Introducción a Unix: Unidad 5

Curso basado en uno propuesto por William Knottenbelt, UK, 2001

Daniel Millán, Nora Moyano & Evelin Giaroli

Facultad de Ciencias Aplicadas a la Industria, UNCuyo.

Junio de 2017

Comandos avanzados para **shell scripting**

Contenido

1. Introducción

Comúnmente es necesario realizar ***guiones*** que requieren utilizar ciertas órdenes estándares de **Programación Estructurada** [wiki].

La **Programación Estructurada** es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora, utilizando únicamente subrutinas y tres estructuras lógicas (también llamadas estructuras de control): ***secuencia, selección, e iteración***.

- **Secuencia**: ejecución de una instrucción tras otra.
- **Selección:** por medio de sentencias condicionales
  - **if/else**: de acuerdo a una condición
  - **case/switch**: de acuerdo al valor de una variable
- **Iteración** mediante bucles
  - **for**: un número determinado de veces
  - **while**: mientras se cumpla una condición
- Subrutina o subproceso:
  - **function**: realiza una tarea específica

Teorema del programa estructurado

El teorema del programa estructurado es un resultado en la teoría de lenguajes de programación (Corrado Böhm y Giuseppe Jacopini, 1966). Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas.

- Secuencia.
- Instrucción condicional.
- Iteración (bucle de instrucciones) con condición al principio.

Solamente con estas tres estructuras se pueden escribir todos los programas y aplicaciones posibles. Si bien los lenguajes de programación tienen un mayor repertorio de estructuras de control, estas pueden ser construidas mediante las tres básicas citadas.

Este teorema demuestra que la instrucción GOTO no es estrictamente necesaria y que para todo programa que la utilice existe otro equivalente que no hace uso de dicha instrucción.

Ventajas de la programación estructurada

Ventajas de la programación estructurada comparada con el modelo anterior (hoy llamado despectivamente código *spaghetti*[1]).

- Los programas son más fáciles de entender, pueden ser leídos de forma secuencial y no hay necesidad de hacer engorrosos seguimientos en saltos de líneas (GOTO) dentro de los bloques de código para intentar entender la lógica.
- La estructura de los programas es clara, puesto que las instrucciones están más ligadas o relacionadas entre sí.
- Reducción del esfuerzo en las pruebas y depuración. El seguimiento de los fallos o errores del programa (debugging) se facilita debido a su estructura más sencilla y comprensible, por lo que los errores se pueden detectar y corregir más fácilmente.
- Reducción de los costos de mantenimiento. Análogamente a la depuración, durante la fase de mantenimiento, modificar o extender los programas resulta más fácil.
- Los programas son más sencillos y más rápidos de confeccionar.
- Se incrementa el rendimiento de los programadores.

Nuevos paradigmas

Posteriormente a la programación estructurada se han creado nuevos paradigmas tales como la programación modular, la programación orientada a objetos, programación por capas, etc., y el desarrollo de entornos de programación que facilitan la programación de grandes aplicaciones y sistemas.

2. Expresiones aritméticas

- El ***shell Bourne*** (**sh**) no tiene ninguna capacidad incorporada para evaluar expresiones matemáticas sencillas.
- Para ello UNIX ofrece la orden ***expr***. Se utiliza con frecuencia en los ***scripts*** para actualizar el valor de una variable:
  - `var=`expr $var + 1``
- Es por esta razón que se prefiere utilizar la ***Bourne again shell*** (**bash**), la cual permite realizar operaciones aritméticas cuando se utilizan expresiones regulares dentro de paréntesis dobles:
  - `var=$((var+1))`

[1] Término peyorativo para los programas de computación que tienen una estructura de control de flujo compleja e incomprensible. Su nombre deriva del hecho que este tipo de código parece asemejarse a un plato de espaguetis, es decir, un montón de hilos intrincados y anudados.
Tradicionalmente suele asociarse este estilo de programación con lenguajes básicos y antiguos, donde el flujo se controlaba mediante sentencias de control muy primitivas como ***goto*** y utilizando números de línea.

3. Sentencia condicional

En programación, una **sentencia condicional** es una instrucción o grupo de instrucciones que se pueden ejecutar o no en función del valor de una condición.

- Los tipos más conocidos de sentencias condicionales son el SI... ENTONCES (if... then), el SI... ENTONCES... SI NO (if... then... else) y el SEGÚN (case o switch), aunque también podríamos mencionar al manejo de excepciones[2] como una alternativa más moderna para evitar el "anidamiento" de sentencias condicionales.

a. if-then-else-fi

❑ En ***shell scripting*** es posible realizar saltos dependiendo que se cumpla o no alguna condición ***test***:

```
if [ test ]
then
      ordenes-si-test-es-verdadero
else
      ordenes-si-test-es-falso
fi
```

❑ La condición ***test*** puede implicar características de archivos o de cadenas de caracteres sencillas o comparaciones numéricas.

❑ El corchete **[** utilizado aquí es en realidad el nombre de una orden (**/bin/[**) que lleva a cabo la evaluación de la condición en ***test***. Por lo tanto debe haber espacios antes y después de esta orden, así como antes y después del corchete de cierre **]**.

