## PROPIEDAD Y PROTECCIÓN DE FICHEROS (PERMISOS):

En LINUX por cada fichero o directorio del sistema de ficheros, hay 3 clases de usuarios que pueden tener acceso:

- **PROPIETARIO (usuario):**

  Usuario que inicialmente lo creó. El propietario tiene capacidad para transferir la propiedad a otro usuario. Normalmente solo el superusuario realiza esta acción. Linux asigna permisos a nivel de Usuario.

- **GRUPO:**

  Un conjunto de usuarios relacionados bajo un nombre de grupo común. Todo usuario ha de encuadrarse en un grupo. La creación, borrado y gestión de grupos depende del administrador. Al combinarse varios usuarios en un grupo, LINUX permite permisos a nivel de Grupo.

- **PUBLICO (otros):**

  Todos los demás usuarios del sistema, es decir cualquier otra persona que tenga nombre de usuario y pueda conseguir acceso al sistema (y que no pertenezca al grupo). Este tipo de personas también tiene sus permisos.

Los Directorios y ficheros de LINUX tienen **tres tipos** de permisos que determinan las operaciones que se pueden realizar sobre ellos (Los significados difieren según se trate de un fichero o un directorio):

- **Lectura** **r**
- **Escritura** **w**
- **Ejecución** **x**

Estos permisos se suelen mostrar como una serie de **nueve caracteres** con las letras r, w,x :

| rwx | rwx | rwx |
|---|---|---|
| Usuario | Grupo | Otros |

Si alguno de los permisos está retirado, se muestra con un símbolo – así, por ejemplo:

**rwxr-xr--**

Significaría que:

Usuario: Lectura, Escritura y Ejecución.

Grupo: Lectura y Ejecución.

Otros: Lectura.

Otra forma de ver los permisos es como un nº octal de 3 cifras, cada una de ellas se corresponde con el peso en bits del usuario, grupo y otros:

**rwx r-x r--**

**111 101 100 7 5 4**

Visto en forma octal tenemos las siguientes posibilidades para cada uno de los 3 tipos de usuarios es la siguiente:

| | | | |
|---|---|---|---|
| 00 | 0 | Ningún permiso | --- |
| 001 | 1 | Solo ejecución. | --x |
| 010 | 2 | Solo escritura. | -w- |
| 011 | 3 | Solo escritura y ejecución. | -wx |
| 100 | 4 | Solo lectura. | r-- |
| 101 | 5 | Solo lectura y ejecución. | r-x |
| 110 | 6 | Solo lectura y escritura. | rw- |
| 111 | 7 | Lectura, escritura y ejecución. | rwx |

Este sistema de notación octal es una de las formas más sencillas de cambiar los permisos de un fichero o directorio con la orden **chmod**. El significado de los permisos es el siguiente:

- **Lectura (r):**

El permiso de lectura para un directorio significa que se pueden listar los ficheros hay en ese directorio. La información en detalle requiere permiso de ejecución. Si no se tiene este permiso, solo se puede acceder a los archivos si se sabe exactamente como se deletrean. Es independiente del permiso de lectura para los ficheros.

El permiso de lectura para un fichero permite ver el contenido del fichero.

- **Escritura (w):**

El permiso de escritura para un directorio significa que se pueden cambiar los contenidos de dicho directorio como crear nuevos ficheros, directorios y suprimir los existentes, cambiar nombre, mover, etc.

Los usuarios que tienen permisos de escritura en un directorio tienen control administrativo sobre el. Además de crear archivos puede borrar y cambiar el nombre de archivos de otros usuarios, incluso si no tiene permisos para modificar esos archivos.

El permiso de escritura para un fichero significa que se puede cambiar el contenido de dicho fichero, pero no se podrá eliminar si no hay permiso de escritura para el directorio.

- **Ejecución (x):**

El permiso de ejecución para un directorio, significa que se puede entrar (hacer un **cd**) o referenciar a ese directorio y también se pueden copiar ficheros desde ese directorio (si se tiene permiso de lectura), copiar hacia el (si se tiene de escritura).

El permiso de ejecución para un fichero significa que se puede usar ese fichero como una orden ejecutable.

Los permisos para directorios son a veces difíciles de entender, aquí hay un resumen de algunas posibilidades:

- Sin permiso de lectura ni ejecución: Directorio cerrado.

- Con permiso de lectura y ejecución: se pueden listar contenidos y subdirectorios (si estos lo permiten).

- Con permiso de lectura, sin ejecución: Caso raro, se pueden ver los nombres de los ficheros, pero no puede acceder a ellos, ni a sus atributos, ni a sus subdirectorios, ni copiarlos.

