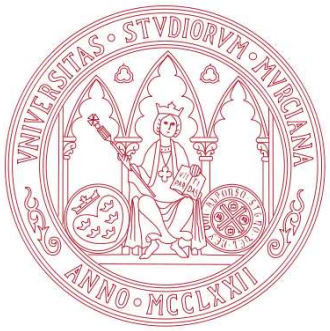

Universidad de Murcia

Facultad de Informática

TÍTULO DE GRADO EN
INGENIERÍA INFORMÁTICA

Fundamentos de Computadores

Tema 4: Introducción a los Sistemas Operativos

Boletines de prácticas

CURSO 2020 / 21

Departamento de Ingeniería y Tecnología de Computadores

Área de Arquitectura y Tecnología de Computadores

Índice general

Boletines de prácticas

B4.1. Guía de introducción a la línea de comandos en Linux

B4.1.1. Introducción

La consola de texto, el terminal, el intérprete o línea de comandos, el *shell* o simplemente el `bash` (nombre con el se conoce a la aplicación propia de Linux) es una ventana de texto en la que, desde el teclado y línea a línea, podemos introducir órdenes que serán analizadas sintácticamente y ejecutadas por el sistema operativo (SO). El intérprete de órdenes es sin duda la forma más básica y directa de interactuar con el sistema operativo. Las órdenes introducidas pueden ser muy sencillas, como por ejemplo para obtener un listado de los ficheros existentes en un directorio, o bien órdenes mucho más complicadas que permitan hacer cosas mucho más complicadas o tediosas. Las órdenes introducidas pueden ser *órdenes internas* u *órdenes o programas externos*. Las órdenes internas son órdenes propias del intérprete de comandos o *shell*, por tanto ya están residentes en memoria. El intérprete de comandos posee una tabla gracias a la cual reconoce un conjunto de órdenes internas. Las órdenes o programas externos son ficheros ejecutables ya instalados en el sistema operativo. El intérprete de comandos buscará estos ficheros ejecutables en ciertos directorios por defecto. Destacar que cualquier aplicación instalada en el sistema se podrá ejecutar mediante la línea de comandos.

Los intérpretes de comandos (y de manera muy especial el `bash` de Linux) suelen permitir configurar, personalizar y monitorizar el sistema de una manera muy avanzada, así como realizar tareas repetitivas y tediosas con el empleo de los llamados *guiones shell* o *scripts*. Los *guiones shell* son órdenes que se podrían teclear secuencialmente pero que, en lugar de ello, se agrupan dentro de un fichero para posteriormente ejecutarlas cuantas veces queramos como una sola orden, correspondiente al propio nombre del fichero de texto una vez creado. En un *guión shell* se usan variables y estructuras de control (*if*/*while*/etc.), permitiendo que convertirlos en "pequeños" programas que son ampliamente utilizados en el mundo de Unix/Linux.

Es por ello que, a pesar de que introducir órdenes manualmente desde el teclado pueda parecer dificultoso y/o intimidante, el usuario medio de Linux antes o después trabajará habitualmente con una ventana de la consola abierta. Las ventajas de usar el `bash` son, por lo tanto, muchas y merece la pena el esfuerzo inicial que supone su uso. Entre ellas:

- Algunas aplicaciones se lanzan exclusivamente o más cómodamente desde la línea de comandos.
- La labor reiterada de producir distintas salidas de un proceso se puede automatizar con el uso de un *script*.
- Instalar paquetes de programas.
- Hacer copias de seguridad o llevar un repositorio de archivos.
- Administrar y gestionar usuarios y grupos, y sus correspondientes permisos en ficheros y directorios.
- Personalizar las órdenes del SO (añadiendo más, modificando sus aspectos y funcionalidades).
- Velar por el buen estado del sistema (*backups*, agregar y quitar usuarios, instalar, quitar, actualizar y configurar aplicaciones, etc.).
- El control de los procesos que se empeñan en dejar el sistema bloqueado hace al uso del intérprete de comandos prácticamente imprescindible.
- Modificar órdenes con sintaxis alternativa.
- Realizar tareas especificas.
- Etc.

B4.1.2. Recomendaciones iniciales

¿De dónde venimos?

¿Hemos usado anteriormente el intérprete de comandos `cmd` de Windows? ¿No? Esta es una serie de normas iniciales para los usuarios noveles en Linux:

- Respetar siempre los espacios en blanco: El `bash` tiene una gramática tan potente que su principal forma de interpretar dónde empiezan y acaban los elementos de una orden es usar como separación el espacio en blanco. Es una costumbre habitual entre los usuarios de Linux no usar espacios en blanco en el nombre de los ficheros, no porque no esté permitido sino por comodidad, pues casi todas las aplicaciones nos obligarían a anteponer la barra invertida `\` ante cada espacio en blanco que contenga el nombre (por ejemplo, como en la línea de comandos `gedit mi\ archivo.txt`), con el fin de no confundir el espacio del nombre del fichero con el separador habitual del `bash`.
- Mejor no usar tampoco en el nombre de fichero los caracteres `/|\!?*<>&~()[];#`: Se trata de caracteres especiales usados en la gramática `bash`. Aunque están permitidos como parte del nombre de ficheros, su posible uso en éstos nos obligaría a estar utilizando continuamente secuencias de escape como la anterior (`\`), que son a la vez engorrosas y en general innecesarias, salvo en casos muy concretos que no cubriremos en este primer curso.
- `A` $\neq$ `a`: Escribir diferenciando mayúsculas de minúsculas. Para Linux `Programa.c` y `programa.c` son dos ficheros totalmente distintos.
- Para Linux todo el conjunto de dispositivos de almacenamiento es considerado como un sistema único de ficheros jerárquico que parte desde el directorio raíz (`/`). No existen, por tanto, las unidades lógicas (`A:`, `B:`, `C:`, etc.) que usa Windows.
- Usar el separador de rutas a directorios y archivos `/` (y no `\`) : A la hora de escribir la ruta de acceso a nuestros ficheros o directorios se usa la barra inclinada normal (al igual que las URLs en Internet como separación de directorios). Por ejemplo, `/home/pepe/archivo.txt`.
- El carácter `.` (punto) se puede usar tantas veces como queramos en el nombre de ficheros y directorios, y su uso suele ser por comodidad, para agrupar varios ficheros que serán abiertos con la misma aplicación, diferenciándolos por *extensiones*, como en los ejemplos `fich-binario.bin`, `fich-texto.txt` o incluso `fich-comprimido.tar.gz`. También es perfectamente válido no incluir ningún punto en el nombre de un archivo o directorio, por ejemplo, `nombre_archivo` o `directorio`.
- `.nombre_fichero_o_directorio`: El punto antepuesto al nombre de un fichero o directorio[1] se usa para nombrar entradas *ocultas* (entradas que no aparecerían en un listado normal, a menos que usemos la opción `-a` del comando `ls`).

¿Y si no me gusta teclear?

El `bash` de Linux aporta unas ayudas para hacer un uso rápido del intérprete de líneas:

- Tecla del tabulador `Tab` : Completa la ruta cuando hemos escrito suficientes caracteres de ésta como para poder distinguir una ruta de otra que comiencen por las mismas letras. Presionando dos veces seguidas, muestra las posibles rutas cuyo nombre empiecen por los caracteres que ya hemos tecleado. También es aplicable a las órdenes.

[1] Cuando hablemos de archivos o directorios indistintamente, nos referiremos a ellos con el término genérico de *entradas*.

- Tecla arriba ↑ o tecla abajo ↓ : Nos permite navegar por las órdenes que ya hayamos introducido anteriormente y que guarda en un historial. También podemos acceder a éste con `CTRL+R` y tecleando una subcadena de una orden anterior.
- Comando interno `history` : Orden para conocer el historial de órdenes introducidas anteriormente. Tened en cuenta que guarda tanto las órdenes válidas como las órdenes que eran incorrectas y produjeron un error. Delante de cada una de ellas indica un número que establece el orden de ejecución.
- `!`*número_de_orden* : para ejecutar una orden en concreto después de consultar el historial de órdenes con el comando del punto anterior. El *número_de_orden* es el número mostrado por `history` delante de cada comando ejecutado.
- Arrastrar ficheros o directorios desde el navegador de ficheros: Si tenemos abierta una ventana del navegador de ficheros, basta que, con el ratón, arrastremos una entrada (directorio o fichero) dentro del terminal para que aparezca su ruta escrita.
- `Ctrl+Mayúsculas+C`, `Ctrl+Mayúsculas+V` : Copia (`Ctrl+Mayúsculas+C`) lo que esté remarcado en el terminal y pega (`Ctrl+Mayúsculas+V`) en el terminal lo que hayamos copiado al portapapeles (tanto desde el terminal como desde cualquier otro sitio con `Ctrl+C`).

¿Me lo tengo que aprender todo de memoria?

