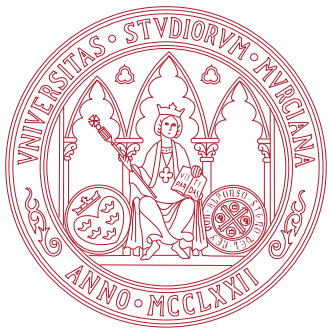

Universidad de Murcia

Facultad de Informática

Departamento de Ingeniería y Tecnología de Computadores

Área de Arquitectura y Tecnología de Computadores

Prácticas de

Introducción a los Sistemas Operativos

2º de Grado en Ingeniería Informática

Boletín de prácticas 1 – Manejo básico del shell de Linux

Curso 2021/2022

Índice

1. Introducción **2**
1.1. Objetivos . . . 2
1.2. Órdenes utilizadas . . . 2

2. Intérprete de órdenes en terminales de texto **2**

3. Jerarquía de directorios **3**

4. Listado de ficheros y directorios **5**

5. Uso de comodines **7**

6. Manejo de ficheros y directorios **8**

7. Permisos básicos de ficheros y directorios **11**

8. Búsqueda de ficheros **14**
8.1. Tests . . . 15
8.2. Acciones . . . 16
8.3. Opciones . . . 16

9. Buscando ayuda: páginas de manual y otras fuentes de información **17**

10. Ejercicios propuestos **19**

1. Introducción

En este boletín repasaremos algunos de los conceptos básicos del shell de Linux que se introdujeron en la asignatura «Fundamentos de Computadores» de primer curso.

1.1. Objetivos

Al terminar el boletín el alumno debe ser capaz de utilizar el intérprete de órdenes de Linux para:

- Interactuar con el sistema operativo.
- Consultar la jerarquía de directorios del sistema, navegar a través de ella y modificarla oportunamente.
- Distinguir entre ruta absoluta y relativa de un fichero o directorio.
- Consultar y modificar los permisos básicos de ficheros y directorios.
- Buscar ficheros que cumplan ciertos criterios y, opcionalmente, realizar operaciones sobre ellos.
- Acudir a las páginas de manual del sistema, así como a otras fuentes de información, para buscar ayuda sobre el manejo de una orden del shell.

1.2. Órdenes utilizadas

Las órdenes que usaremos en este boletín son:

- `cat`
- `cd`
- `chmod`
- `cp`
- `find`
- `help`
- `less`
- `ls`
- `man`
- `mkdir`
- `mv`
- `pwd`
- `rm`
- `rmdir`
- `touch`

En las páginas de manual de cada una de estas órdenes encontrarás información detallada de cómo usarlas.

2. Intérprete de órdenes en terminales de texto

La forma más básica y directa de interactuar con un sistema operativo como Linux es mediante un terminal de texto en donde el usuario introduce, línea a línea, órdenes directas desde el teclado. Estas órdenes serán procesadas por un *intérprete de órdenes*, llamado comúnmente *shell*.

En el mundo UNIX/Linux existen tres grandes familias de shells: `sh`, `csh` y `ksh`. Se diferencian entre sí, fundamentalmente, en la sintaxis de sus órdenes internas (las que implementa el propio shell) y en su interacción con el usuario. En estas prácticas nos centraremos en el uso del shell *Bash*, una variante libre de la familia `sh`.

El *prompt* es una indicación que muestra el intérprete para anunciar que espera una orden del usuario para ejecutarla a continuación. Dicha orden puede ser interna o externa. Como hemos dicho, las internas son aquellas que vienen incorporadas en el propio intérprete, como, por ejemplo, `cd` (esta orden se describe en la sección 3). Las externas, por el contrario, son programas separados como, por ejemplo, todos los programas que residen en los directorios `/usr/bin` (como `ls`) o `/usr/sbin` (como `mkfs`). El intérprete Bash suele mostrar un prompt como el siguiente:

```
[usuario@nombremáquina ~]$
```

donde `usuario` es el nombre del usuario conectado al sistema que hace uso de Bash y `nombremáquina` es el nombre del computador que se está usando.

De cara a tener esta comunicación con el intérprete de órdenes mediante una terminal de texto, tenemos varias opciones:

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

3. Jerarquía de directorios

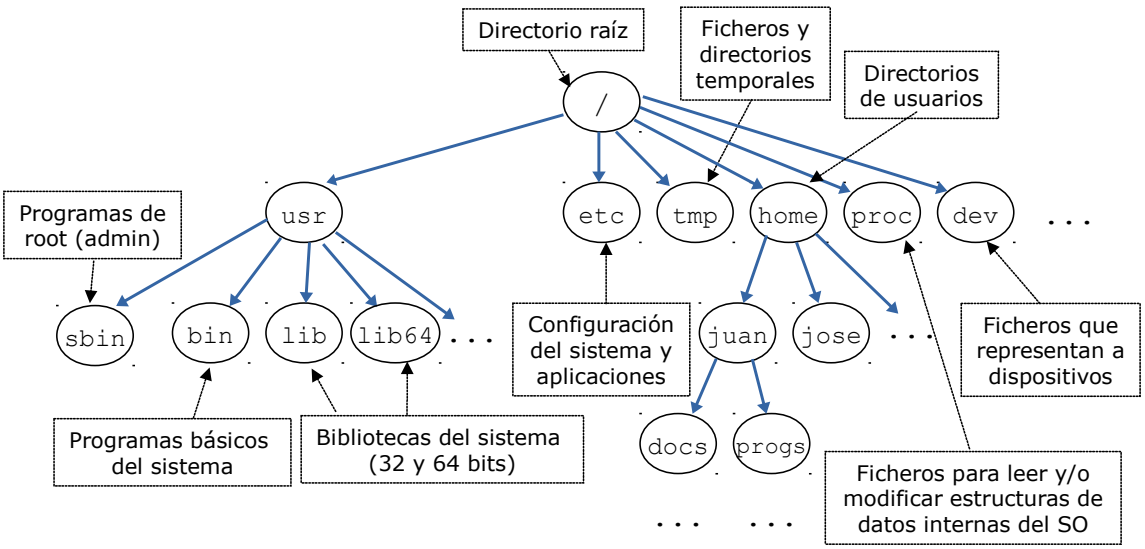

Figura 1: Jerarquía de directorios en Linux.

