Universidad de Murcia

Facultad de Informática

Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

TÍTULO DE GRADO EN
CIENCIA E INGENIERÍA DE DATOS

Fundamentos de Computadores

Práctica 4: Sistema de ficheros en el *shell* de Linux

Boletines de prácticas

CURSO 2022 / 23

Índice general

I. Boletines de prácticas **2**
B4.1. Boletín 1: Manejo básico del sistema de ficheros desde el shell . . . 2
B4.1.1. Objetivos . . . 2
B4.1.2. Plan de trabajo . . . 2
B4.1.3. Algunas recomendaciones iniciales . . . 2
B4.1.4. Usuarios y grupos. Permisos. . . . 3
B4.1.5. Consulta y manipulación del sistema de ficheros . . . 5
B4.1.6. Ejercicios a realizar durante la sesión . . . 8
B4.1.7. Referencias . . . 15
B4.2. Boletín 2: Herramientas de gestión del sistema de ficheros. . . . 16
B4.2.1. Objetivos . . . 16
B4.2.2. Plan de trabajo . . . 16
B4.2.3. Gestión de los permisos . . . 16
B4.2.4. Búsqueda de ficheros . . . 17
B4.2.5. Búsqueda de texto en ficheros . . . 18
B4.2.6. Compresión y descompresión de ficheros . . . 18
B4.2.7. Enlaces . . . 19
B4.2.8. Espacio en disco . . . 20
B4.2.9. Ejercicios a realizar durante la sesión . . . 20

Boletines de prácticas

B4.1. Boletín 1: Manejo básico del sistema de ficheros desde el shell

B4.1.1. Objetivos

El objetivo de este boletín de prácticas es continuar familiarizando al alumno con la consulta y manipulación del sistema de ficheros a través de la interfaz de línea de comandos, introduciendo el uso de comodines como herramienta clave detrás del potencial del shell a la hora de listar, mover, copiar o borrar ficheros. Además, se pretende que el alumno adquiera soltura en la búsqueda de información en la documentación existente en las páginas del manual de los diferentes comandos. También se introducen conceptos tales como el propietario y grupo de un fichero, así como sus permisos.

B4.1.2. 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 de los ejercicios propuestos en el boletín, con supervisión del profesor, y completándolos en su caso como trabajo autónomo.

B4.1.3. Algunas recomendaciones iniciales

Recomendaciones sobre la nomenclatura de ficheros

- **Espacios en blanco.** Dado que la sintaxis del shell utiliza el espacio en blanco como separador entre los diferentes elementos que componen una orden (nombre del comando, opciones, parámetros), es una costumbre habitual entre los usuarios de Linux no usar espacios en blanco en el nombre de los ficheros o directorios; no es porque no esté permitido sino por comodidad, pues su uso nos obliga bien a entrecomillar el nombre del fichero/directorio o bien a anteponer la barra invertida `\` ante cada espacio en blanco que contenga el nombre (por ejemplo, `ls mi\ directorio`). Esto es necesario para que el shell no confunda el espacio del nombre del fichero con el separador habitual.
- **Caracteres especiales.** Los caracteres `/|\!?*<>&~()[];#`: también tienen un significado especial en la gramática `bash`, por lo que se recomienda igualmente evitar su uso a la hora de nombrar ficheros.
- **Mayúsculas y minúsculas:** `A` $\neq$ `a`. Ten en cuenta siempre que en Linux, los nombres de ficheros `Programa.c` y `programa.c` son distintos.
- **Uso del carácter `.` (punto)**. Puede aparecer tantas veces como queramos en el nombre de ficheros y directorios, y su uso suele ser por comodidad o convención, para diferenciar a modo de *extensiones* los ficheros del mismo tipo, por ejemplo, aquellos que serán abiertos con la misma aplicación (p.ej., `documento.pdf`). No obstante, también es perfectamente válido no incluir ningún punto en el nombre de un fichero o directorio (p.ej., `LEEME`). Aquellos ficheros o directorios cuyo nombre comience por el carácter punto se considerarán **ocultos** y no aparecerían en un listado normal, a menos que usemos la opción `-a` o `--all` del comando `ls`).

Recomendaciones a la hora de teclear comandos

- **Autocompletado.** Como se vio en anteriores prácticas, la pulsación tecla del tabulador, `Tab`, puede ayudarnos a ahorrar miles de pulsaciones de teclas, al autocompletar los nombres de los ficheros o directorios que pasamos como parámetros a nuestros comandos.
- **Historial de comandos.** Podemos hacer un uso más rápido del intérprete de órdenes haciendo uso de las teclas flecha arriba ↑ o abajo ↓ para navegar por las órdenes que ya hayamos introducido anteriormente y que se guardan en un historial. También podemos buscar en el historial con la combinación `CTRL+R` y tecleando una subcadena de una orden anterior, de forma que pulsando `CTRL+R` de nuevo podemos ir pasando al siguiente resultado de la búsqueda.
- **Copiar y pegar.** Podemos copiar el texto que previamente hayamos seleccionado en el terminal con la combinación `Ctrl+Mayúsculas+C`. Con `Ctrl+Mayúsculas+V` podemos pegar en el terminal lo que hayamos copiado previamente al portapapeles, ya sea desde el terminal o desde cualquier otro programa.
- **Obteniendo ayuda.** 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. El comando `man orden` muestra la documentación o manual de uso de la orden tecleada, del que se sale pulsando la tecla `q`. Lo habitual es buscar una determinada palabra clave en la documentación, para lo cual se utiliza `/` seguido de la cadena a buscar. Por otro lado, la mayoría de comandos acepta también la opción `--help`, que muestra un resumen sobre su uso así como las opciones más relevantes.

B4.1.4. Usuarios y grupos. Permisos.

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 para proteger y limitar el acceso a los datos de cada usuario, así como proteger y asegurar el correcto funcionamiento del propio sistema.

Existen tres tipos de usuarios:

1. Usuarios normales: Son individuos particulares que pueden acceder al sistema para hacer uso de los recursos que ofrece el mismo, sin disponer de los privilegios necesarios para administrarlo. Como indicador el *prompt* utiliza para ellos por defecto el símbolo `$` (dólar).
2. Usuarios de 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`, `bin`, `kvm`, 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 adminstrar su funcionamiento, incluyendo los grupos, usuarios, instalación de software, etc. Debido a su omnipotencia, al entrar como superusuario el *prompt* nos avisa de ello utilizando el signo `#`.

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.

- `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 superususario, aún estando conectados como usuario normal (por supuesto, y al igual que el comando anterior, exige contraseña).
- `exit` : Termina la sesión del usuario actual.

Grupos

Para poder administrar los permisos de acceso al sistema de ficheros de una forma más flexible, Linux permite la organización de usuarios en *grupos*, de forma que cada fichero o directorio, además de tener un usuario propietario, siempre pertenece a un determinado grupo. 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. Así pues, los usuarios de un sistema Linux se organizan en *grupos*, de forma que todos los miembros del mismo grupo disponen de los mismos privilegios para poder acceder a ficheros y servicios del sistema. Con el comando `groups` : podemos ver a qué grupos pertenecemos.

