Prácticas de Fundamentos del Software

Módulo I. Órdenes UNIX y Shell Bash

Sesión Nº4: Variables, alias, órdenes de búsqueda y guiones

20-Oct-2011

1 Objetivos principales

- Conocer el concepto de variable y los tipos de variables que se pueden usar.
- Distinguir entre variables de entorno o globales y variables locales.
- Saber usar los alias y conocer su utilidad.
- Conocer y usar órdenes de búsqueda en archivos y directorios.
- Conocer qué son los guiones del shell (*scripts*) y cómo podemos ejecutarlos.

Además, se verán las siguientes órdenes:

Órdenes Linux			
`set, unset`	`env, printenv`	`declare`	`expr`
`export`	`alias, unalias`	`find`	`grep, fgrep, egrep`
`printf`			

Tabla 1. Órdenes de la sesión.

2 Variables

Las variables son muy útiles tanto para adaptar el entorno de trabajo al usuario como en la construcción de guiones (o scripts).

2.1 Tipos de variables

El bash contempla dos tipos de variables. Las **variables de entorno** o variables globales son aquellas que son comunes a todos los shells. Para visualizarlas puede probar a usar las órdenes **`env`** o **`printenv.`** Por convención, se usan las letras en mayúscula para los nombres de dichas variables.

Ejercicio 1: Escriba, al menos, cinco variables de entorno junto con el valor que tienen.

Otro tipo de variables son las llamadas **variables locales**, éstas son sólo visibles en el shell donde se definen y se les da valor. Para ver las variables locales se puede usar la orden **`set.`**

Puede consultar una lista de las variables propias del shell bash comentadas usando la orden de ayuda para las órdenes empotradas:

```
$ help variables
```

2.2 Contenido de las variables

Podemos distinguir también las variables según su contenido:

- Cadenas: su valor es una secuencia de caracteres.
- Números: se podrán usar en operaciones aritméticas.
- Constantes: su valor no puede ser alterado.
- Vectores o arrays: conjunto de elementos a los cuales se puede acceder mediante un índice. Normalmente el índice es un número entero y el primer elemento es el 0.

2.3 Creación y visualización de variables

Para asignar un valor a una variable bastará con poner el nombre de la variable, un signo igual y el valor que deseamos asignar, que puede ser una constante u otra variable.

¡Cuidado! a cada lado del signo igual no debe haber ningún espacio en blanco. Si delante o detrás del signo igual dejamos un espacio en blanco obtendremos un error, porque lo tomará como si fuera una orden y sus argumentos, y no como una variable. Y, además, el nombre de una variable puede contener dígitos pero no puede empezar por un dígito.

Para visualizar el valor de una variable se puede usar la orden **`echo`**.

```
$ numero=1
$ echo $numero
1
```

Para crear variables de tipo vector utilizamos la misma forma de definición pero los elementos del vector se ponen entre paréntesis y separados por espacios. Un ejemplo de creación de un vector es:

```
$ colores=(rojo azul verde)
```

Para acceder a uno de sus elementos:

```
$ echo ${colores[0]}
rojo
$ echo ${colores[1]}
azul
```

Existe una serie de variables especiales y otras que se crean al entrar el usuario. A continuación describiremos algunas de ellas (en el resumen disponible en Tutor aparece una lista más completa):

Nombre de variable	Descripción
`$BASH`	Contiene la ruta de acceso completa usada para ejecutar la instancia actual de bash.
`$HOME`	Almacena el directorio raíz del usuario; se puede emplear junto con la orden **`cd`** sin argumentos para ir al directorio raíz del usuario.
`$PATH`	Guarda el camino de búsqueda de las órdenes, este camino está formado por una lista de todos los directorios en los que queremos buscar una orden.
`$?`	Contiene el código de retorno de la última orden ejecutada, bien sea una instrucción o un guion.

Tabla 2. Variables de entorno predefinidas.

Para borrar una variable se usa la orden **`unset`** junto con el nombre de la variable (o variables) a eliminar.

