Introducción al entorno Unix

Introducción

Consultar en la wikipedia “open source” y “licencias GPL”.

Empezando…

Hoy en día, todos los computadores ofrecen un interfaz gráfico manipulable con el ratón o con una pantalla táctil. La mayoría de aplicaciones y herramientas se manejan de esa manera.

Sin embargo, antes de los entornos gráficos los computadores sólo eran capaces de mostrar texto o caracteres alfanuméricos. El manejo del ordenador requería muchas veces escribir órdenes y leer los resultados que no eran más que líneas de texto en una pantalla. El programa o entorno que el usuario tenía para escribir estas órdenes y ver su respuesta se llamaba "terminal". El motivo es que los computadores eran muy grandes y tenían varios puestos de trabajo para distintos usuarios simultáneamente (al contrario que hoy , que son todo computadoras personales). De ahí el nombre de "terminal".

De aquellos días han perdurado muy pocos entornos o aplicaciones que mantengan esta forma de trabajar. Pero uno de los que ha perdurado y sigue siendo utilizado es el terminal del intérprete de comandos de unix.

En este tutorial vamos a realizar una introducción a este entorno y, a partir de él, a la gestión y administración de ficheros, directorios, entrada y salida de datos, ejecución de programas, gestión de usuarios y permisos, etc. Muchos de los conceptos y habilidades explicados aquí parecen inútiles, sin embargo, en la programación de guiones que se estudia más adelante se requiere dominar estos conceptos.

Usuarios, *login* y *password*

Al igual que Unix, Linux es un sistema operativo multiusuario y multitarea.

- Multitarea: puede realizar más de una tarea a la vez
- Multiusuario: el sistema puede ser empleado por más de un usuario. Por ser multiusuario,

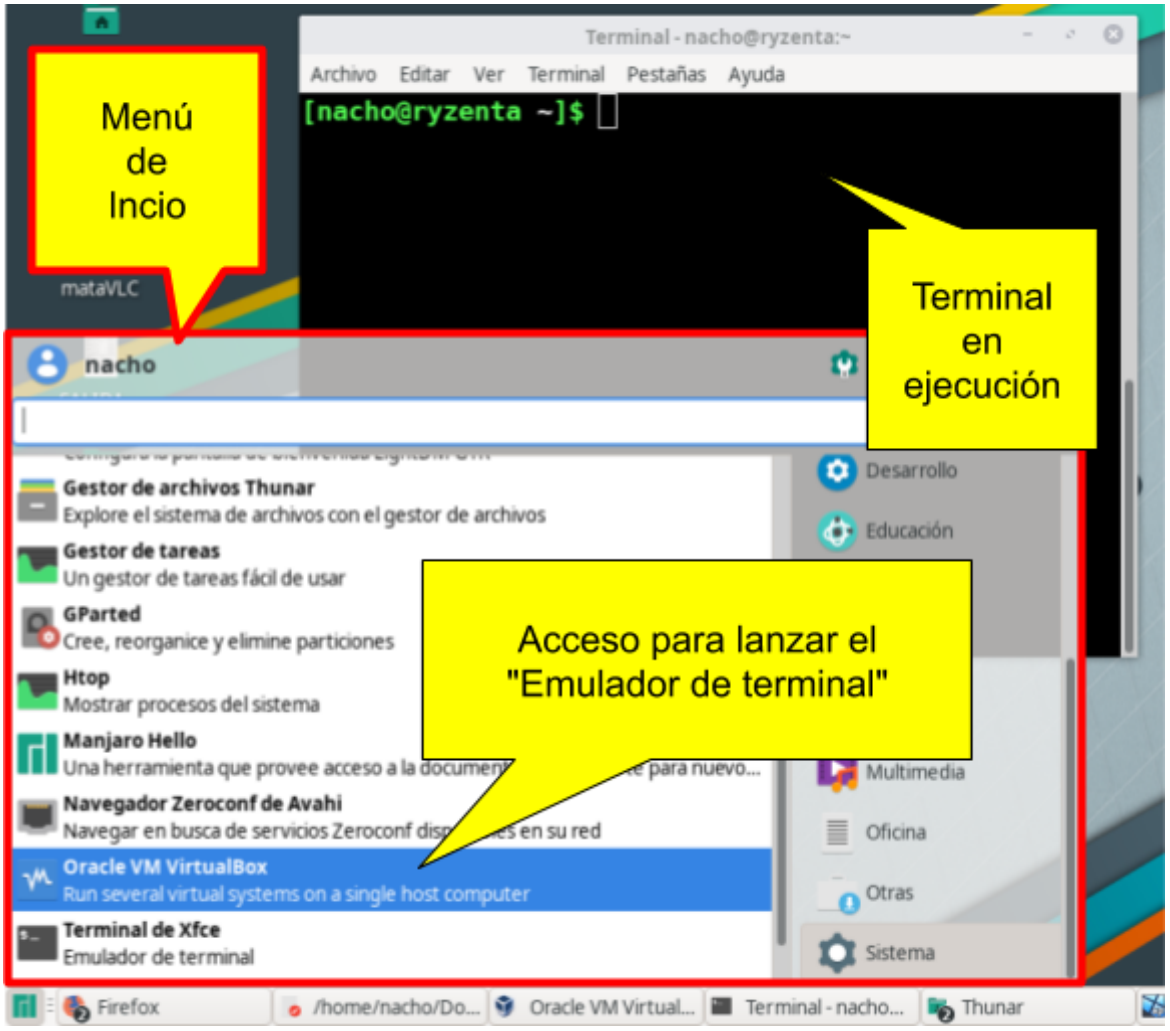
Unix diferencia a los usuarios y su información. Para lograrlo, a cada usuario se le asigna una identificación única (en inglés login), una clave de acceso (en inglés password) y un espacio independiente donde guardar sus archivos.

Para usar un sistema Linux, una persona debe estar autorizada y registrada en el sistema. Conocidos el nombre de usuario y la clave, un usuario puede introducirlas y empezar a usar el sistema.

Terminales virtuales.

Cuando arranque linux, seguramente estarás delante de un entorno gráfico. En él suele haber un menú de inicio que está repleto de iconos y entradas para facilitar el lanzamiento de programas desde él. Necesitarás localizar en tu distribución y entorno dónde aparece el programa "terminal", "Emulador de terminal", "QTerminal", "lxterm", "shell" etc. Para así, poder lanzar un terminal y seguir con el resto de la lección.

El emulador de terminal es la ventana, con menús, marco y demás que en su interior muestra texto. Un posible aspecto de todos los elementos anteriores aparece en la siguiente captura de pantalla.



Mientras que "el emulador de terminal" es la ventana con sus menús, marcos, etc, lo que muestra en su interior es el programa que atiende a la escritura de comandos y responde al usuario. Este programa es "bash" y es el que debemos aprender a usar. Los menús de dicha ventana no son importantes. Todo lo que vamos a hacer es trabajar con el interior de esta ventana, o sea con "bash"

Elementos básicos del interfaz de comandos

El prompt y la introducción de comandos

El intérprete de órdenes desplegará algún indicador o *prompt*. Este indicador avisa de que el intérprete espera que el usuario escriba una orden para ejecutarla. La orden se introduce tecleando y después se ejecuta al pulsar **[intro]**. Si la orden tiene algún resultado que mostrar, generalmente veremos en pantalla el resultado y después, otra vez el prompt invitándonos a teclear el siguiente comando.

El prompt tiene una forma variable y configurable por el usuario, pero en la mayoría de sistemas viene preconfigurada de la siguiente forma:

```
usuario @ máquina directorio símbolo
```

Ejemplos

```
[pepe@servidor] home$
[root@ordenador23 fotos]#
```

El prompt muestra el nombre de usuario conectado en el terminal, una arroba y a continuación, el nombre de la máquina. Esto tiene su fundamento en que Unix está diseñado para que los usuarios puedan trabajar remotamente, en otras máquinas distantes, y por ello, es de ayuda situar continuamente al usuario en la máquina en la que está.

El directorio aparece a continuación, de manera que el usuario tiene más fácil saber dónde se están ejecutando sus comandos en todo momento.

El último símbolo separa el prompt de la línea donde introduce el usuario los comandos y puede ser un “dolar” ($) si se es un usuario normal o una almohadilla (#) si se es un usuario con privilegios de administrador (root).

Como regla general, si la última línea del terminal muestra el prompt (fíjese en el símbolo “$”) entonces es que el sistema está esperando que usted introduzca un comando. Si no observa el prompt, estará haciendo otra cosa o esperando que introduzca algo que no es un comando

Caracteres de control.

Se denominan caracteres de control a aquellos que producen un efecto inmediato cuando se pulsan Los más importantes son:

- <Crtl> c: Termina o aborta la ejecución de la orden que se esté ejecutando
- <Crtl> d: Es utilizado por aquellos programas que aceptan datos desde teclado para indicar el final de los datos.
- <Ctrl> + z : Detiene el proceso en ejecución pero no lo mata

Sintaxis de los comandos

La forma de introducir los comandos sigue unas reglas (reglas de escritura = "sintaxis") bastante rígida pero simple. Se permite indicar el comando y un número cualquiera de argumentos o parámetros que lo acompañan y detallan la acción del comando.

```
nombre_comando arg1 arg2 arg3 arg4...
```

o también

```
nombre_comando [args]
```

Ejemplo: El comando echo se utiliza para mostrar por pantalla frases o palabras introducidas como argumentos. Pruebe su ejecución:

- `echo hola`
- `echo hola me llamo Rodrigo`

Pruebe ahora:

- `echoholamellamoRodrigo`

Cuando se introduce un comando, se escribe primero el nombre del comando, a continuación y mediando siempre una separación con un carácter en blanco (un espacio) los argumentos que se requiera para la ejecución del comando.

Si un argumento lleva espacios, entonces surge un conflicto, pues el espacio indica *separación de argumentos*. Se debe entrecomillar el argumento para que se tome como un único argumento y no como varios. Por ejemplo, supón que tienes un archivo llamado "facturas emitidas" y quieres lanzar un comando sobre él:

MAL: `ls facturas emitidas`

BIEN: `ls “facturas emitidas”`

Ejecución del comando.

Una vez escrito el comando, junto con sus argumentos, la pulsación de la tecla enter desencadenará su ejecución. Es posible que la ejecución del comando tome su tiempo o que el comando se quede a la espera de que el usuario escriba datos. En ambos casos el comando no finaliza rápidamente y el intérprete de comandos se espera a que finalice para continuar.

Veamos esto con un ejemplo: Introduzca el comando cat y observe que el prompt no aparece

```
[pepe@servidor] cat
```

El comando cat está a la espera de datos, el proceso está en marcha y por ello bash no muestra el prompt. Ahora pulse `[ctrl + c]` (interrumpe el comando cat)

```
pepe@servidor cat
^C
pepe@servidor
```

Después de pulsar ctrl+c el prompt vuelve a aparecer porque el programa “cat” ha terminado (aunque lo ha hecho porque lo hemos matado)

Normalmente los comandos terminan enseguida y generalmente muestran algún resultado por la pantalla. Después aparece otra vez el prompt. Pruébelo introduciendo el comando “echo hola”. ¿Vé cómo aparece el prompt enseguida?

Introducción de varios comandos

El símbolo ";" termina totalmente la escritura de un comando y sus argumentos. Es un símbolo que puede usarse para separar varios comandos que se escriben en la misma línea. Gracias a este símbolo, los comandos independientes que tendrían que escribirse en varias líneas se pueden escribir de golpe en una línea y se ejecutarán uno detrás de otro

```
$ echo uno; echo  dos ; echo tres;
```

Produce la siguiente salida :

```
uno
dos
tres
```

La línea anterior equivale a la siguiente secuencia de comandos (con sus respuestas):

```
$echo uno;
```

```
uno
$echo dos
dos
$echo tres
tres
$
```

Ojo! El hecho de que el carácter ";" se use para separar comandos significa -desgraciadamente- que no puede ser usado de la misma manera para otros objetivos. Observa el siguiente comando y su resultado

```
lliurex@1smx:~$ echo Hoy voy a almorzar; y después a dar clase
Hoy voy a almorzar
y: no s'ha trobat l'ordre
lliurex@1smx:~$
```

Como ves, hay un error en ese comando y no es el "echo". El error está originado por el carácter ";", al escribirlo, bash entiende que el comando termina ahí y lo que viene a continuación ("y después… ") Es el siguiente comando y sus argumentos.

Tipos de argumentos.