Un concepto fundamental en Linux es el de *fichero*. Como veremos a lo largo de la asignatura, hay muchos tipos de ficheros en Linux que se usan para fines de lo más diverso. De esos tipos, dos de los más importantes son los ficheros regulares y los directorios. Los *ficheros regulares* son aquellos ficheros que almacenan información que suele ser relevante para los usuarios en general, como código ejecutable de programas, texto, imágenes, etc. Los *directorios*, en cambio, son ficheros que se utilizan para organizar y llevar un registro de otros ficheros (veremos los detalles en el tema de teoría correspondiente).

Ficheros, ficheros regulares y directorios

Cuando en la asignatura hablemos de «ficheros» sin más debemos entender ficheros de cualquier tipo. Hablaremos, en cambio, de «ficheros regulares» y de «directorios» cuando la distinción sea importante. Con frecuencia también usaremos la expresión «ficheros y directorios» o «ficheros o directorios», y no solo «ficheros», para evitar que alguien piense únicamente en ficheros regulares cuando lo que se está describiendo puede aplicarse igualmente a otros ficheros, incluyendo directorios (el que finalmente pueda aplicarse o no sobre un cierto tipo de fichero dependerá de cada caso).

Linux distribuye sus ficheros y directorios en una estructura arbórea que parte de su *directorio raíz* y que se denomina *jerarquía de directorios*. En esta jerarquía, existen ciertos directorios predeterminados que merece la pena conocer para encontrar con rapidez y eficacia aplicaciones, bibliotecas, ficheros de configuración, datos de usuario y demás tipos de ficheros. La figura 1 esquematiza una jerarquía de directorios típica de Linux.

Cuando trabajamos con un intérprete de órdenes, siempre ejecutaremos las órdenes dadas en el prompt desde nuestro *directorio actual*, que es el directorio dentro de la jerarquía en el que nos encontramos en ese momento. Podemos conocer en cada instante el nombre de nuestro directorio actual ejecutando la orden `pwd`, y podemos cambiarlo con la orden `cd`, como veremos después.

El prompt también nos puede ayudar a saber dónde estamos, es decir, cuál es nuestro directorio actual. Por ejemplo, si, como antes, el prompt es:

```
[usuario@nombremáquina ~]$
```

entonces sabremos que nos encontramos en nuestro *directorio personal* (también llamado de inicio, de trabajo o *home*) porque, tras el nombre de la máquina, aparece el carácter `~`, que representa a dicho directorio. Debemos saber que cada usuario tiene su propio directorio personal a partir del cual puede almacenar y organizar su información a través de ficheros y directorios. Este directorio suele llamarse `/home/<usuario>` siendo `<usuario>` el nombre del usuario propietario. Por ejemplo, para el usuario `alumno`, su directorio personal sería `/home/alumno`.

Muchas de las órdenes que ejecutamos en un terminal trabajan sobre uno o más ficheros o directorios, por lo que se hace necesario poder indicar de alguna manera la ubicación de cada uno de ellos. Una forma de hacer esto es a través del «camino» a seguir en cada caso hasta encontrar el fichero o directorio necesario. Este camino se puede indicar por medio de una *ruta absoluta* o de una *ruta relativa*. Una ruta absoluta parte siempre del directorio raíz, por lo que siempre comienza con una barra inclinada (carácter `/`) que representa precisamente al directorio raíz. Cualquier ruta que no comienza por una barra inclinada es una ruta relativa y, por tanto, parte del directorio actual en el que nos encontramos.

A la hora de describir una ruta, se puede hacer uso del carácter punto, «.», para referirnos al directorio actual, y de dos caracteres punto seguidos, «..», para referirnos al directorio padre del directorio actual. Aquí debemos entender que el directorio actual es el que aparece justo antes de «.» o «..». Por ejemplo, si nuestro directorio actual es `/home/alumno` y tenemos la ruta relativa `../../etc`, entonces el primer «..» representará al directorio padre de `/home/alumno`, es decir, a `/home`, mientras que el segundo «..» representará al directorio padre de `/home`, o sea, al directorio raíz `/`.

Aunque los caracteres «.» y «..» pueden aparecer en cualquier lugar de una ruta, tanto absoluta como relativa, lo habitual es que aparezcan al principio de una ruta relativa.

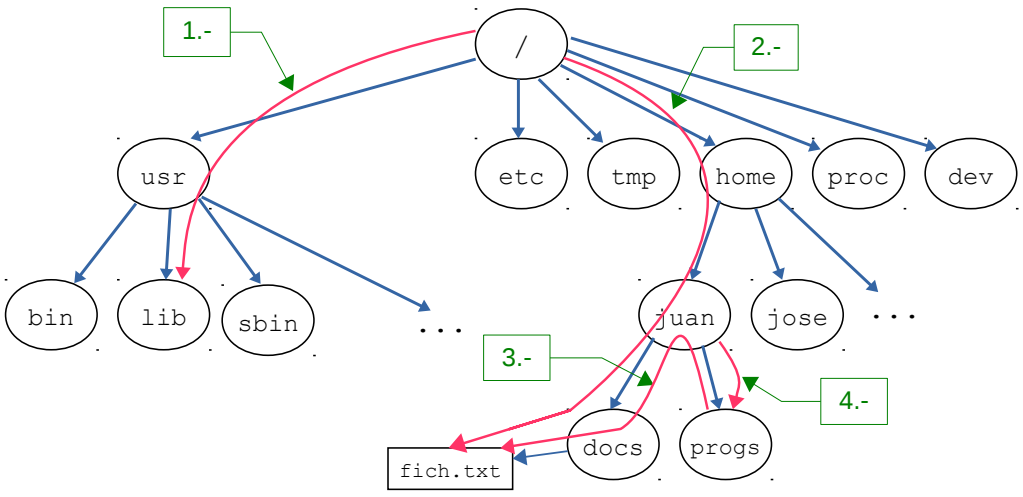

Figura 2: Ejemplos de rutas absolutas y relativas (ver texto).

He aquí algunos ejemplos de rutas absolutas y relativas, mostrados en la figura 2:

1. Ejemplo de ruta absoluta al directorio `/usr/lib`:
   `/usr/lib/`

2. Ejemplo de ruta absoluta al fichero `fich.txt` en el directorio `docs` bajo el directorio del usuario `juan`:
   `/home/juan/docs/fich.txt`

3. Ejemplo de ruta relativa al fichero `fich.txt` desde el directorio `progs`:
   `../docs/fich.txt`

4. Ejemplo de ruta relativa al directorio `progs` desde el directorio `juan`:
   `./progs/`
   o bien, simplemente:
   `progs/`

Para cambiar el directorio actual, debemos usar la orden `cd`, como ya hemos dicho. A continuación se muestran algunos usos de esta orden:

- `cd` *`ruta_directorio`*: cambia nuestro directorio actual al indicado con la ruta (relativa o absoluta).
- `cd` (sin parámetros): nos mueve a nuestro directorio personal (`/home/<usuario>`). También podemos usar `cd ~`, ya que Bash *expande*, es decir, sustituye el carácter «~» por la ruta absoluta de nuestro directorio personal antes de ejecutar la orden `cd`.
- `cd -`: nos mueve al último directorio en el que habíamos estado con anterioridad al actual.