Cuanto más se usa el terminal más diestro se es, pero al principio es muy recomendable consultar continuamente el manual y experimentar con órdenes y sus opciones para familiarizarnos. Para ello, se pueden usar los siguientes comandos u opciones:

- `man` *orden* (o, alternativamente, `info` *orden*): Muestra un manual de la orden tecleada (sobre todo órdenes externas), del que se sale pulsando la tecla `q`.
- `help` *orden* (comando interno): Manual más somero, utilizado sobre todo para obtener ayuda sobre órdenes internas.
- Opciones de ayuda para cada orden concreta: Habitualmente siguen la sintaxis *orden* `-h`, *orden* `-?`, *orden* `-h` u *orden* `-help`.
- Consultar en Internet mediante Google y otro buscador.

¿Cómo abrir un intérprete de órdenes en terminales de texto?

Para abrir un intérprete de órdenes tenemos varias opciones:

- De manera directa, si el arranque del sistema operativo se produce en modo texto.
- En caso de que el sistema operativo se haya iniciado en modo gráfico, podemos acceder a un terminal de texto mediante la pulsación combinada de las teclas `CTRL + ALT + Fi`, para i=3, 4, . . . , pudiendo volver al terminal gráfico inicial mediante `CTRL + ALT + F1` o `CTRL + ALT + F2`, dependiendo del sistema.
- En caso de que el sistema operativo se haya iniciado en modo gráfico, otra opción de comunicación con el shell sería mediante la ejecución de un programa de emulación de terminal. Estos programas permiten a los usuarios ejecutar órdenes usando una interfaz de línea de órdenes dentro de un entorno de ventanas. Los programas de emulación de terminal más conocidos y usados son `konsole`, desarrollado para el proyecto de escritorio libre KDE, y `gnome-terminal`, que es parte del proyecto de escritorio libre GNOME.

El *prompt* es una indicación que muestra el intérprete para anunciar que espera una orden del usuario para ejecutarla a continuación. Tal como hemos dicho previamente, esta orden puede ser interna o externa. Un ejemplo de una orden interna (aquellas que vienen incorporadas en el propio intérprete) es `cd`. Las externas (programas separados) son, por ejemplo, todos los programas que residen en los directorios `/usr/bin` (como `ls`) o `/usr/sbin` (como `mkfs`). El intérprete Bash suele mostrar un prompt como el siguiente:

```
[usuario@nombre_máquina directorio_actual]$
```

donde `usuario` es el nombre del usuario conectado al sistema que hace uso de Bash, `nombre_máquina` es el nombre del computador que se está usando y `directorio_actual` es el directorio en el que se está trabajando y será donde se ejecuten las órdenes.

B4.1.3. Movimiento básico por el sistema: usuarios, grupos, directorios

¿Quiénes somos?

Linux (al igual que su predecesor Unix) organiza a los usuarios dentro de grupos que disponen de los mismos privilegios para poder acceder a ficheros y servicios del sistema. Linux es un SO multiusuario, lo que significa que más de un usuario puede trabajar en el sistema de forma simultánea con otros, ejecutando una o más tareas a la vez. Para que múltiples usuarios puedan hacer uso del sistema de una forma segura y ordenada es necesario que el sistema disponga de mecanismos de administración y seguridad y pueda proteger los datos de cada usuario, así como proteger y asegurar el correcto funcionamiento del propio sistema. Linux permite que cada usuario pueda hacer uso del espacio en disco, organizándolo ese espacio en subdirectorios con permisos y privilegios propios de cada usuario. En concreto, en el directorio `/home/`*`nombre_de_usuario`*, que comúnmente se llama directorio de inicio, de trabajo o directorio *home*.

Existen 3 tipos de usuarios:

1. Usuarios normales: Son los individuos particulares que pueden entrar en el sistema, con más o menos privilegios, y que harán uso de los recursos del sistema. Como indicador el *prompt* utiliza para ellos por defecto el símbolo `$` (dólar). Ejemplos: `alumno`, `jaime`, `juan`, `agarcia`, etc. También se les conoce con el nombre de *usuarios de login*.

2. Usuarios del sistema: Son usuarios especiales vinculados a ciertas tareas que debe realizar el sistema operativo. No están vinculados a personas, ya que este tipo de usuarios no pueden ingresar al sistema con un login normal. Ejemplos: `mail`, `ftp`, `bin`, `sys`, etc. También se les conoce como *usuarios sin login*.

3. Usuario `root`, o superusuario: Todo sistema Linux cuenta con un superusuario, que tiene los máximos privilegios. Dichos privilegios le permitirán efectuar cualquier operación sobre el sistema (incluida su destrucción). Su existencia es imprescindible ya que se encarga de gestionar los servicios de red, grupos, usuarios, instalación de software, etc. Debido a su omnipotencia, al entrar como superusuario el *prompt* nos avisa de ello utilizando el signo `#`.

Para poder administrar los permisos de los usuarios de una forma más flexible, Linux permite la organización de usuarios en grupos y establecer permisos a los grupos. Los grupos, por tanto, permiten otorgar una serie de privilegios a un conjunto de usuarios sin tener que dárselos de forma individual a cada uno de ellos. Un usuario puede pertenecer a varios grupos, y el superusuario es el responsable de organizar la distribución de usuarios por grupos.

A continuación se expone la lista de órdenes más habituales relacionadas con lo visto en este apartado:

- `whoami` : Muestra nuestro nombre de usuario.
- `who` : Muestra los usuarios actualmente conectados al sistema, con su hora de inicio de sesión, y la máquina desde la que accedió al equipo.

- `w` : Similar al anterior, pero además muestra qué está ejecutando en este momento cada usuario.
- `groups` : Indica a qué grupos pertenecemos.
- `su` : Cambia de usuario normal a superusuario (o `root`). El prompt acaba con el carácter `#` en lugar del carácter `$` para advertírnoslo. Si se especifica como parámetro un usuario, se cambia a ese usuario. En cualquier caso, hay que indicar la contraseña del usuario al que se desea cambiar.
- `sudo` *`orden`* : Ejecuta el comando *`orden`* como superusuario, aún estando conectados como usuario normal (por supuesto, y al igual que el comando anterior, exige contraseña).
- `login` *`usuario`* : Cambia al usuario especificado, pidiendo la contraseña correspondiente.
- `exit` : Termina la sesión del usuario actual.

¿Dónde estamos?

A pesar de que el *prompt* nos indica en todo momento el directorio donde estamos situados (el carácter ~ indica que estamos en el directorio inicial o de trabajo), podemos siempre obtener la ruta completa del mismo con la orden:

- `pwd` : muestra la ruta completa del directorio actual.

Linux distribuye sus ficheros y directorios a partir de su directorio raíz (/) con una estructura predeterminada que merece la pena recordar y respetar para encontrar aplicaciones, librerías, archivos de configuración, datos de usuario y demás tipos de archivos con rapidez y eficacia. Dicha jerarquía se esquematiza en la figura **??**.

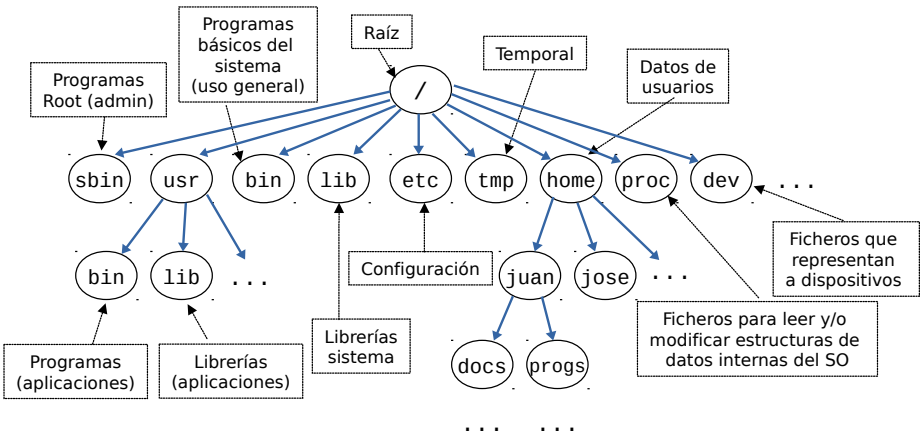

Figura I.1: Jerarquía de directorios en Linux.

¿A dónde vamos?

