

Ejercicios de exámenes anteriores de guiones shell

Noviembre 2008

Notas:

- Los siguientes ejercicios formaron parte de los exámenes de guiones shell de anteriores convocatorias.
- Los exámenes constan siempre de dos ejercicios de este tipo, de los cuales al menos uno debe funcionar correctamente para superar la prueba. El tiempo disponible para la realización de la misma es de una hora y quince minutos.
- En la página web de la asignatura se pueden encontrar las soluciones de los ejercicios 1 a 8 (ambos inclusive). Por supuesto, se recomienda a los alumnos intentar resolver cada ejercicio sin consultar previamente la solución (que, por otra parte, es tan sólo una de las múltiples posibles).
- Por este mismo motivo pedagógico, las soluciones de los ejercicios restantes (9 a 16) no se publican, instándose a los alumnos a que los resuelvan por su cuenta, y en su caso consulten directamente las posibles dudas al profesor.

LISTADO DE EJERCICIOS

1. Implementar un juego (adivina) en el que el ordenador debe adivinar el número del 1 al 10 que el usuario ha pensado. El programa muestra al principio un número al azar y debe pedir al usuario que le diga si el número mostrado es el correcto, es mayor o es menor. Dependiendo de la respuesta, el programa deberá mostrar un número menor o mayor, o informar de que se ha encontrado el número. Por ejemplo:

```
$ adivina
¿El número es el 8? ¿mayor, menor o correcto?
menor          <- escrito por el usuario
¿El número es el 5? ¿mayor, menor o correcto?
correcto       <- escrito por el usuario
¡¡ El número era el 5 !!
```

Así mismo, en su caso el programa deberá detectar e informar al usuario de que éste ha hecho trampa (es decir, ha mentado en alguna de sus afirmaciones).

2. Escribir un guión llamado `terminaminuto` que reciba como parámetros una lista de comandos (posiblemente con argumentos), y que ejecute en segundo plano todos ellos. El guión deberá entonces esperar hasta que llegue el siguiente minuto “en punto” y, para cada proceso lanzado que aún no haya terminado, deberá terminarlo de forma inmediata, mostrando un mensaje adecuado. Para los procesos que terminaron antes de dicho instante no es necesario que salga ningún mensaje.

Ejemplo:

```
$ date
mar dic  5 14:09:20 CET 2006

$ ./terminaminuto kcalc "ls -la" "ps" gedit &
...
```

Suponemos aquí que tanto el `ls` como el `ps` han terminado antes de que lleguen las 14:10:00, pero el `kcalc` y el `gedit` no. Entonces, justo al expirar el minuto actual (es decir, a las 14:10:00), deberían morir estos últimos dos procesos, saliendo los siguientes mensajes por pantalla:

El proceso 8190 ha sido matado por expirar su minuto
El proceso 8192 ha sido matado por expirar su minuto

(Por supuesto, si 8190 y 8192 eran los PIDs del `kcalc` y el `gedit`, respectivamente).

3. Implementar un guión de bash (`lsacum`) que imprima en su salida estándar (por defecto, la pantalla) un listado de todos los ficheros que se le pasan por la línea de comandos, uno por línea (como el que mostraría un `ls -l lista-ficheros`), pero con las siguientes particularidades:
- Deberá mostrar sólo aquellos argumentos correspondientes a ficheros regulares (no directorios, ni enlaces, ni ficheros especiales, etc.)
 - Cada línea mostrará únicamente los permisos del fichero, el tamaño acumulado (ver punto siguiente) y el nombre del fichero, en ese orden.
 - El campo de tamaño de cada fichero individual será sustituido por el tamaño acumulado de los ficheros impresos hasta ese momento, incluyendo el actual.

Ejemplo: si la salida de un comando `ls -l` normal en un directorio dado fuese, por ejemplo:

```
$ ls -l
-rw-r--r--  1 pgarcia profesor   470 dic 10 13:21 Makefile
-rw-r--r--  1 pgarcia profesor 15339 dic 10 13:21 safe.c
-rw-r--r--  1 pgarcia profesor  2076 dic 10 13:21 sender.c
-rw-r--r--  1 pgarcia profesor 15311 dic 10 13:21 simple.c
drwxrwxr-x  2 pgarcia profesor  4096 dic 10 13:21 saved
```

Entonces una llamada `lsacum s*` a nuestro programa debería mostrar en su salida lo siguiente:

```
$ lsacum s*
-rw-r--r--  15339      safe.c
-rw-r--r--  17415      sender.c
-rw-r--r--  32726      simple.c
```

Puesto que `safe.c`, `sender.c` y `simple.c` son ficheros regulares, pero `saved` no (es un directorio), y puesto que $17415=15339+2076$ y $32726=15339+2076+15311$.