Si deseamos crear una variable con ciertos atributos, utilizaremos la orden **declare**, cuya sintaxis completa se puede ver con **help declare**. Podemos indicar que una variable es numérica con la opción **-i** y ver los atributos con la opción **-p**:

```
$ declare -i IVA=18
$ declare -p IVA
declare -i IVA=”18”
$ declare -i IVA=hola
$ declare -p IVA
declare -i IVA="0"
```

De esta forma cualquier intento de asignar otra cosa diferente de un número a la variable no dará un error. Otros atributos para las variables son: **-r** indica que es de solo lectura, **-a** indica que es una matriz (vector o lista), **-x** indicará que es exportable (ver el sub-apartado siguiente). Estos atributos se pueden mezclar.

2.3 Exportar variables

Las variables locales sólo son visibles en el shell actual o en el guion en el que se definen, para emplear estas variables fuera de ellos, debemos exportar su valor para que el sistema lo reconozca, en caso contrario, cuando acaba el guion o el shell las variables toman su valor original:

```
export variable
```

Ejercicio 2. Ejecute las órdenes del cuadro e indique qué ocurre y cómo puede resolver la situación para que la variable NOMBRE se reconozca en el shell hijo.

```
$ NOMBRE=FS
$ echo $NOMBRE

$ bash
$ echo $NOMBRE
```

A veces se puede poner en una línea la definición y exportación de una variable:

```
export variable=valor
```

2.4 Significado de las diferentes comillas en las órdenes

En el shell bash se puede hacer uso de lo que se denomina *sustitución de órdenes*, que permite la ejecución de una orden, con o sin argumentos, de forma que su salida se trata como si fuese el valor de una variable. La sustitución de órdenes se puede hacer poniendo **$(***orden argumentos***)**, o usando los apóstrofos inversos, es decir `` `orden argumentos` ``, y puede utilizarse en cualquier tipo de expresión, en particular en las expresiones aritméticas que se verán en la sesión siguiente.

Con el ejemplo siguiente se plantean dos expresiones que son equivalentes:

```
$ echo “Los archivos que hay en el directorio son: $(ls -l)”

$ echo “Los archivos que hay en el directorio son: `ls -l`”
```

Ejercicio 3: Compruebe qué ocurre en las expresiones del ejemplo anterior si se quitan las comillas dobles del final y se ponen después de los dos puntos. ¿Qué sucede si se sustituyen las comillas dobles por comillas simples?

En los casos anteriores, las comillas dobles se utilizan como mecanismo de *acotación débil* (*weak quotation*), para proteger cadenas desactivando el significado de los caracteres especiales que haya entre ellas, salvo los caracteres `!`, `$` y `` ` ``, que no quedan protegidos. También se pueden proteger cadenas usando comillas simples como mecanismo de *acotación fuerte* (*strong quotation*), aunque en este caso se protegen los caracteres especiales salvo `!`; en consecuencia, las comillas simples serán útiles cuando se quiera proteger una variable o una orden.

Cuando se usan comillas simples dentro de una expresión, por ejemplo, si se deseara imprimir un mensaje como el siguiente: `'En el libro de inglés aparece Peter's cat'`, si se emplea la orden **`echo`**, nos aparecerá en la línea siguiente el carácter **>** que representa una línea de continuación de la orden dada, es decir, es como si no se hubiera completado el mensaje de texto. Para hacerlo correctamente, habría que escapar la comilla de la palabra Peter's con el metacarácter **** y partir la frase en dos partes acotadas con comillas simples como se muestra a continuación:

```
$ echo 'En el libro de inglés aparece Peter's cat'
>
$ echo 'En el libro de inglés aparece Peter'\''s cat'
En el libro de inglés aparece Peter's cat
```

2.5 Asignación de resultados de órdenes a variables

Podemos asignar el resultado de una orden a una variable a través del operador ` (comilla invertida). Para ello utilizamos la siguiente declaración:

```
variable=`orden`
```

Por ejemplo, si queremos declarar una variable que se llame `listadearchivos` y que contenga el listado de archivos del directorio actual, podemos hacerlo con la declaración siguiente, donde la variable que se usa será de tipo lista:

```
$ listadearchivos=`ls .`
```

Ahora, la variable contendrá la lista de todos los archivos existentes en el directorio actual.