Superusuario

Una buena razón para aprender los fundamentos de la línea de comandos es que las instrucciones que solemos encontrar en la red suelen favorecer el uso de comandos del shell en lugar de una interfaz gráfica. Cuando esas instrucciones requieran cambios en tu máquina que vayan más allá de la modificación de unos pocos ficheros en tu directorio personal, inevitablemente te encontrarás con comandos que deben ser ejecutados como administrador de la máquina (o superusuario en la jerga de Unix). Antes de que empieces a ejecutar comandos arbitrarios que encuentres en algún rincón oscuro de Internet, vale la pena entender las implicaciones de ejecutar como administrador, para que puedas valorar mejor si son seguras de ejecutar o no.

El superusuario es un usuario con superpoderes, pues puede modificar o eliminar cualquier fichero en cualquier directorio del sistema, independientemente de quién sea su propietario; `root` puede reescribir las reglas del cortafuegos o iniciar servicios de red que podrían abrir la máquina a un ataque; o apagar la máquina aunque haya otras personas utilizándola. En resumen, el usuario `root` puede hacer prácticamente cualquier cosa, saltándose fácilmente las protecciones que se suelen poner en marcha para evitar que los usuarios se pasen de la raya. Por supuesto, una persona conectada como `root` es tan capaz de cometer errores como cualquier otra. Los anales de la historia de la informática están llenos de relatos en los que un comando mal escrito borra todo el sistema de ficheros o mata un servidor vital. Además, existe la posibilidad de un ataque malicioso: si un usuario se conecta como `root` y abandona su escritorio, no es demasiado difícil que un colega descontento se meta en su máquina y cause estragos.

Para tratar de minimizar la cantidad de tiempo que se pasa conectado como `root`, muchas distribuciones de Linux comenzaron a fomentar el uso del comando `su`. Este comando se describe como la abreviatura de “cambio de usuario”, y le permite cambiar a otro usuario en la máquina sin tener que salir y entrar de nuevo. Cuando se utiliza sin parámetros, asume que quieres cambiar al usuario `root`, pero puedes pasarle un nombre de usuario para cambiar a una cuenta de usuario específica.

En muchas distribuciones de Linux, como Ubuntu, la cuenta de `root` está deshabilitada por completo para no permitir sesiones de terminal de larga duración con poderes peligrosos. En su lugar, se requiere que el usuario solicite específicamente los derechos de superusuario en cada comando mediante `sudo` (que viene de “cambiar de usuario y hacer este comando”). `sudo` se utiliza antepuesto a un comando que debe ejecutarse con privilegios de superusuario. Se utiliza un fichero de configuración para definir qué usuarios pueden utilizar `sudo`, y qué comandos pueden ejecutar. Cuando se ejecuta, al usuario se le pide su propia contraseña, que se almacena en la memoria durante un período de tiempo (por defecto 15 minutos), por lo que si necesita ejecutar varios comandos de nivel de superusuario no se le pide continuamente que la escriba. En un sistema Ubuntu, el primer usuario creado cuando se instala el sistema se considera el superusuario.

B4.1.5. Consulta y manipulación del sistema de ficheros

Los distintos entornos de escritorio que podemos encontrar en cualquier distribución de Linux actual cuentan con exploradores de ficheros que permiten interactuar con el sistema de ficheros mediante una interfaz de usuario gráfica (GUI). Por ejemplo, en GNOME (el *desktop environment* por defecto que encontramos en varias distribuciones de Linux, como Ubuntu Linux 22.04), el programa explorador de ficheros se llama *Nautilus*. Si bien la interfaz que nos ofrece el shell no resulta tan intuitiva, merece la pena aprender los comandos de manejo del sistema de ficheros ya que estos nos ofrecen muchas más posibilidades de listado, búsqueda y manipulación, como veremos en este boletín.

A la hora de aprender a utilizar los principales comandos de manipulación del sistema de ficheros, resulta fundamental tener presente los conceptos de *rutas absolutas y relativas* vistos en una práctica anterior, ya que esto nos permite explotar una poderosa propiedad de la línea de comandos: **es posible operar con ficheros y directorios en ubicaciones totalmente diferentes sin importar en qué parte del sistema de ficheros nos encontremos**.

¿Qué hay por aquí?

Como vimos en una práctica anterior, el comando `ls` nos permite obtener diversa información sobre el contenido del sistema de ficheros. Si lo ejecutamos sin parámetros, este comando lista en formato "corto" las entradas del directorio de trabajo actual; en caso de que se le pasen como parámetro rutas a ficheros o directorios, `ls` mostrará el contenido de dichos directorios (o información sobre ficheros). Aunque existen muchas opciones a la hora de elegir qué información muestra `ls` y cómo lo hace (prueba a verlas con `man ls`), sin duda la opción más utilizada es `ls -l` ("ele"), que nos ofrece un "listado largo" con información detallada sobre cada entrada (fichero o directorio). Otras opciones habituales son: `-a` (o `--all`), que lista todos los ficheros incluyendo los ficheros ocultos (cuyo nombre comienza por el carácter punto `.`); `-R` (o `--recursive`), que lista recursivamente mostrando el contenido de todos los subdirectorios, a cualquier profundidad; `-S`, que lista en orden de tamaño, o `-t`, que lista en orden de fecha y hora de la última modificación.

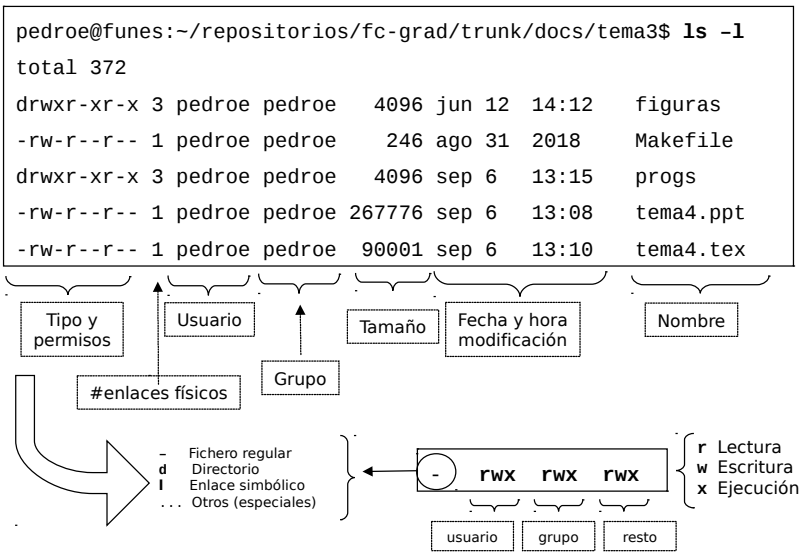

Figura I.1: Formato mostrado por el comando de listado largo `ls -l`.

Como muestra la figura I.1, en el listado en formato largo que obtenemos con `ls -l`, 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[1];

[1] Hay más tipos de ficheros y por tanto, más «letras» identificativas, pero quedan fuera el alcance de este curso.

continúa con los permisos; el número de enlaces físicos que tiene esa entrada; el usuario propietario del fichero (normalmente, el que lo creó); el grupo al que pertenece el fichero; el tamaño del fichero o directorio; su fecha y hora de última modificación y su nombre.