- Sin lectura, con ejecución: No se puede listar su contenido, pero si acceder a sus ficheros si se conoce su nombre.

**Orden chmod (Cambiar permisos a ficheros):**

Permite cambiar los permisos asociados a un fichero o directorio. Únicamente el usuario que posee el archivo puede cambiar los permisos del mismo (o el superusuario).

Se permiten rutas de acceso absolutas y relativas, y caracteres comodines.

Modo numérico:

**chmod** [-opcion/es] **permisos file1 file2 file3 ....**

chmod [-opcion/es] 444 catalogo mitexto

chmod 754 usr/vdai/cap*

Modo simbólico:

**chmod** [-opcion/es] permisos **file1 file2 file3 .....**

En este caso los permisos son una combinación de estas 3 partes:

- u usuario
- g grupo
- o otros
- a todos

¿Quién?

---

- \+ agregar permiso/s
- \- quitar permiso/s
- = agregar permiso/s

y quitar cualquiera no especificado

Agregar/Quitar

---

- r lectura
- w escritura
- x ejecución

| Permisos |
|---|

La estructura de los permisos ha de seguir la siguiente sintaxis:

| ¿Quién? | Agregar/Quitar | Permisos |
|---|---|---|

Se usar simultáneamente más de un tipo de usuario y más de un tipo de permiso

Se pueden usar varios permisos separados por comas.

Ejemplos de permisos con este sistema (no se puede poner blancos):

- chmod u+r datos1
- chmod a=g mifiche
- chmod ug+rw datos1 datos2
- chmod a=rw tea*
- chmod u-x, g+w,o-x mifichero
- chmod ug-wx otromas.c

Otras construcciones menos claras:

- *chmod rw datos1 Asigna los permisos rw a todos.*
- *chmod = datos1 Desactiva todos los permisos.*

**Orden umask (Mascara de creación de ficheros):**

Es la abreviatura de *user file-creation mode mask*, y sirve para establecer los permisos por defecto que tendrán los nuevos ficheros y directorios que creemos.

- umask Consultar la máscara.
- umask <valor> Cambiar la máscara. <valor> debe ser entre 000 y 777.

Esta orden no tiene opciones.

Cuando se crea un **directorio o fichero** en LINUX, los permisos por defecto suelen estar establecidos por el administrador a los valores **777 y 666** respectivamente. Esto significaría:

```
777        rwx rwx rwx   directorios
666        rw- rw- rw-   ficheros
```

Estos valores por defecto se pueden cambiar utilizando una máscara que limite los permisos establecidos por defecto. Esta mascara suele estar establecida a **022 0 002** por el administrador.

Estas 3 cifras representan los permisos que se le eliminan al usuario, grupo y otros, en nuestro caso:

```
777 – 022 = 755 rwx r-x r-x     directorios
 666 – 022 = 644 rw- r--  r--     ficheros
```

En todo momento se puede visualizar la máscara que se está utilizando o cambiarla a otro valor. Este cambio solo afecta a la sesión actual. Para modificar el valor de umask de forma permanente será necesario incluir dicha configuración en /etc/profile o /etc/bash.bashrc afectando el cambio a todo el sistema; o en los ficheros ~/.profile o ~/.bashrc si se quiere aplicar el cambio para un usuario en concreto.

**Orden chown (Cambio del propietario de un fichero).**

chown [-opcion/es] nuevo_propietario fichero/s

Esta orden traspasa la propiedad de uno o más ficheros (se pueden usar comodines) a un nuevo propietario.

nuevo_propietario puede ser el login del mismo o su UID.

Opciones:

-h Si el fichero es un enlace simbólico, solo se cambia el propietario del enlace, si no se pone esta opción se cambia el enlace y el fichero referenciado.

-R Cambio Recursivo, en el caso de que se trate de un directorio, cambia toda su estructura.

Lo normal es que al cambiar de propiedad un fichero, se le mueva también (mv) al directorio del nuevo propietario.

Este tipo de órdenes las suele dar el administrador.

**Orden chgrp (Cambio de grupo de un fichero).**

chgrp [-opcion/es] nuevo_grupo fichero/s

Esta orden traspasa la propiedad de uno o más ficheros (se pueden usar comodines) a un nuevo grupo.

nuevo_grupo puede ser el nombre del mismo o su GID.

Opciones:

-h Si el fichero es un enlace simbólico, solo se cambia el propietario del enlace, si no se pone esta opción se cambia el enlace y el fichero referenciado.

-R Cambio Recursivo, en el caso de que se trate de un directorio, cambia toda su estructura.

Esta orden es posible que solo esté disponible para el administrador en algunos sistemas.