Es posible que la ejecución de algunas órdenes situadas entre comillas invertidas pudiera producir algún error. Un ejemplo de una situación como esta es cuando se ejecuta la orden `cat <archivo>` y el archivo dado como argumento de la misma no existe. En esta situación, con objeto de depurar el correcto funcionamiento, puede ser útil conocer el estado de la ejecución de la última orden, que valdrá `0` si se ejecutó correctamente, o `1` si hubo algún error, como es el caso del ejemplo que se muestra a continuación:

```
$ cat archivomio
cat: archivomio: No existe el fichero o el directorio
$ echo $?
1
```

Ejercicio 4: Pruebe la siguiente asignación:

```
$ numero=$numero + 1
$ echo $numero
```

¿Qué ha ocurrido?

Como vemos en el ejemplo anterior, todo se ha convertido en carácter, y no se ha realizado la operación matemática que deseábamos. La solución a este problema viene de la mano de la orden del sistema **`expr`**, con la que podemos evaluar la expresión que le sigue.

```
$ numero=1
$ echo $numero
1
$ numero=`expr $numero + 1`
$ echo $numero
2
```

Hemos de fijarnos en que la orden **`expr`** y sus argumentos se encuentran entre apóstrofes inversos, de tal forma que a `numero` no se asigne la palabra **`expr`**, sino el resultado de la ejecución de la orden **`expr`**.

3 La orden empotrada printf

La orden **`echo`** puede tener comportamientos diferentes según el sistema Unix que utilicemos, por ello es recomendable utilizar la orden **`printf`**. La orden empotrada **`printf`** (*print format*) imprime un mensaje en la pantalla utilizando el formato que se le especifica. Su sintaxis es:

```
printf formato [argumentos]
```

Donde `formato` es una cadena que describe cómo se deben imprimir los elementos del mensaje. Esta cadena tiene tres tipos de objetos: texto plano, que simplemente se copia en la salida estándar; secuencias de *caracteres de escape*, que son convertidos y copiados en la salida (ver Tabla 3); y especificaciones de formato, que se aplican cada una a uno de los argumentos (ver Tabla 4).

Secuencia de escape	Acción
`\b`	Espacio atrás
`\n`	Nueva línea
`\t`	Tabulador
`\'`	Carácter comilla simple
`\\`	Barra invertida
`\0n`	n = número en octal que representa un carácter ASCII de 8 bits

Tabla 3. Algunos códigos de escape.

Código de formato	Representa
`%d`	Un número con signo
`%f`	Un número en coma flotante (decimal) sin notación exponencial
`%q`	Entrecomilla una cadena
`%s`	Muestra una cadena sin entrecomillar
`%x`	Muestra un número en hexadecimal
`%o`	Muestra un número en octal

Tabla 4. Algunos códigos de formato.

A continuación veremos algunos ejemplos de la orden. En el primero, se imprime un número en una columna de 10 caracteres de ancho:

```
$ printf "%10d\n" 25
       25
```

Podemos justificar a la izquierda, si utilizamos un número negativo:

```
$ printf "%-10d %-10d\n" 11 12
11      12
```

Si el número que especificamos en el formato es un decimal, la parte entera se interpreta como la anchura de la columna y el decimal como el número mínimo de dígitos:

```
$ printf "%10.3f\n" 15,4
    15,400
```

Podemos convertir un número de octal o hexadecimal a decimal:

```
$ printf "%d %d\n" 010 0xF
8 15
```

Y a la inversa, de decimal a octal/hexadecimal:

```
$ printf "0%o 0x%x\n" 8 15
00 0xf
```

También podemos usar variables como argumentos de la orden **printf**. Por ejemplo, si queremos mostrar la variable IVA, antes declarada, junto con un mensaje explicativo, escribiríamos:

```
$ printf "El valor actual del IVA es del %d\n" $IVA
El valor actual del IVA es del 18
```

4 Alias

Los alias se crean con la orden empotrada **alias** y se borran o eliminan con la orden **unalias**. Puedes usar la orden **help** para conocer cuál es la sintaxis de estas dos órdenes.

Los alias son útiles para definir un comportamiento por defecto de una orden o cambiar el nombre de una orden por estar acostumbrado a usar otro sistema. Por ejemplo, los usuarios de Windows pueden estar acostumbrados a usar la orden **dir** para listar el contenido de un directorio, para ellos sería útil usar un alias llamado **dir** que realice esta función:

```
$ alias dir='ls -l'
$ dir
```

Ejecute alias sin argumento y comprobará los alias por defecto que están definidos en su sistema.

