Ejemplos de guiones shell

- **Ruta absoluta vs. ruta relativa**

Es muy importante saber en qué directorio estamos ubicados para poder referenciar correctamente un archivo/directorio determinado.

El comando **pwd** nos muestra la ruta completa del directorio donde estamos trabajando.
La variable de entorno **HOME** (se accede a ella con $HOME) nos proporciona el directorio de trabajo asociado al usuario.

Supongamos que nos encontramos en el directorio */home/usuario/Practica1*. Después de ejecutar el comando *ls -l* obtenemos la siguiente salida. Aquí hay dos subdirectorios denominados *Practica2* y *public_html*.

```
-rwxr-xr-x 1 jcarlos users      52 2007-03-09 20:32 ejemplo1
drwxr-xr-x 4 jcarlos users     728 2007-03-26 19:08 Practica2
-rwxr-xr-x 1 jcarlos users      56 2007-03-09 20:32 progreso
-rwxr-xr-x 1 jcarlos users     106 2007-03-01 22:52 prueba1
drwxr-xr-x 2 jcarlos users      80 2007-02-24 16:30 public_html
-rw-r--r-- 1 jcarlos users    1362 2007-03-05 20:54 rr
-rw-r--r-- 1 jcarlos users       7 2007-03-10 20:16 rr2
-rw-r--r-- 2 jcarlos users   21284 2007-03-10 20:15 rr2link
lrwxrwxrwx 1 jcarlos users       3 2007-03-10 20:14 rr2slink -> rr2
-rw-r--r-- 2 jcarlos users   21284 2007-03-10 20:15 rrlink
-rw-r--r-- 1 jcarlos users       0 2007-03-05 22:53 sal
```

Pues bien, cualquier referencia a este subdirectorio utilizando la ruta relativa será simplemente a través de su nombre ya que se encuentra justo debajo de donde estamos.

```
ls -l  Practica2
cd Practica2
```

Si nos encontráramos en el subdirectorio *public_html*, entonces

```
ls -l  ../Practica2
cd ../Practica2
```

Todo éstos son ejemplos de ruta relativa.

Su ruta absoluta será */home/usuario/Practica1/Practica2*. Ahora:

```
ls -l  /home/usuario/Practica1/Practica2
cd /home/usuario/Practica1/Practica2
```

Con estos comandos siempre obtendremos los mismos resultados, independientemente del directorio desde donde los lancemos.

Si esto lo aplicamos a un shell script tendremos lo siguiente:

if [-d Practica2] será cierto si *Practica2* es un subdirectorio del directorio donde ejecutemos el shell script.

if [-d /home/usuario/Practica1/Practica2] será cierto si */home/usuario/Practica1/Practica2* es un directorio. Aquí no importa desde dónde lanzamos el script ya que estamos utilizando la runta absoluta.

Si estamos en el directorio */home/usuario/Practica1* y lanzamos el script *ejemplo1* cuyo código es el siguiente:

```
#!/bin/bash

if [ -d $1 ]
then
        echo $1 es un directorio
else
        echo $1 no es un directorio
fi
```

Aquí $1 representa el valor del primer argumento introducido al ejecutar lo siguiente:

```
. ejemplo1  Practica2
```

Dentro del script anterior $1 será sustituido por el valor *Practica2*. Como *Practica2* sí que es un subdirectorio del directorio donde hemos lanzado el script, la salida será cierta.

Pero una ejecución como *. ejemplo1 /home/usuario/Practica1/Practica2* también será cierta, ya que ahora $1 será sustituido por el valor */home/usuario/Practica1/Practica2* que, evidentemente, es un directorio.

Supongamos ahora que el directorio de trabajo por defecto del usuario (directorio al que se conecta el usuario) es */home/usuario*. $HOME tendrá pues, el valor */home/usuario*. Entonces:

if [-d $HOME/Practica1] será cierta.

if [-d $HOME/Practica2] será falsa pues no existe el directorio */home/usuario/Practica2*

if [-d $HOME/Practica1/Practica2] será cierta.

