Introducción a Unix

Unidad 5

Daniel Millán
Evelin Giaroli & Nora Moyano
Facultad de Ciencias Aplicadas a la Industria
Universidad Nacional de Cuyo

2017

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

Comandos avanzados para ***shell scripting***

Los temas que se cubrirán son:

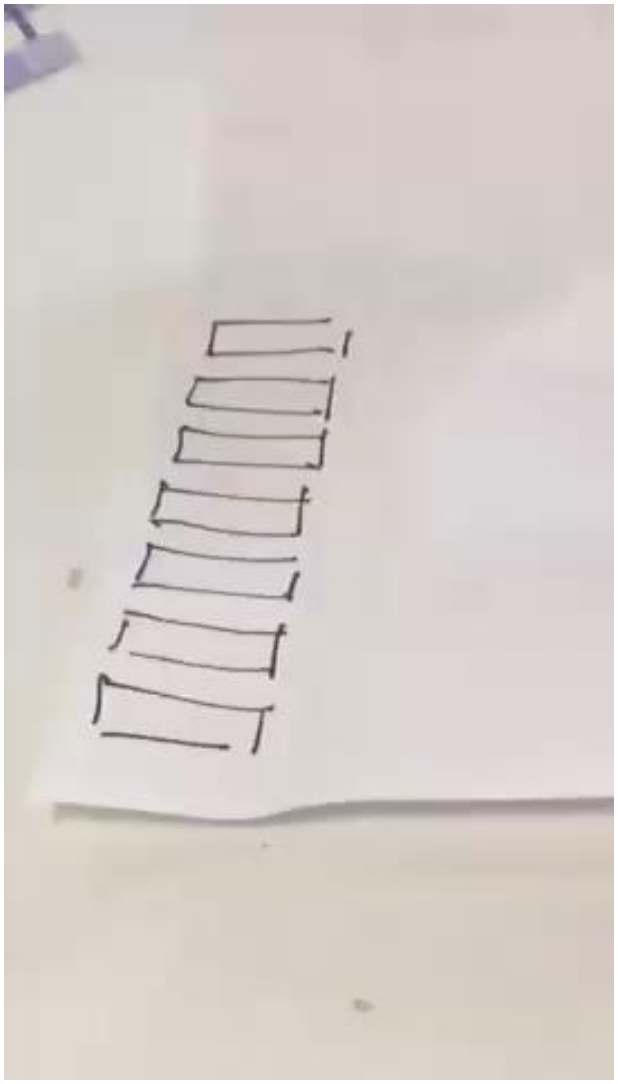
- ***if/else***
- ***for***
- ***while***
- ***case***
- ***function***

Programación Estructurada

Objetivos

- Adquirir un conocimiento básico de órdenes avanzadas y su modo de empleo.

- Desarrollar un pensamiento sistemático y analítico de programación en *shell sripting.*

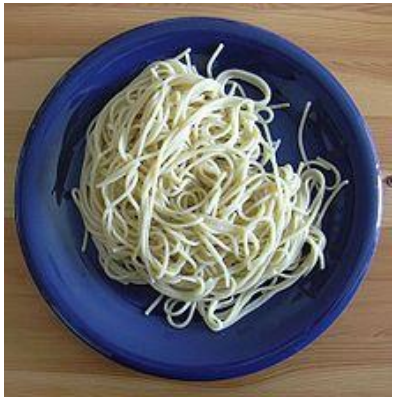


1. Introducción

- Comúnmente es necesario realizar *guiones* que requieren utilizar ciertas órdenes estándares de **Programación Estructurada**.
- 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: *secuencia, selección, e iteración*.
  - ➢ 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

1. Introducción

- Programación estructurada comparada con el código *spaghetti*.



- ➢ Los programas son más fáciles de entender, dado que es posible su lectura secuencial y no hay necesidad de hacer engorrosos GOTO.
- ➢ 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) es más simple.
- ➢ Reducción de los costos de mantenimiento. Modificar o extender los programas resulta más fácil.
- ➢ Los programas son más sencillos y más rápidos de confeccionar.

