Práctica 3: **Comandos habituales y útiles**

Marzo 2017

Índice

1. Introducción

La práctica 3 se centra en aprender a utilizar los comandos más comunes de la línea de comandos de la shell. En la práctica 1 hemos aprendido a programar bucles, condiciones if-then-else, etc. así como algunos comandos de manipulación de archivos como el `ls` o el `chmod`. En esta práctica vamos a analizar comandos más avanzados que permiten también manipular archivos. También veremos como se puede utilizar la redirección y las tuberías (en inglés, pipe) para combinar comandos entre sí.

Es importante que copiéis las instrucciones a mano. Evitad hacer un "copy+paste" puesto que puede producir que salgan por pantalla errores raros.

2. Redirección y tuberías

Hemos visto en clase de teoría que cuando se crea un proceso se abren por defecto tres ficheros: la entrada estándar que proviene de teclado (y que se puede capturar, por ejemplo, con la función `scanf`), la salida estándar que va dirigida a pantalla (podemos dirigir mensajes a la salida estándar con la función `printf`, por ejemplo), y la salida estándar de error (podemos dirigir mensajes a esta salida mediante la función `fprintf`, por ejemplo).

La redirección es una técnica mediante la cual se puede redirigir la salida estándar a un fichero o el contenido de un fichero a la entrada estándar. En clase de teoría de la asignatura se ha visto cómo se programan en C y los principios básicos de utilización desde la línea de comandos. Aquí repasaremos su uso desde el terminal y veremos a lo largo de este documento su uso en combinación con otros comandos.

Ejecutemos el siguiente comando desde el terminal:

```
$ ls
```

El comando anterior imprime por pantalla el contenido del directorio en el que estamos situado. Ejecutemos a continuación el siguiente comando:

```
$ ls > ls-out.txt
```

Mediante este último comando se vuelca el contenido del directorio actual en el fichero `ls-out.txt`. Visualizar el contenido del fichero `ls-out.txt` para comprobar que ha sido así. En caso que el fichero `ls-out.txt` exista previamente, se trunca el fichero a longitud cero. Mediante el siguiente comando se añade la salida estándar al fichero de salida a continuación de las lineas que este pueda tener (en caso que el fichero no exista, se crea)

```
$ ls >> ls-out.txt
```

La técnica de redirección de la salida estándar funciona para cualquier aplicación que imprima mensajes por la salida estándar (por defecto, la pantalla). Tomemos el código `imprime-mensajes.c` situado en el directorio `redireccion`:

```c
#include <stdio.h>

int main(void)
{
  int i;

  for(i = 1; i < 10; i++)
    printf("Esto es la linea %d\n", i);

  printf("Hemos acabado\n");

  return 0;
}
```

Ejercicio 1. Compilad y ejecutad el comando con y sin redirección. Observad que con la redirección se vuelcan todos los mensajes que se imprimen por la salida estándar en un fichero.

Se pueden redirigir también los mensajes que se imprimen por la salida de error. Con los ejemplos anteriores hemos visto como redirigir la salida estándar a un fichero. La salida estándar tiene asociado el descriptor de fichero 1, mientras que la salida de error tiene asociado el descriptor de fichero 2. Para redirigir la salida de error a un fichero se puede utilizar "`2>`" o "`2>>`". Típicamente los mensajes de error de los comandos se imprimen por la salida de error.

```
$ ls noexiste.txt
$ ls noexiste.txt 2> error.txt
```

De forma similar, podemos redirigir el contenido de un fichero a la entrada estándar. Tomemos como base el siguiente código

```c
#include <stdio.h>
#include <string.h>

int main(void)
{
    int i;
    char str[200];

    i = 0;
    while (scanf("%s", str) > 0) {
        i++;
    }

    printf("Numero total de palabras: %d\n", i);

    return 0;
}
```

Ejercicio 2. Ejecutad el código anterior y probad de poner una sola palabra en cada linea. Para terminar de introducir las palabras pulsar `<Control+D>`. Esto hace que se envíe un "End-Of-File" (EOF) a la función `scanf`, de forma que éste devuelva un EOF y, por lo tanto, se finalice el bucle. Probar también de introducir varias palabras en una única línea y pulsar `<Control+D>`. El resultado es acorde con lo que esperáis de la funcionalidad de `scanf`?

A continuación ejecutaremos la siguiente instrucción

```
$ ./lee-palabras < ls-out.txt
```

Donde el fichero `ls-out.txt` es el fichero obtenido anteriormente.

Podemos redirigir tanto la entrada como la salida estándar

```
$ ./lee-palabras < ls-out.txt > salida.txt
```

Qué es lo que debería contener el fichero `salida.txt`? Compruébalo!