**Orden newgrp (Cambio de grupo de un usuario).**

newgrp [nuevo_grupo]

Esta orden cambia al usuario de grupo. El usuario ha de pertenecer previamente a ese grupo.

Usando newgrp solo, el usuario cambia al grupo principal.

nuevo_grupo puede ser el nombre del mismo o su GID.

Esta orden es posible que solo esté disponible para el administrador en algunos sistemas.

**REDIRECCIONES**

El redireccionamiento se utiliza para modificar la salida o entrada estándar, normalmente tras la ejecución de un comando en modo consola. La entrada estándar, la salida estándar y la salida de error se asocian a los programas mediante tres ficheros con los cuales se comunican con otros procesos y con el usuario. Estos tres ficheros son:

stdin (entrada estándar): A través de este descriptor de fichero los programas reciben datos de entrada. Normalmente stdin está asociado a la entrada del terminal en la que está corriendo el programa, es decir, al teclado. Cada descriptor de fichero tiene asignado un número con el cual podemos referirnos a él dentro de un script, en el caso de stdin es el 0.
stdout (salida estándar): es el descriptor de fichero en el que se escriben los mensajes que imprime el programa. Normalmente estos mensajes aparecen en la pantalla para que los lea el usuario. Su descriptor de fichero es el número 1. stderr (salida de error): es el descriptor de fichero en el que se escriben los mensajes de error que imprime
el programa. Normalmente coincide con stdout. Tiene como descriptor de fichero el número 2.

<

Redireccionamiento de entrada: Con él podremos introducir los datos que nos solicite el comando mediante un fichero o la salida de otro comando. Por ejemplo, el comando siguiente

cat < ficheros

>

Redireccionamiento de salida: Con él podremos enviar la salida de un comando hacia un fichero, otro comando o un dispositivo.

ls –l > fichero
ls –l > /dev/null >>
Redireccionamiento de adición: Es un redireccionamiento de salida con la peculiaridad de que la salida la añade al destino indicado, normalmente un fichero, y por tanto no borra su contenido.
man ls >> fichero

ls /etc >> /dev/tty4

**Orden sort (Ordenar archivos Texto):**

Se puede usar la orden sort para ordenar el contenido de un archivo por orden alfabético o numérico. Por defecto la salida se visualiza en el terminal, pero se puede especificar un nombre de fichero como argumento o redireccionar la salida un archivo.

La ordenación se efectúa siguiendo el criterio ASCII si no indicamos nada.

- sort [–opcion/es] archivo/s

Estas son algunas opciones de las opciones más importantes.

- b Se ignoran los espacios blancos iniciales en los campos (tabuladores en otras versiones).
- d Ordenación alfabética (diccionario). Ignora signos de puntuación y caracteres de control.
- f Ignora distinción entre mayúsculas y minúsculas.
-n Los números se ordenan por sus valores aritméticos en lugar de usar el primer dígito.
- o *fich* Almacena la salida en el archivo especificado.
- r Ordenación inversa (descendente).
-m Fusiona ficheros que esten previamente ordenados.
+*num* Ignora num campos y empieza a ordenar por el siguiente campo. Num puede ser 0. Esta opción admite subformatos:

+ numx - numy ❼ (Rango de campos). +num.posición ❼ (Columna dentro del Campo)
+numx.posicx -numy.posicy ❼(Rango de campos ampliado a columnas).

- u Elimina de la salida, las líneas repetidas que pudiera tener el archivo.
-t*carac* Usa el carácter especificado como delimitador de campo en vez del blanco o tabuladores.

*$ sort mifichero*
*$ sort mifichero –o ordenado*
*$ sort -fr -o ordenado mifichero*
*$ sort -t: +2 -uf –o nuevo miagenda*
*$ sort –u ca* > nuevo*
*$ sort arch1 arch 2 arch3 –u –o mezcla*

**La orden sort es un filtro** y como tal se puede usar, en este caso se toma como entrada de otra orden que muestre su información por la salida standard (stdout):

*$ ls –la | sort +2 $ ls*
*–la | sort –n +3*
*$ who | sort*

Algunos ejemplos:
***$ cat flores***
*Rosas*
*Petunias*
*Orquídeas margaritas*

*Begonias*

***$ sort flores*** *Begonias*
*Orquídeas Rosas*
*margaritas petunias*

***$ sort -f flores*** *Begonias*
*margaritas*
*Orquídeas*
*petunias*
*Rosas*

La opción -f hace que sort ignore las diferencias entre las mayúsculas o minúsculas de la misma letra.