1. **Modificadores**. Generalmente los comandos pueden realizar acciones que son genéricas (por ejemplo, informar de la hora). A la hora de utilizarlos, el usuario puede necesitar matizar o especificar detalladamente la acción del comando (por ejemplo: informar de la hora y la fecha en un formato especial).

   En dichos casos, los comandos pueden llevar un argumento que especifique o concrete la acción precisa que se desea del comando.

   Ejemplos:

   - El comando "`rm`" borra ficheros, pero por defecto no pregunta si el usuario está seguro. Se puede incluir un modificador para que el comando pregunte antes de borrar
   - El comando "`ls`" lista los nombres de ficheros existentes en un directorio. Mediante el modificador `-l` se obtiene un listado donde se pueden ver los atributos.
   - El comando "`ls`" muestra los ficheros ordenados en orden alfabético según su nombre. Existen opciones para ordenarlos for otros atributos como fechas de creación, alteración, etc., por tamaño, etc.

2. **Ficheros o elementos sobre los que se aplica el comando**. Una gran parte de comandos actúa o realiza acciones sobre ficheros, usuarios, datos, etc.

   Cada argumentos es un fichero, un usuario, un directorio, etc. Cada argumento es el objeto sobre el que se aplica el comando. Si pueden llevar

ninguno, uno o varios argumentos de este tipo, depende del comando en concreto.

- Ver el contenido del un fichero: "`cat nombre_fichero`"
- Borrar varios ficheros: "`rm fichero_1 fichero_2 fichero_3`"
- Dar de alta al usuario "juanito" en el sistema con el password "bartolo":

  "`useradd –p bartolo juanito`"

Rodillo de comandos

Se pueden consultar y repetir los comandos ejecutados mediante las teclas "flecha arriba" y "flecha abajo" del teclado. Esto nos sirve tanto para repetir un comando ya ejecutado, como para recordar y revisar qué se ha hecho con el computador cuando hay algún problema o anomalía.

Autocompletado de comandos y ficheros

Bash, siguiendo las reglas de sintaxis de los comandos y argumentos, y suponiendo que todos los argumentos representan ficheros y directorios, es capaz de autocompletar durante la escritura, los nombres de los ficheros y directorios.

Ejemplo: Ejecute el siguiente comando

```
mkdir SoyUndirectorio
```

Teclee exactamente ésto:

```
cd SoyU
```

Si a continuación, pulsa la tecla del tabulador, verá cómo se autocompleta el comando y aparece:

```
cd SoyUndirectorio
```

El autocompletado en ubuntu es fantástico. Autocompleta o sugiere argumentos adecuados para cada comando. Así, por ejemplo, con el comando "dig" completará direcciones ipv6 e ipv4 de interfaces de red locales .

Ejecución de los comandos y mensajes de resultado

Si un comando se ejecuta con éxito, no suele haber ningún mensaje de salida indicando que todo ha ido bien.

Sin embargo, si algo va mal, el comando (o el intérprete de comandos) muestran algún mensaje de error. Observa la siguiente secuencia de comandos capturada de un terminal

```
nacho@dhcppc1:~> ls
```

```
Documents  bin  fichero_prueba  public_html
nacho@dhcppc1:~> rm fichero_prueba
nacho@dhcppc1:~>
nacho@dhcppc1:~> echo hola
hola
nacho@dhcppc1:~> rm fichero_prueba
rm: cannot remove `fichero_prueba': No such file or directory
nacho@dhcppc1:~>
```

El comando "ls" ha mostrado un resultado por pantalla, mientras que el siguiente comando "rm fichero_prueba" no ha sacado nada. Ambos han tenido éxito. El comando "echo" también acaba bien. Pero el último comando "rm fichero_prueba" muestra un mensaje de error porque algo ha ido mal

2 Ayuda. El comando 'man' (opcional)

El comando man sirve para obtener información sobre la manera de usar los distintos comandos disponibles. Para obtener dicha información basta con invocar al comando de la siguiente manera:

```
man nombre_de_comando
```

El resultado de esta invocación es la aparición en la pantalla de un texto de ayuda (que puede estar en inglés). A este texto se le llama “página del manual”, las cual tiene unas secciones que aparecen en todas las descripciones de uso de la mayoría de comandos:

- **Referencia a la página:** Indica en qué página del manual está este comando
- **Nombre**: El nombre es quizá conocido, porque ha sido usado para invocar la página de ayuda que se está consultando. En ocasiones, sin embargo, una página del manual recoge descripciones para varios comandos idénticos o similares en acción. Se puede comprobar aquí, por tanto, qué comandos van a ser descritos a continuación.
- **Sinopsis**. Una breve referencia de cómo invocar el comando y sus argumentos, cuáles son obligatorios, cuáles opcionales y el orden que deben seguir.
- **Descripción**: Una sencilla explicación de cuál es el objetivo del comando; para qué se usa y cuáles son sus principales efectos
- **Opciones**: Un listado, en ocasiones extenso, de todos los argumentos modificadores, junto con sus opciones, detalles y efectos sobre la acción del comando.
- **Ficheros**: Algunos comandos complejos están configurados por ficheros, o bien los utilizan durante su ejecución. El usuario puede saber qué ficheros son, dónde están y para qué se usan.(ej: man adduser)
- **Véase También**: Otras páginas o comandos muy relacionados con el comando

consultado

- **Bugs**: Descripciones limitadas de errores de funcionamiento
- **Autor**: Autor

En teoría, cualquier programa instalado, debe instalar también su página del manual explicando su uso y descripción. Es la manera por defecto de que los programas instalen ayuda

Consultar la página del manual requiere algunas teclas para desplazarse por el mismo:

La información se proporciona paginada por pantallas. Al final de la pantalla la indicación `--More--` interroga si se desea avanzar a la siguiente página. Se puede contestar:

- **<Space>** (Barra espaciadora): Avanzar a siguiente página.
- **q** (quit): Abandonar
- **?** ó **h** (help). Para ver otros mandatos disponibles.

Ejercicio: Buscar la manera de realizar un listado en el que se muestren los atributos de los ficheros y éstos salgan ordenados por fecha de creación

Synopsis (Sintaxis de las opciones)

En la página del manual se describe las reglas básicas con las que se pueden componer las diferentes opciones de los comandos. Aunque cada comando es muy particular y tiene sus propios argumentos que deben formarse arreglo a una lógica única, existen algunas reglas aplicables a la mayoría de comandos que son mostradas en este apartado del manual.

Existe un convenio de signos y anotaciones donde se indica:

- La obligatoriedad de que aparezca algún argumento
- El contenido de algunos argumentos
- La precedencia de algunos argumentos sobre otros
- La complejidad de algunos grupos de argumentos, que deben ser estudiados cuidadosamente
- La posibilidad de usar el comando de dos o más formas diferentes, en cuyo caso hay combinaciones de argumentos diferentes.
- La repetición posible de algunos argumentos.

Orden y precedencia de los argumentos

La mayor parte de veces, los comandos llevan parámetros modificadores y objetos sobre los que deben actuar. Muchos comandos esperan que las opciones estén escritas al principio y después vengan los ficheros, patrones u otros elementos. El orden en el que aparecen las palabras la sinopsis es importante. Por ejemplo, El comando grep awk tiene la siguiente línea en la sinopsis

```
SYNOPSIS

     awk [-F sepstring] [-v assignment]... program [argument...]
```

La opción -F es la primera y lleva a continuación un argumento de tipo "sepstring", desupés viene la opción -v con su parámetro acompañante "assignment". A continuación, un argumento denominado "program" y finalmente, un argumento llamado "argument". Estos dos últimos argumentos tienen un significado particular que debe estudiarse en los otros apartados del manual.

Usos alternativos del comando

Si un comando puede ser usado de dos formas muy diferentes, seguramente aparecerán dos o más líneas en la sinopsis, ya que cada una puede variar mucho de la otra y vale la pena indicarlo. Por ejemplo el comando fdisk se usa para particionar discos duros en una sesión interactiva con menús y comandos propios. Sin embargo, el comando puede ser usado para una función puntual y diferente de particionar un disco. Por ello existe dos líneas distintas en la sinopsis

```
SYNOPSIS
     fdisk [options] device
     fdisk -l [device...]
```

Obligatoriedad

Cuando un comando requiera llevar un argumento concreto necesariamente, éste aparecerá reflejado en la synopsis. En el caso del comando fdisk anterior. Si no se usa para listar los discos del sistema, el usuario debe escribir un dispositivo para que fdisk trabaje con él. Por tanto, en la línea que describe este uso, la palabra "device" aparece sin más.

```
SYNOPSIS
     fdisk [options] device
```

Sin embargo, las opciones que el comando puede llevar son todas prescindibles (al menos para que el comando fdisk pueda cumplir un mínimo de funcionalidad), por ello, la no obligatoriedad de escribir algún argumento se indica entre corchetes.

El comando awk visto antes tiene como obligatorio el argumento "program"

```
SYNOPSIS
      awk [-F sepstring] [-v assignment]... program [argument...]
```

Repetición

los puntos suspensivos indican repetición de argumentos

Exclusión mútua de opciones

la barra vertical "|" indica que se dos opciones son incompatibles entre sí. **Ejemplo** comando faad

```
NAME
      faad – Process an Advanced Audio Codec stream
SYNOPSIS
   faad [options] [-w | -o <output_filename> |-a <output_filename>]
      input_filename
```

Se pueden usar la opción "-w", "-o" o "-a", pero sólo una de ellas, seguidas de algún parámetro particular de la opción elegida. Seguida y obligatoriamente debe aparecer el argumento "input_filename" que se trata (por el nombre) de un archivo de entrada

ls [OPTION]... [FILE]...

descubrir el significado de ls -a y todas sus implicaciones

3 Listado y movimiento entre directorios

Se supone conocido el concepto de directorio o carpeta. Para tratar el tema de la manipulación y navegación de directorios bajo Linux, vamos a dar unas definiciones previas.

Directorio raíz y rutas absolutas

Como es conocido, los directorios forman relaciones anidadas jerárquicas en las qué unos directorios están dentro de otros. Existe un, y sólo uno, directorio que no está contenido dentro de ningún otro. Es el **directorio raíz**, del cual cuelgan directa o indirectamente todos los demás.

Para especificar o nombrar un fichero concreto, es posible indicar el nombre de todos los directorios que hay que atravesar partiendo de este directorio raíz hasta llegar al fichero en cuestión. Este nombre compuesto a su vez de nombres de directorios se denomina **ruta**.

Por ejemplo, si dentro del directorio raíz existe un directorio llamado “facturas”, y

dentro de éste último existe un subdirectorio llamado “pagadas”, entonces un fichero ubicado en este último y llamado “camioneta” sería referenciado con el siguiente nombre:

```
/facturas/pagadas/camioneta
```

Esta ruta es **absoluta** porque determina desde el origen del sistema de ficheros la ruta completa. Más adelante se explica qué son las rutas relativas.

Las rutas absolutas se reconocen porque empiezan con el carácter “barra”: “/”.

Directorio actual. Rutas relativas

Durante las sesiones, el usuario ejecuta diversos comandos. Éstos actúan generalmente sobre ficheros. Es posible especificar siempre la ubicación precisa de los ficheros mediante rutas absolutas; pero hay una manera más sencilla de operar.

Para facilitar la operación del usuario, el intérprete de comandos sitúa la ejecución de los comandos en un directorio por defecto. Se puede pensar que el usuario “está virtualmente” en dicho directorio y allí es donde actúan los comandos y órdenes. Este es el “**directorio actual**”

Si deseamos que los comandos actúen sobre directorios fuera de nuestro directorio actual se puede seguir utilizando rutas absolutas, pero ahora también, existe la posibilidad de usar **rutas relativas** en las que la especificación de la ubicación de un directorio no parte desde la raíz, como en las rutas absolutas, sino que parte del directorio actual.

Por ejemplo si el directorio actual es “`/home/juan`”, entonces una ruta relativa al directorio “`/home/juan/facturas/nuevas`” sería:

```
facturas/nuevas
```

Las rutas relativas se reconocen porque no empiezan por el carácter “barra” “/”

Cambio de directorio actual.

