Programando con el intérprete de órdenes (shell) de UNIX

Fundamentos de Informática II.
Práctica 3

12 de mayo de 2004

Resumen

Con el fin de facilitar la lectura y comprensión de los guiones de prácticas seguiremos el siguiente convenio que se presenta a continuación.

- Las palabras reservadas de UNIX se presentarán en **negrita** y deben escribirse tal y como se presentan.
- La escritura en letra *cursiva* está reservada para variables y constantes. Es decir, las palabras escritas de esta manera deben sustituirse por las variables o constantes que se consideren convenientes.
- Las palabras escritas en letras MAYÚSCULAS están reservadas para las variable de entorno que el intérprete predefine por defecto u omisión (*default* en inglés). Estas variables se analizarán brevemente en este documento.
- Siempre que aparezca el símbolo **$** al principio de la línea se trata del **indicador** (*prompt* en inglés) de la línea de órdenes del intérprete y no debe escribirse como parte de las órdenes. De aquí en delante se tomarán las siguientes convenciones para la descripción de las órdenes del intérprete de UNIX.

1. Uso de *read* para aceptar entradas desde el teclado

Cuando se desea que un guión acepte entradas desde el teclado, durante la ejecución del proceso, se puede utilizar la sentencia **read**. Esta sentencia toma una linea introducida desde el teclado y la asigna a una o varias variables. La sintaxis es:

read nombre(s) de la(s) variable(s)

NOTA: Si se leen varias variables a la vez, hay que separar sus nombres con un espacio y el número de valores introducidos debe coincidir con el de variables a leer.

EJEMPLO: A continuación se usa la orden **cat** para generar un guión que pide al usuario su nombre y apellidos para luego desplegarlo en pantalla.

$ cat > saludo.sh
echo Escribe tu nombre y apellidos
read nombre apellido1 apellido2
echo Hola $apellido1 $apellido2 , $nombre , mucho gusto.
$< Ctrl > + < d >$

2. Ejecución de expresiones aritméticas usando la sentencia *expr*

La orden **expr** realiza operaciones aritméticas con enteros y presenta el resultado en pantalla. La sintaxis de esta sentencia es:

expr expresión_aritmética

Los operadores aritméticos que pueden utilizarse son:

+ Suma
- Resta
* Producto
/ Cociente de división
% Residuo de división

NOTAS:

- Debido a que el símbolo * tiene un significado especial para el intérprete, cuando se empleé hay que anular ese significado especial usando las formas * , " * " o ' * '.
- Tanto los operandos como los operadores deben ir separados por espacios y el orden de la prioridad de las operaciones es %, * /, + -.
- Se puede asignar el resultado de **expr** a una variable delimitando la sentencia y sus operandos con el acento grave.

EJEMPLOS: Ejecuta las siguientes sentencias en la linea de órdenes

```
$ expr 13 + 49
62
$ a=3
$ expr $a \* 10 / 2
15
$ x=`expr 3 + 5`
$ echo $x
8
```

PRÁCTICA: Elaborar un guión que solicite dos números enteros que son precio y unidades; comprobar que se dan solo dos argumentos y en caso contrario salir del guión. Si solo hay dos argumentos, mostrar el importe total aplicando un IVA del 16 %. Hay que ignorar decimales.

NOTA: Se pueden introducir comentarios en los guiones usando el carácter '#' al inicio de cada linea de comentario que documente el guión.

3. Bucles usando la sentencia *for*

La sentencia compuesta **for** produce la ejecución repetida de una o más órdenes que forman el cuerpo del bucle; la ejecución del cuerpo del bucle se repite para cada uno de los valores que toma una variable dentro de una lista. La sintaxis de esta sentencia es:

for *variable* **in** *lista de valores*
do
Grupo de órdenes a ejecutar (cuerpo del bucle)
done

donde:

variable es el nombre de la variable que tomará los valores indicados en la **lista de variables** y se le puede referenciar en el cuerpo del bucle en la forma *$variable*
do indica el inicio del cuerpo del bucle
done indica el fin del cuerpo del bucle

Esta sentencia se puede ejecutar desde la línea de órdenes del sistema operando de manera análoga a la sentencia **if**. Es decir, al haber introducido la primera línea de la sentencia (la que contiene **for** *variable* **in** *lista de valores*), el indicador del sistema cambia a > y se mantiene así hasta que se introduce la última línea de la sentencia (es decir, **done**), procediéndose a la ejecución del bucle.

EJEMPLO: Ejecutar la siguiente sentencia **for** desde la línea de órdenes.

```
for num in 1 2 3 4 5 6 7 8 9
do
echo El número es $num
done
```

EJEMPLO: Ejecutar la siguiente sentencia **for** desde la línea de órdenes. Se ordenará el contenido de todos los ficheros con extensiones "sh"generando para cada fichero ordenado uno nuevo que tiene el mismo nombre que el original más la extensión "*ord*". Cada vez que se ordena un fichero sale un mensaje indicando tal evento.

```
for fichero in *.sh
do
sort $fichero > $fichero.ord
echo El fichero $fichero ha sido ordenado
done
```

NOTA: Se pueden anidar bucles.

4. Alterando bucles con las órdenes *break* y *continue*

Las órdenes **break** y **continue** se pueden utilizar para alterar incondicionalmente el funcionamiento de un bucle. La orden **break**, en particular, suspende la ejecución del bucle por completo, mientras que la orden **continue** hace que el resto de las instrucciones del bucle sean ignoradas y se ejecute una nueva iteración del bucle.

EJEMPLO: El siguiente guión permite ver el contenido de unos determinados ficheros dados como argumentos al ejecutarlo. Se comprueba que cada fichero existe y se despliega su contenido; si el fichero no existe se pasa al siguiente. Se fija el número máximo de ficheros en tres.

```
for fichero in $1 $2 $3
do
if test ! -s "$fichero"
then
echo El fichero $fichero no existe
sleep 3
continue
else
cat $fichero
sleep 3
fi
done
```

EJEMPLO: El siguiente guión ordena el contenido de tres ficheros dados como argumentos y lo almacena en un único fichero llamado *tres.ord*. Si alguno de los tres ficheros no existe, se abandona la operación, se borra el fichero creado (si existe) y se despliega un mensaje de error.

```
for fichero in $1 $2 $3
do
if test ! -s "$fichero"
then
echo Error: el fichero $fichero no existe
rm -f tres.ord
break
else
sort $fichero >> tres.ord
fi
```

done
echo Terminé la operación

PRÁCTICA: Genera un guión llamado tabla.sh que muestre en pantalla la tabla de multiplicar del número dado como argumento. Al mismo tiempo, la tabla generada debe almacenarse en un fichero llamado tabla.*num*, donde *num* es el número de la tabla de multiplicar (es decir, el argumento). Antes de realizar cualquier operación, comprobar que se ha dado un (y solo un) argumento; si no es así salir inmediatamente del guión desplegando antes un mensaje de error.