***$ cat flores***
*12 Rosas*
*100 petunias*
*22 Orquídeas*
*4 margaritas*
*130 Begonias*

***$ sort flores***
*100 petunias*
*12 Rosas*
*130 Begonias*
*22 Orquídeas*
*4 margaritas*

***$ sort -n flores***
*4 margaritas*
*12 Rosas*
*22 Orquídeas*
*100 petunias*
*130 Begonias*

El primero produce una ordenación alfabética regular, el segundo muestra cómo -n produce el ordenamiento apropiado.

A menudo, cuando se ordena un archivo, se necesita sustituir el original por la versión ordenada.

*$ sort flores > flores.sort*
*$ mv flores.sort flores*

La opción -o del comando sort permite sustituir el archivo de entrada con la versión ordenada.

*$ sort -o flores flores*

**Orden uniq (Eliminar líneas repetidas):**

Elimina líneas de un fichero que estén repetidas y sean seguidas, esto implica que el fichero deberá estar ordenado previamente si no tiene sentido el uso de la orden..

Si solo deseamos eliminar las líneas duplicadas, se puede hacer con la orden sort con la opción –u si nuestro sistema soporta esta opción.

uniq [-opcion/es] ficheroentrada [ficherosalida]

Opciones:

-u Mostrar solo las líneas que son únicas
-d Mostrar solo las líneas que están duplicadas -c
Mostrar el nº de duplicaciones junto con cada línea.

Ejemplo:

*$ uniq mismarcas*
*$ uniq mismarcas marcas.unicas*

**La orden uniq es un filtro** y como tal se puede usar, en este caso se toma como entrada de otra orden que muestre su información por la salida estándar (stdout):

*$ sort datos | uniq > nuevosdat*

**Orden cut (Extraer columnas o campos):**

Esta orden se utiliza para extraer columnas o campos de cada una de las líneas de un archivo.

cut -opcion/es fichero

Opciones:

-dc Indica en c cuál es el carácter separador de campos, por defecto es el tabulador.

-clista Selecciona columnas de datos. Por ejemplo; -c10-15, significa "procesar las columnas de la 10 a la 15". Una columna es un carácter en una posición determinada de una línea.

-flista Indica que se quiere visualizar campos separados por caracteres separadores. Ejemplo: f1,3 significa "procesar los campos 1 y 3". Un campo es un grupo de caracteres cuya extensión está delimitada por un separador de campos. Esta opción siempre hay que usarla con –d. En el caso de que el separador sea un blanco se usan comillas –d” “.

Las (lista) no deben contener espacios para representar campos o caracteres. Se utiliza un guión (-) para indicar un rango y coma ( , ) para los valore individuales:

*1,2,3 representa los campos o caracteres 1, 2 y 3*
*1,2-5,9 representa los campos o caracteres 1,2,3,4,5 y 9.*

*3, es lo mismo que 1-3, representa los campos o caracteres 1,2,3. 3-, es lo mismo que 3-fin.*

**La orden cut es un filtro** y como tal se puede usar, en este caso se toma como entrada de otra orden que muestre su información por la salida estándar (stdout):

Ejemplo:

$ ls -l | cut -c40-48,66-

**Orden paste (Unir ficheros por columnas)**

Este comando es opuesto al comando cut. Une líneas de ficheros diferentes, o encadena líneas consecutivas en un mismo archivo.

- paste archivo1 archivo2...
- paste -dc list archivo1 archivo2 ...

Opciones:

-dc Por omisión, se utiliza el carácter tab como elemento de unión entre las líneas unidas. -dc permite reemplazar el carácter de tabulación como carácter separador por el carácter especificado por c. Puede especificarse el blanco (" ") como carácter separador.

Ejemplo, supongamos que disponemos de los siguientes ficheros:

***$ cat prov***
*Sevilla*
*Córdoba*
*Jaen*
*Huelva*
*Málaga*
*Almería*
*Granada*
*Cádiz*

***$ cat abrev.prov***
*SE*
*CO*
*J*
*H*
*MA*
*AL*
*GR*
*CA*

***$ paste prov abrev.prov > provincias***

```
Sevilla     SE
Córdoba     CO
```

```
Jaen        J
Huelva      H
Málaga      MA
Almería     AL
Granada     GR
Cádiz       CA
```

***$ paste -d: prov abrev.prov***
*Sevilla:SE*
*Córdoba:CO*
*Jaen:J*
*Huelva:H*
*Málaga:MA*
*Almería:AL*
*Granada:GR*
*Cádiz:CA*

El contenido de los archivos debe de coincidir tanto en número de entradas como en el orden, puesto que si no el resultado no sería el deseado.