Una de las operaciones primordiales al utilizar el terminal es indicar en las órdenes los directorios o los ficheros a los que se haga referencia. Esta ruta se puede indicar por medio de su ruta absoluta (cualquiera de las que parten del directorio raíz) o por medio de una ruta relativa (cualquiera de las que parten del directorio actual). La ruta absoluta debe indicarse comenzando siempre con la barra inclinada (carácter `/`), mientras que la relativa puede indicarse bien comenzando con el punto y la barra (caracteres `./`) si la ruta empieza desde el directorio actual, o bien con los dos puntos y la barra (caracteres `../`) si vamos a empezar la ruta subiendo al directorio padre. He aquí algunos ejemplos (ver también figura **??**):

1. Ejemplo de ruta absoluta al directorio `/usr/lib`:
   `/usr/lib/`
2. Ejemplo de ruta absoluta al fichero `fich.txt` en el directorio `docs` bajo el directorio del usuario `juan`:
   `/home/juan/docs/fich.txt`
3. Ejemplo de ruta relativa al fichero `fich.txt` desde el directorio `progs`:
   `../docs/fich.txt`
4. Ejemplo de ruta relativa al directorio `progs` desde el directorio `juan`:
   `./progs/`

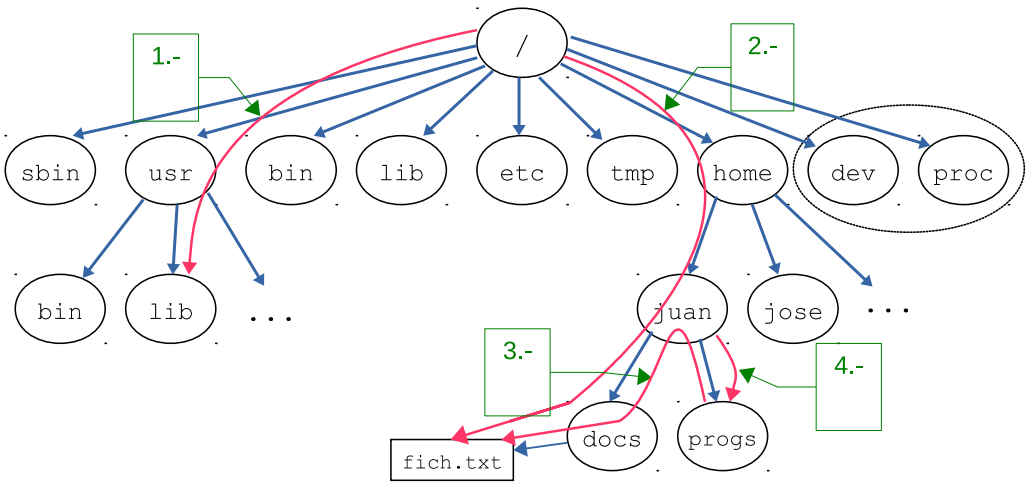

Figura I.2: Ejemplos de rutas absolutas y relativas (ver texto).

Órdenes relacionadas:

- `cd ruta_directorio` : Nos mueve de nuestro directorio actual al indicado con la ruta (relativa o absoluta).
- `cd` (sin parámetros), o bien `cd ~` : Nos mueve al directorio de trabajo (el de inicio, `/home/usuario`).
- `cd -` : Nos mueve al directorio último en el que habíamos estado con anterioridad al actual.

Por ejemplo, supongamos que estamos en `docs` y queremos ir a `jose`. Podremos hacerlo usando una ruta relativa o una absoluta:

- Usando una ruta relativa: `juan@casa:~/docs$ cd ../../jose`
- Usando una ruta absoluta: `juan@casa:~/docs$ cd /home/jose`

B4.1.4. Comandos sobre el sistema de archivos

¿Qué hay por aquí?

Aunque las distribuciones de Linux poseen navegadores de ficheros que permiten inspeccionar los directorios e incluso copiar arrastrando con el ratón los ficheros de una ventana a otra del navegador, el terminal de Linux nos ofrece muchas más posibilidades de listado y copia de ficheros de un modo más personalizado. Para obtener un listado de ficheros se dispone de la orden `ls`, cuyos usos más habituales resumimos aquí, con los parámetros correspondientes:

- `ls` : Lista en formato corto las entradas del directorio actual.
- `ls` *`ruta_a_un_directorio`* : Lista en formato corto las entradas del directorio indicado.
- `ls -l` : Lista en formato largo (ver figura **??**). Cada línea se corresponde con una entrada. Si la línea empieza por `d`, se trata de un directorio, si empieza por un guión (`-`) es un fichero[2]; continúa con los permisos; el número de enlaces físicos que tiene esa entrada; el usuario que creó el fichero; a qué grupo pertenece; tamaño (normalmente bytes) del fichero o subdirectorio; su fecha y hora de última modificación y su nombre. Volveremos sobre los permisos en apartados posteriores de esta guía.
- `ls -R` : Lista recursivamente todas las entradas (ficheros o subdirectorios) que cuelguen de él y de cualquiera sus subdirectorios, a cualquier profundidad.
- `ls -a` : Lista también los ficheros y directorios ocultos (es decir, cuyos nombres comiencen por el carácter punto `.`, por ejemplo, `.hiddenfile` o `.hiddendir`).
- `ls -S` : Lista en orden de tamaño.
- `ls -t` : Lista en orden de fecha y hora de la última modificación.
- `ls -u` : Lista en orden de fecha y hora del último acceso.
- `ls -U` : Lista en orden de fecha y hora de su creación.
- `ls -r[S|t|u|U]` : Invierte el orden del listado (según criterio de ordenación elegido)[3].
- `ls -d` *`subdirectorio`* : Lista la entrada correspondiente al subdirectorio indicado (en lugar de las entradas contenidas en dicho subdirectorio, que sería el comportamiento por defecto para el listado de directorios).

Uso de comodines

Si a un comando `ls` le pasamos como parámetro una ruta (absoluta o relativa) indicando un sólo nombre de entrada (fichero o directorio), el listado tomará en cuenta sólo esa entrada. También pueden pasarse como parámetros varias entradas en secuencia (como en el comando `ls -l fich1.txt fich2.doc ../fich3.avi`, por ejemplo). Sin embargo, podemos combinar comodines para obtener listados múltiples acotados, sin necesidad de teclear una a una todas las entradas deseadas. Los comodines nos ayudarán a ello, puesto que sirven para especificar varios nombres de ficheros de una sola vez.

Los comodines pueden usarse en cualquier lugar de la línea de comandos, aunque lo más habitual es emplearlos en los listados a realizar con el comando `ls` visto en el apartado anterior (como en el comando `ls -l *.c *.h`, por ejemplo), o bien con comandos para copiar, mover o borrar archivos, que veremos un poco más adelante.

El repertorio completo de comodines que utilizaremos es el siguiente:

- `*` : Significa cero o más caracteres cualesquiera.
- `?` : Un carácter cualquiera.
- `[a-z]` : Cualquier carácter del rango indicado.

[2] Hay más topo de ficheros y por tanto, más «letras» identificativas, pero quedan fuera el alcance de este curso. Los veréis todos en la asignatura de Sistemas Operativos de segundo curso

[3] En general, todos los comandos que aceptan varias opciones simultáneamente (la inmensa mayoría) permiten dos sintaxis distintas para ello: la primera es ir poniendo las opciones una a una, precedidas por su correspondiente guión (como en `ls -l -a -S -r`, por ejemplo; la segunda, juntar en una sola cadena todas las opciones, precedidas por un sólo guión (como en `ls -laSr`, comando que sería completamente equivalente al anterior).

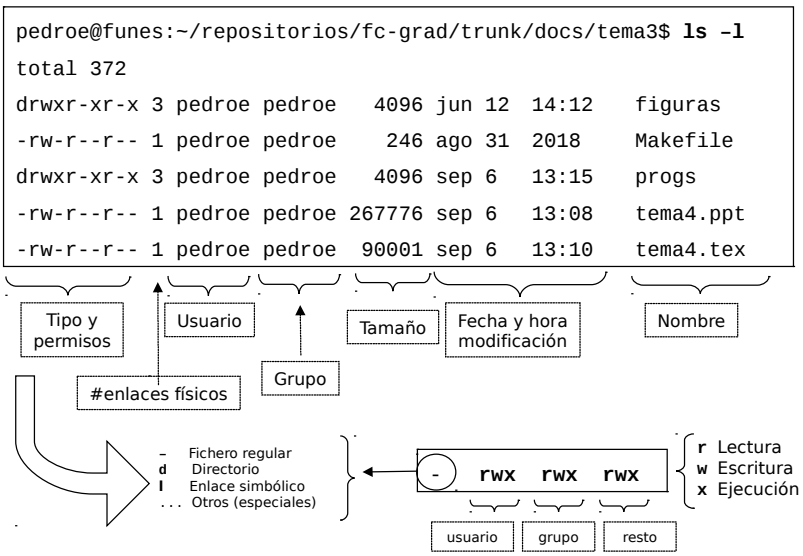

Figura I.3: Formato mostrado por el comando de listado largo `ls -l`.

- `[!b-d]` : Cualquier carácter que no esté en el rango indicado.
- `{nom1,nom2,...}` : Cualquier secuencia de caracteres o nombres de la lista.

Algunos ejemplos:

- `*.{c,h}` : Recogería nombres como `pepe.c`, `f1.c`, `hola.h`, `antonio.h`, ...
- `{fi1,fi2}.[a-c]` : Recogería los nombres `fi1.a`, `fi2.a`, `fi1.b`, `fi2.b`, `fi1.c`, `fi2.c`.
- `*.?` : Recogería todos los ficheros con extensión de un carácter, como `f1.a`, `hola.c`, `adios.txt.c`, ...
- `*.[!c]` : Recogería todos los ficheros con extensión de un carácter que no sea `c`, como `pepe.h`, `f1.a`, o `f2.z`, pero no `fichero`, `hola.doc` o `antonio.c`.