El usuario puede necesitar cambiar de directorio actual. Por ejemplo, si un usuario desea revisar y manipular unos ficheros que están situados en el directorio `/home/pepe/facturas/facturas2006`, le será más fácil situarse allí y escribir los comandos acompañados del nombre de los ficheros directamente. De esta manera el comando

```
cat /home/pepe/facturas/facturas2006/factura_0392
```

se convierte en el comando

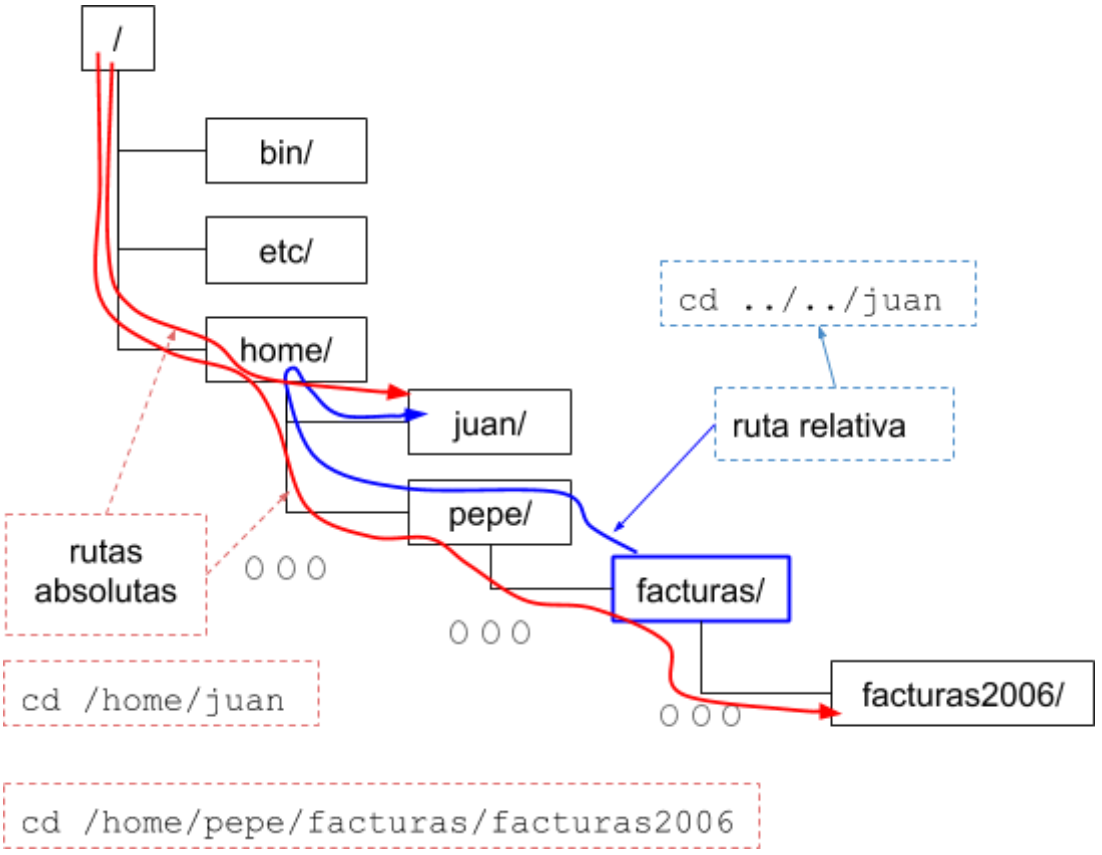
```
cat factura_0392
```

Para cambiar el directorio se utiliza el comando cd acompañado de la ruta (absoluta o relativa) del directorio al que se desea cambiar. En el caso anterior:

```
cd /home/pepe/facturas/facturas2006
```

o, si estamos en `/home/pepe`

```
cd facturas/facturas2006
```



Directorio de inicio o “home”

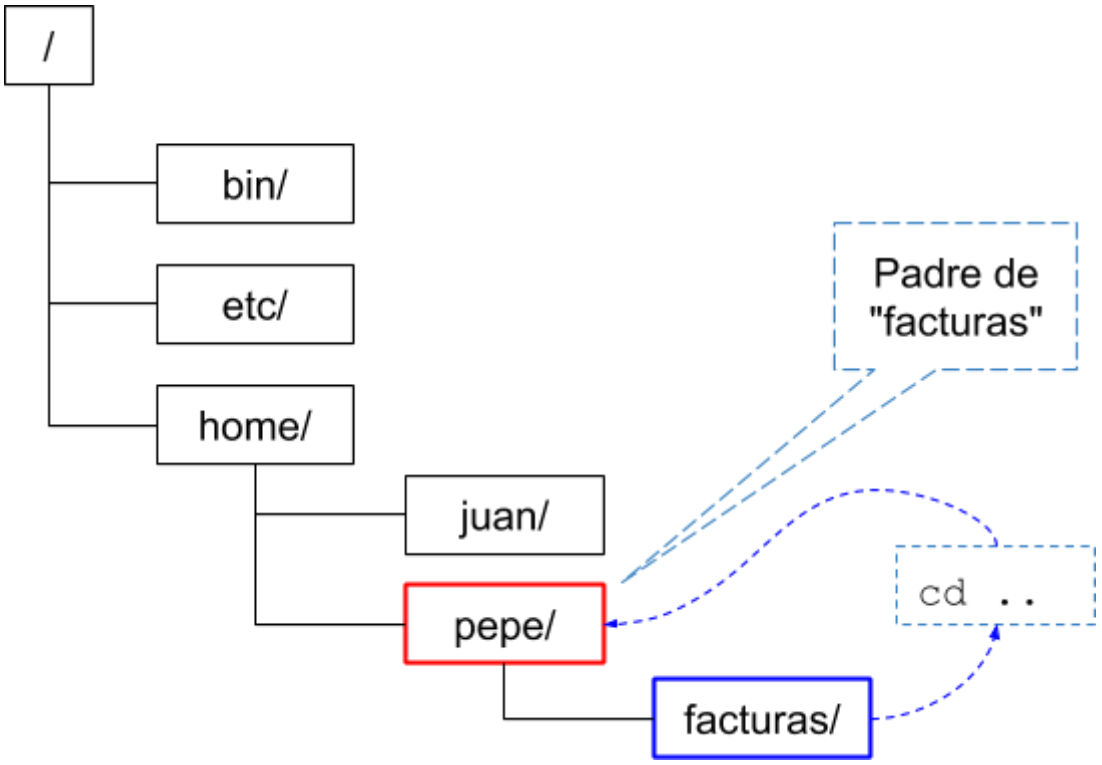
Todo usuario comienza a trabajar en un directorio particular, suele tener allí sus documentos y tiene cierta posesión de dicho directorio. Este directorio se denomina “directorio de inicio” o “directorio home”.

Si se ejecuta el comando “cd” sin argumentos se cambia el directorio actual al de inicio.

Directorio padre

Todo directorio, excepto el raíz, está contenido dentro de otro que denominamos “*directorio padre*”. Para situarnos en el directorio padre del actual, disponemos de un nombre de directorio presente en todos y cada uno de los directorios existentes. Este nombre es “..” (los dos puntos).

Ejecutando `cd . .` Conseguimos regresar al directorio padre



Creación y borrado de directorios

El comando para crear un directorio es "`mkdir`" y para borrarlo "`rmdir`", aunque también se puede usar una modificación de "`rm`". Para borrar un directorio, éste debe estar vacío.

Obtención del directorio actual

Mediante el comando "pwd" se obtiene el directorio actual. Es un comando que se suele ejecutar con mucha frecuencia porque mantiene informado al usuario del directorio en el que está.

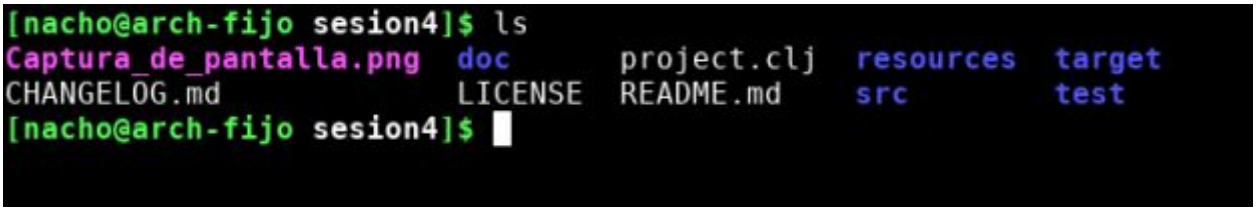
Recuerda:

- Sólo hay un directorio **raíz** para todo el sistema
- Cada usuario tiene un sólo directorio de **inicio** ("home")
- El directorio **actual** y **padre** pueden ir cambiando a lo largo de una sesión.
- Una **ruta relativa** NO empieza por barra "/". Una **ruta absoluta** empieza por la /.

Ficheros

Listar ficheros

La orden `ls` es una comando para mostrar el contenido de un directorio. Este contenido son ficheros cada uno con su nombre. Por tanto ls muestra los nombres de los ficheros de una carpeta.



Las principales opciones usadas para alterar el comportamiento de ls son :

- `-a` : `ls` ignora los nombres de ficheros que empiezan por "." (punto). Esto se hace para proporcionar una forma fácil de tener ficheros y carpetas que al no aparecer en un listado permiten la localización rápida de ficheros importantes. En Windows se llaman "ficheros ocultos" y ocurre lo mismo: no aparecen por defecto al visualizar las carpetas. La opción "-a" (all) cambia este comportamiento y muestra todos los ficheros y carpetas.

- `-l` : Así mismo, el listado mediante ls es por defecto simple, es decir, sólo vemos los nombres. Si deseamos conocer los atributos de los ficheros, debemos pedir un listado "largo". Esto se hace mediante la opción "-l". Más adelante se estudia en profundidad la información obtenida.

- `-F` Si en un momento dado queremos saber qué ficheros en un directorio son ficheros ordinarios y cuáles son directorios, utilizamos la opción –F. Los ficheros cuyo nombre acaba en / son directorios y los que acaban en "*" son ejecutables (código).

- `-t` y `-r` : La ordenación de los ficheros por defecto es en forma alfabética ascendente. Si deseamos efectuar una ordenación por fechas podemos utilizar junto con los modificadores anteriores, el modificador –t (por defecto primero los más nuevos) y si además queremos invertir el orden (primero los más antiguos) añadimos el modificador –r.

- `-R` : Podemos hacer notar que la opción –R (mayúscula) es diferente de la anterior, ya que lista recursivamente un conjunto de directorios, bien a partir del directorio donde nos encontramos (pwd), o bien a partir del directorio que le

pasemos como argumento.

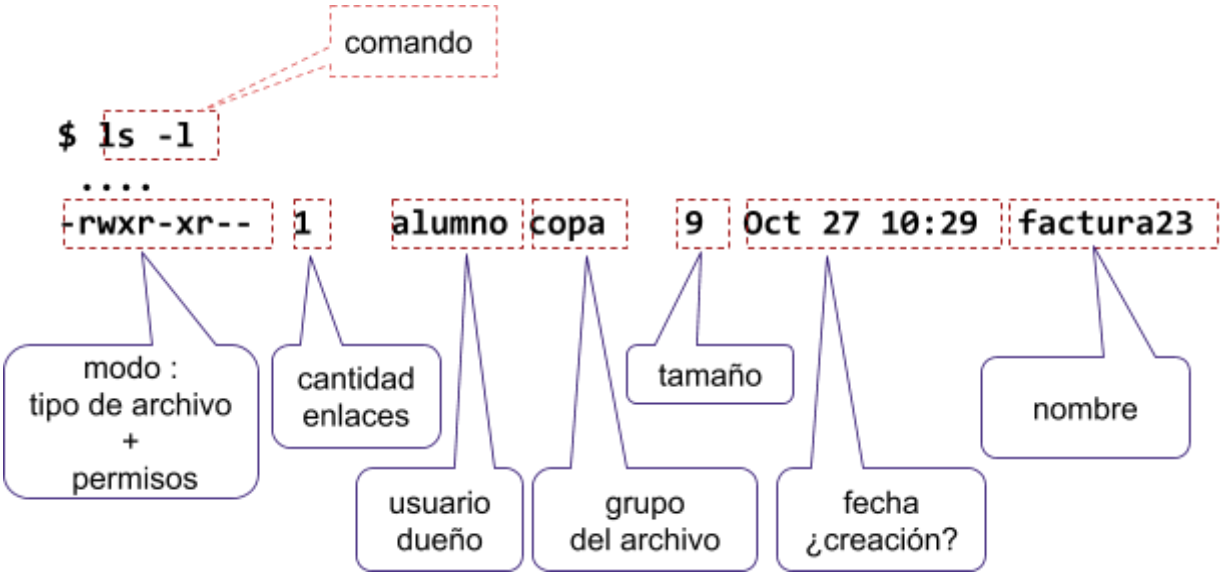
Podemos combinar varias opciones a la vez. Por ejemplo las opciones –a y –l se pueden usar junta (e incluso combinar como `ls -la`).

```
[nacho@arch-fijo sesion4]$ ls -la
total 312
drwxr-xr-x 7 nacho nacho   4096 nov 21 10:51 .
drwxr-xr-x 9 nacho nacho   4096 oct 28 23:44 ..
-rw-r--r-- 1 nacho nacho 249334 oct 21 13:14 Captura_de_pantalla.png
-rw-r--r-- 1 nacho nacho    768 oct 15 13:52 CHANGELOG.md
drwxr-xr-x 2 nacho nacho   4096 oct 15 13:52 doc
-rw-r--r-- 1 nacho nacho     99 oct 15 13:52 .gitignore
-rw-r--r-- 1 nacho nacho    136 oct 15 13:52 .hgignore
-rw-r--r-- 1 nacho nacho   5317 oct 22 16:06 .lein-repl-history
-rw-r--r-- 1 nacho nacho  11219 oct 15 13:52 LICENSE
-rw-r--r-- 1 nacho nacho    361 oct 22 15:46 project.clj
-rw-r--r-- 1 nacho nacho    465 oct 15 13:52 README.md
drwxr-xr-x 2 nacho nacho   4096 oct 15 13:52 resources
drwxr-xr-x 3 nacho nacho   4096 oct 15 13:52 src
drwxr-xr-x 4 nacho nacho   4096 oct 22 15:48 target
drwxr-xr-x 3 nacho nacho   4096 oct 15 13:52 test
[nacho@arch-fijo sesion4]$
```