Los permisos disponibles para cada entrada son 9, agrupados de 3 en 3. Los 3 grupos se refieren, respectivamente, al *propietario del fichero*, a los *usuarios pertenecientes al grupo del fichero* y por último al *resto de usuarios del sistema*. Para cada uno de esos grupos se definen 3 permisos, llamados de *lectura* (`r`), *escritura* (`w`) y *ejecución* (`x`). Volveremos sobre los permisos en el siguiente boletín.

En el caso del tamaño de los directorios, cabe destacar que se refiere al tamaño que ocupa la *lista* de entradas (ficheros o directorios) que contiene, y no el tamaño de su contenido. De hecho, el tamaño de los directorios suele ser habitualmente 4 KiB, aunque puede crecer de 4 en 4 KiB dependiendo del número de entradas que contenga.

Uso de comodines

Una de las capacidades destacadas que ofrece cualquier *shell* es la posibilidad de utilizar caracteres especiales, llamados *comodines*, para manejar muchos ficheros de nombre similar a la vez a la hora de pasar parámetros a los diferentes comandos. Podemos combinar comodines para obtener listados múltiples acotados, sin necesidad de teclear una a una todas las entradas deseadas. 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.
- `[!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.
- `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 otra ubicación, ya sea manteniendo su nombre o con uno nuevo; o bien copia varios ficheros a un directorio, en este caso manteniendo sus nombres originales. También puede copiar directorios, incluyendo su contenido. Para ilustrar su uso, veamos varios ejemplos de utilización del comando `cp`. En los siguientes ejemplos, el directorio de trabajo actual contiene un directorio llamado `Escritorio`; por su parte, `/etc/hosts` y `/etc/hostname` son ficheros regulares, mientras que `/etc/init.d` es un directorio.
  - `cp /etc/hosts Escritorio` : Copia el fichero `/etc/hosts` al directorio `Escritorio` (que está en el directorio actual), manteniendo su nombre. Si el fichero `Escritorio/hosts` ya existe, sobreescribe su contenido.
  - `cp /etc/hosts Escritorio/hosts.backup` : Copia el fichero `/etc/hosts` al directorio `Escritorio`, con el nombre `hosts.backup`. Si ya existe un fichero con ese nombre, sobreescribe su contenido.
  - `cp /etc/hosts escritorio/hosts.backup` : Error, el directorio destino `escritorio` no existe.
  - `cp /etc/hosts /etc/hostname Escritorio` : Copia los ficheros `hosts` y `hostname` del directorio origen `/etc` al directorio destino `Escritorio` , manteniendo sus respectivos nombres.
  - `cp /etc/hosts /etc/hostname Escritorio/hosts` : Error, el destino `hosts` debe ser un directorio ya que hay más de un fichero origen.
  - `cp /etc/init.d /etc/hosts Escritorio` : Se omite la copia del directorio `init.d`, sin embargo sí se copia el fichero `hosts` al directorio destino.
  - `cp -r /etc/init.d Escritorio` : Copia recursivamente todo el contenido del subdirectorio `init.d` al directorio destino.
- `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.
- `rm fichero_1|directorio_1 ...` : El comando `rm` sirve para borrar ficheros o directorios.

  Advertencia importante. A diferencia de las interfaces gráficas, `rm` no mueve los archivos a una carpeta llamada "Papelera" o similar. En su lugar, los borra total, absoluta e irrevocablemente. Debes tener mucho cuidado con los parámetros que utilizas con `rm` para asegurarte de que sólo estás borrando el/los archivo/s que pretendes. Debes tener especial cuidado cuando utilices comodines, ya que es fácil borrar accidentalmente más archivos de los que pretendías. Un carácter de espacio erróneo en tu comando puede cambiarlo por completo: `rm t*` significa *eliminar todos los archivos que empiezan por t*, mientras que `rm t *` significa *eliminar el archivo t así como cualquier archivo cuyo nombre esté formado por cero o más caracteres*, ¡lo que sería todo

en el directorio! Si no está seguro, utilice la opción `-i` (interactiva) de `rm`, que le pedirá que confirme la eliminación de cada archivo; introduzca S para eliminarlo, N para conservarlo y pulse Ctrl-C para detener la operación por completo. En particular, debes tener muchísimo cuidado al usar `rm -rf` ya que esto borra de forma recursiva (`-r`), y sin confirmación (`-f`), los parámetros que se le pasan como parámetros (usualmente directorios).

En relación a la creación, borrado y copia de directorios, 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.

B4.1.6. Ejercicios a realizar durante la sesión

NOTA IMPORTANTE:
Antes de realizar los ejercicios, asegúrate de grabar la sesión con `script -a typescript_prac4_bol1`. Debes empezar cada ejercicio escribiendo el comentario de documentación `### EJERCICIO N ###` . Finalmente, no olvides añadir dicho fichero a tu repositorio-bitácora, respetando la organización de la misma en directorios y subdirectorios.

1. **Creación de directorios.** Como hemos visto, para crear un directorio se utiliza el comando `mkdir`, una abreviatura de "make directory". Recuerda escribir `### EJERCICIO 1 ###` antes de reproducir los siguientes pasos.

   a) En esta sección vamos a crear algunos ficheros reales con los que trabajar. Para evitar pisotear accidentalmente cualquiera de tus ficheros reales, vamos a empezar por crear un nuevo directorio, bien alejado de tu carpeta de inicio, que servirá como un entorno más seguro en el que experimentar:

   ```
   mkdir /tmp/tutorial
   cd /tmp/tutorial
   ```

   Observe el uso de una ruta absoluta, para asegurarse de que creamos el directorio del tutorial dentro de `/tmp`. Sin la barra al principio, el comando `mkdir` trataría de encontrar un directorio `tmp` dentro del directorio de trabajo actual, y luego trataría de crear un directorio tutorial dentro de él. Si no puede encontrar un directorio tmp, el comando fallará.

   b) Ahora que estamos seguros dentro de nuestra área de pruebas (comprueba con `pwd` si no estás seguro), vamos a crear algunos subdirectorios:

   ```
   mkdir dir1 dir2 dir3
   ```

   Hay algo diferente en este comando. Hasta ahora sólo hemos visto comandos que funcionan solos (`cd`, `pwd`) o que tienen un solo elemento después (`cd /`, `cd ~/Escritorio`). Pero esta vez hemos añadido tres cosas después del comando `mkdir`. Esas cosas se denominan parámetros o parámetros, y diferentes comandos pueden aceptar diferentes números de parámetros. El comando `mkdir` espera al menos un parámetro, mientras que el comando cd puede funcionar con cero o uno, pero no más.