Naturalmente, los comodines pueden también formar parte de una ruta arbitrariamente larga, como en el comando `ls -l ../../dir/p*.txt`, por ejemplo.

¿Qué hay dentro?

El terminal nos permite también mostrar el contenido de los ficheros de texto sin necesidad de abrir un editor. Los comandos más útiles para ello son:

- `cat` *`fichero`* : Muestra el contenido del fichero, de golpe, en el terminal.
- `more` *`fichero`* : Muestra el contenido del fichero, con posibilidad de mostrarlo poco a poco en el terminal. Para terminar de visualizarlo hay que pulsar la letra q.
- `less` *`fichero`* : Muestra el contenido del fichero, con posibilidad de *scroll* bidireccional en el terminal, y búsqueda de palabras (con `/palabra`). Para terminar de visualizarlo hay que pulsar la tecla q.
- `hexdump -C` *`archivo`* : Vuelca el archivo en formato hexadecimal en el terminal. Éste comando es interesante porque permite también inspeccionar archivos binarios genéricos (no necesariamente de texto), por el terminal, muy al estilo del programa *okteta*.

Crear, copiar, mover y borrar ficheros y directorios

A continuación se muestran los comandos más comunes para la creación (`touch`), copiado (`cp`), traslado (`mv`) y borrado (`rm`) de ficheros y directorios:

- `touch nombre_fichero` : Crea un nuevo fichero llamado *nombre_fichero*, vacío, o si ya existía le cambia la fecha y hora de modificación a la actual sin modificar su contenido.
- `cp fich_origen_1 [fich_origen_2 ...] fich_destino|dir_destino` : Copia un fichero a otro (o varios ficheros a un directorio).

  Las opciones más interesantes para el comando `cp` son las siguientes:

  - `cp -R ...` : Copia desde el(los) origen(orígenes) recursivamente. Usado para copiar directorios y sus contenidos.
  - `cp -i ...` : Pide confirmación si el fichero destino ya existía (muy útil para no machacar posibles ficheros destino ya existentes sin darnos cuenta).
  - `cp -p ...` : Realiza la copia conservando la fecha de modificación de los archivos y directorios originales, así como los permisos, el propietario y el grupo propietario.
  - `cp -u ...` : No realizará la copia si el destino tiene una fecha de modificación posterior o igual a la de origen.

  He aquí varios ejemplos de utilización del comando `cp`, incluyendo algunos casos de uso de comodines:

  - `cp fichero_origen fichero_destino` : Copia un fichero a otro fichero del mismo directorio (con otro nombre distinto, obviamente).
  - `cp fichero_origen directorio_destino` : Copia un fichero a otro directorio (que ha de existir previamente).
  - `cp fichero_origen1 fichero_origen2 fichero_origen3 directorio_destino` : Copia varios ficheros a la vez sobre el directorio destino.
  - `cp -R directorio_origen directorio_destino` : Copia un directorio completo (incluyendo recursivamente todo lo que cuelga del directorio de origen).
  - `jaime@casa:~/juan/progs$ cp ../docs/fich.txt .` : Copia el fichero `fich.txt` desde `/home/jaime/juan/docs` a mi directorio actual (en este caso, y para todos los ejemplos que siguen, el directorio `/home/jaime/juan/progs/`).
  - `jaime@casa:~/juan/progs$ cp -i fich.txt ../docs` : Copia el fichero `fich.txt` (que acabamos de copiar en el directorio actual) a `/home/jaime/juan/docs`, pidiendo confirmación pues ya existe.
  - `jaime@casa:~/juan/progs$ cp ../docs/f1.c ../docs/f2.c ../docs/f3.c .` : Copia los ficheros `f1.c`, `f2.c` y `f3.c` desde `/home/jaime/juan/docs` a mi directorio actual.
  - `jaime@casa:~/juan/progs$ cp ../docs/F* .` : Copia todos ficheros cuyo nombre empiece por `F` del directorio `/home/jaime/juan/docs` a mi directorio.
  - `jaime@casa:~/juan/progs$ cp ../docs/Fich? .` : Copia todos ficheros cuyo nombre sea `Fich` seguido exactamente por un carácter cualquiera del directorio `/home/jaime/juan/docs` a mi directorio actual.
  - `jaime@casa:~/juan/progs$ cp -R ../docs ./copia_de_docs` : Copia recursivamente el directorio `/home/jaime/juan/docs` completo (con todas las entradas en él contenidas) en el nuevo directorio `/home/jaime/juan/progs/copia_de_docs`.

- `mv fichero_origen_1|dir_origen_1 [...] fichero_destino|dir_destino` : El comando `mv` es similar al comando `cp`, pero borra el(los) fichero(s) y/o directorio(s) de origen tras copiarlos. Podemos también usarlo para cambiar el nombre de un fichero o subdirectorio si tanto el origen como el destino están ubicados en el mismo directorio. Un posible ejemplo de uso de este comando para mover un subdirectorio completo a otro lugar utilizando rutas relativas sería el siguiente:

  `jaime@casa:~/juan/progs$ mv ../docs ../../jose` : Pasa el directorio `docs` del directorio `/home/jaime/juan` al directorio `/home/jaime/jose`.

  Mientras que un ejemplo para renombrar un fichero sería el siguiente:

  `jaime@casa:~/juan/progs$ mv salida.txt datos.txt` : Le cambia el nombre al fichero `salida.txt` que pasa a llamarse `datos.txt`. El «nuevo» fichero sigue estando en el mismo directorio.

- `rm fichero_1|directorio_1 ...` : El comando `rm` sirve para borrar ficheros o directorios. Las opciones más interesantes del mismo son las siguientes:

  - `rm -v ...` : Muestra lo que se va borrando.
  - `rm -i ...` : Pide confirmación según va borrando.
  - `rm -rf ...` : Borra de forma recursiva (`-r`), y sin confirmación (`-f`), los argumentos que se le pasan como parámetros (usualmente directorios). ¡Usar con mucho cuidado!

En relación a la creación, borrado y copia de subdirectorios, los comandos más relevantes son:

- `mkdir directorio` : Crea un directorio vacío en la ruta designada o en el directorio actual.
- `rmdir directorio` : Borra el directorio indicado, siempre que esté vacío. Si no, hay que borrar primero todas sus entradas, o bien simplemente usar el comando `rm` con la opción `-r`, como se indicó anteriormente.
- `cp -r directorio_origen directorio_destino` : Copia el directorio origen en el directorio destino. También puede hacerse usando el comodín `*` dentro del directorio origen, pero para ello el directorio destino debe haber sido creado previamente.

Compresión y descompresión de ficheros

`tar` es el comando tradicionalmente usado para empaquetar (y desempaquetar) varios ficheros en uno sólo. La herramienta `tar` por sí sola no comprime el fichero resultante, para comprimir el fichero resultante (y reducir su tamaño) hay que usar otras herramientas. `Gzip` es la herramienta que más se suele usar en Linux, y que se llama pasando un simple parámetro a la orden `tar`. Los ficheros empaquetados con `tar` suelen tener la "extensión" `.tar`, mientras que los ficheros comprimidos usando `Gzip` suelen tener extensión `.tar.gz` o, más abreviadamente, `.tgz`[4]. Éstas son las tres formas más comunes de uso de este comando, correspondientes a la *compresión*, el *listado* y la *descompresión*:

- `tar czvf result.tgz directorio[s]|fichero[s]` : Empaqueta y comprime los directorios (recursivamente) y ficheros indicados en un solo fichero `.tgz`. Se guardan rutas relativas y permisos.
- `tar tzvf result.tgz` : Lista en el terminal los contenidos del fichero `.tgz`.
- `tar xzvf result.tgz` : Descomprime y desempaqueta el fichero `.tgz`, recuperando los ficheros, permisos y la estructura de directorios original.

[4] Naturalmente existen también utilidades Linux para tratar otros tipos de ficheros comprimidos igualmente populares, como los tipos de ficheros `.zip`, `.rar`, `.arj`, etc. Sin embargo, aquí nos centraremos exclusivamente en el comando `tar`, dado que como ya se ha dicho, el formato `.tgz` es el más comúnmente usado para la compresión en Linux.

Con permiso

Las operaciones anteriores de lectura y modificación de ficheros y directorios sólo se pueden realizar si disponemos de los permisos pertinentes sobre los ficheros o directorios involucrados. En el listado con formato largo de un directorio cualquiera (obtenido con el comando `ls -l`, ver de nuevo la figura **??**) se muestran los permisos de cada entrada. Como se puede observar en dicha figura, los permisos disponibles para cada entrada son 9, agrupados de 3 en 3. Los 3 grupos se refieren, respectivamente, al *propietario del fichero*, al *resto de usuarios pertenecientes al grupo del fichero* y *resto de usuarios del sistema*. Para cada uno de esos grupos se definen 3 permisos, llamados de *lectura* (`r`), *escritura* (`w`) y *ejecución* (`x`).

