Laboratorio Sesión 01: Ficheros, scripts y usuarios

En esta práctica vamos a ver comandos básicos para el manejo de ficheros y directorios. Todo lo que vamos a ver al principio es casi tan fácil de hacer con ventanas como mediante consola, pero enseguida veremos algún ejemplo que NO se puede hacer con el sistema gráfico. A continuación veremos algun primer ejemplo de scripts y como gestionan los sistemas *NIX los usuarios.

1.1. Comandos de ficheros y directorios.

En primer lugar las órdenes para crear y borrar directorios: `mkdir` y `rmdir`[1]. Probad a ir a vuestro directorio de trabajo y crear un directorio llamado prueba. A continuación podéis hacer un `ls` para ver que efectivamente la orden ha funcionado y borrarlo.

Ahora vamos a crear unos cuantos ficheros vacíos. Para ello hay un comando llamado `touch`. Este comando originalmente se diseñó para modificar la fecha y hora de un fichero (de ahí que se llame "tocar"), pero se suele usar más para crear un fichero vacío. Probad a crear en vuestro directorio 10 ficheros vacíos llamados desde `fichero1` a `fichero0`. Recordad que tenéis como ayuda la función autocompletar (con el tabulador) o la flecha arriba que hará aparecer en la línea de comandos la orden anterior. Si aún así os da mucha faena probad con la orden `touch fichero{0,1,2,3,4,5,6,7,8,9}`. Cread también otro fichero llamado `fichera1`.

Vamos a probar a copiar ficheros. Para ello cread un subdirectorio llamado `sub` y ejecutad la orden `cp fichero0 sub`. Si hacéis un `ls` al directorio veréis que se han copiado los ficheros. A continuación probad a borrar el subdirectorio con `rmdir`. El sistema no os dejará ya que `sub` no está vacío. También podéis copiar el fichero con otro nombre con `cp fichero0 sub/otrofichero0`. O bien podéis mover algún fichero desde donde estáis hasta `sub`: `mv fichero1 sub/`, o incluso cambiarle el nombre: `mv fichera1 fichera0` o ambas cosas: `mv fichera0 sub/fichera1`.

Algo que también es muy útil es poder mover o copiar de golpe varios ficheros. El `*` substituye cualquier cadena de caracteres en un nombre, mientras que el `?` substituye a un solo carácter. Probad la diferencia ejecutando las siguientes ordenes:

```
mkdir sub2
cp fich* sub2
mkdir sub3
cp fichero? sub3
```

y comprobad el resultado con `ls`. A veces incluso es útil copiar o mover directorios y todo lo que contienen:

Pregunta 1.1: *¿Qué pasa si ejecutamos* `cp -r sub? sub/`*?*

Eso sí, recordad que Linux diferencia entre mayúsculas y minúsculas, así que procurad no equivocaros.

El último comando útil para el manejo de ficheros es el de borrar: `rm`. Fijaros en que el sistema no os pregunta confirmación (probad con `rm fichero5`). Esto se puede forzar con la opción `-i` o se puede evitar siempre con `-f`. Probad ahora a ejecutar `rm -r sub`. Además tened en cuenta que en Linux, cuando se borra, se borra de verdad y los ficheros no suelen ser recuperables. Cuando borráis desde el entorno de ventanas en realidad solo movéis el fichero a la carpeta `.local/share/Trash/files` que está dentro de vuestro home (podéis probarlo).

Pregunta 1.2: *¿Por qué nunca se debe ejecutar* `rm -rf /`*? Probadlo y cruzad los dedos... ¿qué sucede?*

[1] Se da por supuesto que sabéis listar (`ls`) y moveros por directorios (`cd`) en modo comando.

1.2. Discos y dispositivos.

Ahora que ya sabéis algo sobre trabajar con ficheros vamos a hablar de discos y dispositivos (es probable que en este apartado tengáis que haceros superusuario en algún momento, recordad que podéis ejecutar como superusuario escribiendo `sudo` antes de un comando o cambiar a superusuario con el comando `sudo su`). En Linux se tratan todos los dispositivos como si fueran ficheros. En el caso concreto de los discos, lo que hace es ver su contenido como si fuera un directorio, de forma que todo lo que se grabe en el directorio en realidad se graba en el disco (y lo mismo con lo que se borra). El proceso que vincula un disco y un directorio se llama montar y, aunque siempre se suelen usar los mismos directorios para los mismos dispositivos, no siempre tiene por que hacerse así.

