Capítulo 1

Identificación; páginas de manual; estructura de carpetas del sistema UNIX (i)

1.1 Identificación

La primera tarea del usuario es acceder al entorno del sistema y accionar aquellos programas necesarios para validarse frente a el (hacer `login`). A partir de ese momento se inicia un *shell* conectado a un terminal (pantalla + teclado) con el que interacciona el usuario vía mensajes textuales.

Una vez conseguido esto, el usuario acciona el intérprete mediante líneas que tienen este aspecto:

`mandato␣[[-]`*`ops`*`]␣[`*`opsArgs`*`␣[`*`args`*`]`	Donde *`ops`* son las opciones del mandato, *`opsArgs`* los argumentos de las opciones, y *`args`* los datos de partida del programa

Al final de la línea, el usuario debe pulsar la tecla de entrada o fin de línea <R>.

`ls`	Lista de archivos y directorios
`ls␣-l`	idem en formato amplio
`ls␣-al`	idem más archivos ocultos
`ls␣-l␣..`	Lista de la carpeta anterior a la actual

Al final del trabajo, el usuario debe *cerrar* la sesión, para ello utiliza el mandato

`exit`	Salir de la sesión

Existen combinaciones de teclas que permiten realizar acciones (por si solas). Entre ellas encontramos

<Back Space> *o* <Ctrl-H> (^H)	Borra el caracter anterior y mueve el cursor a la posición anterior.
<Ctrl-U> (^U)	Borra la línea completa.
<Ctrl-C> (^C)	Interrumpe y termina el mandato actual.
<Ctrl-S> (^S)	Para la salida de datos por pantalla.
<Ctrl-Q> (^Q)	Reactiva la salida de datos por pantalla.
<Ctrl-D> (^D)	Cierra el terminal (*exit*).

Algunos mandatos que podemos utilizar para conocer algo de información sobre nuestra sesión son: `date`, `who`, `hostname`, `domainname`

`date`	Indica la fecha y hora actual.
`who`	Indica quién está conectado al sistema.
`hostname`	Muestra el nombre completo del sistema
`domainname`	Muestra el nombre del dominio de comunicación
`uname`	Proporciona información sobre el sistema

1.1.1 Ejercicios

- Pruebe con los mandatos `date`, `who`,`who␣am␣i`, `hostname`, `domainname`.
- Pruebe con `uname␣-a`
- Según escribe mandatos, pruebe con las combinaciones de teclas importantes que se describen más arriba.

  Por ejemplo:

  1. Escriba `xwhoami`
  2. Cancele la línea con `^U`
  3. Escriba `whoami`
  4. Borre los caracteres `ami`
  5. Añada `␣am␣i` y pulse `<R>`

1.2 El manual de UNIX

La documentación se agrupa en dos bloques principales: páginas de manual (o páginas *man*) y documentos suplementarios. Las primeras se pueden consultar en papel y haciendo uso del mandato «`man`». Los segundos son documentos adicionales que explican temas concretos y pueden estar disponibles en formato electrónico.

Las páginas de manual se organizan en secciones, que en Solaris y HP-UX siguen el siguiente esquema:

1 Mandatos de nivel de usuario y aplicaciones.

1m Mandatos de administración del sistema (8 en Linux y FreeBSD).

2 Llamadas al sistema y códigos de error del núcleo.

3 Biblioteca de llamadas.

4 Formatos de archivos estándar (5 en Linux y FreeBSD).

5 Miscelánea de archivos y documentos (7 en Linux y FreeBSD).

6 Juegos y demostraciones.

7 Manejadores de dispositivos y protocolos de red.

8 Especificaciones obscuras del núcleo e interfaces.

Los contenidos de las páginas de manual se organizan uniformemente en párrafos que siguen el esquema: nombre, sinopsis, descripción, lista de archivos, información relacionada, errores/advertencias, problemas conocidos (y algunos temas más).

`man␣`*`título`*	Muestra la página de manual que corresponde al *título*, pantalla a pantalla.
`man␣-s␣`*`núm. título`*	Idem pero indicando la sección, dado que es posible que dos páginas de diferente sección tengan el mismo *título*. En Linux y HP-UX no es necesario incluir la opción `-s`.
`man␣-k␣`*`lista claves`*	Muestra un conjunto de títulos donde se encuentra alguna de las palabras clave de la *lista de claves* en el apartado de sinopsis.

Ojo: Muchos de los mandatos del nivel de usuario son en realidad mandatos propios del shell. No los busque en el manual directamente, sino en la página del shell (ksh, sh, csh, ...) que esté usando (p.ej. `jobs`).

1.2.1 Ejercicios

- Pruebe en primer lugar con el mandato `man␣man`. En él se describe su sintaxis de uso. Compruebe las partes en que se organiza la tabla. Como el documento es un poco largo solo se muestra una pantalla cada vez. Para pasar a la siguiente, pulse la tecla `<R>`.

  Además dispone de teclas especiales, consúltelas pulsando la tecla «`h` mientras se muestra una página de manual.

- Pruebe con mandatos conocidos:
  - Escriba `man␣who`
  - Escriba `man␣uname`
  - Escriba `man␣hostname`
  - Escriba `man␣-s1m␣init`

- Vea como la misma palabra puede dar título a dos páginas distintas:
  1. Escriba `man␣-k␣sleep` para ver qué páginas están disponible sobre este tema.
  2. Observe cómo aparece en el manual 1 y el 3.
  3. Escriba `man␣-s1␣sleep`
  4. Escriba `man␣-s3␣sleep`

1.3 Estructura de carpetas del sistema UNIX (i):

El sistema estructurado de nomenclatura de archivos se traduce en la existencia aparente de un árbol de carpetas o directorios.

La carpeta o directorio de trabajo actual es el nombre que se da a la carpeta que sirve para referirse a los archivos y carpetas, utilizando su nombre relativo. Pruebe

`pwd`	Indica el directorio de trabajo actual (*present working directory*).

Pruebe

`ls␣-p`	Igual que `ls` pero añade `/` a los nombres de archivos
`cd␣`*`nombre`*	Cambia el directorio de trabajo al indicado en *nombre*
`cd␣..`	Cambia el directorio de trabajo al *padre* del directorio actual
`cd␣.`	Permanece en el mismo directorio

De los directorios del sistema de nombres de cualquier sistema UNIX, cada usuario *real* posee uno para alojar sus archivos y carpetas. Esta carpeta se denomina *home* y es la carpeta raíz del usuario.

Para situarse en este directorio basta con utilizar:

`cd`	(*sin opciones*)
`cd ~`	La tilde representa esta carpeta
`cd $HOME`	La variable `HOME` contiene el nombre de esta carpeta

1.3.1 Ejercicios

- Con los mandatos `cd`, `ls` ya está listo para hacer un mapa de la estructura de directorios del sistema de archivos del sistema donde está conectado (si el sistema está instalado al modo habitual). ¡Hágalo hasta el nivel 3!.
- Averigüe cuál es el nombre absoluto de la carpeta que es su directorio de inicio.

Capítulo 2

Archivos (i); nombres mediante metacaracteres

2.1 Archivos (i)

Gran parte de la gestión y utilización del sistema está implementada a través del sistema de nombres de los archivos del sistema. En UNIX se dá nombre a:

- Archivos ordinarios (datos, programas).
- Carpetas (directorios).
- Enlaces simbólicos a otros nombres (!) de archivos.
- Canales de acceso a dispositivos.
- *Pipes* con nombre (colas FIFO).
- Otros, como procesos, variables del sistema, ...

Para conocer a qué se refiere el nombre, se recurre a `ls␣-p` (listado con marcas) o a veces a `ls␣-l` (listado largo). Supongamos que utilizamos este último sobre una carpeta ficticia, un poquito especial, donde el resultado es:

```
crw-rw----  1 root     sys       26,   0 Apr 17  1999 wvisfgrab
brw-r-----  1 root     disk      13,   0 May  5  1998 xda
prw-r--r--  1 jaimeba  Usuarios        0 Oct 25 11:44 clientes
drwxr-xr-x  5 berbaz   Usuarios     1024 Sep 18 10:11 proyecto
-rwxr--r--  1 lercom   infor         374 Apr 19  1997 setb
lrwxrwxrwx  1 root     root           10 Jan 15  1999 tmp -
> ../var/tmp
```

El primer carácter de la columna distingue el tipo de nombre. `c` alude a un dispositivo de tipo *carácter*, `b` de tipo *bloque*, `p` distingue un *pipe* con nombre, `d` un directorio que podemos explorar. La penúltima línea se refiere a un archivo corriente que contiene un programa, y la última (`l`) al caso de un directorio que resulta ser un enlace simbólico a otro directorio que pudiera encontrarse en otro volúmen lógico del sistema de archivos.

Como quiera que explorar algunos directorios es difícil, si aparecen muchos nombres, aconsejamos emplear `ls -l` con la siguiente sintaxis (que se explicará más adelante).

`ls␣-l␣|␣more`	La salida de `ls` se canaliza a un proceso de paginación

Para conocer el tipo y cometido de un archivo, se suele recurrir a `ls␣-l`, y también se suele asignar un nombre apropiado que dé información de su utilidad. En esta línea existe un acuerdo implícito según el cual, los archivos terminados en:

- `.c`, `.p` y `.f` contienen programas en C, Pascal y Fortran, respectivamente.
- `.cf` contienen datos de configuración.
- `.txt` contienen archivos en texto ASCII.
- `.tex` contienen archivos con documentos TEX o LATEX.
- ...

En general, la operación `file␣*` intentará proporcionar esta información tras analizar el contenido de los archivos involucrados.

```
Maker:    Bourne-Again shell script text
Readme:   symbolic link to /usr/local/Readme
clientes: fifo (named pipe)
datos:    perl commands text
ls:       ELF 32-bit LSB executable, Intel 80386, ...
pp4.jar:  Zip archive data, at least v1.0 to extract
prog:     C program text
prog.c:   C program text
tea:      directory
```

2.1.1 Ejercicios

- Generemos unos cuantos archivos sobre la marcha:
  1. Escriba `touch␣mi_archivo` y pulse <R>.
  2. Escriba `cat␣>␣hola.sh` y pulse <R>.
  3. Escriba `#!/bin/bash` y pulse <R>.
  4. Escriba `echo␣hola␣mundo␣exterior` y pulse <R>.
  5. Pulse ^D
  6. Escriba `cat␣>␣hola.pl` y pulse <R>.
  7. Escriba `#!/bin/perl` y pulse <R>
  8. Escriba `print␣"hola␣mundo␣exterior\n"␣;` y pulse <R>
  9. Pulse ^D
- Ya tenemos algunos archivos. Pruebe qué ocurre si escribe `file␣*`.
- Recorra un poco el árbol de directorios del sistema utilizando `file␣*`; Específicamente sobre `/bin`, `/dev`, ...

2.2 Nombres mediante metacaracteres

Los metacaracteres nos ayudan a especificar conjuntos de archivos de una forma breve y formal. Hablando técnicamente, nos ayudan a construir expresiones regulares.

Como ejemplos podemos poner:

`*`	se refiere a cualquier nombre incluso uno vacío.
`a*`	cualquier nombre que empiece por `a`.
`[ab]*`	cualquier nombre que empiece por `a` o `b`.
`[0-9]*`	cualquier nombre que empiece por cualquier dígito.
`?`	se refiere a cualquier nombre de una sola letra.
`[a-b]*`	cualquier nombre que empiece por cualquier letra minúscula.
`ho?a.c`	cualquier nombre que empiece por `ho` y acabe por `a.c` y contenga una letra en medio.

Las expresiones regulares (véase `man␣regexp`) pueden ser muy complejas y muy potentes.

El shell del UNIX, cuando ejecutamos mandatos como `ls␣*` captura ciertos caracteres especiales y los interpreta «antes» de evaluar el mandato (en este caso ls). Aquí en primer lugar busca en el directorio actual todos aquellos archivos cuyo nombre concuerda (*match*) con la expresión regular `*`; a continuación le proporciona la lista completa (de una vez) al mandato `ls`.

En definitiva, es el *shell* quien *expande* nuestra línea «*antes*» de ejecutar el mandato.

2.2.1 Ejercicios

- Pruebe escribiendo combinaciones como:
  - `file␣hola.*`
  - `file␣*.c`
  - `ls␣ho?a*`
  - `ls␣[hm]*`, etcétera...
- Pruebe `file␣*`
- Recorra un poco el árbol de directorios del sistema utilizando otros directorios un poco más poblados.

No deje de consultar los apuntes y las referencias bibliográficas (y los apuntes escritos) para ver formas de utilizar expresiones regulares. Compruebe la línea «antes» de ejecutar cada mandato.

Capítulo 3

El editor vi (i)

UNIX dispone de diversos editores estándar de archivos de texto estándar. Entre ellos *vi* permite crear y manipular archivos de texto, muy grandes, de una forma rápida y muy confortable. De los editores como *ed*, *ex* y *sed*, hereda los mandatos en «modo línea», y a estos añade el modo de edición a pantalla completa mediante el uso de un «cursor de texto».

