3. Prácticas de scripts

PRECAUCIÓN: Los ejercicios se someterán a pruebas automatizadas de detección de plagios, cruzando tanto los presentados este curso como los previos.

Entrega: Los ficheros de script deberán denominarse: `ej1.sh`, `ej2.sh`, `ej3.py`, `ej4.sh` y `ej5.py`, se empaquetarán en un `.tar.gz` sin subdirectorios, y este se entregará en la tarea de la USC virtual.

Los scripts pueden programarse y ejecutarse en cualquier ordenador. En la USC virtual se proporcionan ficheros para hacer pruebas y comprobar los resultados. Puede utilizarse el comando `diff` para ver diferencias entre dos archivos.

En los scripts de python, la versión que se usará será la `3.x`, por lo que la primera línea del script ha de contener: `#!/usr/bin/env python3`

3.1. Renombrar ficheros con sed (1 punto)

Disponemos de una serie de ficheros en el formato `tituloXautor.epub` donde `X` es un número de uno o mas dígitos. Elaborar un comando basado en `sed` para cambiar los nombres de los ficheros al formato `autor-titulo-X.epub`. El comando tomará la forma:

```
ls *.epub | ./ej1.sh | bash
```

Donde la sentencia sed (`./ej1.sh`) ha de realizar la siguiente sustitución:

`tituloXautor.epub` → `mv tituloXautor.epub autor-titulo-X.epub`

Probar el comando con `ebooks.tar.gz` (proporcionado en el material de la práctica) y comprobar que el resultado de la ejecución es `renombrados.tar.gz`.

3.2. Procesamiento de textos (1 punto)

Escribe un script de procesamiento de textos que tome como entrada dos ficheros y proporcione como salida las líneas que son diferentes de los dos archivos, indicando la línea. Por ejemplo:

```
$ cat fichero1.txt
main() {
for(i=0;i<10;i++) a+=1;
p=p+2;
s=s-2; }
```

```
$ cat fichero2.txt
main() {
for(i=0;i<11;i++) b+=1;
p=p+2;
s=s+2; }
```

```
$ ./ej2.sh fichero1.txt
* Las lineas 2 difieren
<- for(i=0;i<10;i++) a+=1;
-> for(i=0;i<11;i++) b+=1;
```

```
* Las lineas 4 difieren
<- s=s-2; }
-> s=s+2; }
```

Requisitos del script:

1. Usar solo comandos de procesamiento de texto (*sed, awk, sort, grep, tr*, etc)
2. No utilizar lazos.
3. Utilizar únicamente tuberías para pasar los resultados parciales entre los comandos.

AYUDA: En los ficheros no va a aparecer el símbolo $, por lo que este podrá utilizarse como separador.

3.3. Etiquetado de regiones en python (2 puntos)

Escribe un script en `python` que obtenga las regiones conectadas de una matriz de ceros y unos (`https://en.wikipedia.org/wiki/Connected-component_labeling`), considerando conectividad 4 (arriba, abajo, izquierda y derecha, sin tener en cuenta las diagonales) y con etiquetas de un dígito consecutivas (1, 2, 3, ...). Por ejemplo:

```
$ cat fichero.txt
0100011100000000
0111011000011100
0011101111000100
0000100000001100
0000001111000000
0000001100000000
```

```
$ ./ej3.py fichero.txt
0100022200000000
0111022000033300
0011102222000300
0000100000003300
0000004444000000
0000004400000000
```

Requerimientos:

- Estructurar el programa con al menos una función.
- El script tendrá como argumento un fichero y mostrará en pantalla la matriz etiquetada.
- Utilizar `argparse` para gestionar el argumento (que será posicional).

AYUDA: un algoritmo simple comienza por asignar etiquetas (números naturales) a cada 1 de la matriz y después en una serie de pasadas se detecta cada celda diferente de cero, se comprueba su vecino de la derecha y si es diferente de cero se asigna a ambos la etiqueta más pequeña, luego se hace lo mismo con el vecino de abajo. El proceso se repite hasta que no se produzcan cambios. Para finalizar, las etiquetas se hacen consecutivas. Para ello lo más fácil es hacer una

tabla de conversión que indique para cada índice si la etiqueta es usada o no y a continuación numerar las usadas secuencialmente. Finalmente se recorre la matriz y se reetiqueta de acuerdo a la tabla de conversión.

Por ejemplo (los pasos no son exhaustivos, sino que se han elegido para ser ilustrativos):

```
$ paso 1  $ paso 2  $ paso 3  $ paso 4  $ paso 5  $ paso 6
01000     01000     01000     01000     01000     01000
11101     23405     31405     11105     11105     11102
11101     67809     66805     66605     11105     11102
```

3.4. Procesamiento de smaps en bash (3 puntos)

El fichero `/proc/$pid/smaps` tiene para el proceso de identificador `pid` información sobre los segmentos de memoria que está usando ese proceso, indicando para cada segmento los tamaños de distintos tipos de memoria: *memoria total* (Size o virtual set size), *memoria residente* (Rss o resident set size), *memoria residente proporcional* (Pss o proportional set size), *memoria compartida* (limpia y usada) y *memoria privada* (limpia y usada).