   c) Mira lo que ocurre cuando intentas pasar un número incorrecto de parámetros a un comando:

```
mkdir
cd /etc ~/Desktop
```

d) Volvamos a nuestros nuevos directorios. El comando anterior habrá creado tres nuevos subdirectorios dentro de nuestra carpeta. Observa que `mkdir` ha creado todas los directorios en un solo directorio. No creó `dir3` dentro de `dir2` dentro de `dir1`, ni ninguna otra estructura anidada. Pero a veces es útil poder hacer exactamente eso, y `mkdir` tiene una manera:

```
mkdir -p dir4/dir5/dir6
ls
```

Esta vez verás que sólo se ha añadido `dir4` a la lista, porque `dir5` está dentro de ella, y dir6 está dentro de eso.

e) Más adelante instalaremos una herramienta útil para visualizar la estructura, pero ya tienes conocimientos suficientes para confirmarlo:

```
cd dir4
ls
cd dir5
ls
cd ../..
```

f) La opción `-p` que hemos utilizado significa "crear también los directorios padre". Las opciones se utilizan para modificar la forma en que opera un comando, permitiendo que un mismo comando se comporte de diferentes maneras. Desafortunadamente, debido a las peculiaridades de la historia y de la naturaleza humana, las opciones pueden tomar diferentes formas en diferentes comandos. A menudo las verás como caracteres simples precedidos por un guión (como en este caso), o como palabras más largas precedidas por dos guiones. La forma de un solo carácter permite combinar varias opciones, aunque no todos los comandos lo aceptan. Y para confundir aún más las cosas, algunos comandos no identifican claramente sus opciones en absoluto, ¡si algo es o no una opción está dictado puramente por el orden de los parámetros! No tienes que preocuparte por todas las posibilidades, sólo saber que las opciones existen y que pueden tomar varias formas diferentes. Por ejemplo, todas las siguientes significan exactamente lo mismo:

```
# No los escriba, sólo están aquí para fines demostrativos
mkdir --parents --verbose dir4/dir5
mkdir -p --verbose dir4/dir5
mkdir -p -v dir4/dir5
mkdir -pv dir4/dir5
```

g) Ahora sabemos cómo crear múltiples directorios simplemente pasándolos como parámetros separados al comando mkdir. Pero supongamos que queremos crear un directorio con un espacio en el nombre. Intentémoslo:

```
mkdir otra carpeta
ls
```

Probablemente ni siquiera has necesitado teclearlo para adivinar lo que sucedería: dos nuevos directorios, una llamada `otra` y la otra llamada `carpeta`. Si quieres trabajar con espacios en los nombres de directorios o ficheros, tienes que *escaparlos*. No te preocupes, nadie está escapando de la cárcel; escapar es un término informático que se refiere al uso de códigos especiales para decirle al ordenador que trate determinados caracteres de forma diferente a la normal.

h) Introduce los siguientes comandos para probar diferentes formas de crear directorios con espacios en el nombre:

```
mkdir "carpeta 1"
mkdir "carpeta 2
mkdir carpeta\ 3
mkdir "carpeta 4" "carpeta 5"
mkdir -p "carpeta 6"/"carpeta 7"
ls
```

Aunque la línea de comandos puede utilizarse para trabajar con ficheros y directorios con espacios en sus nombres, la necesidad de escaparlos con comillas o barras invertidas dificulta un poco las cosas. A menudo se puede distinguir a una persona que utiliza mucho la línea de comandos sólo por los nombres de sus ficheros: tenderá a ceñirse a letras y números, y a utilizar guiones bajos ("_") o guiones ("-") en lugar de espacios.

2. **Creación de ficheros mediante redireccionamiento.** Recuerda escribir el comentario `### EJERCICIO 2 ###` antes de reproducir los siguientes pasos.

   a) Nuestra carpeta de demostración empieza a estar bastante llena de directorios, pero sin embargo carece por completo de ficheros. Vamos a remediarlo redirigiendo la salida de un comando para que, en lugar de imprimirse en la pantalla, acabe en un nuevo fichero. Primero, recuerda lo que el comando `ls` está mostrando actualmente; ahora, supongamos que queremos capturar la salida de ese comando como un fichero de texto que podamos consultar o manipular posteriormente. Todo lo que tenemos que hacer es añadir el carácter mayor que (">") al final de nuestra línea de comandos, seguido del nombre del fichero en el que escribir:

   ```
   ls > salida.txt
   ```

   Esta vez no se imprime nada en la pantalla, porque la salida se redirige a nuestro fichero.

   b) Si ejecuta `ls` por sí solo, deberías ver que el fichero `salida.txt` ha sido creado. Podemos usar el comando `cat` para ver su contenido:

   ```
   cat salida.txt
   ```

   De acuerdo, no es exactamente lo que se mostraba en la pantalla anteriormente, pero contiene todos los mismos datos, y está en un formato más útil para su posterior procesamiento.

   c) Veamos otro comando, `echo`:

   ```
   echo "Esto es una prueba"
   ```

   Sí, `echo` sólo imprime sus parámetros de nuevo (de ahí su nombre). Pero combínalo con una redirección y tendrás una forma de crear fácilmente pequeños ficheros de prueba:

   ```
   echo "Esto es una prueba" > test_1.txt
   echo "Esta es una segunda prueba" > test_2.txt
   echo "Esta es una tercera prueba" > test_3.txt
   ls
   ```

   d) Muestra con `cat` el contenido de cada uno de estos ficheros. Pero `cat` es más que un simple visor de ficheros: su nombre viene de "con**cat**enar" ya que se puede utilizar para *unir* su contenido. Si le pasas más de un nombre de fichero a cat, éste mostrará cada uno de ellos, uno tras otro, como un único bloque de texto:

   ```
   cat test_1.txt test_2.txt test_3.txt
   ```

e) Cuando quiera pasar varios nombres de fichero a un solo comando, existen algunos atajos útiles que pueden ahorrarle mucho trabajo si los ficheros tienen nombres similares. Un signo de interrogación ("`?`") puede utilizarse para indicar una ocurrencia de *cualquier carácter* dentro del nombre del fichero. Un asterisco ("`*`") puede utilizarse para indicar *cero o más caracteres*. A estos caracteres se les denomina **comodines**. Para entender mejor el uso de comodines, veamos un par de ejemplos de comandos que hacen lo mismo:

```
cat test_1.txt test_2.txt test_3.txt
cat test_?.txt
cat test_*
```

f) Si miras la salida de `ls`, te darás cuenta de que los únicos ficheros o directorios que empiezan por "t" son los tres ficheros de prueba que acabamos de crear, así que podrías simplificar aún más ese último comando a `cat t*`, que significa "concatenar todos los ficheros cuyos nombres empiezan por t y van seguidos de cero o más caracteres". Utilicemos esta capacidad para unir todos nuestros ficheros en un único fichero nuevo, y luego visualizarlo:

```
cat t* > combinado.txt
cat combinado.txt
```

g) ¿Qué crees que pasará si ejecutamos estos dos comandos por segunda vez? ¿Se quejará el ordenador porque el fichero ya existe? ¿Agregará el texto al fichero, de modo que contenga dos copias? ¿O lo reemplazará por completo? Pruébalo para ver qué pasa, pero para evitar escribir los comandos de nuevo puedes usar las teclas de flecha arriba y flecha abajo para moverte hacia adelante y hacia atrás en el historial de comandos que has usado. Pulsa la ↑ un par de veces para llegar al primer gato y pulsa Enter para ejecutarlo, luego haz lo mismo para llegar al segundo.