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

```
$ cd /home/juan/progs
```

como una una ruta relativa:

```
$ cd ../progs
```

Observa que el `$` que precede a cada orden representa al prompt de nuestro intérprete de órdenes. Esta forma de indicar posibles órdenes a ejecutar la usaremos con frecuencia en los boletines de prácticas, incluyendo este.

4. Listado de ficheros y directorios

Aunque las distribuciones de Linux poseen navegadores de ficheros que permiten inspeccionar los directorios, e incluso copiar arrastrando con el ratón los ficheros de una ventana a otra del navegador, el terminal de Linux nos ofrece muchas más posibilidades de listado de ficheros de un modo más personalizado. Para ello se dispone de la orden `ls`, cuya sintaxis, según su página de manual, es:

```
ls [OPCIÓN]... [FICHERO]...
```

Sintaxis

En la sintaxis de una orden, los *corchetes* indican que algo es opcional (puede aparecer o no), mientras que los *puntos suspensivos* indican una repetición de lo que le precede, en un número indeterminado de veces. Así, según la sintaxis mostrada para `ls`, tras el nombre de la orden, podemos indicar una o más opciones (o ninguna) seguidas de uno o más ficheros (o ninguno).

Como vemos, en la sintaxis se habla únicamente de «fichero», por lo que debemos entender que el fichero puede ser de cualquier tipo, incluyendo directorios.

Sin ninguna opción, `ls` lista las entradas de los directorios dados como argumentos en formato corto (es decir, mostrando solo el nombre). Si aparecen ficheros en los argumentos, los nombres de estos también se muestran en el listado, a no ser que algún fichero sea en realidad un directorio, en cuyo caso lo que se muestra es su contenido y no su nombre (este comportamiento se puede modificar con la opción `-d`, descrita a continuación). Si no se pasa ningún argumento a `ls`, se lista el contenido del directorio actual, lo cual es equivalente a haber ejecutado «`ls .`».

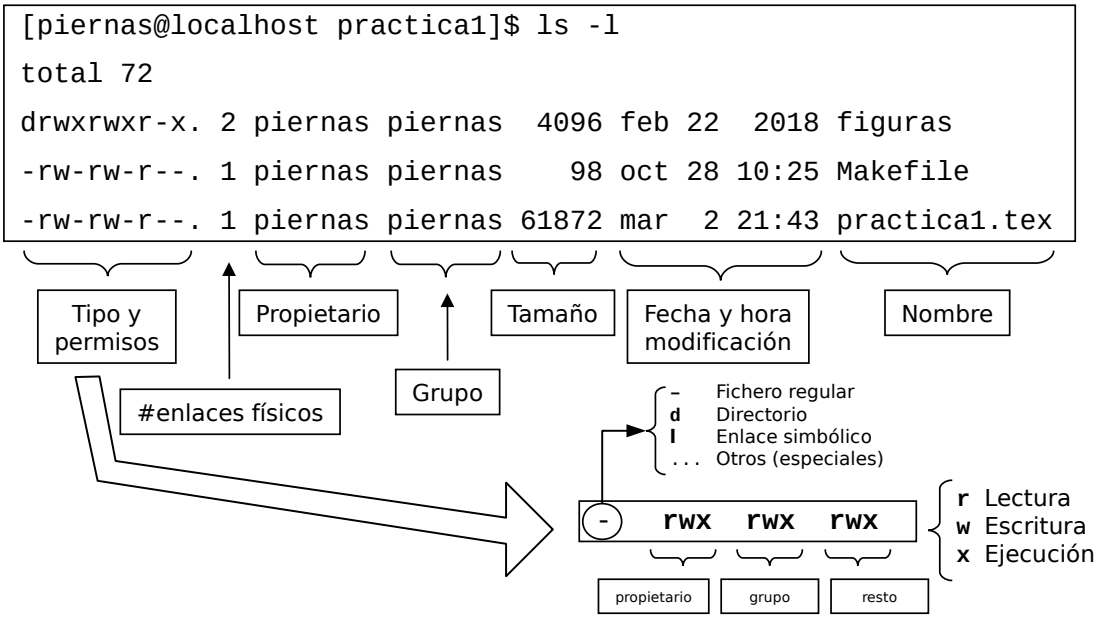

Figura 3: Formato mostrado por la orden de listado largo `ls -l`.

Entre las opciones más habituales de `ls`, tenemos las siguientes:

- `-l`: lista, en formato largo, las entradas del directorio indicado (ver figura 3). Cada línea se corresponde con una entrada. Si la línea empieza por `d`, se trata de un directorio, si empieza por un guion (`-`) es un fichero regular, si empieza por `l` es un enlace simbólico, ... Continúa con los permisos; el número de enlaces físicos que tiene esa entrada; el usuario al que pertenece el fichero; el grupo al que pertenece el fichero; el tamaño del fichero en bytes; fecha y hora de su última modificación (o solo la fecha con el año si el fichero se modificó hace más de 6 meses); y, finalmente, su nombre. Volveremos sobre los enlaces (físicos y simbólicos) y los permisos en boletines posteriores de este curso.

- `-R`: lista recursivamente todas las entradas (ficheros o subdirectorios) que cuelguen del directorio indicado a cualquier profundidad.

- `-a`: lista también los ficheros y directorios «ocultos», es decir, cuyo nombre comience por el carácter punto «`.`», por ejemplo, `.config` o `.bash_history`.

- `-S`: lista las entradas en orden decreciente de tamaño de los ficheros correspondientes.

- `-t`: lista las entradas en orden decreciente de fecha y hora de la última modificación, es decir, desde la entrada modificada recientemente hasta la entrada modificada hace más tiempo.

- `-r`: invierte el orden del listado de las entradas del directorio indicado según el criterio de ordenación elegido.

- `-d`: trata a los argumentos que son directorios como simples ficheros, por lo que la orden `ls` muestra información de los directorios en sí (sus permisos, tamaños, ...) y no de las entradas

que contienen dichos directorios (esto último es el comportamiento por defecto, como ya hemos explicado).

