GUIÓN 3

SISTEMA DE FICHEROS

Bibliografía

Sarwar, S. M.; Koretsky, R.; Sarwar, S. A. El libro de LINUX. Addison Wesley, 2005.
Sobell, Mark G. Manual práctico de Linux. Comandos, editores y programación Shell. Anaya Multimedia, 2008
Sánchez Prieto, S. UNIX y LINUX. Guía práctica (Tercera edición). Ra-Ma, D.L. 2004.

1. CARACTERES DE REDIRECCIÓN

Generalmente, el "shell" funciona de modo que la entrada estándar de una orden proviene del teclado del terminal y la salida estándar va a la pantalla del terminal. Sin embargo, es posible *redirigir* la entrada y salida estándar a un fichero cualquiera.

1. El carácter (**<**) sirve para redirigir la entrada de un comando desde un fichero determinado, en lugar de la entrada estándar. La redirección de entrada no modifica el fichero de entrada utilizado.

```
Ejemplo:
        wc < f1.txt
```

2. El carácter (**>**) sirve para redirigir la salida de un comando hacia un fichero determinado, en lugar de a la pantalla. Actúa como si fuera una copia, es decir, si el fichero especificado no existe, lo crea, y si ya existe, machaca su contenido con la nueva información (salida del comando).

```
Ejemplos:
       cat f1.txt f2.txt > f3.txt
       ls  -l  >listado
```

3. Los caracteres (**>>**) también sirven para redirigir la salida estándar de un comando hacia un fichero, pero en este caso, la información se añade al final del fichero y no se machaca el contenido anterior del mismo.

```
Ejemplos:
       cat >> f3.txt
       ls  -l  /home >> listado
```

4. El carácter (**|**) sirve para redirigir la salida de un comando como entrada de otro.

```
Ejemplos:
1.  Para visualizar el listado largo de un directorio extenso, se podrían hacer las siguientes
    acciones:
        ls –l /etc >listado
        more listado
        rm listado
        Sin embargo, se puede obtener el mismo resultado sin necesidad de crear ningún fichero
intermedio de la siguiente manera:
        ls -l  /etc  | more

2.  ls –l /etc | wc

3.  ls –l /bin | head -50 | more
```

Una línea de órdenes construida con este carácter se denomina "cauce" o "línea de cauce" (*pipe*). Estas líneas de cauce no están limitadas a dos órdenes, se pueden construir de cualquier longitud, sin más que asociar la salida estándar de una orden a la entrada estándar de la siguiente. A estas órdenes se las suele denominar "filtros" o "tuberías".

2. PERMISOS DE ACCESO A FICHEROS Y DIRECTORIOS

Los permisos de acceso a un fichero o a un directorio se ven con el comando **ls -l**. Dichos permisos de acceso corresponden a los nueve caracteres siguientes al primer carácter, que es el que indica el tipo de fichero (fichero normal, directorio, o fichero especial). Suponiendo permitidos todos los accesos, esos nueve caracteres mostrarían lo siguiente: **rwxrwxrwx**

Usualmente no están todos los permisos habilitados, en cuyo caso aparece un guión en lugar de la letra correspondiente (ej. **rw-r--r--**). Estos caracteres representan a las tres categorías de usuarios que distingue el Linux. Los tres primeros caracteres corresponden al propietario del fichero (normalmente, el que lo creó), los segundos corresponden al grupo primario al que pertenece el propietario del fichero y los últimos corresponden al resto de usuarios del sistema. Para cada una de esas tres categorías, el significado de los caracteres es el mismo:

1er carácter: **r** Permiso de lectura (y copia)
2º carácter: **w** Permiso de escritura (modificación y borrado)
3er carácter: **x** Permiso de ejecución

El orden de esos caracteres es invariable y si en alguno de ellos aparece el guión, significa que el/los usuarios pertenecientes a esa categoría no disponen de ese permiso. Para el caso de los directorios, los permisos tienen un significado especial. El acceso de lectura significa que el usuario tiene permitido ver los nombres de los archivos contenidos en el directorio (con **ls**, por ejemplo). El acceso de escritura significa que se puede crear un fichero nuevo dentro del directorio. Por último, el permiso de ejecución indica que se puede acceder al directorio y lo puede atravesar en la búsqueda de subdirectorios. En algunos casos, pueden aparecer otros caracteres (en el lugar del permiso de ejecución) tales como **l** y **s**. Dichos caracteres indican condiciones especiales de protección en la ejecución de algunos programas.

