PROGRAMACIÓN INTEGRATIVA

PRIMERA SESIÓN PRÁCTICA

EJERCICIO RESUELTO

1. Programar un script que encuentre todos los ficheros dentro del directorio actual (recursivamente) cuyo nombre cumpla un patrón determinado. El patrón debe proporcionarse como parámetro posicional.

   Conceptos básicos relacionados:

   - Entrada/salida estándar: tal como se estudia en la asignatura Sistemas Operativos de 2º curso del Grado en Ingeniería Informática, los flujos de entrada/salida/error estándar son canales de comunicación preconectados de un comando Unix. Toda la comunicación a través de ellos se realiza usando texto estándar (ver Sección 2.1.2 de los apuntes).
   - Ejecución de `pipes` de comandos (ver Sección 2.2.2.3 de los apuntes).
   - Expresiones regulares (ver Sección 2.3.1 de los apuntes).
   - Creación de scripts del shell (ficheros tipo `.sh`, ver Sección 2.2.1 de los apuntes).
   - Paso de parámetros a los scripts del shell (ver Sección 2.2.3 de los apuntes).
   - Significado de comillas simples (`'...'`) y dobles (`''...''`) (ver Sección 2.5.4 de los apuntes).

   Comandos relacionados: (ver también las páginas de `man` para más información)

   - `find`: se trata de un comando que lista ficheros y directorios contenidos en la ruta actual. Por defecto lo hace recursiva y exhaustivamente.
   - `sed`: stream editor. Utilizado para hacer transformaciones básicas sobre texto que recibe de un flujo de entrada (fichero o entrada estándar).
   - `grep`: filtra la entrada, imprimiendo tan solo las líneas que cumplan un patrón especificado.

   Existen diferentes formas de resolver este ejercicio. En la primera y más sencilla, utilizaremos los comandos `find`, `grep` y `sed` en un único `pipeline`. Cada pieza de este `pipeline` llevará a cabo una función diferente:

   a) En primer lugar, ejecutaremos un comando que listará todos los ficheros contenidos dentro del directorio actual:

```
[user@host:/usr]$ find
.
./lib32
./lib32/libresolv.o
./lib32/libm.a
./lib32/libnss_files.so
./lib32/crt1.o
./lib32/gconv
```

```
./lib32/gconv/IBM1025.so
./lib32/gconv/IBM851.so
                                        ⋮
```

Nos encontramos con un primer problema: queremos listar sólo ficheros, y `find` está mostrando también directorios y enlaces simbólicos (marcados en rojo en el listado anterior). Una posible solución es utilizar la opción `-type f` del comando `find`, que especifica que sólo debe listar ficheros regulares:

```
[user@host:/usr]$ find -type f
./lib32/libm.a
./lib32/crt1.o
./lib32/gconv/IBM1025.so
./lib32/gconv/IBM851.so
                                        ⋮
```

b) Una vez que hemos generado la lista de ficheros a analizar, debemos filtrarla para eliminar todos aquellos que no cumplan el patrón especificado. Esto puede realizarse mediante el comando `grep`:

```
[user@host:/usr]$ find -type f|grep "\.so$"
./lib32/gconv/IBM1025.so
./lib32/gconv/IBM851.so
./lib32/gconv/SHIFT_JISX0213.so
                                        ⋮
```

Véase cómo hemos escapado el punto en la expresión regular mediante la barra invertida (‘`\.so`’). Si no lo hubiésemos hecho así, ese punto hubiera sido interpretado como un caracter especial, significando “cualquier caracter”. Puede verse la diferencia ejecutando el comando y comparando las salidas de ambos mediante `diff`:

```
[user@host:/usr]$ find -type f|grep "\.so$" > /tmp/out1
[user@host:/usr]$ find -type f|grep ".so$" > /tmp/out2
[user@host:/usr]$ diff /tmp/out1 /tmp/out2
                                        ⋮
```

Al usar un caracter de anclaje (el ‘`$`’ final) en la expresión regular anterior hemos garantizado que la coincidencia sólo puede producirse en el nombre del fichero (la última parte de cada línea). Comprobemos la diferencia entre utilizar el anclaje y no hacerlo:

```
[user@host:/usr]$ find -type f|grep "\.so$" > /tmp/out1
[user@host:/usr]$ find -type f|grep "\.so" > /tmp/out2
[user@host:/usr]$ diff /tmp/out1 /tmp/out2
                                        ⋮
```

Como puede observarse, al obviar el anclaje la coincidencia con el patrón puede ocurrir en cualquier lugar de la línea. Esto podría ser problemático si existiesen directorios cuyo nombre coincidiese con el patrón a buscar. El enunciado establece que el patrón debe buscarse tan solo en el nombre del fichero, y no en la ruta completa. Por ejemplo:

```
[user@host:/usr]$ find -type f|grep "video"
./include/video/uvesafb.h
./include/video/sisfb.h
./include/video/edid.h
./include/linux/uvcvideo.h
./include/linux/videodev2.h
                                        ⋮
```