Con esta combinación de opciones hemos conseguido obtener más información. En este listado nos aparecen los ficheros (uno por cada fila de información, excepto la primera línea), la ocupación en sectores de disco (la primera fila "`total 312`"). Hay dos ficheros especiales que son el "." y "..". El fichero "." hace referencia al directorio actual y el fichero ".." hace referencia al directorio padre.

Interpretación del "listado largo"

El comando ls usado con la opción "-l" muestra una serie de líneas o "entradas de directorio", de tal forma que cada una de ellas hace referencia a un fichero y a varias de sus características o atributos.. Una línea de estas o *entrada de directorio* típica consta de varios campos. El significado de cada uno de ellos es el siguiente:



- **Modos fichero (bits de protección):** El primer elemento (el de más a la izquierda) especifica el tipo de fichero. Los valores posibles son '**d**', si hace referencia a un **directorio**, vacío('-'), si hace referencia a un **fichero ordinario** y '**l**' si hace referencia a un **enlace**. El resto de los elementos son los llamados bits de protección. Están compuestos por tres secuencias contiguas de valores 'r', 'w' y 'x'. El significado de estas secuencias y sus valores lo estudiaremos más adelante.
- **Número de enlaces:**

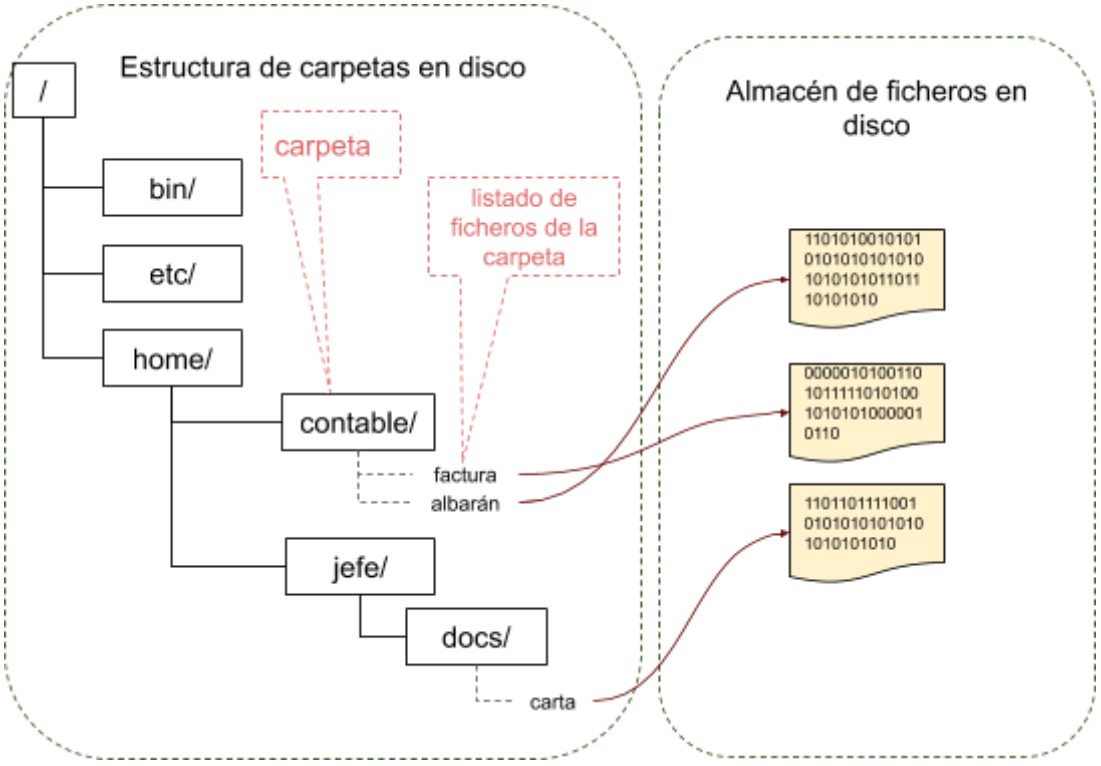
  Para ficheros, indica el número de enlaces físicos que se refieren a ese fichero.

  Para el caso de directorios este número coincide con el número de subdirectorios existentes en ese directorio más dos. Es decir, si el directorio no tiene subdirectorios, su número de enlaces es 2, si tiene un subdirectorio su número sería 3, y así sucesivamente.
- **Nombre propietario:** Indica el nombre del propietario o dueño del fichero.
- **Nombre grupo:** Un conjunto de usuarios puede formar parte de un grupo con una serie de características en común. Este campo hace referencia al grupo al que pertenece el fichero.
- **Tamaño fichero**: Muestra el tamaño de fichero en bytes.
- **Fecha y hora de la última modificación:** Hace referencia a la hora en que el fichero fue modificado por última vez. Si dicha fecha supera el medio año de antigüedad entonces éste también aparece en la fecha. Si queremos saber cuando se accedió por última vez utilizamos la opción –u combinada con –l.

<ejercicio, quitarse permisos y verificar que falla la escritura o lectura para el dueño>

Enlaces

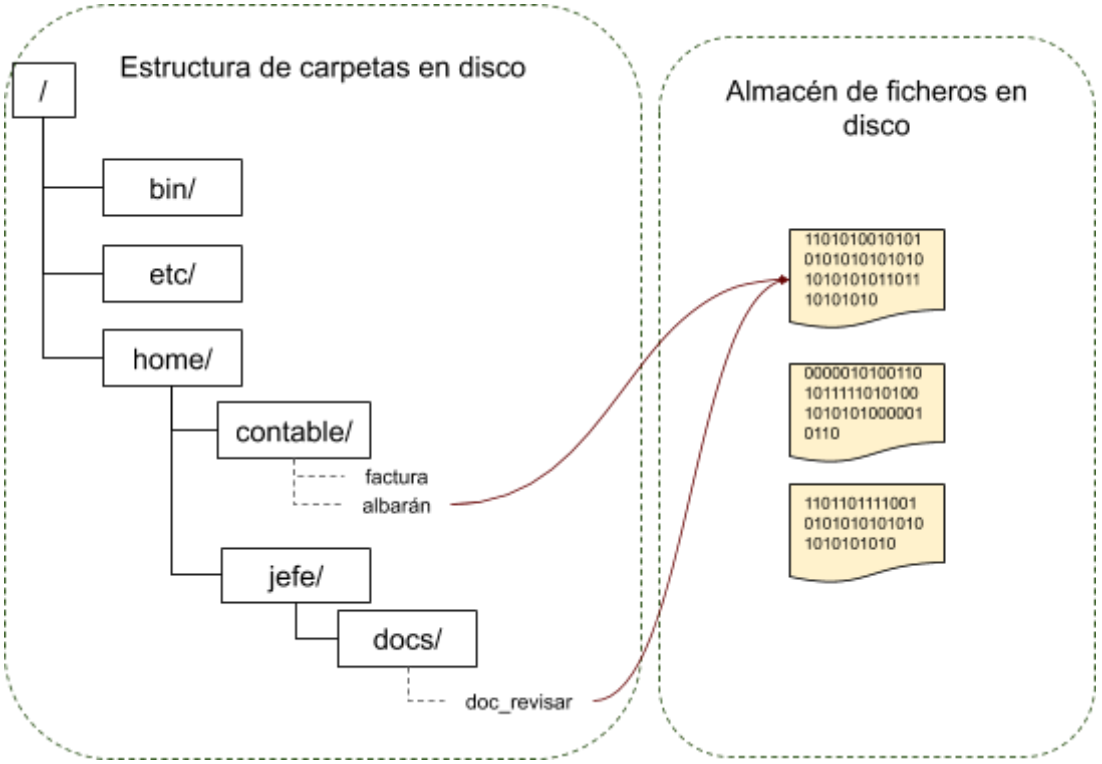
Para entender mejor los enlaces debemos conocer exactamente qué ocurre cuando ejecutamos el comando ls. Realmente no vemos archivos, sino "entradas de directorio". <poner una captura de pantalla de ls >. Cada entrada de directorio es una referencia a un fichero, no el fichero en sí. Podemos pensar que los ficheros existen en el disco sin más, sin estar organizados dentro de carpetas, más bien están todos juntos. Sin embargo, en las carpetas aparecen referencias a esos ficheros. Esto es lo que organiza todos los archivos de un sistema



En el diagrama se muestra una serie de carpetas y en alguna de ellas se indica qué ficheros parece haber en ellas (ficheros "factura" y "albarán" en carpeta "contable"). Pero realmente los ficheros están todos en otro lugar del disco. En las carpetas sólo aparece un enlace para poder acceder a su contenido. La única manera de acceder a un archivo es acceder a la carpeta que contiene el enlace a dicho archivo y usar el propio enlace, por ello nos da la impresión de que los ficheros están real y físicamente dentro de las carpetas.

Por tanto, **Un enlace no es más que una entrada de directorio que apunta a un fichero** o directorio. Pero la novedad en este tema es que puede haber varios enlaces que apunten al mismo archivo.

La consecuencia, es que "un fichero" puede **aparecer** en dos directorios diferentes, y con dos nombres distintos, pero continúa siendo el mismo fichero. Este concepto es muy diferente de lo que los usuarios están acostumbrados y debes abrir tu mente.



El directorio contable contiene una entrada a un archivo que es el mismo que la entrada "doc_revisar" en el directorio jefe/docs tiene. Ambas entradas son el mismo fichero.

