrucciones si se cumple la condición
fi

Sintaxis b:

if [condición]
then
bloque de instrucciones si se cumple la condición
else
bloque de instrucciones si no se cumple
fi

Sintaxis c: (o if anidados)

if [condición]
then
bloque de instrucciones si se cumple la condición
elif [condición 2]
then
bloque instrucciones si se cumple condición1 y no condición2
fi

case

Sentencia de selección múltiple

case expresión **in**

```
        patrón1) bloque de instrucciones;;
        patrón2) bloque de instrucciones;;
        patrón3) bloque de instrucciones;;
        ..
        ..
        *) bloque de instrucciones;;
    esac
```

Si “expresión” coincide con alguno de los patrones se ejecutga el bloque de instruccioens que figura a continuación.

El asterisco representa el patrón por defecto. Su bloque de instrucciones asociado se ejecuta en el caso de que “expresión” no coincida con ninguno de los patrones precedentes.

“expresión” puede ser una constante o una variable.

Los patrones pueden contener metacaracters (*,?,[]).

La última instrucción de cada bloque debe ir seguida de doble punto y coma.

```
case $carnet in
        E) echo “Remolques”;;
         D) echo “Autobuses”;;
        C1|C2) echo “Camiones”;;
        A*) echo “Motos”;;
        *) echo “No es ningún carnet”;;
esac
```

while

Bucle de instrucciones que se ejecutan reiteradamente mientras se cumple la condición

```
while [condición]
do
        bloque de instrucciones
done
```

Ejemplo:

```
n=1
while [ $n -lt 10 ]
do
        $n=`expr $n + 1`
```

```
        echo $n
done
```

Mostrará de 1 a 9, cada número en una línea.

for

Bucle de instrucciones que se ejecutan reiteradamente para cada uno de los valores que toma una variable entre los de una lista.

```
for variable in lista
do
        bloque de instrucciones
done
```

Ejemplo:

```
for compa in Marta Carlos Lucia
do
        echo $compa >> fichero
done
```

Añade al fichero “fichero” los nombres Marta, Carlos, Lucia.

Si no se incluyera in lista se tomarían los valores pasados como variables parámetros posicionales.