He aquí una explicación de los mismos (que varía de interpretación dependiendo de si los permisos se refieren a un fichero o un directorio):

- Para ficheros:

  El significado de los permisos de lectura y escritura es obvio. El de lectura nos permite acceder al contenido del fichero con una orden o un editor de texto, pero si no tenemos permiso de escritura no se habilitará en el menú de éste la opción de salvar fichero. El permiso de ejecución se le añade al fichero sólo cuando éste es un programa ejecutable o cuando, después de editar un fichero con un editor de texto ASCII plano, queremos que éste se ejecute como un *script* (también llamado *guión de shell*).

  Aunque se permite cualquier combinación de los tres permisos, habitualmente un fichero tiene permiso de sólo lectura, o de lectura y también de escritura, o de lectura y ejecución, o de lectura, escritura y ejecución. Es atípico que un fichero sólo tenga permiso de escritura, pues su carencia del permiso de lectura impide que el editor lo lea[5].

- Para directorios:

  Aquí el significado de los permisos no es tan obvio. El permiso de lectura permite conocer las entradas del directorio, por ejemplo realizando un listado de su contenido. El permiso de escritura permite modificar las entradas al directorio (por ejemplo crear, borrar o renombrar ficheros o subdirectorios dentro de él). Es importante notar que dicho permiso no tiene relación con la modificación del contenido en sí de los ficheros, puesto que ese otro es un permiso propio de cada fichero individual. Finalmente, el permiso de ejecución permite entrar dentro de un directorio (por ejemplo con el comando `cd directorio`) y modificarlo si tenemos habilitado el permiso de escritura.

  Habitualmente se suelen habilitar los de escritura y ejecución conjuntamente si queremos dar permisos para borrar entradas o crear nuevas. La habilitación de sólo el permiso de lectura de un directorio nos permite conocer sus entradas (nombres de ficheros y directorios) pero nada más (ni permisos, ni fechas, ni tamaños, etc.). El permiso de sólo escritura por sí sólo ni siquiera permite modificar las entradas del directorio (borrar o crear ficheros o directorios), pero si éste va acompañado del permiso de ejecución sí nos permite modificar las entradas del directorio (crear o borrar ficheros o directorios), a pesar de no poder luego consultar el contenido del subdirectorio. Obviamente, de todas formas, todos estos usos no son demasiado comunes, por tener una utilidad dudosa. Lo normal para un subdirectorio será que tenga los permisos de lectura, escritura y ejecución (`rwx`) para el propietario del fichero, y los permisos de lectura y ejecución (`r-x`) para el resto (grupo propietario y resto de usuarios).

Los comandos relacionados con el cambio de permisos, usuarios y grupos son los siguientes:

- `chmod 3_cifras_en_octal fichero|directorio` : Cambia permisos de ficheros o directorios. Como los permisos de propietario, de grupo y de resto de usuarios son tres para cada tipo, éstos se pueden designar con 3 cifras octales. Por ejemplo, `754 = 111 101 100 = rwx r-x r--`.

[5] La escritura en estos ficheros podría hacerse en cualquier caso, por ejemplo, con una orden redireccionada al fichero, como veremos en el apartado de redireccionamientos.

- `chown nuevo_prop fichero|directorio` : Cambia el propietario de un fichero o directorio. Obviamente, se necesita ser superusuario para poder ejecutar con éxito este comando. Esta orden también permite cambiar el grupo propietario del fichero o directorio.
- `chgrp nuevo_grupo fichero|directorio` : Cambia el grupo de un fichero o directorio. De nuevo, se necesita ser superusuario.

¿Dónde está?

Una tarea muy frecuente en el manejo del SO es encontrar la ubicación de ficheros o directorios. Una orden de Linux muy útil para ello es `find`, que maneja una gramática derivada de C. Esta orden dispone de muchas opciones y parámetros por lo que permite realizar búsquedas con mucha precisión. El comando `find` busca recursivamente entradas (ficheros, subdirectorios, etc.) en el(los) directorios que cuelguen de la(s) ruta(s) especificada(s) como parámetro(s). El formato general es `find opciones ruta1 [ruta2...] expresión`. A continuación se muestran varios ejemplos concretos de uso (la explicación de cada opción individual se mostrará inmediatamente después):

- `find dir -type f -iname "*.tgz"` : Busca dentro de `dir` ficheros con extensión `.tgz` (o `.TGZ`, `.Tgz`, etc.).
- `find /etc/default dir -type d -name "dir_*"` : Busca tanto bajo `/etc/default` como bajo `dir` los subdirectorios cuyo nombre empiece exactamente con la cadena `dir_`.
- `find dir -size +140k -o -mtime -100` : Busca en `dir` los ficheros y directorios de más de 140 KiB, o bien que hayan sido modificados hace menos de 100 días.
- `find . ! -user pedro -printf "%s %u %p\n"` : Busca en el directorio actual entradas que no pertenezcan del usuario `pedro`, y para cada una de ellas imprime una línea con el tamaño, el usuario y el camino completo al fichero.

He aquí la explicación de las opciones empleadas en dichos ejemplos, así como otras igualmente interesantes:

- `-name nombre` : Nombre del fichero a buscar (se pueden utilizar comodines, en cuyo caso se debe encerrar el patrón a buscar entre comillas dobles).
- `-iname nombre` : Similar al anterior pero sin distinguir entre mayúsculas y minúsculas.
- `-user usuario` : Propietario al que debe pertenecer el fichero.
- `-group grupo` : Grupo al que debe pertenecer el fichero
- `-type f|d` : Tipo del fichero (fichero regular o directorio). Hay muchas más opciones, pero quedan fuera del alcance de un curso introductorio de primero.
- `-mtime [+|-]n` : Fichero modificado hace más de, menos de o exactamente n días.
- `-atime [+|-]n` : Fichero utilizado hace más de, menos de o exactamente n días.
- `-amin [+|-]n` : Fichero utilizado hace más de, menos de o exactamente n minutos.
- `-size [+|-]n` : Fichero con un tamaño de más, menos o exactamente $512 \times n$ bytes. Se puede especificar una unidad de mayor tamaño, por ejemplo `-size 28k` para indicar ficheros que tengan un tamaño de 28 KiB.
- `!` : Negación de un criterio lógico.

- `-o` : O lógico de dos criterios.
- `-a` : Y lógico de dos criterios (esta es en realidad la operación lógica implícita por defecto cuando se indican varios criterios).

  Un ejemplo de combinación de varios predicados lógicos podría ser el siguiente, que buscaría, a partir del directorio `/usr`, aquellos ficheros regulares cuyo tamaño sea superior a 1 KiB o no pertenezcan al usuario `miguel`:

  ```
  find /usr -type f -size +1k -o ! -user miguel
  ```

- `-printf "..."` : Finalmente, la opción `-printf` del `find` es muy versátil. Está basada en la función `printf` (imprimir con formato) del lenguaje C, para imprimir cadenas de texto mezclado con otra información. Un ejemplo del uso de esta función en C sería el siguiente:

  ```
  printf("La variable i contiene el valor %d \n ",i);
  ```

  Que, si `i=10`, imprimiría la siguiente línea (el `\n` final es el carácter retorno de carro):

  ```
  La variable i contiene el valor 10.
  ```

  En el caso de la opción `-printf` del comando `find`, los distintos valores de la cadena de formato (que comienzan siempre con el carácter `%`) indican distintos tipos de información útil para cada fichero o directorio encontrado. Por ejemplo, `%s` = tamaño del fichero, `%u` = usuario del fichero, `%p` = ruta absoluta del fichero, etc.[6] Un ejemplo concreto de uso podría ser el siguiente comando, que busca ficheros que cuelguen del directorio actual, y para cada uno de ellos imprime una línea de la forma "`El fichero ejemplo.txt ocupa 1239 bytes`".

  ```
  find . -type f -printf "El fichero %p ocupa %s bytes\n":
  ```

- En ocasiones (si somos un usuario normal) cuando la búsqueda realizada por el comando `find` se hace de manera que es necesario entrar en directorios en los que no tenemos permisos, suelen aparecer unos mensajes de error advirtiéndonos de que no disponemos de tales permisos. Para evitar ese (largo) listado de errores podemos descartar éstos haciendo un redireccionamiento de la salida estándar de error, o *stderr* a `/dev/null` que es un fichero especial «nulo» que descarta todo lo que se escribe en él. Comentaremos el redireccionamiento de la salida de los comandos en un apartado más adelante. Un ejemplo de este redireccionamiento podría ser el siguiente:

  ```
  find / -name fichero_que_busco 2> /dev/null
  ```

Espacio en disco

- `df` : Informa de espacio total, libre y ocupado en todos los sistemas de ficheros montados (discos duros, disquetes, discos USB, etc.).
- `du -hs directorio1 ...` : Muestra los bloques que ocupan realmente en disco el(los) directorio(s) indicado(s) y todo lo que cuelga de él(ellos). Las opciones `-h` y `-s` sirven para indicar que el listado lo queremos en unidades (más legibles) de KiB, MB o GB (y no en bloques de 1024 bytes, como se muestra por defecto), y que queremos que se resuman los contenidos de cada directorio que se pasa como parámetro (en lugar de mostrar detalladamente lo que ocuparía cada entrada contenida dentro de dichos directorios).

