## Shell Scripting - Introducción - Continuación

Ahora que sabemos cómo determinar si cierta condición es verdadera o no, podemos combinar eso con la sentencia “if” para tomar decisiones en nuestros scripts:

- La sentencia “if” comienza con la palabra “if” y va seguida de un test.
- La siguiente línea contiene la palabra “then”.
- A continuación, hay un comando o una serie de comandos que se ejecutarán si la condición evaluada devuelve true.
- Por último, la sentencia “if” termina con “fi”.

### Making Decisions - The if statement

```
if [ condition-is-true ]
then
   command 1
   command 2
   command N
fi
```

He aquí un ejemplo:

```
#!/bin/bash
MY_SHELL="bash"

if [ "$MY_SHELL" = "bash" ]
then
   echo "You seem to like the bash shell."
fi
```

Output:
You seem to like the bash shell.

Hay que tener en cuenta que es una buena práctica encerrar la variable entre comillas para evitar efectos secundarios inesperados al realizar tests condicionales.

También podemos realizar una acción si la condición no es verdadera utilziando una sentencia “if/else”. Basta con insertar la palabra clave “else” y a continuación los comandos que queremos ejecutar si la condición no es verdadera:

## if/else

```
if [ condition-is-true ]
then
  command N
else
  command N
fi
```

Así que vamos a actualizar nuestro script para realizar una acción si la declaración no es verdadera. Debido a que “$MY_SHELL” = “bash” evalúa a falso, las declaraciones que siguen a “else” serán ejecutadas:

```
#!/bin/bash
MY_SHELL="csh"

if [ "$MY_SHELL" = "bash" ]
then
   echo "You seem to like the bash shell."
else
   echo "You don't seem to like the bash
shell."
fi
```

Si queremos realizar una acción sobre una lista de elementos, utilizamos un blucle “for”:

## For loop

```
for VARIABLE_NAME in ITEM_1 ITEM_N
do
  command 1
  command 2
  command N
done
```

- La primera línea de un bucle “for” comienza con la palabra “for”, seguida del nombre de una variable y, a continuación, una lista de elementos.
- La siguiente línea contiene la palabra “do”.
- En las líneas siguientes colocamos las sentencias que deben ejecutarse.
- Finalmente, terminamos el bucle con la palabra “done”.

Básicamente, lo que ocurre es que el primer elemento de la lista se asigna a la variable y se ejecuta el conjunto de sentencias. A continuación, el siguiente elemento de la lista se asigna a la variable y se ejecutan los comandos. Esto sucede para cada elemento de la lista.

He aquí un ejemplo que muestra cómo funciona un bucle “for”:

```
#!/bin/bash
for COLOR in red green blue
do
  echo "COLOR: $COLOR"
done
```

Output:

```
COLOR: red
COLOR: green
COLOR: blue
```

También es habitual que la lista de elementos se almacene en una variable, como en el siguiente ejemplo:

```
#!/bin/bash
COLORS="red green blue"

for COLOR in $COLORS
do
  echo "COLOR: $COLOR"
done
```

El siguiente script renombra todos los archivos que terminan en “jpg”, insertando la fecha actual antes del nombre original del archivo:

```
#!/bin/bash
PICTURES=$(ls *jpg)
DATE=$(date +%F)

for PICTURE in $PICTURES
do
  echo "Renaming ${PICTURE} to ${DATE}
-${PICTURE}"
  mv ${PICTURE} ${DATE}-${PICTURE}
done
```

Esto es lo que ocurre cuando ejecutamos este script:

```
$ ls
bear.jpg  man.jpg  pig.jpg  rename-pics.sh
$ ./rename-pics.sh
Renaming bear.jpg to 2015-03-06-bear.jpg
Renaming man.jpg to 2015-03-06-man.jpg
Renaming pig.jpg to 2015-03-06-pig.jpg
$ ls
2015-03-06-bear.jpg  2015-03-06-man.jpg
2015-03-06-pig.jpg  rename-pics.sh
$
```

Vemos que el archivo “bear.jpg” se convierte en “2015-03-06-bear.jpg”, y esto continúa para cada archivo que termine en “jpg”.

