INF-113 Programación Web I

Programación de Shell Scripts (II)

Basado en diapositivas de Antonio Durán

Control de flujo

- Veremos las sentencias de control de flujo que ofrece BASH
- Muy parecidas al resto de los lenguajes de programación
  - if, else, for, while, until, case, select

Sentencias condicionales

Formato de la sentencia condicional

```
if  condicion
then
    sentencias
elif condicion
then
    sentencias
else
    sentencias
fi
```

Sentencias condicionales

Otro formato, el más común

```
if condicion ; then
    sentencias
elif condicion ; then
    sentencias
else
    sentencias
fi
```

Sentencias condicionales

- Podemos usar los códigos de terminación (exit status) en sentencias condicionales
- La sentencia if comprueba el código de terminación de un comando en la condición. Si éste es 0, la condición se evalúa como cierta

Sentencias condicionales

La forma normal de escribir la sentencia condicional

```
if comando ; then
    procesamiento normal
else
    procesamos el error
fi
```

Sentencias condicionales

Ejemplo

```
#!/bin/bash

if ls; then
        echo "El comando se ejecutó
correctamente"
else
        echo "Se produjo un error"
fi
```

Práctica 16

Modificar el script anterior (nombre: “condicional.sh”)

- Para que acepte un parámetro con el archivo o directorio que queremos listar
- Usar ese parámetro en la llamada a ls
- Descartar la salida del comando ls, de modo que solo obtengamos el mensaje del script

Operadores lógicos y códigos de terminación

Se pueden combinar varios códigos de terminación de comandos mediante los operadores lógicos:

- and (representado por &&)
- or (representado por ||)
- not (representado por !)

En estas operaciones, el segundo operando sólo se evalúa si el primero no determina el resultado de la condición

Operadores lógicos y códigos de terminación

Ejemplos

- Sólo se ejecuta *ls* si tiene éxito el *cd*

```
$ cd /tmp && ls
$ cd /xxx && ls
```

- Sólo se ejecuta el segundo cp si ha fallado el primero

```
$ cp /tmp/1 ~/ || cp /tmp/2 ~/
```

- El ! niega un código de terminación

```
if ! cp xx /tmp; then
    procesa el error
```

Práctica 17

- Escribir un script “autenticacion.sh” que reciba dos parámetros:
  - Un nombre de usuario
  - Un grupo de usuarios
- Debe mostrar un mensaje, sólo si el usuario pertenece al grupo indicado
  - Usar comando groups y && con grep

Tests condicionales

La sentencia if lo único que sabe es evaluar códigos de terminación

El comando test nos permite comprobar otras muchas condiciones, que le pasamos como argumento, y nos devuelve un código de terminación

Forma alternativa al comando test: []

test cadena1 = cadena2 es equivalente a

[cadena1 = cadena2]

Nota: espacios entre todos los elementos

Tests condicionales

Usando tests condicionales podemos

- Evaluar atributos de un fichero (si existe, de que tipo es, permisos ...)
- Comparar dos ficheros para buscar el más reciente
- Comparar cadenas
- Comparar números

Comparación de cadenas

Operadores de comparación de cadenas

- str1 = str2 :las cadenas son iguales
- str != str2 :las cadenas son distintas
- str1 < str2 :str1 es menor lexicográficamente que str2
- str1 > str2: al revés
- -n str1 :str1 no es nula y tiene longitud mayor que cero
- -z str1 :str1 es nula (tiene longitud cero)

Comparación de cadenas

Ejemplo, versión 1

- Comprobar que pasa si no le pasamos argumentos

```
#!/bin/bash

if [ $1 = "hola" ]; then
        echo "Las cadenas son iguales"
else
        echo "Las cadenas son distintas"
fi
```

Comparación de cadenas

Ejemplo, versión 2

- Comprobar que pasa si no le pasamos argumentos

```
#!/bin/bash

if [ “$1” = "hola" ]; then
        echo "Las cadenas son iguales"
else
        echo "Las cadenas son distintas"
fi
```

Comparación de cadenas

Ejemplo, versión 3

Se comprueban antes los argumentos

```
#!/bin/bash

if [ -z $1 ]; then
        echo "Uso: $0 cadena" #si no hay argumento, salimos
        exit 1
fi

if [ $1 = "hola" ]; then
        echo "Las cadenas son iguales"
else
        echo "Las cadenas son distintas"
fi
```

Práctica 18

- Escribir un script “condicionales2.sh” que compare dos cadenas pasadas como parámetros y diga cual es mayor lexicográficamente, o si por el contrario son iguales
- Debe comprobar que los argumentos se han pasado correctamente
  - Usar $#

Comparación numérica de enteros

El shell también permite comparar variables que almacenan cadenas, interpretando estas cadenas como números. Operadores:

-lt :less than (menor que)

-le: less than or equal (menor o igual que)

-eq: equal (igual)

-ge: greater than or equal (mayor o igual)

-gt: greater then (mayor que)

-ne: not equal (no igual)