3.1 Iniciar vi; modos del vi

Se puede invocar el vi con el fin de crear un archivo que no existe o modificar uno nuevo.

`vi␣[`*`nombre`*`]`	El *`nombre`* es opcional y se puede indicar al guardar el archivo

En el momento de entrar en el vi estamos en modo mandato, siendo éste uno de los tres modos que se pueden utilizar.

A partir de aquí se nos ofrece un conjunto de mandatos que nos permiten añadir, borrar, buscar y modificar texto, en modo pantalla. Al teclear los mandatos no se aprecia mas que el resultado de la acción, por lo que se puede decir que se teclean «a ciegas». En este punto lo único que se aprecia, es el texto y un bloque que llamamos «cursor» y que nos marca el «carácter actual».

Como consecuencia de un mandato de inserción, nos encontraremos en situación de añadir o sobreescribir texto. Cuando finalicemos de insertar texto se pulsa la tecla `<ESC>` y se vuelve al modo de mandato a pantalla completa.

Finalmente para realizar alguna operación compleja, se dispone del modo mandato en «modo línea». Al pulsar la tecla `:` el cursor se desplaza a la última fila del terminal y allí se podrá completar el mandato, que se puede apreciar según se escribe. Al final del mandato se pulsa la tecla `<R>` y éste se completará.

3.1.1 Ejercicios

Vamos a iniciar algunos mandatos con el fin de disponer de algún archivo de texto con el que trabajar, y así ir ilustrando la operación con el editor.

1. Invoque el vi de la siguiente forma «`vi`».
2. Inserte texto en el archivo de la siguiente forma.
   (a) Primero compruebe el aspecto del cursor.

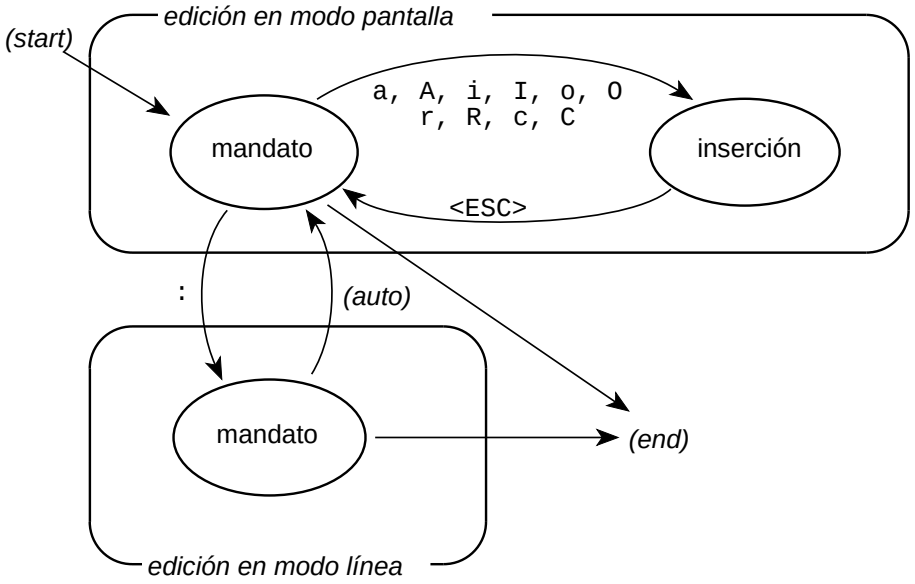

Figura 3.1: Modos de trabajo con *vi*

(b) Teclee: «`i`» y observe si se produce algún cambio en el aspecto del terminal: texto adyacente, cambio de la forma del cursor, ...

(c) Insertemos el siguiente texto:
`Esta␣es␣la␣primera␣línea␣de␣mi␣archivo<R>`
`"ejemplo.txt"␣escrita␣con␣el␣editor␣vi<R>`

(d) En este punto si desea introducir más texto hágalo.

(e) Pulse la tecla «`<ESC>`» y compruebe si se ha producido algún cambio en el entorno del terminal.

3. Salga del vi de la siguiente forma «`:wq␣ejemplo.txt<R>`».

4. A partir de aquí ya tenemos un archivo y podemos visualizarle desde el shell con el mandato «`cat␣ejemplo`».

3.2 Moviéndose por el documento

Al realizar cambios en un punto de un archivo de texto la primera tarea en este tipo de editores es: situar el cursor en el punto que nos interesa. Para saber hacer esto debemos saber como «movernos» por el documento y controlar el uso del cursor.

El primer *handicap* es que las 24 o más líneas del terminal no son suficientes para visualizar todo el texto. La imagen de nuestra situación se refleja en la figura 3.2. En élla además se incluyen algunos mandatos que nos permiten mover la zona visible del documento (pantalla) por el documento.

Algunos de estos mandatos nos permitirán mover el cursor línea a línea. Cuando intentemos avanzar más allá de la pantalla, ésta se moverá para alojar la línea del texto que buscamos. Además se incluye gran variedad de mandatos para avanzar o retroceder la pantalla más deprisa o incluso realizar búsquedas de cadenas de texto.

En primer lugar para movernos línea a línea y carácter a carácter se puede utilizar el teclado de flechas, aunque es mejor que *olvide* esta posibilidad si realmente quiere aprender a

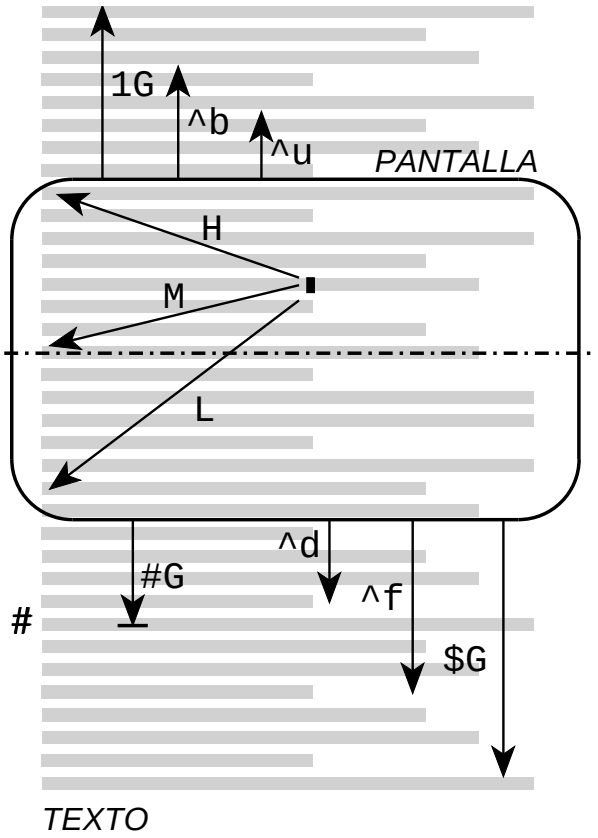

Figura 3.2: Moviendo la pantalla y el cursor

utilizar el vi. La alternativa son las teclas `h`, `j`, `k` y `l`. Su disposición es bastante lógica y muy útil, al menos cuando el terminal no está bien configurado.

En la figura 3.3 se muestran los mandatos para moverse «habilmente» dentro de la línea.

3.2.1 Ejercicios

Antes de ilustrar los mandatos de movimiento necesitamos disponer de algún archivo suficientemente largo, para experimentar. Busque uno apropiado, pero como sugerencia a fecha de 21 de Noviembre de 2000, en nuestra máquina «duero» podemos teclear:

`cp␣/etc/devlink.tab␣~`	Traemos un archivo del sistema cualquiera a nuestro directorio «*home*»

- Averigue el tamaño del archivo desde el shell. Pruebe con el mandato «`wc␣devlink.tab`». El resultado le dice por orden:

  a) número de líneas.

  b) número de palabras.

  c) número de caracteres.

- A partir de aquí se puede seguir el siguiente guión:

  1. Entre en el editor: `vi␣devlink.tab`
  2. Averigue cierta información a cerca del archivo, teclee: `^G`

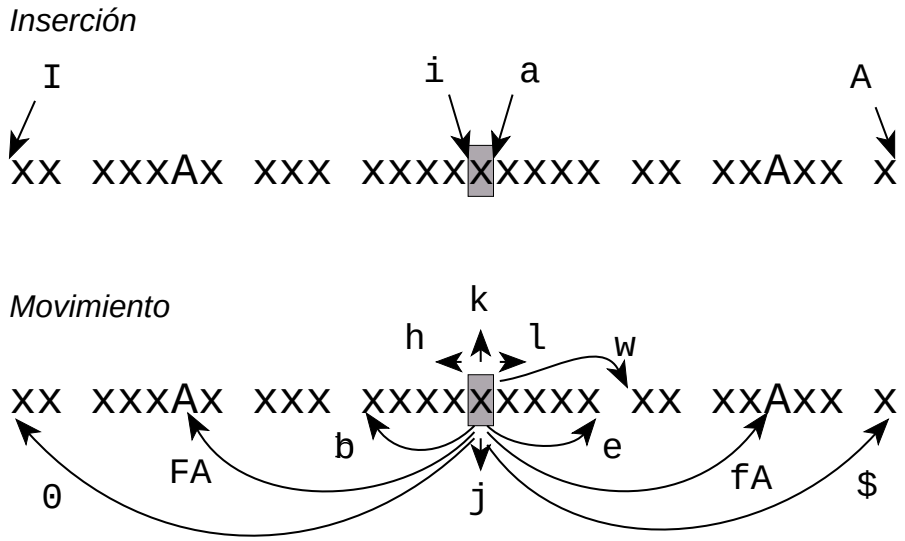

Figura 3.3: Moviendo el cursor e insertando en una sola línea

3. El mandato `G` (*Go*) es muy interesante para localizar líneas. Localice la línea 48: `48G`
   Mire en qué línea se encuentra (`^G`).
4. Localice la última línea: `$G`
5. Localice la primera línea: `1G`
6. Mover la pantalla hacia abajo de modo rápido se puede hacer con: `^d` (*down*), o más rápidamente con `^f` (*forward*). Pruebe.
7. Vaya al principio del documento.
8. Se puede avanzar un número de líneas concreto, pongamos que 12, teclee: `12^d`.
   Mire en qué línea se encuentra.
9. Y seguir avanzando de 12 en 12: `^d` hasta que no se diga lo contrario. ¡Hágalo!
   Mire en qué línea se encuentra.
10. Para mover la pantalla hacia arriba los mandatos son similares: `^u` (*up*) y `^b` (*back*).
    Vaya al final del documento y reproduzca los 3 puntos anteriores pero en el sentido de abajo hacia arriba.

- Mover el cursor utilizando las flechas o las teclas (`hjkl`) no tiene ningún secreto. Si no lo ha probado ya, hágalo ahora, pero tampoco es la forma habitual de desplazarse

1. Busque la palabra `devinfo`. Teclee: `/devinfo<R>`
   Al introducir «`/`*`texto`*» se puede buscar *`texto`* hacia delante en el documento[1].
2. Desplácese elegantemente de palabra en palabra hacia delante: `w` (*word*).
3. Desplácese elegantemente de palabra en palabra hacia atrás: `b` (*back*).
4. Vuelva a buscar la palabra `devinfo`. Teclee: `//<R>` o `n` (*next*) (la que más le guste)[2].
5. Por último pruebe a moverse al final y al principio de la línea utilizando `0` y `$`.

[1] Si quiere buscar en sentido hacia atrás, utilice `?` en lugar de `/`.
[2] Si quiere cambiar el sentido de búsqueda utilice `N`.

3.3 Insertar y borrar texto

Las modificaciones que se realizan sobre un archivo de texto pueden reducirse a dos: insertar y borrar texto. En la práctica, se presentan también otras operaciones de maquillaje como: cambiar, substituir, mover, recortar y pegar texto, que en el fondo se pueden realizar con las dos primeras.

Al insertar texto podemos indicar si se quiere realizar antes de la posición actual del cursor o despues. Además, como *vi* administra el texto por líneas, si queremos añadir líneas tendemos a utilizar mandatos específicos

`i`	Insertar (delante del cursor)
`a`	Añadir (detrás del cursor)
`o`	Insertar una línea debajo de la actual (*open*)
`O`	Insertar una línea encima de la actual (*open*)

Cada vez que se borra un fragmento de texto, éste se almacena en una zona de memoria (búfer). De este modo nos ponemos a salvo de cometer un error en una operación y no poder volver atrás. Además, si movemos el cursor, podemos pegar ése texto en otro lugar diferente, las veces que deseemos.

`u`	Deshace el último mandato del vi (salvo la modificación de archivos)
`U`	Deshace los cambios realizados sobre la línea actual (idem)

Al borrar texto se suele hacer por letras, por palabras enteras y por líneas

`x`	Borrar caracter actual
`dd`	Borrar línea actual (*delete*)
`dw`	Borrar palabra (*delete word*)

Más tarde podremos volver a poner ésta línea en cualquier otro sitio utilizando las bondades de los mandatos de pegado (*paste*).

