11.Loops

En este capítulo se verá como se utilizan los loops (bucles) en un script. Los loops permiten ejecutar una serie de comandos repetidas veces. Existen 3 tipos principales de loops:

- El loop while
- El loop for
- El loop select

El loop while permite ejecutar un conjunto de comandos en forma repetida hasta que cierta condición ocurra. Generalmente se usa cuando se necesita manipular el valor de una variable en forma repetida.

El loop for permite ejecutar un conjunto de comandos en forma repetida por cada ítem en una lista. Uno de los usos comunes es ejecutar el mismo conjunto de comandos para un número grande de archivos.

El loop select provee una forma fácil de crear un menú numerado del cual los usuarios pueden seleccionar una acción. Es útil cuando se necesita preguntar al usuario que elija uno o más ítems de una lista de opciones.

El Loop while

La sintaxis del loop while es

```
while command
do
        list
done
```

command es un comando simple a ejecutar, mientras que list es un conjunto de uno o más comandos a ejecutar. Si bien command puede ser cualquier comando Linux válido,generalmente es utilizada una expresión test.

La ejecución de un loop while procede de acuerdo a los siguientes pasos:

1. Ejecutar command.
2.1. Si el código de salida de command es distinto de 0, termina el loop.
2.2. Si el código de salida de command es 0, ejecuta list.
3. Cuando list termina de ejecutar, vuelve al Paso 1.

Si command y list son cortos, el loop se suele escribir en una sola línea:

```
while command ; do list ; done
```

El siguiente es un ejemplo de while que imprime los número de 0 a 9:

```
x=0
```

```
while [ $x -lt 10 ]
do
        echo $x
        x=$((x + 1))
done
```

La salida debería ser:

```
0
1
2
3
4
5
6
7
8
9
```

Cada vez que se ejecuta el loopp, se verifica la variable x para ver si es menor que 10. Si se cumple la condición se imprime el valor actual de x y se incrementa en 1.

Whiles Anidados

Es posible usar un while en el cuerpo de otro de la siguiente manera:

```
while command1 ; # this is loop1, the outer loop
do
        list1
        while command2 ; # this is loop2, the inner loop
        do
                list2
        done
l       list3
done
```

No existen restricciones respecto a la profundidad de los anidamientos, sin embargo es conveniente evitar loops anidados de más de 3 o 4 niveles. Esto hace difícil la comprensión del script y encontrar errores.

El siguiente ejemplo agrega un loop anidado al ejemplo anterior:

```
x=0
while [ "$x" -lt 10 ] ; # loop1
do
        y="$x"
        while [ "$y" -ge 0 ] ; # loop2
        do
                echo -n "$y"
                y=$(($y - 1))
        done
```

```
        echo
        x=$(($x + 1))
done
```

Que produce la siguiente salida

```
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
```

Ejemplo: Validación de la Entrada del Usuario

Suponemos que necesitamos un script que pida que se ingrese el nombre de un directorio. El script continúa hasta que se ingresa un nombre de directorio válido. Esto se resume en la siguiente secuencia de pasos:
1. Preguntar al usuario el nombre del directorio.
2. Leer la respuesta del usuario.
3. Verificar que el usuario ingresó un directorio válido.
4. Validar la respuesta.
5. Si la respuesta es inválida setear la variable a null. Esto permite que el while continúe.
6. Si la respuesta es válida, el valor de la variable mantiene la respuesta del usuario y al no ser null termina.

```
RESPUESTA=
while [ -z "$RESPUESTA" ] ;
do
        echo -n "Ingrese el path donde se encuentran sus archivos: "
        read  RESPUESTA
        if [ ! -d "$RESPUESTA" ] ; then
                echo "ERROR: Ingresar un directorio válido."
                RESPUESTA=
        fi
done
```

Cuando el loop while se ejecuta por primera vez, se pide al usuario que ingrese un directorio:

```
Ingrese el path donde se encuentran sus archivos:
```

El usuario puede ingrese el nombre de un directorio. Cuando el usuario termina presiona Enter, el comando read pone la entrada del usuario en la variable RESPUESTA. Si la entrada no es un directorio, se imprime un error y se repite el

loop. El mensaje de error se emite para que el usuario sepa cual fue el error.

El Loop until

El loop while se adecúa perfectamente cuando se debe ejecutar un conjunto de comandos se debe ejecutar mientras cierta condición es verdadera. Sin embargo algunas veces se necesita ejecutar hasta que una sea verdadera.

Una variante del loop while, es el loop until que provee la funcionalidad mencionada anteriormente. La sintaxis es:

```
until command
do
        list
done
```

Generalmente command es un comando test como se vio previamente.

La ejecución de until es idéntica a la de while y tiene los siguientes pasos:

1. Ejecuta command.
2. Si el estado de terminación de command es != 0, termina el until.
3. Si el estado de terminación de command es 0, ejecuta list.
4. Cuando list termina, vuelve al paso 1.

En la mayoría de los casos until es igual al while con list1 negado usando el operador !. Por ejemplo, el siguiente while

```
x=1
while [ ! $x -ge 10 ]
do
        echo $x
        x= `echo "$x + 1" | bc`
done
```

es equivalente al siguiente until:

```
x=1;
until [ $x -ge 10 ]
do
        echo $x
        x=`echo "$x + 1" | bc`
done
```

Los Loops for y select

Al contrario que el while, que termina cuando cierta condición es falsa, tanto el for como el select trabajan con una lista de ítems. El for repite un conjunto de comandos por cada ítem en una lista, mientras que el select permite al usuario seleccionar un ítem de una lista.