Podéis ver los dispositivos que hay montados en el sistema tecleando simplemente el comando `mount`. Veréis que hay varios dispositivos montados por defecto (uno por línea). Podéis entender algunos claramente (`/dev/sda2` es la partición donde está el sistema y por tanto la raíz `/`, el `2` puede variar según el ordenador o la sala). Mientras que el resto probablemente no os digan nada (`/proc`, por ejemplo, es un dispositivo virtual que usa el kernel para saber que programas se están ejecutando). De momento nos preocuparemos solo por aquellos dispositivos que representan un disco físico real. Los discos son básicamente los siguientes:

- `/dev/hdax` o `/dev/sdax`: representa todas las particiones del primer disco (x empieza en 1).
- `/dev/sdbx`: son las particiones de los siguientes discos (por ejemplo si insertáis una memoria USB. Más discos se representan con letras consecutivas.
- `/dev/cdrom`: representa el cd.
- `/dev/fdx`: representaba los disquetes (x empieza en 0), a estas alturas es difícil que veáis uno.

¿Cómo trabajaríais con un disco extraible? Pues muy fácil, primero enchufaríais el disco (esta era obvia) y a continuación lo montaríais con:

```
mount /dev/sdb /directorio
```

Y ahora ya podríais trabajar con el disco simplemente trabajando con el contenido de `/directorio/`. En realidad este proceso (el `mount`) lo realiza automáticamente Linux al conectar el disco. Al acabar, ANTES de extraer el disquete deberéis desmontarlo (expulsar el disco):

```
umount /dev/sdb
```

Si no lo hacéis así, puede que los cambios que hayáis realizado no se reflejen en el disco. Fijaros además que `/directorio` es un directorio cualquier que podéis crear en cualquier momento con `mkdir` (ha de estar vacío).

Para ver si lo tenéis claro, probad a hacer lo siguiente: cread en vuestro home un directorio vacío. A continuación montad en él la primera partición del disco duro vacia (en estos ordenadores suele ser `sda5`, pero dependiendo de la instalación que tengáis puede ser que tengáis que montar otra partición). Para esto seguramente tendréis que escribir `sudo` delante del comando `mount` (ya veremos más de este comando, pero en esencia, os permite ejecutar ordenes como si fuerais `root` poniendo vuestra contraseña). Una vez montado el directorio contiene todo lo que había en la partición del disco que acabáis de montar (si estaba vacía el directorio estará vacio, claro). Cread allí un nuevo directorio llamando prueba y copiad en él los ficheros que os hayan quedado del apartado anterior. A continuación desmontad la partición, observaréis que el directorio ha quedado vacio de nuevo (los ficheros siguen en el disco, claro).

Fijaros que también podéis acceder a las otras particiones a partir de los enlaces que aparecen en el entorno gráfico. En este caso los comandos `mount` se ejecutan automáticamente.

Pregunta 1.3: *¿Qué órdenes habéis tenido que usar para hacer lo anterior? Para comprobar que todo está en orden deberéis volver a montar el directorio y comprobar que los ficheros*

son los mismos que los originales comparándolos (comando `cmp`*) ¿Que devuelve* `cmp` *si comparamos dos ficheros iguales? ¿Y si son diferentes?*

Siempre se puede montar un nuevo sistema de ficheros (un dispositivo), sobre la marcha. Sin embargo, a veces interesa que automáticamente Linux arranque con ciertos sistemas montados por defecto. Un ejemplo de esto es cuando tenemos dos particiones para Linux, una para el sistema y otra para los ficheros de datos. Esto se consigue simplemente, montando la segunda partición (la de datos) en el directorio `/home` e incluyendo esta entrada en el fichero de particiones. Este fichero siempre es el mismo y se llama `/etc/fstab`. Podéis ver el contenido con el comando `cat /etc/fstab`. Veréis que en cada línea hay un sistema de ficheros. Las opciones de cada línea del fichero podéis consultarlas con `man` o `info fstab`.

Pregunta 1.4: *¿Para que sirve la opción* `user` *al especificar un sistema de ficheros en* `fstab`*?*

Pregunta 1.5: *¿Qué línea añadiríamos o cambiaríamos en* `fstab` *para que siempre apareciera montada la partición* `sda1` *en el directorio* `/part1` *al iniciar la máquina?*

Un error bastante típico es intentar escribir en un sistema de ficheros que solo está montando como de lectura (`ro`, read only). En estos casos basta con remontar el dispositivo como escribible (`rw`, read-write).