Modificación de los permisos de acceso.

Se puede modificar los permisos de un fichero o directorio mediante el comando **chmod**, el cual tiene dos modos de funcionamiento: modo simbólico y modo absoluto. Evidentemente, sólo el propietario de un fichero podrá cambiar los permisos del mismo.

1. Modo simbólico

Sirve para cambiar uno o varios de los nueve caracteres de acceso a los ficheros o directorios.

Sintaxis:

chmod *categoría* operador *permiso* [lista_ficheros | lista_directorios]

Dentro de *categoría* tenemos cuatro opciones:

u (= user): permisos del propietario

g (= group) : permisos del grupo

o (= other): permisos del resto

a (= all) : todos

Los operadores son los siguientes:

+ añade un derecho

- retira un derecho

Los permisos son los tres ya nombrados (**r** , **w** , **x**).

Se pueden combinar varias modificaciones de estas separadas por comas y sin dejar espacios en blanco. Por ejemplo si se desea añadir el permiso de lectura a todo el mundo y el permiso de escritura al propietario, teclearíamos la siguiente línea de comando:

$ chmod a+r,u+w nombre_fichero

También se pueden juntar varias categorías o varios permisos. Por ejemplo, si se quisiera añadir el permiso de ejecución para el propietario y el grupo, y se quisiera quitar el permiso de lectura y escritura para el resto de usuarios, se puede poner:

$ chmod ug+x,o-rw nombre_fichero

2. Modo absoluto

De esta forma se cambian de una vez todos los permisos de un fichero o directorio.

Sintaxis: chmod NNN nombre_fichero

Donde NNN representa tres cifras que pueden ir del cero al siete. La primera cifra corresponde al propietario, la segunda al grupo y la tercera al resto de usuarios. Cada una de estas cifras se transforma a binario, originando tres cifras binarias con las que se construyen los bits de acceso al fichero (1-permiso activado, 0-permiso desactivado).

```
Ejemplo:
       Si se desea que los permisos de acceso a un fichero sean:
       r  w  x    r - x    r -  -
       1  1  1    1 0 1    1 0 0
          7        5        4
       El comando para asignar esos permisos sería:
       chmod  754  nombre_fichero
```

Permisos de acceso asignados por defecto.

Cada vez que se crea un fichero o directorio nuevo, el sistema operativo le asigna automáticamente unos permisos de acceso por defecto. Para calcular dichos permisos utiliza lo que se denomina la *máscara*, que es un número de tres cifras que van del cero al siete. Un valor de máscara frecuente es el 022.

El proceso para calcular los permisos por defecto varía ligeramente si se trata de un fichero o de un directorio. Para el caso de los directorios, el sistema realiza la operación XOR (OR exclusivo) bit a bit entre la *máscara* y el 777. El resultado de esa operación (que será una serie de 9 bits) son los permisos de acceso que se aplican al directorio que acabamos de crear (1- permiso activado, 0-permiso desactivado).

```
Ejemplo:
       Si la máscara es 023 (000010011), para calcular cuáles serían los permisos por defecto
asignados a un directorio de nueva creación se realiza la operación XOR con 777 (111111111):
         111111111
         000010011
         _________
         111101100

       Por lo tanto, los permisos serían: rwxr-xr--
```

Para el caso de los ficheros, el proceso es el mismo, salvo que ignora el resultado para los permisos de ejecución en las tres categorías. *Nunca se puede dar por defecto permiso de ejecución para los ficheros*. Con la máscara anterior, los permisos que se asignarían a un fichero de nueva creación serían: **rw-r--r--**

Para consultar la máscara que está usando el sistema se utiliza el comando **umask**.

Sintaxis: umask

3. ENLACES O VÍNCULOS

Supongamos que, en un momento dado, uno o varios usuarios necesitan acceder de una forma u otra a un fichero determinado, del que no les interesa tener una copia a cada uno dentro de su estructura propia de directorios (la que cuelga de su directorio de recepción). En este caso, cualquier referencia al fichero debe incluir su *pathname* completo o el "camino" desde el directorio de trabajo, que podrían llegar a ser muy largos. Aunque puede tener más utilidades, el utilizar un enlace (*link*) nos puede solucionar este problema.

