Shell (estructuras condicionales y repetitivas)

Moisés García Villanueva

28 de Agosto de 2012

1 Estructuras de control de flujo

La programación en shell dispone de las sentencias de control del flujo de instrucciones necesarias para poder controlar perfectamente la ejecución de las órdenes necesarias.

1.1 Estructuras condicionales

La estructura condicional `if` se describe en forma general para el shell de la siguiente forma:

```
if [ lista_órdenes ]
then
       lista_órdenes
fi
```

1.1.1 Operadores condicionales

En el shell existen tres tipos de operadores de comparación:

1. **De archivos**

`-e archivo`	Verdadero si archivo existe.
`-d archivo`	Verdadero si archivo existe y es un directorio.
`-f archivo`	Verdadero si archivo existe y es un archivo normal.
`-w archivo`	Verdadero si archivo existe y se puede escribir.
`-x archivo`	Verdadero si archivo existe y es ejecutable.
`archivo1 -nt archivo2`	Verdadero si archivo1 es más reciente que archivo2 (modificación)
`archivo1 -ot archivo2`	Verdadero si archivo1 es más antiguo que archivo2 (modificación)
`archivo1 -ef archivo2`	Verdadero si son el mismo dispositivo

2. **De cadenas**

`-z str`	Verdadero si la longitud de str es igual a cero.
`-n str`	Verdadero si la longitud de str es distinta de cero.
`str1 == str2`	Verdadero si las str son iguales.
`str1 = str2`	Verdadero si las str son iguales.
`str1 != str2`	Verdadero si las str son distintas.
`str1 <str2`	Verdadero si str1 está antes que str2 (depende del local).
`str1 >str2`	Verdadero si str1 está después que str2 (depende del local).

3. **Aritmética entera**

valor1 -gt valor2	valor1 mayor que valor2
valor1 -ge valor2	valor1 mayor o igual que valor2
valor1 -eq valor2	valor1 igual a valor2
valor1 -ne valor2	valor1 no igual a valor2
valor1 -lt valor2	valor1 menor que valor2
valor1 -le valor2	valor1 menor o igual que valor2

1.1.2 Ejemplos

Para comparar dos cadenas tenemos el siguiente ejemplo:

```
if [ $1 = $2 ]
then
        echo "son igules $1 y $2";
else
        echo "no son iguales";
fi
```

Para realizar un menú de números, podemos realizar lo siguiente:

```
num=$1;

if [ $num -eq 1 ]
then
        echo "Se recibió el número uno: $num, mostraremos el calendario del mes:";
        cal
elif [ $num -eq 2 ]
then
        echo "Se recibió el número dos: $num, mostraremos el calendario del año:";
        cal 2012;
elif [ $num -eq 3 ]
then
        echo "Se recibió el número tres: $num, se mostrará el mes anterior, actual
              y siguiente:\n"; cal -3;
elif [ $num -eq 100 ]
then
        echo -n "Con el $num indicamos la fecha: "; date +%d-%m-%Y
else
        echo "Opción no reconocida";
fi
```

1.1.3 Actividades

1. Hacer un script que recibe como parámetros de entrada tres números y nos indica cual es el mayor o menor de los tres. Adicionalmente se debe señalar si algunos de ellos son iguales.

2. Realizar un script que contenga un menú de opciones para los comandos `find`, `grep`, `tr` y `uniq`

1.2 Estructuras condicionales

El shell aporta mecanismos para realizar tareas repetitivas mediante el empleo de estructuras que permiten repetir un bloque de comandos.

1.2.1 for ... in ...

Esta estructura permite repetir un bloque de comandos asignando valores de una serie a una variable en cada iteración.

```
for VARIABLE in SERIE; do
    bloque de comandos
done
```

En cada iteración la variable `VARIABLE` toma un valor de `SERIE`, que en caso de no contener elementos hará que no se ejecute nada y se devuelva un valor `0`. En caso de que se ejecuten comandos, el resultado devuelto tras el bucle es el del último comando ejecutado.

Ejemplos de bucle:

```
# equivalente a seq 1 5
for i in 1 2 3 4 5; do
    echo $i
done

# lo mismo pero con palabras
for palabra in uno dos tres cuatro cinco; do
    echo $palabra
done
```

1.2.2 Actividades

1. Realizar un script que dado un número 'n' muestre los diez primeros elementos de su tabla de multiplicar, mostrando el resultado en la forma: `i x n = resultado`. Emplear un bucle y `seq` (si está disponible). Si no se proporciona un número, mostrar como se usa el programa.