- **Bucle for**

En shell script esta instrucción de control es bastante elaborada. Su sintaxis general es:

```
for variable in lista
do
        órdenes
        ...
done
```

Aquí *lista* puede ser algo tan variado como un conjunto de cadenas de caracteres, el conjunto de archivos de directorio o la salida de un comando.

Ejemplos:

```
for i in 1 Ana Juan 58
```

El bucle se repetirá 4 veces. La variable *i* (acceso con $i) tomará en cada iteración, respectivamente, los valores 1, Ana, Juan y 58.

```
for i in *
```

Aquí * representa todos los archivos del directorio donde ejecutemos el script. Probar si no, a ejecutar desde el prompt el comando ls *.

Por tanto la variable *i* (acceso con $i) irá tomando sucesivamente como valor el nombre de cada archivo del directorio.

```
for i in Practica2/*
```

Hará lo mismo que el ejemplo anterior, pero ahora con los archivos del subdirectorio *Practica2* (recuérdese la ruta relativa). Este ejemplo solamente funcionará si lanzamos el script desde */home/usuario/Practica1*.

```
for i in $HOME
```

Lo mismo pero con el directorio */home/usuario*

Compliquemos un poco el asunto. En el bucle for, *lista* también puede ser un comando. Pero este comando debe estar encerrado entre acentos graves.

```
for i in `ls -R`
```

Ahora se recorrerán recursivamente todos los archivos que cuelgan del directorio desde donde ejecutamos el script.

Alternativamente podemos obtener el mismo resultado con las siguientes líneas:

```
A=`ls -R`
for i in $A
```

Sin embargo, debe tenerse en cuenta qué valores toma la variable *i* en todos estos casos. La orden `ls` obtiene los nombres de un cierto grupo de archivos. Si lo que pretendemos en procesar dichos nombres, esta orden nos servirá. En cambio, una orden como `ls –l` no nos será de utilidad, ya que se procesarán también los grupos de archivos, usuarios, fecha, hora, etc. Es decir, la variable *i* del bucle for, irá tomando sucesivamente valores como los grupos de archivos, usuarios, etc.

rwxr-xr-x, 1, jcarlos, users, 52, 2007-03-09, 20:32, ejemplo1, drwxr-xr-x, 4, etc

- **Asignación de valores a variables**

Hemos visto que en shell script no hay tipos de variables. Éstas son asignadas en el momento que se definen:

```
A=1
B=Ana
```

Pero también podemos asignarlas con el valor de la salida de un comando:

```
A=`comando`
A=$(comando)

A=`ls | wc -w`  En este caso A tomará como valor el número de archivos del directorio actual

A=$(ls |  wc -w)  Lo mismo que en el caso anterior
```

Como la asignación de variables es alfanumérica, podemos componer cualquier variable a nuestra conveniencia sin más que concatenar variables:

```
A=Ana
B=Practica1
C=$A/$B     toma el valor  Ana/Practica1
```

Esto puede ser muy útil para crear archivos o directorios que puedan ser identificados fácilmente por algún criterio determinado. Por ejemplo, nos podría interesar ir guardando sucesivas versiones de un cierto archivo del tal manera que se identifique el número de la versión.

Supongamos que nuestro archivo es un programa denominado *movcontables.c* y que como máximo hay 9 versiones. La n-enésima versión se denominará *movcontables.c.n*

Procederemos como sigue:

