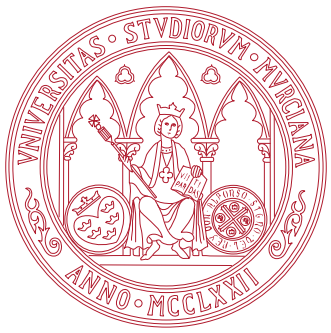

Universidad de Murcia

Facultad de Informática

Departamento de Ingeniería y Tecnología de Computadores

Área de Arquitectura y Tecnología de Computadores

Prácticas de

Introducción a los Sistemas Operativos

2º de Grado en Ingeniería Informática

Relación de Ejercicios Propuestos – Programación de guiones *shell* en Linux

Curso 2021/2022

1. Codifica un guion *shell* de nombre `bisiesto.sh` que devuelva el código de salida 0 si el año introducido como parámetro es bisiesto y 1 en caso contrario. Ejemplos de uso:

```
[alumno@localhost ~]$ bash bisiesto.sh 2016
[alumno@localhost ~]$ echo $?
0
[alumno@localhost ~]$ bash bisiesto.sh 2013
[alumno@localhost ~]$ echo $?
1
```

   NOTA: puedes usar la orden `cal` para averiguar si el año pasado como parámetro es bisiesto o no (febrero tiene día 29).

2. Implementa un guion *shell* llamado `monton.sh` 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

```
[alumno@localhost ~]$ bash monton.sh 5
```

   el guion deberá mostrar en su salida lo siguiente:

```
    *
   ***
  *****
 *******
*********
```

   El guion deberá comprobar que fue llamado con corrección (es decir, que tiene un parámetro y solo 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).

3. Crea un guion *shell* llamado `aleatorio.sh` que genere un número natural aleatorio entre el 1 y el 100 con las siguientes características:

   - Si se llama **sin** argumentos, entonces se podrá generar **cualquier** número en el rango indicado (1 al 100)
   - Si se llama **con** argumentos (pueden ser tantos como desee el usuario), entonces el número generado no podrá ser ninguno de los especificados por el usuario (estará en el rango 1 a 100 pero será distinto de los pasados como argumentos)

   El guion debe funcionar aún cuando el usuario haya introducido algún argumento incorrecto (es decir, algo que no sea un número natural). En ese caso, se saltará dicho argumento y se continuará con el resto.

   Al final el guion imprimirá por pantalla "Número aleatorio entre 1 y 100 (eliminando los argumentos) : 45"

   Ejemplos de uso:

```
[alumno@localhost ~]$ bash aleatorio.sh 10 20 30 40 50 60 70 80 90 100
Número aleatorio entre 1 y 100 (eliminando los argumentos) : 23
[alumno@localhost ~]$ bash aleatorio.sh 10 20 30 ffff 50 60 70 80 90 100
Número aleatorio entre 1 y 100 (eliminando los argumentos) : 43
```

   NOTA: Para obtener números aleatorios puedes leer el valor de la variable $RANDOM

4. Implementa un guion *shell* llamado `capicuasinrev.sh` 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 guion *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. Ejemplos de uso:

```
[alumno@localhost ~]$ bash capicuasinrev.sh 3113
[alumno@localhost ~]$ echo $?
0
[alumno@localhost ~]$ bash capicuasinrev.sh 3112
[alumno@localhost ~]$ echo $?
1
[alumno@localhost ~]$ bash capicuasinrev.sh 43,5
Sintaxis: capicuasinrev.sh enteropositivo
[alumno@localhost ~]$ echo $?
2
```

5. Implementa el guion *shell* `totalvsz.sh` que imprime en su salida estándar (por defecto, la pantalla) un listado de todos los usuarios que disponen de algún proceso en el sistema, junto con un número indicando el tamaño total de memoria virtual ocupado por todos los procesos de dicho usuario (en KiB). Para obtener la lista de procesos en el sistema se ha de ejecutar el comando `ps aux`. Concretamente, la primera columna (USER) del resultado de la ejecución de dicho comando indica el usuario propietario del proceso, mientras que la quinta (VSZ) nos informa acerca de la memoria virtual ocupada por el proceso (en KiB). Ejemplo: si la salida del comando `ps aux` es:

```
[alumno@localhost ~]$ 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   [...]
pepito    5390 [...]  12721   [...]
juan      5392 [...]   8009   [...]
pepito    5399 [...]    125   [...]
pedro     6001 [...]     10   [...]
```

