

Sumario

Gestión de archivos GNU/Linux.....	2
Directorio actual.....	2
Cambiar de directorio.....	2
Rutas.....	2
Mostrar el contenido de un directorio.....	2
Caracteres comodín * y ?.....	2
Borrar directorios.....	2
Copiando ficheros y directorios.....	3
Moviendo objetos.....	3
Eliminando objetos.....	4
Creando ficheros.....	4

Gestión de archivos GNU/Linux

Directorio actual

- Sintaxis **pwd**
- En el prompt del sistema también da esta información.
- El símbolo `~` representa nuestro directorio personal.



```
ricardo@AH512: /var/www
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
ricardo@AH512: /var/www$ pwd
/var/www
ricardo@AH512: /var/www$
```

Cambiar de directorio.

Sintaxis **cd [ruta]**

- `cd ~` nos lleva a nuestro directorio personal.
- `cd` nos lleva a nuestro directorio personal.
- `cd ..` nos lleva al directorio padre del actual, si estamos en el raíz nos deja en el raíz.

Rutas

Absolutas. Son las rutas a un fichero o directorio que parten desde el directorio raíz, siempre empiezan por el símbolo `/`.

Relativa. Son las rutas hasta un fichero o directorio partiendo desde el directorio actual. Nunca empiezan por el símbolo `/`.

Mostrar el contenido de un directorio.

Sintaxis **ls [opciones] [rutas_ficheros_o_directorios]**

Opciones:

- `-l` formato largo, da información de cada entrada.
- `-a` muestra los ficheros o directorios ocultos.
- `-h` muestra la información para humanos.

Caracteres comodín * y ?

Sintaxis **mkdir [opciones] directorio [directorio ...]**

Opciones:

- `-p` Crear directorios padres si es necesario.
- `-v` Salida detallada. (modo verboso).
- `-m 777` establecemos las opciones de seguridad.

Borrar directorios.

Sintaxis **rmdir [directorios] [opciones]**

Elimina los archivos o directorios, solo si están vacíos.

Copiando ficheros y directorios

El comando cp es un abreviatura de copy (copiar); permite copiar archivos y directorios.

Sintaxis

- **cp [Opciones] archivo_fuente directorio_destino**
- **cp [Opciones] archivo_fuente archivo_destino**

Opciones

- -a conserva todos los atributos de los archivos.
- -d copia un vínculo pero no el fichero al que se hace referencia.
- -i pide confirmación antes de sobrescribir archivos.
- -p conserva permisos del archivo origen: propietario, permisos y fecha.
- -R o -r copia los archivos y subdirectorios.
- -s crea enlaces en vez de copiar los ficheros.
- -u únicamente procede a la copia si la fecha del archivo origen es posterior a la del destino.
- -v muestra mensajes relacionados con el proceso de copia de los archivos.

Descripción

El comando cp copia un archivo a otro. También puede copiar varios ficheros en un directorio determinado.

Moviendo objetos

Mover un fichero consiste en cambiar de ubicación un fichero o directorio.

Cuando usamos mv primero comprueba si existe, si es así la acción por defecto es preguntar si queremos sobrescribir el destino. Si queremos evitarlo hemos de usar la opción -f. Con la opción -r hacemos la copia de forma recursiva (incluyendo los ficheros y directorios que contiene).

El uso es similar a cp con la diferencia de que no mantiene el original.

Sintaxis

- **mv [Opciones] fuente destino**

Opciones

- -f elimina los archivos sin solicitar confirmación.
- -v pregunta antes de sobrescribir los archivos existentes.
- -r mueve archivos y directorios que contiene

Descripción

El comando mv se puede utilizar para modificar el nombre o mover un archivo de un directorio a otro. Trabaja tanto con archivos como con los directorios.

Eliminando objetos

rm permite eliminar archivos y directorios, puede eliminar un directorio completo con la opción `-r`.

Sintaxis

rm [Opciones] archivos o directorio

Opciones

- `-f` elimina todos los archivos sin pedir confirmación
- `-i` pregunta antes de eliminar un archivo.
- `-r` elimina todos los archivos que se encuentran en un subdirectorio y por último borra el propio subdirectorio.
- `-v` muestra el nombre de cada archivo antes de eliminarlo.