`p`	Pega el búfer detrás de la posición del cursor
`P`	Pega el búfer delante de la posición del cursor

3.3.1 Ejercicios

- Inserte sobre la primera línea del documento las siguientes tres líneas:

```
############################################################
# Este es un archivo de prueba
############################################################
```

  Para ello siga los siguientes pasos

  1. Sitúese en la primera línea.
  2. Escriba: `O`
  3. Diréctamente escriba las tres líneas que figuran más arriba.
  4. Finalmente pulse la tecla `<ESC>` (tecla de oro de *vi*).
  5. A continuación vaya a la última línea.
  6. Pulse la tecla `.` (la tecla de plata de *vi*).
     Verá como se acaba de repetir la operación. Con el mandato `.` repetir operaciones es así de fácil.

7. Desgraciadamente la inserción se ha producido antes de la última línea; corrijamos el problema. Muévase a la última línea.
8. Bórrela con `dd`.
9. Sitúese con el cursor en el lugar donde debiera estar esta línea.
10. Pegue la línea con `p`.
11. Si la línea aparece debajo de donde piensa que debería haber aparecido, anule el cambio con `u` y utilice `P`
    De cualquier forma utilice `P` para ver cómo funciona.

- Tarea: elimine en el archivo de trabajo la palabra `rdsk` en todas las líneas.

  Sugerencias

  - Utilice el mandato `/` para localizar la palabra concreta.
  - Recuerde que se puede repetir la búsqueda con `n`.
  - Recuerde el mandato de borrar palabra: `dw`.
  - Recuerde que con el mandato `.` se pueden repetir mandatos de inserción, borrado, etc. de una forma sencilla.

- Elimine todas las líneas que contengan la cadena de caracteres `blk`

3.4 Salvaguardar y salir de *vi*

Tras realizar ciertas operaciones sobre el texto podemos guardar los cambios en el mismo archivo –u otro diferente, si tenemos permiso para ello–. Más adelante podemos salir del editor, hacer ambas operaciones a la vez, o incluso salir sin realizar cambio alguno.

`:w`	Graba las modificaciones efectuadas en el archivo actual (*write*)
`:w` *archivo*	Escribe el texto actual en *archivo* (Sólo si no existiera)
`:q`	Salir siempre que almacenemos el archivo actual o no haya cambios (*quit*)
`:q!`	Salir incondicionalmente
`:wq` o `:x` o `ZZ`	Grabar cambios y salir (ZZ: dormir)

3.4.1 Ejercicios

- Partiendo del archivo que se ha modificado en las secciones anteriores, o cualquier otro,

  1. Guarde el archivo bajo otro nombre en su directorio *home*: `:w␣~/segundo.txt<R>`
  2. Salga del editor sin modificar el archivo original: `:q!<R>`

Capítulo 4

Archivos (ii) y permisos sobre archivos

En el conjunto de herramientas que proporciona el sistema para manipular archivos encontramos un núcleo de ellas (`cp`, `mv`, ...) que se emplean en la administración de archivos, sin tener en cuenta el contenido.

Paralelamente existe una segunda categoría de aplicaciones que sirven para realizar operaciones con archivos, en las que interviene la información contenida en ellos (`diff`, `compress`, `cmp`, ...).

Capítulo aparte son las herramientas pensadas para la manipulación de archivos de texto organizados en columnas; desde las herramientas sencillas como `cut`, `paste`, `wc`, `grep`, *etcétera*, hasta lenguajes de programación completos como `awk`, pasando por editores de texto en modo batch como `sed`.

4.1 Archivos (ii)

En esta sección se describen los mandatos más importantes para manejar archivos, entre los que se comentará (sobre todo en la sección de ejercicios) algún mandato importante de la segunda y de la tercera categoría descritas en los párrafos anteriores.

Entre los mandatos, más importantes de manejo de archivos se encuentra `cp` (*copy*):

`cp␣`*`origen␣destino`*	Crea una copia del archivo *`origen`* con el nombre *`destino`*.
`cp␣`*`lista␣carpeta`*	Copia una *`lista`* de archivos en una *`carpeta`* existente.
`cp␣-R`*`lista␣carpeta`*	Copia «recursivamente» una *`lista`* de archivos y carpetas en una *`carpeta`* existente.

En los dos últimos casos, la *`lista`* de archivos puede ser un solo archivo, y que duda cabe que se pueden utilizar metacaracteres para construir dicha lista.

A este mandato se une otro de naturaleza similar: `ln`. Crea un nombre de archivo que referencia otro archivo que ya existe. Sin embargo, si esta ligadura supone rebasar el ámbito de un volúmen de archivos, solo podrá ser de tipo «simbólico».[1]

`ln␣`*`original␣nuevo`*	Crea un nombre *`nuevo`* a partir de un enlace *`original`* existente.
`ln␣-s␣`*`original␣nuevo`*	idem del anterior, pero de tipo simbólico

[1] La razón de ésto se encuentra en que al iniciar un sistema UNIX, se decide en tiempo de arranque qué volumenes forman el sistema de archivos, y si se estableciera una referencia directa desde un volumen a otro (que puede no estar «montado» en ese momento) podría causar problemas

Le siguen en importancia, a `cp`, los mandatos `mv` (*move*) y `rm` (*remove*).

`mv␣`*`origen`*`␣`*`destino`*	Mueve el archivo o directorio *`origen`* al nuevo nombre *`destino`*.
`mv␣`*`lista`*`␣`*`carpeta`*	Mueve la *`lista`* de archivos a la ubicación *`carpeta`*.
`rm␣`*`lista`*	Deshace la *`lista`* de nombres de archivo
`rm␣-rf␣`*`lista`*	idem del anterior, donde en la *`lista`* puede haber nombres de carpetas. Además lo hace de forma recursiva, de modo que borra sub–árboles de carpetas completos.

El mandato `mv` no recoloca físicamente los datos en el dispositivo de almacenamiento masivo, si no es necesario (es decir, que si los archivos no están en volúmenes diferentes), sino que se limita a modificar las listas de nombres de las carpetas, en definitiva, un cambio de nombre.

En el caso de que *`origen`* y *`destino`* estén en volúmenes diferentes se realiza una copia del archivo en el volumen destino y un «borrado» del archivo en el volumen origen (si es preciso).[2]

En cuanto al mandato `rm`, se limita a desligar el nombre de archivo del archivo físico. Si ocurre que esta ligadura es la única referencia al archivo físico se produce también un «borrado» del archivo para reaprovechar el espacio.

4.1.1 Ejercicios

En cada paso de los ejercicios, conviene utilizar repetidamente el mandato `ls` para cerciorarnos de que cada operación se ha realizado correctamente.

- Para empezar vamos a crear un par de carpetas nuevas en nuestro directorio de trabajo y algún archivo auxiliar:

  1. `mkdir␣origen`
  2. `mkdir␣destino`
  3. O las dos a la vez: `mkdir␣origen␣destino`
  4. `touch␣origen/uno`
  5. `touch␣origen/dos`
  6. `touch␣origen/tres`
  7. O los tres a la vez:
     ```
     cd␣origen
     touch␣uno␣dos␣tres
     cd␣..
     ```

- Copiemos un archivo, como muestra de la primera forma de la sintaxis del mandato `cp`: `cp␣origen/uno␣origen/copia_de_uno`

- Copiemos el conjunto de archivos `origen/*` sobre el directorio `destino`, como muestra de la segunda forma de la sintaxis: `cp␣origen/*␣destino`

- Copiemos la carpeta completa `origen` sobre `destino`, como ejemplo de la tercera forma de la sintaxis: `cp -R origen destino`.

  Comprobemos que en el directorio `destino` se ha creado una carpeta denominada `destino` con los contenidos adecuados.

[2] En UNIX, cuando se habla de borrar un archivo, en el fondo lo que se produce es una operación en virtud de la cual el espacio físico ocupado por el archivo se pone a disposición para ser utilizado de nuevo. En un sistema multitarea, de tiempo compartido, lo normal es que a los pocos segundos, si cabe, el espacio que antes ocupaba el archivo se haya reutilizado para otros menesteres.

- Intente copiar los contenidos de una carpeta que contiene archivos y directorios en otra carpeta diferente.
  - ¿Se han incluido los subdirectorios?
  - Si se quieren incluir los subdirectorios ¿Cómo debemos proceder?
  - Si en una copia recursiva se incluye el destino en si mismo ¿qué ocurre?

En lo referente a la operación de enlazar nombres se puede proponer el siguiente guión:

- Efectuamos un enlace a un archivo existente: `ln␣uno␣cinco`.
  1. Realice un listado largo del directorio y observe los cambios.
  2. Haga otro enlace sobre el mismo archivo físico.
  3. Observe los cambios.
  4. Borre el archivo original: `rm␣uno` y observe los cambios.
  5. Siga borrando los nombres de los enlaces y observe los cambios.
- ¿Se puede hacer un enlace de un directorio existente? Por ejemplo `ln␣original␣D1`, pruebelo!
- Y si el enlace es simbólico ¿Se puede hacer? ¿Qué aspecto tiene?
- Cree un archivo en el directorio `D1` que se encuentra ligado simbólicamente a `original`. Compruebe los resultados.
- Finalmente: borre el directorio `original` por pasos.
  1. `rm␣original/*`
     Si el directorio original tiene subdirectorios habría que ir por pasos.
  2. `rmdir␣original`

  ¿Permanece el enlace simbólico?

En este punto, las posibilidades de combinar los mandatos anteriores con los mandatos `rm` y `mv` crecen exponencialmente, por lo que nos vamos a limitar a lo más esencial.

- Partiendo de cero, cree un directorio `ABC` y un par de archivos en su interior.
- Intente cambiar de nombre el directorio `ABC` al nuevo nombre `ABS` mediante el mandato `mv`
- Intente borrar la carpeta utilizando el mandato `rm`.

  En caso de no se haya borrado, qué opción necesitaría añadir a `rm`. Añádala y pruebe.

4.2 Permisos sobre archivos

Los permisos sobre archivos son muy importante en UNIX, dado que mediante éstos no solo se gobierna para cada usuario qué información es accesible, sino también qué programas podrá ejecutar!

Como ya sabe, cada elemento del sistema de nombres de archivos lleva asociado dos números (uid: *User Identity*, gid: *Group Identity*) que indican el usuario y grupo propietario del

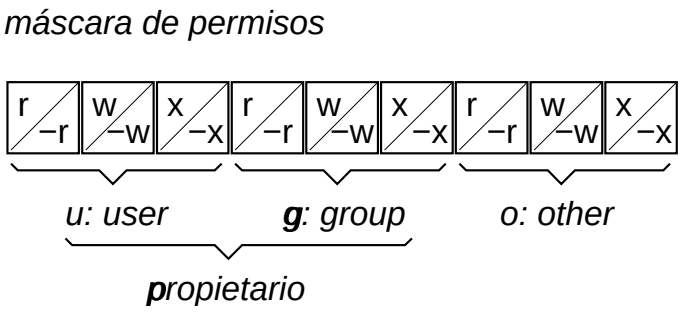

Figura 4.1: Máscara de permisos UNIX, salvo suid, sgid y *stycky bit*

archivo y un patrón de bits que codifica los derechos de acceso, modificación y ejecución (véase la figura 4.1). A esto se añaden los recursos de las listas de control de acceso (ACL: *Access Control List*) que no se comenta en esta sección.

Para modificar los propietarios se utilizan, principalmente, dos mandatos: `chgrp` y `chown`. El primero permite cambiar el grupo y se podrá utilizar si el usuario que ejecuta el mandato es propietario del archivo y pertenece al nuevo grupo al que se asigna. El segundo se utiliza habitualmente por el usuario *root* y permite hacer las dos operaciones a la vez.

Ambas admiten la posibilidad de hacer un cambio recursivo sobre un directorio (opción `-R`).

`chgrp␣`*`gDest␣nombre`*	Cambia el grupo del archivo o carpeta *`nombre`* al grupo destino *`gDest`*
`chown␣`*`uDest␣nombre`*	Cambia el usuario del archivo o carpeta *`nombre`* al usuario destino *`uDest`*
`chown␣`*`uDest`*`:`*`gDest␣nombr`*	Cambia el usuario y grupo del archivo o carpeta *`nombre`* al usuario y grupo destino *`uDest`* y *`gDest`*, respectivamente.

Con lo que respecta a la máscara de permisos, se establece al crear cada archivo de acuerdo con una patrón de permisos denominado *umask*. Esta máscara, a la vez que los posibles permisos los establece el usuario propietario del archivo o carpeta, y puede expresarse tanto en formato numérico octal como en forma de modificaciones al patrón actual.

`chmod␣`*`###␣nombre`*	Cambia los permisos del archivo o carpeta *`nombre`* según el patrón numérico de tres dígitos en octal: *###*.
`chmod␣`*`instrucc␣nombre`*	idem del anterior, salvo que en *`instrucc`* se especifican las variaciones del patrón nuevo con respecto al actual.