Un enlace a un fichero no es más que otro nombre distinto para referirse al mismo fichero. No se trata de una copia, si se hace una modificación en el fichero, mediante cualquiera de sus nombres (o enlaces), esa modificación es visible desde todos los nombres (enlaces) del fichero, ya que sólo existe una copia física del mismo.

Para hacer un enlace de un fichero se utiliza el comando **ln**.

Sintaxis: ln nombre_fichero nombre_enlace

```
Ejemplo:
        ln  f1  enlace_f1
```

Hay que señalar que no existe ningún tipo de jerarquía entre los enlaces de un fichero. Cuando se borra un enlace, únicamente se borra ese nombre del fichero. El fichero permanece hasta que se borra el último enlace, que provoca su eliminación del disco. No importa que se borre el nombre "origen" del fichero, si hay otro enlace, el fichero permanecerá.

La forma de implementar los enlaces es muy sencilla si se conoce como organiza Linux los ficheros. A cada nuevo fichero se le asocia un identificador único dentro de todo el sistema de ficheros (*inodo*). De esta manera, al crear un enlace, basta con asignar ese identificador al nuevo nombre (en el mismo directorio o en otro). Cuando se realiza la copia de un fichero, los *inodos* son distintos puesto que hay dos copias físicas. Se puede visualizar los *inodos* de los ficheros con la opción **-i** del comando **ls**.

4. EXPRESIONES REGULARES

Las expresiones regulares se usan para buscar/reemplazar texto, manipular cadenas, etc.; usando editores de texto, utilidades del Sistema Operativo o comandos del sistema.

Una **expresión regular** (o patrón de búsqueda) es una expresión que describe un conjunto de cadenas sin enumerar sus elementos. Las expresiones regulares se forman usando literales y metacaracteres. Un *literal* es cualquier carácter ordinal y un *metacarácter* es uno o varios caracteres especiales que tiene un significado único y no son usados como literales en la expresión de búsqueda.

La expresión regular más básica consiste en un solo carácter, por ejemplo **a**.

Las *secuencias de escape* son la forma de usar los metacaracteres como literales en una expresión regular, ya que eliminan el carácter especial de dichos metacaracteres, y consisten en anteponer la barra invertida "\" delante del metacarácter que se desea usar como literal.

A continuación, se van a describir los metacaracteres más usuales:

- El punto "." representa cualquier carácter en esa posición de la expresión regular. Por ejemplo, si se busca en la cadena *La piel del pulgar tocó la pila y el plato* la expresión regular **p.l** se localizarán las cadenas "pil" y "pul". Debido a que el punto representa un solo carácter, las cadenas "pla" y "piel" no se obtendrían de la búsqueda anterior. En el primer caso faltaría el carácter que representa el punto, y en el segundo caso sobraría el carácter *e* pues el punto solo sustituiría a la vocal *i*.

- Los caracteres "\<" y "\>" delimitan una palabra.

Para emplear estos metacaracteres con el comando **grep** (ver apartado 5. Filtros), es necesario que toda la cadena de búsqueda esté entre comillas dobles. Así, si se busca la palabra 1 en el fichero datos, que contiene los 20 primeros números enteros ordenados de forma ascendente en distintas líneas, el comando **grep "\<1\>" datos** proporcionaría solo la primera línea del fichero ya que es en la única línea donde la palabra 1 se encuentra. Existen otras líneas del fichero que contienen el carácter 1 pero no suponen una palabra pues no está delimitado por espacios en blanco.

- Los corchetes "[ ]" permiten definir "clases de caracteres", o sea, agrupar caracteres en grupos o clases. Son útiles cuando es necesario buscar un carácter de un grupo de posibles caracteres. Independientemente del número de caracteres que se encuentren entre corchetes, sólo corresponderá a un carácter del patrón de búsqueda. Por ejemplo, la expresión regular **[0123456789]** emparejará con cualquier cadena que contenga un digito decimal. Otro ejemplo, con la expresión regular **s[aio]l** se identificarían las cadenas "sal", "sil" y "sol".

  Dentro de los corchetes es posible utilizar el guión "-" para especificar rangos de caracteres. Por ejemplo, la expresión regular **[a-z]** representa a un solo carácter en minúscula. Es posible combinar varios rangos de caracteres empleando varias veces el guión, tal como figura en la siguiente expresión regular **[0-9a-fA-F]** que emparejaría con cualquier dígito hexadecimal. Se puede apreciar en el último ejemplo, que entre los distintos rangos de caracteres no se ha empleado ningún otro carácter (ni coma, ni espacio en blanco, etc.).