Estas opciones se pueden combinar en una misma orden `ls` y el efecto obtenido será la suma de los efectos producidos por cada una de ellas salvo, claro está, que haya dos opciones contradictorias, por ejemplo, opciones que indiquen distintas ordenaciones de las entradas, en cuyo caso predominará el efecto de la última opción dada.

Uso de opciones

En general, todas las órdenes que aceptan varias opciones simultáneamente (la inmensa mayoría) permiten dos sintaxis distintas para ello. La primera es ir poniendo las opciones una a una, precedidas por su correspondiente guion; por ejemplo: `ls -l -a -S -r`. La segunda consiste en juntar en una sola cadena todas las opciones, precedidas por un solo guion, como en `ls -laSr`, que es equivalente a la orden anterior. Como tanto en un caso como en otro el orden de las opciones no es importante (salvo que haya opciones contradictorias, como hemos explicado), las órdenes anteriores también se pueden escribir de otras muchas formas y, aun así, obtener el mismo resultado; por ejemplo: `ls -r -S -a -l`, `ls -Slra`, `ls -arSl`, etc.

5. Uso de comodines

Si a una orden como, por ejemplo, `ls`, le pasamos como parámetro una ruta (absoluta o relativa) indicando un solo nombre de entrada (fichero o directorio), el listado tomará en cuenta solo esa entrada. También pueden pasarse como parámetros varias entradas en secuencia (como en la orden `ls -l fich.txt fich.doc fich.avi`). Sin embargo, podemos usar comodines para especificar varios nombres de ficheros de una sola vez, sin necesidad de teclear una a una todas las entradas deseadas. Por ejemplo, si introducimos `ls -l fich.*`, la orden `ls` actuará sobre todas las entradas cuyo nombre comience por `fich.` seguido por cualquier cosa, incluida la cadena vacía.

Los comodines pueden usarse en cualquier lugar de la línea de órdenes, aunque lo más habitual es emplearlos en los listados a realizar con la orden `ls`, o bien con órdenes para copiar, mover o borrar ficheros, que veremos un poco más adelante.

El repertorio de comodines, con su significado, es el siguiente:

- `*`: cero o más caracteres cualesquiera.
- `?`: un carácter cualquiera.
- `[...]`: un único carácter, que puede ser cualquiera de los indicados entre corchetes. Por ejemplo, `[12a]` representa a un carácter que puede ser «1», «2» o «a», pero ningún otro.
- `[!...]`: un único carácter cualquiera que no sea uno de los indicados entre corchetes.
- {`nom1,nom2,...`}: cualquier elemento de la lista.

En el caso de los comodines `[...]` y `[!...]`, podemos indicar también rangos de caracteres. Así, `[2-5]` representa a un carácter que puede ser «2», «3», «4» o «5», mientras que `[!7-9]` representa a un carácter cualquiera (ya sea número o no) a excepción de «7», «8» y «9». Y lo mismo podemos hacer con rangos de letras del abecedario, por ejemplo `[a-c]`.

En ocasiones puede ser interesante representar a una letra cualquiera minúscula, a un carácter alfanumérico cualquiera, etc. Esto lo podemos hacer especificando la *clase* adecuada dentro de `[...]` o `[!...]` siguiendo la sintaxis `[:clase:]`. Así, `[[:lower:]]` representa a cualquier letra minúscula, `[[:upper:]]` representa a cualquier letra mayúscula, `[![:alnum:]]` equivale a cualquier carácter no alfanumérico (es decir, distinto de letra o número), etc. Podemos obtener un listado de clases disponibles consultando la página de manual de Bash.

Algunos ejemplos:

- `*.[ch]`: recogería nombres como `pepe.c`, `f1.c`, `hola.h`, `antonio.h`, ...
- `*.{c,h}`: equivale al patrón anterior.
- `{fi1,fi2}.[1-3]`: recogería los nombres `fi1.1`, `fi2.1`, `fi1.2`, `fi2.2`, `fi1.3` y `fi2.3`.
- `*.?`: recogería todos los nombres de ficheros con extensión de un carácter, como `f1.a`, `hola.c`, `adios.txt.c`, ...
- `*.[!c]`: recogería todos los nombres de 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`.
- `[[:upper:]]*`: representaría a todos los ficheros cuyo nombre empiece por una letra mayúscula.

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

Expansión de rutas

Es *muy importante* tener en cuenta que es el shell el encargado de efectuar la expansión de rutas con comodines antes de ejecutar la orden correspondiente. Es decir, cuando, por ejemplo, tecleamos la orden `cp *.c /tmp`, lo que se termina ejecutando es `cp fic.c calcula.c muestra.c /tmp` (suponiendo que `fic.c`, `calcula.c` y `muestra.c` son los únicos ficheros terminados en `.c` en el directorio actual). Por lo tanto, la orden `cp` no ve y no sabe de la existencia del comodín. Además de la expansión de rutas, hay otros tipos de expansiones (de variables, aritméticas, de parámetros, ...) que también realiza el intérprete de órdenes sin el conocimiento de las órdenes que finalmente se ejecutan y que son muy frecuentes en guiones shell, como ya veremos.

Un detalle a tener en cuenta a la hora de usar comodines es que, si no hay ningún fichero cuyo nombre cumpla con el patrón dado, no se llevará a cabo la expansión de rutas y la orden a ejecutar recibirá el patrón tal cual como argumento. Esto lo podemos ver fácilmente ejecutando la orden `ls` con un patrón que no va a tener coincidencias:

```
$ ls /dfas*
ls: no se puede acceder a '/dfas*': No existe el fichero o el directorio
```

Observa que, como no hay ningún fichero en el directorio raíz llamado `dfas*`, ni ningún fichero cuyo nombre comience por `dfas` en ese directorio, `ls` muestra el error que podemos ver.

6. Manejo de ficheros y directorios

A continuación se muestran las órdenes más comunes para la creación, copiado, cambio de nombre/directorio y borrado de ficheros y directorios:

La orden `touch` permite cambiar los diferentes tiempos asociados a un fichero existente (por defecto, actualiza el instante en el que se modificó un fichero al instante actual), pero, si se le pasa como parámetro un fichero que no existe, lo crea. Su sintaxis es:

```
touch [OPTION]... FILE...
```

Por ejemplo, podemos crear un fichero regular vacío ejecutando `touch ficherovacio`.