h) Como puedes ver, el fichero tiene el mismo aspecto. Esto no se debe a que no se haya tocado, sino a que el shell borra todo el contenido del fichero antes de escribir la salida de su comando `cat` en él. Debido a esto, debe tener mucho cuidado cuando utilice la redirección para asegurarse de que no sobrescribe accidentalmente un fichero que necesita. Si quiere añadir, en lugar de reemplazar, el contenido de los ficheros, duplique el carácter mayor que:

```
cat t* >> combinado.txt
echo "¡He añadido una línea!" >> combinado.txt
cat combinado.txt
```

i) Repite el primer `cat` unas cuantas veces más, utilizando la tecla ↑ para mayor comodidad. Luego añade algunas líneas más mediante unos cuantos comandos `echo`, hasta que su documento de texto sea tan grande que no quepa todo en el terminal a la vez cuando utilice `cat` para mostrarlo.

j) Para ver todo el fichero, ahora necesitamos usar un programa diferente, llamado paginador (porque muestra el fichero en una "página" a la vez). El paginador estándar de antaño se llamaba `more`, porque pone una línea de texto en la parte inferior de cada página que dice "–More–" para indicar que aún no has leído todo. Hoy en día hay un paginador mucho mejor que deberías usar en su lugar: como sustituye a `more`, los programadores decidieron llamarlo `less`.

```
less combinado.txt
```

Al ver un fichero a través de `less` puedes usar las teclas de ↑, ↓, subir y bajar página, inicio y fin para moverte por el fichero. Pruébalas para ver la diferencia entre ellas. Cuando haya terminado de ver su fichero, pulsa `q` para salir de `less` y volver a la línea de comandos.

k) *Mayúsculas y minúsculas.* Los sistemas Unix distinguen entre mayúsculas y minúsculas, es decir, consideran que "A.txt" y "a.txt" son dos ficheros diferentes. Si se ejecutan las siguientes líneas, se obtendrán tres ficheros:

```
echo "Minúsculas" > a.txt
echo "Mayúsculas" > A.TXT
echo "Mayúsculas y minúsculas" > A.txt
```

Por lo general, hay que tratar de evitar la creación de archivos y carpetas cuyo nombre sólo varía por el uso de mayúsculas. No sólo ayudará a evitar confusiones, sino que también evitará problemas al trabajar con diferentes sistemas operativos. Windows, por ejemplo, no distingue entre mayúsculas y minúsculas, por lo que trataría los tres nombres de archivo anteriores como si fueran un único archivo, lo que podría provocar la pérdida de datos u otros problemas. Puede que te sientas tentado a pulsar la tecla de bloqueo de mayúsculas y utilizar mayúsculas para todos tus nombres de archivo. Pero la gran mayoría de los comandos del shell están en minúsculas, por lo que acabarías teniendo que activarlas y desactivarlas con frecuencia mientras escribes. La mayoría de los usuarios experimentados de la línea de comandos tienden a usar principalmente nombres en minúsculas para sus archivos y directorios, de modo que rara vez tienen que preocuparse por los conflictos de nombres de archivos.

3. **Movimiento y manipulación de ficheros.** Ahora que tenemos unos cuantos ficheros, veamos el tipo de tareas cotidianas que puedes necesitar realizar en ellos. En la práctica, lo más probable es que utilices un programa gráfico cuando quieras mover, renombrar o eliminar uno o dos ficheros, pero saber cómo hacerlo utilizando la línea de comandos puede ser útil para cambios masivos, o cuando los ficheros están repartidos en diferentes carpetas. De paso aprenderás algunas cosas más sobre la línea de comandos.

   a) Comencemos poniendo nuestro archivo `combinado.txt` en nuestro directorio `dir1`, usando el comando `mv` (mover):

   ```
   mv combinado.txt dir1
   ```

   Puedes confirmar que el trabajo se ha hecho usando `ls` para ver que falta en el directorio de trabajo, luego `cd dir1` para cambiar a `dir1`, `ls` para ver que está allí, luego `cd ..` para mover el directorio de trabajo de nuevo. O puedes ahorrarte un montón de tecleo pasando una ruta directamente al comando `ls` para llegar directamente a la confirmación que estás buscando:

   ```
   ls dir1
   ```

   b) Ahora supongamos que resulta que ese archivo no debería estar en `dir1` después de todo. Volvamos a moverlo al directorio de trabajo. Podríamos hacer un `cd` en `dir1` y luego usar `mv combinado.txt ..` para decir "mover combinado.txt al directorio padre". Pero podemos usar otro atajo de ruta para evitar cambiar de directorio. De la misma manera que dos puntos (`..`) representan el directorio padre, se puede utilizar un solo punto (`.`) para representar el directorio de trabajo actual. Como sabemos que sólo hay un archivo en `dir1`, también podemos usar `*` para que coincida con cualquier nombre de archivo en ese directorio, ahorrándonos unas cuantas pulsaciones más. Nuestro comando para mover el archivo de vuelta al directorio de trabajo por lo tanto se convierte en esto (tenga en cuenta el espacio antes del punto, hay dos parámetros que se pasan a `mv`):

   ```
   mv dir1/* .
   ```

   c) El comando `mv` también nos permite mover más de un archivo a la vez. Si pasas más de dos parámetros, el último se toma como el directorio de destino y los demás se consideran ficheros (o directorios) a mover. Utilicemos un solo comando para mover `combinado.txt`, todos nuestros ficheros `test_n.txt` y `dir3`, al directorio destino `dir2`.

   ```
   mv combinado.txt test_* dir3 dir2
   ls
   ls dir2
   ```

d) Con `combinado.txt` ahora movido a `dir2`, ¿qué pasa si decidimos que está en el lugar equivocado de nuevo? En lugar de en `dir2` debería haberse puesto en `dir6`, que es el que está dentro de `dir5`, que a su vez está en `dir4`. Con lo que ahora sabemos sobre rutas, eso tampoco es un problema. No olvides utilizar el tabulador para autocompletar las rutas en las siguientes órdenes:

```
mv dir2/combinado.txt dir4/dir5/dir6
ls dir2
ls dir4/dir5/dir6
```

Observa cómo nuestro comando `mv` nos permite mover el archivo de un directorio a otro, aunque nuestro directorio de trabajo sea algo completamente diferente.

e) Ya que parece que vamos a usar (y mover) mucho ese archivo, quizás deberíamos mantener una copia del mismo en nuestro directorio de trabajo. Al igual que el comando `mv` mueve los ficheros, el comando `cp` los copia (de nuevo, nótese el espacio antes del punto):

```
cp dir4/dir5/dir6/combinado.txt .
ls dir4/dir5/dir6
ls
```

