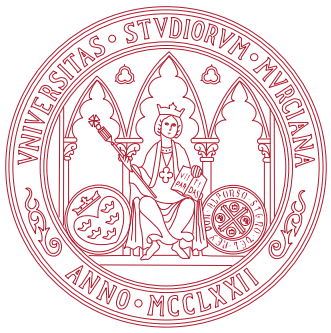

Universidad de Murcia

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

Boletín de prácticas 2 – Manejo avanzado del shell de Linux

Curso 2021/2022

Índice

1. Introducción

Partiendo de las destrezas en el manejo de órdenes básicas, adquiridas en la asignatura Fundamentos de Computadores de primer curso y repasadas en el boletín 1, en este boletín veremos primero cómo podemos modificar el modo en el que interactuamos con una orden redirigiendo su entrada y/o salidas de datos a ficheros y/u otras órdenes. Tras ello, describiremos algunas órdenes que son de gran utilidad para filtrar y procesar la información generada por una orden o almacenada en ficheros.

1.1. Objetivos

Al terminar el boletín el alumno debe ser capaz de:

- Manejar correctamente los redireccionamientos, tanto de la entrada como de las salidas de una orden.
- Integrar diferentes órdenes mediante la interconexión de sus entradas y salidas estándares a través de tuberías.
- Usar apropiadamente las principales ordenes de filtrado y procesamiento de datos, tanto individualmente como combinadas a través de tuberías.

1.2. Órdenes utilizadas

Las órdenes que veremos en este boletín son:

- `grep`
- `tr`
- `uniq`
- `tail`
- `tee`
- `sort`
- `cut`
- `head`
- `wc`
- `column`

En las páginas de manual de cada una de estas órdenes encontrarás información detallada de cómo usarlas.

2. Entrada/salida estándar y redirección

La filosofía de UNIX/Linux es en extremo modular. Se prefieren las herramientas pequeñas que realizan tareas concretas a las macro-herramientas que realizan de todo. Para completar el modelo, es necesario proporcionar mecanismos que permitan ensamblar estas herramientas pequeñas de tal forma que sea posible realizar un procesamiento complejo de la información. Estos mecanismos son el redireccionamiento de las entradas y salidas estándares de los procesos y las tuberías. A continuación describimos ambos.

Habitualmente, los procesos disponen de tres descriptores de fichero[1] a través de los que se comunican con otros procesos y con el usuario. Estos tres descriptores son:

- Descriptor 0, conocido como *entrada estándar*: los procesos leen de este descriptor para recibir datos de entrada. Normalmente, el descriptor 0 está asociado a la entrada del terminal en la que se está ejecutando el proceso, es decir, al teclado.
- Descriptor 1, conocido como *salida estándar*: los procesos escriben en este descriptor para mostrar sus resultados o datos de salida. Normalmente, este descriptor está asociado a la salida del terminal en el que se está ejecutando el proceso, es decir, la pantalla.
- Descriptor 2, conocido como *salida estándar de error*: los procesos escriben en este descriptor para mostrar mensajes de error que indiquen la causa de algún fallo. Al igual que el descriptor anterior, normalmente este descriptor está asociado a la salida del terminal en el que se está ejecutando el proceso, es decir, la pantalla.

[1] En UNIX/Linux, cuando se realiza una llamada al sistema `open` para abrir un fichero, el núcleo del sistema operativo devuelve un número para operar con ese fichero. A este número se le llama *descriptor de fichero*.

Aunque, como hemos dicho, estos tres descriptores están asociados al terminal en la que se ejecuta un proceso, es posible *redireccionarlos* para:

- Almacenar los datos de salida en un fichero determinado.
- Recibir los datos de entrada de un fichero concreto.
- Comunicar unos procesos con otros, de forma que trabajen como una unidad, haciendo cada uno una tarea especializada.

Cuando redireccionamos, lo que hacemos es cambiar el fichero al que representa un descriptor. Por ejemplo, cuando redireccionamos la entrada estándar, lo que hacemos es que el descriptor 0 deje de estar asociado al teclado para estar asociado a, por ejemplo, un fichero. Así, cuando un proceso lea del descriptor 0 para recibir nuevos datos de entrada, lo que ocurrirá es que recibirá dichos datos del fichero y no del teclado. Es más, el proceso ni se dará cuenta del cambio pues para él el descriptor 0 sirve para recibir datos de entrada, sin importarle de dónde vienen esos datos.

Es importante remarcar que toda la labor de redireccionamiento de la entrada/salida de un proceso la realiza el shell que estemos utilizando, siendo totalmente transparente y ajena a dicho proceso. Como acabamos de explicar, el proceso recibirá datos de su entrada estándar y escribirá sus resultados y mensajes de error en su salida estándar y salida estándar de error, respectivamente, sin importarle qué sea en cada momento esa entrada o esas salidas estándares.

Hay varios operadores para redireccionar la entrada y las salidas de un proceso:

- >: redirecciona la salida estándar a un fichero; si el fichero existe, lo sobrescribe:

```
$ who > usuarios.txt
# Escribe en usuarios.txt el listado de usuarios conectados
```

- >>: redirecciona la salida estándar a un fichero; si el fichero existe, añade los datos al final del mismo.
- 2 >: redirecciona la salida estándar de error a un fichero; si el fichero existe, lo sobrescribe:

```
$ find / -type d  2>errores.txt
# Busca todos los directorios existente a partir del directorio '/'
# Los mensajes de error los escribe en el fichero errores.txt
```

- 2 >> : similar a >> pero para la salida estándar de error.
- `n>&m`: redirecciona el descriptor de fichero `n` al descriptor de fichero `m`; en caso de que `n` se omita, se sobrentiende un 1 (es decir, la salida estándar):

```
$ cat file directorio > salida.txt 2>&1
# Redirecciona la salida estándar al fichero salida.txt y la salida
# estándar de error a la salida estándar. El resultado es que toda
# la salida genera por la orden va al mismo fichero.
```