La orden `cp` permite copiar ficheros y directorios (usando la opción adecuada, como veremos ahora). Su sintaxis (que hemos simplificado ligeramente por sencillez) es:

```
cp [OPCIÓN]... ORIGEN DESTINO
cp [OPCIÓN]... ORIGEN... DIRECTORIO
```

Como vemos, la sintaxis varía en función de lo que se copia y cómo se copia. Así, la primera sintaxis se usa cuando se copia un único fichero o directorio, mientras que la segunda se usa cuando se copian varios ficheros y directorios. En este caso, el destino debe ser, necesariamente, un directorio existente.

Al hacer una copia, los nuevos ficheros y directorios pertenecerán al usuario que hace la copia, los tiempos asociados a ficheros y directorios (como el de modificación) serán los correspondientes al instante en el que finaliza la copia de cada elemento, los permisos dependerán de la configuración del sistema y del usuario, etc. Este comportamiento se puede cambiar con las opciones adecuadas (ver la opción `-a` más abajo).

Las opciones más interesantes para la orden `cp` son las siguientes:

- `-r` o `-R`: copia desde el(los) origen(orígenes) recursivamente. Usada para copiar directorios y sus contenidos. Observa que no hay diferencia entre usar `-r` y usar `-R`.
- `-i`: pide confirmación antes de sobrescribir un fichero existente.
- `-a`: equivale a las opciones `-dR --preserve=all` cuyo efecto es copiar recursivamente los orígenes conservando todos los atributos posibles, como propietario, grupo, permisos, tiempos, etc. Hay que tener en cuenta que, si la copia la hace un usuario no administrador, algunos atributos no se conservarán, como el propietario y el grupo.

He aquí varios ejemplos de utilización de la orden `cp`, incluyendo algunos casos de uso de comodines (suponemos que el directorio actual es `/home/juan/progs/` de la jerarquía de directorios mostrada en la figura 1):

- Copia el fichero `fich.txt` desde `/home/juan/docs` a mi directorio actual:

```
$ cp ../docs/fich.txt .
```

- Copia el fichero `fich.txt` (que acabamos de copiar en el directorio actual) a `/home/juan/docs`, pidiendo confirmación pues ya existe:

```
$ cp -i fich.txt ../docs
```

- Copia los ficheros `f1.c`, `f2.c` y `f3.c` desde `/home/juan/docs` a mi directorio actual:

```
$ cp ../docs/f1.c ../docs/f2.c ../docs/f3.c .
```

- Copia todos los ficheros del directorio `/home/juan/docs` cuyo nombre empiece por `F` a mi directorio actual:

```
$ cp ../docs/F* .
```

- Copia a mi directorio actual desde el directorio `/home/juan/docs` todos los ficheros cuyo nombre sea `Fich` seguido exactamente por un carácter cualquiera:

```
$ cp ../docs/Fich? .
```

- Copia recursivamente todo el directorio `/home/juan/docs` en el nuevo directorio `/home/juan/progs/copia_de_docs`:

```
$ cp -r ../docs copia_de_docs
```

La orden `mv` sirve para mover ficheros y directorios de un lugar a otro de la jerarquía de directorios. Cuando solo se mueve un elemento, esta orden también permite cambiarle el nombre. La sintaxis de `mv` (de nuevo simplificada ligeramente por sencillez) es:

```
mv [OPCIÓN]... ORIGEN DESTINO
mv [OPCIÓN]... ORIGEN... DIRECTORIO
```

Como vemos, la sintaxis es similar a la de la orden `cp` y debe interpretarse de la misma manera.

La orden `mv` admite varias opciones, siendo una de las más útiles la opción `-i`, que tiene el mismo significado que el que tiene en la orden `cp`, es decir, pide confirmación antes de sobrescribir un fichero existente. A continuación aparecen algunos ejemplos de uso de la orden `mv`:

- Mueve todos los ficheros cuyo nombre termina en `.txt` del directorio `/home/juan/docs` al directorio actual:

  ```
  $ mv /home/juan/docs/*.txt .
  ```

- Cambia, en el directorio actual, el nombre del fichero `memoria.odt` por el de `memoria_antigua.odt`:

  ```
  $ mv memoria.odt memoria_antigua.odt
  ```

- Mueve desde el directorio actual el fichero `memoria.odt` al directorio `/home/juan/progs/`, cambiándole además el nombre a este fichero por el de `memoria_antigua.odt`:

  ```
  $ mv memoria.odt /home/juan/progs/memoria_antigua.odt
  ```

La orden `rm` borra ficheros y directorios. Su sintaxis es:

```
rm [OPCIÓN]... FICHERO...
```

Las opciones más interesantes de `rm` son:

- `-i`: pide confirmación antes de borrar una entrada.
- `-r`: borra recursivamente directorios y sus contenidos.

Como ya veremos, es posibles definir un alias de una orden para que, al ejecutar dicho alias, se ejecute la orden dada con ciertas opciones. Es más, el alias de una orden puede llamarse igual que la orden, lo que permite ejecutar órdenes con ciertas opciones por defecto. En el caso de las órdenes `cp`, `mv` y `rm` que acabamos de ver, es típico que estos alias incluyan la opción `-i` para evitar la pérdida accidental de información. El problema es que, para operaciones sobre directorios en los que se usa la opción `-r`, el tener que confirmar muchas operaciones puede ser tedioso. Pensemos, por ejemplo, en el borrado de un directorio, donde habría que confirmar el borrado de todos y cada uno de los ficheros. Por eso, estas órdenes admiten también la opción `-f` que tiene como efecto anular a la opción `-i`. Así, si queremos borrar completamente un directorio y todo lo que contiene sin confirmación, deberíamos usar las opciones `-rf`.

Orden `rm -fr ...`

Aunque las opciones `-fr` de la orden `rm` son útiles para borrar directorios no vacíos completamente, hay que ser conscientes de que su poder de destrucción de información es tremendo. Por eso, *deberíamos pensárnoslo dos veces* antes de ejecutar una orden como esta.

En relación a la creación y borrado de directorios, las órdenes más relevantes son `mkdir` y `rmdir`, respectivamente, que tienen las siguientes sintaxis:

```
mkdir [OPCIÓN]... DIRECTORIO...
rmdir [OPCIÓN]... DIRECTORIO...
```

`mkdir` crea uno o más directorios vacíos, mientras que `rmdir` borra uno o más directorios vacíos. Si un directorio no está vacío, `rmdir` no lo podrá borrar. La solución pasa por borrar primero todas sus entradas, o usar la orden `rm` con la opción `-r` como se indicó anteriormente.