Descripción

El comando `rm` se utiliza para borrar los archivos que se le especifiquen. Para eliminar un fichero ha de tener permiso de escritura en el directorio en el que se encuentra.

Creando ficheros

touch

Crea fichero vacío con el nombre especificado si no existe o lo “toca” (le cambia la fecha a la fecha actual“

Sintaxis

touch [archivo]

Opciones

no tiene

Descripción

Crea un un fichero si no existe y, si existe, modifica su atributo fecha y hora a la fecha y hora actual.

Mostrar contenido de un fichero.

cat

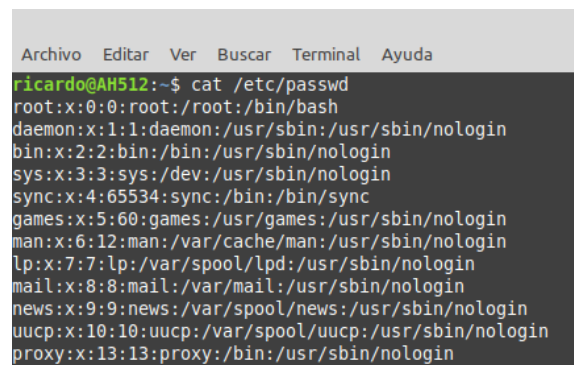
El comando **cat** (concatenate), es un comando que ha sido desarrollado para que ejecute básicamente tres funciones asociadas a los archivos de texto, estas son:

- Poder visualizarlos
- Realizar una combinación con copias de ellos
- Crear nuevos archivos.

Su sintaxis de uso es:

cat [opciones] [archivo] [-] [archivo]

Opciones:



```
Archivo  Editar  Ver    Buscar  Terminal  Ayuda
ricardo@AH512:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
```

-n, --number (numero)

more

Al igual que 'cat', 'more' permite visualizar por pantalla el contenido de un fichero de texto, con la diferencia con el anterior de que 'more' pagina los resultados.

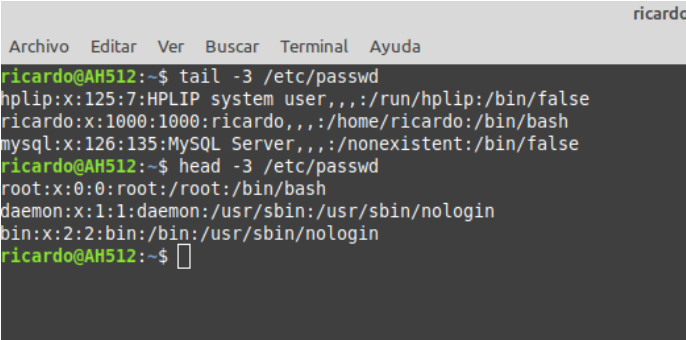
Primero mostrará por pantalla todo lo que se pueda visualizar sin hacer scroll y después, pulsando la tecla espacio avanzará de igual modo por el fichero.

less

El comando 'less' es el más completo de los tres, pues puede hacer todo lo que hace 'more' añadiendo mayor capacidad de navegación por el fichero (avanzar y retroceder) .

head y tail.

Los comandos head y tail permiten mostrar de forma parcial el contenido de un fichero. Como su nombre indica, head muestra las primeras líneas del fichero (la cabecera) y tail muestra las últimas líneas (la cola).



```
ricardo@AH512:~$ tail -3 /etc/passwd
hplip:x:125:7:HPLIP system user,,,:/run/hplip:/bin/false
ricardo:x:1000:1000:ricardo,,,:/home/ricardo:/bin/bash
mysql:x:126:135:MySQL Server,,,:/nonexistent:/bin/false
ricardo@AH512:~$ head -3 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
ricardo@AH512:~$
```

Redirecciones

Cada vez que se inicia un intérprete de órdenes, se abren automáticamente tres archivos:

stdin: archivo estándar de entrada, es de donde los programas leen su entrada

stdout: archivo estándar de salida, es a donde los programas envían sus resultados

stderr: archivo estándar de error, es a donde los programas envían sus salidas de error

Los operadores de redirección `> | >> | <` permiten cambiar los flujos E/S estándar.

stdin se identifica generalmente con el teclado.

stdout y **stderr** se identifican normalmente con la pantalla.