Los “Parámetros Posicionales” son variables que contienen el contenido de la línea de comandos.

Las variables son de $0 a $0. El nombre del script se almacena en la variable $0. El primer parámetro se almacena en la variable $1, el segundo en la variable $2 y así sucesivamente.

Veamos el siguiente ejemplo:

**Positional Parameters**

```
$ script.sh parameter1 parameter2 parameter3
```

$0 : "script.sh"
$1 : "parameter1"
$2 : "parameter2"
$3 : "parameter3"

- El contenido de $0 es "script.sh".
- $1 contiene "parameter1".
- $2 contiene "parameter2".
- $3 contiene "parameter3".

El siguiente script llamado “archive_user.sh” acepta un parámetro que resulta ser un nombre de usuario:

```
#!/bin/bash

echo "Executing script: $0"
echo "Archiving user: $1"

# Lock the account
passwd -l $1

# Create an archive of the home directory.
tar cf /archives/${1}.tar.gz /home/${1}
```

- Todo lo que sigue después del símbolo # es un comentario. La única excepción a esto es el shebang #! de la primera línea.

- En cualquier otra parte del script donde se encuentre un signo #, marcará el comienzo de un comentario.
- Los comentarios son ignorados por el intérprete ya que son para el beneficio de de la persona que lee el código.
- Todo lo que sigue al signo # es ignorado.
- Si un signo # se coloca al inicio de una línea, se ignora toda la línea.
- Si se encuentra un signo # en medio de una línea, sólo se ignoran las instrucciones a la derecha del signo #.

Este es el resultado de la salida cuando ejecutamos el script:

```
$ ./archive_user.sh elvis
Executing script: ./archive_user.sh
Archiving user: elvis
passwd: password expiry information changed.
tar: Removing leading `/' from member names
$
```

En lugar de referirnos a la variable $1 en todo el script, asignamos su valor a un nombre de variable más significativo, en este caso, vamos a asignarlo a una variable llamada user:

```
#!/bin/bash
USER=$1 # The first parameter is the user.

echo "Executing script: $0"
echo "Archiving user: $USER"

# Lock the account
passwd -l $USER

# Create an archive of the home directory.
tar cf /archives/${USER}.tar.gz /home/${USER}
```

La salida de este script sigue siendo la misma.

También podemos acceder a todos los parámetros posicionales comenzando por $1 hasta el último en la línea de comandos, utilizando la variable especial $@.

He aquí como actualizar el script para que acepte uno o más parámetros:

```
#!/bin/bash

echo "Executing script: $0"
for USER in $@
do
  echo "Archiving user: $USER"
  # Lock the account
  passwd -l $USER
  # Create an archive of the home directory.
  tar cf /archives/${USER}.tar.gz /home/${USER}
done
```

Ahora podemos pasar múltiples usuarios al script. El bucle “for” se ejecutará para cada usuario que hayamos suministrado en la línea de comandos.

Este es el aspecto que tendrá el script si le pasamos 2 usuarios:

```
$ ./archive_user.sh chet joe
Executing script: ./archive_user.sh
Archiving user: chet
passwd: password expiry information changed.
tar: Removing leading `/' from member names
Archiving user: joe
passwd: password expiry information changed.
tar: Removing leading `/' from member names
$
```

Si queremos aceptar la entrada estándar usamos el comando “read”. Recordemos que la entrada estándar normalmente proviene de una persona que escribe en el teclado, pero también puede provenir de otras fuentes, como la salida de otro comando.

El formato del comando “read” es el siguiente:

## Accepting User Input (STDIN)

The read command accepts STDIN.

Syntax:

```
read -p "PROMPT" VARIABLE
```

Esta versión del script archive_user.sh solicita el usuario:

```
#!/bin/bash

read -p "Enter a user name: " USER
echo "Archiving user: $USER"

# Lock the account
passwd -l $USER

# Create an archive of the home directory.
tar cf /archives/${USER}.tar.gz /home/${USER}
```

En este ejemplo, ejecutamos el script y escribimos el nombre de usuario “mitch”:

```
$ ./archive_user.sh
Enter a user name: mitch
Archiving user: mitch
passwd: password expiry information changed.
tar: Removing leading `/' from member names
$
```