❑ Algunas condiciones comunes son:

-s *archivo*	verdadero si *archivo* existe y no está vacío
-f *archivo*	verdadero si *archivo* es un archivo ordinario
-d *archivo*	verdadero si *archivo* es un directorio
-r *archivo*	verdadero si *archivo* es legible
-w *archivo*	verdadero si se puede escribir en *archivo*
-x *archivo*	verdadero si *archivo* es ejecutable
$X -eq $Y	verdadero si ***X*** es igual a ***Y*** **(==)**
$X -ne $Y	verdadero si ***X*** no es igual a ***Y*** **(!=)**
$X -lt $Y	verdadero si ***X*** menor que ***Y*** **(<)**
$X -gt $Y	verdadero si ***X*** mayor que ***Y*** **(>)**
$X -le $Y	verdadero si ***X*** menor que o igual a ***Y*** **(≤)**
$X -ge $Y	verdadero si ***X*** mayor que o igual a ***Y*** **(≥)**
"$A" = "$B"	verdadero si la cadena ***A*** es igual a la cadena ***B***

[2]El manejo de excepciones es una técnica de programación que permite al programador controlar los errores ocasionados durante la ejecución de un programa informático. Cuando ocurre cierto tipo de error, el sistema reacciona ejecutando un fragmento de código que resuelve la situación, por ejemplo retornando un mensaje de error o devolviendo un valor por defecto.

`"$A" != "$B"`	verdadero si la cadena ***A*** no es igual a la cadena ***B***
`$X ! -gt $Y`	verdadero si la cadena ***X*** no es mayor que ***Y***
`$E -a $F`	verdadero si las expresiones ***E*** y ***F*** son verdaderas
`$E -o $F`	verdadero si la expresión ***E*** o ***F*** es verdadera

b. case

- En ***shell scripting*** se puede utilizar **case** como una forma conveniente para llevar a cabo tareas multipunto, donde un valor de entrada ***variable*** se debe comparar con varias alternativas:

```
case variable in
      patrón1)
            declaración      (ejecutado si variable coincide con patrón1)
            ;;               (fin declaración de patrón1)
      patrón2)
            declaración
            ;;
etc.
esac
```

- Ejemplo **if/case:** estima el tipo de archivo que es pasado como argumento, sobre la base de sus extensiones (ver ***tipoarchivo***).
  - ➢ Observar: (***a***) que el operador "***or***" **|** puede ser utilizado para denotar varios patrones; (***b***) que "*" es empleado para clasificar "***otros***"; y (***c***) el efecto de las comillas simples invertidas **`**.

```
#!/bin/sh
if [ -f $1 -a ! -x $1 ]      #archivo común y no ejecutable
then
    case $1 in
        *.cpp|*.cc|*.cxx)
            echo "$1: a C++ program"
            ;;
        *.txt)
            echo "$1: a text file"
            ;;
         *)
            echo "$1: appears to be " `file -b $1`
            ;;
    esac
fi
```

4. Bucle o ciclo (***loop***)

En programación, es una sentencia que se realiza repetidas veces en un trozo aislado de código, hasta que la condición asignada a dicho bucle deje de cumplirse.

- Generalmente, un bucle es utilizado para hacer una acción repetida sin tener que escribir varias veces el mismo código, lo que ahorra tiempo, procesos y deja el código más claro y facilita su modificación en el futuro.
- Los dos bucles más utilizados en programación son el bucle while, y el bucle for.
- En ***bash shell***:

```
$ for var in {1..30}; do echo -n $var/Abril; done
$ d=1; while [ $d -le 30 ]; do echo $d/Abril; d=$((d+1)); done
```

a. for-do-done

❑ En ***shell scripting*** es posible realizar bucles **for**, que nos permite realizar ciertas operaciones un número determinado de veces.

❑ Este tipo de bucle es muy útil por ejemplo cuando queremos movernos a través de una lista de archivos, e ir ejecutando algunas órdenes en cada archivo de la lista.

```
for variable in list
do
        declaraciones (en referencia a $variable)
done
```

❑ Otro modo de empleo es utilizar **for** con un contador (en **bash**):

```
for ((i=0; i<10; i++))          ➔ notar: ((expresión aritmética))
do
        declaraciones (en referencia a $i)
done
```

b. while

❑ En ***shell scripting*** también es posible realizar bucles **while**, que permite realizar ciertas operaciones de forma cíclica mientras se cumpla alguna condición ***test***:

```
while [ test ]
do
        ejecuta-órdenes-mientras-test-es-verdadero
done
```

- El siguiente ***script*** espera mientras el archivo input.txt esté vacío.

```
#!/bin/sh
while [ ! -s input.txt ]
do
    echo waiting...
    sleep 5
done
echo input.txt is ready
```

5. Subrutina o subprograma

En computación, una subrutina o subprograma (procedimiento, función, rutina o método) como idea general se presenta como un subalgoritmo que forma parte del algoritmo principal, el cual permite resolver una tarea específica.

- Algunos lenguajes de programación, como Fortran, utilizan el nombre función para referirse a subrutinas que devuelven un valor.
- Concepto
  - ➢ Se le llama subrutina a un segmento de código separado del bloque principal y que puede ser invocado en cualquier momento desde este o desde otra subrutina.
  - ➢ Una subrutina, al ser llamada dentro de un programa, hace que el código principal se detenga y se dirija a ejecutar el código de la subrutina.

a. function

- En ***shell scripting*** también pueden incluirse funciones. Las funciones se declaran como:

```
function nombrefun () {
      declaraciones ;      ➔ NO olvidar el punto y coma!!!
}
```

y se invoca como: ***nombrefun param1 param2*** ...

- Los parámetros pasados a la función son accesibles a través de las variables $1, $2, etc.
- El siguiente ***script*** permite encontrar un archivo de forma recursiva dentro del directorio actual de trabajo

```
#!/bin/bash
function findfile() { find . -name '*'$1'*' ; }
findfile $1
```