1.3. Sobre la edición de texto en la consola

En Linux en modo consola, la mayoría de trabajos y configuraciones se realizan con un editor de textos plano, es decir, sin formato, tipo el bloc de notas en Windows. Aunque hay una infinidad de editores (NEdit, KEdit, Joe, etc.), los dos más utilizados (y con un cierto "sectarismo" :-) son `vi` y `emacs`. En este entorno tenéis vi que es el que vamos a ver. `vi` tiene la ventaja de ser el único editor que seguro que va a estar en CUALQUIER sistema *NIX y que va a funcionar a través de cualquier tipo de conexión remota y con cualquier teclado.

`vi` es, además un editor un poco peculiar en el sentido en que no funciona de la forma acostumbrada. Si lo abrís sin tener ni idea, lo más probable es que acabéis perdiendo el tiempo, la paciencia y, a lo mejor, el fichero que queríais editar. Para evitarlo lo mejor es perder media hora aprendiendo lo más básico, mediante el fichero de autoaprendizaje `vimtutor`.

Pregunta 1.6: *¿Cómo se copian y pegan 10 líneas en* `vi`*?*

Una vez que se conocen los comandos básicos en `vi` la forma usual de trabajar es simplemente ir buscando en la ayuda, que se activa con el comando `:h` dentro del editor, (o en la red) aquellas ordenes que necesitamos.

Pregunta 1.7: *¿Cómo se editan dos ficheros en la misma pantalla en* `vi`*? ¿Qué combinación de teclas permite saltar de uno a otro?*

1.4. Un primer shellscript.

Vamos a ver ahora nuestra primera utilidad de consola. La mayor potencia de la consola es que tiene un completo lenguaje de programación que nos permite, muy fácilmente, automatizar tareas. El sistema es a la vez potente y simple, ya que como vais a ver se puede empezar a utilizar desde el principio.

Cread un fichero nuevo con `vi`, e introducid en el las ordenes que habéis copiado en la respuesta a la primera pregunta de la pregunta 1.3. Salvad el fichero. Bien, ya habéis creado un script. Ahora solo queda dar permiso de ejecución al script con el comando: `chmod +x nombredelscript` (luego hablaremos más de los permisos).

Ahora montad el disco `sda5` (la primera vacía) y aseguraros que NO están copiados los ficheros de prueba (borrarlos si es necesario), volved a desmontarlo y ejecutad el script (simplemente escribid su nombre, quizás necesitareis escribir `./` delante para que lo encuentre... ya hablaremos del "path"). Si todo ha ido bien, fijaros que esta es una posible forma de automatizar las copias de seguridad. Igualmente podéis hacer de forma automática cualquier tarea que realicéis muy a menudo.

Si queréis automatizar aún más una tarea podéis ponerla en un fichero que se ejecute automáticamente cada vez que se abra una terminal. El terminal que se abre por defecto cada vez que pulsáis en la pantalla es el terminal `bash` (también hablaremos más sobre el tema). Si editáis el fichero `.bashrc` de vuestro directorio `home` y añadís alguna línea al final del fichero, esta se ejecutará cada vez que iniciéis otra terminal. Probad con el comando `echo hola chico` (`echo` es una orden que simplemente imprime por pantalla lo que tiene a continuación). Si en lugar de la orden que acabáis de poner pusierais el nombre del script que habéis creado antes, lo que se ejecutaría sería todo el script. Unos scripts pueden llamar a otros tantas veces como se quiera, como si fuesen subrutinas.

1.5. Usuarios varios.

En esta parte de la práctica veremos como administrar usuarios y permisos. Para muchas de las tareas necesitaréis ser superusuarios, así que podéis trabajar directamente como "root". Para que exista un usuario "root" en Ubuntu solo hay que darle una contraseña mediante el comando `sudo password`. En la instalación de trabajo que tenéis no lo han hecho, así que podéis cambiar directamente con la orden `sudo su`, o bien, usar `sudo` cuando sea necesario.

1.5.1. Usuarios del sistema.

En un sistema Linux debería haber, como mínimo, dos usuarios, el "root" y el "usuario de trabajo". Sin embargo, Linux es un sistema multiusuario que permite que tengamos cientos de usuarios trabajando a la vez. A continuación vamos a ver como se administran los usuarios y formas útiles de trabajar con ellos.