En el primer caso de sintaxis no puede ser más fácil

`777`	Representa: `rwx rwx rwx`.
`540`	Representa: `r-x r-- ---`.
`754`	Representa: `rwx r-x r--`.

En el segundo la instrucción consta de una letra que identifica la parte del patrón que se modifica (`u`: usuario, `g`: grupo, `o`: otros), un símbolo (`+`: admitir, `-`: denegar) y el permiso concreto (`r`, `w`, `x`) (éste último puede incluir una combinación de estas tres letras).

Por ejemplo, para los permisos `rwx r-x r--` sobre el archivo ABC las siguientes instrucciones obtienen un patrón como se indica.

`chmod␣u-w␣ABC`	Nuevos permisos: `r-x r-x r--`.
`chmod␣g-rx␣ABC`	Nuevos permisos: `rwx --- r--`.
`chmod␣o-r␣ABC`	Nuevos permisos: `rwx --x ---`.
`chmod␣o+rwx␣ABC`	Nuevos permisos: `rwx r-x rwx`.

4.2.1 Ejercicios

Como ejercicio, se plantea consignar que los ejemplos anteriores, referentes a `chmod` funcionan tal y como se indica.

- En primer lugar veamos como es la máscara de creación de archivos. Introduzca el mandato `umask`. El resultado será típicamente `022`.

  La «máscara de usuario» indica en octal qué permisos, por defecto, se anulan cuando se crea un archivo. En el caso anterior cada archivo o directorio lleva anulado, por defecto, los permisos de escritura para el grupo propietario y el resto de usuarios.

- Fíjese en un archivo cualquiera de los que tiene a su alcance y haga un enlace. Por ejemplo:

  1. `touch␣un-archivo`
  2. `ln␣un-archivo␣nuevo-enlace`
     ¿Qué permisos tienen el archivo original y el enlace?
  3. Cambiemos los permisos con `chmod␣700`
     ¿Qué permisos tiene ahora?
  4. Añadamos los permisos de lectura al grupo y al resto
     `chmod␣g+r␣un-archivo`
     `chmod␣o+r␣nuevo-enlace`
     ¿Y ahora qué permisos tiene?

- Fíjese en un archivo cualquiera de los que tiene a su alcance y haga un enlace simbólico con él.

  1. ¿Qué permisos tiene el enlace simbólico?
  2. ¿Se modifican al cambiar los permisos en el archivo original?
  3. Modifique uno de los permisos en el enlace.
     ¿¡Verdad que han cambiado los permisos del archivo original!?

- Finalmente, cambiemos los permisos de exploración y de escritura de un directorio (para el usuario propietario, por supuesto!).

  1. Cuando cambiamos el permiso de exploración...
  2. ¿funciona el mandato `ls`?
  3. ¿Podemos entrar en el sudodicho directorio?

Capítulo 5

Utilidades con archivos y directorios; Variables de entorno

Entre las herramientas de usuario de UNIX, un conjunto de ellas se dedican al trato con archivos exclusivamente y otras se dedican al trabajo con archivos y carpetas. Se podría decir que en estas últimas se auna el esfuerzo del resto de las utilidades, en una visión de conjunto.

5.1 Utilidades con archivos

Existe variedad de mandatos para comparar archivos: `cmp` para archivos de cualquier tipo y `diff` y sus derivados (`diff3`, `sdiff`, `comm`) para archivos de texto.

`cmp␣`*`nombre1`*`␣`*`nombre2`*	Compara los archivos *`nombre1`* y *`nombre2`*. Muestra la posición del primer byte diferente y retorna `0` si los archivos son iguales y `1` si son diferentes.

El valor de salida del programa `cmp` puede utilizarse en expresiones de tipo lógico para construir mandatos de carácter compuesto. En este caso, el valor `0` equivale a «falso» y el valor `1` a «cierto».

En el caso de utilizar archivos de texto, es de utilidad poder saber en qué línea se da la diferencia, e incluso que las muestre en la salida estándar. Aparte de esto, `diff` es una utilidad que no se conforma con mostrar la primera discordancia entre archivos, sino que intenta «resincronizar» de la mejor forma posible los contenidos de los archivos para comparar todo el archivo.

`diff␣`*`nombre1`*`␣`*`nombre2`*	Compara los archivos de texto *`nombre1`* y *`nombre2`*.

Como resultado produce un conjunto de líneas que expresan las diferencias y que puede utilizarse, incluso para reconstruir un archivo apartir del otro mediante la utilidad `ed`.

Supongamos que disponemos del archivo *`ABC`*

```
Este es el contenido de un archivo de texto
que contiene como ejemplo tres líneas
de texto, para ilustrar la aplicación
del mandato diff.
```

y del archivo *`abc`*

```
Este es el contenido de un archivo de texto
que contiene como ejemplo tres líneas
de texto, con el fin de ilustrar la aplicación
del mandato diff.
```

El resultado de `diff␣ABC␣abc` es:

```
3c3
< de texto, para ilustrar la aplicación
---
> de texto, con el fin de ilustrar la aplicación
```

Como quiera que los archivos de texto tienen una mala relación información/tamaño, UNIX dispone de una herramienta de compresión y descompresión de archivos que emplea el algoritmo de Lempel-Ziv adaptativo. Al comprimir/descomprimir modifica el nombre del archivo, por lo cual éste no deberá tener más que un enlace.

`compress␣`*nombre*	Comprime el archivo *nombre* y genera el archivo *nombre.Z* si el resultado es de talla inferior.
`uncompress␣`*nombre*	Descomprime el archivo *nombre.Z* y genera el archivo *nombre*.
`zcat`	Análogo a `cat` para una entrada estándar comprimida.

Una utilidad interesante es `grep`. Permite buscar un patrón en un archivo. Existe una versión rápida para buscar cadenas de texto de longitud fija: `fgrep` y otra para buscar de forma muy general expresiones regulares: `egrep`.

Otros mandatos que resultan interesantes, se incluyen en la tabla que sigue:

`crypt`	Encrypta/desencripta la entrada estándar mediante una clave.
`grep␣`*patrón*	Busca una línea que corresponda con un *patrón* en la entrada estándar y lo muestra por la salida estándar.
`fgrep␣`*texto*	Busca una línea que contenga cierto *texto* en la entrada estándar y la muestra por la salida estándar. Es más eficiente que `grep` en el caso de texto fijo.
`sort`	Ordena lexicográficamente las líneas de la entrada estándar hacia la salida estándar.
`sort␣`*lista*	Ordena lexicográficamente las líneas de la *lista* de archivos y lo muestra hacia la salida estándar.
`uniq`	Elimina líneas repetidas consecutivas de la entrada estándar hacia la salida estándar
`more`	Muestra pantalla a pantalla las líneas que se le presentan por la entrada estándar.
`more␣`*nombre*	Muestra pantalla a pantalla las líneas del archivo *nombre*.

5.1.1 Ejercicios

- En primer lugar es menester disponer de un archivo interesante para poder hacer manipulaciones. Para ello utilizaremos el mandato `ps` que muestra los procesos activos en el sistema.

  1. `ps␣-a␣>␣ps1.txt`
     la opción `-a` sirve para mostrar todos los que hay en el sistema. La siguiente parte del mandato sirve para redirigir la salida a un archivo y se abordará en el siguiente capítulo.
  2. `ps␣-a␣>␣ps2.txt`

- Podemos comparar los dos archivos, en primer lugar con `cmp` y despues con `diff` para ver qué diferencias hay al usar cada uno.
- Mire el tamaño de `ps1.txt`, e inspeccione el interior del archivo con `cat` para cerciorarse de que es un archivo de texto.

  A continuación comprimimos el archivo con `compress`. Compruebe si existe todavía el archivo `ps1.txt` y compare el tamaño del nuevo archivo con el antiguo.

  Si enseñamos el archivo `ps1.txt.Z` por pantalla (`cat`) veremos que ha cambiado.
- Con `uncompress` podemos descomprimir el archivo, pero es más interesante utilizar `zcat`.
- Podemos probar con `crypt` para encriptar el archivo, aunque el método implementado por el mandato no sea nada robusto.

  1. `crypt␣<␣ps2.txt␣>␣ps2.crypt`
     Tenemos que proporcionarle una clave.
  2. Mostremos el archivo `ps2.cryptpor` pantalla para ver que sale.
  3. Desencriptamos el archivo con la misma clave y de la misma forma
     `crypt␣<␣ps2.crypt`
- Si queremos clasificar las líneas de un archivo, disponemos de un mandato muy potente: `sort`. Es muy potente y tiene muchas opciones.

  1. `sort␣<␣ps2.txt␣|␣more`
     ordena utilizando el primer campo de cada línea (el identificador de proceso en este caso). El mandato `more` solo sirve para verlo de pantalla en pantalla.
  2. `sort␣-k␣4␣<␣ps2.txt␣|␣more`
     es lo mismo de antes pero ordenando por el cuarto campo (el nombre del programa en ejecución en este caso).
- Si queremos buscar, lo mejor es `grep`.

  – Busquemos cuantos terminales tenemos desplegados:
    `who␣-a␣|␣grep␣$LOGNAME`
    El símbolo `|` funciona para construir canalizaciones y ya lo veremos más adelante. En este caso canalizamos la salida de `who␣-a` hacia el mandato `grep $LOGNAME` donde `LOGNAME` es una variable de entorno que contiene nuestro nombre de usuario actual.
  – Busquemos cuantos «shell» estában ejecutándose en el momento de arrancar el `ps` de unos puntos más atrás.
    `grep bash < ps2.txt`

5.2 Directorios (ii)

Conocemos de los ejercicios de las secciones anteriores los mandatos más importantes de manipulación de carpetas, a parte de `ls`, `cd`, aunque quizás desconozca que el mandato `ls -R` es la versión recursiva. Aparte de ésto, los siguientes mandatos son muy importantes.

`mkdir␣`*nombre*	Crea una carpeta denominada *nombre*.
`rmdir␣`*nombre*	Destruye una carpeta vacía denominada *nombre*.
`pwd`	Muestra el directorio actual
`du␣`*nombre*	Muestra el tamaño ocupado por archivos y carpetas

Existen aun así otros mandatos importantes que sirven para administrar directorios entre los que están:

`find`	Recorre árboles de carpetas efectuando operaciones.
`tar␣`*`opciones␣nombre`	Archiva/Desarchiva árboles de carpetas en un único archivo.

5.2.1 Ejercicios

- Conocemos de sobra las posibilidades de `mkdir` y `rmdir`, según las hemos utilizado. Aunque no estaría de más que las recordara haciendo algún ejemplo muy simple; concretamente, elimine el permiso `x` para el usuario propietario de una carpeta e intente eliminarla para ver lo que pasa.
- En cuanto al mandato `find` solo cabe decir que su utilización es tan complicada como para dedicarla un capítulo para ella sola (mejor más adelante). Aun así, pruebe las siguientes líneas y deduzca qué es lo que hacen.

  1. Sitúese en el directorio `HOME`
  2. `find␣.␣-print`
  3. `find␣.␣-name␣\*error\*␣-print`
  4. `find␣.␣-newer␣ps1.txt␣-print`
     Donde `ps1.txt` es algún archivo que tengamos que tenga algo de antigüedad.
  5. `find␣.␣-newer␣ps1.tex␣-name␣\*.tex␣-exec␣banner␣\{\}␣\;`
     Mire a ver para qué sirve `banner`.

- Finalmente el mandato `tar` se utilizaba mucho para empaquetar y desempaquetar arboles completos de carpetas en un archivo, en tiempos de los dispositivos de cinta.

  Tomemos un arbol de directorios, por ejemplo el que yace bajo `.dt` (lo tendrá si ha ejecutado el desktop CDE), o cualquier otro arbol que conozca, y pruebe:

  1. Pruebe el mandato `du` para saber cuanto ocupa el directorio completo con sus hijos.
     `du␣-k␣.dt`
     La opción `-k` es para ver los datos en kilobytes.
  2. `tar␣cf␣dt.tar␣.dt`
     Para crear el archivo `dt.tar` con los contenidos del directorio `.dt`. La «`c`» viene de *create* y la «`f`» sirve para decir que deposite el resultado en el archivo `dt.tar`.
  3. `tar␣tf␣dt.tar`
     Para vér qué contiene el archivo `dt.tar`. La «`t` proviene de *tell*.
  4. `mkdir␣prueba.tar`
     para rescatar el «tar» en otro archivo. `cd␣prueba.tar`
     `tar␣xf␣../dt.tar`
     Rescata el archivo «`tar`» en este lugar. Compruébelo! Compruebe también su tamaño en bytes comparado con `du`.

5.3 Variables

El entorno de trabajo que ofrece cualquier shell permite crear, definir y consultar variables de entorno. Tales variables suelen escribirse en letras mayúsculas y están a disposición de las aplicaciones que necesiten hacer uso de ellas.

`HOME`	Contiene el nombre del directorio home de cada sesión de trabajo
`PATH`	Contiene una lista de directorios separados por ':' en los que se hallan las aplicaciones.
`TERM`	Contiene el tipo de terminal que se está empleando.