entonces la ejecución de `totalvsz.sh` debería mostrar en su salida lo siguiente:

```
[alumno@localhost ~]$ bash totalvsz.sh

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

6. Implementa un guion *shell* llamado `dibujatexto.sh` que, dado un fichero describiendo un texto, dibuje dicho texto en pantalla. Para describir el texto, el fichero tendrá un número de

líneas arbitrario y en cada línea habrá una secuencia de números (de 1 a 9) y caracteres, donde cada número irá seguido por un carácter. El número indicará cuántas veces se tiene que repetir el carácter que le sigue. Así, si tenemos el fichero `letraA.txt` con el siguiente contenido:

```
3*
1*1 1*
3*
1*1 1*
1*1 1*
```

la siguiente ejecución del guion *shell* producirá el resultado mostrado:

```
$ bash dibujatexto.sh letraA.txt
***
* *
***
* *
* *
```

El guion *shell* debe comprobar que el número de argumentos es 1 y que puede leer el fichero dado como argumento. Si no se cumple alguna de estas condiciones, debe mostrar un mensaje de error y un código de error adecuados. A continuación se muestran algunas ejecuciones con error:

```
$ bash dibujatexto.sh
Uso: dibujatexto.sh <fichero>
$ echo $?
1
$ bash dibujatexto.sh letraB.txt
Imposible acceder al fichero letraB.txt
$ echo $?
2
```

NOTA: Ten cuidado con los asteriscos del fichero de entrada ya que, si no se protegen adecuadamente, pueden dar problemas dentro del guion.

7. Implementa un guion *shell* llamado `diracum.sh` que, dado un directorio como parámetro, muestre los ficheros regulares del mismo ordenados por tamaño. Para cada fichero deben aparecer sus permisos, su tamaño, el tamaño acumulado (es decir, la suma de los tamaños de los ficheros mostrados hasta ahora, incluyendo el actual) y su nombre. Al final del listado se incluirá el número total de ficheros y el tamaño medio. Por ejemplo, si tenemos el siguiente directorio:

```
[alumno@localhost ~]$ ls -l Latexdoc/
total 7536
-rw-r--r--. 1 alumno alumno   14947 mar 25  2014 beamer_rk.sty
-rw-rw-r--. 1 alumno alumno  330873 oct 26  2011 enumitem.pdf
-rw-rw-r--. 1 alumno alumno  188619 ene 17  2012 geometry.pdf
drwxrwxr-x. 2 alumno alumno    4096 oct 18 11:37 guias
-rw-rw-r--. 1 alumno alumno  198080 oct 26  2011 labels.pdf
drwxrwxr-x. 2 alumno alumno    4096 oct 18 11:37 plantillas
-rw-rw-r--. 1 alumno alumno  641692 nov 15  2011 subfig.pdf
-rw-rw-r--. 1 alumno alumno  436896 nov 25  2011 tocloft.pdf
```

entonces, la siguiente ejecución del guion *shell* produce el resultado mostrado:

```
[alumno@localhost ~]$ bash diracum.sh Latexdoc
PERMISOS TAMAÑO ACUMULADO NOMBRE
-rw-r--r-- 14947 14947 beamer_rk.sty
-rw-rw-r-- 188619 203566 geometry.pdf
-rw-rw-r-- 198080 401646 labels.pdf
-rw-rw-r-- 330873 732519 enumitem.pdf
-rw-rw-r-- 436896 1169415 tocloft.pdf
-rw-rw-r-- 641692 1811107 subfig.pdf
Total de ficheros: 6 - Tamaño medio: 301851 bytes
```

El guion *shell* debe comprobar que el número de argumentos es 1 y que el argumento dado es un directorio. Si no se cumple alguna de estas condiciones, debe mostrar un mensaje de error y un código de error adecuados. A continuación se muestran algunas ejecuciones con error (observe los mensajes y códigos de error devueltos):

```
[alumno@localhost ~]$ bash diracum.sh
Uso: diracum.sh <directorio>
[alumno@localhost ~]$ echo $?
1
[alumno@localhost ~]$ bash diracum.sh noexiste
noexiste no es un directorio
[alumno@localhost ~]$ echo $?
2
```

8. Escribe un guion *shell* llamado `nombremaslargo.sh` 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.sh dir1 [dir2] ... [dirN]
   ```

   Ojo, se tendrán en cuenta solo **ficheros**, y se mostrará el fichero que tenga el nombre más largo, contando únicamente su nombre (para facilitar la resolución, puedes suponer que los nombres de fichero no contienen espacios), 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), se deberá mostrar en la salida los nombres de todos ellos.