B4.1.5. Gestión de procesos

¿Qué hacemos?

Cualquier programa que podamos lanzar desde el menú del entorno gráfico se puede lanzar también desde el terminal. Hasta ahora habíamos estado lanzando, sobre todo, comandos que tienen una respuesta textual inmediata

[6] La lista completa de cadenas de formato permitidas se puede encontrar en la página del manual del comando `find`.

por el mismo terminal. Es decir, el programa correspondiente al comando se cargaba en memoria, se ejecutaba, y el proceso correspondiente acababa enseguida, liberando de esa forma el terminal y dejándolo listo para ejecutar más comandos. Sin embargo, también pueden lanzarse programas desde el terminal que no tienen por qué terminar inmediatamente. Por ejemplo, el editor de textos `gedit`, el reloj gráfico `xclock`, el navegador `firefox`, etc.

En realidad, `bash` permite, incluso, lanzar varios programas usando una sola línea de comandos, y ello se puede hacer básicamente de dos maneras:

1. `comando1 ; comando2 ; comando3 ...` : Lanza procesos en secuencia, uno después de otro, cada uno de ellos siempre en primer plano. Lanzar un proceso en primer plano significa que, hasta que no acaba, no se lanza el siguiente; y hasta que no acaba el último, no vuelve el *prompt* del `bash`. De esta forma, además, se tiene un cierto control del proceso lanzado desde la propia línea de comandos, pulsando combinaciones de teclas. Así, pulsando `Ctrl-C` en el terminal se puede matar (forzar su finalización) el proceso que en ese momento se encuentra en primer plano, para pasar a ejecutar el siguiente comando, o liberando el terminal si el comando que estaba en ejecución era el último (o el único) de la secuencia. De forma similar, pulsando `Ctrl-Z` se duerme al proceso en primer plano (es decir, se para "en seco" su ejecución, pero sin matarlo, puesto que más tarde querremos continuar su ejecución por donde lo habíamos dejado). Dicho proceso dormido se puede despertar luego con cualquiera de los comandos `fg` (que lo despertaría devolviéndolo de nuevo a primer plano) o `bg` (que lo despertaría, pero pasándolo a segundo plano en este caso; describiremos el lanzamiento en segundo plano justo a continuación).

2. `comando1 & comando2 & comando3 & ...` : Lanza procesos en segundo plano. Lanzar uno o varios procesos en segundo plano implica que cada uno no tiene que esperar a que termine el anterior, y que el *prompt* vuelve al `bash` inmediatamente. Por ejemplo, el comando `firefox &` lanza una ventana con el navegador, y sin necesidad de que éste termine pueden seguir tecleándose comandos en el terminal. El `bash` siempre imprime el PID (identificador de proceso, un entero único para cada proceso del sistema) de cada proceso lanzado en segundo plano.

Los comandos más importantes para el control de procesos desde la línea de comandos son los siguientes:

- `kill [-9] PID` : Lanza una señal al proceso identificado por su número de PID. Sin parámetro indicando el número de señal, se envía la señal `-15` que termina (mata) al proceso. Esto funciona en general, pero existe un mecanismo por el cual ciertos procesos pueden capturar dicha señal, evitando así su finalización. Si, aún así, se desea matar a uno de dichos procesos, la opción `-9` lo termina forzosamente (se dice que la señal no es "capturable"). Para saber el PID de un proceso determinado pueden utilizarse cualquiera de los dos comandos que vienen a continuación.

- `ps [-Af]` : El comando `ps` muestra un listado de los procesos activos. Usando la primera de las opciones indicadas (`-A`), muestra todos los procesos del sistema (no sólo los lanzados desde ese terminal, que sería lo que ocurriría por defecto, en caso de lanzar el comando `ps` sin opciones). Si se usa la opción `-f`, entonces se presenta un listado de procesos más largo, que muestra distintas informaciones de interés, tales como el PID de cada proceso, su consumo de CPU, el PID de su proceso padre (PPID), la hora de lanzamiento, etc. Es muy habitual usar ambas opciones a la vez, en la forma `ps -Af`. La figura **??** muestra un ejemplo de la información mostrada por dicho comando.

Redireccionamiento

Una de las potencialidades de los SOs (no exclusiva de Linux) es la de encadenar las salidas que generan las órdenes con las entradas de otras. Cada proceso tiene una entrada por defecto (denominada *stdin*) que habitualmente es el teclado, una salida por defecto (denominada *stdout*) que habitualmente es la propia ventana de texto, y una salida de error (denominada *stderr*), que también va por defecto a la ventana de texto. Sin embargo, un usuario siempre puede redireccionar cualquiera de dichas entradas/salidas desde/a un fichero en el momento de lanzar un comando. He aquí

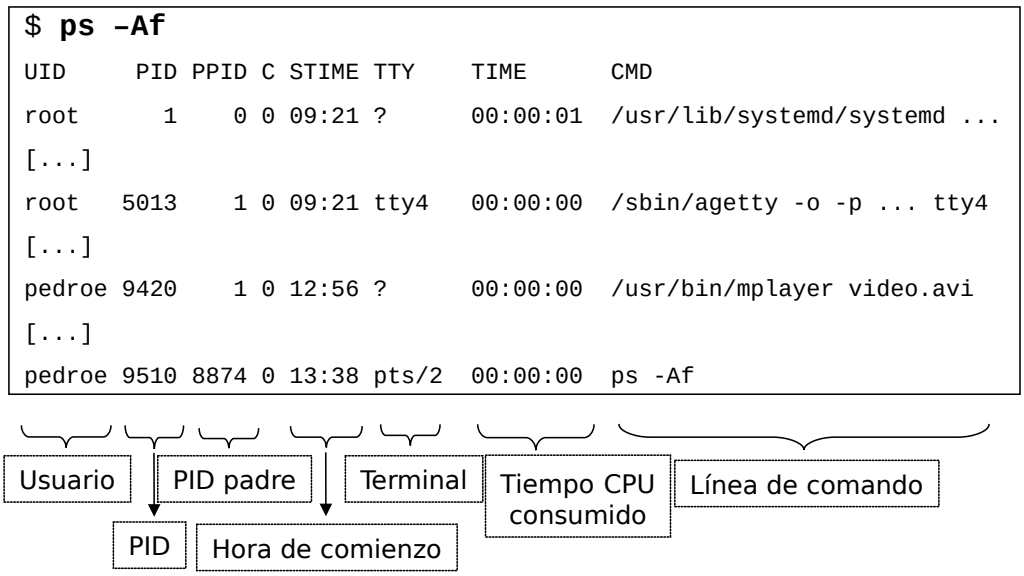

Figura I.4: Ilustración del comando `ps` (ver texto).

las distintas formas de hacerlo (usaremos como ejemplo el comando `ls -l` para ilustrar el redireccionamiento de las salidas, y el comando `sort` para el de la entrada, aunque por supuesto los mecanismos comentados se pueden emplear con cualesquiera comandos que lean escriban en *stdout* y/o *stderr* y/o lean de *stdin*):

- `ls -l > fich.txt` : En lugar de escribir la salida *sdout* en pantalla, se escribe en un fichero llamado `fich.txt` (creándolo si no existía, o sobreescribiéndolo si ya existía).
- `ls -l >> fich.txt` : Ídem, pero añadiendo la salida del comando al final del fichero, en lugar de "machacarlo".
- `ls -l noexiste.c *.h 2> err.txt` : La salida de error del comando (*stderr*) se redirecciona al fichero `err.txt`. También se puede usar `2>>`, para añadir en lugar de "machacar".
- `sort < fichero` : El comando `sort` (que ordena líneas por orden alfabético) toma su entrada (*stdin*) de un fichero en lugar de por teclado.

Algunos comandos adicionales

Para finalizar, dejamos aquí algunas otras órdenes de interés que no han encontrado sitio en los apartados anteriores, al ser un poco más de tipo miscelánea.

- `date` : Devuelve la fecha y la hora actuales.
- `cal` : Un calendario del mes/año en curso.
- `clear` : Limpia el terminal.
- `reset` : Resetea el terminal (para cuando queda "tonto", con caracteres extraños; por ejemplo, tras hacer `cat` de un fichero binario).