- <: redirige la entrada estándar a un fichero:

```
$ grep cadena < fichero.txt
# Busca «cadena» dentro de fichero.txt
```

- | (tubería): redirecciona la salida estándar de una orden a la entrada estándar de la orden que le sigue:

```
$ who | grep pilar
# Busca «pilar» entre el listado de usuarios conectados
```

- `|&`: tubería en la que se redireccionan tanto la salida estándar como la salida estándar de error de una orden a la entrada estándar de la orden que le sigue:

```
$ find /etc |& grep 'Permiso denegado'
# Busca aquellas entradas a partir de /etc para las que no se
# tiene permiso de acceso
```

  Observa que ni `|` ni `|&` permiten redireccionar solo la salida estándar de error de una orden a una tubería. Si queremos hacer esto, debemos usar la siguiente fórmula:

```
$ find /etc 2>&1 > /dev/null | wc -l
# Cuenta cuántos errores se han producido en la búsqueda de ficheros
# a partir de /etc (generalmente, por falta de permisos).
```

  A la hora de ejecutar esta línea de órdenes, Bash redireccionará primero la que será la salida estándar de la orden «`find /etc 2>&1 > /dev/null`» a la tubería. A continuación procesará la orden y, antes de ejecutar `find /etc`, redireccionará la salida estándar de error al mismo fichero al que apunta la salida estándar (es decir, a la tubería) y después redireccionará otra vez la salida estándar, lo que provocará que deje de estar conectada a la tubería. En este ejemplo, la salida estándar se redirecciona al fichero `/dev/null`, descartándola así por completo (evidentemente, también se podría haber guardado en un fichero regular cualquiera para procesarla posteriormente). El resultado final es que únicamente la salida estándar de error de la orden `find` es la que se envía a la tubería.

La tubería es quizás el tipo de redirección más importante, puesto que se usa para integrar diferentes órdenes y programas, mediante la interconexión de sus entradas y salidas estándares. Más concretamente, con una *tubería* o *pipe* (símbolo |) hay varias órdenes que se ejecutan de manera interrelacionada, de forma que la salida estándar de la primera orden se envía a la entrada estándar de la segunda, la salida estándar de la segunda orden se envía a la entrada estándar de la tercera, y así sucesivamente:

```
$ orden 1 | orden 2 | ... | orden n
```

Las siguientes secciones de este boletín describen la funcionalidad de diversas órdenes para el procesamiento y el filtrado de información. A estas órdenes se les llama *filtros* porque son capaces de «filtrar» la información que reciben para seleccionar y/o procesar solo aquella información que nos interesa. Es más, puesto que estas órdenes pueden recibir la información por su entrada estándar y mostrar el resultado por su salida estándar, pueden colocarse en cualquier lugar de una tubería para filtrar la información que reciben de la orden que queda a su izquierda en la tubería antes de pasar dicha información a la orden que queda a su derecha.

3. Orden `grep`

La orden `grep` constituye una útil herramienta para buscar cadenas de texto en ficheros. Su sintaxis es:

```
grep [opción]... patrón [fichero]...
```

Por defecto, esta orden realiza una búsqueda del patrón de texto especificado en uno o más ficheros, mostrando todas las líneas de texto de estos ficheros que contienen dicho patrón. Si se especifica más de un fichero, esta orden muestra justo antes de cada línea encontrada el nombre del fichero al que pertenece. Ejemplos:

```
$ grep calle datos/clientes
calle mayor, 12, murcia
calle europa, 22, alicante
calle verde, 33, almeria

$ grep calle datos/*
datos/clientes: calle mayor, 12, murcia
datos/clientes: calle europa, 22, alicante
datos/clientes: calle verde, 33, almeria
datos/proveedores: calle roja, 11, madrid
datos/proveedores: calle asia, 9, barcelona
```

Si no se especifica ningún fichero, esta orden toma la entrada estándar del sistema como entrada de datos, lo cual será bastante útil para su uso en tuberías.

Las opciones más comunes de la orden `grep` son:

- `-i`: ignora las diferencias entre mayúsculas y minúsculas, las considera equivalentes.
- `-n`: muestra las líneas y el número de cada línea.
- `-c`: muestra la cuenta de líneas coincidentes, pero no las líneas en sí.
- `-l`: muestra los nombres de los ficheros con líneas coincidentes, pero no las líneas en sí.
- `-h`: muestra las líneas coincidentes, pero no los nombres de ficheros.
- `-v`: muestra las líneas que no coinciden.
- `-w`: muestra las líneas que contienen el patrón como una palabra completa.
- `-x`: muestra las líneas que coinciden completamente con el patrón buscado.
- `-o`: muestra solo las partes coincidentes de una línea (no líneas coincidentes enteras) mostrando cada parte coincidente en una línea separada.

En los patrones de búsqueda se pueden utilizar expresiones regulares[2], construida mediante ciertos caracteres especiales. Los más habituales son:

- `.`: un único carácter cualquiera.
- `c*`: cero o más ocurrencias del carácter indicado, `c`.
- `c\+`: una o más ocurrencias del carácter indicado, `c`.
- `^`: comienzo de línea.
- `$`: final de línea.
- `[...]`: uno de entre un conjunto de caracteres. En el interior de esta expresión, el carácter «`^`» tiene el significado de negación cuando aparece al principio, justo detrás de «[». Al igual que en Bash, es posible indicar rangos de caracteres y clases de caracteres.
- `c\{n,m\}`: entre n y m repeticiones del carácter indicado, `c`.
- `\c`: para deshabilitar el significado especial del carácter `c` que se indique a continuación.

Puesto que algunos de estos caracteres especiales tienen también un significado especial para Bash, cuando los usemos en un patrón, deberíamos encerrar dicho patrón entre comillas simples para evitar que Bash los interprete y así lleguen intactos a la orden `grep`. Hablaremos más de estos mecanismos de protección de Bash en el siguiente boletín.

[2]Una expresión regular es una serie de caracteres que forman un patrón, normalmente representativo de otro grupo de caracteres mayor. Por ejemplo, el grupo formado por las cadenas Handel, Händel y Haendel se describe mediante el patrón `'H[aäae]ndel'`.