En primer lugar deberemos saber donde almacena Linux la información sobre usuarios. Esta información es pública y está en el fichero `/etc/passwd`. Si observáis el contenido de este fichero (con `cat /etc/passwd`, `more /etc/passwd` o `vi /etc/passwd`) veréis los usuarios que tiene el sistema en este momento. Hay tantos como líneas tiene el fichero.

Pregunta 1.8: *¿Qué diferencia hay entre* `cat` *y* `more`*?*

Pregunta 1.9: *¿Cuántos usuarios hay? (probad la opción* `-n` *del comando* `cat`*)*

Fijaros que a pesar de que nuestro sistema tiene pocos usuarios reales, en el fichero aparecen muchos más. Estos usuarios son ficticios y son usados por el sistema por motivos de seguridad. Cuando un proceso tiene que hacer determinadas tareas críticas adquiere la identidad de estos usuarios ficticios para evitar conflictos de seguridad. ¿Cómo se pueden identificar los usuarios ficticios? En primer lugar deberéis averiguar que significa cada elemento de la línea del fichero "passwd". Para ello ejecutad la página `man` de "passwd".

Si todo va bien, no os habrá servido de nada. ¿Por qué? Pues porque os habrá aparecido la información de ayuda sobre el comando "passwd". Y aunque es interesante (ahora ya sabéis cambiar contraseñas), no es lo que queréis. Sin embargo, la información que buscáis si que está en el sistema, solo es difícil de encontrar porque hay dos informaciones a las que se accede por el mismo nombre, la del comando `passwd` y la del fichero "passwd". Si vais al final de la información del comando "passwd", veréis que hay un apartado que se llama "SEE ALSO". En este apartado aparece "passwd" seguido de un número entre paréntesis.

Pregunta 1.10: *¿Que número aparece entre paréntesis?*

Bien, si ahora tecleáis `man <numero> passwd` os aparecerá la información que buscáis. La información de `man` está clasificada por apartados y la forma de acceder a uno en concreto

es anteponer al comando que buscáis el número de apartado que queréis. A veces lo que uno busca no aparece en el apartado por defecto.

Pregunta 1.11: *¿Cuantos campos hay en cada línea del fichero* `/etc/passwd`*? ¿Que significa cada uno de* ellos?

Podéis ahora fijaros en la información que hay en el fichero "passwd" sobre los dos usuarios reales. Para verla más fácilmente podéis filtrar solo las líneas que os interesen:

```
grep -e root -e alumne /etc/passwd
```

El usuario "root" siempre tiene identificador ("uid", user identifier) 0, mientras que los usuarios "normale" siempre tienen identificadores del 1000 en adelante. Es una forma fácil de distinguirlos del resto.

Además ya os habréis fijado que en el fichero "passwd" no está escrita la contraseña. Este detalle (también lo explica la ayuda de "passwd") se debe a que, a pesar de que la contraseña se guarda encriptada, llegó un momento en que era muy fácil averiguarla, así que se trasladó esta información a otro fichero de más difícil acceso: `/etc/shadow`. ¿Por qué demás difícil acceso? Luego veremos más sobre permisos, pero si intentáis hacer un cat a ese fichero (o verlo de cualquier otro modo) siendo el usuario "alumne" veréis. El fichero "shadow" contiene la misma cantidad de líneas que el fichero "passwd", y para cada usuario contiene información referente a su contraseña.

Pregunta 1.12: *¿Que significa cada campo del fichero* `/etc/shadow`*?*

En Linux, además de los usuarios hay los grupos. Sirven para gestionar mejor la seguridad (lo veréis en el siguiente apartado), pero baste decir que cada usuario pertenece a un grupo principal (el identificador del grupo, el "gid", está en el fichero "passwd", justo después del "uid").

La información concreta sobre cada grupo de usuarios (no tiene porqué haber tantos como usuarios, varios usuarios pueden pertenecer al mismo grupo) se encuentra en el fichero `/etc/group`.

Pregunta 1.13: *¿Que significa cada campo del fichero* `/etc/group`*?*

Fijaros que al final de cada grupo se pueden incluir tantos usuarios como se quiera, de forma que cada grupo puede tener cualquier número de usuarios y un usuario puede pertenecer a tantos grupos como se desee.