Además, dentro de un alias y entre comillas podemos poner varias órdenes separadas por ";" de tal forma que se ejecutarán cada una de ellas secuencialmente.

Para ignorar un alias y ejecutar la orden original (por ejemplo **ls**, y siempre y cuando no haya más de una orden en el alias) se antepone una barra invertida (****) al nombre del alias, de la siguiente forma:

```
$ \ls –l $HOME
```

En algunas distribuciones de Linux, como por ejemplo Ubuntu o Guadalinex, cuando se escribe la orden **ls**, aparece el listado de manera coloreada, es decir, los directorios aparecen en color azul, los archivos con permiso de ejecución aparecen en color verde, etc. En realidad se debe a que ya existe un alias definido con la opción de listar los archivos con la opción de color de forma automática (**ls --color=auto**). Usando la distribución de Linux disponible en el aula de ordenadores, liste los archivos y observe si se muestran con la opción de color, si no se muestra de manera coloreada, defina un alias para que la orden **ls** los muestre.

5 Órdenes de búsqueda: find y grep, egrep, fgrep

Es importante disponer de herramientas para realizar búsquedas dentro de los archivos y en la estructura de directorios. Para ello son útiles las siguientes órdenes.

5.1 Orden find

Se utiliza para buscar por la estructura de directorios los archivos que satisfagan los criterios especificados. Su formato es el siguiente:

find lista-de-directorios [expresiones]

donde lista-de-directorios es la lista de directorios a buscar, y las expresiones son los operadores que describen los criterios de selección para los archivos que se desea localizar y la acción que se quiere realizar cuando **find** encuentre dichos archivos.

Los criterios se especifican mediante una palabra precedida por un guion, seguida de un espacio y por una palabra o número entero precedido o no por un **+** o un **-**. Ejemplos de criterios comunes para localizar archivos son:

1. Por el nombre del archivo: se utiliza la opción **-name** seguida por el nombre deseado. Este nombre puede incluir la expansión de metacaracteres de archivo debidamente acotados. Por ejemplo:

```
$ find / -name "*.c"
```

2. Por el último acceso: se utiliza la opción **-atime** seguida por un número de días o por el número y un signo **+** o **-** delante de él. Por ejemplo:

-atime 7 busca los archivos a los que se accedió hace 7 días.
-atime -2 busca los archivos a los que se accedió hace menos de 2 días.
-atime +5 busca los archivos a los que se accedió hace más de 5 días.

3. Por ser de un determinado tipo: se utiliza la opción **-type** seguida de un carácter que indique el tipo de archivo. Se usa la opción **f** para referirse a archivos regulares y la opción **d** para directorios. En el ejemplo siguiente se muestra la búsqueda de los archivos del directorio actual que sean archivos regulares:

```
$ find . -type f
```

4. Por su tamaño en bloques: se utiliza la opción **-size** seguida de un número con o sin signo (**+** o **-**). Si el número va seguido de la letra **c** el tamaño dado es en bytes. Por ejemplo: **-size 100** busca los archivos cuyo tamaño es de 100 bloques.

Además, se puede negar cualquier operador de selección o de acción utilizando el operador **!** que debe ir entre espacios en blanco y antes del operador a negar. Por ejemplo, para buscar los archivos del directorio raíz que no pertenezcan al usuario llamado pat:

```
$ find / ! -user pat
```

También se puede especificar un operador u otro utilizando el operador **-o**. Este operador conecta dos expresiones y se seleccionarán aquellos archivos que cumplan una de las dos expresiones (y no las dos, como hasta ahora). En el siguiente ejemplo se muestra la búsqueda de los archivos de tamaño igual a 10 bloques o cuyo último acceso (modificación) se haya efectuado hace más de dos días:

```
$ find . -size 10 -o -atime +2
```

Las acciones más comunes son:

-print: visualiza los nombres de camino de cada archivo que se adapta al criterio de búsqueda. Es la opción por defecto. Por ejemplo, para visualizar los nombres de todos los archivos y directorios del directorio actual:

```
$ find . -print
```

-exec: permite añadir una orden que se aplicará a los archivos localizados. La orden se situará a continuación de la opción y debe terminarse con un espacio, un carácter **** y a continuación un **;**. Se utiliza **{}** para representar el nombre de archivos localizados. Por ejemplo:

```
$ find . -atime +100 -exec rm {} \;
```

eliminará todos los archivos del directorio actual (y sus descendientes) que no ha sido utilizados en los últimos 100 días.