La tubería (en inglés, pipe) es una de las formas más básicas de comunicación interproceso. Permite redirigir la salida estándar de un proceso a la entrada estándar de otro proceso. En un terminal se utiliza el carácter '`|`' para especificar una tubería entre dos procesos. Los dos procesos a intercomunicar están indicados a la izquierda y derecha, respectivamente, de este carácter. Un ejemplo sencillo es éste

```
$ cat ls-out.txt | ./lee-palabras
```

Mediante este comando redirigimos la salida estándar del comando `cat` a la entrada estándar de `lee-palabras`.

Es interesante recordar que los *strings* que se imprimen por pantalla pueden capturarse fácilmente mediante una variable. Por ejemplo, ejecutad

```
$ var=$(./imprime-mensajes)
$ echo $var
```

Mediante el comando anterior capturamos la salida estándar de la aplicación `imprime-mensajes` y la asignamos a la variable `var`. De forma similar, podemos ejecutar

```
$ var=$(cat ls-out.txt)
$ echo $var
```

3. Comandos útiles

Se describen a continuación algunos comandos de uso común en la línea de comandos.

3.1. `dirname` y `basename`: extraer directorio y fichero de una ruta

Los comandos `dirname` y `basename` pueden ser especialmente útiles para procesar fichero. Permiten extraer de un fichero, el directorio y su nombre de fichero. Por ejemplo, ejecutad

```
$ dirname "$(pwd)"/prueba.txt
$ basename "$(pwd)"/prueba.txt
```

Estos comandos pueden ser útiles en comandos que procesan ficheros, como por ejemplo el comando `find` de la sección 3.6. El resultado de la ejecución se puede capturar fácilmente en una variable, tal y como hemos realizado hace un momento

```
$ var=$(basename "$(pwd)"/prueba.txt)
$ echo $var
```

3.2. `file`: determinar el tipo de fichero

El comando `file` es muy útil para determinar el tipo de fichero de un archivo cualquiera. En particular, hagamos el siguiente experimento: a) coge el fichero de la práctica, cópialo a otro fichero y cambia su extensión a un fichero con extensión "`txt`" por ejemplo, b) abre el navegador de ficheros (en concreto, el Dolphin de KDE) y intenta abrirlo haciendo clic sobre él, c) el navegador intentará abrirlo con un editor de texto sencillo en vez del

visor `pdf` puesto que identifica, en primera instancia, un fichero a partir de su extensión.

El comando `file` permite identificar el tipo de fichero de un archivo cualquiera, y el comando lo hace a partir del contenido del archivo no de su extensión. Vamos a probarlo con el archivo que acabamos de crear

```
$ file practica-copia.txt
```

Probad de hacer lo mismo con otros archivos y veréis que el comando es capaz de identificar el tipo de fichero independientemente de la extensión del fichero.

El gestor de ficheros (en concreto, el Dolphin de KDE) utiliza, de hecho, el comando `file` para identificar el tipo de fichero en caso que no pueda identificar el tipo de archivo. Prueba de cambiar la extensión del fichero a "`xxx`" y prueba de abrirlo con un clic. Lo hará bien!

3.3. `touch`: cambiar la fecha de acceso y modificación de un fichero

El comando `touch` que permite cambiar la fecha de acceso y modificación de un fichero. La fecha de acceso de un fichero es la fecha en que fue accedido por lectura por última vez, mientras que la fecha de modificación es la fecha en que fue modificado por última vez. Situémonos en el directorio `redireccion` y ejecutemos

```
$ ls -l imprime-mensajes.c
$ touch imprime-mensajes.c
$ ls -l imprime-mensajes.c
```

Mediante el comando `touch` se modifica la fecha de acceso y modificación de un fichero a la fecha actual. El comando `touch` también permite, en caso necesario, establecer una fecha diferente de la actual. La fecha puede ser tanto del pasado como del futuro. Por ejemplo, podemos cambiar la fecha al 23 de abril del 2030, a las 10:00!

```
$ touch --date="2030-04-23 10:00" imprime-mensajes.c
```

Uno de los usos típicos del comando `touch` es en combinación con `make`: hemos visto en la segunda práctica que este comando utiliza el fichero `Makefile` para establecer en qué orden se compilan los ficheros. En el caso en que, por ejemplo, un fichero objecto (extensión .o) sea más nuevo que la fuente en C (extensión .c) el comando `make` decide no compilar el fichero objecto. Si

queremos forzar la compilación del fichero fuente podemos borrar el fichero objeto o bien podemos cambiar la fecha de modificación del fichero fuente a la fecha actual.