   El guion 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 `bash nombremaslargo.sh dir1`, la salida debería ser algo así como:

```
[alumno@localhost ~]$ bash nombremaslargo.sh dir1
./dir1/dir12/fichero30.jpg
./dir1/dir123/fichero23.jpg
./dir1/fichero11.jpg
```

9. Se quiere implementar un guion *shell* llamado `tamarelativo.sh` 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 692 bytes («fichero1») y otro de 896 bytes («largolargolargo»), la ejecución `tamarelativo.sh .` mostraría:

```
[alumno@localhost ~]$ bash tamarelativo.sh .
Total : 1588 bytes
fichero1  ************************* 43%
largolarg ******************************** 56%
```

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 que el tamaño de cada fichero representa con respecto al tamaño total de los ficheros del directorio. Solo 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.

Implementa el guion *shell* resolviendo los siguiente apartados:

a) Vamos a calcular primero la suma de los tamaños de todos los ficheros que hay en el directorio que nos pasan como parámetro.

```
[alumno@localhost ~]$ bash tamarelativo.sh .
Total : 1588 bytes
```

b) A continuación vamos a mostrar un listado de los ficheros, indicando para cada uno de ellos su tamaño en bytes. Los nombres de los ficheros se mostrarán con hasta 9 caracteres (como se ha explicado antes), y el tamaño se mostrará a partir de la columna 10.

```
[alumno@localhost ~]$ bash tamarelativo.sh .
Total : 1588 bytes
fichero1  692 bytes
largolarg 896 bytes
```

c) En lugar del tamaño en bytes, vamos a mostrar el mismo listado anterior pero incluyendo el porcentaje que el tamaño de cada fichero representa con respecto al tamaño total de los ficheros del directorio.

```
[alumno@localhost ~]$ bash tamarelativo.sh .
Total : 1588 bytes
fichero1  43%
largolarg 56%
```

d) Vamos a incluir por último las barras proporcionales a los porcentajes calculados, tal y como se ha explicado.

10. Se pretende diseñar un guion *shell* llamado `permisosraros.sh` que reciba como parámetros uno o más directorios y devuelva un listado de los ficheros de esos directorios (y sus subdirectorios), que contenga aquellos con permisos que podemos considerar como "raros". En particular, entendemos que un fichero tiene permisos "raros" cuando el usuario propietario del mismo no es "root" y además se cumple alguna de las siguientes dos condiciones: que el propietario no tenga

permiso de lectura, o que no tenga permiso de ejecución cuando alguien del resto (grupo y otros) sí lo tiene.

Implementa el guion *shell* resolviendo los siguiente apartados:

a) Escribe una sola orden que devuelva un listado de los ficheros del directorio actual y subdirectorios de este, que contenga aquellos con permisos "raros". En el listado debe aparecer el nombre del fichero (con ruta) y los permisos (como se hace en ls -l). Ejemplo de listado:

```
[alumno@localhost ~]$ orden
./dir1/b -rw-rwxr-x
./dir1/a --w--w----
./dir1/dir2/e -rw-rwxr--
./dir3/h -rw-rwxr--
```

b) Crea un guion llamado *permisosraros.sh* que reciba como parámetros uno o más directorios y devuelva el listado del apartado anterior, pero en este caso la salida tendrá una lista de usuarios propietarios de los ficheros ordenada alfabéticamente, junto con los ficheros de cada uno (y los permisos). No deberán aparecer aquellos usuarios que no tengan ningún fichero que cumpla la condición. Ejemplo:

```
[alumno@localhost ~]$ bash permisosraros.sh dir1 /home/alumno/dir3
alumno:
        ./dir1/b -rw-rwxr-x
        ./dir1/a --w--w----
usuar1:
        ./dir1/dir2/e -rw-rwxr--
        ./dir3/h -rw-rwxr--
```