2. Realizar un script que dado una lista de directorios, cree un archivo tar comprimido con gzip con nombre igual a la fecha en formato `yyyy-mm-dd.tar.gz`. Además se generará un archivo `yyyy-mm-dd.lst` con los nombres de los directorios contenidos en el archivo tar, UNO POR LINEA usando un bucle. Si el archivo `lst` existe, mostrar un error y terminar el programa. Si alguno de los elementos no es un directorio, mostrar un error y finalizar el programa.

3. Hacer un script que imprima lo siguiente:

```
P3
#Imagen PPM creada en el LABORATORIO DE COMPUTACIÓN DE LA FIE
10 10
255
10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 0
10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 0
10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 0
10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 0
10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 0
10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 0
10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 0
10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 0
10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 0
10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 255 10 10 0
```

4. Hacer un script que amplié 10 veces la imagen del ejercicio anterior y dibuje una línea de color rojo en los puntos (x_1, y_1) y (x_2, y_2) en la imagen resultante de la transformación.

2 Funciones

Una de las mayores utilidades que posee una shell es el permitirnos crear funciones para realizar tareas repetitivas fácilmente. El funcionamiento de estas funciones es parecido al que posee cualquier lenguaje de programación, en el cual se agrupan un conjunto de comandos y se los llama por un nombre. La estructura básica de una función es la siguiente:

```
nombre_funcion(){
  primero_a_realizar
  segundo_a_realizar
}
```

Las funciones pueden ser definidas en cualquier lugar, incluso en la misma línea de comandos. Lo que habrá que recordar que hechas de esta forma se borraran de la memoria una vez que se a salido del shell. Una forma de hacer que queden disponibles en forma permanente es incorporarla en el archivo de inicio del shell, el archivo `.bash_profile`, en el cual se pondrá la función. De esta forma podremos seguir utilizándola dado que será cargada en memoria cada vez que ejecute el shell.

2.0.3 Ejemplos

Veamos ahora un ejemplo de funciones:

```
 #!/bin/bash
let A=100
let B=200
#
# Funcion suma()
# Suma las variables A y B
#
function suma(){
  let C=$A+$B
  echo "Suma: $C"
}
#
# Funcion resta()
# Resta las variables A y B
#
function resta(){
  let C=$A-$B
  echo "Resta: $C"
}
suma
resta
```

Un ejemplo de una función que regresa un valor:

```
 #!/bin/bash
let A=100
let B=200
#
# Funcion suma
# Suma los dos parametros que se le dan
#
function suma () {
  let C=$1+$2
  return $C
}
#
# Funcion resta()
```

```
# Resta las variables A y B
#
function resta () {
  let C=$1-$2
  return $C
}
suma $A $B
echo Suma: $?
resta $A $B
echo Resta: $?
```

2.1 Actividades

1. Hacer una función que recibe dos argumentos y nos regresa la concatenación de los argumentos.
2. Hacer una función que crea una imagen PPM del color y tamaño que recibe como parámetros.

3 Arreglos

Los arreglos son un conjunto de variables del mismo tipo representadas por un nombre. EN el shell existen varias formas de representar arreglos:

- utilizando los paréntesis:

  ```
  nombres=( Juan Pedro José Jesús María )
  ```

- Indicando las posiciones o índices en el arreglo:

  ```
  colores[0]=Rojo;colores[1]=Verde;
  colores[3]=Azul;colores[5]=Amarillo;
  ```

- A partir de un archivo:

  ```
  arregloDesdeArchivo=( ‘cat archivo | tr ’\n’ ’ ’‘ );
  ```

Operaciones con los arreglos:

- Para imprimir todos los elementos del arreglo utilizamos la expresión: `${arreglo[@]}`.
- Los elementos individuales de un arreglo pueden ser accesados por el índice de la forma siguiente:

  ```
  ${colores[3]}
  echo ${nombres[3]}
  ```

- El número de elementos en el arreglo puede ser obtenido mediante la instrucción: `${#arreglo[@]}`
- Un rango de elementos puede ser fácilmente especificado de la forma siguiente: `${arreglo[@]:3:6}`

3.1 Ejemplo

Un ejemplo de como iterar utilizando un arreglo es:

```
for persona in ${nombres[@]} do
   echo $persona
done
```

3.2 Actividades

1. Hacer una función que lea del teclado `n` números, los almacene en un arreglo y nos imprima a partir del último número ingresado al primero.

2. Hacer un script que almacene en un arreglo el contenido de un archivo que recibe como parámetro y elimina la palabra que indique el usuario.