Observa la siguiente captura de pantalla tomada de un terminal, que demuestra el concepto de "dos enlaces al mismo fichero" (hecho en un directorio para facilitar la captura)

```
[nacho@arch-fijo ejemplos]$ ls -l
total 0
-rw-r--r-- 2 nacho nacho 0 oct 20 00:07 comentario
-rw-r--r-- 2 nacho nacho 0 oct 20 00:07 texto
[nacho@arch-fijo ejemplos]$ echo "hola" > comentario
[nacho@arch-fijo ejemplos]$ cat texto
hola
[nacho@arch-fijo ejemplos]$
```

En este ejemplo, tenemos dos enlaces al mismo fichero en un directorio (los

enlaces no necesariamente tienen que estar en diferentes directorios). El fichero "comentario" es el mismo que "texto". Antes de escribir nada en alguno de ellos, observa dos cosas:

- Su tamaño es 0, están vacíos, sin contenido.
- Aparece un "2" en el lugar donde se indica el número de enlaces. Cada línea indica que existe otro ~~fichero~~ **entrada** de directorio en alguna otra parte que referencia al mismo fichero
- Después de escribir "hola" en el fichero "comentario", se consulta el otro fichero "texto" y ¡ Allí aparece lo escrito en el fichero "comentario" !... porque son el mismo fichero

Una confusión frecuente al estudiar el concepto de enlace es pensar que hay una *copia* de un fichero en otro lugar porque estamos muy acostumbrados a no trabajar con enlaces. Pero no es así, son dos nombres para el mismo fichero. Una equivalencia sería pensar que una persona tiene dos nombres diferentes y está registrado con DNIs diferentes, No es legal pero es posible. Veámoslo con un ejemplo:

Creación de enlaces

Los enlaces se crean usando un comando especial llamado "`ln`" (ln : LiNk). En este comando hay que indicar:

- La ruta del fichero YA existente
- La ruta donde debe aparecer una NUEVA entrada de directorio

En el ejemplo anterior, suponiendo que el fichero "comentario" ya existe, la nueva entrada "texto" se habría creado:

```
ln  comentario  texto
```

Habitualmente el nuevo enlace se crea en alguna carpeta diferente. Por ejemplo:

```
ln   /etc/passwd   ./configuracion/usuarios
```

Enlaces simbólicos y enlaces duros

Sugerencia: mover esto después de una práctica con el comando cp (en la que al final se modifica un archivo y se comprueba que no se modifican sus copias), de forma que se pueda repetir la misma práctica con el comando ln

Existen dos tipos de enlaces… ¡Mentira! existe sólo un tipo de enlace: el enlace duro o normal, el otro ("enlace simbólico") es una cosa descafeinada. El enlace duro, es el enlace normal, y también es lo que se ha explicado antes. Una vez existe un

fichero y tiene una entrada de directorio en alguna carpeta, puedes crear otra entrada en otra carpeta al mismo fichero. A partir de ese momento son indistinguibles ambas en cuanto a cuál es la principal. Hay dos maneras de acceder al archivo en sí.

Por contra, existe algo llamado "enlace simbólico", que sirve para lo mismo, pero es distinto y en algunas circunstancias importa mucho si es o no enlace simbólico. Un enlace simbólico es **un fichero** que se usa para acceder a algún **otro fichero**.

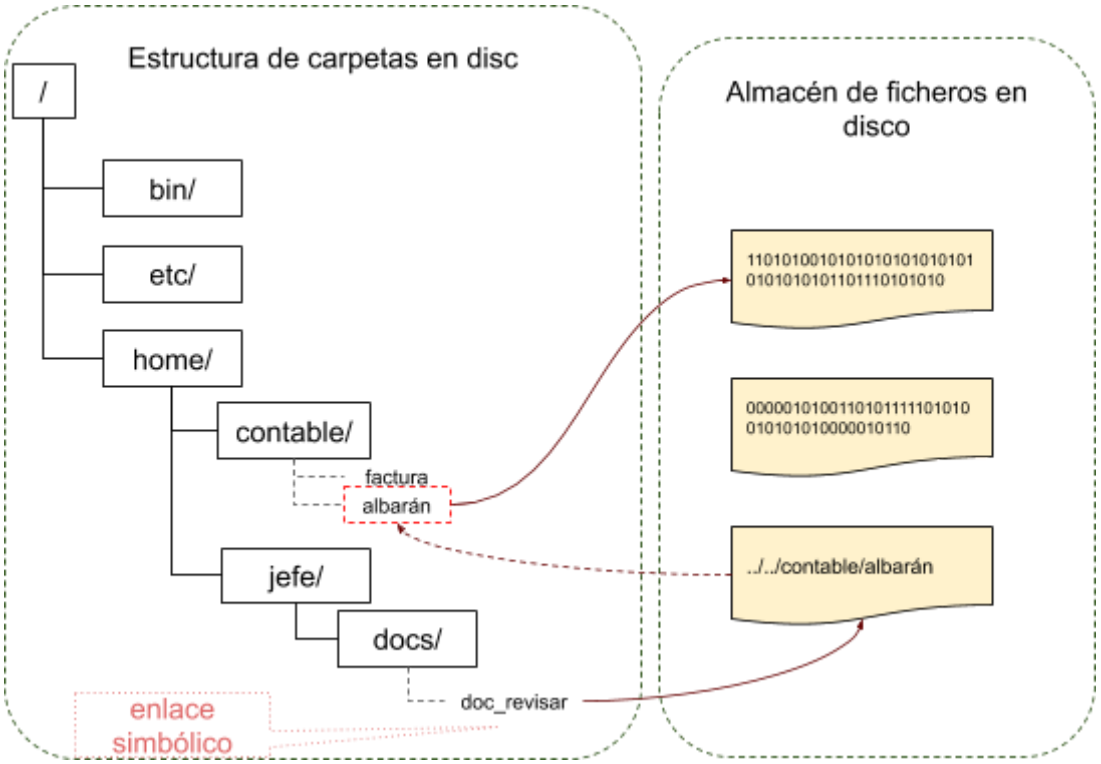
Cuando listas el contenido de una carpeta y existe un enlace simbólico se aprecia una diferencia con el resto de archivos.

```
$ ls -l

 ....

lrwxr-xr-x  1 alumno copa 9 Oct 27 10:29  mensaje -> /etc/motd
```

en el ejemplo anterior "mensaje" es una entrada que apunta a un fichero. Pero es un fichero especial (lo indica la "l" del principio del modo). Se trata de un enlace simbólico . Este enlace es en sí un fichero (o algo muy similar), pero también es realmente un acceso directo o enlace simbólico a otro archivo, el cual aparece indicado en el listado (aparece en rojo "**-> /etc/motd**"). Podríamos pensar que "el contenido del enlace simbólico es la ruta de otro fichero en el disco" y que sólo podemos usar el enlace simbólico para acceder al fichero enlazado.



Prueba

Creemos un enlace simbólico un fichero existente en cualquier UNIX que da el primer mensaje que sale a todos los usuarios, `/etc/motd` al entrar en su terminal. Vea y siga el ejemplo:

```
$ pwd
/home/pepe
$ ln -s /etc/motd mensaje
$ ls -l
 ....
lrwxr-xr-x  1 alumno copa    9 Oct 27 10:29  mensaje -> /etc/motd
```

El fichero `mensaje` es un enlace a `/etc/motd`.

Así, en el ejemplo que acabamos de ver, del fichero `/etc/motd` sólo tenemos un original en el disco, pero ahora puede ser accedido con dos nombres diferentes:

- `/etc/motd`
- `/home/pepe/mensaje` (suponiendo que hemos creado el enlace desde el directorio `/home/pepe/`).

Como ambos nombres se refieren al mismo fichero, cualquier modificación que efectuásemos trabajando con uno de los nombres sería inmediatamente visible usando el otro enlace (el otro nombre).

Se puede hacer un listado que indique el número de ***i-nodo*** de un fichero de manera que podríamos constatar que ambos son en el fondo el mismo fichero:

```
ls –i –l /etc/motd /home/pepe/mensaje
```

A partir de ahora, hemos de ser conscientes de que el listado de un fichero sólo muestra enlaces a los ficheros, aunque hablaremos de manera informal confundiendo enlaces con ficheros.

Ejercicio sobre enlaces "duros" o normales.

1. Crear un fichero con algo de contenido.
   `echo “contenido” >> fichero`

2. Situarse en otro directorio. Y crear allí un enlace duro al fichero creado antes
   `mkdir otrodirectorio; cd otrodirectorio;`
   `ln <Ruta al fichero primero> enlaceCreado`

3. Modificar uno de los enlaces (duros) y verificar que la modificación es visible en el otro. Hacerlo en los dos sentidos.
   `echo “nueva línea” >> enlaceCreado;`
   `cat <ruta al fichero primero>`

4. Crear un enlace simbólico a uno de los dos.
   `ln –s ‹ruta  fichero existente› enlaceSimbólico`

5. Repetir la modificación y revisión usando el nuevo enlace simbólico.

6. Eliminar el fichero que se usó para realizar el enlace simbólico y tratar de leer y escribir en él, verificando los resultados. Explicar los resultados

Copia, borrado, movimiento y renombrado de ficheros (cp , mv)

Orden cp

Si queremos copiar un fichero utilizamos el mandato cp. En su uso normal, el primer argumento del comando es el fichero origen y el segundo el destino. En el momento de la copia, se replica el contenido pero a partir de entonces cada fichero es independiente. Ejemplo:

```
$ cp /etc/profile miprofile
```

Esto copia el fichero `/etc/profile` a nuestro directorio y con el nombre `miprofile`. Esto es equivalente a utilizar el mandato:

```
$ cp /etc/profile ./miprofile
```

Recordemos que el punto "." es nuestro directorio actual.

También podemos efectuar la copia a un directorio concreto sin escribir el nombre destino, que en dicho caso se replica también

```
$ cp /etc/profile prueba3
```

crea una copia del fichero `profile` en el directorio `prueba3`. El fichero copiado también se llama "profile".

La orden cp también copia directorios. Lógicamente querremos copiar tanto un directorio como su contenido. Para esto utilizamos el modificador recursivo –R.

```
$ cp –R prueba3 prueba4
```

Orden mv

El cometido de la orden mv es mover ficheros entre diferentes directorios. Si se usa sobre el mismo directorio el efecto obtenido consiste en cambiar el nombre al

fichero.

Ejemplos:

```
$ mv miprofile nuevo_profile
```

Cambia el nombre del fichero `miprofile` a `nuevo_profile`. Mientras ...

```
$ mv nuevo_profile prueba4
```

coloca el fichero `nuevo_profile` en el directorio `prueba4`.

Para comprobarlo utilizar el mandato `ls prueba4` que devolverá el contenido del directorio `prueba4`. La orden `mv` ha cambiado el fichero de sitio (ha movido el fichero). Si ejecutamos la orden `ls` directamente podremos observar que el fichero `nuevo_profile` ha desaparecido del directorio en el que se encontraba.

Si el fichero destino al que copiamos o movemos ya existe y no tiene permisos de escritura entonces el sistema nos pide confirmación. Los permisos del fichero copiado o movido son los mismos que los del fichero original. Estudiaremos los permisos más adelante en esta práctica.

Para más información sobre ambos mandatos consultar el manual.

Borrado de ficheros (rm)

La orden `rm` suprime un fichero de un directorio. Si queremos borrar el fichero que habíamos creado anteriormente...

```
$ rm profile
```

Realmente el comando 'rm' elimina el enlace. Sólo si el enlace eliminado era el único enlace al fichero, se elimina efectivamente el fichero. Es decir: Si un fichero es referenciado desde dos enlaces y se elimina uno (con 'rm'), el otro persiste y el fichero puede ser usado. Si, por el contrario, éste era el último enlace al fichero, entonces el fichero sí desaparece.

Incluir casos de comandos erróneos con su explicación

Visualización del contenido de los ficheros. Comandos cat, more, less, tail, head.

La orden `cat` se utiliza (entre otras cosas) para visualizar el contenido de un fichero.

Ejemplo: Visualicemos el fichero de configuración del *shell* (intérprete de órdenes de UNIX) `/etc/profile`

```
$ cat /etc/profile
```

Podemos listar por ejemplo la lista de usuarios del sistema, que suelen encontrarse en el fichero `/etc/passwd`

```
$ cat /etc/passwd
```

```
...
```

Si el fichero no cabe en pantalla, como en este caso, podemos utilizar las órdenes <Crtl>-S (para detener la salida) y <Crtl>-Q (para reanudarla)

La orden `cat` permite listar varios ficheros secuencialmente. Si tenemos dos ficheros llamados `fichero1` y `fichero2`, la orden:

```
$ cat fichero1 fichero2
```

lista en primer lugar el fichero `fichero1` y a continuación `fichero2`.

Una orden alternativa a `cat` es la orden `more` que da más control que la anterior, ya que automáticamente lista un fichero y cuando llena la terminal) se para, esperando que pulsemos la tecla espacio para reanudar la salida. Esto permite ir "pasando página" cuando vemos un fichero extenso.

```
$ more /etc/termcap
```

Nos dice además el porcentaje de fichero que ya ha sido listado.

La orden `more` tiene varias opciones interesantes:

- Con el modificador –n, lista el fichero presentando de n en n líneas y no con el número de líneas que posee nuestra pantalla.
- Con el modificador +n, lista el fichero a partir de la línea n.

Sin embargo, la orden **`more`** no permite retroceder en la pantalla a la hora de visualizar el contenido de un fichero. Si se ha avanzado más allá de lo que se desea ver, debemos detener (Control+C) y volver a ejecutar el comando para revisar líneas anteriores.