Redirección de salida (stdout)

orden `>` fichero

orden `>>` fichero

Se puede redireccionar la salida de cualquier orden a un determinado archivo en vez de hacerlo por la salida estándar stdout o pantalla.

Para obtener una redirección de salida, se utiliza el carácter mayor que, `>`.

Si el archivo al que redireccionamos no existe, el shell lo creará automáticamente, si, por el contrario ya existía, entonces se sobrescribirá la información, machacando el contenido original del archivo.

date `>` prueba.txt → Se almacena la fecha actual del sistema en el archivo prueba.txt

Si lo que queremos es añadir información a un archivo sin destruir su contenido, deberemos utilizar para la redirección el doble símbolo de mayor que, `>>`.

ls `>>` prueba.txt → Se almacena al final del fichero prueba.txt el contenido del directorio activo.

Redirección de la salida de errores (stderr)

\$ orden `2>` fichero

\$ orden 2>> fichero

Este tipo de redireccionamiento se utiliza para almacenar los errores producidos en la ejecución de un comando en un archivo específico, en vez de utilizar la pantalla. Para ello se utilizan los caracteres 2> para crear y 2>> para añadir.

cat fichero > fichero1 2> errores

Si el archivo fichero no existe o no está en ese lugar, o se da cualquier otro error, se crea un fichero llamado errores con el contenido del mensaje del error que se hubiera mostrado por pantalla. Si el fichero errores existe, se destruye y se queda con la última información; si no existe, se crea.

Si **no queremos ver errores** en pantalla, podemos redireccionar la salida estándar de errores a /dev/null, es el cubo de basura en Linux. En el siguiente ejemplo, si no colocamos 2> /dev/null aparecerán errores de acceso en pantalla

find / -name *.c 2> /dev/null

Redirección de entrada (stdin)

\$ orden < fichero

cat < hotla.txt

Redirección de la salida estándar y de errores

\$ orden &> fichero

Cualquier orden que lea su entrada en stdin puede ser avisada para que tome dicha entrada de otro archivo en lugar de tomarla por el teclado.

Esto se hace utilizando el carácter menor que, <. La redirección de entrada no produce ningún cambio en el archivo de entrada.

mail pepe < listado2 → Se envía el fichero listado2 por correo electrónico al usuario pepe.

Expresiones regulares

Las expresiones regulares son un medio para describir patrones de texto. Imaginemos que no sólo queremos buscar en un texto todas las líneas que contienen una palabra, como por ejemplo Barcelona, sino que sólo nos interesan las líneas que empiezan por la palabra Barcelona, pero no las que contengan la palabra en cualquier otra posición. Describir el patrón “Barcelona” es trivial, tan sólo hay que escribir “Barcelona”, pero ¿cómo podemos describir “La línea comienza por la palabra Barcelona”. Las expresiones regulares permiten describir este tipo de patrones de texto y muchos más por lo que nos serán de una gran utilidad.

En la web hay varios sitios en los que se pueden probar las expresiones regulares, como por ejemplo regex101.

Fichero con los ejemplos.

Hospital	apellido_paciente	Biobanco	Tipo_Usher
Barcelona	Castells	Barcelona	Usher2A
Mallorca	Pagan	Barcelona	usher1b
Albacete	Blanca	Madrid	usher2
madrid	Cardosa	madrid	usher1a
almeria	Meliá	Valencia	sano
Valencia	Paniagua	Castellón	usher1B
Castellón	Sogorb	valencia	usher2B
castellon	Galiana	Barcelona	usher3
Granada	Bastella	Valencia	sana
Alicante	Carrícola	Barcelona	usher3A

https://bioinf.comav.upv.es/courses/unix/demo_data/origenes.txt

Expresiones alternativas

Una expresión u otra. Imaginemos que tenemos un fichero con los orígenes de los pacientes de un estudio sobre síndrome de Usher. Queremos seleccionar todos los pacientes que vienen de Valencia o Castellón. Podemos hacerlo utilizando la sintaxis para expresiones alternativas:

```
~$ grep -E 'Valencia|Castellón' origenes.txt
```

```
almeria Meliá Valencia sano
```

```
Valencia      Paniagua      Castellón      usher1B
```

```
Castellón     Sogorb  valencia      usher2B
```

```
Granada Bastella Valencia sana
```

La barra vertical (|) separa las expresiones alternativas. En este caso significa que la palabra encontrada puede ser Valencia o Castellón.