Se recomienda probarlo con el `Makefile` de la práctica 2. Ejecutad dos o más veces el comando `make` y veréis que a partir de la segunda ejecución ya no realiza ninguna acción. Para recompilar las fuentes ejecutad

```
$ touch *.c
$ make
```

3.4. `ps` y `kill`: gestión de procesos

Los comandos `ps` y `kill` se utilizan para gestionar procesos. Abrid un terminal y ejecutad

```
$ ps
```

El comando anterior imprime por pantalla los procesos (con su correspondiente identificador, PID) por pantalla. Por defecto sólo imprimirá aquellos procesos que han sido ejecutados desde el terminal que hemos abierto. Probad lo siguiente

```
$ kwrite &
$ ps
```

Observar que ahora se imprimirá por pantalla el proceso `kwrite` que hemos ejecutado desde terminal. En caso que se ejecute kwrite desde otro terminal o desde la interfaz gráfica, no aparecerá el proceso `kwrite` en la lista.

Podemos listar todos los procesos que se ejecutan actualmente en el ordenador con

```
$ ps aux
```

La lista es larga. Para cada proceso se indica, entre otros datos, el usuario propietario del proceso, su identificador (PID), el consumo de CPU y el consumo de memoria RAM. Podemos guardar la lista de todos los procesos que se ejecutan en un fichero

```
$ ps aux > procesos.txt
```

Editad el fichero y identificad, dentro de la lista, la aplicación `kwrite` que hemos ejecutado hace un momento. Podemos "matar" el proceso mediante la aplicación kill

```
$ kill <pid>
```

donde `<pid>` referencia al número que identifica el proceso a matar. El comando ha recibido el nombre de "`kill`" dado que el comando se utiliza típicamente para enviar la señal de terminación a un proceso ("matar"). El comando, permite, sin embargo, enviar otras señales.

3.5. `wc`: contar líneas, palabras y caracteres de un archivo

El comando `wc` (word count) es un comando que permite contar el número de palabras, líneas y caracteres de un archivo de texto plano. La entrada de datos es por la entrada estándar y los resultados se imprimen por la salida estándar.

Para comenzar ejecutaremos el comando desde el terminal sin ningún argumento

```
$ wc
```

Una vez introducido el comando, podemos introducir (mediante el teclado) palabras en la misma o diferentes líneas. Para acabar de introducir palabras pulsar `<Control+D>`. Se imprimirán por pantalla tres números: el número de líneas, el número de palabras y el número de caracteres que hemos introducido por teclado.

Podemos especificar un archivo (de texto) en la línea de comandos. En vez de analizar los datos introducidos por teclado, wc analizará el fichero de texto.

```
$ wc ls-out.txt
```

donde `ls-out.txt` es el fichero obtenido en la sección 2.

Podemos aplicar una tubería al comando `wc`

```
$ ls | wc
```

Mediante el comando anterior redirigimos la salida del comando `ls` a la entrada de `wc`.

Además de permitir especificar un fichero como argumento, el comando `wc` permite especificar otros argumentos. Ejecutemos por ejemplo

```
$ ls | wc -l
```

Ejercicio 3. Qué es lo que hace la opción "`-l`"? Qué información da, por lo tanto, el comando "`ls | wc -l`"?

Finalmente, ejecutemos

```
$ num=$(ls | wc -l)
$ echo $num
```

Ejercicio 4. Què hacemos mediante éste último comando?

3.6. `find`: buscar archivos en un directorio

El comando `find` es una herramienta que permite buscar ficheros (incluyendo directorios) dentro de una jerarquía de directorios. Permite una funcionalidad mucho mayor que las típicas herramientas de búsqueda de ficheros disponibles en las interfaces gráficas. Aquí sólo vamos a describir las opciones más interesantes y comunes.

Una de las formas más básicas de utilización es la búsqueda de ficheros. El patrón es éste:

```
$ find <directorio_inicial> -name <fichero>
```

donde `<directorio_inicial>` es el directorio a partir del cual queremos comenzar a buscar, y `<fichero>` es el fichero a buscar.

Para experimentar situémonos en el directorio `coreutils-8.25`. Por ejemplo, el comando

```
$ find . -name wc.c
```

buscará el fichero `wc.c` a partir del directorio actual, especificado mediante un punto. Se imprimirá por pantalla la ruta del fichero `wc.c` (este fichero corresponde a la implementación del comando `wc` visto anteriormente).

Podemos también especificar otros directorios iniciales de búsqueda

```
$ find /home/lluis/Escritorio -name wc.c
```

Para buscar todos los ficheros con extensión "`*.c`" a partir del directorio actual ejecutamos

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

Ejercicio 5. Mediante el comando anterior se imprimirán por pantalla tanto ficheros como directorios que tengan extensión "`*.c`". Hacer la prueba de crear un directorio con extensión `.c` y observar que aparece en los resultados de búsqueda.

Podemos restringir la búsqueda únicamente a ficheros mediante

```
$ find . -type f -name *.c
```

done "`-type f`" especifica que los resultados únicamente queremos incluir ficheros. Mediante "`-type d`" indicamos que queremos únicamente directorios en los resultados de búsqueda.