- Introduciremos como argumento el valor *movcontables.c*.
- Lo primero que hará el script es comprobar si existe el archivo *movcontables.c*. Si no existe, el script presentará un mensaje acorde con el hecho y terminará.
- En el caso de que exista, contamos el número de versiones existentes del archivo. Esto lo hacemos con el comando ls $1* | wc -w. El valor resultante se lo asignamos a la variable A.
- A continuación incrementamos el valor de A. Como hemos dicho previamente, las variables son alfanuméricas. Para poder operar con ellas de forma numérica hacemos uso del operador ***expr*** y encerrado entre acentos graves. El resultado es asignado a la variable *Num* mediante la orden Num=`expr $A + 1`
- Posteriormente componemos la variable *Version* añadiendo al nombre del archivo original una nueva extensión con el valor de la nueva versión: Version=$1.$Num
- Copiamos el archivo *movcontables.c*
- Finalmente creamos el archivo *movcontables.c* vacío y listo para ser codificado de nuevo.

```
#!/bin/bash

if [ ! -f "$1" ]
then
        echo $1 no existe
else
        A=$(ls $1*  |  wc -w)

        if [ $A  -ge  9 ]
        then
                echo Se ha superado el número máximo de versiones
        else
                Num=`expr $A + 1`
                Version=$1.$Num
                cp $1  $Version
        fi
fi
```

1. Guión shell que copia todos los archivos de trabajo con extensión *.dat* y *.c* del directorio actual al directorio pasado como argumento. Si este directorio no existe, el guión lo debe crear.

```
#!/bin/bash

test ! –e “$1”  &&  mkdir “$1”
if [ -d  “$1” ]
then
    cp *.dat *.c  “$1”  2> ficherr
fi
```

Aquí en la segunda línea, se ejecutará el comando mkdir “$1” si no existe “$1”. Si existe, no se creará el directorio.
Sin embargo, “$1” puede existir y no ser un directorio (por ejemplo, otro tipo de archivo). Por ello, antes de realizar la copia de archivos, es necesario verificar que, efectivamente, es un directorio.
Por último, pudiera ser que no existieran archivos del tipo *.c o *.dat , lo que ocasionaría un mensaje de error por pantalla indicando tal hecho. Esto se puede evitar redireccionando hacia la salida de error estándar (**2>**). También podría hacerse indicando esto mismo a la hora de ejecutar el script con la orden: . script 2> ficherr

2. Crear una función que realice el mismo cometido que el shell script del ejercicio anterior.

Codificar el siguiente código en un archivo de texto denominado, por ejemplo *misfunciones*.

```
function copia_c_dat()
{
    test ! –e “$1”  &&  mkdir “$1”
    if [ -d  “$1” ]
    then
        cp *.dat *.c  “$1”
    fi
}
```

A continuación, ejecutar el archivo *misfunciones* para activar la función. Ahora, desde el prompt del sistema se puede ejecutar la orden:

copia_c_dat nom_dir.

3. Ejecutar el siguiente comando: **date +%Y%m%d** y observar el resultado. Utilizar la función del ejercicio anterior para copiar todos los archivos de trabajo con extensión *.dat* y *.c* del directorio actual al directorio *backupsC/fecha*, siendo fecha el valor del comando anteriormente ejecutado.

```
#!/bin/bash

FECH=`date +%Y%m%d`
DIRBACKUP=backupsC/$FECH

copia_c_dat $DIRBACKUP
```

```
# también valdría   copia_c_dat  $DIRBACKUP 2> ficherr

if [ $? = 0 ]   # Comprobamos si todo ha ido bien o no
then
        echo "Backup realizado correctamente"
else
        echo "Se produjo un problema durante el backup"
fi
```

Aquí el comando *date +%Y%m%d* muestra la fecha del sistema en formato AAAMMDD, lo cual se puede comprobar muy fácilmente. La salida de dicho comando puede ser asignada a una variable utilizando los acentos graves: FECH=`date +%Y%m%d `
Podemos asumir que el directorio que se nos pide en el enunciado es una ruta relativa por lo que *backupsC/fecha* será un subdirectorio del actual. Es fácil crear una variable que contenga dicho valor: DIRBACKUP=backupsC/$FECH.
A continuación invocamos a la función anteriormente creada sin más que pasarle como argumento el valor *$DIRBACKUP*.
Finalmente, comprobamos si todo ha ido bien o no consultando a la variable especial **$?** que nos proporciona el valor del último comando ejecutado.