Contenedores

En muchas ocasiones nos interesa seleccionar patrones que pueden tener en una posición varias letras distintas. Por ejemplo podríamos describir el patrón “comienza por Usher y después tiene un número” o “comienza por usher y después tiene un par de letras”. Busquemos todas las líneas que tienen la palabra Usher o usher:

```
~$ grep -E --color '[uU]sher' origenes.txt  
Hospital    apellido_paciente    Biobanco    Tipo_Usher  
Barcelona   Castells             Barcelona    Usher2A  
...
```

Al poner las letras entre corchetes indicamos que el carácter que aparezca en esa posición puede ser cualquiera de los caracteres indicados. En este caso la expresión coincidente podría ser Usher u usher.

Utilizando esta técnica podemos también indicar rangos de caracteres, como por ejemplo “aquellos con Usher tipo 0,1 ó 2”:

```
~$ grep -E --color '[uU]sher[0-2]' origenes.txt  
Barcelona   Castells             Barcelona    Usher2A  
Mallorca    Pagan                Barcelona    usher1b  
...
```

Existen rangos y tipos de caracteres predefinidos que podemos utilizar como:

POSIX	ASCII	Significado
[[:alnum:]]	[A-Za-z0-9]	Caracteres alfanuméricos (letras y números)
[[:word:]]	[A-Za-z0-9_]	Caracteres alfanuméricos y “_”
[[:alpha:]]	[A-Za-z]	Caracteres alfabéticos
[[:blank:]]	[\t]	Espacio y tabulador
[[:space:]]	[\t\r\n\v\f]	Espacios
[[:digit:]]	[0-9]	Dígitos
[[:lower:]]	[a-z]	Letras minúsculas
[[:upper:]]	[A-Z]	Letras mayúsculas
[[:punct:]]	[!@#\$%^&'()*+,-./:;<=>?@\^_`{ }~]	Caracteres de puntuación

Por ejemplo, podríamos seleccionar cualquier dígito o letra:

```
~$ grep --color -E '[uU]she[[:alpha:]]' origenes.txt
```

Si nos da igual que en una posición haya un carácter u otro podemos utilizar un punto (.). Por ejemplo, seleccionar aquellos individuos con subtipo A o B:

```
~$ grep --color -E '[Uu]sher.[[:alpha:]]' origenes.txt  
Barcelona    Castells    Barcelona    Usher2A  
Mallorca     Pagan      Barcelona    usher1b  
...
```

Dentro de los contenedores resultan también bastante útiles las negaciones. Podemos, por ejemplo, seleccionar cualquier carácter que no sea un 2 o un 3 añadiendo “^” tras el primer corchete:

```
~$ grep --color -E 'sher[^23]' origenes.txt  
Mallorca     Pagan      Barcelona    usher1b  
madrid       Cardosa    madrid       usher1a  
Valencia     Paniagua   Castellón    usher1B
```

Cuantificadores

Además de indicar qué caracteres queremos permitir podemos seleccionar cuantas veces deben aparecer. Si no añadimos nada que indique lo contrario se asume que el carácter debe aparecer una vez, pero podríamos pedir que el carácter aparezca un número distinto de veces:

- “?”, el carácter aparece ninguna o una vez. “usher1?” coincidiría con “usher” o “usher1”.
- “*”, cero, una o varias veces.
- “+”, al menos una vez.
- “{4}”, cuatro veces.
- “{4,10}”, entre 4 y 10 veces

```
~$ grep --color -E 'sher2.?' origenes.txt  
Barcelona    Castells    Barcelona    Usher2A  
Albacete     Blanca     Madrid       usher2  
Castellón    Sogorb     valencia     usher2B
```

Puntos de anclaje

Además de poder indicar qué y cuántas veces queremos que algo aparezca podemos indicar dónde deseamos que lo haga. Los puntos de anclaje más utilizados son:

- ”^”, inicio de línea
- ”\$”, fin de línea
- ”<”, principio de palabra
- ”>”, fin de palabra
- “\b”, límite de palabra