Comparación numérica de enteros

Ejemplo

```
#!/bin/bash

if [ $# -ne 2 ]; then
        echo "Uso: $0 número número"
        exit 1
fi

if [ $1 -eq $2 ]; then
        echo "Los números son iguales"
elif [ $1 -gt $2 ]; then
        echo "$1 es mayor que $2"
else
        echo "$2 es mayor que $1"
fi
```

Comparación numérica de enteros

A nivel de condición, dentro de los [], también se pueden usar operadores lógicos, pero en este caso debemos usar los operadores:

-a para and

-o para or

```
if [ $n -ge 0 -a $n -le 10 ]; then
    echo “El número está entre 0 y 10”
fi
```

Comparación numérica de enteros

Se pueden usar paréntesis para aumentar la claridad, pero deben ir precedidos de \

```
if [ \( $n -ge 0 \) -a \( $n -le 10 \) ]; then
    echo “El número está entre 0 y 10”
fi
```

Práctica 19

Escribir un script “condicionales 3” que reciba tres argumentos y compruebe

- Que el arg1 es menor o igual que mil
- Que los dos argumentos siguientes están entre 0 y arg1
- Imprimir mensajes de error descriptivos para cada condición que no se cumpla

Comprobar atributos de ficheros

El tercer tipo de operadores nos permiten comprobar atributos de ficheros.
Operadores:

-a fichero : fichero existe

-b fichero : fichero existe y es un dispositivo de bloque

-c fichero : fichero existe y es un dispositivo de carácter

-d fichero : fichero existe y es un directorio

Comprobar atributos de ficheros

Operadores:

-e : fichero existe (equivalente a -a)

-f : fichero existe y es un fichero regular

-g : fichero existe y tiene activo el bit sgid

-h, -L fichero : fichero existe y es un enlace simbólico

-k fichero: fichero existe y tiene el sticky bit activado

-N fichero : fichero existe y fue modificado desde la última lectura

Comprobar atributos de ficheros

Operadores:

-p fichero : fichero existe y es una tubería

-r fichero : fichero existe y podemos leerlo

-s fichero : fichero existe y no está vacío

-S fichero : fichero existe y es un socket

-u fichero : fichero existe y tiene activo el bit suid

-w fichero : fichero existe y tenemos permiso de escritura

Comprobar atributos de ficheros

Operadores:

-x fichero : fichero existe y tenemos permisos de ejecución, o de búsqueda si es un directorio

fich1 -nt fich2 : la fecha de modificación de fich1 es más moderna (newer than) la de fich1

fich1 -ot fich2 : la fecha de modificación de fich1 es más antigua (older than) la de fich1

fich1 -ef fich2 : son el mismo fichero

Comprobar atributos de ficheros

Ejemplo

```
#!/bin/bash

if [ ! -d /tmp ] ; then
        echo "No existe el directorio /tmp"
        exit 1
fi

if [ ! -f /tmp/xx ] ; then
        touch /tmp/xx
        echo "Se crea el nuevo fichero"
else
        echo "El fichero ya existe"
fi
```

Práctica 20

Escribir un script “newcd.sh” que imite el comportamiento de la orden cd, mostrando más mensajes:

- Si ya está en el directorio pedido
- Si el directorio no es válido
- Si no tenemos permisos para entrar
- Cuando se cambie de directorio, informe de la antigua y la nueva ubicación

Práctica 21

Escribir un script “copia.sh” que realice una copia más segura que el comando cp, comprobando antes de copiar un archivo si éste ya existe, y preguntando, en ese caso, si desea sobreescribirse.

Práctica 22

- Mejorar el script anterior “copia2.sh”, para el caso de que el segundo argumento sea un directorio.
  - En ese caso, se debe comprobar si existe un fichero llamado igual que el argumento1 en ese directorio

El bucle for

El bucle for en Bash es algo diferente del for de otros lenguajes

Se parece más al bucle for each, ya que no se repite un número fijo de veces, sino que se procesan las palabras de una frase una a una

El bucle for

Sintaxis

```
for var in lista
do
      .....
      sentencias que usan $var
      .....
done

for var in 1 2 3
do
      echo $var
done
```

El bucle for

La lista del bucle for puede contener comodines

Éste muestra información detallada sobre los contenidos del directorio actual

```
for ficheros in *
do
    ls -l “$fichero”
done
```

El bucle for

- Para recorrer los argumentos de un script, lo correcto es usar “$@” entrecomillado
  - $* y $@ interpretan mal los elementos con espacios
  - “$*” cosindera un sólo elemento a todos los argumentos

El bucle for

Ejemplo. Probar con argumentos con espacios

```
#!/bin/sh

for arg in $@
do
	echo "Elemento bucle 1: $arg"
done

for arg in "$*"
do
	echo "Elemento bucle 2: $arg"
done

for arg in "$@"
do
	echo "Elemento bucle 3: $arg"
done
```

Prácticas 23

Escribir un script “crearficheros.sh” que cree, en /tmp, los ficheros f1.txt, f2.txt...f9.txt

touch

Escribir un script “versubdirectorios.sh” que recorra todos los elementos del directorio recibido como parámetro y muestre todos los subdirectorios que contiene