  Si se necesita emplear el guión como un literal en el grupo de caracteres a localizar, se debe poner al principio o al final de dicho grupo. Por ejemplo, para localizar todas las cadenas que contengan un dígito decimal o el guión, las posibles expresiones regulares a utilizar son **[-0-9]** y **[0-9-]**.

  Dentro de los corchetes, el punto y el dólar representan un carácter literal y no un metacarácter, por lo que no es necesario anteponerles la barra invertida. Por ejemplo, si se busca en la cadena *El final del fin.* la expresión regular **fin[a.]** obtendrá las cadenas "fina" y "fin."

  Si la barra invertida es uno de los caracteres que conforman el posible grupo de caracteres a localizar (es necesario emplear corchetes), ésta debe ir precedida de otra barra invertida. Por lo tanto, para localizar todas las cadenas que contengan la barra invertida o una vocal minúscula, la expresión regular a utilizar sería **[\\aeiou]**.

  El estándar POSIX 1003.2 sección 2.8.3.2 define un conjunto de clases de caracteres que corresponden a los rangos de caracteres más usuales y que se pueden usar a la hora de crear expresiones regulares. La ventaja de emplear estas clases ya predefinidas estriba en que tienen en cuenta la máquina dónde se usan, considerando por lo tanto cualquier variante en el lenguaje o en la codificación del sistema local.

  La siguiente tabla presenta estas clases de caracteres predefinidas:

Valor	Significado
[:alnum:]	Cualquier carácter alfanumérico (letras y dígitos): 0-9, A-Z, a-z
[:alpha:]	Cualquier carácter alfabético (letras): A-Z, a-z
[:digit:]	Cualquier carácter numérico (dígitos): 0-9
[:blank:]	Espacios en blanco y tabulador
[:xdigit:]	Cualquier dígito hexadecimal: 0-9, A-F, a-f
[:punct:]	Signos de puntuación: . , " ' ¿ ¡ ; : # $ % & ( ) * + - / <> = @ [ ] \ ^ _ { } | ~
[:print:]	Cualquier carácter imprimible
[:space:]	Cualquier carácter de espacio en blanco: espacios en blanco, tabulador, NL, FF, VT, CR
[:graph:]	Cualquier carácter gráfico excluyendo el espacio en blanco y el tabulador
[:lower:]	Cualquier carácter alfabético (letras) en minúscula: a-z
[:upper:]	Cualquier carácter alfabético (letras) en mayúscula: A-Z
[:cntrl:]	Caracteres de control: NL CR LF TAB VT FF NUL SOH STX EXT EOT ENQ ACK SO SI DEL DC1 DC2 DC3 DC4 NAK SYN ETB CAN EM SUB ESC IS1 IS2 IS3 IS4 DEL.

Estas clases de caracteres deben ir encerradas entre corchetes. Por ejemplo, para localizar las cadenas que contienen dígitos, la expresión regular a usar sería **[[:digit:]].**

- El signo de dólar "$" y el circunflejo "^" se utilizan para especificar la posición que debe tener la cadena a localizar, por lo tanto, no representan un carácter en especial dentro de dicha cadena.

  Con el dólar se especifica que la cadena a localizar, mediante la expresión regular, debe estar al final de la cadena de caracteres o al final de la línea (si es posible utilizar el modo multilínea). Por ejemplo, si se busca en la cadena *El fruto de la granada se da en Granada* la expresión regular **[gG]ranada$,** el resultado que se obtendrá será la cadena "Granada" ya que es la última ocurrencia de la expresión regular en dicha cadena de caracteres.

  El circunflejo tiene una doble finalidad, que difiere cuando se utiliza individualmente de cuando se utiliza en conjunto con otros caracteres especiales.

  1. En primer lugar su funcionalidad como carácter individual consiste en especificar que la cadena a localizar debe estar al inicio de la cadena de caracteres. Por lo tanto, la expresión regular **^[0-9]** localizará todas aquellas cadenas que comienzan por un dígito decimal.
  2. Cuando se utiliza el circunflejo dentro de los corchetes anteponiéndose a un grupo de caracteres, la búsqueda consistirá en localizar cualquier carácter que no coincida con los especificados por dicho grupo. Entonces, la expresión regular **[^0-9]** localizará todas aquellas cadenas que no contengan un dígito decimal.