**La orden paste es un filtro** y como tal se puede usar, en este caso se toma como entrada de otra orden que muestre su información por la salida standard (stdout):

Se puede utilizar cut y paste juntos para reorganizar y reordenar el contenido de un archivo estructurado. Un uso típico consiste en conmutar el orden de alguno de los campos del archivo.

*$ cut -f1,3 telef > temp*

*$ cut -f4 telef > temp1*
*$ cut -f2 telef | paste temp - temp1 > telef.nuevo*

Al final en el fichero *telef.nuevo* en cada línea contiene las columnas 1,3,2 y 4 del fichero *telef.*

Observar en el filtro paste, el uso que se le da al guión (símbolo menos) que hay entre *Temp.* y *temp1*, en la orden paste un símbolo – sin nada más representa a su entrada standard (que es la salida del filtro cut anterior).

**Orden join (Unir dos ficheros por registros):**

Este comando permite la unión de dos archivos, registro a registro siempre que el contenido de un campo en el primero coincida con otro en el segundo. Es similar a **paste**, pero **join** compara las líneas del campo clave en vez de juntarlas simplemente.

**join [-opciones] archivo1 archivo2**

Los archivos a unir deberán estar previamente clasificados en orden ascendente, según los campos por los que se desee comparar los archivos (se usará el comando **sort**, para conseguir la ordenación).

Por defecto, **join** utiliza el primer campo de cada archivo de entrada como campo común. Aunque, **join** permite utilizar otros campos como campo clave con la opción **-j**.

Algunas Opciones:

-jn m Utiliza el campo **m** del fichero **n** como el campo de unión. Si no se especifica **n**, se utiliza el campo **m** de cada archivo. Los campos van numerados de izquierda a derecha empezando por el número 1.
-tc Utiliza el carácter **c** como separador de campos en lugar del carácter por defecto.

Ejemplo:
***$ cat***
***animal.1***
*animal cerdo*
*caballo rana*
*arte*
*grande*
*barato*
*perro lana*
*algodón*

***$ cat***
***animal.2***
*animal*
*caballo*
*burro*
*pájaro*
*aguila*
*vencejo*

*perro*
*tela*
*hilo*

***$ join animal.1 animal.2***

*animal animal cerdo caballo*
*caballo rana*
*caballo burro*
*perro lana algodón tela*
*hilo*

Los dos archivos están ordenados ascendentemente según el campo clave. **join** compara las líneas de los dos archivos dando como resultado el campo común del archivo **animal.1** y el resto de sus líneas y el resto de las líneas del fichero **animal.2** que coinciden con el campo clave del primer fichero. El campo común no se repite, aparece sólo una vez en la salida.

Otro Ejemplo usando un campo de unión diferente y un separdor de campo distinto.

*$ join -t: -j1 3 -j2 1 archivoa archivob*

Añade a los registros de **archivoa** los campos de **archivob** siempre que el contenido del tercer campo del primero (archivoa) coincida con el primer campo del segundo (archivob). El campo comparado aparece el primero. Tanto a la entrada como a la salida, se utiliza el carácter ":" como separador de campos.

El carácter blanco es el separador por omisión.

**La orden join es un filtro** y como tal se puede usar, en este caso se toma como entrada de otra orden que muestre su información por la salida standard (stdout).

**Orden grep (Global Regular Expresión Print. Buscar una cadena en ficheros):**

Esta orden permite buscar cadenas de caracteres en los archivos que le indiquemos. Por defecto la salida es por pantalla (stdout) si no usamos un redireccionamiento o un piping.

- grep [-opcion/es] expresión_a_buscar [archivo/]

Si la expresión que se va buscar contiene más de una palabra se debe poner entrecomillada.

*$ grep "mi casaaaa" película.et*

Si se incluye más de un fichero, la salida de grep mostrará también el nombre del archivo además de las líneas que contienen la expresión a buscar:

*$ grep DNI agen1 agen2 agen3*
*$ grep kvdai12 **

Opciones:

-i Ignorar mayúsculas y minúsculas

-v Visualiza por pantalla aquellas líneas que no contienen el expresión a buscar.

-n Visualiza el nº de línea delante de la misma.

-c Cuenta solamente el nº de líneas que contienen la expresión.

-l Visualiza solo los nombres de los archivos que contienen la expresión.