-ok: es similar a **-exec**, con la excepción de que solicita confirmación en cada archivo localizado antes de ejecutar la orden.

5.2 Órdenes grep, egrep y fgrep

La orden **grep** permite buscar cadenas en archivos utilizando patrones para especificar dicha cadena. Esta orden lee de la entrada estándar o de una lista de archivos especificados como argumentos y escribe en la salida estándar aquellas líneas que contengan la cadena. Su formato es:

grep opciones patrón archivos

En su forma más sencilla, el patrón puede ser una cadena de caracteres, aunque, como se verá en la siguiente sesión, también pueden usarse expresiones regulares. Por ejemplo, para buscar la palabra `mundo` en todos los archivos del directorio actual usaremos:

```
$ grep mundo *
```

Algunas opciones que se pueden utilizar con la orden **grep** son:

- **-x** localiza líneas que coincidan totalmente con el patrón especificado.
- **-v** selecciona todas las líneas que no contengan el patrón especificado.
- **-c** produce solamente un recuento de las líneas coincidentes.
- **-i** ignora las distinciones entre mayúsculas y minúsculas.
- **-n** añade el número de línea en el archivo fuente a la salida de las coincidencias.
- **-l** selecciona sólo los nombres de aquellos archivos que coincidan con el patrón de búsqueda.

Existen dos variantes de **grep** que optimizan su funcionamiento en casos especiales. La orden **fgrep** acepta sólo una cadena simple de búsqueda en vez de una expresión regular. La orden **egrep** permite un conjunto más

complejo de operadores en expresiones regulares. Usando **man** comprueba las diferencias entre estas tres órdenes.

6 Guiones

Un *guion* del shell (script o programa shell) es un archivo de texto que contiene órdenes del shell y del sistema operativo. Este archivo es utilizado por el shell como guía para saber qué órdenes ejecutar.

Siguiendo la similitud de ejemplos de lenguajes de programación de alto nivel, comencemos con el típico ejemplo `"Hola Mundo"` para mostrar dicho mensaje mediante un guion o script bash. Para ello, abriremos un editor de textos ascii (vi, kedit, gedit, emacs, xemacs, ...) y escribimos lo siguiente:

```
echo "Hola Mundo"
```

Mediante el editor guardamos este documento con el nombre `holamundo`; para mostrar el resultado de su ejecución nos vamos al terminal y escribimos lo siguiente:

```
$ bash holamundo
```

Esta orden, lanza un shell bash y le indica que lea las órdenes del guion de prueba, en lugar de usar el propio terminal. De esta forma podemos automatizar procesos al liberar al usuario de estar escribiendo las órdenes repetidamente.

Para tratar de ilustrar las diferentes invocaciones de variables con comillas simples, dobles y con la barra invertida, crearemos un documento que denominaremos `imprimevar` con las siguientes órdenes:

```
variable=ordenador
printf "Me acabo de comprar un $variable\n"
printf 'Me acabo de comprar un $variable\n'
printf "Me acabo de comprar un \$variable\n"
```

Una vez guardado el documento anterior, podremos invocarlo mediante `bash imprimevar` y tendremos el siguiente resultado tras su ejecución:

```
$ bash imprimevar
Me acabo de comprar un ordenador
Me acabo de comprar un $variable
Me acabo de comprar un $variable
```

Otro ejemplo que podemos mostrar es listar los archivos del directorio del usuario. Para ello, creamos un archivo de texto que contenga las siguientes líneas y que denominaremos `prueba`:

```
printf "El directorio $HOME contiene los siguientes archivos:\n"
ls $HOME
```