  La utilización en conjunto de los metacaracteres $ y ^ permite realizar validaciones de forma sencilla. Por ejemplo, la expresión regular **^[0-9]$** permite localizar todas las cadenas compuestas por un único dígito decimal.

- El asterisco "*" designa cero o más ocurrencias de la expresión regular de un solo carácter precedente. Por ejemplo, para localizar una cadena de dígitos de cualquier longitud se utilizaría la expresión regular **[0-9][0-9]*.** No se puede utilizar la expresión regular **[0-9]*** ya que también designa la cadena vacía (cadenas que no contienen ningún digito decimal). Otro ejemplo, si se busca en la cadena *Es un lio leer lo leído* la expresión regular **le***, el resultado serán las cadenas "l" cuando no se da ninguna ocurrencia del carácter e (de las palabras lio y lo), "leído" cuando existe una única ocurrencia del carácter e y "leer" cuando el carácter e se repite dos veces.

  Cuando se usa el asterisco se ha de tener en cuenta que se intenta encontrar la cadena más larga que valide la expresión regular. Así, si se busca en la cadena *abc12345fgh* la expresión regular **[0-9][0-9]***, el resultado corresponderá con todos los dígitos decimales, es decir con la cadena "12345".

- Las llaves "{}" se tratan como caracteres literales cuando se utilizan por separado en una expresión regular. Para que adquieran su función de metacaracteres es necesario que encierren uno ({n}) o varios números separados por coma ({n,m}) y que estén colocados a la derecha de otra expresión regular.

  {n} En el caso de que las llaves encierren sólo un número, se busca que el carácter al que precede este metacarácter aparezca el número exacto de veces que expresa dicho número. Por ejemplo, la expresión regular **[0-9]{3}** localiza todos las cadenas numéricas compuestas sólo de tres dígitos decimales.

  {n,m} Si entre llaves se encierran dos números separados por coma, entonces se busca que el carácter al que precede dichas llaves aparezca al menos el número de veces que indica el primer número y no más del número de veces que indica el segundo número. Así, la expresión regular **[0-9]{3,5}** localiza todas las cadenas numéricas compuestas por 3, 4 o 5 dígitos decimales.

Nota: *No todos los metacaracteres son válidos en cualquier comando Linux.*

5. FILTROS

grep. Busca una cadena

Esta orden permite buscar cadenas dentro de ficheros, utilizando expresiones regulares para especificar la cadena coincidente. La salida que produce este comando son las líneas del fichero que contienen la cadena coincidente.

Sintaxis: grep [-vcniwr] expresión_regular fichero (o lista de ficheros)

```
Ejemplos:
       1. Si se desea encontrar las líneas que comienzan por una letra minúscula seguida de un
dígito se pondría:            grep   "^[a-z][0-9]"   nombre_fichero

       2. Si se desea localizar las líneas que terminan en ; la orden seria:
       grep  ";$" /home/programa.c
```

En el caso de que la cadena a buscar se componga de más de una palabra (o elemento), se debe acotar entre comillas dobles para que el comando **grep** no los interprete como nombres de ficheros.

Nota: *El shell de Linux interpreta siempre todos los metacaracteres antes de ejecutar el comando.*

Si se incluye más de un fichero en la línea de órdenes, el comando **grep** informará del nombre del fichero al comienzo de cada línea que imprima.

*Opciones**:***

1. La opción **-v** selecciona todas las líneas que no contengan el patrón especificado.
2. La opción **-c** devuelve solamente el número de líneas coincidentes.
3. La opción **-n** añade a la salida de cada línea, el número de línea que ocupa en el fichero fuente.
4. La opción **-i** ignora la distinción entre mayúsculas y minúsculas en la comparación.
5. La opción **-w** selecciona solamente aquellas líneas que contienen concordancias con la expresión regular formando palabras completas. Para ello, comprueba si la cadena de caracteres que concuerda esta al principio de la línea o precedida por un carácter que no se considera como parte de una palabra; o bien esta al final de la línea o seguida de un carácter que no se considera como parte de una palabra. Los caracteres que se consideran que pueden formar parte de palabras son letras, dígitos y el signo de subrayado. Por ejemplo, si se busca en la cadena *el elefante del circo se llama El!* la expresión regular [eE]l usando esta opción (**grep -w "[eE]l"**), se obtendrá como resultado la primera palabra de la cadena ("el") y la última (El). La cadena "el" existente en las palabras "elefante" y "del" no se localizan ya que por si solas no se consideran palabras.
6. La opción **-r** (o **-R)** busca recursivamente dentro de todos los subdirectorios del directorio actual.

```
Ejemplos:
       grep   -n  main  /home/programa.c
       grep   -c  void  /home/programa.c
       grep   -v  "^{"  /home/programa.c
       ls -l  | grep ^d
```