Prácticas 24

Escribir un script “directorios.sh” que recorra todos los elementos del directorio recibido como parámetro, indicando para cada uno, si tenemos permisos para leer, escribir y ejecutar el archivo

Operadores y temas relacionados

El comando interno let permite evaluar expresiones con números enteros, para asignar su resultado a una variable

$ let “n=3*4”

$ echo $n

12

Operadores y temas relacionados

Operadores aritméticos

+, -, *, /

** (potencia)

% (módulo)

+=, -=, *=, /=, %=

let “n += 1” #incrementa n

Operadores y temas relacionados

- Operadores lógicos
  - &&, ||
- Operadores de bit
  - <<, >> ...
  - Casi no se usan, no los veremos

Práctica 25

- Escribir un script “parimpar.sh” que reciba un número como parámetro y diga si es par o impar
- Escribir un script “sumar.sh” que sume todos los parámetros recibidos
  - Usar for para recorrerlos

El bucle while

- Comprueba la condición al comienzo del bucle, y continúa ejecutando mientras sea cierta
- La condición del bucle es como las condiciones en las construcciones if

```
while [ condición ]
do
    comandos
done
```

El bucle while

Ejemplo

```
#!/bin/bash

n=0
while [ "$n" -lt 10 ]
do
        echo $n
        let "n += 1"
done
```

El bucle while

Otro ejemplo

```
#!/bin/bash

n=0
while [ "$n" != "fin" ]
do
        echo -n "Introduce variable (fin
para terminar): "
        read n
        echo "n = $n"
done
```

Práctica 26

Escribir un script “factorial.sh” que calcule el factorial del número recibido como parámetro

Escribir un script “red.sh” que ejecute un bucle infinito mostrando las conexiones de red cada cinco segundos

- netstat –inet
- sleep

Práctica 27

Escribir un script “dominio.sh” que funcione como el comando nslookup, usando el comando host

- Un bucle que lee un nombre de teclado hasta que se introduzca 'fin'
- Se usa el comando host para obtener la IP correspondiente

El bucle until

Comprueba la condición al comienzo del bucle, y continúa ejecutando mientras sea falsa, al contrario que el bucle while

```
until [ condición-es-cierta ]
do
    comandos
done
```

El bucle until

Un ejemplo anterior, usando until en lugar de while

```
#!/bin/bash

n=0
until [ "$n" -ge 10 ]
do
        echo $n
        let "n += 1"
done
```

Práctica 28

Escribir el script “factorial2.sh” del factorial usando until en lugar de while

Escribir el mismo script “dominio2.sh” que imita el comando nslookup, usando until en lugar de while

Control de bucles

Comandos que afectan al comportamiento de los bucles:

- break: sale del bucle
- continue: pasa a la siguiente iteración

Control de bucles

Mejorando un ejemplo anterior

```
#!/bin/bash

n=0
while [ "$n" != "fin" ]
do
        echo -n "Introduce variable (fin para
terminar): "
        read n
         if [ “$n” = “fin”]; then
            break
         fi
        echo "n = $n"
done
```

Práctica 29

Escribir un script “salto.sh” que muestre los número de 1 a 10, pero no el 3

(opcional) Modificar el script que imita el comando nslookup, para que no intente resolver el host “fin” con el nombre “dominio3.sh”.

La construcción case

- La construcción case es parecida al switch en C/C++
- Permite saltar a uno de diferentes bloques de código dependiendo de las condiciones
- Apropiado para manejar opciones de un menú

La construcción case

Sintaxis

```
case cadena in
    patron1)
        comandos
        ;;
    patron2)
        comandos
        ;;
    ....
esac
```

La construcción case

Ejemplo

```
#!/bin/sh

case $1 in
"")
        echo "Uso: $0 argumento"
        ;;
-*)
        echo "El argumento es una opcion"
        ;;
*)
        echo "Es un argumento normal"
        ;;
esac
```

La construcción case

Ejemplo

```
#!/bin/sh

case $1 in
"")
	echo "Uso: $0 caracter"
	;;
[0-9])
	echo "El argumento es un numero"
	;;
[a-z])
	echo "El argumento es una letra minuscula"
	;;
[A-Z])
	echo "El argumento es una letra mayuscula"
	;;
*)
	echo "El argumento es un simbolo de puntuacion, un espacio u otro"
	;;
esac
```

Práctica 30

- Realizar un script “eleccion.sh” que reciba dos parámetros:
  - Uno será el nombre de un fichero
  - El otro representa la acción a realizar sobre el fichero
    - mostrar: se muestra el fichero (cat)
    - estadisticas: se muestran sus estadísticas (stat)
    - tipo: muestra el tipo de fichero (file)
    - Si se recibe otra acción, debe mostrarse un error

La construcción select

La construcción select es otra forma de generar menús

Pide al usuario que introduzca uno de los elementos presentados en la lista

```
select variable [in list]
do
    comandos
    break
done
```

La construcción select

Ejemplo

```
#!/bin/bash

echo "Elige tu fruta favorita"
select fruta in "manzana" "pera" "naranja" "kiwi"
"mango"
do
   echo "Tu fruta favortia es $fruta."
   break
done
```