4. Cree un guión shell, llamado `mediana`, que para cada fichero pasado como parámetro, calcule la mediana de la serie de números enteros positivos contenidos en cada uno, a razón de uno por cada línea. La sintaxis será la siguiente:

```
mediana <fichero1> [<fichero2>...<ficheroN>]
```

Un ejemplo de resultado de ejecución del comando sería (en el caso de que se introduzcan varios ficheros):

| Fichero | Mediana |
|----------------------|---------|
| ----- | ----- |
| fichero1.txt | 5 |
| fichero1largo.txt | 6 |
| fichero1largo333.txt | 34 |
| ... | |

La definición de mediana es:

Punto medio de los valores después de ordenarlos de menor a mayor, o de mayor a menor. Se tiene que 50% de las observaciones se encuentran por arriba de la mediana y 50% por debajo de ella.

- Por ejemplo, dado un conjunto impar de números: 5, 3, 2, 1, 4:
 - Los números se ordenan: 1, 2, 3, 4, 5.
 - La mediana sería el 3 ya que tiene dos números por debajo de él (1 y 2) y dos números por encima (4 y 5).
- Si el conjunto contuviese un número par de números: 6, 3, 2, 7, 1, 5:
 - Los números se ordenan: 1, 2, 3, 5, 6, 7.
 - En este caso hay dos números en el centro (3 y 5). Entonces la mediana se calcula como la media aritmética de esos dos números. Es decir, la mediana sería 4 $((3+5)/2)$.

5. Implementar un guión de bash (`monton`) que imprima en su salida estándar (por defecto, la pantalla) un triángulo isósceles formado por asteriscos de texto, simétrico verticalmente, y de altura igual al número que se le pasa como único y obligatorio parámetro. Por ejemplo, si se teclea

```
$ ./monton 5
```

el programa deberá mostrar en su salida lo siguiente:

```

      *
     ***
    *****
   *********
  ***********
 *****

```

El script deberá comprobar que fue llamado con corrección (es decir, que tiene un parámetro y sólo uno, y que éste es un entero mayor que cero). Obviamente, el programa deberá funcionar para cualquier entero positivo (ya que, aunque la salida no cupiese en pantalla, podría redireccionarse sin problemas a un fichero dado).

6. Escribir un guión *shell* llamado `nombremaslargo` que muestre el fichero con el nombre más largo de todos los que se encuentran en los directorios pasados como parámetros, o en alguno de sus subdirectorios. La sintaxis deberá ser la siguiente:

```
nombremaslargo dir1 [dir2] ... [dirN]
```

Ojo, se tendrán en cuenta sólo **ficheros**, y se mostrará el fichero que tenga el nombre más largo, contando sólo su nombre, y no la ruta completa. Eso sí, en la salida deberá mostrarse la ruta completa hasta el fichero. En el caso de que dicho fichero no sea único (es decir, haya varios ficheros con igual longitud, y ésta sea la más larga), el script deberá mostrar en la salida los nombres de todos ellos.

El guión deberá comprobar que ha sido llamado con corrección (con una lista de al menos un directorio, y con ningún parámetro que no sea un directorio).

Ejemplo: si suponemos una estructura de directorios como la siguiente:

```

dir1/
|-- dir11
|   |-- fich2.jpg
|   `-- fichero1.jpg
|-- dir12
|   `-- fichero30.jpg
|-- dir123
|   `-- fichero23.jpg
`-- fichero11.jpg

```

y ejecutamos el comando `nombremaslargo dir1`, la salida debería ser algo así como:

```
$ nombremaslargo dir1
./dir1/dir12/fichero30.jpg
./dir1/dir123/fichero23.jpg
./dir1/fichero11.jpg
```

7. Escribir un guión llamado `ordpal` que reciba como único parámetro un nombre de fichero de texto, y que escriba en la salida estándar, para cada línea de dicho fichero, la lista de palabras de la misma, todas en minúsculas, separadas por espacios, ordenadas alfabéticamente, sin repeticiones, y sin ningún tipo de carácter especial (puntuación, etc.). El guión debe comprobar la corrección de la llamada (esto es, que recibe sólo un parámetro, y que éste es un fichero existente).