cmp. Compara dos ficheros

Este comando permite comparar dos ficheros cualesquiera para comprobar si son iguales o no. Su salida es muy limitada, si los ficheros son iguales, no dice nada, si son distintos, informa del primer carácter en que se diferencian los dos ficheros.

Sintaxis: cmp [-l] fichero1|- fichero2

En lugar del primer nombre del fichero, este comando puede llevar el argumento **-** (menos). En este caso compararía la entrada estándar con el segundo fichero.

La opción **-l** hace que el comando liste el número de carácter y los valores diferentes para cada una de las diferencias entre los dos ficheros.

```
Ejemplo:
       cat   prueba  |  cmp  -  dir/prueba
```

diff. Compara dos ficheros

Esta orden es similar a la anterior. Se utiliza para comparar dos ficheros. Si los ficheros son iguales, no dice nada. Si son distintos, da como salida un listado completo de todas las líneas que difieren entre los dos ficheros junto con sus números de líneas.

Sintaxis: diff [-wi] fichero1|- fichero2

El funcionamiento del **-** (menos) como sustituto del primer nombre de fichero es igual al del comando **cmp**.

Opciones:

1. La opción **-w** hace que el comando ignore las líneas que sólo se diferencian en sus espacios en blanco (también tabuladores).
2. La opción **-i** ignora la diferencia entre mayúsculas y minúsculas en la comparación.

```
Ejemplo:
       dif  f1 f3
```

sort. Muestra el contenido de un archivo en orden

Esta utilidad muestra el contenido de un fichero en orden alfabético por líneas, pero no cambia el contenido del fichero original. Si se incluye más de un fichero en la línea de órdenes, este comando mostrará de forma ordenada por líneas el contenido de todos los ficheros unidos.

Sintaxis: sort [-un] fichero (o lista de ficheros)

Opciones:

1. La opción **–u** muestra el contenido del fichero en orden alfabético por líneas eliminando las líneas duplicadas.
2. La opción **–n** muestra el contenido del fichero (se supone que contiene números) ordenado por líneas siguiendo el orden numérico.

```
Ejemplos:
       1. Si el fichero numeros contiene un número entero por línea elegidos al azar:
       sort numeros
       sort –n numeros

       2. sort -u f1 f2
```

cut. Corta campos de una tabla escrita en un fichero

Una tabla se puede considerar como una colección de líneas, cada una de las cuales contiene un registro; y donde cada registro tiene un número fijo de campos. Normalmente, los campos están separados mediante tabuladores o espacios, aunque se puede utilizar cualquier separador de campos. El comando **cut** permite cortar uno o más campos de una tabla almacenada en uno o más ficheros; es decir permite trocear verticalmente un fichero.

Sintaxis: cut -c *lista* fichero (o lista de ficheros)

cut -f *lista* [-d*car*] [-s] fichero (o lista de ficheros)

lista es una lista de números que especifica el rango de caracteres o campos a extraer. Los caracteres o campos de la tabla se comienzan a enumerar desde 1. Si los números de esta lista están separados por coma, dichos números especifican los campos o caracteres del archivo a extraer. En el caso de que los números de la lista estén separados mediante un guión (–), los números de la lista especifican el rango de caracteres o de campos a extraer.

car indica el carácter utilizado como separador de campos. El tabulador es el separador de campos por omisión.