Crear un script de bash para analizar esta información y obtener, para un proceso con un `pid` concreto:

`Memoria total` (sumando los kB de las líneas que empiezan con Size)
`Memoria residente` (sumando las líneas Rss)
`Memoria proporcional` (sumando las líneas Pss)
`Memoria privada` (suma de todas las lineas Private_Clean y Private_Dirty)
`Memoria compartida` (suma de todas las lineas Shared_Clean y Shared_Dirty)
`Privada + compartida` (suma de las dos anteriores)

Ejemplo de uso:

```
$ ./ej5.sh 10219
  Memoria total:        27452 kB
  Memoria residente:     5972 kB
  Memoria proporcional:  4430 kB
  Memoria privada:       4236 kB
  Memoria compartida:    1736 kB
  Privada + compartida:  5972 kB
```

Requisitos del script:

1. Debe aceptar como parámetro un número. Si no se le pasa ningún parámetro debe indicarlo con un mensaje de uso:

```
$ ./ej5.sh
  Uso: checkmaps.sh pid
```

2. Debe comprobar que el parámetro pasado corresponde al PID de un proceso activo. Además, si el usuario que está ejecutando el script no es root, debe comprobar que el PID indicado es de un proceso de ese usuario. Usar el comando `ps` para estas comprobaciones

   Ejemplo:

   ```
   $ ./ej5.sh 2345678
      El proceso 2345678 no existe.
   $ ./ej5.sh 592
      No eres root y no tienes permisos para acceder al proceso 592.
   ```

3. Utilizar funciones para estructurar el script. Como mínimo, la parte de comprobación del parámetro debe estar en una función separada.

4. El script debe estar debidamente comentado.

5. Se valorará la eficiencia del script en tiempo de ejecución.

Se proporciona un programa C compilado para hacer comprobaciones, `eje5.out`. Ha de tenerse en cuenta que los tamaños de memoria cambian dinámicamente, en concreto, la memoria proporcional es muy variable.

3.5. Desglose de funciones en python (3 puntos)

Escribe un script en python que extraiga las funciones *void* y *float* de un programa C y las guarde en ficheros separados, colocando un *include* en el fichero original. Las funciones *void* se guardarán en el subdirectorio `void` y las *float* en el subdirectorio `float`. El nombre del fichero será el mismo que el nombre de la función con la extension `.c`. Por ejemplo:

```
$ cat fichero1.c
#include <stdio.h>

float suma(float a, float b)
{  float c=a+b;
   return(c); }

void imprime(int i)
{  printf("Numero: %f\n",i); }
```

```
int main()
{  float a=5, b=6, c;
   c=suma(a,b);
   imprime(c);
}
```

```
$ ./ej5.py -i fichero1.c -o salida1.c
```

```
$ cat salida1.c
#include <stdio.h>

#include "float/suma.c"

#include "void/imprime.c"

int main()
{  float a=5, b=6, c;
   c=suma(a,b);
   imprime(c);
}
```

```
$ cat float/suma.c
float suma(float a, float b)
{  float c=a+b;
   return(c); }

$ cat void/imprime.c
void imprime(int i)
{  printf("Numero: %f\n",i); }
```

Para facilitar la conversión, el fichero solo contendrá líneas de 4 tipos: includes, nombre de la función (*void*, *int* o *float*, sin punteros), cuerpo de la función y vacías (solo espacios o tabs). El nombre de la función estará contenido en una línea y no se mezclará con el cuerpo de la función. Solo se extraerán las funciones de tipo *void* y *float*, las de tipo *int* no se extraerán. Los parámetros se gestionarán mediante `argparse` y serán los siguientes:

- `-i` *fichero.c*, el nombre del fichero de entrada.
- `-o` *salida.c*, el nombre del fichero de salida.

Requisitos del script:

1. Usar funciones para estructurar el script, que deberá estar debidamente comentado.
2. En caso de que haya parámetros incorrectos o falte alguno, se le presentará la ayuda al usuario y acabará.
3. Si el fichero de una función ya existe, no se realizará la sustitución.
4. Se generará una excepción si los ficheros no pueden leerse o escribirse.

AYUDA: El programa comenzará clasificando las líneas mediante expresiones regulares en los tipos: *include*, nombre de función *void*, *int* o *float* e instrucciones. El cuerpo de una función serán las instrucciones comprendidas entre nombres de funciones o includes.