Para evitar esto, una solución consiste en forzar a que la parte de la ruta que corresponde con nombres de directorios no sea tenida en cuenta a la hora de evaluar el patrón. Una posible solución es eliminar esa parte de la ruta mediante `sed` antes de evaluar el patrón con `grep`:

```
[user@host:/usr]$ find -type f|sed -e "s/.*\/\(.*\)/\1/"
libm.a
crt1.o
IBM1025.so
IBM851.so
                                        ⋮
```

Como puede verse, el comando `sed`, al usarse con la opción `-e`, está realizando una sustitución línea a línea en su flujo de entrada, y proporcionando las líneas con la sustitución realizada en su salida estándar. El parámetro de la opción `-e` es un string `s/origen/destino/`, en el que `origen` es un patrón a reconocer y `destino` es la sustitución a realizar sobre el contenido que coincida con el patrón. Si analizamos el patrón `origen` utilizado, veremos que contiene las siguientes partes:

- `.` → caracter especial '`.`', que significa "cualquier caracter".
- `*` → caracter especial que modifica al '`.`' anterior, significando que pueden existir cero o más caracteres cualquiera.
- `\/` → se trata del caracter ordinario '`/`'. Debe escaparse usando la barra invertida inicial para evitar que `sed` interprete que ya ha terminado el patrón `origen` (recordemos que la sintaxis de la sustitución es `s/origen/destino/`, es decir, la barra se utiliza para separar los componentes de la sustitución).
- `\(` → caracter especial de apertura de paréntesis que indica que el contenido del texto que coincida con la parte del patrón desde este punto hasta el cerrado ('`\)`') debe ser guardado de modo que pueda ser utilizado más adelante (lo usaremos en la parte `destino`). Obsérvese que, para que el caracter '`(`' sea interpretado de forma especial, es necesario utilizar la barra invertida (sólo en expresiones regulares simples).
- `.*` → igual que los dos primeros caracteres de la expresión: deben encontrarse en este punto cero o más caracteres cualesquiera.
- `\)` → cerrado del bloque especial que almacena el texto para su uso posterior.

En cuanto a la expresión regular `destino`, consiste únicamente en el grupo '`\1`'. Se trata de una expresión especial que indica que debe copiarse aquí el contenido almacenado en la parte encerrada entre '`\(...\)`' en la expresión `origen`. Por tanto, lo que en resumen estamos solicitando a `sed` es que, para cada línea en su entrada estándar, busque un patrón consistente en encontrar "cualquier cosa", un caracter '`/`', y "cualquier cosa"; y sustituya todo el conjunto por el "cualquier cosa" que aparece a la derecha del último caracter '`/`'.

Nótese que, en ocasiones, el motor de expresiones regulares puede encontrar ambigüedades, es decir, diferentes maneras de emparejar el patrón solicitado con una línea de entrada. Por ejemplo, si la línea de entrada fuese:

```
./dir1/dir2/dir3/video
```

Existen cuatro maneras diferentes de emparejar la expresión regular `'.*\/\(.*\)'` con esta línea:

I) Asumir que el caracter `'/'` que aparece en el patrón se corresponde con la primera `'/'` de la línea. En este caso, el primer grupo `'.*'` se correspondería con el string `'.'`, y el grupo almacenado en `'\1'` se correspondería con `'dir1/dir2/dir3/video'`.

II) Asumir que el caracter `'/'` que aparece en el patrón se corresponde con la segunda `'/'` de la línea. En este caso, el primer grupo `'.*'` se correspondería con el string `'./dir1'`, y el grupo almacenado en `'\1'` se correspondería con `'dir2/dir3/video'`.

III) Asumir que el caracter `'/'` que aparece en el patrón se corresponde con la tercera `'/'` de la línea. En este caso, el primer grupo `'.*'` se correspondería con el string `'./dir1/dir2'`, y el grupo almacenado en `'\1'` se correspondería con `'dir3/video'`.

IV) Asumir que el caracter `'/'` que aparece en el patrón se corresponde con la última `'/'` de la línea. En este caso, el primer grupo `'.*'` se correspondería con el string `'./dir1/dir2/dir3'`, y el grupo almacenado en `'\1'` se correspondería con `'video'`.

Para nuestros propósitos, sería deseable que el motor de expresiones regulares escogiese esta última opción, que es la que hace que `'\1'` contenga el nombre del fichero, eliminando todos los componentes de su ruta. Esto es precisamente lo que ocurre, pues en caso de ambigüedad el motor se comporta de forma voraz (*greedy*), expandiendo cada componente de la expresión regular al máximo antes de pasar al siguiente. Por tanto, el primer grupo `'.*'` será del máximo tamaño posible, lo que se corresponde con la última de las opciones vistas anteriormente.