Cuántos ficheros "`*.c`" existen en la jerarquía actual? Lo podemos saber fácilmente con la siguiente instrucción

```
$ find . -type f -name *.c | wc -l
```

En vez de buscar los ficheros por su nombre, podemos buscar los ficheros por sus permisos. Por ejemplo,

```
$ find . -type f -perm 755
```

Mediante el comando anterior buscamos todos los archivos con permisos 755, que corresponden a ficheros ejecutables (ver comando `chmod` de la práctica 1). Mediante esta opción hay que especificar exactamente los permisos del archivo a buscar. Se pueden especificar los permisos de forma parcial, de forma similar que con `chmod`. Para ello ver el manual del comando `find`.

Una funcionalidad útil del comando `find` es la posibilidad de ejecutar un comando para cada archivo encontrado. Atención, aplicar el siguiente comando con cuidado!

```
$ find . -type f -name *.o -exec rm {} \;
```

La opción `exec` permite especificar el comando a aplicar a cada elemento encontrado por `find`. La cadena "`{}`" se sustituye, cada vez que se ejecuta `rm`, por el elemento encontrado por `find`. La cadena "`\;`" se utiliza para indicar dónde acaba el comando a ejecutar especificado mediante `exec`. Por lo tanto, mediante este último comando `find` estamos indicando que queremos borrar todos los ficheros con extensión "`*.o`". Una vez hecho, no hay vuelta atrás!

Ejercicio 6. La opción "`print`" imprime por pantalla cada uno de los elementos encontrados. Qué es lo que hace entonces el siguiente comando?

```
$ find . -type f -perm 755 -print -exec chmod 644 {} \;
```

Se pueden utilizar otros criterios para buscar ficheros. Para ello ejecutemos primero el siguiente comando

```
$ find . -name *.c -exec touch --date="2016-03-30 11:00" {} \;
```

Busquemos ahora los ficheros en función de su fecha de modificación

```
$ find . -type f -mmin -60
$ find . -type f -mmin +60
```

La primera instrucción permite buscar todos aquellos ficheros que han sido modificados hace menos de 60 minutos. La segunda busca los ficheros que han sido modificados como mínimo hace 60 minutos.

Finalmente, comentar que también se pueden buscar ficheros por el tamaño en bytes:

```
$ find . -type f -name *.mp3 -size +10M -print -exec rm {} \;
```

El comando anterior combina diversos criterios de búsqueda: debe ser un fichero, debe tener extensión `mp3` y debe tener un tamaño superior a `10M`. Los criterios de búsqueda se han combinado por defecto mediante una "and" lógica: para cada elemento encontrado que cumple con las condiciones, se imprime su nombre por pantalla y se borra. Los criterios también se pueden combinar mediante una "or" lógica (mirad el manual!).

Hemos visto que la opción `exec` permite ejecutar cualquier comando sobre cada fichero (también se puede ejecutar un script!). De forma similar a lo realizado con los anteriores comandos, es interesante mencionar también que el resultado de la búsqueda se puede asignar a una variable del terminal. Después se puede utilizar la lista en un script para realizar una operación sobre cada uno de los ficheros.

```
$ var=$(find . -type f -name *.c)
$ for i in $var; do echo "Fichero $i"; done
```

3.7. `sort`: ordenar líneas de texto

El comando `sort` permite ordenar, por defecto alfabéticamente, líneas de texto plano. Ejecutemos las siguientes instrucciones en el directorio `coreutils-8.25`.

```
$ find . -name *.c > find-out.txt
$ sort find-out.txt
```

La función sort ordena alfabéticamente los datos del fichero `find-out.txt` y los imprime por la salida estándar. En caso que no se especifique ningún fichero a ordenar en la línea de comandos, `sort` coge los datos de la entrada estándar. Por ello, las dos instrucciones anteriores se pueden unificar en una única instrucción mediante una tubería.

```
$ find . -name *.c | sort
```

Podemos guardar el resultado de la búsqueda en un fichero, o bien asignarlo a una variable como en el siguiente ejemplo

```
$ var=$(find . -name *.c | sort)
```

Veamos otras opciones comunes del comando `sort`. Por ejemplo, podemos ordenar alfabéticamente en sentido inverso mediante la opción "`-r`" (reverse)

```
$ sort -r find-out.txt
```

En caso que el fichero de texto esté formado por múltiples columnas, podemos pedir que la ordenación se haga respecto una determinada columna. Además, también podemos pedir que la ordenación se haga numéricamente (y no alfabéticamente). Por ejemplo,

```
$ ps aux | tail -n +2 | sort -nk3
```

El último comando utiliza dos tuberías. Ignoremos por el momento el comando "`tail -n +2`". El comando "`ps aux`" imprime por pantalla todos los procesos que actualmente se ejecutan en el ordenador. Por defecto "`ps aux`" imprime por pantalla los procesos ordenados según su identificador de proceso (PID); en la 3a columna se imprime el uso de CPU de cada proceso. Mediante el comando "`sort -nk3`" de ordenan los datos de la 3a columna de forma numérica. Por lo tanto, se imprimirán por pantalla los procesos según su uso de CPU.