Para conocer las variables de entorno más importantes que se encuentran definidas en cada momento se emplea el mandato:

`env`	Muestra nombres de variables y su contenido.

Para definir una variable no hay más que utilizar la sintaxis

`nombre=texto`	Crea la variable *`nombre`*, de no existir, y almacena *`texto`* en su interior.

En el caso de que el *`texto`* contenga espacios habrá que encapsular éste texto entre comillas simples o dobles.

Para acceder al contenido de la variable no hay más que utilizar el *`nombre`* de la variable precedido del símbolo `$`.

`ABC='texto plano'`	Para crear la variable `ABC` y almacenar la cadena «`texto plano`».
`echo␣$ABC`	Mostrará en pantalla el mensaje: `texto plano`.

El mandato `echo` tiene la finalidad de enviar por la salida estándar la cadena de texto que se le proporciona a continuación. En este caso, simplemente accede al contenido de la variable `ABC`.

5.3.1 Ejercicios

Las variables son una parte importante del shell, y su importancia se aprecia en cómo funcionan algunos mandatos, que los consultan.

- Comenzamos viendo el valor de la variable `PATH`.
  1. `echo $PATH`
     Para ver qué contiene.
  2. `PATH_RESERVA=$PATH`
  3. `PATH=`
     Borramos el PATH actual.
  4. Compruebe si funciona algún mandato como `ls` o `more`
  5. `PATH=$PATH_RESERVA`
     ¿Y ahora, funcionan?
- La variable `TERM` es también importante. Programas que manejan la pantalla la utilizan, como `vi`, `elm`, `pine`, `lynx`, ...
  1. `echo $TERM`
     Para ver qué contiene. Su valor está determinado por el terminal que utiliza y tiene que ser uno de los que aparecen bajo el directorio `terminfo`, que suele estar en `/lib/terminfo` o `/usr/share/lib/terminfo`
  2. `TERM_RESERVA=$TERM`
  3. `TERM=`
     Borramos el PATH actual.

4. Compruebe si funciona `vi` con algún archivo.
5. `TERM=$TERM_RESERVA`
   ¿Y ahora, funciona?

- Pruebe los mandatos:
  `resize`
  `eval␣‘resize‘`
  Investigue para qué sirven. Ojo con las comitas alrededor de `resize`.

Capítulo 6

Redirección de los canales de entrada y salida; canalizaciones

Una de las abstracciones importantes en UNIX es el *stream*. En pocas palabras, un *stream* es un canal de entrada-salida con con un interfaz peculiar y, sobre todo, estándar.

Cada proceso en UNIX tiene por derecho propio tres streams asignados: `stdin`, `stdout` y `stderr`, aunque se pueden definir muchos otros más. Respectivamente:

0 `stdin` es la entrada estándar y se asocia inicialmente al teclado del terminal.
Por este canal recibe los datos cada proceso.

1 `stdout` (o `&1`) es la salida estándar y se asocia inicialmente al dispositivo de salida del terminal, abreviando: la pantalla.
Por este canal se envían los resultados del procesamiento.

2 `stderr` (o `&2`) es la salida estándar de errores y se asocia inicialmente a la pantalla.
Por este canal se suelen enviar las advertencias y errores que ocurren en el proceso.[1].

Lo interesante está en que estos canales de entrada-salida pueden conectarse (y reconectarse) entre si:

- los canales de un proceso con los de otro;
- con un canal que parte o llega a un archivo;

[1] Un pequeño problema es que la salida de error aparece mezclada con la salida estándar y además, no siempre están «sincronizadas», es decir las líneas que salen por stdout y stderr no tienen que estár ordenadas temporalmente `:-(`

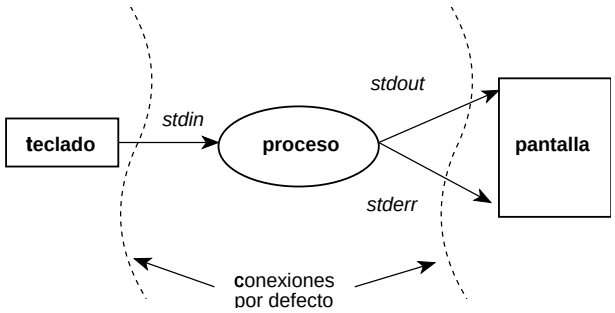

Figura 6.1: Descritores estándar de entrada-salida y su conexión de partida.

- con un canal que parte o llega a un dispositivo.

Consecuentemente realizar tareas complejas se convierte, muchas veces, en conectar procesos simples entre si.[2]

6.1 Redirección de entrada-salida

Al procedimiento de redirigir las conexiones desde su configuración de partida (ver figura 6.1) se le denomina redireccionamiento. Comencemos con la redirección desde-hacia archivos.

mandato < *archivo*	Conecta la entrada estándar de un *mandato* con un *archivo*.

De esta forma, cualquier mandato que recibe sus datos desde el teclado se puede conectar con un archivo. Por ejemplo, supongamos que tenemos un archivo con unas cuantas líneas, denominado `ejemplo`:

`cat < ejemplo`	Muestra en pantalla los contenidos del canal de entrada, que se encuentra asociado al archivo `ejemplo`.
`sort < ejemplo`	Ordena lexicográficamente los contenidos del archivo anterior.
`wc < ejemplo`	Cuenta las líneas, palabras y caracteres del archivo.

La redirección de la salida estándar es fácil también:

mandato > *archivo*	Crea un *archivo* y se envía la salida del *mandato*
mandato 1> *archivo*	(idem del anterior)
mandato >> *archivo*	Añade al *archivo* la salida del *mandato*
mandato 1>> *archivo*	(idem del anterior)

Así, no necesitamos programar la salida de datos hacia un archivo, dado que podemos hacerlo desde el shell de UNIX.

`echo Hola > Hola`	Creamos el archivo `Hola` con una línea de contenido: `Hola`.
`cat < ejemplo > dl.t`	Creamos el archivo `dl.t` a partir del archivo `ejemplo`.
`wc dl.t > abc`	Crea el archivo `abc` con las cuentas del archivo `dl.t`.
`wc dl.t 1> abc`	(idem del anterior)
`wc dl.t >> abc`	... si queremos mantener el contenido anterior del archivo `abc`.

Con la salida de error estándar podemos hacer algo parecido.

mandato 2> *archivo*	Crea un *archivo* y se envía la salida de errores del *mandato*
mandato 2>> *archivo*	Añade al *archivo* la salida de errores del *mandato*

Con lo que, si queremos, podemos guardar los mensajes de error de un mandato en un archivo normal.

`wc no-existo 2> WC`	Suponemos que el archivo `no-existo` no existe
`wc no-existo > WC 2>&1`	(idem del anterior), pero: se conecta la salida estándar al archivo `WC` y la salida de errores a la salida estándar!!.
`wc no-existo 2> WC 1>&2`	(idem del anterior), pero: se conecta la salida de errores al archivo `WC` y la salida estándar a la salida de errores!!.

En el ejemplo anterior se ilustra cómo también podemos conectar `stdout` y `stderr` entre si. Cuidado, porque si cambiamos el orden de las redirecciones el shell se puede quejar.

[2] Este tipo de manipulaciones están en la línea del diseño de UNIX: combinar elementos simples para construir uno mismo otros más complejos. Podemos construir programas que reciben y comunican datos por los canales estándar, y más tarde los conectaremos con el dispositivo adecuado.

6.1.1 Ejercicios

Partimos de la base de que tenemos un archivo llamado `ejemplo.txt` con algo de texto en su interior, y que utilizaremos para ilustrar el comportamiento de la redirección.

- En principio, pruebe los ejemplos de la sección que acaba de ver. Cuando haya terminado, pase al siguiente punto.
- Hay un archivo de dispositivo muy útil denominado `/dev/null`. Pruebe cómo funciona con estos ejemplos, e imagine para qué sirve.
  - `cat␣<␣/dev/null`
  - `wc␣<␣/dev/null`
  - `cat␣ejemplo.txt␣>␣/dev/null`
  - `wc␣archivo-inexistente`
    y ahora
    `wc␣archivo-inexistente␣2>␣/dev/null`
- ¿Que pasa si copiamos un archivo sobre si mismo de la siguiente forma?
  `cat␣ejemplo.txt␣>>␣ejemplo.txt`
- Lo interesante es aprovechar la salida de algunos mandatos existentes para incluirla en archivos de texto que luego podemos editar, como con:
  - `ls␣-l␣>␣archivos-home`
  - o todos mis archivos:
    `ls␣-R␣>␣todos-ls`
    o mejor pruebe con:
    `find␣.␣-print␣>␣todos-find`
  - mire los archivos creados anteriormente (por curiosidad).

6.2 Canalizaciones

Con las canalizaciones, se riza el rizo y se permite conectar también procesos entre si.

mandato␣`|`␣*mandato2*	Conecta la salida estándar de un *mandato* con la entrada estándar de otro *mandato2*.

Si queremos que la salida de error tambien vaya por la canalización debemos redirigirla mediante «`2>&1`». En este caso se mezclará con la información de `stdout`.

Lo bonito de esta construcción es que, cuando sea posible, los datos que va generando cada proceso se envían al siguiente sobre la marcha. Este es el caso de:

```
who␣-a␣|␣grep␣Juan
```

en donde se disparan los dos procesos a la vez, y según `who␣-a` proporciona líneas, `grep␣Juan` expurga aquellas que contienen la cadena `Juan`. Otro caso es cuando se inserta algún proceso donde se necesita toda la entrada para producir una salida con sentido, por ejemplo:

```
who␣-a␣|␣sort
```

donde el mandato `sort` va a necesitar que aparezcan todas las líneas para poder ordenar. En este caso, la canalización anterior *pipe* es equivalente a la siguiente secuencia.

```
who␣-a␣>␣quien-está-ahi
grep␣Juan␣<␣quien-está-ahi
```

6.2.1 Ejercicios

La cantidad de combinaciones que podemos encontrarnos al usar canalizaciones es muy grande, y depende de la cantidad de «filtros» de que dispongamos, que es así como se denominan los procesos que, partiende de una fuente de información, van obteniendo sucesivos refinamientos aplicando canalizaciones.

- En primer lugar, realice los ejemplos que se proponen en la sección, pero, como es lógico, cambiando `Juan` por algún otro nombre de usuario que esté activo en el sistema (por ejemplo en `LOGNAME` hay uno).
- Pruebe con el mandato:
  `ps␣-efl␣|␣tee␣ps-completo␣|␣grep␣bash`
  Donde se busca la información completa sobre los procesos del sistema que están ejecutando el intérprete `bash`

  El mandato `tee` es muy interesante, pues deja pasar la información al siguiente mandato, pero se queda con una copia en un archivo. Compruebe el archivo `ps-completo`!
- Pruebe con el mandato:
  `ps␣-fu␣$LOGNAME␣|␣sort␣|␣more`
  `more` no actua como un filtro precisamente, suele utilizarse al final de una cadena de este tipo, para mostrar la información pantalla a pantalla.
- Finalmente, pruebe con:
  `ls␣-alR␣~␣|␣grep␣$LOGNAME␣|␣sort␣-k␣5␣|␣more`
  que muestra todos los archivos que cuelgan de `HOME` ordenados por tamaño (es la quinta columna del listado `ls -l`, página a página.

  El `grep` sirve para eliminar aquellas líneas que no contienen el nombre de usuario, que son unas cuantas. Pruebe a utilizar la canalización completa sin insertar `grep`.

  Lástima que se pierde el camino al archivo. Pruebe con:
  `ls␣-alr␣‘find␣.␣-print‘␣|␣grep␣$LOGNAME␣|␣sort␣-k␣5␣|␣more`
  por poner algo, pues se puede hacer todo con el mandato `find`.

  Investigue qué es eso de las comitas, y para qué sirve.

  Pista: pruebe primero con `cat␣<␣‘ls␣-l‘`.

Capítulo 7

Procesos y control de trabajos

UNIX se concibió como un sistema multitarea, y multiusuario, y aunque está bien saber que el sistema es capaz de ejecutar varias tareas concurrentemente, tambien podemos hacer uso de ello lanzando sesiones y trabajos, en paralelo.

Los estados aparentes que presenta un proceso se indican en la figura 7.1. Como es lógico, en una máquina monoprocesador solo habrá un proceso en ejecución (0), mientras que puede haber varios en cola (R) y otros más en situación de espera, bien forzada (T) o bien a la expectativa de que se atiendan sus necesidades de acceso a algún recurso o señal (S).

A todo esto se une la peculiar forma en que se multiplican los procesos en UNIX. La creación de procesos nuevos se produce mediante un proceso de clonación (ver figura **??**) diseñado para:

- crear un nuevo hilo de ejecución.
- heredar el entorno de ejecución del proceso «padre».
- heredar los atributos de ejecución del proceso «padre» (usuario, grupo propietario, etc.). Con lo que se conservan las garantías de derechos de acceso a recursos del sistema, etc.