Se pueden usar caracteres comodines en la expresión a buscar:

| Simbolos | Significad | Ejemplo | Identifica |
|---|---|---|---|
| **. (punto)** | Cualquier carácter en esa posición. | chil. | chili, chile |
| * | Cero ó más repeticion-es del carácter anterior. | ap*le | ale ó apple |
| **[ ]** | Cualquiera de los caracteres incluidos entre corchetes. | [Cc]asa | Casa, casa |
| ^ | Comienzo de línea. | ^Hiper | Hiper al comienzo de la línea. |
| **$** | Final de línea. | ado$ | ado al final de la línea. |

$ grep [Aa]randela piezas.*

$ grep 250\.00 piezas.*

$ grep "seat" coches

$ grep "^r" coches Líneas que comienzan por r

$ grep "^[^r]" coches Líneas que no comienzan por r

$ grep "ca$" paises líneas que finalicen en ca

$ grep "seat.*lento" Todas las líneas que coches contengan seat y lento independientemente del texto que haya entre ellas.

**La orden grep es un filtro** y como tal se puede usar, en este caso se toma como entrada de otra orden que muestre su información por la salida standard (stdout):

*$ file * | grep text* Todos los archivos del directorio actual que sean *de texto.*
*$ who | grep grupo1*
*$ ls -l | tee lista.fichero | grep sss*
*$ cut -d: -f1-3 paises | grep "europa" | more*
*$ cat coches | grep "seat" | tee datos | sort -t: +1*

**Ordenes egrep y fgrep:**

egrep (Extended grep) permite usar patrones de búsqueda que no permite grep, consultar la ayuda man de linux para más información de esta orden.

Uno de las extensiones de egrep más importantes es el uso de patrones múltiples con un O lógico usando las expresiones entre paréntesis separadas por el símbolo |:

*$ egrep "juan (perez | martinez | lopez | santiago)" /usr/yo/datos/empleados.dat*

Otra de las extensiones es que el patrón de búsqueda puede estar incluido dentro de un fichero, usando la opción *– f fichero:*

*$ cat misdatos*
*tuercas*
*arandelas*
*tornillos*
*cable*
*martillo*

*$ egrep –f misdatos alamacen > encontrados*

**Orden wc (Contar palabras):**

Esta orden cuenta el nº de caracteres, palabras o líneas de uno mas archivos. Si no se indica un fichero lee la entrada standard (stdin). Obviamente esta orden solo se debe utilizar con archivos de texto.

- wc [-opcion/es] fichero/s

*$ cat prueba Buenos dias Esto*
*es una prueba.*

*$ wc prueba*

*2 6 32 prueba*

Al contar, el comando **wc** entiende que una línea está delimitada por el carácter (\n), una palabra por espacios, tabuladores o (\n). El carácter \n equivale a retorno de carro).

El carácter \n de fin de línea también es contado, por eso en el ejemplo anterior salen 32 letras en vez de 30, el nº de caracteres reales serian el (nº_car – nª lineas.)

Opciones:

-l Informa solo del número de líneas.
-w Informa solo del número de palabras.
-c Informa solo del número de caracteres.

Si no se especifica ninguna opción, aparecerán en pantalla y por este orden (**-lwc**) el número de líneas, palabras, caracteres y nombre del archivo o archivos.

**La orden wc es un filtro** y como tal se puede usar, en este caso se toma como entrada de otra orden que muestre su información por la salida standard (stdout):

$ who | wc –l ❼ Nº de usuarios conectados al sistema

**Orden head (Acceder a la parte inicial de un fichero):**

Esta orden muestra por defecto las 10 primeras líneas de un fichero.

- head [-nc] fichero/s

Si se usa la opcion –nc, se puede indicar con c el nº de líneas a mostrar:

*$ cat misdatos*
*tuercas*
*arandelas*
*tornillos*
*cable*

*$ head – n2 misdatos*
*tuercas*
*arandelas*

Puede que en algunas revisiones UNIX esta orden no esté dentro del path del usuario.

**La orden head es un filtro** y como tal se puede usar, en este caso se toma como entrada de otra orden que muestre su información por la salida standard (stdout).

**Orden tail (Acceder a la parte final de un fichero):**

Esta orden muestra por defecto las 10 últimas líneas de un fichero.

- tail [-nc] fichero/s o tail [+nc] fichero/s

Con la opcion –nc, se puede indicar con c el nº de líneas finales a mostrar:

*$ cat misdatos*
*tuercas*
*arandelas*
*tornillos*
*cable*

*$ tail –n2 misdatos*
*tornillos*
*cable*

Con la opción +nc, se puede indicar con c el nº de líneas a mostrar desde el principio (como head):

*$ cat misdatos*
*tuercas*
*arandelas*
*tornillos*
*cable*

*$ tail +n3 misdatos*
*tuercas*
*arandelas*
*tornillos*

Comprobar a que es debido que no funcione (En el aula)

La orden tail también admite las opciones:

–c para usar con palabras
–b para usar con bloques (un bloque normalmente son 512K).