Ejercicio 7. Cuál es aquí la utilidad del comando "`tail -n +2`"? Prueba de quitarlo e investiga mirando el manual!

3.8. `grep`, `egrep` y `fgrep`: buscar cadenas en un fichero de texto

El comando `grep` y la familia de comandos asociados, `egrep` y `fgrep`, son una familia de comandos que permiten buscar una cadena en un fichero de texto e imprimir por la salida estándar aquellas líneas en las que hay coincidencia. Para ejecutar los comandos mostrados en esta sección se recomienda ejecutar antes

```
$ alias grep='\grep --color=auto'
$ alias egrep='\egrep --color=auto'
$ alias fgrep='\fgrep --color=auto'
```

Para evitar haber de introducir estos comandos múltiples veces, incluid estos comandos en el fichero `~/.bashrc`, el fichero de script que se ejecuta cada vez que abrimos un terminal.

Dentro del directorio `grep-experiments` ejecutemos la siguiente instrucción

```
$ grep their book.txt
```

Esta instrucción busca la palabra "their" en el fichero `book.txt`, que es un libro en formato de texto plano. Al ejecutar esta instrucción se imprimirán por pantalla todas las líneas en las que aparece esta palabra. Deberían de aparecer en color las coincidencias encontradas.

Ejercicio 8. En cuántas líneas diferentes aparece la palabra "their" en el libro? Hay alguna forma rápida de averiguarlo?

Para simplificar la explicación de la funcionalidad del comando `grep` utilizaremos a partir de ahora el fichero `words`, que es un fichero que contiene miles de palabras de diccionario, una por cada línea. Busquemos la cadena `operating` dentro del fichero `words`

```
$ grep operating words
```

El comando nos imprime por pantalla todas aquellas líneas del fichero `words` en las que aparece la cadena `operating` (recordar que `grep` imprime toda la línea en la que se produce la coincidencia; aquí tenemos por casualidad una palabra por línea). Generemos ahora dos ficheros nuevos para realizar experimentos

```
$ grep operating words > sub-operating
$ grep system words > sub-system
```

A continuación buscaremos la cadena `ing` en los dos ficheros anteriores

```
$ grep ing sub-*
```

Este último comando nos imprime por pantalla, para cada uno de los dos ficheros, las líneas de los ficheros en los que aparece la cadena `ing`. Mediante la opción "`-v`" indicamos a `grep` que imprima por pantalla todas aquellas líneas en las que no hay coincidencia con la búsqueda

```
$ grep -v ing sub-*
```

Otra opción interesante es "`-l`": esto hace que únicamente se imprima por la salida estándar el nombre del fichero en caso que se produzca al menos una coincidencia dentro del fichero. Observar que con el siguiente comando buscamos en los tres ficheros: `words`, `sub-operating` y `sub-system`.

```
$ grep -l operating *
```

Se deberían imprimir únicamente dos ficheros por pantalla: `words` y `sub-operating`, que son los dos ficheros que contienen la cadena. También podemos pedir a `grep` que se impriman los ficheros en los que no aparece la palabra `operating`:

```
$ grep -L operating *
```

En caso que no se especifique ningún fichero a procesar en la línea de comandos, `grep` coge los datos de la entrada estándar. Es por ello que `grep` puede ser especialmente útil mediante tuberías. Por ejemplo, ejecutad

```
$ kwrite &
$ ps aux | grep kwrite
```

Ejercicio 9. Qué es lo que hace el comando anterior?

Es útil combinar el comando `find` y `grep` para buscar ficheros que contienen determinadas cadenas. Por ejemplo, si nos situamos en el directorio `coreutils-8.25`:

```
$ find . -name *.c -exec grep -l printf {} \;
```

Mediante el comando anterior buscamos en todos los ficheros "`*.c`" la cadena `printf`. Se imprimirán por pantalla todos los nombres de archivos en los que aparezca al menos una vez la cadena `printf`. Cuántos archivos son, en total? Fácil

```
$ find . -name *.c -exec grep -l printf {} \; | wc -l
```

El comando `grep` permite buscar una cadena en un fichero de texto y por defecto permite el uso de expresiones regulares básicas en la cadena de búsqueda. Para poder utilizar todo el potencial de las expresiones regulares, hay que ejecutar `grep` con la opción "`-E`", "`grep -E`", equivalente a ejecutar la instrucción `egrep`.