c) Amplía el guion del apartado anterior para en lugar de mostrar los nombres de los ficheros precedidos de la ruta, ahora tras cada uno de los nombres de usuarios propietarios vendrá la lista de los directorios en los que hay algún fichero ordenados también alfabéticamente, y tras esto, la lista de los nombres de los ficheros detectados junto con los permisos. Ejemplo:

```
[alumno@localhost ~]$ bash permisosraros.sh dir1 /home/alumno/dir3
alumno:
    ./dir1:
        --w--w----    a
        -rw-rwxr-x    b
usuar1:
    ./dir1/dir2:
        -rw-rwxr--    e
    /home/alumno/dir3:
        -rw-rwxr--    h
```

d) Vamos a considerar ahora el chequeo de los parámetros. Amplía el guion del apartado anterior para que compruebe que todos los argumentos pasados son directorios válidos. Si no lo son, habrá que mostrar un mensaje de uso por la salida estándar de error y devolver el código 1. Si no se pasa ningún argumento, se tomará como directorio el actual.

Todos los ejemplos asumen el siguiente árbol de directorios:

```
/home/alumno/dir1:
    --w--w---- 1 alumno alumno   10 may 27 21:26 a
    -rw-rwxr-x 1 alumno alumno   20 may 27 21:26 b
    -rw-rw-r-- 1 usuar1 usuar1   15 may 27 21:26 c
    -rwxrwxr-x 1 alumno alumno    7 may 27 21:33 d
    drwxrwxr-x 2 alumno alumno 4096 may 28 10:13 dir2
```

```
/home/alumno/dir1/dir2:
    -rw-rwxr-- 1 usuar1 usuar1  8 may 28 10:14 e
    -rwxrw-r-- 1 usuar1 usuar1 11 may 28 10:14 f
    -r-xr-xr-x 1 alumno alumno 17 may 28 10:14 g
/home/alumno/dir3:
    -rw-rwxr-- 1 usuar1 usuar1 3 may 28 10:14 h
    -rwxrw-r-- 1 alumno alumno 4 may 28 10:14 i
    -r-xr-xr-x 1 alumno alumno 5 may 28 10:14 j
```

11. Se pretende diseñar un guion *shell* llamado `palabrasfrecuentes.sh` que, dado un fichero de texto y un número n, devuelva las n palabras diferentes más frecuentes que haya en dicho fichero, junto con el porcentaje de veces que cada palabra aparece en el fichero. No se distinguirá entre mayúsculas y minúsculas.

    Para probar el funcionamiento del guion *shell*, puedes usar el siguiente fichero de texto:

    ```
    [alumno@localhost ~]$ cat elquijote.txt
    01 En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho
    02 tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua,
    03 rocín flaco y galgo corredor.
    ```

    Implementa este guion *shell* resolviendo los siguiente apartados:

    a) El guion *shell* debe comprobar el número de parámetros, el permiso de lectura del fichero y si el segundo parámetro es realmente un número entero positivo. En caso de error, debe mostrar un mensaje y devolver un código de retorno apropiados. A continuación se muestran algunas ejecuciones incorrectas del guion *shell*:

    ```
    [alumno@localhost ~]$ bash palabrasfrecuentes.a.sh noexiste
    Uso: palabrasfrecuentes.a.sh fichero número
    [alumno@localhost ~]$ echo $?
    1
    [alumno@localhost ~]$ bash palabrasfrecuentes.a.sh noexiste 10
    El fichero noexiste no existe o no se puede leer
    [alumno@localhost ~]$ echo $?
    2
    [alumno@localhost ~]$ bash palabrasfrecuentes.a.sh elquijote.txt a
    a no es un número o no es mayor que 0
    [alumno@localhost ~]$ echo $?
    3
    ```

    b) Amplía el guion *shell* para que muestre el total de palabras que contiene el fichero dado. Se entiende por palabra cualquier cadena de caracteres alfabéticos separada de otras palabras por cualquier signo de puntuación o espacio (ya sea un espacio en blanco, un tabulador, un retorno de carro, etc.). Las cadenas que contengan algún número no se considerarán palabras.

    A continuación se muestran el resultado de la ejecución del guion *shell* (observa que el segundo parámetro se ignora por ahora):

    ```
    [alumno@localhost ~]$ bash palabrasfrecuentes.b.sh elquijote.txt 5
    33
    ```

    c) Amplía de nuevo el guion *shell* para que muestre también el número de veces que cada palabra aparece en el fichero dado. La lista debe aparecer ordenada por número de repeticiones. Recordemos que tampoco se debe distinguir entre mayúsculas y minúsculas.