Desde el momento de la creación del nuevo proceso, el proceso hijo mantiene una relación de subordinación con el proceso padre que en algunas circunstancias puede llegar a romperse y a degenerar en procesos perdidos o como se dice vulgarmente «colgados».

En este apartado vemos los mandatos más comunes para observar e interactuar con los procesos que circulan en UNIX.

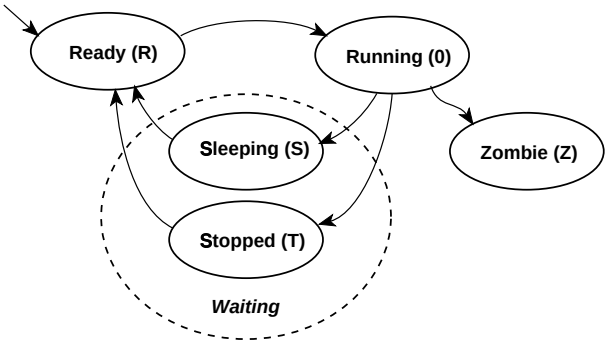

Figura 7.1: Diagrama de estados de un proceso

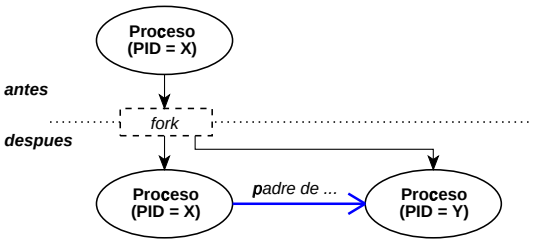

Figura 7.2: Clonación de procesos con *fork*.

7.1 Remitir trabajos

El modo usual de remitir un trabajo al sistema es confeccionar una línea de órdenes y pulsar la tecla de retorno. Las tareas así accionadas, se dice que se ejecutan en «*primer plano*», y una vez accionadas el *shell* no devuelve el control al usuario hasta que no se han completado los mandatos.

Existen otros modos de ejecutar trabajos al sistema:

- En «*trasfondo*», segundo plano o *background*: cuando en la misma sesión se ejecutan otros mandatos pero el control del terminal pertenece al usuario para seguir introduciendo más mandatos.
- «*Emplazados*»: cuando se envían trabajos a ejecutar a una cierta hora, independientemente de que el usuario tenga abierta una sesión con el *shell* en ese momento.

Los procesos en trasfondo y emplazados se ejecutan con una prioridad ligeramente inferior, dado que no tienen requerimientos de E/S de terminal de usuario. Para los procesos en trasfondo la sintaxis es:

mandato ␣&	Envía un mandato a ejecutar en trasfondo
netscape ␣&	Despliega el visualizador de páginas Web en trasfondo
acroread ␣x.pdf ␣&	Despliega el lector de documentos Acrobat en trasfondo

En esta tesitura, y bajo `bash` (Bourne Again Shell) y `ksh` (Korn Shell), es posible interrumpir un trabajo en ejecución mediante <^Z> y mandarlo a trasfondo mediante el mandato `bg` (*background*).

Más tarde podemos volver a traerlo a primer plano mediante el mandato `fg`

bg	Manda a trasfondo el último trabajo «parado»
bg ␣%#	Manda a trasfondo el trabajo identificado por el número #.
fg	Manda a primer plano el último trabajo enviado a trasfondo.
fg ␣%	Manda a primer plano el trabajo identificado con el número #.
jobs	Muestra los trabajos que se están ejecutando en la sesión.

Para poder identificar los trabajos por su número necesitamos el mandato `jobs`.

Cuando queremos que un trabajo en trasfondo siga su ejecución aun cuando salgamos de la sesión, podemos utilizar el mandato `nohup`, lo que ocurre típicamente cuando tenemos la impresión de que este mandato no acabará antes de que nos despidamos de la sesión.[1]

[1] Con `nohup` el proceso toma como padre a `init` (PID=1), e ignorará señales de interrupción y suspensión. Además los streams de estándar de salida se conectan al archivo «`nohup.out`».

`nohup␣`*`mandato`*`␣&`	Envía un proceso a trasfondo aislándolo de la sesión.

Finalmente, se puede mandar ejecutar trabajos de modo «emplazado» con el mandato `at`, siempre y cuando el usuario tenga permiso para ello. La sintaxis simplificada del mandato es como sigue:

`at␣-m␣-s␣-t␣`*`tiempo`*	Manda ejecutar en el momento indicado por *`tiempo`* los mandatos contenidos en *`archivo`* bajo el Bourne shell `bash`.
`at␣-l`	Lista los trabajos enviados por el usuario bajo el mandato `at`.
`at␣-r␣`*`idTrabajo`*	Borra el trabajo identificado por *`idTrabajo`* de la lista.

El `at` quedará a la espera de que introduzcamos el mandato que tiene que lanzar en el momento adecuado.

```
at -m 2pm next week
       sort < ‘ps -ef‘ > procesos.txt
```

```
at now + 1 hour <<!
     diff archivo1 archivo2 2>&1 >resultados | mailx pedro
```

7.2 Control de trabajos

El primer mandato interesante es `ps` (*process status*). Su sintaxis, con sus opciones más importantes son:

`ps`	Muestra los proceso activos del usuario en la sesión en la que se ejecuta el mandato `ps`.
`ps␣-f`	Idem del anterior pero expandiendo la información y mostrando cada mandato completo (*full*).
`ps␣-u␣`*`usuario`*	Muestra todos los procesos de un *`usuario`*.
`ps␣-e`	Muestra todos los procesos del sistema entero.
`ps␣-l`	Idem de `ps` mostrando toda la información de los procesos.

Las opciones anteriores se pueden combinar, y el resultado es una tabla donde los principales descriptores, en virtud de las opciones de `ps`, son los son los siguientes:

`S`	Estado del proceso: 0→en ejecución, R→listo (ready), S→esperando un evento (sleeping), T→parado (stopped), Z→finalizado pero aun con recursos (zombie).
`UID`	Número de identificación de usuario.
`PID`	Número de identificación de proceso.
`PPID`	Número de identificación del proceso padre.
`PRI`	Número que indica en qué medida puede bajarse la prioridad de un proceso.
`TTY`	Terminal al que está asociado el proceso.
`CMD`	El mandato que lanzó el proceso.

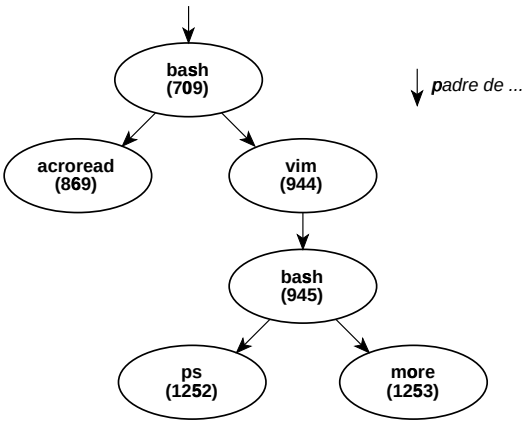

Figura 7.3: Árbol de parentesco de varios procesos.

El PID resulta muy útil para averiguar la relación de parentesco entre procesos. Supongamos la siguiente secuencia de operaciones:

1. `acroread␣raiz.pdf␣&`
   El usuario `perico` lanza desde el shell el lector `acrobat` en *background*.
2. `vim␣s5.tex`
   A continuación lanza el editor `vi` sobre un archivo.
3. `:sh`
   Dentro del `vi` sale temporalmente del editor.
4. `ps␣-fl␣|␣more`
   Ejecuta `ps` página a página con `more`.

```
  F S UID     PID PPID C PRI NI ADDR   SZ WCHAN  STIME TTY       TIME CMD
000 S perico  709  707 0  60  0    -  439 wait4  09:44 pts/1 00:00:00 -bash
000 S perico  869  709 0  60  0    - 1457 do_sel 10:42 pts/1 00:00:02 acroread r.pdf
000 S perico  944  709 0  60  0    - 1215 wait4  11:00 pts/1 00:00:00 vim s5.tex
000 S perico  945  944 0  77  0    -  450 wait4  11:00 pts/1 00:00:00 /bin/bash
000 R perico 1252  945 0  77  0    -  629 -      15:52 pts/1 00:00:00 ps -fl
000 S perico 1253  945 0  77  0    -  280 pipe_r 15:52 pts/1 00:00:00 more
```

Apartir de los datos sobre PID y PPID sabemos que los procesos están emparentados según el gráfico de la figura 7.3.

Además podemos ver en la columna S que salvo el proceso `ps -fl`, que está listo para ejecución, todos los procesos están a la espera de una señal. La prioridad (PRI) es mayor en los tres últimos procesos. La bondad de cada proceso (NI) es 0.[2]

Otro mandato interesante para analizar el tiempo invertido en un mandato es

`time␣`*`mandato`*	Muestra el tiempo invertido al procesar un mandato.

Específicamente muestra:

real: El tiempo durante el cual el mandato ha estado en estado de ejecución.

[2] Un pequeño detalle es que cuando se muestra la tabla de procesos, debido a condiciones de velocidad del sistema, en lugar de presentarse el proceso `more` se puede presentar una copia de `bash` a falta del subsiguiente `exec`.

user: El tiempo que transcurre desde que se acciona el mandato hasta que el shell devuelve el control al usuario.

sys: El tiempo invertido por el sistema en procesar las llamadas a recursos del núcleo del sistema.

Uno de los mandatos menos utilizados suele ser `nice`, que sirve para que nuestros procesos en trasfondo dejen sitio, cuando sea necesario, a otros procesos que consumen más recursos. Éstos últimos usualmente son procesos interactivos y con fuerte componente de entrada/salida.

`nice␣`*`mandato`*	Disminuye la demanda de prioridad para el *`mandato`* indicado.

Por ejemplo, podemos mandar un proceso costoso en tiempo de ejecución, para que se ejecute en *background* y que se descuelgue de la sesión si ésta finaliza en algún momento.

```
nohup␣nice␣proceso-costoso␣&
```

Finalmente hay que hacer hueco a uno de los mandatos más utilizados en el entorno de un estudiante de UNIX: `kill`. Este mandato sirve para enviar «señales» a un proceso. Estas señales disparan en dicho proceso rutinas de tratamiento de señales, y son programables por el desarrollador de aplicaciones. Algunas de ellas tienen un cometido muy concreto, y entre éstas las más utilizadas son:

`SIGHUP`	(1) *Hangup* – por defecto provoca: `exit`. (ver `termio`).
`SIGINT`	(2) *Interrupt* – por defecto provoca: `exit`. (ver `termio`).
`SIGQUIT`	(3) *Quit* – por defecto provoca: `core`. (ver `termio`).
`SIGABRT`	(6) *Abort* – por defecto provoca: `core`.
`SIGKILL`	(9) *Killed* – por defecto provoca: `exit`.
`SIGTERM`	(15) *Terminated* – por defecto provoca: `exit`.
`SIGSTOP`	(23) *Stopped* – por defecto provoca: `stop`.

Aproximadamente desde la señal 1 a la 15 son estándar. Apartír de ahí, dependen del sistema. Para ver una lista completa recomiendo utilizar la página de manual `signal`, tanto en el manual 2, como en el manual `3HEAD` para Solaris o 7 para Linux.

`kill␣-l`	Muestra una lista de señales disponibles.
`kill␣`*`#-PID`*	Envía una señal de terminación `SIGTERM` al proceso identificado con el PID *`#-PID`*.
`kill␣-`*`sig␣#-PID`*	Envía una señal de número *`sig`* al proceso identificado con el PID *`#-PID`*.
`kill␣-`*`sig␣`*`%#`	Igual que el anterior, pero cuando identificamos el proceso mediante el mandato `jobs`.

En algunas versiones de UNIX, el número de señal se precede con la opción `-s`.

Un uso habitual de `kill` es acabar con procesos colgados que aparecen cuando por alguna razón se interrumpen anormalmente sus canales de entrada/salida y no podemos hacerlos terminar de forma natural. Por ejemplo, si se pierde la conexión de una sesión, a menudo el proceso `shell` sigue ejecutándose aunque no podamos interactuar con él. Cuando volvemos a entrar en el sistema, al ejecutar

`ps -fu pedro`	Si somos el usuario `pedro`, miramos los procesos que nos corresponden.

podríamos ver una lista como la siguiente:

```
  UID   PID  PPID  C    STIME TTY      TIME CMD
pedro 18547 18545  0 11:29:12 pts/22   0:00 -bash
pedro 18571 18547  0 11:29:12 pts/22   0:00 vi texto.txt
pedro 18600 18593  0 11:29:12 pts/22   0:00 -bash
```

donde el proceso 18600 corresponde con nuestra sesión actual[3], y sabemos de buena tinta que el resto de los procesos se corresponde con una sesión anterior. Matémoslos:

`kill␣18547`	Terminamos (`SIGTERM`) con el proceso padre de los que están colgados.