Pregunta 1.14: *¿A cuantos grupos pertenece el usuario "alumne"?*

Una forma fácil de contestar a la pregunta anterior es probar la orden:

```
grep alumne /etc/group | cat -n
```

Vamos a verla en detalle: el símbolo `|` indica una "tubería" (pipe), es decir, la salida de `grep` es la entrada de `cat`. Podéis probar la diferencia con `cat -n /etc/group | grep alumne`. Para saber los grupos de un usuario también podéis usar la orden `groups`. Como podéis ver siempre hay muchas formas de hacer lo mismo.

Finalmente, los grupos, al igual que los usuarios, pueden tener claves de acceso (aunque no se usan tanto). Estas claves están en el fichero "shadow" de grupos que es `/etc/gshadow`.

1.5.2. Gestión de usuarios.

Para añadir usuarios y grupos al sistema se pueden editar directamente (con `vi` o cualquier otro editor) los ficheros del sistema, pero sin embargo esta práctica está desaconsejada. Lo normal es usar unos comandos pensados ya directamente para realizar estas tareas que además gestionan otros pequeños detalles (como, por ejemplo, crear un directorio "home" para cada usuario nuevo). Probad a añadir un nuevo usuario al sistema llamado "prueba". Es muy fácil, teclead `adduser` y seguid las instrucciones.

Pregunta 1.15: *¿Que pasos realiza el comando* `adduser`*?*

Podéis probar ahora a salir del sistema y volver a entrar usando el nuevo usuario (ojo, no apaguéis el ordenador, solo haced un `logout`). Como superusuarios también podéis crear nuevos grupos (`addgroup`) o añadir un usuario ya existente a un grupo ya existente (`adduser prueba alumne`).

Pregunta 1.16: *¿Cómo modifica el fichero* `/etc/group` *el comando* `adduser prueba alumne`*?*

Además podemos eliminar usuarios y grupos mediante `deluser` y `delgroup`. Podéis probar a eliminar el usuario "prueba". Fijaros que, sin embargo, como medida de seguridad, sus ficheros no se borran. Los ficheros "adduser.conf" y "deluser.conf" (los dos en `/etc/`) definen el funcionamiento de `adduser` y `deluser`. Podéis editarlos y ver sus opciones.

Pregunta 1.17: *¿Cómo se puede conseguir que* `deluser` *borre los ficheros de usuario?*

Hay más comandos interesantes sobre usuarios y grupos: `usermod`, `groupmod`, `chfn`, `chsh`, y, naturalmente, `passwd`. Haceros a una idea de para que sirven (probad a ejecutarlos o a echarle un vistazo a la página de `man`).

Pregunta 1.18: *¿Para que sirven los comandos* `usermod`*,* `groupmod`*,* `chfn`*,* `chsh` *y* `passwd`*?*

Fijaros que los comandos `adduser` y `deluser` utilizan los comandos de más bajo nivel `useradd` y `userdel`. Estos se pueden usar para realizar tareas más delicadas o más automatizadas. Vamos primero a crear un usuario mediante `useradd`:

```
useradd -m prueba2
```

Pregunta 1.19: *¿Para que sirve la opción* `-m`*?*

Podéis ver ahora que tenemos el nuevo usuario creado (aunque sin contraseña), podemos cambiar al nuevo usuario mediante `su prueba2` o asignarle una contraseña mediante `passwd prueba2`.

Ahora vamos a ver como podríamos automatizar el proceso. En primer lugar cread un fichero con una lista con dos usuarios, por ejemplo, "prueba3" y "prueba4". A continuación crearemos un script que lea los usuarios de este fichero y los cree:

```
for i in `cat fichero`;
do
   useradd -m $i;
done;
```

Una vez que hayáis creado el script, guardadlo, dadle permisos de ejecución, y ejecutadlo. Si todo va bien aparecerán los nuevos usuarios que haya en el fichero. Vamos a analizar un poco el script. En realidad solo hay tres sentencias: el `for`, el `cat` y el `useradd`. El `for` dice que la variable `i` tomará todos los valores de la lista que hay detrás del `in` y hasta el `;`. Como veis en vez de una lista hay un `cat fichero` entre comillas simples invertidas (acentos abiertos, si preferís). La gracia son las comillas ya que le dicen a la consola de Linux (`bash`, ya hablaremos más de esto) que ejecute el comando que tenga dentro y substituya las propias comillas y lo que incluyen por la salida del comando. Es decir, las comillas transforman la primera linea en: `for i in prueba3 prueba4;`. Así que `i` toma dos valores, "prueba3" y "prueba4". A continuación, para cada valor de `i` se ejecuta lo que hay entre el `do` y el `;` siguiente, es decir, `useradd -m $i`. Como ya habréis adivinado, `$i` quiere decir el valor de la variable `i`. El `done` es simplemente la finalización del `for`.