3.1. Ejemplos de uso

A lo largo de los siguientes ejemplos se utilizarán algunos ficheros de texto cuyo contenido podría ser el que se muestra a continuación, aunque podrías crear tú mismo estos ficheros con otro contenido que estimes oportuno.

```
$ cat compras/compras_nacionales
manzanas Lleida 200 Kilogramos
peras Zaragoza 150 kilogramos
manzanas Girona 10 kilogramos
naranjas Valencia 500 Kilogramos
uvas Alicante 1000 Kilogramos
uvas Logroño 300 Kilogramos

$ cat compras/compras_europeas
manzanas Italia 1000 kilogramos
uvas Francia 500 Kilogramos

$ cat compras/compras_extracomunitarias
manzanas Chile 2000 Kilogramos
peras Argentina 1500 Kilogramos
manzanas Argentina 100 kilogramos
naranjas Chile  500 kilogramos
uvas Marruecos 1000 Kilogramos
uvas Argelia 3000 Kilogramos

$ cat compras/proveedores
Chile Frutosa 34
Argentina Agricolasa
Francia BFF 54
Italia FDI
```

1. Busca la cadena «manzanas» en el fichero `compras/compras_nacionales`, mostrando cada línea que contenga esta cadena.

   ```
   $ grep manzanas compras/compras_nacionales
   ```

2. Busca la cadena «manzanas» en todos los ficheros del directorio `compras`.

   ```
   $ grep manzanas compras/*
   ```

3. Lista los nombres de todos los ficheros del directorio `compras` que contienen la cadena «naranjas». Se debe listar los nombres de los ficheros, pero no las líneas de estos que contienen la cadena buscada.

   ```
   $ grep -l naranjas compras/*
   ```

4. Busca la cadena «peras» en todos los ficheros que empiezan por `compras` del directorio `compras`, ignorando la distinción entre mayúsculas y minúsculas.

   ```
   $ grep -i peras compras/compras*
   ```

5. Busca la cadena «peras» en todos los ficheros que empiezan por `compras` del directorio `compras`, ignorando la distinción entre mayúsculas y minúsculas. Muestra las líneas que contengan esta cadena, pero no los nombres de los ficheros.

```
$ grep -i -h peras compras/compras*
# o bien
$ grep -ih peras compras/compras*
```

6. Busca la cadena «1500 Kilogramos» en todos los ficheros que empiezan por `compras` del directorio `compras`. Muestra las líneas que contengan esta cadena y sus respectivos números de línea dentro del fichero.

   ```
   $ grep -n '1500 Kilogramos' compras/compras*
   ```

   Observa que, en este caso, el patrón es una cadena que contiene espacios en blanco. Como Bash considera que los espacios en blanco separan los diferentes elementos de una orden (nombre de la orden, argumentos, opciones, etc.), debemos usar comillas simples para proteger el espacio del patrón y evitar así que Bash lo tome como un carácter separador.

7. Muestra todas las líneas del fichero `compras/compras_extracomunitarias` que no contienen la cadena «uvas».

   ```
   $ grep -v uvas compras/compras_extracomunitarias
   ```

8. Cuenta el número de veces que aparece la cadena «manzanas» en cada uno de los ficheros del directorio `compras`.

   ```
   $ grep -c manzanas compras/*
   ```

9. Muestra los datos de todas las compras de frutas que se hayan realizado en cantidades de 10, 100, 1000, ... kilogramos.

   ```
   $ grep '100* [Kk]ilogramos' compras/*
   ```

10. Busca todas las líneas del fichero de proveedores que contengan un dígito.

    ```
    $ grep '[0-9]' compras/proveedores
    ```

11. Muestra los datos de todas las compras de frutas que se hayan realizado en cantidades múltiplo de 1000 kilogramos, mayores de 1000 kilogramos y menores de 10000 kilogramos.

    ```
    $ grep '[^1]000 [Kk]ilogramos' compras/*
    ```

12. Muestra todas las líneas del fichero `compras/compras_nacionales` que empiecen con una letra «m».

    ```
    $ grep ^m compras/compras_nacionales
    ```

13. Encuentra todos los ficheros del directorio `compras` que acaben en `les`.

    ```
    $ ls -l compras | grep les$
    ```

14. Muestra todos los subdirectorios del directorio actual que tengan permiso de ejecución para otros usuarios.

    ```
    $ ls -l | grep d........x
    ```

4. Orden `sort`

Esta orden sirve para ordenar las líneas de un fichero alfabéticamente o numéricamente. Puede considerar como clave de ordenación cada línea del fichero al completo, o bien, uno o más campos de cada línea, como veremos después. Su sintaxis es:

```
sort [opción]... [fichero]...
```

4.1. Ordenaciones sencillas

Por defecto, `sort` ordena alfabéticamente las líneas de los ficheros. Si no se especifica ningún fichero, entonces ordena las líneas que recibe por su entrada estándar.

Opciones de ordenación más comunes:

- `-n`: ordena numéricamente, incluyendo números negativos y con decimales. En el caso de los números con decimales, hay que tener en cuenta el idioma usado en el terminal, ya que en inglés el separador de decimales es el punto «.» y en español la coma «,». También hay que tener en cuenta que `sort` considera que un número termina cuando aparece el primer carácter no numérico, incluyendo los espacios en blanco.
- `-r`: orden inverso.