4. Implementar un shell script que verifique cada 30 segundos si existe en el directorio actual un fichero *prueba.txt*. Para probar este guión es necesario ejecutarlo en segundo plano.

```
#!/bin/bash

until test –e prueba.txt
do
        sleep 30
done
echo Ya apareció el archivo prueba.txt
```

5. Crear un shell script que liste todos los directorios y subdirectorios recursivamente de uno dado. El directorio será introducido como argumento y el guión lo primero que hará será verificar si es precisamente un directorio.

```
#!/bin/bash

if [ ! $#  -eq 1 ]   # Comprobar número de parámetros introducidos
then
        echo "¡¡ Error !!  Uso: "$0" nom_dir"
        break
fi
if [ -d "$1" ]
then
        ls –lR "$1" | grep '^d'
else
        echo "No existe el directorio $1"
fi
```

La clave de este ejercicio estriba en utilizar un comando apropiado para recorrer recursivamente un directorio dado y que únicamente muestre los archivos que son directorios. Esto se puede hacer con la orden *ls* y la opción *–R* .
Sabiendo que el primer carácter mostrado al ejecutar *ls –l* es el tipo de archivo, solamente nos falta, por tanto, filtrar cada línea del listado por dicho carácter. Esto se hace fácilmente con **grep** y el patrón **'^d'** que busca el carácter *d* (de directorio) al principio de una línea.
El resto del script es sencillo. Lo primero que se hace es comprobar el número de argumentos introducidos. Debe ser 1, el nombre del directorio. Ello se hace consultando la variable especial **$#.** Si no es 1, se muestra un mensaje de error y se indica la forma de ejecutar el script ("$0" nom_dir)

6. Ejecutar varias veces seguidas en modo background los comandos *xeyes* y *xclock*. Implementar un script que, al ejecutarlo, elimine todos y cada uno de los procesos generados con dichos comandos. El script tomará como argumentos los nombres de dichos comandos: xeyes y xclock.

```
#!/bin/bash

if [ ! $#  -eq 2 ]   # Deben introducirse 2 argumentos
then
        echo "¡¡ Error !!  Uso: $0 nom_prog1 nom_prog2"
        break
fi

for PROG in $*
do
        kill $(ps –a | grep "$PROG" | tr –s ' ' | cut –f2 –d ' ')
done
```

Sabemos que la orden ***kill*** *pid_proceso* elimina el proceso cuyo PID es *pid_proceso*. Por otro lado, la orden *ps –a* nos muestra, entre otros datos, los nombres de los procesos en ejecución y sus respectivos PIDs. Precisamente este último dato es el segundo campo de la salida *ps –a*. Luego, vamos a extraer dicho valor:

```
ps –a | tr –s ' ' | cut –f2 –d ' '
```

Como solamente necesitamos aquellos PID´s de los procesos indicados como argumentos, filtraremos las líneas con la orden **grep:**

```
ps –a | grep "$PROG" | tr –s ' ' | cut –f2 –d ' '
```

donde $PROG tomará los valores de todos y cada uno de los argumentos pasados al script, a los que se accede a través de la variable especial **$***.

La salida del comando anterior puede ser asignada a una variable a través del formato **$(comando)**. Esta variable es precisamente lo que necesita la orden kill:

```
kill $(ps –a | grep "$PROG" | tr –s ' ' | cut –f2 –d ' ')
```

Alternativamente podría haberse hecho uso de la orden **xargs** como sigue. Dentro del bucle *for* poner:

ps –a | grep $PROG | tr –s ' ' | cut –f2 –d ' ' | xargs kill

Una tercera opción sería la siguiente:

```
for PROG in `ps –a | grep "$1\$2" | tr –s ' ' | cut –f2 –d ' ' `
do
        kill $PROG
done
```

Aquí se hace uso del comando **grep** con el patrón "valor1\valor2" que indica que filtrará todas las líneas que contengan los valores *valor1* o *valor2* o ambos.