Líneas que comienzan por B:

```
~$ grep --color -E '^B' origenes.txt  
Barcelona    Castells    Barcelona    Usher2A
```

Líneas que terminan por B:

```
~$ grep --color -E 'B$' origenes.txt  
Valencia    Paniagua    Castellón    usher1B  
Castellón    Sogorb    valencia    usher2B
```

Sustituciones

Las expresiones regulares además de ser útiles para buscar patrones sirven para sustituirlos. Por ejemplo podemos sustituir madrid por Madrid con el siguiente comando:

```
~$ sed -r 's/madrid/Madrid/' origenes.txt  
...  
Albacete    Blanca Madrid usher2  
Madrid Cardosa madrid usher1a  
...
```

Ya vimos que si queríamos que la expresión se utilizase más de una vez en cada línea debíamos utilizar el modificador “g”:

```
~$ sed -r 's/madrid/Madrid/g' origenes.txt
...
Albacete      Blanca Madrid usher2
Madrid        Cardoso Madrid usher1a
...
```

Estas sustituciones además de soportar los patrones sencillos soportan todo lo que hemos visto anteriormente. Por ejemplo, podemos eliminar la primera columna del fichero con el comando:

```
~$ sed -r 's/^[[:alnum:]]*\t/' origenes.txt
apellido_paciente      Biobanco      Tipo_Usher
Castells      Barcelona      Usher2A
...
```

Si lo que queremos es guardar el patrón encontrado para utilizarlo en la sustitución debemos utilizar paréntesis. Los paréntesis indican que lo que hay dentro de ellos debe ser recordado para poder utilizado en la substitución. Por ejemplo, podríamos permutar las columnas 1 y 2 del fichero con el comando:

```
~$ sed -r 's/^\([[:alnum:]]*\)\t([[:alnum:]]*)\2\t1/' origenes.txt
```

Escapes

Algunos caracteres tienen significados especiales, como ., \$, (,), [,], \ o ^ y si se quieren utilizar hay que escaparlos precediéndolos con \.

El campo de las expresiones regulares es muy amplio y esta pequeña introducción sólo ha pretendido mostrar algunas de las posibilidades de esta gran herramienta. El mejor modo de aprender a utilizarlas es intentar hacer uso de ellas en problemas concretos y echar una ojeada a algunos de los libros y tutoriales que se han escrito sobre ellas.

Ejercicios

1. Buscar las líneas en las que aparece la palabra bash en el archivo /etc/passwd.

2. Buscar en el archivo /etc/group todas las líneas que empiezan por m.

3. En el fichero anterior imprimir todas las líneas que no empiezan por m.

4. ¿Cuántos ficheros README hay en los subdirectorios de /usr/share/doc?

5. ¿Qué ficheros o directorios en /etc contienen un número en el nombre?

Trabajando con el fichero de los [orígenes](#) de los pacientes de Usher resolver las siguientes cuestiones.

6. Seleccionar los pacientes enviados por el hospital de Castellón.

7. Seleccionar los que vienen del Biobanco de Barcelona.

8. Buscar los pacientes que no tienen Usher3.

9. En el fichero de pacientes de Usher hay algunas líneas separadas por espacios y otras por tabuladores, cambiar todos los separadores a comas.

Disponemos de un fichero con una [caracterización morfológica](#) de unas plantas.

10. Recuperar aquellas plantas numeradas con 1 o con 3 dígitos.

11. Añadir ceros al identificador de las plantas para que tengan todos 3 dígitos.

12. Construir un nuevo fichero que incluya solo las filas sin datos faltantes.

Tuberías y filtros.

Podemos en una misma línea ejecutar dos o más comandos separados por el carácter tubería | , pasando salida estándar del primer comando con la entrada estándar del siguiente comando.

Se uso para es filtrar datos utilizando comandos denominados filtros.