Por ejemplo, en orden alfabético:

```
$ cat f1
.this line begins with a period
a line that begins with lowercase a.
This is a line.
abracadabra
1234
Where will this line sort?
A line that begins with uppercase a.

$ sort f1
1234
abracadabra
a line that begins with lowercase a.
A line that begins with uppercase a.
This is a line.
.this line begins with a period
Where will this line sort?
```

En orden numérico:

```
$ cat n1
-18
18
0
-1,4
0,54
0,0
3
0,1

$ sort -n n1
-18
-1,4
```

```
0
0,0
0,1
0,54
3
18
```

4.2. Ordenaciones por columnas

Con `sort` también se puede ordenar el contenido de un fichero de acuerdo a las columnas (campos) de sus líneas, usando la opción `-k`. Por ejemplo:

```
$ cat fic
Susan Jones
Jill Zane
John Smith
Andrew Carter

$ cat fic_comas
Susan,Jones
Jill,Zane
John,Smith
Andrew,Carter

$ cat fic_largo
Susan Jones 123 Plaza Mayor
Jill Zane 34 Calle Real
John Smith 455 Calle Libertad
Andrew Carter 344 Plaza Agosto
```

La siguiente ejecución de `sort` ordenará el fichero `fic_largo` a partir del cuarto campo inclusive:

```
$ sort -k4 fic_largo
John Smith 455 Calle Libertad
Jill Zane 34 Calle Real
Andrew Carter 344 Plaza Agosto
Susan Jones 123 Plaza Mayor
```

mientras que la siguiente ordenará solo por el cuarto campo (observa que el resultado no coincide con el de la orden anterior):

```
$ sort -k4,4 fic_largo
Jill Zane 34 Calle Real
John Smith 455 Calle Libertad
Andrew Carter 344 Plaza Agosto
Susan Jones 123 Plaza Mayor
```

Esta opción se puede combinar con las opciones de ordenación general anteriormente descritas, como `-n` y `-r`. En ese caso, estas opciones serán las usadas por defecto para la ordenación de los campos, salvo que se indique otra cosa para cada campo, como veremos ahora.

De igual forma, se puede especificar el carácter que marca la separación entre los campos con la opción `-t`. Si no se usa esta opción, el separador por omisión es la *cadena vacía* que hay en una transición desde un carácter «no blanco» (letra, número, etc.) a un carácter «blanco» (espacio o tabulador, principalmente).

También podemos repetir la opción `-k` con campos distintos y así realizar la ordenación según cierto orden de prioridad entre los campos.

Veamos algunos ejemplos:

- Ordena el fichero `fic` numéricamente empezando la clave de ordenación por el primer carácter del primer campo y continuando hasta el final de cada línea. O sea, es equivalente a una ordenación general numérica.

```
$ sort -k 1n fic
```

- Ordena el fichero `fic_comas` únicamente por el primer campo, tomando el carácter «,» como el delimitador entre los campos.

```
$ sort -k 1,1 -t, fic_comas
```

- Ordena el fichero `fic_largo` usando como clave los campos del 3 al 5 y a continuación el campo 2.

```
$ sort -k 3,5 -k 2,2 fic_largo
```

Uso de la opción `-k` en `sort`

Observa que `-k3,5` no es lo mismo que `-k3,3 -k4,4 -k5,5`. En el primer caso, hay una única clave de ordenación formada por los campos del 3 al 5 de cada línea, *incluyendo* todos los espacios y tabuladores que formen parte de esos campos o, si se ha usado la opción `-t`, todos los separadores que haya entre campos. En el segundo caso, se ordena por el campo tres de cada línea; *solo si hay empate*, se usa también el campo cuatro para ordenar; y, *solo si hay empate de nuevo*, se usa el campo cinco. Por lo tanto, hay tres claves de ordenación, y no solo una. A continuación tienes un ejemplo que muestra las diferencias.

Para comprobar que no es lo mismo usar varios campos conjuntamente que usarlos individualmente, supongamos el siguiente fichero, donde los campos vienen separados por el carácter «:»:

```
$ cat pruebaopcionk
ll:jj:cc:b:ee
mm:xx:cc:a:gg
mm:xx:cc:aa:pp
```

La ordenación utilizando los campos 3, 4 y 5 conjuntamente producirá la siguiente ordenación:

```
$ sort -t: -k3,5 pruebaopcionk
mm:xx:cc:aa:pp
mm:xx:cc:a:gg
ll:jj:cc:b:ee
```

ya que la cadena «cc:aa:pp» va lexicográficamente antes que la cadena «cc:a:gg» y ésta antes que «cc:b:ee». Sin embargo, si se usan los campos individualmente, el resultado será este otro:

```
$ sort -t: -k3,3 -k4,4 -k5,5 pruebaopcionk
mm:xx:cc:a:gg
mm:xx:cc:aa:pp
ll:jj:cc:b:ee
```

porque, al haber empate en el tercer campo (cadena «cc»), se consulta el cuarto campo, y aquí no hay duda de que la cadena «a» va antes que «aa» y esta antes que «b». Al no haber empate en el cuarto campo, no es necesario consultar el quinto.

4.3. Ejemplos de uso

A lo largo de los siguientes ejemplos se utilizarán algunos ficheros de texto cuyo contenido podría ser el que se muestra a continuación, aunque podrías crear tú mismo estos ficheros con otro contenido que estimes oportuno.

```
$ cat empleados
pepe lopez 123 calle mayor
juan gutierrez 22 calle quevedo
pepe lopez 23 calle quijote
juan gutierrez 22 calle lope
juan gutierrez 3 calle cervantes

$ cat empleados2puntos
pepe:lopez:123:calle:mayor
juan:gutierrez:22:calle:quevedo
pepe:lopez:23:calle:quijote
juan:gutierrez:22:calle:lope
juan:gutierrez:3:calle:cervantes
```

1. Ordena el fichero `empleados` por el primer campo, si este coincide entonces por el tercer campo y, finalmente, si este también coincide, por el quinto campo.

   ```
   $ sort -k 1,1 -k 3,3 -k 5,5 empleados
   ```

2. Ordena el fichero `empleados` por el primer campo usando orden alfabético, si este coincide entonces por el tercer campo usando orden numérico.

   ```
   $ sort -k 1,1 -k 3,3n empleados
   ```

3. Ordena el fichero `empleados2puntos` por el tercer campo usando orden alfabético, teniendo en cuenta que la separación entre campos está marcada por el carácter «:».