Opciones:

1. La opción **–c** trata cada carácter como una columna y extrae los caracteres especificados en la *lista*.
2. La opción **–f** extrae los campos especificados en la *lista*.
3. La opción **–d** utiliza el carácter *car* en lugar del tabulador como separador de campos.
4. La opción **–s** no muestra las líneas del fichero que no contengan el carácter delimitador.

```
Ejemplos:
       Suponiendo que el contenido del fichero original maquinas es el siguiente:

       eixe           LaboratorioSO8        192.132.111.81        pcpardo
       pindo          LaboratorioSO7        192.132.222.33        nrufino
       faro           Laboratorio37         192.132.333.37        nanny

donde los campos han sido separados usando tabulador, los resultados que se obtendrían con los
siguientes comandos son:
       a.     cut –f1 maquinas       // Extrae el primer campo de cada línea
              eixe
              pindo
              faro

       b.     cut –f1,3 maquinas     // Extrae el primero y el tercer campo de cada línea
              eixe           192.132.111.81
              pindo          192.132.222.33
              faro           192.132.333.37

       c.     cut –f1-3 maquinas     // Extrae los tres primeros campos de cada línea
              eixe           LaboratorioSO8        192.132.111.81
              pindo          LaboratorioSO7        192.132.222.33
              faro           Laboratorio37         192.132.333.37

       d.     cut –c1,3 maquinas     // Extrae el primero y el tercer carácter de cada línea
              ex
              pn
              fr

       e.     cut –c1-4 maquinas     // Extrae los cuatro primeros caracteres de cada línea
              eixe
              pind
              faro
```

Es conveniente tener en cuenta que la salida de esta orden se visualiza en pantalla y no se almacena en un fichero.

paste. Concatena ficheros horizontalmente

Este comando muestra el resultado de concatenar el contenido de los ficheros que se especifican como argumentos horizontalmente (la orden cat concatena ficheros verticalmente). Este comando, por lo tanto, complementa al comando anterior (cut), pues permite pegar tablas por columnas. No obstante, se debe tener en cuenta que la salida de esta orden se visualiza en pantalla y no se almacena en un fichero.

Sintaxis: paste lista de ficheros

```
Ejemplo:
        paste   f1   f2
```

tr. Sustituye/Elimina caracteres

Esta orden permite reemplazar o eliminar determinados caracteres de los datos de entrada. Al contrario de muchos comandos, éste no acepta nombre de ficheros como argumentos, pues sólo acepta como datos de entrada los que provengan de la entrada estándar o los obtenidos mediante los operadores de redireccionamiento.

Sintaxis: tr [-s][-t][-c][-d] conjunto1 [conjunto2]

Los parámetros *conjunto1* y *conjunto2* son secuencias de caracteres, que pueden ser expresadas mediante expresiones regulares, y que pueden ir entre comillas dobles o simples. El parámetro designado por *conjunto1* lista los caracteres que se deben sustituir o eliminar de los datos de entrada. El segundo parámetro (*conjunto2*) solo se emplea para reemplazar, pues lista los caracteres que sustituirán a los especificados por el primer parámetro.

Cuando se usa sin opciones, el comando tr reemplaza los caracteres especificados en *conjunto1* con aquellos proveídos por *conjunto2* en el orden indicado, es decir, asocia cada carácter de *conjunto1* con su correspondiente carácter de *conjunto2*.

```
Ejemplos:
        Los siguientes ejemplos detallan las distintas asociaciones entre los caracteres del
conjunto1 y los caracteres del conjunto2:

a.      Sustituye cada letra a del texto que se introduce mediante el teclado por la letra 1.
        Usando la tecla Enter se pueden añadir nuevas líneas en el texto y con Ctrl+d se
        especifica el final del mismo.
        tr  "a"  "1"

b.      Del texto que se introduce mediante el teclado, se sustituye cada letra a por la letra
        1, cada letra b por la letra 2 y cada letra c por la letra 3. Es decir, se asocia el
        carácter a con el 1, el b con el 2, y así sucesivamente.
        tr  "abc"  "123"

c.      En caso de que la secuencia conjunto1 sea menor que la secuencia conjunto2, la asociación
        se realiza con la misma lógica ignorando los sobrantes de la secuencia conjunto2,
        mientras que si se da el caso contrario de que la secuencia conjunto1 sea mayor, el
        comando asume que el último carácter de la secuencia conjunto2 estará asociada a los
        sobrantes de la secuencia conjunto1.
        tr "abc"  "12"
        En este caso, se asocia la letra a con 1 y las letras b y c con 2.
```

Opciones:

1. La opción **–t** trunca la longitud del *conjunto1* según la longitud del *conjunto2*. Esto hace que si el *conjunto1* es más largo que el *conjunto2*, solo se tendrán en cuenta los n primeros caracteres del *conjunto1*, siendo n la longitud del *conjunto2*.

```
Ejemplo:
        tr -t  "abc"  "12"            // se asocia la letra a con 1 y la letra b con 2.
```