A continuación aparece un ejemplo en el que se usan algunas de las órdenes que acabamos de describir; observa que, al intentar borrar un directorio no vacío con `rmdir`, se produce un error; después, una vez que el directorio queda vacío, lo podemos borrar con `rmdir` sin problemas:

```
$ mkdir docs
$ cp -a /usr/share/dict docs/copia_dict
$ rmdir docs
rmdir: fallo al borrar 'docs': El directorio no está vacío
$ rm -fr docs/copia_dict
$ rmdir docs
```

Finalmente, existen varias órdenes que nos permiten mostrar el contenido de un fichero en el terminal sin necesidad de usar un editor. Las órdenes más útiles para ello son:

- `cat` *`fichero`*: muestra todo el contenido del fichero dado en el terminal.
- `less` *`fichero`*: muestra el contenido del fichero «página» a «página» (es decir, pantalla a pantalla), con posibilidad de avanzar y retroceder páginas en cualquier momento, y con posibilidad también de buscar palabras (por ejemplo, introduciendo `/palabra`).

7. Permisos básicos de ficheros y directorios

En el listado con formato largo de un directorio cualquiera obtenido con la orden `ls -l`, se muestran los permisos de cada entrada. Como se puede observar en la figura 3, los permisos disponibles para cada entrada son 9, agrupados de 3 en 3. Los 3 grupos se refieren, respectivamente, al propietario del fichero o directorio, a los usuarios pertenecientes al grupo del fichero o directorio, y 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`).

La interpretación de los permisos varía dependiendo de si se refieren a un fichero o a un directorio. En el caso de los ficheros, el significado de los permisos de lectura y escritura es obvio. El de lectura nos permite leer el contenido de un fichero, mientras que el escritura nos permite modificar el contenido de un fichero. El permiso de ejecución se debe añadir a un fichero solo cuando el contenido de este sea ejecutable (código máquina o lenguaje interpretado) y su presencia indicará que podremos ejecutar dicho contenido.

Aunque se permite cualquier combinación de los tres permisos, habitualmente un fichero tendrá permiso de lectura posiblemente combinado con los de escritura y ejecución, según el caso, pero será más atípico (aunque legítimo) que solo tenga permiso de escritura, pues la carencia del permiso de lectura impediría que un editor lo lea.

Para cambiar permisos de ficheros (y directorios) tenemos la orden `chmod`, una de cuyas posibles sintaxis es:

```
chmod [OPCIÓN]... MODO-EN-OCTAL FICHERO...
```

Como vemos, a los permisos se les llama *modo*.

Como los permisos del propietario, del grupo y del resto de usuarios son tres en cada caso, estos se pueden representar con 3 cifras octales. Por ejemplo, los permisos `rwx r-x ---` se pueden representar

en binario como `111 101 000`, donde hemos sustituido cada permiso por 1, indicando así que dicho permiso está presente, y cada guion por 0, indicando de esta forma que el permiso correspondiente no está presente. Dicho número binario es 750 en octal, por lo que 750 y `rwx r-x ---` representan los mismos permisos.

Supongamos ahora que somos el propietario del fichero `calcula.sh` y que este fichero pertenece al grupo de usuarios `alumnos`. Al ejecutar la orden `chmod 750 calcula.sh`, y por lo que acabamos de explicar, nos otorgaremos como propietarios todos los permisos sobre el fichero `calcula.sh` (`rwx`), mientras que los usuarios que pertenecen al grupo `alumnos` podrán leer este fichero y ejecutarlo, pero no modificarlo (`r-x`) y el resto de usuarios del sistema no podrá hacer ninguna operación con el fichero (`---`).

En el caso de los directorios, 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 con la orden `ls`. El permiso de escritura permite modificar las entradas del directorio (por ejemplo crear, borrar o renombrar ficheros o subdirectorios dentro de él). Finalmente, el permiso de ejecución permite acceder al nodo-i[1] asociado a cada entrada y, por ejemplo, es necesario para que una orden como `ls -l` funcione. Este permiso también es necesario para que el directorio sea nuestro directorio actual (moviéndonos a él con la orden `cd`) o para atravesarlo para acceder a sus subdirectorios (siempre que estos tengan, a su vez, los permisos adecuados).

Lo habitual es que un directorio tenga los permisos `rwx`, para poder hacer cualquier operación con sus entradas, o bien `r-x`, si queremos poder acceder al directorio y a su contenido, pero no queremos que sus entradas se modifiquen, es decir, no queremos que se puedan ni crear ni borrar ficheros o directorios, ni modificar sus nombres.

Además de esos permisos que son habituales en directorios, también son posibles otras combinaciones. Así, si habilitamos solo el permiso de lectura de un directorio (`r--`), podremos conocer sus entradas (nombres de ficheros y directorios), pero nada más (ni permisos, ni fechas, ni tamaños, etc.). Si habilitamos solo el permiso de ejecución (`--x`), no podremos conocer ni modificar las entradas del directorio, pero podremos atravesarlo para acceder a sus subdirectorios, siempre que conozcamos sus nombres y dichos subdirectorios tengan los permisos adecuados. Finalmente, si habilitamos los permisos de escritura y ejecución (`-wx`), no podremos consultar el contenido del directorio, pero sí podremos crear, borrar y renombrar ficheros y directorios dentro de él. Como podemos ver, el permiso de escritura debe ir acompañado del permiso de ejecución. De lo contrario, no podremos hacer nada con el directorio.