    A continuación se muestra el resultado de la ejecución de esta nueva versión del guion *shell* (observa que el segundo parámetro se ignora de nuevo):

```
[alumno@localhost ~]$ bash palabrasfrecuentes.c.sh elquijote.txt 5
33
      4 de
      2 un
      2 no
      2 en
      1 y
    ... ...
```

d) Finalmente, modifica y amplía el guion *shell* para que solo se quede con las n palabras más frecuentes, siendo n el segundo parámetro del guion, y, para cada una de ellas, muestre el porcentaje de veces que aparece en el fichero dado. A continuación se muestra una ejecución de la versión final del guion *shell*:

```
[alumno@localhost ~]$ bash palabrasfrecuentes.d.sh elquijote.txt 5
de: 4 (12%)
un: 2 (6%)
no: 2 (6%)
en: 2 (6%)
y: 1 (3%)
```

12. Se pretende diseñar un guion *shell* llamado `usodiscoporusuario.sh` que, dada una lista de directorios, calcule el tamaño total de todos los ficheros (de cualquier tipo) que se encuentren a partir de ellos de forma recursiva y muestre además, para cada usuario propietario de ficheros, el tamaño total de todos sus ficheros. Si se da el caso, el guion *shell* también debe mostrar al final una lista de todos los ficheros y directorios que no se han podido contabilizar por falta de permisos.

    No es necesario que el guion *shell* realice ninguna comprobación de parámetros.

    Resuelve los siguientes apartados para implementar este guion *shell*:

    a) Haz que el guion *shell* devuelva una lista de todos los usuarios con ficheros en los directorios dados como argumentos. Suprima cualquier mensaje de error que pueda aparecer. Lo siguiente es una ejecución del guion *shell* donde se muestra el tipo de salida que debe producir:

    ```
    [alumno@localhost ~]$ bash usodiscoporusuario.a.sh /etc /usr/bin /mnt
    inadyn
    mail
    polkitd
    root
    tss
    ```

    b) Modifica el guion *shell* para que muestre el tamaño total, en bytes, de todos los ficheros de cada uno de los usuarios obtenidos anteriormente. Al igual que antes, se debe suprimir cualquier mensaje de error que se pueda producir. El resultado que se debe obtener ahora debe parecerse a lo siguiente:

    ```
    [alumno@localhost ~]$ bash usodiscoporusuario.b.sh /etc /usr/bin /mnt
    inadyn: 136
    mail: 93
    polkitd: 4096
    root: 915098478
    tss: 7046
    ```

c) Modifique de nuevo el guion *shell* para que se muestre también el tamaño total de todos los ficheros de todos los usuarios. Además, haga que los tamaños inferiores a un mebibyte (1 MiB) se den en bytes, y los que sean mayores o iguales se den en mebibytes (MiB), indicando, en cada caso, la unidad utilizada. Lo siguiente es un ejemplo de ejecución de esta nueva versión del guion *shell*:

```
[alumno@localhost ~]$ bash usodiscoporusuario.c.sh /etc /usr/bin /mnt
inadyn: 136 bytes
mail: 93 bytes
polkitd: 4096 bytes
root: 872 MiB
tss: 7046 bytes
Total de todos los usuarios: 872 MiB
```

d) Finalmente, modifique el guion *shell* para mostrar al final una lista de todos los ficheros y directorios que no se han podido contabilizar por falta de permisos para acceder a ellos. Si no se da ningún caso, el guion *shell* no mostrará nada. A continuación se muestran un par de ejecuciones de esta última versión del guion *shell*:

```
[alumno@localhost ~]$ bash usodiscoporusuario.d.sh /etc /usr/bin
inadyn: 136 bytes
mail: 93 bytes
polkitd: 4096 bytes
root: 872 MiB
tss: 7046 bytes
Total de todos los usuarios: 872 MiB
El contenido de los siguientes directorios no se han podido contabilizar:
 ‘/etc/audit’
 ‘/etc/ntp/crypto’
 ‘/etc/cups/ssl’
 ....
[alumno@localhost ~]$ bash usodiscoporusuario.d.sh /mnt
root: 8192 bytes
Total de todos los usuarios: 8192 bytes
```