Una vez que hemos aislado el nombre del fichero, es posible aplicar `grep`:

```
[user@host:/usr]$ find -type f|sed -e "s/.*\/\(.*\)/\1/"| \
                  grep "video"
uvcvideo.h
videodev2.h
video.h
video.h
SDL_video.h
                                          ⋮
```

c) Sin embargo, la solución anterior no es ideal, dado que, aunque somos capaces de aislar los nombres de los ficheros que nos interesan, se pierde la información sobre la ruta a los mismos. Esto además provoca que aparezcan ficheros aparentemente repetidos. Una alternativa consiste, en lugar de eliminar la parte de los directorios mediante `sed`, en integrarla en el patrón buscado por `grep`:

```
[user@host:/usr]$ find -type f|grep "video[^/]*$"
./include/linux/uvcvideo.h
./include/linux/videodev2.h
./include/linux/usb/video.h
./include/linux/dvb/video.h
./include/SDL/SDL_video.h
                                          ⋮
```

Si analizamos ahora el patrón utilizado en el `grep`:

`video`	→	la cadena de caracteres ordinarios `'video'`.
`[^/]*`	→	cero o más caracteres cualesquiera, excepto el caracter `'/'`.
`$`	→	anclaje de final de línea.

Se pide, por tanto, que el grupo `'video'` aparezca inmediatamente antes del final de línea, permitiéndose solamente caracteres que no sean una barra entre su final y el final de la misma. De esta manera, garantizamos que `'video'` sólo se acepta si aparece tras el último caracter `'/'`.

d) Existe una solución alternativa utilizando únicamente el comando `find`:

```
[user@host:/usr]$ find -type f -regextype posix-basic \
        -regex ".*video[^/]*"
./include/linux/uvcvideo.h
./include/linux/videodev2.h
./include/linux/usb/video.h
./include/linux/dvb/video.h
./include/SDL/SDL_video.h
                                        ⋮
```

Como puede verse, existen muy pocas diferencias entre esta solución y la anterior. Destacan dos de ellas:

- Es necesario especificar el parámetro `-regextype posix-basic` en la llamada a `find`. Esto se debe a que el comando `find` por defecto no utiliza expresiones POSIX, sino de tipo Emacs.
- No es necesario especificar el caracter de anclaje `'$'` al final de la expresión regular. Esto se debe a que `find` asume implícitamente un anclaje de principio de línea (`'^'`) al principio de la expresión, así como un anclaje de final de línea (`'$'`) al final de la misma. Por el mismo motivo, es necesario especificar que, antes del grupo `'video'`, pueden aparecer cualquier número de caracteres de cualquier tipo (el grupo `'.*'` añadido con respecto a la versión del apartado anterior). En caso contrario, `find` sólo aceptará líneas que comiencen por el grupo `'video'`, debido al caracter de anclaje `'^'` implícito al comienzo de la expresión regular.

e) El ejercicio propone la escritura de un script que realice la operación de búsqueda automáticamente, proporcionándole el patrón a buscar. Para ello, insertamos los comandos vistos anteriormente en un fichero de texto y lo marcamos como fichero ejecutable usando `#! /bin/sh` en su primera línea. Utilizará como patrón a buscar el primer argumento recibido por línea de comandos:

```
[user@host:~]$ cat > ej1.sh << EOF
#! /bin/sh

find -type f|grep "\$1"
EOF
[user@host:~]$ chmod +x ej1.sh
[user@host:~]$ ./ej1.sh "video[^/]*$"
                                        ⋮
```

Se ha utilizado un `HEREDOC` (ver Sección 2.5.1.1 de los apuntes) para escribir el contenido del fichero `ej1.sh`. Se inserta el primer parámetro proporcionado al script, `$1`, en el medio de la expresión regular utilizada para procesar los nombres de fichero dentro de la ruta.

Nótese que el uso de `'\$1'`, con barra invertida, es necesario debido a la escritura del script mediante un `HEREDOC`, para que el shell interprete ese caracter `'$'` como un caracter ordinario y no realice una sustitución paramétrica. Si el script se crea directamente con un editor de texto, deberá omitirse el caracter `'\'`.

Préstese especial atención al uso de comillas dobles siempre que se proporciona una expresión regular al intérprete de comandos. Aunque técnicamente no es necesario cuando la expresión regular es un string sencillo (por ejemplo, al proporcionar la expresión `'video'`), sí podría ser problemático si la expresión regular a buscar contuviese caracteres especiales. Se deja como ejercicio para el lector experimentar a este respecto.

EJERCICIOS PROPUESTOS

2. Programar un script que encuentre todos los ficheros dentro del directorio actual (recursivamente) que contengan texto que cumpla un patrón determinado. El patrón debe proporcionarse como parámetro posicional.

3. Usar el script del paso anterior para encontrar todos los ficheros dentro del directorio actual cuyo contenido incluya la palabra `código`, con o sin tilde.