Pregunta 1.20: *¿Que hace el comando* `cat `ls`*? ¿Por qué?*

Bien, ahora podéis echarle un vistazo a la herramienta gráfica de gestión de usuarios: está en sistema, administración, usuarios y grupos. Ejecutadla y fijaros que en el fondo solo son unos

cuantos botones asociados a todo lo que habéis acabado de ver. Muy cómodo para añadir un usuario pero incomodo para añadir muchos (para eso es mucho mejor nuestro pequeño script). De todas formas, hay comandos (de consola) que ya hacen algo muy parecido a nuestro script.

Pregunta 1.21: *¿Para que sirven los comandos* `newusers` *y* `chpasswd`*? Describid como podríais usarlos para crear 1000 usuarios a partir de un fichero texto que tuviera sus datos. ¿Seríais capaces de hacerlo? Podéis probar creando un fichero (cread más de uno si os hace falta) con 4 o 5 usuarios.*

1.6. Permisos.

Vamos a hablar ahora de permisos de ficheros. En Linux la seguridad se implementa usando permisos en todos los ficheros del sistema. Estos permisos nos indican si:

- El fichero se puede leer (`r`).
- El fichero se puede escribir (`w`).
- El fichero se puede ejecutar (`x`).

Y definen esas posibilidades para tres tipos de usuarios distintos:

- El dueño.
- Gente del mismo grupo que el fichero.
- Otra gente.

Estos permisos se pueden ver si ejecutáis la orden `ls -l` que ya vimos en la práctica anterior. La primera columna de los detalles contiene los permisos de ejecución. Como podéis ver hay una ristra de 10 caracteres. El primero de ellos contiene información sobre el fichero (por ejemplo, vale "d" si es un directorio o nada, es decir, "-", si es un fichero normal). Los nueve siguientes son los permisos, por este orden, de lectura, escritura y ejecución, para el dueño, el grupo y otra gente. Cada permiso puede valer una letra si está activado o, "-" si no está activado. A continuación, y después de un número que indica en cuanto sitios está el fichero (ya veremos mas sobre ello), aparecen, por orden, el dueño del fichero y el grupo al que pertenece.
El dueño del fichero (o todo aquel con permiso), puede cambiar los permisos de un fichero mediante el comando `chmod`. `chmod` tiene muchas formas de funcionar, pero las tres más típicas son (para ver más formas consultad `man`):

- `chmod +<letra> <fichero>`: da el permiso `<letra>` al `<fichero>`.
- `chmod -<letra> <fichero>`: quita el permiso `<letra>` al `<fichero>`.
- `chmod xxx <fichero>`: cambia los permisos de `fichero` a `xxx`, donde cada `x` es un número octal que representa los permisos de uno de los tres grupos calculado según la suma de los siguientes códigos:
  - 4 permiso de lectura.
  - 2 permiso de escritura.
  - 1 permiso de ejecución.

Probad ahora a cambiar al usuario "alumne" y leed el fichero "passwd". A continuación leed el fichero "shadow".

Pregunta 1.22: *¿Que ha pasado? ¿Porqué? ¿Que creéis que motiva la diferencia entre el fichero "passwd" y el "shadow"?*

Pregunta 1.23: *¿Como haríais para que el usuario "prueba" (y solo él además del dueño) pudiera leer un fichero del usuario "alumne"?*

Fijaros que los permisos de un fichero o de un directorio se pueden modificar también desde el entorno de ventanas con la opción propiedades y la pestaña permisos. Si picáis en permisos veréis un enlace gráfico a lo que podéis hacer en la consola. Eso si, si queréis cambiar los permisos de muchos ficheros, la opción `-R` (recursivamente) de `chmod` o el `*` seguramente os serán más útiles que el entorno gráfico.

Otra opción útil con un fichero es cambiarlo de dueño. El comando `chown` tiene la siguiente sintaxis:

```
chown usuario[:grupo] fichero
```

y cambia el usuario (y el grupo si se especifica) de `fichero`. Este comando es especialmente útil para el "root" ya que por motivos de seguridad es prácticamente el único que puede asignar ficheros de un usuario a otro.

Pregunta 1.24: *¿Como cambiaríais todos los ficheros y carpetas del directorio* `/home/prueba` *para que pertenecieran al usuario "prueba"?*