
Trasparencias del Curso de I

*Grupo de usuarios de
linux*

*Univ. Carlos III de Madrid
rcortega@gul.uc3m.es*

24 de marzo de 2004

Scripts en Bash

- Algo más que una secuencia de comandos
- Posibilidad de:
 - Bucles
 - Sentencias condicionales
 - Funciones
 - ...

Mi primer script en Bash

- No es mas que un fichero de texto del formato:

```
#!/bin/bash
```

```
echo ‘‘Hola Mundo!!’’
```

- `#!/bin/bash` indica que el fichero ha de ser pasado a bash

- Equivale a escribir

```
echo ‘‘Hola Mundo!!’’
```

en un terminal

Ejecutando el script

- Lo primero tras escribir el script será darle permisos
`chmod 755 HolaMundo.sh`
- Y después ya podremos ejecutar:
`./HolaMundo.sh`

Asignación de variables

`<nombre>=<valor>`

- **No** debe haber espacios a los lados del “=”
- **No** debe haber “\$” al inicio del nombre

`var1=$var2`

- A la derecha de la igualdad **sí** se usa “\$”

Principales variables predefinidas

- **\$?** resultado del último comando:
 - 0 éxito
 - 1 error
- **\$\$** PID del proceso actual
- **\$PPID** PID del proceso padre del actual
- **-** directorio de trabajo anterior (haciendo “cd -” vo directorio en que hemos estado)
- **\$PWD** directorio de trabajo actual
- **\$PATH** rutas en que buscaremos los ejecutables en
- ... y un largo etc

Redirecciones

Son de la forma

"<comando> [> | &> | ...] <fichero>"

- > redirige la salida estandar (sobreescribe el fichero)
- >> redirige la salida estandar (anexa al fichero)
- 2> redirige la salida de errores
- &> redirige la salida estandar y la salida de errores
- >&2 redirige la salida estandar a la salida de errores
- 2>&1 redirige la salida de errores a la salida estandar
- << redirige en un documento "*here*"
- - redirige a la salida estandar o de la entrada estandar (según el parámetro esperado)

Algunos operadores de Test (I)

- Para Ficheros
 - **-d** es un directorio
 - **-f** es un fichero (ni directorio ni dispositivo)
 - **-s** no es vacío
 - **-e** existe
 - **-r** tiene permisos de lectura
 - **-w** tiene permisos de escritura
 - **-x** tiene permisos de ejecución

Algunos operadores de Test (I)

- Comparación de enteros
 - **-eq** igual
 - **-ne** no igual
 - **-gt** o $>$ mayor que
 - **-ge** o $\geq$ mayor que o igual a
 - **-lt** o $<$ menor que
 - **-le** o $\leq$ menor o igual que

Algunos operadores de Test (y I)

- Comparación de cadenas
 - `=` o `==` igual que
 - `!=` distinto a
 - `<` menor que (orden falfabético)
 - `>` mayor que (orden alfabetico)
 - `-z` la cadena es vacía
 - `-n` la cadena no es vacía

Operadores aritméticos

Se utilizan para realizar operaciones numéricas (con enteros)

- + suma
- - resta
- \ división
- * multiplicación
- ** exponenciación
- % módulo (resto de la división entera)

Operadores lógicos

Se utilizan para realizar operaciones lógicas (con booleanos)

- `||` “or” lógico
- `&&` “and” lógico

Operadores lógicos a nivel de bit

Se usan para las operaciones lógicas bit a bit

- << desplazamiento de un bit a la izquierda
- >> desplazamiento de un bit a la derecha
- & “and” lógico (bit a bit)
- | “or” lógico (bit a bit)
- ! “not” lógico (bit a bit)
- ^ “xor” lógico (bit a bit)

Sentencias condicionales (I)

```
if [ <condicion> ]  
then  
    ...  
else  
    ...  
fi
```

Sentencias condicionales (y II)

```
if [ <condicion1> ]
then
    ...
elif [ <condicion2> ]
    ...
else
    ...
fi
```

- Atención a los espacios alrededor de los corchetes

```
if [ <condicion> ]
  ^ ^           ^
```

Case (I)

Es como los “switch” de alto nivel

```
case $variable in
<condicion1> )
    ...
;;
<condicion2> )
    ...
;;
* )
    ...
;;
esac
```

Case (y II)

Ej:

```
case $i in
1 )
    echo ‘‘Es uno’’
;;
0 )
    echo ‘‘Es cero’’
;;
* )
    echo ‘‘No es uno ni cero’’
;;
esac
```

Bucle for

Se utiliza para repetir una tarea un numero **determinado**

```
for <arg> in [list]
do
    ...
done
```

A cada iteración, la variable “arg” tendrá el valor del elemento siguiente de la lista.

Ej:

```
for num in 1 2 3 4 5 6 7 8 9 10
do
    echo ‘‘$num’’
done
```

Nota: también se pueden poner cadenas

Bucle for al estilo C

```
END=10
for (( i=1; i <= END ; i++))
do
    echo '$num'
done
```

Bucle while

```
while [ <condicion> ]  
do  
    ...  
done
```

o a la inversa

```
until [ <condicion> ]  
do  
    ...  
done
```

Interacción con el programa

- Salida por pantalla
 - `echo`
 - `printf <format> <parametros>`
- Entrada de teclado
 - `read <variable>`

Parámetros al script

- `$x` argumento número “x” pasado al script
- `$#` número de argumentos pasados al script
- `$@` array con todos los argumentos
- `$*` todos los argumentos concatenados en un único string

Nota: el comando `shift` realiza un desplazamiento a la izquierda, pierde el primer elemento, el segundo pasa a ser el primero, etc.

Funciones (I)

Se pueden escribir como

```
function nombre {  
    ...  
}
```

o

```
nombre (){  
    ...  
}
```

Funciones (II)

- Los parámetros se pasan como al script (\$1, \$2 ...)
- El “return” sólo puede ser un entero, que sera el estado de la variable \$?)
- Las variables por defecto son globales (ojo a iteraciones llamadas desde funciones, recursividad...) Para hacerlas locales al bucle o función se usa el modificador “local”

BIGLIOGRAFIA

- `man bash ;)`
- “*The Advanced Bash Scripting Guide*”
<http://www.tldp.org/LDP/abs/>
(está en HTML, PDF, PS, PDB...)