La orden "**`less`**" sí que permite retroceder (generalmente con las teclas de dirección) y visualizar el contenido de un fichero de manera no lineal y hacia atrás.

La orden `tail` permite visualizar el final de un fichero. Por defecto visualiza las 10

últimas líneas. Así por ejemplo:

```
$ tail /etc/profile
```

lista las últimas 10 líneas de nuestro fichero `/etc/profile`.

Si queremos listar por ejemplo las últimas 5 líneas

```
$ tail –5 /etc/profile
```

El final de un fichero puede ser establecido también a partir de una línea en concreto. Es decir “desde la línea N hasta el final” (con head esta es la única opcion). Por ejemplo, mostrar desde la línea 2 hasta el final :

```
$ tail –n +2 /etc/profile
```

Al igual que en el caso anterior se anima al alumno que consulte las páginas del manual y practique por su cuenta.

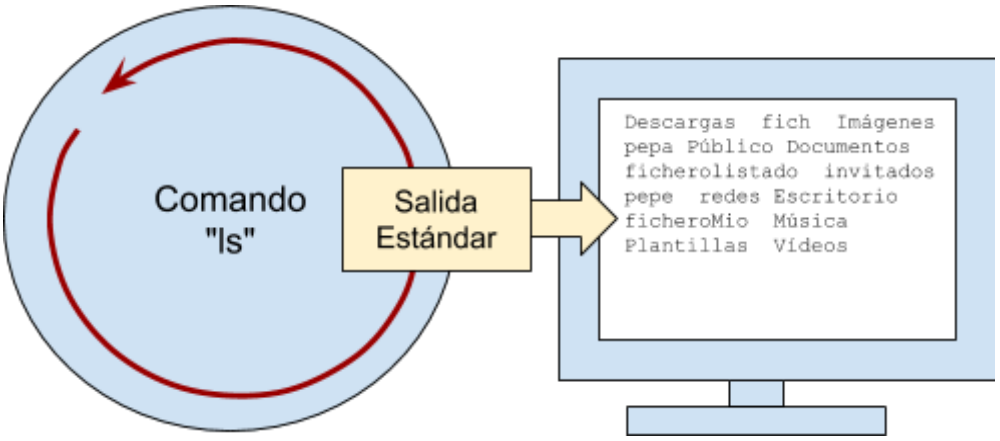
Entrada y salida

Salida estándar

Todos los comandos (en general programas) pueden emitir información de salida propia de los cómputos o acciones realizadas con éxito. Este canal de salida se llama "Salida Estándar"

La siguiente imagen refleja ésta situación. El círculo representa la ejecución de un comando (como ejemplo: '`ls –l`'). Dicho comando genera un listado de nombres de archivos. Este listado debe emitirse y mostrarse en algún lugar. El canal por el que los comandos emiten información se llama "salida estándar" o "standard output" ("standard" porque es el canal por defecto y habitual). La mayoría de comandos emite información por la salida estándar. Cada comando emite la información correspondiente a la función que tiene.

La información emitida, acaba mostrándose habitualmente en la pantalla del ordenador; y más concretamente, en el terminal donde lanzamos el comando. Por defecto, la salida estándar de los comandos está conectada al terminal.



En este apartado vamos a ver cómo podemos enviar los datos de la salida estándar a otro lugar o destino

Salida de error

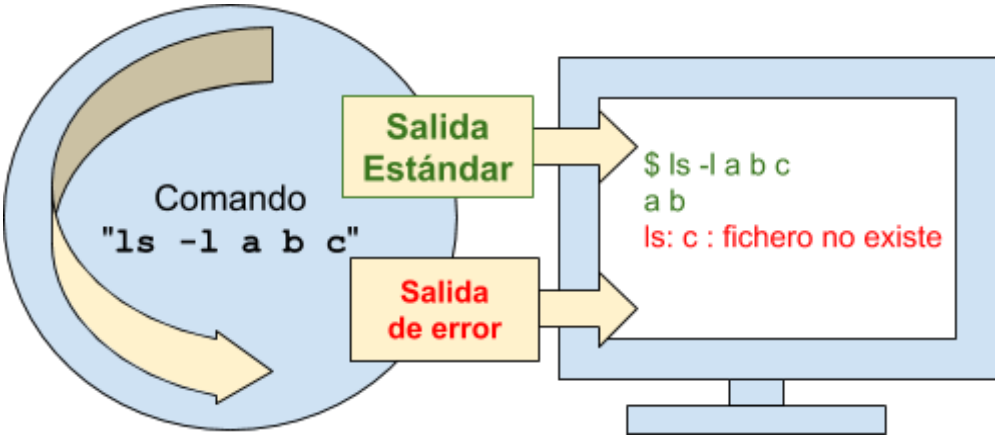
En ocasiones Los comandos encuentran problemas, fallos o errores, durante su ejecución. Éstos también deben ser comunicados, pero de una forma diferente. ¿Por qué "diferente"? Porque un error se supone que no debe ocurrir y si ocurre, la información del error no debe ser confundida con la información que el comando debería sacar como resultado de su ejecución. Observa el siguiente ejemplo de ejecución del comando ls

```
$ls pipo

ls: el fichero no existe
```

¿Como sabes que ha habido un error o que el comando ha encontrado los ficheros "ls:", "el fichero", "no" y "existe" y los ha listado?

Existe otro canal de comunicación alternativo llamado "salida de error" que los comandos usan para informar de errores o situaciones anómalas.



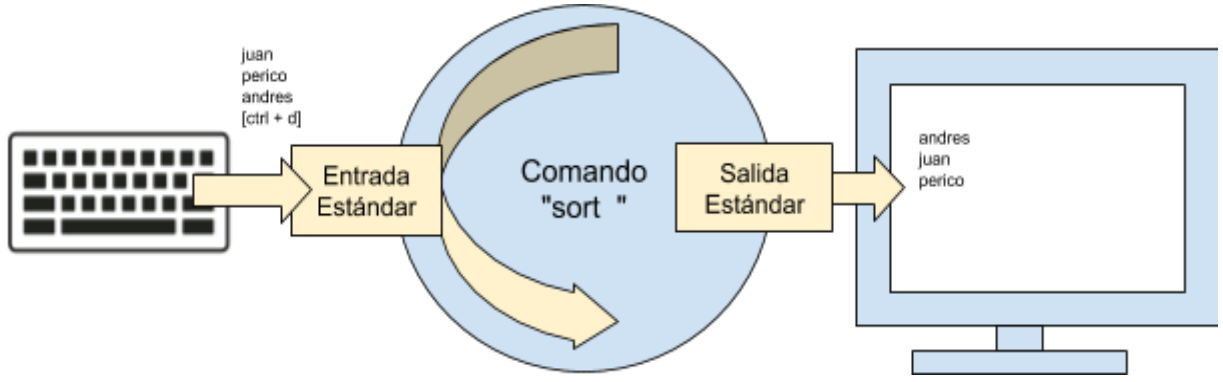
Normalmente, tanto la salida de error como la estándar están conectadas al terminal o pantalla, y por eso vemos todos los mensajes de la misma manera: por la pantalla.

Entrada Estándar

Algunos comandos toman datos a los que les aplican algún proceso. "ls", por ejemplo, no es uno de ellos ya que simplemente, lista el contenido de un directorio. "echo" tampoco, porque el dato que muestra por pantalla lo tiene en el propio argumento. Sin embargo, "sort" es un buen ejemplo de comando que toma datos y aplica un proceso.

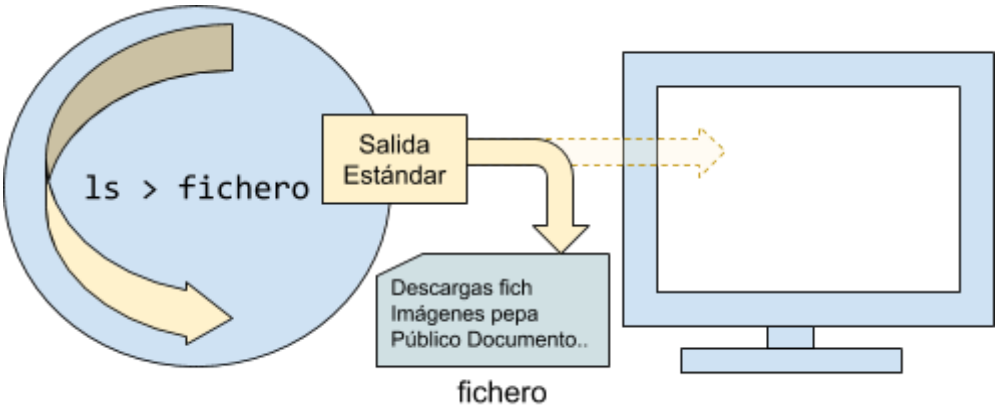
Escriba sort en el terminal y ejecútelo. Introduzca algunas líneas con nombres o palabras sueltas y después de introducir unas cuatro o cinco pulse [ctrl + d] (para finalizar la introducción). Observará que las palabras introducidas se han vuelto a escribir pero en orden alfabético. En este ejemplo, el comando sort ha leído las palabras del teclado. Esos eran los datos con los que trabajaba el comando. Y el proceso ha sido ordenar esos datos.

Al igual que sort antes, los comandos pueden leer datos de un canal que se llama "entrada estándar". Este es un canal que lleva datos hasta el comando, no como antes, donde el canal emitía datos.



Redirección.

La entrada y salida estándar, y la salida de error de un proceso pueden ser redirigidas. Esto es, en el caso de las salidas no se muestran por pantalla, sino que se envían a un fichero o a otro comando. O bien, un comando que generalmente lee del teclado, toma sus datos de otro lugar



Redirección de salida

Para redirigir la salida estándar a un fichero se añade el símbolo “>” al final del comando, seguido por el nombre del fichero donde se desea depositar los resultados. Esto provoca que la pantalla ya no muestre los resultados de la ejecución de dicho comando y que lo que saldría por pantalla, se escriba en el fichero especificado.

Pruebe el siguiente ejemplo:

```
ls –l > fichero_listado
```

Este comando creará el fichero llamado "fichero_listado" a continuación dejará allí el resultado de su ejecución, esto es: el listado *largo* de los ficheros del directorio. Compruébelo ejecutando `cat fichero_listado`.

Para redirigir la salida de error se emplean los símbolos “2>” seguidos del nombre del fichero.

```
rm fichero_inexistente 2> fichero_errores
```

el mensaje de error no aparecerá por la pantalla, se depositará en el fichero “fichero_errores”. Ejecútelo y compruébelo

En ambos casos, si en vez de utilizar el símbolo “>”, se utiliza el símbolo “>>” , los datos se añaden al final del fichero (si existe) y no se elimina el contenido previo del fichero. Pruebe estas órdenes como ejemplo:

```
echo "linea1" > fichero_lineas
echo "linea2" >>fichero_lineas
```

Compruebe el contenido de fichero_lineas. ¿Qué hubiese ocurrido si la segunda línea hubiese tenido un solo símbolo “>”?

Recuerda una regla importante:

cuando en una línea de comando aparece el símbolo ">" seguida de un

fichero, el fichero se creará y borrará al inicio de la ejecución de la línea, incluso aunque exista un error en el comando y éste no se ejecute bien, incluso aunque no exista el comando que se pretende ejecutar.

Suponga un fichero cuyo contenido son nombres de persona. Llamémosle a este fichero "invitados" y sea éste su contenido:

```
juan
perico
andres
maria
```

Si deseamos reordenar el fichero, podríamos intentar realizar el siguiente comando:

```
$ sort invitados > invitados
```

Si se prueba el comando, no se verá nada por pantalla. Es normal, se ha redirigido la salida a un fichero y por la pantalla no sale nada. Pero, después ¡En el fichero "invitados" tampoco verá nada! ¿Por qué?

Porque lo primero que se hace en esa línea de comando es borrar el contenido del fichero "invitados" debido a la redirección existente. Después de borrarlo, el comando sort no tiene datos que leer y ordenar, y termina sin haber hecho ningún trabajo.