La invocación del guión anterior sería mediante la orden `bash prueba`, sin embargo, es posible simplificar el proceso de invocación de un guion para que, en lugar de hacerlo explícitamente con la palabra bash, podamos poner dentro del propio archivo el tipo de shell que se debe utilizar aprovechando las facilidades que nos ofrece el sistema operativo. Para ello, debemos poner siempre en la primera línea del archivo los símbolos **#!** seguidos del nombre del programa del tipo de shell que deseamos ejecutar, para nuestro caso, `/bin/bash`. Con esto, nuestro ejemplo anterior quedaría:

```
#!/bin/bash
```

```
printf "El directorio $HOME contiene los siguientes archivos:\n"
ls $HOME
```

Ahora, podemos hacer nuestro archivo ejecutable (`chmod +x prueba`) con lo que la ejecución sería[1]:

```
$ ./prueba
```

Aclaración: Hemos antepuesto **`./`** al nombre de la orden por la siguiente razón: tal y como podemos observar, la variable **`$PATH`** (que contiene la lista de directorios donde el sistema busca las órdenes que tecleamos) no contiene nuestro directorio de trabajo, por lo que debemos indicarle al shell que la orden a ejecutar está en el directorio actual (.).

Nombre	Descripción
`$0`	Nombre del guion o script que se ha llamado. Sólo se emplea dentro del guion.
`$1 .. $9` **`${n}, n>9`**	Son los distintos argumentos que se pueden facilitar al llamar a un guion. Los nueve primeros se referencian con **`$1,$2,...,$9`**, y a partir de ahí es necesario encerrar el número entre llaves, es decir, **`${n}`**, para **`n>9`**.
`$*`	Contiene el nombre del guion y todos los argumentos que se le han dado. Cuando va entre comillas dobles es equivalente a `"$1 $2 $3 … $n"`.
`$@`	Contiene el nombre del guion y todos los argumentos que se le han dado. Cuando va entre comillas dobles es equivalente a `"$1" "$2" … "$n"`.
`$#`	Contiene el número de argumentos que se han pasado al llamar al guion.
`${arg:-val}`	Si el argumento tiene valor y es no nulo, continua con su valor, en caso contrario se le asigna el valor indicado por *val*.
`${arg:=val}`	Si el argumento no tiene valor o es nulo, se le asigna el valor indicado por *val*.
`${arg:?val}`	Si el argumento tiene valor y es no nulo, sustituye a su valor; en caso contrario, imprime el valor de *val* y sale del guion. Si *val* es omitida, imprime un mensaje indicando que el argumento es nulo o no está asignado.

Tabla 5. Variables de entorno definidas para los argumentos de un guion.

Como se ve en la tabla anterior, las variables numéricas nos permiten pasar argumentos a un guion para adaptar su comportamiento, son los *parámetros* del guion. Se puede pasar cualquier número de argumentos, pero por compatibilidad con versiones anteriores del shell que sólo permitía del 0 al 9, los argumentos por encima del 9 se suelen ponen entre llaves, por ejemplo, **`${12}`**.

Ahora podemos adaptar el guion `prueba`, creado anteriormente, para que nos muestre el contenido de cualquier directorio:

```
#!/bin/bash
printf "El directorio $1 contiene los siguientes archivos:\n"
```

[1] Por defecto, en las aulas de la ETSIIT, los terminales se inician con una shell csh (C-Shell). Es necesario poner la orden bash o en el encabezado del guion o en el propio terminal para disponer de una shell acorde a la sintaxis del lenguaje de órdenes de la shell que se verá, es decir, la shell de tipo Bourne again shell, también conocida como Shell Bash. Algo descriptivo es que con el intérprete de órdenes de tipo C-Shell, el terminal aparece con el prompt `%`, mientras que con la Shell Bash aparecerá información del usuario, nombre del host e información del directorio donde se encuentra y el símbolo **`$`**.

```
ls $1
```

De esta forma, si queremos ver el contenido del directorio `/bin` solo debemos ejecutar:

```
$ ./prueba /bin
```

Otro ejemplo sencillo de guion bash sería el de realización de una copia en otro directorio de todos los archivos y subdirectorios del directorio home de un usuario. Al igual que antes, se abre el editor con el que más cómodos nos sintamos y escribimos lo siguiente:

```
#!/bin/bash
printf "Haciendo copia de seguridad en $HOME...\n"
cp -r $HOME/* /tmp/backupuser/
printf "Copia realizada\n"
```

Guardamos el documento anterior con el nombre `mybackup` y ya podremos realizar cuando deseemos una copia de seguridad completa de nuestro directorio `$HOME` (suponemos que el directorio `backupuser` ya está creado previamente).