   ```
   $ sort -t: -k 3,3 empleados2puntos
   ```

5. Orden `tr`

La sintaxis de la orden `tr` es:

```
tr [opción]... conjunto1 [conjunto2]
```

Esta orden copia el texto desde su entrada estándar, reemplazando los caracteres indicados en `conjunto1` por los caracteres correspondientes de `conjunto2`, reemplazando múltiples ocurrencias de caracteres de `conjunto1` por un único carácter, o eliminando los caracteres de `conjunto1`, dependiendo de las opciones usadas en casa caso. El resultado final lo muestra en su salida estándar. Por ejemplo:

```
$ tr abc xyz < infile > outfile
```

sustituye el carácter «a» por «x», el «b» por «y» y «c» por «z» en el fichero `infile`, guardando los resultados en `outfile`. No requiere que los caracteres «abc» aparezcan juntos como un patrón en `infile`. Esto se ve claramente en el siguiente ejemplo:

```
$ echo La ballena estaba cansada | tr abc xyz
Lx yxllenx estxyx zxnsxdx
```

La orden `echo` simplemente escribe a su salida estándar los parámetros que recibe y, gracias a la tubería, dichos parámetros llegan a la orden `tr` que hace las sustituciones indicadas.

Las opciones de uso más frecuentes son:

- `-s`: elimina repeticiones contiguas de los caracteres especificados en `conjunto1`.
- `-d`: elimina los caracteres especificados en `conjunto1`.
- `-c`: utiliza el conjunto complementario de caracteres de `conjunto1`, es decir, todos los caracteres excepto los indicados en `conjunto1`.

Para indicar `conjunto1` y `conjunto2` se pueden utilizar también ciertas especificaciones especiales:

- `x-y`: para especificar un rango de caracteres desde el carácter «x» hasta el «y».
- `\c`: para especificar una secuencia de escape, donde «c» puede ser el carácter «n» para indicar un salto de línea (`\n`), el carácter «t» para indicar un tabulador (`\t`), ...
- `[:class:]`: para especificar una clase de caracteres, por ejemplo, los caracteres alfanuméricos, `[:alnum:]`, los dígitos, `[:digit:]`, los caracteres de espaciado horizontal, `[:blank:]`, ...
- `[c*n]`: para indicar `n` repeticiones del carácter `c`.

Recuerda que algunos de los caracteres usados en estas especificaciones, como «\», «[» y «*», son especiales para Bash, por lo que tendrás que encerrarlos entre comillas simples para evitar que Bash los interprete y así puedan llegar a la orden `tr`.

5.1. Ejemplos de uso

A lo largo de los siguientes ejemplos se utilizarán algunos ficheros de texto cuyo contenido podría ser el que se muestra a continuación, aunque podrías crear tú mismo estos ficheros con otro contenido que estimes oportuno.

```
$ cat empleadostabulados
pepe lopez     123 calle mayor
juan gutierrez 22  calle quevedo
pepe lopez     23  calle quijote
juan gutierrez 22  calle lope
juan gutierrez 3   calle cervantes
```

1. Muestra en pantalla el contenido del fichero `empleadostabulados` comprimiendo todos los espacios en blanco consecutivos en uno único.

   ```
   $ cat empleadostabulados | tr -s ' '
   ```

2. Muestra en pantalla el contenido del fichero `empleadostabulados` cambiando cualquier conjunto de espacios en blanco consecutivos en un único tabulador.

   ```
   $ cat empleadostabulados | tr -s ' ' '\t'
   ```

3. Muestra en pantalla el contenido del fichero `empleadostabulados` cambiando todas las letras minúsculas por mayúsculas.

```
$ cat empleadostabulados | tr '[a-z]' '[A-Z]'
```

o bien así:

```
$ cat empleadostabulados | tr '[:lower:]' '[:upper:]'
```

4. Muestra en pantalla el contenido del fichero `empleadostabulados` cambiando todos los caracteres que no pertenezcan a la clase «espacios en blanco» por la letra «x».

```
$ cat empleadostabulados | tr -c '[:space:]' x
```

5. Completa la tubería

```
echo Secreto a voces es un oxímoron | tr ...
```

para que cada palabra aparezca en una línea distinta.

```
echo Secreto a voces es un oxímoron | tr ' ' '\n'
```

6. Orden `cut`

La orden `cut` sirve para seleccionar columnas de un fichero de texto y mostrarlas en la salida estándar. Si no se especifica ningún fichero, toma su entrada estándar. Las columnas se pueden especificar en bytes, caracteres o campos, indicando qué delimitadores concretos queremos manejar para diferenciarlas. Su sintaxis es:

```
cut opción... [fichero]...
```

Las opciones más comunes son:

- `-c`: las columnas son especificadas por posiciones de caracteres.
- `-f`: las columnas son especificadas por campos. Por defecto, el delimitador es el tabulador.
- `-d`: usada junto a la opción `-f` para indicar un delimitador específico distinto del tabulador para los campos.
- `-s`: usada junto a la opción `-f` para indicar que, si una línea no contiene ningún delimitador, no debe mostrarse en la salida.

A la hora de indicar los caracteres o campos a mostrar de cada línea, se pueden indicar varios caracteres o campos, separados por comas, por ejemplo, `-c1,2,3` o `-f3,5`. También se pueden indicar rangos, por ejemplo, `-c1-8` o `-f1-3,5`.

El separador de campos en la orden `cut`

Hay dos consideraciones importantes respecto a la orden `cut`. La primera es que, como hemos dicho, el carácter separador de campos por defecto es el tabulador. Sin embargo, es raro encontrar dicho carácter separador en ficheros y en salidas de otras órdenes. Por eso, es importante no olvidar el usar la opción `-d` cuando se use la opción `-f`. La otra consideración es que `cut` no considera un mismo carácter separador repetido como uno solo, sino que considera cada ocurrencia individualmente. Los siguientes ejemplos muestran estas consideraciones.