Se pueden unir mas de dos comandos

orden1 | orden2

De esta forma, puedes ejecutar un comando, y su salida puede ser la entrada para otro, que a su vez puede servir de entrada a un tercero:

comando1 | comando2 | comando3,

Ejemplos:

```
more /etc/bash.bashrc
```

```
cat /etc/bash.bashrc | more
```

```
ls | more
```

```
less /etc/bash.bashrc
```

```
cat /etc/bash.bashrc | less
```

Obtiene los 10 últimos caracteres de fich.

```
tail -10c fich
```

Quita la primera línea del fichero /etc/bash.bashrc

```
tail -n+2 /etc/bash.bashrc
```

Quita las 2 últimas líneas del fichero /etc/bash.bashrc

```
cat -n /etc/bash.bashrc|head -n-2
```

WC

Cuenta el número de caracteres, palabras y líneas de uno o más fichero(s), incluye espacios en blanco y caracteres de salto de línea.

Cuando se indica el nombre de más de un fichero wc proporciona los contadores individuales y su totalidad y se presentan los de cada fichero etiquetados con su nombre.

Si no se especifica ningún fichero, leerá de la entrada estándar. Su sintaxis es:

sintaxis

`wc [opciones] [fichero(s)]`

Donde las opciones pueden ser:

-c visualiza sólo el número de caracteres del fichero.

-l visualiza sólo el número de líneas del fichero.

-w visualiza sólo el número de palabras del fichero.

Ejemplos:

`cat fichero|wc -l`

Cuenta las líneas de fichero.

`ls -l /etc | grep ^-|wc -l`

Cuenta el número de archivos de un directorio.

`ls -l /etc | grep ^d|wc -l`

Cuenta el número de directorios de un directorio.

`ls -la|wc -l`

Cuenta las líneas del listado completo de ficheros.

`ps aux|grep ubuntu|wc -l`

Cuenta los procesos del usuario ubuntu.

tr

Traduce, comprime y borra caracteres de la entrada estándar, escribiendo el resultado en la salida estándar.

`tr [opciones] conjunto1 [conjunto2]`

Opciones:

`-c` opera sobre el complemento (sobre cada carácter que no coincida)

`-d` borra caracteres de conjunto1, no traduce

`-s` reemplaza cada sucesión de entrada de un carácter repetido del conjunto1 por una sola aparición de dicho carácter → muy útil para convertir varios espacios en uno solo.

Su uso más habitual es procesar una salida antes de pasársela a `cut` para extraer columnas, por ejemplo para sustituir todos los espacios en blanco consecutivos por uno solo.

Ejemplos:

```
ls -l|grep ^d|tr -s » «|cut -f8 -d» «
```

Listado largo `-l`, donde filtro sólo los directorios `^d`, traduzco todos los espacios en blanco consecutivos en uno solo, y corto el campo 8 que en mi caso (en la distribución que utilizo) coincide en `ls -l`, con el nombre de archivo o directorio.

Podemos usar `-d` para borrar caracteres, combinado con `-c` para borrar lo que no coincida (el complemento):

```
cat vocales.txt | tr -dc [aeiou] → borro todo lo que no sean vocales
```

Su uso más habitual es procesar una salida antes de pasársela a `cut` para extraer columnas, por ejemplo para sustituir todos los espacios en blanco consecutivos por uno solo, en salidas de comandos `ls`, `ps`, etc.

```
ps aux|tr -s » «|cut -f11 -d» «
```

Extrae la columna 11 correspondiente al ejecutable del proceso.

Compararlo con este otro:


```
ps aux|cut -f11 -d» «
```

Pero también permite traducir caracteres, incluidos especiales como la tabulación \t.

```
ls -l|tail -n+2|tr -s » » «\t»
```

Con tail quito la primera línea (el resumen que saca ls -l)

Con tr sustituyo los espacios por tabuladores.

cut

Se usa para cortar columnas en ficheros y pasar a la salida estándar las columnas o campos de la entrada estándar o del archivo especificado. Su sintaxis es:

```
cut [opciones] lista [fichero(s)]
```

Donde las opciones pueden ser:

-c corta caracteres.

-f corta campos.