Ahora vamos a crear otra copia del archivo, en nuestro directorio de trabajo pero con un nombre diferente. Podemos usar el comando `cp` de nuevo, pero en lugar de darle una ruta de directorio como último parámetro, le daremos un nuevo nombre de archivo:

```
cp combinado.txt backup_combinado.txt
ls
```

f) Quizás la elección del nombre de la copia de seguridad podría ser mejor. Por qué no renombrarlo para que siempre aparezca junto al archivo original en una lista ordenada. La línea de comandos tradicional de Unix maneja un renombramiento como si estuvieras moviendo el archivo de un nombre a otro, así que nuestro viejo amigo `mv` es el comando a utilizar. En este caso, sólo tienes que especificar dos parámetros: el archivo que quieres renombrar y el nuevo nombre que deseas utilizar.

```
mv backup_combinado.txt combinado_backup.txt
ls
```

Esto también funciona con los directorios, dándonos una forma de ordenar esos difíciles con espacios en el nombre que creamos antes.

g) Para evitar volver a escribir cada comando después del primero, utilice la flecha hacia arriba ↑ para sacar el comando anterior en el historial. A continuación, puedes editar el comando antes de ejecutarlo, moviendo el cursor a la izquierda y a la derecha con las teclas de flecha, y eliminando el carácter de la izquierda con la tecla de retroceso o el que el cursor tiene encima con `Supr`. Por último, escribe el nuevo carácter en su lugar y pulsa `Enter` para ejecutar el comando una vez que hayas terminado. Asegúrate de cambiar ambas apariciones del número en cada una de estas líneas.

```
mv "carpeta 1" carpeta_1
mv "carpeta 2" carpeta_2
mv "carpeta 3" carpeta_3
mv "carpeta 4" carpeta_4
mv "carpeta 5" carpeta_5
mv "carpeta 6" carpeta_6
ls
```

4. **Borrado de ficheros y directorios.** Ahora sabemos cómo mover, copiar y renombrar archivos y directorios. Sin embargo, dado que estos son sólo archivos de prueba, quizás no necesitemos realmente tres copias diferentes de `combinado.txt`.

   a) Pongamos un poco de orden, usando el comando `rm` (*remove*, eliminar).

   ```
   rm dir4/dir5/dir6/combinado.txt combinado_backup.txt
   ```

   Vamos a eliminar también algunos de esos directorios sobrantes con el comando `rmdir`:

   ```
   rmdir carpeta_*
   ```

   Si ejecutas `ls` verás que la mayoría de las carpetas han desaparecido, pero la `carpeta_6` todavía está por ahí. Como recordarás, la `carpeta_6` todavía tiene una `carpeta 7` dentro de ella, y `rmdir` sólo borrará las carpetas vacías. Es una pequeña red de seguridad para evitar que borres accidentalmente una carpeta llena de archivos cuando no era tu intención.

   b) La adición de opciones a nuestros comandos `rm` o `rmdir` nos permitirá realizar acciones peligrosas sin la ayuda de una red de seguridad. En el caso de `rmdir` podemos añadir una opción `-p` para decirle que elimine también los directorios padre. Piensa en ello como el contrapunto a `mkdir -p`. Así, si ejecutamos `rmdir -p dir1/dir2/dir3`, primero eliminará `dir3`, luego `dir2` y finalmente eliminará `dir1`. Sin embargo, sigue las reglas normales de `rmdir` de borrar sólo los directorios vacíos, por lo que si también hubiera un archivo en `dir1`, por ejemplo, sólo se eliminarían `dir3` y `dir2`.

   c) Cuando estás muy, muy, muy seguro de que quieres eliminar un directorio completo y cualquier cosa dentro de él, es común decirle a `rm` que trabaje recursivamente usando el parámetro `-r`, en cuyo caso eliminará felizmente carpetas así como archivos. Con eso en mente, aquí está el comando para deshacerse de `carpeta_6` y el subdirectorio dentro de ella:

   ```
   rm -r carpeta_6
   ls
   ```

   Recuerda: aunque `rm -r` es rápido y cómodo, también es peligroso. Lo más seguro es borrar explícitamente los archivos para limpiar un directorio, luego `cd ..` al padre antes de usar `rmdir` para eliminarlo.

5. **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 `nano` o `gedit` (GNOME). Intenta modificar dicho fichero, ¿puedes?. Podrás modificar `pracFC/group` pero no `/etc/group`, ya que este último es 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.

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.

6. **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 (p.ej., `gedit`) inicialmente vacío usando el botón central del ratón. Guarda el fichero generado en un fichero llamado `salida.txt` en el directorio *home*.

7. **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`.

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

   a) Repite el comando que listaba el fichero `group` de dos 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

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

   c) 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 `less`, y observa cómo el `bash` guarda ahí el historial de comandos para recordarlo en sesiones posteriores.

9. **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`.

B4.1.7. Referencias

- Traducción adaptada del tutorial:
  `https://ubuntu.com/tutorials/command-line-for-beginners`

B4.2. Boletín 2: Herramientas de gestión del sistema de ficheros.

B4.2.1. Objetivos

El objetivo de este boletín de prácticas es familiarizar a los alumnos con varias herramientas básicas de la línea de comandos que forman parte del repertorio básico de cualquier usuario, y que permiten desde la gestión de la metainformación de los ficheros o la búsqueda avanzada en el sistema de ficheros, hasta la búsqueda por contenido, la compresión y archivado, o la monitorización del espacio en disco. También se introducirá por primera vez el concepto de enlace y sus tipos.

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 de los ejercicios propuestos en el boletín, con supervisión del profesor, y completándolos en su caso como trabajo autónomo.

B4.2.3. Gestión de los permisos

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. 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[2].

- 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 los permisos de lectura y escritura no tiene relación con la lectura y modificación, respectivamente, del contenido en sí de los ficheros, puesto que éstos son permisos propios de cada fichero individual. Finalmente, el permiso de ejecución permite entrar dentro de un directorio (por ejemplo con el comando `cd directorio`) o utilizar este directorio en programas que necesiten acceder a sus contenidos como por ejemplo en búsquedas realizadas con el comando `find`.

  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

[2] 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.

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 no tienen ningún efecto (no permite modificar las entradas del directorio ya que no se tiene permiso para acceder al directorio). Sin embargo, 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--`.
- `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.

B4.2.4. Búsqueda de ficheros

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 nos permite realizar búsquedas con mucha precisión. El comando `find` busca recursivamente entradas (ficheros, directorios, etc.) en las ruta(s) especificada(s) como parámetro(s), que concuerden con la(s) expresión(es) de búsqueda especificadas.

El formato general es `find [ruta1 ruta2...] expresión`. Entre las opciones más empleadas como parte de la expresión de búsqueda, tenemos las siguientes:

- `-iname nombre` : Nombre del fichero a buscar, sin distinguir el uso de mayúsculas. Se pueden utilizar comodines, en cuyo caso se debe entrecomillas el patrón. También podemos utilizar `-name` si queremos que la búsqueda distinga entre mayúsculas y minúsculas.
- `-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.
- `-user usuario` : Propietario al que debe pertenecer el fichero.
- `-mtime [+|-]n` : Fichero modificado hace más de, menos de o exactamente n días.
- `-size [+|-]n` : Fichero con un tamaño de más, menos o exactamente $512 \times n$ bytes. Lo habitual es especificar una unidad de mayor tamaño, por ejemplo `-size +28k` para indicar ficheros que tengan un tamaño mayor de 28 KiB.
- `-printf "..."` : Sirve para imprimir cadenas de texto formateado junto con información acerca de los ficheros encontrados por find, tales con su ruta (`%p`), tamaño (`%s`), usuario propietario (`%u`), etc..