**La orden tail es un filtro** y como tal se puede usar, en este caso se toma como entrada de otra orden que muestre su información por la salida standard (stdout).

*$ who | tail –n1*

**tr**

Cambia, borra o comprime caracteres.

- "set1" "set2": va cambiando cada carácter de set1 por el correspondiente de set2.
- -d: borra caracteres indicados.
- -s: comprime a uno sólo secuencias repetidas de los caracteres indicados.
- -c: complementa el conjunto de caracteres indicados.
- [:lower:] [:uppe:] convierte en mayúsculas los caracteres en minúsculas

Veamos un par de ejemplos o tres:

**tr ':' ' ' < /etc/passwd > ficheropasswd tr '[a-z]' '[A-Z]' <> listaalumnosmayusculas tr ' ' '\n' <> lineasusuarios tr -s " " <> prueba2**

**Orden more (Visualización página a página de un fichero.):**

- more fichero/s

Comandos:

| | | |
|---|---|---|
| ENTER | para avanzar una línea. | |
| B.ESPACIADORA | para avanzar una página | |
| Ctrl-B | ir a página anterior | ** |
| G | ir al final | ** |
| 1G | ir al principio | ** |

Puede no estar disponibles en

**OPCIONES:**

| | |
|---|---|
| **-c** | **Limpia la pantalla antes de mostrar.** |
| -e | Salir inmediatamente después de escribir la última línea del último archivo en la lista de argumentos. |
| -n | Especifica cuántas líneas se muestran en la pantalla para un archivo dado. |
| +n | Inicia el archivo desde el número dado. |

Ejemplo: *$ more carta.rollo*

**Orden less (Visualización página a página de un fichero):**

- less [-opcion/es] fichero/s

**OPCIONES:**

| | |
|---|---|
| **-c** | **Limpia la pantalla antes de mostrar.** |
| +n | Inicia el archivo desde el número dado. |
| :p | Examina el archivo previo en la lista de línea de comandos. |
| :d | Elimina el archivo actual de la lista de archivos. |

Algunos Comandos:

<return> Avanza una pantalla.

| | |
|---|---|
| +n | Avanza n pantallas. |
| -n | Retrocede n pantallas. |
| n | Presenta la pantalla n. |
| +nl | Avanza n líneas. |
| -nl | Retrocede n líneas. |
| nl | Presenta la línea n. |
| n/patron/ | Busca hacia delante hasta la aparición de patrón. Por defecto el valor de n es 1. La búsqueda comienza con la siguiente pantalla y continúa hasta el final del fichero. |
| n?patron? | Busca hacia atrás hasta la aparición de patrón. El valor por defecto es 1. La búsqueda comienza justo antes de la actual pantalla y continúa hasta el principio del fichero. |
| in | Ir al siguiente fichero i. El valor por defecto de i es 1. |
| ip | Ir hasta el fichero i anterior. El valor por defecto es 1. |
| q | Abandona el comando pg. |

*$ less carta.rollo*

**Las ordenes more y less son filtros** y como tal se puede usar, en este caso se toma como entrada de otra orden que muestre su información por la salida standard (stdout).

*$ ls –la | more*

*$ cat micarta | more*

*$ cat micarta | less*

*$ more micarta*

***$ less micarta*** ← Mejor

***La orden find;***

Hemos dejado la orden find para el final de filtros y pipes, en realidad find no es un filtro. find es una de las herramientas más potentes que posee el UNIX, pero también una de las que tiene una sintaxis mas compleja.

Este comando se utiliza para examinar toda la estructura de directorios, o la parte que le indiquemos, buscando los archivos que cumplan ciertos criterios y después ejecutar acciones sobre ellos.

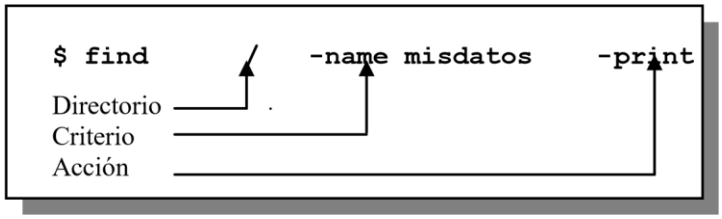
La sintaxis es compleja pero se puede resumir así:

- **find directorio criterio acción**

Donde

- directorio: Es el nombre del directorio o directorios de comienzo de la búsqueda. Una vez iniciada ésta desde los puntos indicados, continuará por todas las ramas existentes, hasta llegar al final de cada una.