2. La opción **-s** sustituye un conjunto de caracteres repetidos consecutivos pertenecientes al *conjunto1* por sólo uno o por su correspondiente carácter del *conjunto2*.

```
Ejemplos:

a.      Elimina los caracteres a y e repetidos. Si el dato de entrada es "Feeeeliiiciiidaaaaades"
        el resultado del comando será "Feliiiciiidades"
        echo   Feeeeliiiciiidaaaaades | tr –s  ae

b.      Elimina los caracteres a y e repetidos y después asocia la letra a con 1 y la letra e
con     2. Si el dato de entrada es "Feeeeliiiciiidaaaaades" el resultado del comando será
        "F2liiiciiid1d2s"
        echo   Feeeeliiiciiidaaaaades | tr –s  ae  12
```

3. La opción **-c** sustituye todos los caracteres que no se encuentran especificados en el *conjunto1* por el carácter especificado en el *conjunto2*.

```
Ejemplo:
        Si se quiere sustituir por guión todos los caracteres no numéricos de la cadena de
entrada "Los numeros 15, 3, 71 y 9 son impares" se podría usar esta opción de la siguiente
forma:
        echo "Los numeros 15, 3, 71 y 9 son impares" | tr –c   "0-9"   "-"
```

4. La opción **–d** elimina de los datos de entrada todos los caracteres especificados en el *conjunto1*.

```
Ejemplo:
        Si se quiere eliminar todos los caracteres numéricos de la cadena de entrada "Los numeros
15, 3, 71 y 9" se podría usar esta opción de la siguiente forma:
        echo "Los numeros 15, 3, 71 y 9 son impares" | tr –d  "0-9"
```

Tanto *conjunto1* como *conjunto2* pueden especificar rangos de caracteres, en especial cuando son numerosos los caracteres implicados.

```
Ejemplo:
        Para visualizar el resultado de convertir cada carácter en minúscula, del fichero texto,
a mayúscula, se podrían emplear los siguientes comandos:
        more texto  |  tr   "a-z"   "A-Z"
```

El comando **tr** también puede usar los caracteres de barra invertida.

Ejemplo:
Si se quiere sustituir todos los espacios en blanco por salto de líneas en el fichero *texto*, se emplearían los siguientes comandos:

```
more texto | tr " " "\n"
```

También se pueden usar las clases de caracteres predefinidas para expresar el *conjunto1*. Si se trata de una sustitución, como *conjunto2* sólo se pueden usar las clases de caracteres [:lower:] y [:upper:].

Ejemplos:

a. Transforma todos los caracteres no alfanuméricos del fichero *texto* en signos de admiración.

```
more texto | tr -c "[:alnum:]" "!"
```

b. Elimina todos los caracteres alfabéticos del fichero *texto*.

```
more texto | tr -d "[:alpha:]"
```

c. Transforma todos los caracteres numéricos del fichero *texto* en signo de resta.

```
more texto | tr "[:digit:] " "-"
```

d. Sustituye todos los espacios en blanco del fichero *texto* por coma.

```
more texto | tr "[:blank:]" ","
```

e. Transforma todos los caracteres alfabéticos escritos en minúscula del fichero *texto* en mayúscula.

```
more texto | tr "[:lower:]" "[:upper:]"
```

f. Elimina todos los signos de puntuación del fichero *texto*.

```
more texto | tr -d "[:punct:]"
```

Aunque el comando **tr** no acepta nombres de ficheros como argumentos, se puede usar para modificar copias de sus contenidos. Para ello, es necesario usar el operador que redirecciona la entrada.

Ejemplo:
Se visualiza el resultado de eliminar todos los signos de puntuación del fichero *texto*.

```
tr -d "[:punct:]" < texto
```

Es más normal almacenar en un fichero el resultado del comando **tr** que visualizarlo por pantalla. Para ello, basta usar el operador que redirecciona la salida.

Ejemplo:
Se almacena en el fichero *salida* el resultado de sustituir todos los caracteres no alfabéticos del fichero *entrada* por espacios en blanco.

```
tr -c "[:alpha:]" "" <entrada >salida
```

También se puede usar el operador de redireccionamiento para añadir el texto modificado al final del fichero usado como entrada del comando **tr**.

Ejemplo:
Se añade al final del fichero *texto* el resultado de eliminar los espacios en blanco.

```
tr -d "[:blank:]" < texto >> texto
```