Los criterios de búsqueda usados en la expresión se pueden combinar con operadores lógicos: `!` para la negación; `-o` para la disyunción (“OR”); y `-a` para la conjunción (“AND”), que está implícito por defecto cuando se indican varios criterios). Veamos varios ejemplos concretos de uso del comando `find` con diferentes expresiones de búsqueda:

- `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 directorios cuyo nombre empiece exactamente con la cadena `dir_`.
- `find dir -size +140k -a -type f` : Busca en `dir` los ficheros de más de 140 KiB.
- `find . -type f -printf "El fichero %p ocupa %s bytes\n":`

B4.2.5. Búsqueda de texto en ficheros

Otra tarea recurrente al interactuar con el sistema de ficheros es encontrar determinados ficheros en función de su contenido, o bien localizar exactamente en qué lugar de un determinado fichero aparece una cierta palabra. El comando por excelencia para ello es `grep`, que es una contracción de *global regular expresion print*, y que se encarga de encontrar e imprimir las líneas de los ficheros que coinciden con un patrón dado.

El formato general es `grep [opciones] patrón [ruta...]`. Es conveniente entrecomillar el patrón, que puede contener caracteres literales y otros con signficado especial tales como: `^` (inicio de línea), `$` (fin de línea) o `.` (una ocurrencia de cualquier carácter). También es posible usar los corchetes para especificar un grupo/rango de caracteres. Por otro lado, entre las opciones más empleadas tenemos las siguientes:

- `-i` : Busca ignorando el uso de mayúsculas.
- `-v` : Invierte la búsqueda, selecciona las líneas que *no* concuerdan con el patrón dado.
- `-l` : Muestra el nombre del fichero que contiene la línea coincidente en vez de dicha línea.
- `-n` : Muestra el número de línea para cada línea coincidente, empezando por 1.
- `-w` : Busca únicamente palabras completas, no fragmentos de palabras.
- `-r` : Busca recursivamente en los directorios.

Veamos varios ejemplos concretos de uso del comando `grep` con diferentes opciones y patrones:

- `grep -rl "memoria" .`: Buscaría recursivamente la palabra exacta "memoria" en todos los ficheros que cuelgan del directorio de trabajo actual, a cualquier profundidad, y mostraría únicamente el nombre de aquellos ficheros en los que se encontraron coincidencias.
- `grep -in "total" /proc/meminfo`: Mostraría las líneas del fichero indicado que contienen la subcadena "total", sin tener en cuenta las mayúsculas, anteponiendo el número de línea.
- `grep " $" fich.txt`: Mostraría las líneas del fichero indicado que acaban con el carácter espacio.
- `grep "^[A-Z].s" fich.txt`: Mostraría las líneas del fichero indicado que empiezan con una letra mayúscula, seguida de cualquier carácter y a continuación la letra 's'.

B4.2.6. 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 sino que 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 empaquetados con `tar`

y comprimidos usando `Gzip` suelen tener extensión `.tar.gz` o, más abreviadamente, `.tgz`[3]. Éstas son las tres formas más comunes de uso de este comando, correspondientes a la *compresión*, el *listado* y la *descompresión*:

- `tar czvf result.tar.gz directorio[s]|fichero[s]` : Empaqueta (indicado por la opción `c`) y comprime (opción `z`) los directorios (recursivamente) y ficheros indicados como segundo y sucesivos parámetros, en un solo fichero `.tar.gz` (opción `f`), cuyo nombre se pasa como primer parámetro. Se guardan rutas relativas y permisos. Indicando la opción `v` muestra una descripción detallada del proceso de comprensión.
- `tar tzvf result.tar.gz` : Lista en el terminal los contenidos del fichero `result.tar.gz`. La opción `z` indica que los archivos se leen a través de opción `gzip`. Es recomendable usar esta opción debe ser usada para ficheros comprimidos con opción `gzip`.
- `tar xzvf result.tar.gz` : Descomprime y desempaqueta el fichero `result.tar.gz`, recuperando los ficheros, permisos y la estructura de directorios original. Si se desea extraer los ficheros en un directorio diferente al de trabajo actual se debe usar la opción `-C` como por ejemplo: `tar xzvf result.tar.gz -C ./otrodir`

B4.2.7. Enlaces

Crear enlaces a ficheros o directorios nos permite tener una copia "virtual" de un fichero o directorio. Si por ejemplo editamos el contenido del fichero enlazado, al abrir el enlace aparece tal modificación tanto en el fichero enlazado como en el original. Decimos "virtual" porque en realidad el enlace no ocupa un lugar adicional en el disco, es sólo un sistema para referenciar a otro fichero o directorio desde otro lugar del sistema de ficheros. Existen dos tipos de enlaces, físicos y simbólicos, que describimos a continuación.

Enlaces físicos o *duros*

Se crean con el comando `ln fichero_a_enlazar enlace`.

Los enlaces duros se caracterizan porque hay que borrar los enlaces y también el fichero enlazado para que desaparezca el fichero totalmente. De otra manera (es decir, mientras no borremos tanto todos los enlaces como el fichero enlazado original), tendremos en algún lugar una copia del fichero completo (y por tanto no se liberará el espacio correspondiente en el disco duro). Esto se debe a que cada enlace duro actua como una copia exacta del fichero enlazado (y por tanto se puede borrar el fichero enlazada y sus copias seguirán existiendo con su contenido).

Los enlaces duros se identifican por el número natural que sigue a los permisos en los listados largos realizados con `ls -l`. En un fichero normal, dicho número es siempre 1 (indicando que es copia única, ésto es, que si se borra dicho fichero ya se perderá definitivamente en el disco). Sin embargo, en un fichero sobre el que se han realizado $n-1$ enlaces duros dicho número es `n` (lo que indica que hay que borrar hasta n copias del mismo para terminar de borrarlo del disco duro).

Por ejemplo, éste podría ser el aspecto de un listado largo de nuestro directorio actual después de ejecutar el comando `ln fich_orig.txt fich_enlazado.txt` (donde suponemos que el fichero original ya existía, y no tenía inicialmente ningún enlace duro adicional):

```
[...]
-rw-rw-r-- 2 alumno alumno  31076 oct 28 16:59 fich_orig.txt
-rw-rw-r-- 2 alumno alumno  31076 oct 28 16:59 fich_enlazado.txt
[...]
```

Naturalmente, aunque por claridad en este ejemplo se han ubicado tanto el fichero original como el enlazado en el mismo subdirectorio, ésto no es en absoluto obligatorio.

[3] 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.

Enlaces simbólicos o *blandos*

Si deseamos hacer un enlace blando usaremos la opción `-s` del comando `ln`, es decir, el comando utilizado es en este caso `ln -s fichero_a_enlazar enlace`.