- criterio: Es el modo de seleccionar los archivos.
- acción: Qué hacer con los archivos seleccionados




***Muchas veces se recomienda usar la orden find como un proceso en background (2º plano) dado la cantidad de tiempo que puede tomar en algunos casos para realizar una búsqueda.***

1.- A continuación se explican las principales opciones de criterios de búsqueda:

- name *nombre* Buscar ficheros con ese nombre. Se pueden usar comodines.

*$ find / -name misdatos -print*
*$ find /usr/vdai03 -name '*.c' -print*

- user *usuario* Seleccionar archivos que pertenezcan a un usuario de terminado.

-group *grupo* Seleccionar archivos que pertenezcan a un grupo de terminado.

$ find /usr –user vdai10 –print
$ find /usr –user 56 –print
$ find /usr –group mifila –print
$ find /usr –group 234 –print

-type x Muestra los ficheros especificados por x. Donde x puede ser:

b Archivo especial tipo bloque c
Archivo especial tipo carácter
d Directorio
f
Fichero
normal
l Enlace simbólico

-perm máscara Busca ficheros que tengan la mascara de permisos indicada.

-newer archivo Busca aquellos archivos que hayan sido modificados después que "archivo".

-atime n Busca ficheros a los que se ha accedido por última vez en n dias
-atime -n Busca archivos que han sido accedidos hace menos de n días.
-atime +n Busca archivos que han sido accedidos hace más de n días.

-mtime n Busca archivos que han sido modificados hace n días.

| | |
|---|---|
| -mtime -n | Busca archivos que han sido modificados hace menos de n días. |
| -mtime +n | Busca archivos que han sido modificados hace más de n días. |
| -ctime n | Busca archivos cuyo modo ha sido modificados hace n días. |
| -ctime -n | Busca archivos cuyo modo ha sido modificado hace menos de n días. |
| -ctime +n | Busca archivos cuyo modo ha sido modificado hace más de n días. |
| -size n | Busca todos los archivos con n bloques (512 bytes). |
| -size +n | Busca archivos con más de n bloques (512 bytes). |
| -size -n | Busca archivos con menos de n bloques (512 bytes). (En UNIX un bloque suele ser 512K y es unidad de almacenamiento). |
| -size nc | Busca todos los archivos con n caracteres. |
| -size +nc | Busca archivos con más de n caracteres. |
| -size -nc | Busca archivos con menos de n caracteres. |
| -links n | Busca todos los archivos con n enlaces duros |
| -links +n | Busca todos los archivos más de n enlaces duros. |
| -links - n | Busca todos los archivos menos de n enlaces duros. |
| -inum n | Busca archivos con un numero n de i-nodo. |
| -level 0 | No examina ningún subdirectorio |
| -level 1 | Examina un nivel de profundidad de directorios |
| -level2 | Examina 2 niveles de profundidad de directorios. |

2.- Los siguientes operadores se pueden usar con las opciones de búsqueda anteriores:

- ! Selecciona los archivos que no coincidan con una opción.

| | |
|---|---|
| *$ find / -type f -print* ❼ | Todos los archivos ordinarios |
| *$ find / ! -type f -print* ❼ | todos los archivos que no son ordinarios |

- Si se ponen varias opciones de búsqueda se considera un AND lógico:

*$ find -name alpha –size +50 user juan -print*

Busca en el disco duro todos los ficheros que se llamen alpha y que tengan mas de 50 bloques y que sean del usuario juan

- Si se ponen varias opciones de búsqueda separadas por –o se considera un OR lógico:

*$ find -name alpha -o user juan -print*

3.- Tipos de acción:

-print Muestra en pantalla el archivo(s) en su forma pathname completo (el nombre de ruta de acceso completo).
En los sistemas UNIX actuales es la opción por defecto y no es necesario especificarla.

-exec xx Si la condición de búsqueda se cumple, se ejecuta el comando xx, la ejecución del comando afecta al fichero(s) evaluado.

A continuación del comando separado por un blanco se colocan dos llaves {} un blanco y una secuenca de escape con un punto y coma \; esto se sustituye por el nombre del archivo (con su ruta) que se está evaluando.

*$ find /usr/vdai01 -name '*.c' -exec cat {} \;*

Teóricamente la orden anterior visualizaría los archivos con extensión c que se hallan situados en /usr/vdai01

-ok xx Exactamente igual que – exec, pero pide confirmación al usuario para cada búsqueda acertada.

**Notas:**

**Si no se especifica nada –print es la opción por defecto (excepto en versiones antiguas de UNIX en las que es obligatorio ponerlo)**

**Se puede fusar find en redirecionamientos y como salida para un encaucamiento (pipe).**