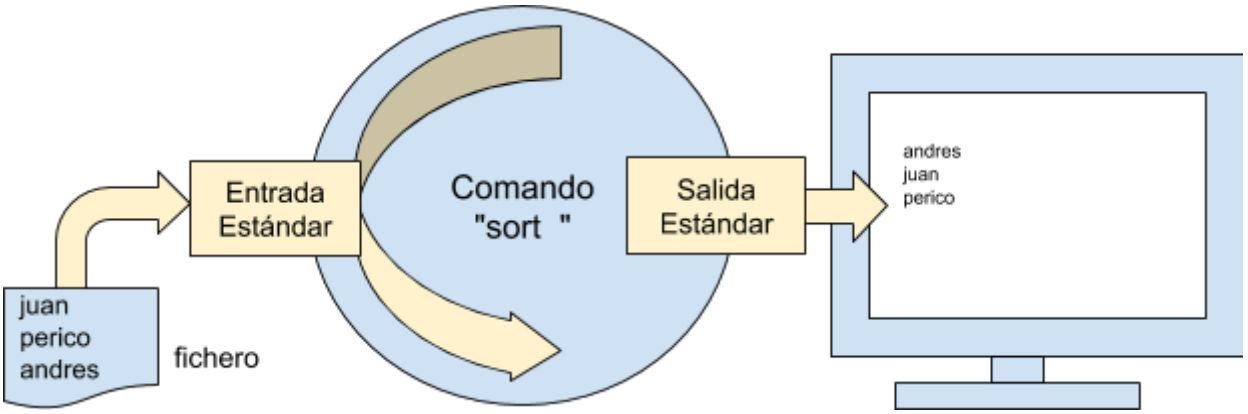
Redirección de entrada.

En los comandos que toman sus datos de la entrada estándar también puede aplicarse una redirección similar a la de la salida. Es posible tomar datos de un fichero y utilizarlos como entrada de un comando directamente. La mayoría de veces esto es innecesario, pues muchos comandos pueden leer los datos de los ficheros si se les pasa su nombre por argumentos. Sin embargo, existen comandos no preparados para ello y en ciertas ocasiones es necesario recurrir a ello.

La redirección se consigue con el símbolo '<' (justo el contrario de la redirección de salida '>'). Suponga que “fichero_nombres” contiene un nombre de persona por cada línea:

```
sort  < fichero_nombres
```

“sort” lee líneas de su entrada estándar, pero ésta está conectada con el fichero "fichero_nombres" .



Por tanto este comando mostrará los nombres contenidos en fichero, ordenados. Sin la redirección de entrada, usted podría haber lanzado el comando sort y a continuación escribir por el teclado los nombres de las personas. Al pulsar ctrl+d habría terminado de introducir datos y entonces el comando se habría ejecutado, mostrando por la salida estándar el resultado.

La redirección de entrada o salida no la realiza el comando, sino el intérprete de comandos o terminal, que prepara la ejecución convenientemente. El comando no es generalmente consciente de a dónde está conectada la entrada o salida estándar

Redirección de entrada tipo “here documents” (no en SMX, no en DAM, solo ASIX)

Un caso particular de redirección de entrada es el que los datos de entrada están en la propia línea de argumentos. No se confunda, una cosa son las opciones de un comando -que se escriben en la línea de comandos y por tanto son argumentos- pero no constituyen los datos de un comando, y otra cosa son los datos que un comando procesa (como sort, que ordena una serie de líneas). Y recordemos: hay comandos (como 'ls') que no toman datos de entrada.

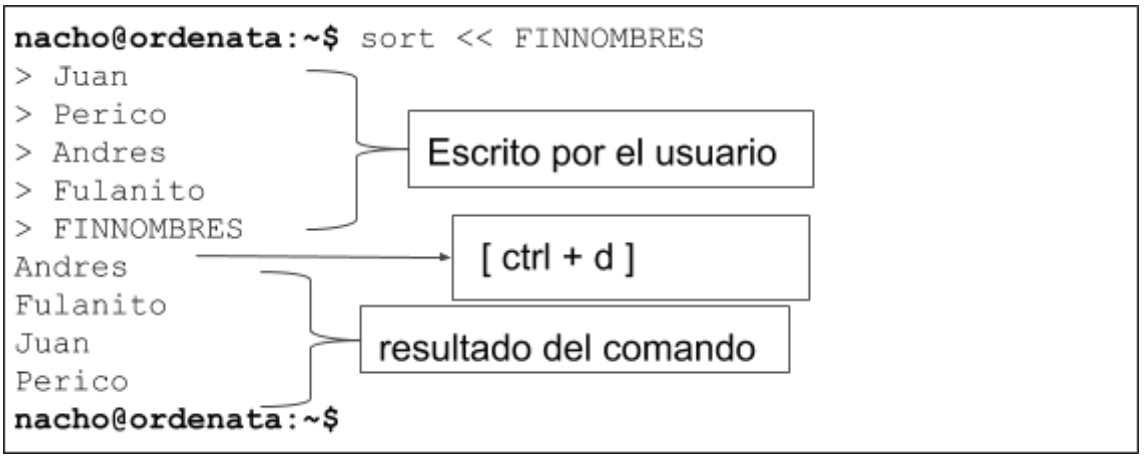
Supongamos el comando sort anterior si lanzamos la orden.

```
sort  << FINNOMBRES
```

Observaremos que el prompt no aparece, luego el sistema está esperando que introduzca algo que no es un comando. En este caso está esperando que termine la línea de comandos (el “<<” ha producido que no dé por finalizado el comando. A continuación podrá escribir varias líneas hasta que escriba una con el texto “FINNOMBRES” (lo que ha escrito a continuación del “<<”). Esto es una marca o indicación (que puede ser cualquier palabra) que señala el final de las líneas del comando: hasta que el usuario no escriba la marca, todo lo que se teclee se considerará

como línea de comandos.

Cuando teclee la marca, el comando tomará los datos introducidos, los procesará y mostrará su resultado por la salida estándar



Las líneas con > son parte de la línea de comandos y han sido escritas por el usuario (el ">" lo pone el propio shell). Las 4 líneas siguientes con los nombres son el resultado del comando. La última línea, que muestra el prompt indica que el comando ya ha terminado.

Redirección de la salida de error

Los errores ocurridos durante la ejecución de un comando, generan una información que no debe ser confundida con la información procesada que se espera de un comando como salida estándar. Los mensajes de error constituyen una salida de datos adicional a la estándar. Al igual que ésta, normalmente la salida de error está conectada al terminal, de forma que vemos del mismo modo mensajes de error y de datos normales en un terminal.

Es posible, sin embargo, redireccionar esta salida de error de forma independiente y procesarla igual que la salida estándar. Para ello se utiliza el símbolo " 2> ". Observa el ejemplo:

```
$ rm fichero_no_existente
bash: rm: "fichero_no_existente" no existe
$ rm fichero_no_existente 2> fichero_errores
$ cat fichero_errores_
bash: rm: "fichero_no_existente" no existe
```

Para redirigir tanto la salida estándar como la salida de error al mismo fichero se

utiliza el símbolo "&>". De esta foma `cat "facturas.txt" &> fich_salida` depositaría su resultado en el fichero "fich_salida", tanto si es el fichero como algún mensaje de error.

Redirección de la salida estándar a la salida de error

En ocasiones desearemos emitir un mensaje por la salida de error. Si utilizamos echo, el mensaje sale por la salida estándar, pero si deseamos que salga por la salida de error, debemos utilizar :

```
$ echo "mensaje de error " >&2
mensaje de error
```

La utilidad se verá cuando hagamos nuestros propios programas y en ellos deseemos emitir un mensaje de error por la salida de error.

Concatenación de comandos

Otra posibilidad en la redirección consiste en utilizar los datos de salida de la ejecución de un comando como datos de entrada para otro comando.

Un ejemplo servirá para explicarlo. Supongamos el siguiente problema: Mostrar las líneas tercera y cuarta de un fichero de seis líneas:

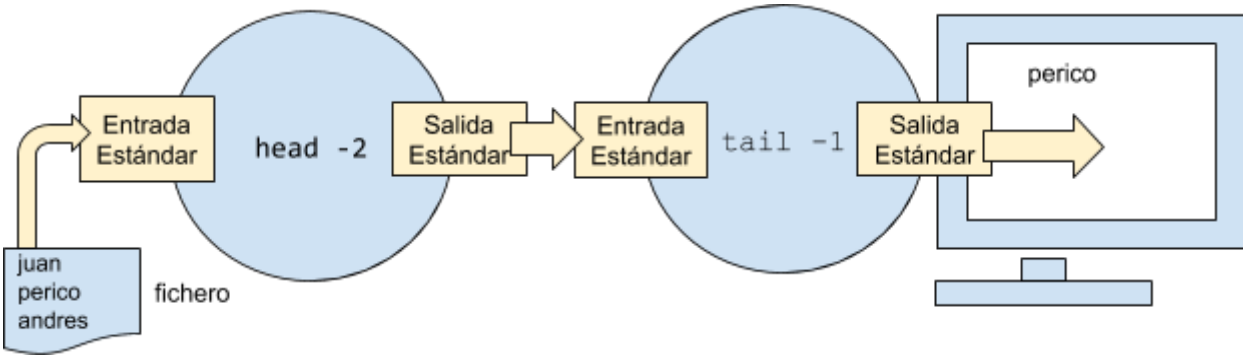
La solución se consigue utilizando el comando head para listar las cuatro primeras líneas y, a continuación, el comando tail para quedarse con las dos últimas. Queda por determinar cómo se pasa ese listado de cuatro líneas que obtiene el primer comando al segundo comando (tail):

Una solución es usar un fichero temporal:

```
head –4 fichero › fichero_temporal

tail –2 fichero_temporal
```

La solución idónea es otra: pasar los datos **directamente de un comando al otr**o.



Ese lugar virtual por donde pasan los datos se denomina tubería. Así, se entiende que existe un flujo de datos entre un comando y otro. Para concatenar dos comandos sescribe el primero, seguido de un símbolo "|" (Alt gr + 1), y seguido del segundo

comando (el que recibe los datos y se ejecuta después)

```
head -4 fichero | tail -2
```

Redirección con algunos comandos:

La redirección se vuelve interesante cuando se combinan diversos comandos. En este apartado se va a realizar una presentación breve, pero estos comandos con sus varias opciones pueden ser necesarios en ejercicios muy complejos.

cat

El comando cat explicado arriba tiene otros usos, incluso más primitivos. Cat toma los datos de la entrada estándar (es decir, del teclado, excepto sí se suministra un fichero o existe redirección). "cat" muestra lo leído por la salida estándar (pero se puede redirigir) Pruebe a ejecutar cat sin más y observará el resultado: se muestra por pantalla lo que se escribe por teclado (hasta pulsar ctrl+d) sin otro efecto.

¿Qué ocurre si redireccionamos la salida del comando a un fichero? Pruébelo. Verá como esta es una manera directa de crear un fichero de varias líneas

Cat tiene varios usos:

- Mostrar un fichero por la pantalla (aquí no hay redirección, pero cat cambia el origen de los datos):

  `cat fichero`

- Escribir un texto y guardarlo en un fichero

  `cat > ficheroEscrito`

- Concatenar varios ficheros y mostrarlos por pantalla o escribirlos en un fichero de resultado:

  `cat ficherofuente1 ficherofuente2 … > ficheroTodoJunto`

echo

echo puede utilizarse para dar contenido a un fichero, igual que el caso anterior de cat pero el contenido del fichero se proporciona como argumentos al comando.

tail y head

Como se ha visto con el ejemplo, permiten recortar trozos finales o iniciales de fichero hasta obtener las líneas deseadas.

sort

Sort ordena líneas introducidas desde la entrada estándar. Si queremos ordenar el contenido de un fichero, sort actuará como el comando que ordena. Cree un fichero con cinco líneas y ejecute “`cat fichero |sort`”

ls

El comando para listar el contenido de directorios puede ser una fuente de datos para otros comandos. Pruebe "`ls | sort |more`"

cut

Este comando extrae un subconjunto de palabras o caracteres de una línea. Es capaz de "filtrar" una línea o varias quedándose, por ejemplo, con "la quinta palabra", "los caracteres del quinto al décimo", etc.

Con ello conseguimos eliminar información no interesante proporcionada por otro comando, de manera que obtenemos la que buscamos. Pruebe "`ls –l | cut –b 37–50`"

grep

grep filtra de un conjunto de entrada compuesto por varias líneas, aquellas que cumplen un patrón. La salida incluye la línea

Tome un fichero con varias líneas y elija una palabra o patrón que aparezca en algunas (pero no todas las líneas). Pruebe "`grep palabra fichero`".