A continuación se muestra un ejemplo en el que se crea un directorio al que se le van cambiando sus permisos y sobre el que se ejecutan algunas operaciones. Dependiendo de los permisos, ciertas operaciones tendrán éxito mientras que otras fallarán debido a la falta de permisos:

```
$ mkdir directorio
$ chmod 700 directorio/
$ ls -ld directorio/
drwx------. 2 piernas piernas 4096 may 12 20:15 directorio/
$ touch directorio/fichero1
$ ls -l directorio/
total 0
-rw-rw-r--. 1 piernas piernas 0 may 12 20:15 fichero1
$ chmod 400 directorio/
$ ls -ld directorio/
dr--------. 2 piernas piernas 4096 may 12 20:15 directorio/
$ ls -l directorio/
ls: no se puede acceder a 'directorio/fichero': Permission denied
total 0
-????????? ? ? ? ?            ? fichero1
```

Observemos que, con solo el permiso de lectura, podemos conocer el nombre de las entradas del

[1] En los sistemas Unix/Linux, el nodo-i es una estructura de datos asociada a cada fichero o directorio que almacena cierta información sobre el fichero o directorio, como permisos, tiempos, propietario, etc. El contenido del fichero o directorio, sin embargo, suele almacenarse en bloques de disco y no en el propio nodo-i, como ya veremos.

directorio (`fichero1` en este caso), pero no tenemos acceso a su nodo-i y, por tanto, no podemos obtener sus permisos, tamaño, propietario, etc.

```
$ chmod 300 directorio/
$ ls -ld directorio/
d-wx------. 2 piernas piernas 4096 may 12 20:15 directorio/
$ touch directorio/fichero2
$ rm directorio/fichero1
rm: ¿borrar el fichero regular vacío 'directorio/fichero1'? (s/n) s
$ ls -l directorio/
ls: no se puede abrir el directorio 'directorio/': Permission denied
```

Ahora, con los permisos de escritura y ejecución, podemos crear y borrar ficheros, pero no ver el contenido del directorio.

```
$ chmod 100 directorio/
$ ls -ld directorio/
d--x------. 2 piernas piernas 4096 may 12 20:16 directorio/
$ cd directorio/
$ ls
ls: no se puede abrir el directorio '.': Permission denied
$ touch fichero3
touch: no se puede efectuar `touch' sobre 'fichero3': Permission denied
$ rm fichero2
rm: ¿borrar el fichero regular vacío 'fichero2'? (s/n) s
rm: no se puede borrar 'fichero2': Permission denied
```

Finalmente, con solo el permiso de ejecución, no podemos ni ver ni modificar el contenido del directorio, aunque sí podemos usarlo como directorio actual.

Es importante interpretar bien los permisos en ficheros y directorios para saber lo que es posible y lo que no. Específicamente, es importante darse cuenta de que el permiso de escritura de un directorio no tiene relación con la modificación del contenido en sí de los ficheros de ese directorio, puesto que dicha modificación depende del permiso de escritura propio de cada fichero. Por ejemplo, si un directorio no tiene permiso de escritura, pero uno de sus ficheros sí, podremos modificar el contenido de ese fichero, pero no borrarlo. También, si un directorio tiene permiso de escritura, pero uno de sus ficheros no, podremos borrar el fichero, pero no modificar su contenido. El siguiente ejemplo pone de manifiesto estas diferencias:

```
$ mkdir directorio
$ chmod 700 directorio/
$ ls -ld directorio/
drwx------. 2 piernas piernas 4096 may 12 20:40 directorio/
$ cp /etc/passwd directorio/
$ ls -l directorio/
total 4
-rw-r--r--. 1 piernas piernas 4006 may 12 20:40 passwd
```

Hasta aquí, hemos creado un directorio, le hemos dejado permisos `rwx` solo al propietario y hemos copiado el fichero `/etc/passwd` en él. Ahora, vamos a eliminar el permiso de escritura del directorio y a modificar el fichero que acabamos de copiar. La modificación la hacemos sobrescribiendo el fichero con otro mediante otra copia:

```
$ chmod 500 directorio/
$ ls -ld directorio/
dr-x------. 2 piernas piernas 4096 may 12 20:40 directorio/
$ cp /etc/fstab directorio/passwd
cp: ¿sobreescribir 'directorio/passwd'? (s/n) y
$ ls -l directorio/
total 4
-rw-r--r--. 1 piernas piernas 921 may 12 20:41 passwd
$ rm directorio/passwd
rm: ¿borrar el fichero regular 'directorio/passwd'? (s/n) y
rm: no se puede borrar 'directorio/passwd': Permission denied
```

Como vemos, hemos podido modificar el fichero, pero no borrarlo. Ahora, vamos a eliminar el permiso de escritura del fichero y vamos a restaurar dicho permiso en el directorio. Tras ello, intentaremos modificar el fichero y luego borrarlo:

```
$ chmod 400 directorio/passwd
$ ls -l directorio/
total 4
-r--------. 1 piernas piernas 921 may 12 20:41 passwd
$ chmod 700 directorio/
$ ls -ld directorio/
drwx------. 2 piernas piernas 4096 may 12 20:40 directorio/
$ cp /etc/motd directorio/passwd
cp: 'directorio/passwd' no escribible (modo 0400, r--------); ¿se intenta ...
                                                    ... a pesar de todo? (s/n) s
cp: no se puede crear el fichero regular 'directorio/passwd': Permission denied
$ rm directorio/passwd
rm: ¿borrar el fichero regular 'directorio/passwd'  protegido contra ...
                                                    ... escritura? (s/n) s
$ ls -ld directorio/
drwx------. 2 piernas piernas 4096 may 12 20:42 directorio/
```

Como se ve, no hemos podido modificar el fichero (sobrescribiéndolo con una copia de otro), pero sí hemos podido borrarlo.

8. Búsqueda de ficheros

La orden `find` se utiliza para la búsqueda de ficheros que cumplen ciertos criterios y, opcionalmente, realizar operaciones sobre ellos. Su sintaxis (que hemos simplificado por conveniencia) es:

```
find [punto-de-inicio...] [expresión]
```

Esta orden realiza, siguiendo la expresión dada, una búsqueda de ficheros en los subárboles de los directorios indicados como puntos de inicio. Una expresión está formada por una secuencia de diferentes elementos, entre los que destacan:

- Los *tests*: devuelven verdadero o falso según cierta propiedad del fichero considerado en cada momento. Por ejemplo, el test `-empty` devuelve verdadero solo cuando el fichero actual está vacío.
- Las *acciones*: como su propio nombre indica, llevan a cabo una cierta acción (como mostrar algo por la terminal) y devuelven verdadero o falso dependiendo de si la acción tiene éxito o no. Por ejemplo, la acción `-print` siempre devuelve verdadero y muestra el nombre del fichero actual.
- Los *operadores*: que sirven para juntar al resto de elementos de una expresión. Los tres principales operadores son `-a`, que representa un Y-lógico, `-o`, que representa un O-lógico, y `!`, que representa un NO-lógico. Si no se indica ningún operador, se entiende el operador `-a`. También es posible usar paréntesis para agrupar partes de una expresión antes o después de un operador. Los paréntesis tendremos que escribirlos como `\(...\)` para evitar que Bash los interprete de forma especial, como veremos en próximos boletines.