Para entender mejor el funcionamiento del separador de campos en `cut`, observa el resultado de la siguiente orden (hemos usado comillas simples para evitar que Bash interprete los espacios en blanco):

```
$ echo 'campo1    campo2' | cut -f2 -d ' '

$
```

Como vemos, no se la seleccionado el segundo campo, como cabría esperar, sino el segundo espacio en blanco tras «campo1». Si queremos obtener el resultado esperado, debemos usar la orden `tr` para eliminar repeticiones del carácter separador. Observa el resultado de las siguientes órdenes:

```
$ echo 'campo1    campo2' | tr -s ' '
campo1 campo2
$ echo 'campo1    campo2' | tr -s ' ' | cut -f2 -d ' '
campo2
```

Por eso, es importante no olvidar el uso de la orden `tr` antes de una orden `cut` cuando se usa la opción `-f` de esta última.

6.1. Ejemplos de uso

A lo largo de los siguientes ejemplos se utilizarán algunos ficheros de texto cuyo contenido podría ser el que se muestra a continuación, aunque podrías crear tú mismo estos ficheros con otro contenido que estimes oportuno.

```
$ cat dataset1
Pine 906 26 1.0 211
Beech 933 26 2.3 160
Fur 1246 27 2.44 162
Palm 671 25 3.8 888

$ cat dataset3
Trees of the Forest
Pine,906,26,020079,130.0,80.3,17.1,211
Beech,933,26,030079,48.0,85.2,22.7,160
Fur,1246,27,070079,31.0,86.5,6.9,162
Palm,671,25,100077,41.0,87.3,15.0,888
```

1. Corta los caracteres del 13 al 17 del fichero `dataset1`.

   ```
   $ cut -c 13-17 dataset1
   ```

2. Corta los caracteres 4, 5 y del 11 al 15 del fichero `dataset1`.

   ```
   $ cut -c 4,5,11-15 dataset1
   ```

3. Corta los campos 1, 3, 4, 6 y 8 del fichero `dataset3`, teniendo como delimitador de campos el carácter «,». En la solución final no debe aparecer la línea «Trees of the Forest» al no contar con ningún carácter delimitador.

   ```
   $ cut -d, -f 1,3,4,6,8 -s dataset3
   ```

4. Lista los 8 primeros caracteres de los nombres de todos los ficheros regulares del directorio actual.

```
$ find -maxdepth 1 -type f -printf '%f\n' | cut -c1-8
```

5. Usando la salida de la orden `ps aux`, que lista todos los proceso existentes en el sistema en el momento de su ejecución, obtén los PIDs de todos los procesos (este dato es el segundo campo de cada línea mostrada por `ps aux`).

```
$ ps aux | tr -s ' ' | cut -f2 -d ' '
```

Consejo para construir tuberías

Mediante tuberías, es posible combinar un gran número de órdenes para obtener los datos deseados, como acabamos de ver en este último ejemplo. Al menos en un primer momento, intentar escribir una tubería completa, con todas las órdenes necesarias y sus correspondientes opciones y argumentos, puede ser bastante difícil. Por eso, intenta construir la tubería que necesites orden a orden, comprobando en cada paso si la salida producida es la que esperas. Además, en el caso de que la salida sea bastante larga, conviene que al ir construyendo la tubería uses al final la orden `less`, para ver cómodamente toda la salida en cada paso y así comprobar si es correcta o no.

7. Orden `uniq`

La orden `uniq` sirve para eliminar líneas duplicadas de un fichero, siempre que estas aparezcan juntas. Por eso, para poder llevar a cabo su labor, el fichero se suele ordenar previamente. Su sintaxis es:

```
uniq [opción]... [fichero_entrada [fichero_salida]]
```

Si no se indica fichero de entrada o de salida, se utilizará la entrada estándar o la salida estándar, respectivamente. Observa que, según la sintaxis, no es posible indicar un fichero de salida si no se indica un fichero de entrada.

Las opciones más importantes de `uniq` son:

- `-c`: escribe a la salida el número de veces que aparece cada línea en el fichero de entrada justo antes de escribir la propia línea.
- `-d`: cada línea duplicada es escrita a la salida solamente una vez. No escribe ninguna línea que no estuviera duplicada en la entrada.
- `-D`: igual que `-d`, pero sin sustituir las líneas duplicadas por una sola.
- `-u`: escribe a la salida únicamente las líneas no duplicadas en la entrada.
- `-i`: no distingue entre mayúsculas y minúsculas a la hora de comparar.
- `-s N`: no tiene en cuenta los `N` primeros caracteres de cada línea a la hora de comparar.
- `-w N`: solamente tiene en cuenta los `N` primeros caracteres de cada línea a la hora de comparar.

Por ejemplo, si partimos del fichero `errlog` con todos los errores que se han ido produciendo en un ordenador durante la última sesión:

```
$ cat errlog
error 11: /tmp directory not found
error 22: out of memory
error 11: /tmp directory not found
error 17: low disk space
error 11: /tmp directory not found
error 22: out of memory
error 04: connection failure
error 11: /tmp directory not found
```

podemos obtener, con la ayuda de la orden `sort`, un listado con los tipos de errores de esta manera:

```
$ sort errlog > serrlog

$ cat serrlog
error 04: connection failure
error 11: /tmp directory not found
error 11: /tmp directory not found
error 11: /tmp directory not found
error 11: /tmp directory not found
error 17: low disk space
error 22: out of memory
error 22: out of memory

$ uniq serrlog > uerrlog

$ cat uerrlog
error 04: connection failure
error 11: /tmp directory not found
error 17: low disk space
error 22: out of memory
```

Evidentemente, también podríamos haber escrito simplemente lo siguiente para obtener el mismo resultado, en el caso de no necesitar los ficheros intermedios:

```
$ sort errlog | uniq
```

Combinando esta orden con las órdenes `cut` y `sort` podemos obtener la contabilidad resumida de los códigos de error producidos:

```
$ cut -f1 -d ':' serrlog
error 04
error 11
error 11
error 11
error 11
error 17
error 22
error 22
# Con «cut» cortamos el primer campo,
# tomando como delimitador entre campos el carácter «:».

$ cut -f1 -d ':' serrlog | sort | uniq -c
1 error 04
4 error 11
```

```
1 error 17
2 error 22
# En este caso, «sort» no habria sido necesario
# porque las filas iguales están ya consecutivas.
```