-d especifica el carácter de separación entre los distintos campos (delimitador). Por defecto el separador de campos es <TAB>

lista especifica que columnas se van a cortar. Estas se identifican por su número a partir del 1 y caben las siguientes posibilidades:

Ejemplo: ¿Qué Shell utiliza el usuario Ubuntu?

```
cat /etc/passwd | grep ubuntu | cut -f7 -d:
```

grep

Es un filtro que permite buscar cadenas de caracteres (patrón) en los archivos que se le indica y muestra la línea del fichero que contiene el patrón. Su sintaxis:

```
grep [opciones] patrón [fichero(s)]
```

Donde las opciones pueden ser:

- c visualiza el nº total de líneas donde se localiza el patrón.
- i elimina la diferencia entre mayúsculas y minúsculas.
- l visualiza sólo el nombre de los ficheros donde se localiza el patrón.
- n visualiza el nº de línea donde aparece el patrón, así como la línea completa.
- v visualiza las líneas del fichero donde no aparece el patrón.
- w obliga a que patrón coincida solamente con palabras completas
- x obliga a que patrón coincida solamente con líneas completas

Ejemplos: Se utiliza más como filtro con tuberías que como comando:

```
cat /etc/passwd|grep -w jblanco
```

Saca solo la linea de mi usuario. O bien:

```
cat /etc/passwd|grep jblanco
```

Igualmente también saca solo la linea de mi usuario → utiliza cadena como patrón

```
ps aux|grep ubuntu|wc -l
```

Cuenta los procesos del usuario ubuntu

→ Lo mejor de este comando es que admite “Expresiones Regulares” → Ver apartado 4 Expresiones regulares.

Un ejemplo: Mostrar sólo los directorios que empiecen por un número

```
ls -ld [0-9]*|grep «^d»
```

En este caso la ER es `^d` el gorro indica comienzo de línea y la `d` corresponde al tipo de archivo “directorio” cuando hacemos un listado largo con `ls -l`. Otro más:

```
ls -ld [A-Z]*|grep «^d»
```

Directorios que comiencen por letra mayúscula → utiliza una expresión regular.

Otro ejemplo algo más rebuscado:

Mostrar del fichero `/etc/login.defs` solo los valores establecidos, es decir, los que no llevan almohadilla delante, `#` → comentario

```
cat /etc/login.defs|grep -v «^#»|sort -u
```

`sort`

Se utiliza para ordenar líneas de un fichero o un flujo, puede usarse como comando o como filtro.

Puede ordenar alfabética o numéricamente, teniendo en cuenta que cada línea del fichero es un registro compuesto por varios campos, los cuales están separados por un carácter denominado separador de campo (tabulador, espacio en blanco, dos puntos...). Su sintaxis es:

```
sort [opciones] [fichero/s]
```

Donde las opciones más interesantes son:

- f ignora mayúsculas y minúsculas.

- n ordena campos numéricos por valor no por caracteres.

- r orden inverso.

- u suprime líneas repetidas en el fichero de salida

- t indica el delimitador de campos, por defecto el espacio en blanco.

- kN indica el campo por el que ordenar (hasta el final)

- kN,M indica el campo por el que ordenar (hasta el campo M)

-kN,N indica que ordene sólo por el campo N (N=M)

ficheros nombres de los ficheros a ordenar.

Ejemplos:

```
ps aux|cut -f1 -d» «|sort|uniq
```

Averiguar los usuarios que están ejecutando procesos en el sistema

ps aux → lista todos los procesos de todos los usuarios

cut -f1 -d» » → el flujo recibido se separa en campos usando como delimitador “ ” espacio en blanco, y nos quedamos con el primer campo -f1

Por último con sort se ordena y con uniq se deja solo uno de los repetidos.

Es equivalente a: ps aux|cut -f1 -d» «|sort -u

```
ls -l|grep ^-|sort -k5n
```

Lista solo ficheros y los ordena de forma numérica por la quinta columna, que en mi distro es el tamaño.

ls -l → Lista

grep ^- → filtra sólo los ficheros, que comienzan por –

-k5 → ordeno por el quinto campo/columna

-n → de forma numérica

```
sort -k3 -nt»:» /etc/passwd
```

Ordena de forma numérica por el tercer campo (UID) el fichero de usuarios que tiene como separador :

uniq

Descarta todas las líneas sucesivas idénticas, menos una de ENTRADA (o entrada estándar), escribiendo en SALIDA (o en la salida estándar). Se suele usar en combinación de sort.

Un ejemplo típico es averiguar los usuarios que están ejecutando procesos:

```
ps aux|cut -f1 -d" "|sort|uniq
```

Opciones:

c precede a las líneas con el número de ocurrencias

d muestra sólo las líneas duplicadas

u muestra sólo las líneas que son únicas