B4.2. Boletín 1

B4.2.1. Objetivos

El objetivo de este primer boletín de prácticas del tema 4 es familiarizar a los alumnos con el uso básico del SO Linux, haciendo especial hincapié en el manejo de éste desde el intérprete de comandos `bash`. También se introducirán una serie de comandos básicos para dar los primeros pasos en este SO. Más concretamente, se tratarán los siguientes temas:

- Acceso al sistema.
- Interfaz gráfico (KDE/GNOME). Lanzamiento de programas. Utilidades varias (editores, navegadores, etc.).
- Intérprete de comandos `bash`. Formato general de una orden, uso del tabulador, historial de comandos.
- Documentación y búsqueda de información. Páginas del manual.
- Movimiento por la jerarquía de directorios. Listados. Creación y borrado de ficheros y directorios.
- Metainformación sobre ficheros: propietario, permisos, fechas, tamaño, etc.

B4.2.2. Plan de trabajo

El plan de trabajo de esta sesión será el siguiente:

1. Lectura del boletín por parte del alumno.
2. Seguimiento de ejemplos presentados por el profesor.
3. Realización, en grupos de dos personas, de los ejercicios propuestos en el boletín (con supervisión del profesor, y completándolos en su caso como trabajo autónomo).

B4.2.3. Ejercicios a realizar durante la sesión

1. **Terminales de texto.** Antes de acceder al modo gráfico, probaremos primero el interfaz puramente textual del sistema. Para ello, cambia a un terminal de texto con `Ctrl-Alt-Fn` (con n=1-6 tenemos acceso a distintos terminales). Introducir nuestro usuario y contraseña, comúnmente también llamados *login* y *password*. En los laboratorios de prácticas tenemos usuario y contraseña: `alumno`/`alumno`. Prueba los siguientes comandos, indicando el resultado obtenido, en el caso de la orden `ls` date cuenta las diferencias en cada caso.

   a) `date`
   b) `pwd`
   c) `w`
   d) `ls`
   e) `ls -l`
   f) Para terminar, cierra la sesión con `exit`, y vuelve al terminal gráfico con `Ctrl-Alt-F7`.

2. **Interfaz gráfico.** Entra en la sesión gráfica usando de nuevo nuestro *login* y *password*. Dependiendo de la instalación, habitualmente se entrará en uno de los dos entornos gráficos más conocidos, KDE o GNOME (éste último es el configurado por defecto en los laboratorios de prácticas, pero todo lo que veremos vale igualmente para KDE). Una vez allí, lanza un terminal buscando el programa *"Terminal"* en el menú adecuado. Incluso podemos generar un acceso directo a la aplicación en el escritorio, arrastrando desde el menú el icono correspondiente. Una vez lanzado, desde esta ventana de terminal gráfico iremos ejecutando el resto de comandos del boletín.

3. **Movimiento básico por el sistema.** Desde el terminal abierto en el apartado anterior, realiza los siguientes ejercicios:

   a) Muestra la ruta absoluta del directorio en el que te encuentras con la orden `pwd`.

   b) Cámbiate al subdirectorio `bin` bajo el directorio `/usr` que está en la raíz, usando el comando `cd` con una *ruta absoluta*. Comprueba de nuevo cuál es tu directorio de trabajo actual (`pwd`) para ver que la orden se ha ejecutado sin errores.

   c) Usando la opción `-l` de la orden `ls` obtén un listado de todos los ficheros allí almacenados. Date cuenta que todos son los ficheros ejecutables (comandos).

   d) Compara los contenidos de dicha carpeta con los contenidos mostrados por un navegador de archivos tradicional (en una ventana gráfica), `konqueror` (KDE) o `nautilus` (GNOME).

   e) Obtén ahora un listado que muestre sólo aquellos ficheros que empiezan con la letra «a».

   f) Vuelve al directorio de inicio de sesión (directorio *home*) usando el comando `cd` sin parámetros. Comprueba en todo momento nuestra situación en la jerarquía de directorios usando `pwd`.

   g) Cámbiate, con un sólo comando `cd [ruta]`, al directorio `/etc/X11`, pero usando esta vez una *ruta relativa* (que NO comience con una `/`).

   h) Experimenta con la compleción de rutas con el tabulador (tecleando, por ejemplo, `cd /us[tab]/bi[tab]/X[tab]`, etc.)

   i) Regresa de nuevo a tu directorio de inicio de sesión y, sin moverte de él, y usando una única orden, lista todos los ficheros del directorio `/usr/bin` cuyo nombre tenga exactamente 4 caracteres cualesquiera. Para indicar el directorio `/usr/bin` usa una ruta relativa.

   j) Ejecuta la orden anterior de nuevo, pero usando la ruta absoluta para indicar el directorio `/usr/bin`.

4. **Creación de ficheros, directorios, copiado, etc.**

   a) Crea en tu directorio personal el directorio `pracFC` (usando `mkdir`) y, sin moverte de tu directorio personal, copia (`cp`) en él todos los ficheros y directorios del directorio `/etc` que empiecen con la letra `g`. Al hacer la copia conserva todos los atributos posibles para cada fichero (permisos y propietarios, y tiempos, etc.). Usa la orden `man cp` para saber qué parámetro o parámetros tienes que usar.

   A continuación, usa la orden `ls pracFC` para comprobar que se han copiado los ficheros y directorios.

   b) Usando la orden `cat` muestra el contenido de los ficheros `/etc/group` y `pracFC/group`, comprobando que tienen el mismo contenido.

   c) Ejecuta un `ls -l` a ambos ficheros (`/etc/group` y `pracFC/group`). ¿Quién es el propietario? ¿y el grupo propietario?

   d) Intenta editar el fichero `pracFC/group` llamando al editor `kate` (KDE) o `gedit` (GNOME). Intenta modificar dicho fichero, ¿puedes?. Podrás modificar `pracFC/group` pero no `/etc/group`, ya que este último propiedad del superusuario, y como usuarios normales no tenemos permiso de escritura sobre él. Así que no se podrá modificar su contenido desde el editor; sólo podremos consultarlo en lectura.

   e) Consulta los códigos ASCII en hexadecimal, usando el comando `hexdump -C`, de los caracteres contenidos en el fichero `group` que has copiado anteriormente.

   f) Muévete al directorio `/usr` y desde allí, usando rutas relativas, vuelve a realizar el apartado anterior. Al terminar, ejecuta `cd` para volver a tu directorio previo.

   g) Mueve, usando la orden `mv`, el fichero `group` del directorio `pracFC` a tu directorio *home* o de inicio (`/home/alumno` si haces las prácticas en los laboratorios).

   h) Vuelve a mostrar el contenido del fichero `group`, esta vez usando la orden `cat`.

   i) Cambia el nombre al directorio `pracFC` (creado anteriormente) a `copia_etc`.

j) Muestra los nombres de todos los ficheros y directorios ocultos de tu directorio personal.

5. **Selección y copia de texto.** En tu directorio de trabajo ejecuta la orden `ls -l`. A continuación, mediante el ratón, selecciona con el botón izquierdo la salida de la orden `ls` y pégala en la ventana de un editor de texto (`gedit` o `kate`) inicialmente vacío usando el botón central del ratón. Guarda el fichero generado en un fichero llamado `salida.txt` en el directorio *home*.

6. **Metadatos de un fichero.** Comprueba, usando de nuevo el comando `ls -l`, el tamaño lógico del fichero `salida.txt`, y compáralo con el que realmente ocupa en disco con el comando `du -h`. La diferencia es la siguiente. La orden `ls` muestra el tamaño real, en este caso en bytes, del fichero. Sin embargo, los ficheros no se almacenan en disco por bytes, sino en bloques de tamaño fijo, que normalmente son bloques de 4 KiB. La orden `du` muestra cuántos bloques está usando el fichero.

   Observa también los distintos *metadatos* de los ficheros (fecha, propietario, grupo propietario, permisos, etc.) que ha mostrado la orden `ls`.

7. **Historial de comandos.** Una vez tecleados unos cuantos comandos, veamos el histórico de comandos.

   a) Ejecuta el comando `history` y comprueba los comandos tecleados.

   b) Repite el comando que listaba el fichero `group` de tres formas distintas:

      1) utilizando las flechas del cursor para volver sobre los comandos anteriores,
      2) con `Ctrl-R` y tecleando parte del comando (p.e., el nombre del fichero, `group`), y
      3) usando `!num`, donde `num` es el número de orden del comando en el historial.

   c) Cierra el terminal usando con el comando `exit`, y vuelve a abrir otra ventana de terminal.

   d) Lista todos los ficheros (incluyendo los ocultos) con el comando `ls -a`. Comprueba la existencia de un fichero oculto llamado `.bash_history`. Consulta su contenido con `cat`, y observa cómo el `bash` guarda ahí el historial de comandos para recordarlo en sesiones posteriores.

8. **Ayuda y documentación.** El comando `man` se puede usar para consultar la página del manual de un determinado comando. Por ejemplo, usa `man ls` para ver todas las posibles opciones del comando `ls`. Nos podemos mover por la página con las flechas y el avance o retroceso de página. Podemos incluso buscar palabras concretas dentro de páginas del manual muy largas, pulsando `/palabra (intro)`. Pulsando de nuevo `/ (intro)` se busca la siguiente aparición. Para salir pulsa la tecla `q`.