7.1. Ejemplos de uso

Partimos del fichero `purchases`, que contiene una lista de compras realizadas. Cada línea contiene el nombre y apellido de un cliente, la fecha de la compra (mes y día) y el código del artículo comprado («Unit 12», «Unit 05», ...).

```
$ cat purchases
Jimmy James Jan 2 Unit 12
Jane Doe Jan 4 Unit 17
Jimmy James Jan 10 Unit 12
John Huber Jan 15 Unit 17
Sue Butler Jan 2 Unit 05
Jane Doe Jan 10 Unit 12
Liz Tyler Feb 2 Unit 04
Jimmy James Feb 4 Unit 03
```

1. Genera una lista de cuántas unidades se han vendido de cada artículo.

```
$ cut -d ' ' -f5,6 purchases
    Unit 12
    Unit 17
    Unit 12
    Unit 17
    Unit 05
    Unit 12
    Unit 04
    Unit 03

$ cut -d ' ' -f5,6 purchases | sort
    Unit 03
    Unit 04
    Unit 05
    Unit 12
    Unit 12
    Unit 12
    Unit 17
    Unit 17

$ cut -d ' ' -f5,6 purchases | sort | uniq -c
    1 Unit 03
    1 Unit 04
    1 Unit 05
    3 Unit 12
    2 Unit 17
```

2. Genera una lista de clientes, guardándola en el fichero «customers».

```
$ cut -d ' ' -f1,2 purchases | sort | uniq > customers
```

```
# Con «cut» cortamos los campos 1º y 2º del fichero «purchases»
# Usando el espacio como delimitador de campos.
# Con «sort» ordenamos el resultado.
# Con «uniq» eliminamos líneas repetidas.
# Finalmente, la salida se escribe en el fichero «customers».

$ cat customer
    Jane Doe
    Jimmy James
    John Huber
    Liz Tyler
    Sue Butler
```

3. Genera una lista de clientes que han realizado más de una compra.

```
$ cut -d ' ' -f 1,2 purchases | sort | uniq -d
Jane Doe
Jimmy James
```

4. Genera una lista del número de artículos vendidos cada día.

```
$ cut -d ' ' -f 3,4 purchases | sort | uniq -c
```

5. Genera una lista de aquellos artículos de los que se han vendido más de 1 unidad.

```
$ cut -d ' ' -f 5,6 purchases |sort | uniq -d
```

8. Miscelánea

A continuación mostramos de forma resumida algunas otras órdenes que pueden ser útiles como filtros. Como siempre, podemos obtener más información sobre cada una de ellas consultando sus páginas de manual.

8.1. head

Muestra las primeras líneas de un fichero[3]. Su sintaxis es:

```
head [opción]... [fichero]...
```

Sin opciones, `head` muestra las primeras 10 líneas de los ficheros dados como parámetros. Si no se indica ningún fichero, muestra las primeras 10 líneas que recibe por su entrada estándar. La opción más importante de `head` es:

- `-n, --lines=[-]NUM`: muestra las primeras `NUM` líneas en lugar de las 10 primeras. Si el número va precedido por un signo menos «-», muestra todas las líneas excepto las últimas `NUM`.

Por ejemplo, podemos listar los nombres de las 10 primeras entradas del directorio actual ejecutando:

```
$ ls | head
```

[3]Con la opción adecuada también podrían ser los primeros bytes, aunque nosotros no necesitaremos hacer esto.

8.2. tail

Muestra las últimas líneas de un fichero. Su sintaxis es:

```
tail [opción]... [fichero]...
```

Sin opciones, `tail` muestra las últimas 10 líneas de los ficheros dados como parámetros. Si no se indica ningún fichero, muestra las últimas 10 líneas que recibe por su entrada estándar. La opción más importante de `tail` es:

- `-n, --lines=[+]NUM`: muestra las últimas `NUM` líneas en lugar de las 10 últimas. Si el número va precedido por un signo más «+», muestra a partir de la línea `NUM` (es decir, muestra todas las líneas excepto las `NUM-1` primeras).

Por ejemplo, podemos listar los nombres de las entradas del directorio actual, descartando la primera línea, con la siguiente orden:

```
$ ls -l | tail -n +2
```

8.3. column

Muestra una lista de elementos en columnas. La sintaxis de esta orden es:

```
column [opción]... [fichero]...
```

Sin opciones, `column` va leyendo en orden las líneas de los ficheros dados como parámetros (primero todas las líneas del primer fichero, luego todas las del segundo fichero, etc.) y las va mostrando en columnas, rellenando las columnas antes que las filas. El número de columnas dependerá del ancho de la terminal (es decir, del número de caracteres que se pueden mostrar en cada línea) y de la longitud máxima de las líneas de entrada. Se intentará que las columnas queden balanceadas, es decir, que todas tengan el mismo número de filas. Si no es posible, la última columna tendrá menos filas que el resto.

Como es habitual, si no se indica ningún fichero, `column` toma las líneas de su entrada estándar. Las opciones más importantes de `column` son las siguientes:

- `-t, --table`: determina el número de columnas que contiene la entrada y crea una tabla. Por defecto, se considera que las columnas están separadas por espacios en blanco, aunque se puede cambiar el delimitador con la siguiente opción.
- `-s, --separator separadores`: especifica los posibles separadores o delimitadores de columnas en las líneas de entrada. Por defecto, el separador es el espacio en blanco.

Así, si queremos listar el nombre (incluyendo la ruta) y el tamaño de cada fichero regular acabado en `.sh` que hay a partir del directorio actual, alineando las diferentes columnas de datos, podemos ejecutar la siguiente orden:

```
$ find -type f -name '*.sh' -printf 'Fichero: %p Tamaño: %s Bytes\n' | column -t
```

8.4. wc

Muestra el total de líneas, palabras y bytes para cada fichero dado. Su sintaxis es:

```
wc [opción]... [fichero]...
```

Sin opciones, `wc` muestra el total de líneas, palabras y bytes (en ese orden) que hay en cada fichero dado como argumento. Cuando el número de ficheros es dos o más, también muestra una última línea con el total de líneas, palabras y bytes en todos los ficheros. Si no se indica ningún fichero, el conteo se hace sobre la información recibida por la entrada estándar.