Una expresión regular es una secuencia de caracteres que forma un patrón de búsqueda. Situémonos de nuevo en el directorio `grep-experiments`. Veamos un par de ejemplos

```
$ egrep '[aeio]' some-words.txt
```

Observar que se utilizan las comillas simples en la búsqueda. Este comando buscará en el fichero `some-words.txt` todas las cadenas en las que aparezca una "`a`", una "`e`", una "`i`" o una "`o`". Es importante entender que la cadena de búsqueda puede encontrarse múltiples veces en una misma línea (o palabra, como ocurre en este caso).

Podemos buscar las líneas en las que no aparezcan los caracteres "`a`", "`e`", "`i`" ni "`o`".

```
$ egrep -v '[aeio]' some-words.txt
```

Observar que se imprime por pantalla la palabra "`ubuntu`", que sólo contiene la letra "`u`".

Una expresión regular es sensible a las minúsculas y mayúsculas. Por ejemplo, observad la diferencia de los resultados de las siguientes instrucciones

```
$ egrep -v '[aeio]' words
$ egrep -v '[aeioAEIO]' words
```

Cuando se ejecuta el primer comando pueden aparecer por pantalla líneas que contienen las letras "`A`", "`E`", "`I`" o "`O`", mientras que no será así en el segundo caso.

Mediante la siguiente expresión regular buscamos las cadenas "`hello`" y "`hallo`" en el fichero

```
$ egrep 'h[ae]llo' words
```

Las expresiones regulares permiten especificar un conjunto de letras mediante un guión. Por ejemplo, mediante '`[a-q]`' hacemos referencia a todas las letras entre la '`a`' y la 'q'. Analizad bien la expresión regular

```
$ egrep '[a-q]*lo' words
```

El carácter "*" indica que el carácter indicado antes puede aparecer cero o más veces. Es decir, la expresión regular '`[a-q]*`' indica que queremos que los caracteres '`a-q`' aparezcan cero o más veces. La siguiente expresión expresión regular permite encontrar líneas (atención, no palabras!) que comiencen por la letra "`d`" y acaben con "`lo`". En medio puede haber cualquier cadena de caracteres

```
$ egrep '^d.*lo$' words
```

El carácter "^" hace referencia al principio de la línea mientras que "$" al final de la línea; "." hace referencia a cualquier carácter (incluyendo espacios) y "*" indica que el carácter indicado justo antes (el ".") puede aparecer cero o más veces.

Ejercicio 10. Utilizando expresiones regulares, busca en el fichero `words` cadenas que tengan una "d", seguida de una "o" y seguida de una "b". Entre la "d" y la "o", así como de la "o" y la "b" debe haber al menos un carácter. Es interesante que comprobéis el efecto de la expresión regular utilizada con la cadena "dkokbokb". Nótese que el patrón tiene 2 coincidencias: una con la subcadena "dkokb" y otra con la cadena completa. Qué ocurre en este caso?

Además de los caracteres "^" y "$", es interesante mencionar también el metacaracter "\b", que hace referencia a un límite de palabra, ver http://www.regular-expressions.info/wordboundaries.html. Por ejemplo, cread un fichero de texto plano `fichero.txt` con la cadena "This island is beautiful" y analizar bien la ejecución de las siguientes dos instrucciones

```
$ grep "is" fichero.txt
$ grep '\bis\b' fichero.txt
$ grep '\bis.*is\b' fichero.txt
```

Como se puede ver, las expresiones regulares permiten realizar búsquedas de patrones de búsqueda complejas. En este documento no se describirá todo el lenguaje asociado a las expresiones regulares. Hay multitud de páginas web en las que se puede encontrar información al respecto, por ejemplo https://en.wikipedia.org/wiki/Regular_expression y http://www.regular-expressions.info.

Observar que `egrep` (equivalente a ejecutar "`grep -E`") utiliza una serie de metacaracteres en la expresión regular para especificar la cadena a buscar. Tal y como se ha comentado antes, `grep` (sin la "`e`" del inicio) permite por defecto el uso de un subconjunto del léxico utilizado las expresiones regulares. En caso que queramos buscar una cadena fija se recomienda utilizar el comando `fgrep` (equivalente a "`grep -F`") para evitar tener que utilizar el carácter de escape "\" en los caracteres reservados en las expresiones regulares.

3.9. `sed`: buscar y sustituir cadenas

El comando sed es un comando que habitualmente se utiliza para buscar y sustituir cadenas en un fichero de texto plano. La forma habitual de utilización es similar a la utilizada en el siguiente comando. Ejecutar los siguientes comandos dentro del directorio `grep-experiments`.

```
$ sed 's/lo/la/g' some-words.txt
```

El primer argumento a `sed`, en el caso que queramos buscar y sustituir cadenas, es `'s/<cadena a buscar>/<cadena a sustituir>/g'`. La "`s`" hace referencia a "search", y la cadena a buscar así como la cadena a sustituir se separan por la barra "`/`". La "g" del final es una opción que indica que se debe realizar la búsqueda y sustitución por todo el documento.