El Loop for

La sintaxis es

```
for name in word1 word2 ... wordN
do
        list
done
```

En este caso name es el nombre de una variable mientras que word1 a wordN son secuencias de caracteres separados por espacios(words). Cada vez que el for ejecuta, se setea el valor de la variable name con la siguiente palabra de la lista. La primera vez, name se setea con word1; la segunda vez con word2; y así sucesivamente hasta que se finaliza la lista.

Esto significa que el número de veces que se ejecuta un for depende del largo de la lista.

Un ejemplo simple de for es

```
for i in 0 1 2 3 4 5 6 7 8 9
do
        echo $i
done
```

donde la salida es

```
0
1
2
3
4
5
6
7
8
9
```

Manipulación de un conjunto de archivos

Supongamos que se necesita copiar un conjunto de archivos de un directorio a otro y cambiar los permisos durante la copia. Una forma podría ser copiar los archivos manualmente uno por uno y cambiar los permisos.

Una solución mejor sería determinar los comandos que se deben ejecutar para copiar y cambiar los permisos y hacer que la computadora lo haga para cada archivo. Este es el uso más común del for – iterar sobre un conjunto de nombres de archivos realizando alguna operación sobre estos archivos.

Un ejemplo de esto es:

```
for FILE in $HOME/.bash*
do
        cp $FILE ${HOME}/public_html
        chmod a+r ${HOME}/public_html/${FILE}
done
```

En este loop se utiliza la substitución de archivos para obtener la lista de archivos del home que comienzan con .bash*. En el cuerpo del loop, se copian los archivos al directotio public_html, y luego se hacen legibles para todos los usuarios.

El Loop select

El select provee una forma fácil de crear un menú numerado del cual el usuario puede seleccionar opciones.

La sintaxis del select es

```
select name in word1 word2 ... wordN
do
        list
done
```

En este caso name es el nombre de una variable y word1 a wordN secuencias de caracteres separados por espacios(words). El conjunto de comandos a ejecutar están especificados por list.

La ejecución de un select es de la siguiente forma:

1. Se muestra cada ítem de list1 con un número.
2. Se muestra un prompt, generalmente #?.
3. Cuando el usuario ingresa un valor, se setea $REPLAY a ese valor.
4. Si $REPLY contiene un número correspondiente a una de la las opciones mostradas, se setea la variable especificada por name con el ítem de list1 que fue seleccionado. Si no, se muestra nuevamente los ítems de list1.
5. Cuando se realiza una selección válida, se ejecuta list2.
6. Si list2 no termina el select, por ejemplo con break, comienza de nuevo en el punto 1.

Si el usuario ingresa más de una opción válida, $REPLY contiene todos los valores, pero en este caso no se setea name.

Un ejemplo de select

Un uso común del select es en scripts de configuración de software.

```
select COMPONENTE in comp1 comp2 comp3 todos ninguno
do
```

```
        case $COMPONENTE in
                comp1|comp2|comp3) CompConf $COMPONENTE ;;
                todos) CompConf comp1
                        CompConf comp2
                        CompConf comp3
                        ;;
                ninguno) break ;;
                *) echo "ERROR: Selección inválida, $REPLY." ;;
        esac
done
```

El menú presentado es como el siguiente:

```
1) comp1
2) comp2
3) comp3
4) todos
5) ninguno
#?
```

Se puede observar que cada elemento en la lista comp1 comp2 comp3 all none es mostrado con un número que los precede. el usuario puede ingresar uno de estos números para seleccionar un componente en particular.

Control de Loops

Hasta el momento se vio como crear y trabajar con loops que realizan diferentes tareas. A veces es necesario detener un loop o saltear iteraciones. En esta sección se verán los comandos utilizados para controlar los loops:

- break
- continue

Loops infinitos y el Comando break

Cuando se vio el while, este terminaba cunado una se daba una condición particular.

Si se comete un error especificando la condición de terminación de un while, este puede continuar indefinidamente. Por ejemplo, si se olvida el $ antes de x en la expresión test:

```
x=0
while [ x -lt 10 ]
do
        echo $x
        x=$((x + 1))
done
```

En la mayoría de los casos los loops infinitos son indeseables y resultado de errores de programación, pero en ciertos casos pueden ser útiles.

Por ejemplo, si se necesita esperar por un evento particular, como que ocurra el logging en un sistema. Se puede usar un loop infinito para verificar cada cierta cantidad de segundos que si ocurrió el evento. Como no se sabe cuando puede ocurrir se necesita ejecutar el loop infinito, cuando el evento ocurre se puede terminar el loop con el comando break.

La sintaxis general del loop infinito es

```
while :
do
        list
done
```

En lugar de : se puede utilizar el comando true.

```
while :
do
        read CMD
        case $CMD in
                [qQ]|[qQ][uU][iI][tT]) break ;;
                *) process $CMD ;;
        esac
done
```

El el loop anterior se lee un comando al comienzo de cada iteración. Si el comando es q o Quit, el loop termina; de otra forma, el loop intenta procesar el comando.

Resumen

Los loops son herramientas de programación poderosas que permiten ejecutar un conjunto de comandos en forma repetida. En este capítulo se examinaron los siguientes tipos de loops disponibles en el shell:

- while
- until
- for
- select

Además se cubrieron los conceptos de loops anidados, infinitos y control de loops.