Ejercicio 5. Construya un guion que acepte como argumento una cadena de texto (por ejemplo, su nombre) y que visualice en pantalla la frase `Hola` y el nombre dado como argumento.

Ejercicio 6. Varíe el guion anterior para que admita una lista de nombres.

6.1 Normas de estilo

Entre las normas que se dan a la hora de escribir un guion, se indica que es buena costumbre comentarlo para conocer siempre quién lo ha escrito, en qué fecha, qué hace, etc. Para ello, utilizaremos en símbolo "**#**", bien al inicio de una línea o bien tras una orden. Por ejemplo, nuestros guiones pueden empezar:

```
#!/bin/bash
# Titulo:      prueba
# Fecha:       5/10/2011
# Autor:       Profesor de FS
# Versión:     1.0
# Descripción: Guion de prueba para la Sesión 4
# Opciones: Ninguna
# Uso: prueba directorio

printf "El directorio $1 contiene los siguientes archivos:\n"
ls  $1      # lista los archivos del directorio que se le pase como argumento
```

Podemos encontrar más normas de estilo en el documento "Bash Style Guide and Coding Standard" de Fritz Mehner, 2009, disponible en http://lug.fh-swf.de/vim/vim-bash/StyleGuideShell.en.pdf.

También, la Free Software Foundation tiene disponible una serie de guías para escribir software GNU en las que se describe la forma estándar en la que deben operar las utilidades Unix. Está accesible en http://www.gnu.org/prep/standards/. Por ejemplo, un guion debe al menos soportar dos opciones `-h` (o `--help`) para la ayuda y `--version`) para indicar la versión, nombre, autor, etc.

Tomando como referencia el ejemplo de la copia de seguridad visto anteriormente se podría modificar el guion con un argumento que indique el destino de los archivos que se desean copiar.

El guion quedaría como se muestra en el siguiente ejemplo:

```
#!/bin/bash
# Titulo:      mybackup
# Fecha:       5/10/2011
# Autor:       Profesor de FS
# Versión:     1.0
# Descripción: Realiza una copia de seguridad de los archivos del usuario
#              en un directorio dado como argumento.
# Opciones: Ninguna
# Uso: mybackup destino

printf "Haciendo copia de seguridad de $HOME...\n"
cp -r $HOME/* $1
printf "Copia realizada\n"
```

Ejercicio 7. Cree tres variables llamadas `VAR1`, `VAR2` y `VAR3` con los siguientes valores respectivamente "hola", "adios" y "14".

a) Imprima los valores de las tres variables en tres columnas con 15 caracteres de ancho.
b) ¿Son variables locales o globales?
c) Borre la variable VAR2.
d) Abra otra ventana de tipo terminal, ¿puede visualizar las dos variables restantes?
e) Cree una variable de tipo vector con los valores iniciales de las tres variables.
f) Muestre el segundo elemento del vector creado en el apartado e.

Ejercicio 8. Cree un alias que se llame **`ne`** (nombrado así para indicar el número de elementos) y que devuelva el número de archivos que existen en el directorio actual. ¿Qué cambiaría si queremos que haga lo mismo pero en el directorio home correspondiente al usuario que lo ejecuta?

Ejercicio 9. Indique la línea de orden necesaria para buscar todos los archivos a partir del directorio home que tengan un tamaño menor de un bloque. ¿Cómo la modificaría para que además imprima el resultado en un archivo que se cree dentro del directorio donde nos encontremos y que se llame `archivosP`?

Ejercicio 10. Indique cómo buscaría todos aquellos archivos del directorio actual que contengan la palabra "ejemplo".

Ejercicio 11. Complete la información de `find` y `grep` utilizando para ello la orden `man`.

Ejercicio 12. Indique cómo buscaría si un usuario dispone de una cuenta en el sistema.

Ejercicio 13. Indique cómo contabilizar el número de ficheros de la propia cuenta de usuario que no tengan permiso de lectura para el resto de usuarios.

Ejercicio 14. Modifique el ejercicio 8 de forma que, en vez de un alias, sea un guion llamado **numE** el que devuelva el número de archivos que existen en el directorio que se le pase como argumento.