En el caso de los enlaces blandos (también llamados *simbólicos*), si borramos el fichero enlazado original los enlaces quedan "colgando" (apuntando a ningún sitio), y el contenido del fichero se pierde, puesto que el espacio en disco ocupado por los datos habrá sido definitivamente liberado. El enlace simbólico, sin embargo, puede ser borrado en cualquier momento sin afectar para nada al fichero original.

Los enlaces blandos se identifican al hacer un listado con formato largo (`ls -l`) porque en vez de aparecer en la primera columna un carácter `-` (como un fichero normal) o un carácter `d` (como un directorio), aparece un carácter `l` (de *link*, enlace en inglés). Por ejemplo, éste podría ser el aspecto de un listado largo de nuestro directorio actual después de ejecutar el comando `ln -s pelicula.avi enlace.avi`:

```
[...]
lrwxrwxrwx 1 rtitos rtitos      17 oct 28 19:03 enlace.avi -> pelicula.avi
-rw-rw-r-- 1 rtitos rtitos  996008 oct 28 19:02 pelicula.avi
[...]
```

De nuevo, y al igual que ocurría con los enlaces duros, no sólo no es obligado que el origen y el destino del enlace estén en el mismo subdirectorio, sino que, al contrario que en estos sencillos ejemplos, lo más habitual y útil es que estén ubicados en subdirectorios distintos.

B4.2.8. 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).

B4.2.9. Ejercicios a realizar durante la sesión

NOTA IMPORTANTE:
Antes de realizar los ejercicios, asegúrate de grabar la sesión con `script -a typescript_prac4_bol2`. Debes empezar cada ejercicio escribiendo el comentario de documentación `### EJERCICIO N ###` . Finalmente, no olvides añadir dicho fichero a tu repositorio-bitácora, respetando la organización de la misma en directorios y subdirectorios.

1. **Permisos sobre directorios.**

   a) Copia el contenido íntegro del directorio `/usr/bin` a un nuevo directorio llamado `binprogs` situado en tu directorio de inicio.

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

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

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

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

   f) Establece los permisos (755) al directorio `binprogs`.

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

2. **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 lo siguiente: (NOTA: *Recuerda encerrar el patrón a buscar entre comillas dobles si usas comodines con la opción -name*)

      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.

3. **Búsqueda de patrones en ficheros.**

   a) Utiliza el comando `grep` con la opción adecuada, y los comodines necesarios, para mostrar únicamente el nombre de aquellos ficheros del directorio `/usr/include` cuyo nombre acaba en ".h" y que contienen la cadena "leftover".

   b) Muestra el nombre de aquellos ficheros en `/usr/include` cuyo nombre empieza por "e" y que contienen la cadena "not" al final de la línea.

   c) Muestra el nombre de aquellos ficheros en `/usr/include` cuyo nombre empieza por "e" y que contienen la cadena "extern int" al principio de la línea.

   d) Muestra el nombre de aquellos ficheros en `/usr/include` cuyo nombre empieza por "r" y acaba en ".h" y que contienen la subcadena "byte", con cualquier uso de mayúsculas.

   e) Repite el comando anterior para que se muestre también el número de línea coincidente.

   f) Repite el comando anterior para que se busque únicamente la cadena "byte" cuando aparece como palabra completa (de nuevo sin atender al uso de mayúsculas).

   g) Muestra el nombre de todos los ficheros que cuelguen de `/usr/include` a cualquier nivel, que contengan la palabra "Beware", coincidiendo con este uso exacto de mayúsculas.

4. **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.tar.gz`, 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.

5. **Enlaces.**

a) Haz un listado largo del directorio `binprogs` creado en un ejercicio anterior, mostrando únicamente las entradas que empiezan por la letra "l". Observa el tipo de cada fichero mostrado.

b) Desde tu directorio de inicio, ejecuta la copia del programa `lsmod` que tienes en `binprogs` (*lsmod muestra el estado de los módulos del kernel de Linux que están actualmente cargados*).

c) Haz un listado largo del directorio `binprogs`, mostrando únicamente los ficheros y directorios que empiezan por la letra "k". Observa el número de enlaces de cada fichero mostrado.

d) Crea un enlace físico al fichero `binprogs/kmod` llamado `showkernelmodules`. Haz un listado largo para observar cuántos enlaces tiene el nuevo fichero creado.

e) Repite el último listado largo, y observa el número de enlaces físicos que ahora tiene `binprogs/kmod`.

f) Elimina el fichero `binprogs/kmod` y a continuación haz un listado largo del fichero `showkernelmodules`. ¿Cuántos enlaces hay ahora?

g) Ejecuta de nuevo `./binprogs/lsmod`. ¿Qué ocurre? Repite el listado largo del ejercicio anterior para mostrar los detalles del fichero `lsmod`.

h) Crea un fichero vacío con `touch` llamado `myfile` y a continuación edítalo con `nano` para añadir unas cuantas líneas, no importa el contenido, guardando los cambios y saliendo a continuación.

i) Crea un enlace físico a `myfile` llamado `hardlink_to_myfile`. Haz un listado largo de todos los ficheros cuyo nombre acabe en "file" y observa el número de enlaces.

j) Ahora, abre `hardlink_to_myfile` con el editor y borra parcialmente su contenido. Después, muestra el contenido de `myfile` con el comando `cat`. ¿Qué ha ocurrido?

k) Crea un enlace simbólico a `myfile` llamado `simlink_to_myfile` y utilízalo con el editor para borrar parcialmente su contenido. Después, muestra el contenido de `myfile` con el comando `cat`. ¿Qué ha ocurrido?

l) Elimina el fichero `myfile` y observa mediante un listado largo lo que ocurre con `simlink_to_myfile`. ¿Realmente hemos perdido completamente su contenido?

m) Renombra el enlace físico `hardlink_to_myfile` para que ahora se llame `myfile` y repite el listado largo para ver el estado del enlace simbólico. Prueba a mostrar el contenido del fichero pasando al comando `cat` el nombre del enlace simbólico.

n) Situado en el directorio `Escritorio`, crea un enlace simbólico llamado `downloads` al directorio `Descargas` de tu directorio de inicio de usuario, utilizando para ello una ruta relativa. A continuación, lista el contenido usando el enlace `downloads`. ¿Qué es lo que contiene?.

ñ) Mueve el enlace simbólico recién creado a tu directorio de inicio, y tras cambiarte a dicho directorio, haz un listado largo para observar el estado del enlace simbólico `downloads`. ¿Puedes usar dicho enlace para lista el contenido de `Descargas`?

o) Mueve de nuevo el enlace simbólico al directorio `Escritorio`, y comprueba que `ls downloads` funciona de nuevo. Ahora, cambia el nombre del directorio `Descargas` por `descargas`, y comprueba el estado del enlace simbólico.

p) Por último, crea un fichero cambia el nombre del fichero `myfile` por `testfile`, y repite el listado largo de todos los ficheros cuyo nombre acabe en "file". ¿Qué ha ocurrido con el enlace simbólico?