Grep es especialmente útil cuando filtra el resultado de algún otro comando o transfiere el resultado filtrado a otro comando. Por ejemplo para listar de un directorio los nombres de ficheros que sean directorios: "`ls | grep / `". Esto lista todos los ficheros pero muestra aquello que tienen en su nombre una barra "/"

tr

tr cambia y borra caracteres tomados de la entrada estándar y los muestra por la salida estándar. Para especificar los caracteres sobre los cuáles operar, dispone de un abanico de notaciones que expresan caracteres, dígitos, símbolos de puntuación, etc. Por ejemplo, se puede convertir a mayúsculas una palabra en minúsculas con

tr [:lower:] [:upper:]

tr también puede reemplazar una secuencia de caracteres repetidos por uno sólo

uniq

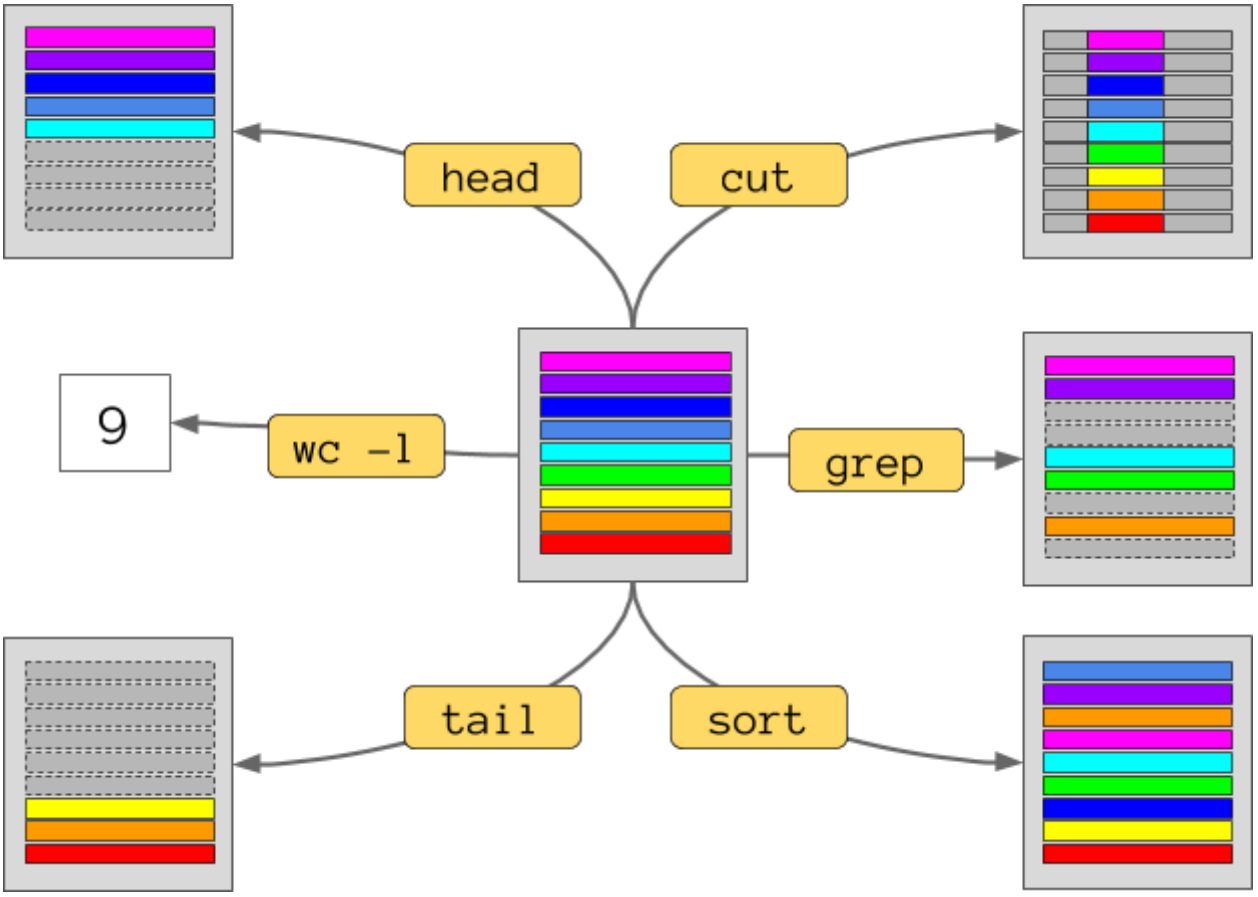
Elimina, muestra o simplifica líneas duplicadas consecutivas de un archivo. Básicamente elimina repeticiones, haciendo que no haya dos líneas iguales en el archivo que estén seguidas.

paste

paste une líneas de distintos ficheros (la primera con la primera; la segunda con la segunda, etc.)

Forzando la redirección desde el teclado.

Cuando un comando como cat lleva un argumento de tipo fichero, cambia su forma de actuar porque toma los datos desde dicho fichero en vez de la entrada estándar, que es lo primitivo. Supón que deseas usar cat para concatenar ficheros, pero además, quieres en el mismo comando concatenar datos escritos desde el teclado.



4 Reglas para nombrar ficheros

Los nombres de los ficheros están formados por caracteres. Se distingue entre mayúsculas y minúsculas. Su número varía entre los diferentes sistemas UNIX (en algunos hasta 14, y en otros hasta 255). Los caracteres válidos pueden ser cualesquiera en teoría. En la práctica hay algunos que debemos evitar:

```
';', '<','>', '$', '|', '*', '?', ' '  (espacio)
```

ya que estos caracteres tienen un significado especial dentro de los mandatos UNIX.

Como regla general se trata de utilizar los caracteres alfabéticos, numéricos, el guión inferior (_) y el (.). Este último no se ha de utilizar como primer carácter del nombre de un fichero a no ser que queramos ocultarlo. UNIX oculta (a nivel de mandato `ls`) los nombres de fichero que comienzan con . excepto si utilRenameizamos la orden `ls –a`.

Ejemplo:

Correctos	Incorrectos

`practica.c`
`mi_practica`
`practica3`
	`practica*`
	
	`>practica`
	
	`prac|tica`

Selección múltiple de ficheros. Patrones de nombres

A veces es interesante referenciar ficheros que tengan en su nombre características comunes. “Todos los ficheros cuyo nombre empiece por la letra c...”. En UNIX esto se consigue utilizando caracteres especiales (llamados metacaracteres o comodines) que representan otras cosas:

- El carácter asterisco '*' representa a cualquier cadena de caracteres arbitraria incluyendo la cadena vacía.
- La interrogación '?' representa a cualquier carácter simple.
- Los corchetes '[' ']' pueden contener un grupo o rango de caracteres y corresponden a un carácter simple. Mediante el símbolo “menos” '-' se delimita un rango, si no se especifica un rango, se entiende un conjunto de caracteres de entre los que elegir uno (pueden estar separados por comas) (ojo. la variable de entorno LC_COLLATE gestiona la diferenciación de mayúsculas minúsculas en rangos de letras)

Los corchetes ([ ]) pueden englobar varios rangos dentro de ellos. Por ejemplo, para seleccionar un caracter alfabético que incluya mayúsculas o minúsculas usaríamos el patrón

[a-zA-Z]

El patrón tiene dos partes, la variable y la invariante:

`mv *.png`

Ejemplos: Vamos a utilizar la orden `ls`, aunque en principio los comodines se pueden aplicar a cualquier orden. Por ejemplo, pruébense los siguientes mandatos sobre los ficheros de prueba propuestos en la práctica:

```
$ ls
$ ls a*
$ ls fichero?
```

```
$ ls c[1-3]
$ ls c[1,3]
$ ls c[13]
$ ls *?[a,e,i,o,u]*
$ ls [1357][24860]
$ ls *2
```

Todas estas opciones pueden ser combinadas entre sí. El alumno debe probar diferentes combinaciones y observar los resultados.

*El Metacaracter **

El shell, no los mandatos, interpreta este carácter antes de ejecutar un mandato. Lo sustituye por los nombres de los ficheros existentes en el directorio actual, y estos nombres son pasados como argumentos.

Para comprobarlo se puede utilizar el mandato echo. Este mandato envía a la salida estándar sus argumentos. Por ejemplo, pruébense, las siguientes líneas de mandato.

```
$ echo *
$ ls *
```

Evitando la interpretación de los metacaracteres

El shell no interpreta los caracteres encerrados entre comillas (') o antecedidos por la barra invertida (\), lo que permite escribir un mandato en varias líneas:

```
$ echo '*'
$echo "Els arxius " * "estan en el directori actual."
$echo "Els arxius \"\*\" son " * "i estan en el directori actual."
$ echo \
aquí \
hay \
cuatro \
argumentos
$echo 'Se puede usar el intro
> dentro de comillas'
```

Patrones para creación de múltiples cadenas

En ocasiones deseamos crear varios cadenas que se forman a partir de una parte fija o invariable unida a otra parte que es variable y procede de una lista de elementos. Por ejemplo las cadenas:

```
facturajuan
facturaperico
```

```
facturaandres
facturafulano
```

Están formadas por la parte “factura” unida a la parte “juan”, “perico”, etc.

Es posible en un sólo comando y de forma compacta generar varios elementos que sigan este patrón. Para ello basta con concatenar la parte fija a una lista de partes variables encerradas entre los símbolos”{“ y “}”. Pruebe el siguiente ejemplo:

```
nacho@Toshibatil:~$ echo factura{juan,perico,andres}
facturajuan facturaperico facturaandres
nacho@Toshibatil:~$
```

Este ejemplo se puede complicar bastante si añadimos otra lista

```
nacho@Toshibatil:~$ echo factura{juan,perico,andres}{2008,2009,2010}
facturajuan2008      facturajuan2009      facturajuan2010      facturaperico2008
facturaperico2009    facturaperico2010    facturaandres2008    facturaandres2009
facturaandres2010
nacho@Toshibatil:~$
```

Manipulación de patrones con grep

El comando grep (Globally Regular Expressions Pattern) busca patrones en ficheros. Por defecto devuelve todas las líneas que contienen un patrón (cadena de texto) determinado en uno o varios ficheros. Utilizando las opciones se puede variar mucho este comportamiento. Si no se le pasa ningún fichero como argumento hace la búsqueda en su entrada estándar.

Sintaxis

```
grep [opciones] <patrón> [ficheros] opciones:
```

Opciones principales

- -c: Devuelve sólo la cantidad de líneas que contienen al patrón.
- -i : Ignora las diferencias entre mayúsculas y minúsculas.
- -H Imprime además de las líneas, el nombre del fichero donde se encontró el patrón. Es así por defecto cuando se hace la búsqueda en más de un fichero.
- -l Cuando son múltiples ficheros sólo muestra los nombres de aquellos donde se encontró al patrón y no las líneas correspondientes.
- -v Devuelve las líneas que no contienen el patrón.
- -r Busca en un directorio de forma recursiva.
- -n Imprime el número de cada línea que contiene al patrón.

Ejemplos:

- grep linux /usr/share/doc
- grep root /etc/passwd

- grep -n error /var/log/messages
- grep -i pepe /etc/passwd
- grep -c root /etc/group
- grep -l -r -i img /var/www/html/

La dificultad con grep es realmente saber especificar el patrón de búsqueda, porque se pueden desear

<sección provisional a falta de encontrar un documento donde destinar el contenido>

Apéndices

Resumen de símbolos especiales

Símbolo	Efecto	Ejemplo
"espacio"	Separa argumentos entre ellos y del comando	ls un fichero grande Lista tres ficheros llamados "un", "fichero" y "grande"
>	Redirige la salida del comando a un directorio	echo "esto es el contenido del futuro fichero" > fichero.txt
<	Redirige desde un fichero a la entrada de un comando	sort < fichero_nombres.txt Mostrará ordenadas las líneas del archivo

;	Separa dos comandos totalmente	`$ ls –l fich1 ; ls fich2`
		`–rwxrwxr–x … fich1`
		`fich2`

'	delimitan argumentos en los que no se interpreta ningún símbolo	`$ echo '?* \l ja ja'`
		`?* \l ja ja`
		
		`$`

Orden de proceso de los comandos.

1. División de tokens
2. Revisión de funciones
3. Expansión de alias
4. Sustitución de variables
5. Expansión de comodines
6. ajuste de redirección
7. Ejecución del comando Estandar

https://docs.google.com/document/d/10wOJKCgAgsOEDmuj5XuliDtFHhkp3w_uwKmLwkCKvxc/edit?usp=sharing

TOMA TOMA TOMA:

https://serverfault.com/questions/178457/can-i-send-some-text-to-the-stdin-of-an-active-process-running-in-a-screen-sessi