Las opciones más importantes de `wc` son:

- `-c, --bytes`: muestra el total de bytes.
- `-m, --chars`: muestra el total de caracteres. Observa que un carácter puede tener un tamaño de uno o más bytes, por lo que esta opción es distinta de la opción «`-c`» anterior.
- `-l, --lines`: muestra el total de líneas.
- `-L, --max-line-length`: muestra la longitud de la línea más larga.
- `-w, --words`: muestra el total de palabras. `wc` considera que una palabra es una secuencia de uno o más caracteres delimitada por espacios en blanco.

Así, si queremos contar el número de subdirectorios que hay a partir del directorio actual, podemos ejecutar:

```
find . -type d | wc -l
```

Es decir, primero listamos todos los directorios y luego contamos el número de líneas de ese listado.

8.5. `tee`

Lo que lee de la entrada estándar, lo escribe en la salida estándar y en los ficheros dados. Su sintaxis es:

```
tee [opción]... [fichero]...
```

Sin opciones, `tee` lee de su entrada estándar y lo que lee lo escribe tanto en su salida estándar como en todos los ficheros datos como argumentos, creando dichos ficheros si no existen. Por lo tanto, tras la ejecución de esta orden, todos los ficheros tendrán exactamente la misma información (serán copias idénticas). Si no se indica ningún fichero, `tee` simplemente copia lo que recibe por su entrada estándar en su salida estándar. La opción más importante de `tee` es:

- `-a, --append`: los ficheros datos como argumentos que ya existen no se sobrescriben. En su lugar, `tee` añade la información que recibe por su entrada estándar al final de los mismos.

Por ejemplo, si queremos contar cuántos ficheros regulares hay con extensión `.odt` a partir del directorio actual y, además, guardar los nombres de esos ficheros en el fichero `listado`, debemos ejecutar la orden:

```
find . -type f -name '*.odt' | tee listado | wc -l
```

9. Bibliografía

- Páginas de manual de las diferentes órdenes descritas y del intérprete de órdenes Bash.
- *Shell & Utilities: Detailed Toc.* The Open Group Base Specifications. http://pubs.opengroup.org/onlinepubs/9699919799/utilities/contents.html.
- *Linux: Domine la administración del sistema*, 2ª edición. Sébastien Rohaut. ISBN 9782746073425. Eni, 2012.

10. Ejercicios propuestos

Los siguientes ejercicios deben resolverse mediante una única línea de órdenes. Si la línea debe estar formada por varias órdenes, estas deben ir separadas por tuberías.

1. A partir de la salida de la orden `ps aux`, que muestra información de todos los procesos existentes en el sistema en el momento de su ejecución, selecciona todos aquellos procesos que pertenezcan al usuario `alumno` (columna USER).

2. Modifica la solución anterior para mostrar cuántos procesos hay de `alumno`, pero sin mostrar la información de dichos procesos.

3. Modifica la solución anterior para mostrar cuántos procesos hay de todos los usuarios, menos del usuario `alumno`.

4. Usando las órdenes `find` y `grep`, busca todos los ficheros regulares que hay a partir de `/etc` que contengan la palabra «alumno». Sólo hay que mostrar los nombres de los ficheros y la búsqueda dentro de cada fichero se debe realizar sin distinguir entre mayúsculas y minúsculas.

5. Muestra la línea del fichero `/etc/passwd` que pertenece al usuario `alumno`.

6. Obtén todas las palabras (deben contener solo letras, sin distinguir entre mayúsculas y minúsculas) que aparezcan en la salida de la orden `ps aux`. Muestra cada palabra en una línea distinta. Usa la orden `tail -n +2` en la posición adecuada de la tubería para eliminar la cabecera de la orden `ps aux` (es decir, la primera línea).

7. Modifica la solución del ejercicio anterior para quedarse únicamente con las palabras de 4 letras. La lista debe aparecer ordenada alfabéticamente en orden creciente.

8. Modifica la solución del ejercicio anterior para que, además, las palabras se muestren en columnas.

9. Obtén todos los números enteros positivos, incluyendo el 0, que aparezcan en la salida de la orden `ps aux`. Los números mostrados deben formar «palabras» por sí mismos, es decir, no pueden formar parte de una subcadena. Por ejemplo, el número 2 que hay en `udisks2` no debe aparecer. Muestra cada número en una línea distinta.

10. Modifica la solución del ejercicio anterior para que los números aparezcan ordenados numéricamente de mayor a menor.

11. Ordena la salida de la orden `ps aux` (descartando su cabecera) por consumo de memoria RAM de los procesos (columna RSS; el número que aparece es el consumo en KiB).

12. Obtén la columna RSS de la orden `ps aux` (solo debe aparecer por pantalla el contenido de esta columna, incluyendo su cabecera).

13. Usando la orden `find`, obtén una lista de todos los ficheros regulares que se encuentren en `/etc` y sus subdirectorios ordenada por tamaño de los ficheros. La salida obtenida debe seguir el siguiente formato:

    ```
    fichero:tamaño
    ```

    donde el primer campo `fichero` debe incluir la ruta absoluta o relativa del fichero regular mostrado. Nota: los posibles mensajes de error que pueda producir la orden `find` por falta de permisos no deben aparecer por pantalla.

14. A partir de la salida de la orden `ps aux`, obtén una lista de todos los usuarios propietarios de los procesos existentes. Cada usuario debe aparecer solo una vez y se debe descartar también la cabecera de la orden `ps`.

15. Modifica la solución anterior para que solo aparezcan los usuarios que son propietarios de más de un proceso.

16. Modifica la solución anterior para que los nombres de los usuarios aparezcan en mayúsculas.

17. Modifica la solución anterior para mostrar el total de usuarios, pero no los nombres de los mismos.