Normalmente con ésto conseguimos que este proceso acabe con el resto de sus hijos al enviarles una señal de terminación, pero a veces no es suficiente, e incluso hay procesos un poco rebeldes. Para ellos reservamos la señal más mortífera:

`kill␣-s␣9␣18571`	Enviamos una señal `KILL` al proceso 18571.

El problema es que con esta señal a veces no conseguirmos liberar los recursos empleados por el proceso (buffers de entrada/salida, archivos de bloqueo,...) con lo que debemos realizar una limpieza posterior.

[3] Ejecutamos símplemente `ps`.

Apéndice A

Transferencia de archivos entre máquinas con ftp

«**ftp**» es un protocolo de transferencia de archivos. El mismo nombre se le da al programa cliente que permite intercambiar archivos bajo el control de un usuario.

Se inicia «contra» una máquina que dispone de tal servicio, y se hace de la siguiente forma:

`ftp␣[`*`opciones`*`]␣`*`host`*	lanza un cliente ftp contra un *`host`* concreto

Este nombre de host puede ser remoto o incluso local, lo cual es útil en algunas ocasiones.

Ojo: Cada servidor tiene un límite de usuarios y en él se especifica qué usuarios pueden acceder y a qué archivos.

Otra posibilidad es invocar directamente el mandato

`ftp`	lanza un cliente ftp sin conectar

Tras lo cual tendremos que iniciar la conexión remota por nosotros mismos:

`open␣`*`host`* `user␣`*`usuario`*`␣[`*`passwd`*`]`	intenta conectar con un *`host`* indica el *`usuario`* y el *`password`* de la conexión

En caso de no incluir el password, se nos solicitará desde el servidor.

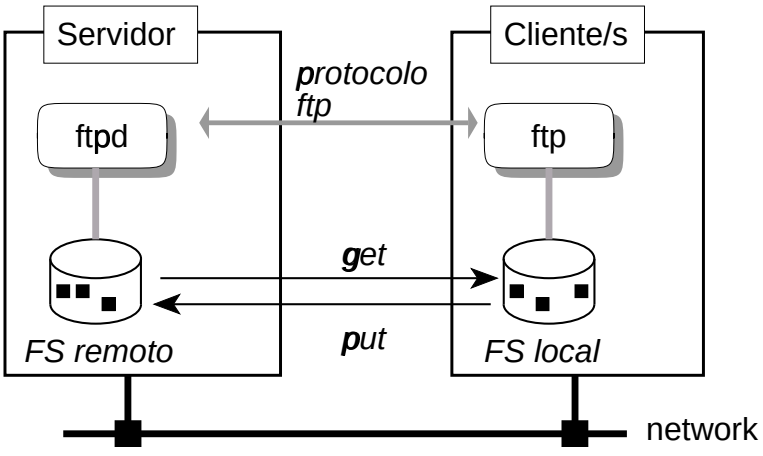

Figura A.1: diagrama cliente servidor con ftp

A.1 Usuarios

ftp solicitará una identificación (nombre de usuario + clave de acceso). Algunos servidores disponen de un usuario especial que permite acceder a archivos de uso público bajo el nombre de usuario: «anonymous», solicitando a continuación la dirección de correo electrónico del que se conecta: En ciertas máquinas no es necesario que este sea una dirección de correo real, pero en otras lo contrastan con direcciones reales.

`ftp ftp.rediris.es`	
`login: anonymous`	ftp.rediris.es admite conexiones anónimas
`(e-mail):`	el password se teclea a ciegas, como por ejemplo: pedro@infor.uva.es

A.2 Algunos mandatos útiles

Una vez se accede al servidor se dispone de un completo juego de mandatos que depende tanto de las funciones operativas del servidor remoto de ftp, como de las funciones que tiene implementadas el cliente ftp que estemos utilizando.

Para conocer las funcionalidades se utiliza:

`help`	muestra las palabras clave disponibles
`help␣`*`mandato`*	ayuda en un *`mandato`* concreto

Una de las funciones primordiales es poder recorrer el árbol de carpetas remoto, para lo cual se utilizan habitualmente los siguientes mandatos

`cd`	para conocer la posición actual
`cd␣`*`carpeta`*	para acceder a una *`carpeta`* concreta (admite ..)
`dir` o `ls`	para conocer los contenidos de la carpeta

`cd␣pub/software`	cambia de carpeta
`dir`	inspecciona la carpeta

Incluso, si se dispone de los permisos adecuados se pueden utilizar los mandatos siguientes:

`mkdir␣`*`nombre`*	crear directorios remotos
`rmdir␣`*`nombre`*	borrado de directorios remotos
`delete␣`*`nombre`*	borrar 1 archivo remoto
`mdelete␣`*`reg-exp`*	borrar varios archivos remotos (m: multiple)
`rename␣`*`n_ant␣n_nuevo`*	cambiar el nombre de un archivo

En el sistema de archivos local también se puede realizar operaciones como

`lcd␣`*`carpeta`*	establece *`carpeta`* en el entorno local

el resto de los mandatos que pueden afectar al sistema de archivos locales, se pueden efectuar mediante la siguiente sintaxis:

`!␣`*`mandato local`*	ejecuta localmente un *`mandato`*

A.3 Transferencia de archivos

El el momento de realizar la transferencia se establece un canal de comunicación (en realidad dos). Este canal puede configurarse para transmitir archivos de texto o archivos binarios.

En el caso de los archivos binarios, éstos se transfieren tal cual byte a byte. Y en el caso de

la transferencia de archivos de texto, y en función de la representación de los archivos de texto en las máquinas local y remota se suele modificar el archivo en la medida correspondiente para que sea legible en el destino.

`binary` o `image`	para transferencias binarias
`ascii` o `text`	para transferencias de texto

La operación de transferencia puede ser de depósito en la máquina remota o de traida de datos. En el primer caso (entre otros mandatos), se utiliza:

`put` *nombre*	deposita el archivo *nombre*
`mput` *reg-exp*	deposita varios archivos concordantes con la expresión regular *reg-exp*

`put readme.txt`	deposita el archivo `readme.txt`
`mput *.c`	deposita los archivos que terminan en `.c`

Para la descarga o traida de archivos a la máquina local se suele utilizar

`get` *nombre*	descarga el archivo *nombre*
`mget` *reg-exp*	descarga los archivos concordantes con la expresión regular *reg-exp*

`get index.html`	descarga el archivo remoto `index.html`
`mget Ab*.gif`	descarga los archivos que comienzan con `Ab` y terminan con `.gif`

A.4 Despedida de la sesión

Es una buena costumbre despedir la conexión con la máquina remota. Para ello se suele utilizar

`bye` o `<^D>`	despedida y cierre del programa cliente
`close`	despedida canal

En este último caso es necesario también cerrar el programa cliente ftp si es que deseamos finalizar las operaciones.

En todo caso, tras un tiempo prolongado (que se establece en la máquina servidora) sin utilizar ningun mandato se produce una desconexión automática del servidor, con lo que se libera el canal para otros posibles clientes.

A.4.1 Ejercicios

- Realice alguna conexión remota anónima con alguna máquina remota como
  - `ftp.rediris.es`
  - `ftp.dante.de`
  - `luna.gui.uva.es`
  - `ftp.uva.es`
- Inspeccione la estructura habitual y vea que algunos de los directorios que se pueden ver no tienen ninguna utilidad concreta. Suelen estar preparados solamente para la descarga de software y datos de utilidad.
- Realice alguna conexión remota en alguna máquina con algun usuario de una máquina, remota o local (por ejemplo: sobre la cuenta del compañero de prácticas):

- `ftp duero.lab.fi.uva.es`
- `ftp l104b1.lab.fi.uva.es`

Puede utilizar esta conexión para compartir datos entre diversas máquinas en las que tenga acceso, e incluso descargar y cargar programas y datos en cuentas de usuario de compañeros suyos.

Apéndice B

Resumen del mandatos de *vi*

Invocación de *vi*	
`vi`	Editar un texto sin nombre archivo asociado
`vi␣`*`archivo`*	Editar *`archivo`* (Nuevo o no)
`vi␣`*`lista de archivos`*	Editar *`lista de archivos`*
`vi␣+`*`n␣archivo`*	Editar *`archivo`* en la línea *`n`*
`vi␣+/`*`txt␣archivo`*	Editar *`archivo`* en la 1ª línea donde aparece *`txt`*

Desplazarse en el documento poco a poco	
← o `h`	un carácter hacia la izquierda
↓ o `j`	un carácter hacia abajo
↑ o `k`	un carácter hacia arriba
→ o `l`	un carácter hacia la derecha

Desplazarse en el documento más rápido	
`0`	Comienzo de línea
`$`	Final de línea
`w`	Comienzo siguiente palabra (*word*)
`b`	Comienzo de la palabra actual (*back*)
`e`	Final de la palabra actual (*end*)
`H`	Esquina superior izquierda de la pantalla (*home*)
`L`	Esquina inferior izquierda de la pantalla (*last*)
`^u`	Subir pantalla (*up*)
`^d`	Bajar pantalla (*down*)
`^b`	Retroceder página (*back*)
`^#b`	Retroceder página # líneas (*back*)
`^f`	Avanzar página (*forward*)
`^#f`	Avanzar página # líneas (*back*)
`#G`	Salta a la #-ésima línea (*go*)
`1G`	Salta a la primera línea
`$G`	Salta a la última línea
`z`	Hacer la línea actual la superior de la pantalla
`zz`	Hace la línea actual la central de la pantalla
`z-`	Hace la línea actual la última de la pantalla
`f`*`car`*	Busca el carácter *`car`* (hacia delante)
`F`*`car`*	Busca el carácter *`car`* (hacia atrás)

Marcas sobre el documento	
m*car*	Marca la línea actual con un *car*ácter (a-z)
'*car*	Mueve a la línea marcada con *car*ácter (a-z)

Insertar texto	
i	Insertar (delante del cursor)
I	Insertar al principio de la línea
a	Añadir (detrás del cursor)
A	Añadir al final de la línea
o	Insertar una línea debajo de la actual (*open*)
O	Insertar una línea encima de la actual (*open*)

Borrar texto	
x	Borrar caracter actual
X	Borrar caracter anterior
dd	Borrar línea actual (*delete*)
D	Borrar hasta final de línea (*delete*)
d*CmdMov*	Borrar hasta la posición indicada por el mandato de movimiento *CmdMov*
dw	Borrar palabra (*delete word*)

Cambiar texto	
r*car*	Reemplazar el caracter actual por *car* (*replace*)
R	Reemplazar texto desde la posición del cursor
s	Substituir el caracter actual por texto a insertar
S	Substituir la línea actual
C	Cambiar hasta el final de la línea
c*CmdMov*	Cambiar hasta la posición indicada por el mandato de movimiento *CmdMov*
cw	Cambiar palabra (*change word*)
J	Unir a la línea actual la siguiente (*join*)

Copiar y pegar	
CmdBorrado	Cortar (El último borrado pasa automáticamente al búfer
yy	Copiar en el búfer la línea actual
#yy	Copiar en el búfer # línea desde la actual
y*CmdMov*	Copiar en el búfer hasta la posición indicada por el mandato de movimiento *CmdMov* (*yank*)
p	Pega el búfer detrás de la posición del cursor
P	Pega el búfer delante de la posición del cursor
"*carCmdBorrado*	Cortar hacia el búfer de nombre *car* (a-z)
"*carCmdCopia*	Copiar hacia el búfer de nombre *car* (a-z)
"*carCmdPegado*	Pegar desde el búfer de nombre *car* (a-z)

Buscar y substituir	
\%	Busca el caracter delimitador ( ) [ ] { } que balancea el actual. Dentro de un entorno salta al delimitador inicial

Buscar y substituir	
/*ExpReg*	Busca hacia delante la expresión regular *ExpReg*
?*ExpReg*	Busca hacia atrás la expresión regular *ExpReg*
n	Repite la última búsqueda (*next*)
N	Repite la última búsqueda en el sentido contrario
:s/*txt*/*txt2*	Substituye el texto *txt* por *txt2* la primera vez que aparece en la línea
:s/*txt*/*txt2*/g	Substituye todas las apariciones de *txt* por *txt2* en la línea (*global*)
:α,βs/*txt*/*txt2*/g	Substituye (idem) en el rango de líneas [$\alpha - \beta$]

Repetir y deshacer	
.	Repetir último mandato de actualización (Borrado/Inserción/Cambio)
u	Deshacer último mandato de actualización
U	Deshacer todos los cambios en la línea actual (ojo, en la línea actual)

Mandatos para el *shell*	
:sh	Invoca un nuevo *shell*. Al salir continua la edición
:!*CmdShell*	Ejecuta el mandato *CmdShell* del *shell*
:r!*CmdShell*	Ejecuta el mandato *CmdShell* del *shell* e inserta su salida en la posición del cursor
!!*CmdShell*	Ejecuta un mandato *CmdShell* del *shell* e inserta su salida en la posición del cursor
:!!	Repite el último comando ejecutado en un *shell*
:α,β!*CmdShell*	Ejecuta un comando del *shell* sobre el rango de líneas [$\alpha-\beta$]