Ejemplo:

```
$ cat quijote.txt
```

```
En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivia un hidalgo de los de lanza en astillero, adarga antigua, rocin flaco y galgo corredor. Una olla de algo mas vaca que carnero, salpicon las mas noches, duelos y quebrantos los sabados, lentejas los viernes, algun palomino de anyadidura los domingos, consumian las tres partes de su hacienda.
```

```
$ ./ordpal quijote.txt
```

```
acordarme cuyo de en ha la lugar mancha mucho no nombre quiero un
adarga antigua astillero de en hidalgo lanza los que tiempo un vivia
algo carnero corredor de flaco galgo mas olla que rocin una vaca y
duelos lentejas las los mas noches quebrantos sabados salpicon y
algun anyadidura consumian de domingos las los palomino tres viernes
de hacienda partes su
```

8. Implementar una versión “visual” de `du` (`duv`) que muestre una barra indicando el tamaño relativo de cada fichero en un directorio dado como parámetro. Por ejemplo, si hay dos ficheros en el directorio actual, uno de 100 bytes y otro de 200 bytes, la ejecución `duv .` mostraría:

```
$ duv .
fichero1 ***** 33%
fichero2 ***** 66%
$
```

El tamaño de la barra es proporcional al porcentaje del tamaño del directorio que corresponde al fichero. La barra empezará en la columna 10, se truncarán los nombres de fichero más largos de 9 caracteres y llegará como máximo hasta la columna 70. Al final de la barra se imprimirá el porcentaje del directorio utilizado por el fichero.

Nota: Sólo hay que mostrar los ficheros existentes dentro del directorio que se pasa como parámetro, en el caso de que haya directorios contenidos en el directorio que se pasa como parámetro, éstos no serán mostrados.

9. Construir un script de nombre `bisiesto` que devuelva 0 si el año introducido como parámetro era bisiesto y un -1 en caso contrario. Ejemplo de uso:

```
$ bisiesto 2000
$
```

devolvería un 0 en \$? (en pantalla nunca aparece nada).

Nota: Un año es bisiesto si febrero tiene día 29.

10. Cree un guión shell llamado `capicuasinrev` que devuelva como *código de salida* 0 si el entero positivo dado como parámetro es capicúa y 1 en caso contrario. En la realización del ejercicio está prohibido usar la orden `rev`. Además, es obligatorio que el guión shell compruebe que el número de parámetros y el entero dado son correctos; si no es así, debe devolver 2 como código de salida y un mensaje de error que indique la sintaxis de la orden.

Ejemplo:

```
$ capicuasinrev 3113
$ echo $?
0
$ capicuasinrev 3112
$ echo $?
1
$ capicuasinrev 43,5
Sintaxis: capicuasinrev enteropositivo
$ echo $?
2
$
```

11. Cree un guión shell, llamado `encolapro`, con la siguiente sintaxis:

```
encolapro [-t tiempo] proceso1 proceso2 proceso3 proceso4 ...
```

El guión deberá ejecutar en segundo plano, uno tras otro, todos los procesos indicados según el orden establecido, pero sin que haya dos procesos al mismo tiempo en ejecución.

Para conseguirlo, ejecutará el primer proceso *proceso1* y esperará los minutos indicados por *tiempo*. A continuación comprobará si ese proceso ha terminado. Si ha terminado, ejecutará el siguiente, *proceso2* y volverá a esperar para lanzar *proceso3*. Sin embargo, si aún no ha terminado, esperará de nuevo los minutos indicados, para después realizar otra vez la comprobación. Sólo podrá lanzar el siguiente proceso, cuando el anterior haya terminado, y la comprobación la deberá realizar según los minutos indicados.

El parámetro de tiempo es opcional, y en caso de que no se indique, su valor por defecto es de **60 segundos**.

Pista: Para esperar se puede usar la orden *sleep*.

12. Cree un guión shell llamado `ordporlinlarga`, que reciba como parámetros una lista de directorios, y que muestre como salida la lista de ficheros regulares que cuelgan (recursivamente) de dichos subdirectorios, pero ordenada de mayor a menor por el número de caracteres que posee la línea más larga de cada uno de los ficheros.