9. **Permisos sobre directorios.**

   a) Desde el directorio de inicio, cambia los permisos del directorio `copia_etc` para que no tenga permiso `x` (permisos octales 600), e intenta hacer de nuevo un `cd copia_etc` al directorio.

   b) Vuelve a cambiar los permisos del directorio ahora a 500 (`r` y `x`, pero no `w`).

   c) Comprueba si se pueden al menos listar los contenidos del directorio

   d) Elimina todos los permisos (octal 000) e intenta listar los contenidos del directorio.

   e) Establece los permisos (755) al directorio `copia_etc`.

10. **Lanzamiento de programas.** Existen varias formas de lanzar comandos en Linux: a través del menú o icono correspondiente (al estilo clásico de Windows), llamando al comando correspondiente desde el terminal, o usando el menú de lanzamiento rápido (`Alt-F2`). Como ejercicio, lanza la calculadora de KDE (comando `kcalc`) y/o GNOME (`gnome-calculator`) usando estos tres métodos. Observa que las aplicaciones gráficas de KDE pueden ejecutarse perfectamente desde GNOME, y viceversa, puesto que en realidad ambos sistemas gráficos corren sobre una base común, llamada X-Windows. También podemos lanzar el famoso programa `firefox` para navegar por la web, o el visor de ficheros PDF `acroread`, por ejemplo.

11. **Terminando** Desde tu directorio personal, intenta borrar el directorio `copia_etc` usando la orden `rm`. ¿Puedes? Si no, usa la orden adecuada para hacer el borrado. Una posible opción es añadir parámetros a la orden.

B4.3. Boletín 2

B4.3.1. Objetivos

En este segundo boletín continuaremos profundizando en el uso de la línea de comandos de Linux con unos cuantos comandos y funcionalidades adicionales de utilidad. Más concretamente:

- Comandos adicionales de manejo del sistema de ficheros (búsqueda, compresión/descompresión, acceso a contenidos).
- Comodines.
- Control y monitorización de procesos.
- Redireccionamiento de la E/S. Comunicación entre procesos mediante tuberías.

B4.3.2. Plan de trabajo

El plan de trabajo de esta sesión será el siguiente:

1. Lectura del boletín por parte del alumno.
2. Seguimiento de ejemplos presentados por el profesor.
3. Realización, en grupos de dos personas, de los ejercicios propuestos en el boletín (con supervisión del profesor, y completándolos en su caso como trabajo autónomo).

B4.3.3. Ejercicios a realizar durante la sesión

1. **Búsquedas en el sistema de ficheros. Comodines.**

   a) Crea un subdirectorio `prueba` que cuelgue del directorio de inicio del usuario, y dentro de `prueba`, crea dos nuevos subdirectorios `etc` e `include`.

   b) Copia, con la orden `cp`, todos los ficheros con extensión `.h` del directorio `/usr/include` al subdirectorio `prueba/include`.

   c) Copia, con la orden `cp`, todos los ficheros y subdirectorios que empiecen por la letra `a` del directorio `/etc` al subdirectorio `prueba/etc`.

   d) Vuelve al directorio de inicio y, desde allí, usa el comando `find` muestra toda la jerarquía construida/copiada. Habría que ejecutar la orden `find` sin parámetros.

   e) Usando convenientemente los comodines y las opciones `-name`, `-type f` y `-type d`, mostrar:

      1) todos los directorios,
      2) todos los ficheros,
      3) todos los ficheros con extensión `.h`,
      4) todos los ficheros con extensión `.h` y su nombre empiece con una letra entre la `a` a la `f`,
      5) todos los ficheros que no tengan la extensión `.h`,
      6) todos los ficheros cuyo nombre empiece por una letra entre la `a` y la `g`.

   f) Busca recursivamente a partir de los directorios `/etc/`, `/usr/bin` y `/usr/sbin`, todos los ficheros regulares que no hayan sido accedidos en los últimos 7 días.

   g) Busca recursivamente a partir del directorio `/etc/`, todos los ficheros regulares con un tamaño mayor que 10 KiB y menor que 20 KiB.

2. **Copias de seguridad. Compresión y descompresión.** Ejecuta todas las órdenes siguientes desde el directorio de inicio, y sin cambiar nunca de directorio:

   a) Genera una copia de todo el árbol de directorios `prueba` llamada `copiaprueba`, usando la orden `cp -Rp`. Consulta en el manual de `cp` qué significan exactamente estas dos opciones.

   b) Genera ahora, usando la orden `tar` con las opciones `czvf`, un fichero comprimido llamado `prueba.tgz`, que contenga todo el directorio `prueba`. Comprueba el tamaño del directorio en disco (con `du -hs`), y compáralo con el tamaño del fichero comprimido generado (con `ls -l`).

   c) Para el fichero creado, lista los contenidos del mismo con la orden `tar tzvf` y comprueba la corrección del mismo.

   d) Si el fichero `tar` creado es correcto, intenta borrar íntegramente el directorio `prueba` con `rmdir`. Al no estar vacío, no se puede borrar, así que tendremos que borrarlo usando las opciones `-rf` del comando `rm` (consulta en el manual su significado).

   e) Comprueba con `ls` que el directorio ha sido convenientemente borrado.

   f) Restaura con `tar xzvf` el directorio borrado desde el fichero `tar` creado. Comprueba usando la orden `ls` el contenido del directorio restaurado.

3. **Más sobre permisos en ficheros. Concepto de PATH.**

   a) Genera una copia (usando el comando `cp`) del fichero `/bin/ls` en nuestro directorio de inicio, con el nombre `myls`.

   b) Comprueba que el comando sigue funcionando tecleando `./myls -l`, por ejemplo[7].

   c) A continuación, usa el comando `chmod` sobre el fichero copiado para eliminar su permiso de ejecución, y vuelve a intentar la ejecución del comando.

   d) Vuelve a añadir el permiso de ejecución y comprueba que ahora sí puedes ejecutarlo.

4. **Control de procesos en primer y segundo plano.**

   a) Lanza los programas `kcalc` (calculadora) y `kate` (editor) desde el terminal, con la única línea de comandos `kcalc ; kate ;`. Observa cómo, al lanzarse en primer plano, hasta que no termine el primer proceso (cuando cerramos la calculadora) no se lanza el siguiente. Cierra ambos programas consecutivamente.

   b) Vuelve a lanzarlos, pero esta vez en segundo plano, tecleando `kcalc & kate &`. Comprueba que ambos se lanzan y funcionan simultáneamente, y podrían terminarse ahora en cualquier orden.

   c) Con el comando `fg`, pasa a primer plano el último programa que se lanzó (el editor `kate`), y, desde el terminal, mátalo pulsando Ctrl-C.

   d) Repite el comando `fg` para hacer lo propio con el comando anterior (`kcalc`), pero esta vez, en lugar de matarlo, páralo pulsando Ctrl-Z desde el terminal. Observa que el programa `kcalc` temporalmente no responde al ratón ni al teclado (puesto que está durmiendo temporalmente, lo acabamos de parar).

   e) Vuelve entonces a despertarlo, pasándolo a segundo plano, con `bg`, y comprueba que su ventana ya responde perfectamente. Finalmente, matar al proceso desde el terminal esta vez usando el comando `kill -9 [PID]`, usando el PID (identificador del proceso) que nos indicó el terminal al lanzar el proceso inicialmente. Puedes usar la orden `ps -Af` para obtener el PID del proceso.

[7] Observar la necesidad de proporcionar la ruta hasta el comando, por no estar nuestro directorio de inicio en la lista de caminos por defecto para buscar los comandos ejecutables, más conocida como `PATH`; podéis consultar esta última lista de caminos mediante el comando `echo $PATH`.

5. **Listado de procesos.** Usa el comando `ps -Af` para mostrar todos los procesos que se están ejecutando actualmente en el sistema, y resuelve los siguientes apartados:

   a) Localiza tres procesos cuyo propietario sea nuestro usuario, y otros tres cuyo propietario sea el superusuario (`root`).

   b) Localiza el proceso que ha consumido más tiempo de CPU.

   c) Localiza también el último que se lanzó, mirando las horas de inicio (lógicamente, será normalmente nuestro propio `ps`).

6. **Redireccionamiento de la E/S**:

   a) Lista todos los ficheros contenidos en `/bin/`, con el comando `ls /bin`.

   b) Repite el comando redireccionando la salida a un fichero llamado `f1.txt`, usando el símbolo $>$.

   c) Comprueba el contenido de dicho fichero con `cat f1.txt`.

   d) Añade ahora a dicho fichero la fecha actual y los ficheros del directorio `/etc`, con los comandos `date` y `ls`, usando esta vez el símbolo $>>$.

   e) Comprueba de nuevo el contenido del fichero, esta vez con un editor de textos (p.e. el `kate`).