2. Expresiones aritméticas

- El *shell Bourne* (**sh**) no tiene ninguna capacidad incorporada para evaluar expresiones matemáticas sencillas.
- Para ello UNIX ofrece la órden ***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))`

3. Sentencias condicionales

- **if-then-else-fi**
- En ***shell scripting*** es posible realizar saltos dependiendo del resultado de cumplir 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 órden (**/bin/[**) que lleva a cabo la evaluación de la condición en *test*. Por lo tanto debe haber espacios antes y después de esta órden, así como antes y después del corchete de cierre **]**.

3. Sentencias condicionales

- 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" = "$B"	verdadero si la cadena *A* es igual a la cadena *B*
"$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

3. Sentencias condicionales

- **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
```

3. Sentencias condicionales

- 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*)

- Un **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
```

4. Bucle o ciclo (*loop*)

- **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

4. Bucle o ciclo (*loop*)

- El siguiente *script* ordena cada archivo en el directorio de trabajo actual

```
#!/bin/sh
for f in *.txt
do
    echo ordenando archivo $f
    cat $f | sort > $f.sorted
    echo archivo ordenado ha sido redirecconado a $f.sorted
done
```

4. Bucle o ciclo (*loop*)

- **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
    ejectua-ordenes-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 programa

- En computación, una **subrutina** o **subprograma** (también llamada **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.

5. Subrutina o programa

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

```
function nombrefun () {
    declaraciones ;
}
```

➔ NO olvidar el punto y coma!!!

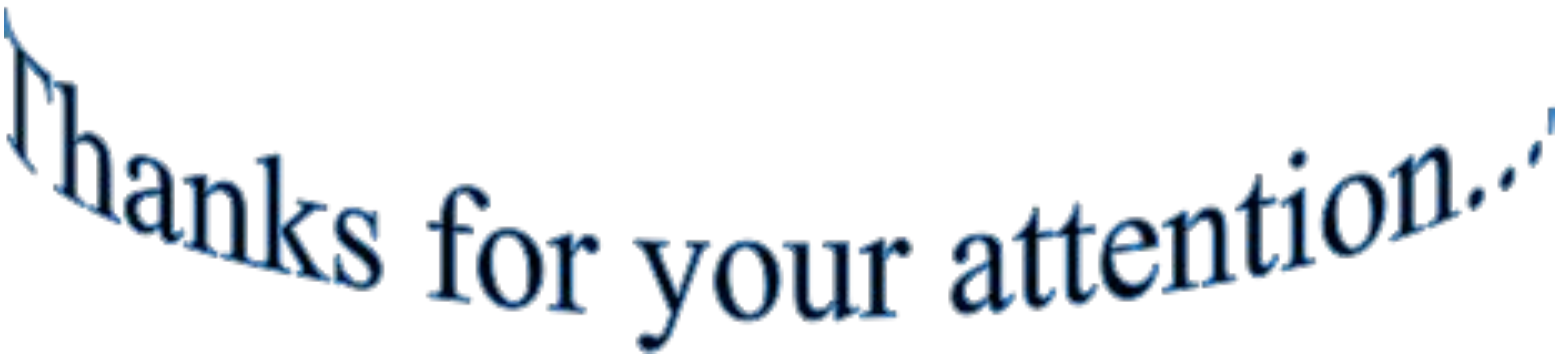
y se invoca como: *nombrefun param1 param2* ...

- 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
```

There is really no secret about our approach. We keep moving forward opening new doors and doing new things because we are curious. And curiosity keeps leading us down new paths. We are always exploring and experimenting. At WED*, we call it **Imagineering**. The blending of creative imagination with technical know-how.

Walt E. Disney 1965 Presentation ``Total Image''



*Disney called WED to ``My back yard laboratory, my workshop away from work.''