Ejemplo: sean los ficheros `f1a.txt`, `f1b.txt`, `f2a.txt` y `f2b.txt`, dentro de los directorios respectivos `d1` (los dos primeros ficheros) y `d2` (para los dos últimos):

```
$ cat d1/f1a.txt
123456789
123
1234567
1

$ cat d1/f1b.txt
1 2 3
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7

$ cat d2/f2a.txt
```

```
abcde
abcdefgh
```

```
$ cat d2/f2b.txt
Esta línea tiene 64 caracteres contando los espacios.
```

Entonces, la ejecución del comando sobre los directorios d1 y d2 debería producir la siguiente salida:

```
$ ./ordporlinlarga d1 d2
./d2/f2b.txt
./d1/f1b.txt
./d1/f1a.txt
./d2/f2a.txt
```

Puesto que la línea más larga de f2b.txt tiene 64 caracteres, la de f1b.txt 17, la de f1a.txt 9, y la de f2a.txt 8.

13. Cree un guión shell, llamado psancestros, con la siguiente sintaxis:

```
psancestros PID
```

donde PID es un identificador de proceso. El guión debe comprobar que el parámetro PID existe, que es un número, y que corresponde a un proceso existente. A continuación, tiene que imprimir la lista de PIDs desde el PID dado hasta el PID 0 (proceso raíz). Para ello, necesita identificar el proceso padre de cada proceso, es decir, su PPID. Por ejemplo:

```
$ ps
PID   TTY    TIME    CMD
5255  pts/0  00:00:01 bash
12527 pts/0  00:00:00 ps
```

```
$ ./psancestros 5255
PID 5255-->5253-->1-->0
```

Nota: La opción -l de ps permite identificar el PPID de un proceso.

14. Implementar un guión de bash (totalvsz) que imprima en su salida estándar (por defecto, la pantalla) un listado de todos los usuarios que se encuentran ejecutando algún proceso en el sistema, junto con un número indicando el tamaño total de memoria virtual ocupado por todos los procesos de dicho usuario.

Ejemplo: si la salida de un comando ps aux fuese, por ejemplo:

```
$ ps aux

USER      PID [...]  VSZ   [...]
root       1 [...]   2912  [...]
root       2 [...]     0    [...]
root      123 [...]  1232  [...]
daemon    5220 [...]   420   [...]
root      5234 [...]   100   [...]
pedro     5387 [...]  1716  [...]
juan      5389 [...] 15020  [...]
```

```
antonio  5390 [...] 12721  [...]
juan     5392 [...]  8009   [...]
antonio  5399 [...]   125    [...]
pedro    6001 [...]   10     [...]
```

Entonces una llamada `totalvsz` a nuestro programa debería mostrar en su salida lo siguiente:

```
$ totalvsz

USER      TOTALVSZ
root      4244
daemon    420
pedro     1726
juan      23029
antonio   12846
```

15. Cree un guión shell llamado `buscapalabras` con la siguiente sintaxis:

```
buscapalabras fichero depalabras fichero debusqueda
```

Donde `fichero depalabras` es el nombre de un fichero que contiene una lista de palabras, una por línea, y `fichero debusqueda` será un fichero de texto donde se buscarán las palabras indicadas en `fichero depalabras`. Como resultado se mostrará, por cada palabra, el número de líneas donde aparece la palabra en el fichero indicado. El resultado deberá estar ordenado de forma creciente por el número de apariciones de las palabras.

Ejemplo:

```
$ cat palabras.txt
Mancha
de
lugar
DE
$ cat citaQuijote.txt
"En un lugar de la Mancha,
de cuyo nombre no quiero acordarme,
no ha mucho vivia un hidalgo de los
de lanza en astillero, ..."
$ buscapalabras palabras.txt citaQuijote.txt
Mancha 1
lugar 1
de 4
$
```

Nota: A la hora de procesar las palabras del fichero `fichero depalabras` no se distinguirá entre mayúsculas y minúsculas y la misma palabra en distinta líneas se considerará una única palabra.

16. Cree un guión shell, llamado `quiniela`, que nos genere de forma aleatoria una quiniela simple de 1 apuesta, es decir un resultado para cada partido.

Cuando ejecute el script:

```
quiniela
```

El resultado debe ser similar al siguiente:

```
1.- 1
2.- X
3.- X
...
14.- 2
```

Observe la alineación de los números y que los todos los resultados del mismo tipo están en la misma columna.

Nota: La variable de entorno \$RANDOM de bash puede utilizarse para generar un número aleatorio entero entre 0 y 32767, distinto cada vez que es invocada.