El comando anterior imprime por la salida estándar todo el fichero, sustituyendo la cadena "`lo`" por "`la`". Se pueden utilizar también expresiones regulares a la hora de buscar

```
$ sed 's/h[ae]llo/bon dia/g' some-words.txt
```

Es interesante utilizar la expresión regular encontrada por la búsqueda en la cadena sustituida. Para ello hay que utilizar el símbolo "&". Analizar bien el comando:

```
$ sed 's/h[ae]llo/ei, &!/g' expresion-regular.txt
```

El comando `sed` puede ser útil para manipular directorios y nombres de ficheros. Para ello situémonos en el directorio `coreutils`-8.25. Supongamos que queremos sustituir en todos los nombres de directorios la cadena "`/test/`" por una cadena "`/`".

Observar que ahora aparece el símbolo "`/`" en la cadena de búsqueda y, como hemos dicho antes, la cadena a buscar y sustituir se separan también por "`/`". Para poder implementar esta búsqueda se puede utilizar el carácter de escape "\". El comando `sed` para buscar y sustituir es `'s/\/test\//\//'`. Esta notación es un poco complicada y fea, y por ello el comando sed permite utilizar otros delimitadores entre la cadena a buscar y a sustituir, como por ejemplo "`:`" o "`_`". El comando sed para buscar y sustituir se convierte en `'s:/test/:/:'`, mucho más fácil de entender.

Implementemos la funcionalidad propuesta: sustituir en todos los nombres de directorios la cadena "`/test/`" por la cadena "`/`". Esto corresponde con el comando

```
$ find . -type d -exec echo {} \; | sed 's:/tests/:/:g'
```

En caso que queramos hacer múltiples sustituciones podemos utilizar la opción "`-e`".

```
$ find . -type d -exec echo {} \; | sed -e 's:/tests/:/:g' -e 's:/old/:/antiguo/:g'
```

3.10. `awk`: extraer columnas de un fichero de texto

El comando `awk` es un comando muy potente y versátil. Es, en cierto modo, similar al comando `grep`. El comando `awk` permite realizar muchas mas cosas de las que se pueden hacer con `grep`, puesto que `awk` dispone de su propio lenguaje de programación, mientras que `grep` sólo es un herramienta de filtrado. Aquí, sin embargo, no vamos a utilizar el comando `awk` para buscar cadenas en un fichero de texto, sino que lo utilizaremos para otra funcionalidad para la cual se utiliza habitualmente: la extracción de determinadas columnas de un fichero de texto.

Recordemos un comando que hemos ejecutado anteriormente en el directorio `coreutils-8.25`:

```
$ find . -name *.c -exec grep -l printf {} \; | wc
```

Se imprimirán por pantalla tres números: el número de líneas, palabras y caracteres de los datos que entran por la entrada estándar al comando `wc`. Para imprimir únicamente el número de líneas que se han encontrado podemos utilizar la opción "`-l`" del comando `wc`.

```
$ find . -name *.c -exec grep -l printf {} \; | wc -l
```

Imaginemos, sin embargo, que wc no dispone de la opción "`-l`". En este caso podemos utilizar el comando `awk` para extraer la información pedida. En particular, observar que estamos interesados en la primera columna de la información que wc ha imprimido por la salida estándar. Ejecutemos el siguiente comando

```
$ find . -name *.c -exec grep -l printf {} \; | wc | awk '{print $1}'
```

El comando `awk` anterior lee la entrada estándar y imprime por pantalla la primera columna. Por defecto, las columnas de texto están separadas en `awk` por (múltiples) espacios y tabuladores.

Es interesante volver a otro ejemplo anterior para ver el uso de `awk`

```
$ kwrite &
$ ps aux | grep kwrite
```

El comando anterior imprime por pantalla los procesos en los que aparece la cadena `kwrite`, incluyendo (seguramente) el comando `grep`. Observar que el identificador de proceso aparece en la segunda columna.

Ejercicio 11. Qué es lo que hace el siguiente comando?

```
$ ps aux | grep kwrite | head -n -1 | awk '{print $2}'
```

Podemos utilizar la idea del comando anterior para automatizar con un script que se mate a un proceso

```
$ var=$(ps aux | grep kwrite | head -n -1 | awk '{print $2}')
$ kill $var
```

4. Problemas a resolver

La práctica consta de una serie de problemas. La práctica se realizará individualmente y se entregarán los scripts en ficheros individuales.

Las fechas de entrega de la práctica se indican en el fichero de planificación colgado en el campus. El test se realizará el último día de la práctica 3 tal y como se indica en el fichero de planificación.