Operaciones con ficheros	
:r *archivo*	Inserta el contenido de *archivo* debajo de la línea actual (*read*)
:w	Graba las modificaciones efectuadas en el archivo actual (*write*)
:w *archivo*	Escribe el texto actual en *archivo* (Sólo si no existiera)
:w! *archivo*	Escribe el texto actual en *archivo*
:α,βw *archivo*	Escribe el rango de líneas [$\alpha - \beta$] en *archivo*
:e *archivo*	Edita un *archivo* alternativo siempre que almacenemos el contenido del archivo actual o no haya cambios
:e! *archivo*	Edita un *archivo* alternativo incondicionalmente
:e #	Conmuta con el archivo alternativo, siempre que almacenemos el contenido del archivo actual o no haya cambios
:e! #	Conmuta con el archivo alternativo incondicionalmente
:n o :next	Editar el siguiente archivo de la lista de archivos que se le han dado al *vi*
:rew o rewind	Volver al primer archivo de la lista
:q	Salir siempre que almacenemos el archivo actual o no haya cambios (*quit*)
:q!	Salir incondicionalmente
:wq o :x o ZZ	Grabar cambios y salir (ZZ: dormir)

Control de comandos	
<ESC>	Finalizar/Anular mandato
^l	Refresca (re-escribe) la pantalla
#Cmd	Repite el mandato *Cmd* #-veces
:α,β*Cmd*	Ejecuta el mandato *Cmd* sobre el rango de líneas [$\alpha - \beta$]. (α y β pueden ser número, mandato de movimiento, marcas, etc, donde $ indica la última línea del archivo y . indica la línea actual -donde está el cursor)
:set *opción*	Activa la opción de *vi* correspondiente
:set *noopción*	Desactiva la opción de *vi* correspondiente
:set all	Muestra todas las opciones y sus valores
:set ai	Modo de autoindentación
:set list	Muestra caracteres de control que acompañan al texto
:set nu[mber]	Muestra numeración de líneas
:set vb	Substituye el pitido por una indicación visual
:set wrap	Mostrar partidas las líneas que no caben en la pantalla
:set ...	Véa usted mismo ;-)

(En la confección de esta página he contado con la colaboración de mi compañero Arturo González Escribano.
De los macros y otras cosas más, ya hablaremos otro día.)

Apéndice C

Exámenes propuestos

En este apéndice se incluyen algunos exámenes anteriores hasta el curso 1999-2000.

1. En un sistema UNIX se encuentran en el directorio `tmp` que cuelga de `HOME` una serie de veinte ficheros de texto cuyos nombres van desde `datos.00.txt` hasta `datos.19.txt`.

   Combinar todos estos ficheros en uno único cuyo nombre sea `datos.txt` siguiendo el orden numérico creciente. El fichero resultante del apartado anterior (`datos.txt`) es un fichero excesivamente voluminoso, y no resulta razonable conservarle tal cual. Por esto se le quiere comprimir, y para ello hay en el directorio `bin` que cuelga del `HOME` (no accesible por la variable `PATH`) un fichero ejecutable llamado `comprimir` que toma la entrada estándar, la comprime y proporciona por la salida estándar el contenido comprimido. Usando esta utilidad de compresión, comprimir el fichero `datos.txt` y generar un fichero llamado `datos.txt.Z`. El fichero `datos.txt.Z` contiene información «sensible» que no conviene que sea accesible por ningún usuario excepto para el propietario (que tendrá control total) y para los demás miembros de su grupo (que sólo podrán ver su contenido). Por esta razón ha de moverse el fichero original a un subdirectorio oculto (el nombre que se le dé es arbitrario) que cuelgue del `HOME`, y han de cambiarse adecuadamente los permisos del fichero y del subdirectorio. También han de borrarse todos los ficheros de datos originales.

2. El usuario `pepe` estaba trabajando en el sistema, pero debido a que ejecutó un comando de manera incorrecta se quedó «colgada» dicha sesión. El usuario `pepe` volvió a entrar y mediante un comando obtuvo entre otras las siguientes líneas:

```
F X UID      PID  PPID C PRI NI   ADDR SZ  WCHAN STIME    TTY   TIME CMD
3 S root       0     0 0 128 20 28df28  0        Jan 1    ?     0:13 swap
1 S root       1     0 0 168 20 604a80 41 7ffe60 Sep 1    ?     0:00 init
1 S javi    7756  7701 0 156 20 777440 18 2dd450 11:16:37 ttyq2 0:00 cat
1 S daemon   144     1 0 154 20 741400 67 74b122 Sep 1    ?     0:00 mail
1 S teresa  2668  2667 0 158 20 750700 45 455dc0 09:34:57 ttyp6 0:00 -sh
1 S pepe    9290  7727 1 154 20 7c4cc0 10 796486 11:17:38 ttypa 0:00 more
1 S pepe    7727  7726 1 158 20 784cc0 55 45b3c0 11:15:58 ttypa 0:00 -sh
1 S pepe    9208  7727 1 154 20 7d3740 27 7bd686 11:17:38 ttypa 0:00 grep
1 S pepe    7681  7680 1 158 20 7c4040 55 45c7c0 11:15:19 ttypd 0:00 -sh
1 S pepe    7721  7681 1 154 20 777540 10 294390 11:15:44 ttypd 0:00 mio
```

   ¿Qué comando permite obtener esta información (incluir las opciones oportunas)? ¿Qué línea de comandos escribirías para eliminar la sesión que se nos quedó colgada?

   En el directorio en el nos encontramos existen varios ficheros que contienen en algún lugar de su nombre la subcadena `log`. Copiarlos a un directorio que cuelga de nuestro `HOME` llamado `tmp`.

En el directorio `bin` (no accesible por la variable `PATH`) que cuelga del HOME está el fichero `a.out`. Se trata de un fichero ejecutable, que al ser invocado pasándole como argumento un fichero llamado `fich1.txt` situado en el directorio padre del actual, produce unas determinadas informaciones de salida. Añadir dicha salida a los contenidos existentes en fichero `salida.log` situado en `tmp` que cuelga del `HOME`. De producirse algún error en la ejecución, depositar los mensajes que lo anuncien en el fichero `errores.log` situado en tmp que el hijo del `HOME`.

3. Puede considerarse que en una máquina UNIX el sistema de correo (`mail`) funciona como sigue: Colgando del directorio raíz hay un directorio llamado `buzon` en el que se encuentran los ficheros que contienen los mensajes recibidos para cada usuario (uno para cada usuario), siendo el nombre de cada fichero igual al nombre del usuario propietario. Supongamos para facilitar las cosas, que todos los usuarios pueden leer/escribir en todos los ficheros que hay en el directorio `buzon`.

   Sabiendo que cada usuario tiene una variable de entorno llamada `LOGNAME` que contiene su nombre de usuario, y suponiendo que no se dispone de ningún comando específico para trabajar con el correo (¡no existe el comando mail!), indicar los comandos UNIX que un usuario teclearía para la realización de las siguientes tareas:

   - Ver el correo recibido por páginas completas.
   - Hacer una copia en el directorio Mail que cuelga del `HOME` y nombre `inbox` de los mensajes recibidos.
   - Eliminar del buzón los mensajes recibidos por el usuario, guardando una copia llamada `inbox` en el subdirectorio `Mail` del directorio `HOME`.
   - Suponiendo que en el directorio actual se encuentra el fichero `carta.txt` que contiene el mensaje que quiere mandar al usuario `pepe` de la misma máquina, enviar dicho mensaje.
   - Borrar los mensajes de la propiedad del usuario que hay en su buzón.

4. **Realizar los apartados siguientes suponiendo que estamos en un lugar desconocido de la estructura de directorios de una máquina UNIX y NO NOS MOVEMOS DE ALLÍ**.

   En un directorio llamado `voz`, hijo del directorio raíz, se encuentran una serie de ficheros de sonido, que se caracterizan porque sus nombres acaban en «`.au`».

   (a) Crear en el directorio `HOME` un directorio llamado `aislados`.

   (b) Copiar en el directorio creado en el apartado anterior, todos los ficheros de sonido del directorio voz.

   (c) En el directorio `aplicaciones`, que es hijo del directorio padre del actual, existe un aplicación llamada `makepree` que filtra ficheros de sonido. Esta aplicación recibe un único argumento, que ha de ser el nombre del fichero o de los ficheros (admite caracteres comodín o metacaracteres) de sonido a filtrar. Por cada fichero de sonido que procesa, `makepree` muestra por la salida estandar un mensaje indicándolo; el mensaje contiene el nombre del fichero que se está procesando. Filtrar todos los ficheros de voz copiados en el apartado 2, sabiendo que el directorio `aplicaciones` NO ESTÁ INCLUIDO EN LA VARIABLE PATH, y guardando los mensajes de salida de la aplicación en un fichero llamado `filtrados`, que ubicaremos en el directorio actual.

   (d) Repetir el apartado anterior, pero ahora sin que la ejecución de `makepree` suponga el bloqueo del terminal.

(e) A partir de la información guardada en el fichero `filtrados`, creado en el apartado dos puntos más atrás, indicar el número de ficheros de sonido, cuyo nombre contenga la cadena «`hablante1`», que han sido filtrados.

5. **Realizar los apartados siguientes suponiendo que estamos en un lugar desconocido de la estructura de directorios de una máquina UNIX y que NO NOS MOVEMOS DE ALLÍ.**

   NOTA: No se permite utilizar el ';' (punto y coma) para separar órdenes en una misma línea.

   En un directorio llamado `tmp` que cuelga del raíz se encuentran una serie de ficheros cuyas extensiones son `.doc` y `.exe`

   (a) Crear con una sola línea de comandos dos directorios que cuelguen del `HOME` llamados documentos y programas.

   (b) Mover los ficheros cuya extensión sea `.doc` a documentos y los que acaben en `.exe` a programas.

   (c) Crear un fichero llamado `busqueda.tmp` en el padre del actual, que contenga todas las líneas de los ficheros `.doc` donde aparezca la palabra «Z80» ordenadas alfabéticamente. (Se debe realizar con una sola línea de comandos).

   (d) Entre los ficheros ejecutables (los que tienen extensión `.exe`) se encuentra uno llamado `conversor.exe`, no accesible por la variable `PATH`, que transforma los ficheros con extensión `.doc` en otros con el mismo nombre pero con extensión `.rtf`. A este ejecutable se le pasan los `.doc` como argumento (admite metacaracteres) y deja los ficheros convertidos en el directorio actual. Queremos convertir todos los ficheros `.doc` que tengan en su nombre la cadena parte, conservando los mensajes de error en caso de que se produzcan, y dejando el terminar libre para continuar con otras tareas mientras se realiza ésta.

6. **Realizar los apartados siguientes suponiendo que estamos en un lugar desconocido de la estructura de directorios de una máquina UNIX y que NO NOS MOVEMOS DE ALLÍ. NOTA: No se permite utilizar el `;` (punto y coma) para separar órdenes en una misma línea.**

   En un directorio llamado `pub` que cuelga del raíz se encuentran dos ficheros: el `aspirin.tar` y el `ejemplos.tar`.

   (a) Crear con una sola línea de comandos dos directorios que cuelguen del padre del actual llamados `herramientas` y `demos`.

   (b) Copiar el fichero `aspirin.tar` en el directorio herramientas que acabas de crear y el fichero `ejemplos` en el `demos`.

   (c) Tanto el fichero `aspirin.tar` como el `ejemplos.tar`, son ficheros especiales ya que su contenido es un conjunto de ficheros. Para extraer estos ficheros existe una aplicación llamada `tar` ubicada en el directorio aplicaciones que cuelga del `HOME`. Este directorio aplicaciones no es accesible por la variable `PATH`. La aplicación `tar` funciona de la siguiente manera: extrae los ficheros contenidos en el fichero `.tar` pasado como argumento, dejando éstos en el directorio actual. Los mensajes de error van a la salida estandar.

   Extraer los ficheros contenidos en `aspirin.tar` y en `ejemplos.tar`, guardando los mensajes de error que se produzcan en el fichero errores que se ubicará en un directorio `tmp` que cuelga de `HOME`.

   (d) Almacenar en un fichero llamado `lineas` que se ubicará en el directorio actual, todas la líneas que contengan la cadena de caracteres «`red neuronal`», de todos los

ficheros del directorio `demos` (directorio creado en el apartado primero) cuyo nombre empiece por los caracteres «`ejemplo`». El contenido del fichero `lineas` debe estar ordenado. Realizar este apartado con una sola línea de comandos.

(e) Indicar qué comando se ha ejecutado si la respuesta del sistema es:

```
 UID    PID      PPID    C     STIME     TTY     TIME   CMD
cevp   28134    28132    0   16:58:36   pts/80   0:00   -ksh
cevp   28377    28134    0   17:01:08   pts/80   0:00   mail usuario
cevp   28382    28134    0   17:01:12   pts/80   0:00   vi fichero.txt
```

El proceso `vi` se nos ha quedado «colgado», eliminarlo. ¿Qué comando, distinto de `exit`, ejecutarías para cerrar la sesión