Restricciones para la resolución de los problemas

Para resolver estos problemas únicamente pueden utilizarse los comandos presentados en ésta práctica y las prácticas anteriores. No se permite utilizar otros comandos a menos que se justifique adecuadamente su uso en el script. Para el caso de las expresiones regulares, sin embargo, se permite utilizar toda la sintaxis disponible.

Problema 1 (3 puntos)

Se pide realizar un script que realice los siguientes pasos utilizando los ficheros del directorio `coreutils-8.25`. Para resolver este problema se puede utilizar el hecho que grep, egrep y fgrep devuelven un 0 si hay una coincidencia con la cadena pedida, y 1 en caso contrario.

1. Encontrar **todos** los ficheros dentro del directorio `coreutils-8.25` y subdirectorios. El resto de puntos de este ejercicio debe resolverse únicamente con esta búsqueda inicial.

2. Utilizando el resultado del paso 1, dar permiso de ejecución a todos los ficheros con extensión ".sh" (i.e. evitar volver a buscar los ficheros ".sh"). Se valorará evitar la utilización de bucles.

3. Crea la carpeta shFiles y copia todos los archivos ".sh" allí. Evitar volver a buscar los ficheros ".sh". Se valorará evitar la utilización de bucles.

4. Utilizando el resultado del paso 1, cambiar la fecha de los ficheros ".sh" al 02 de marzo del 1996 únicamente si el nombre de fichero comienza con una "s". Evitar volver a buscar los ficheros ".sh". Asegurate de mostrar el cambio de fecha.

Problema 2 (3 puntos)

Se pide realizar un script que realice los siguientes pasos sobre la base de datos de libros colgada en el campus.

1. Buscar todos los ficheros "`*.txt`" que están incluidos dentro del directorio.

2. Filtrar la búsqueda anterior y quedarse únicamente con los ficheros "`*.txt`" que se encuentren debajo del subdirectorio "etextXX", donde "X" es un número del 0 al 9. Los ficheros pueden estar a cualquier profundidad.

3. Contar, para cada uno de los ficheros encontrados en el paso anterior, cuántas líneas y palabras tiene éste. Esta información se volcará en un único fichero de texto con nombre `table.txt` con las cuatro columnas que `wc` imprime por defecto a pantalla: el número de líneas, el número de palabras, el número de caracteres y el nombre de fichero. El fichero de texto `table.txt` tendrá por lo tanto tantas líneas como ficheros analizados.

4. Ordenar el fichero obtenido en el paso 3 por el número de palabras (la segunda columna), de mayor a menor, y guardar los resultados en un fichero temporal `tmp.txt`.

5. Imprimir el mensaje "El numero de ficheros analizados es XXX" en un nuevo fichero denominado score.txt, donde XXX es el número de ficheros analizados.

6. Imprimir el mensaje "Los 10 ficheros con mayor numero palabras" seguido de los 10 primeros resultados obtenidos en el punto 4. A continuación imprimir el mensaje "Los 10 ficheros con menos palabras" seguido de los 10 resultados con menos palabras. No es necesario que las columnas estén "perfectamente" alineadas. Mirad el manual de `head` y `tail` para responder el problema.

Problema 3 (4 puntos)

Cada uno de los siguientes puntos corresponde a una instrucción independiente del resto. El objetivo en sí es encontrar la expresión regular que permita implementar la funcionalidad pedida para el fichero `book.txt` que se encuentra dentro del directorio `grep-experiments`. Asegurar que la búsqueda se realiza correctamente haciendo que `grep` imprima los resultados en color por pantalla (ver sección 3.8).

Observar que si se habla de "cadena" se hace referencia a un conjunto de caracteres, mientras que si se habla de "palabra" se hace referencia a una secuencia de caracteres entre la 'a' y la 'z', ambos incluidos, de longitud positiva y sin incluir las mayúsculas.

1. Encuentra las lineas que contienen la cadena `evil` y `will` en una misma línea. Tanto `evil` como `will` pueden formar parte de una palabra más grande y pueden aparecer en cualquier orden.

2. Haz lo mismo que en el apartado anterior mostrando también el número de linea en el que se encuentran las apariciones en el archivo original.

3. Haz lo mismo que en el apartado anterior formateando la salida de manera que primero aparezca el contenido de la linea y después el número de linea en el que se ecuentra, separados por el texto " `- Line number :` ". Ejemplo: `bla bla will bla evil - Line number : 120`.

4. Encuentra todas las lineas que contienen un correo electronico. Explica en el script cada parte de tu expresión regular. Empieza con "`egrep @ book.txt`" y refina la búsqueda.

5. Haz lo mismo que en el apartado anterior pero muestra solo los correos (no la linea entera) y el número de linea en el fichero donde se encuentran.