Es importante observar que una expresión, formada por diferentes tests, acciones y operadores, se evalúa de izquierda a derecha hasta que se conoce el resultado para el fichero que se está procesando actualmente. Esto significa que, en una operación *and*, la evaluación termina en el primer elemento que devuelve falso, o bien se llega al final de la expresión; en una operación *or*, en cambio, la evaluación termina en el primer elemento que devuelve verdadero, o bien se llega al final de la expresión. Así, si una expresión está formada por un test y dos acciones, la primera acción se ejecutará solo si el test ha devuelto verdadero, y la segunda acción se ejecutará únicamente si la primera acción se ha ejecutado y ha devuelto verdadero también (recordemos que, si no se indica operador, se supone `-a`).

Si no se indica un punto de inicio, se supone el directorio «.» que, como ya hemos visto, representa al directorio actual. Si no se indica ninguna expresión, se supone la acción `-print`.

Por ejemplo, la siguiente orden `find` (cuyo resultado aparece a continuación de la misma):

```
$ find .. -name '*.f'
../prog1.f
../stat/mean.f
../stat/var.f
../math/matrix.f
```

indica un punto de inicio, que es el directorio padre del directorio actual y viene representado por «..», y un test, que es `-name '*.f'`, el cual devuelve verdadero para todos aquellos ficheros cuyo nombre acabe en `.f`. Como no se indica ninguna acción, por defecto se supone también `-print`, por lo que se muestra el nombre de todos aquellos ficheros que cumplen el test. Observemos que el patrón dado a `-name` se ha protegido con comillas simples para evitar que Bash interprete el asterisco que aparece como un comodín y haga una expansión de rutas (ver sección 5). Hablaremos más de estos mecanismos de protección de Bash en siguientes boletines.

8.1. Tests

Los *tests* de búsqueda más comunes son (la parte subrayada hay que sustituirla por un valor adecuado, según el caso):

- `-name` patrón: para buscar ficheros cuyo nombre cumpla con el patrón dado. Este patrón sigue el mismo formato que el usado por Bash (ver sección 5), aunque con algunas excepciones (por ejemplo, la llaves no tienen ningún significado especial aquí).
- `-iname` patrón: igual que `-name`, pero sin distinguir entre mayúsculas y minúsculas.
- `-size [+|-]`n`[cwbkMG]`: para buscar ficheros cuyo tamaño sea mayor que `n` bloques (`+n`), exactamente `n` bloques (`n`) o menor que `n` bloques (`-n`). Normalmente, cada bloque es considerado de 512 bytes y equivale a poner `b` como unidad. Otras unidades posibles son `c` para bytes, `w` para palabras de dos bytes, `k` para kilobytes (1024 bytes), `M` para megabytes (1 048 576 bytes) y `G` para gigabytes (1 073 741 824 bytes).
- `-mtime [+|-]`n: para buscar ficheros que fueron modificados por última vez hace más de `n` días (`+n`), exactamente hace `n` días (`n`) o hace menos de `n` días (`-n`).
- `-atime [+|-]`n: igual que `-mtime`, pero teniendo en cuenta el tiempo del último acceso.
- `-ctime [+|-]`n: igual que `-mtime`, pero teniendo en cuenta el tiempo del último cambio de estado. En un fichero, un cambio de estado se produce cuando el fichero cambia de nombre y/o directorio, cambian sus permisos, etc.
- `-newer` fic: para buscar ficheros que hayan sido modificados más recientemente que el fichero dado.
- `-type` c: para buscar ficheros de tipo `c`, donde `c` puede ser «f» para ficheros regulares, «d» para directorios, «l» para enlaces simbólicos, etc.
- `-perm [-/]`permisos: para buscar ficheros cuyos permisos en este momento sean:
  - Exactamente los especificados en `permisos`.
  - Al menos todos los especificados en `permisos`, cuando `permisos` va precedido por «-».
  - Alguno de los especificados en `permisos`, cuando `permisos` va precedido por «/».

  Los permisos a buscar se pueden dar en octal, de la forma que se ha descrito en la sección 7.

Debemos tener en cuenta que, en el caso del test `-size`, los tamaños de los ficheros se redondean hacia arriba durante las búsquedas teniendo en cuenta la unidad usada. Así, por ejemplo, un fichero de 1 byte lo encontrarán las órdenes `find -size 1k` o `find -size 1M`, pero no las órdenes `find -size -1k`, `find -size -1M`, `find -size +1k` o `find -size +1M`.

La orden `find` tiene otros muchos tests que se pueden consultar en la sección «TESTS» de su página de manual.

8.2. Acciones

Las expresiones de acciones más comunes son:

- `-print`: siempre devuelve verdadero y muestra el nombre completo del fichero. Es la acción que se realiza por defecto si no se indica ninguna otra acción o si no se indica ninguna expresión.
- `-exec orden`: ejecuta la orden dada y devuelve verdadero o falso dependiendo de si la orden se ejecutó con éxito o no, respectivamente[2]. Es posible indicar el fichero que se está procesando actualmente usando la cadena «`{}`» (sin las comillas). El final de la orden se debe indicar con la cadena «`\;`» (de nuevo, sin las comillas). Por ejemplo, la siguiente orden borra todos los ficheros regulares modificados hace más de una semana a partir del directorio actual:

  ```
  $ find -type f -mtime +7 -exec rm {} \;
  ```

- `-ok orden`: igual que `-exec`, pero preguntando al usuario primero si desea ejecutar la orden o no sobre cada fichero encontrado. Si el usuario está de acuerdo, se ejecuta la orden. Si no, se devuelve falso.
- `-printf formato`: siempre devuelve verdadero e imprime, por cada fichero encontrado, una cadena de texto con formato, que puede incluir cierta información útil sobre dicho fichero. Está opción está basada en la función `printf` del lenguaje C, aunque hay que tener en cuenta que los especificadores de formato suelen tener un significado diferente aquí. De esta manera, por ejemplo, si en la cadena de texto incluimos `%s`, la orden nos mostrará el tamaño del fichero, con `%u` nos mostrará el usuario propietario del fichero, con `%p` nos mostrará la ruta completa del fichero, etc. En la explicación de `-printf` dentro de la sección «ACTIONS» de la página de manual de `find`, podemos encontrar una lista de todos los especificadores que se pueden indicar.

  Un ejemplo concreto de uso podría ser la siguiente orden, que busca ficheros que cuelguen del directorio actual, imprimiendo para cada uno de ellos una línea de la forma «`El fichero ejemplo.txt ocupa 1239 bytes`»:

  ```
  find -type f -printf 'El fichero %p ocupa %s bytes\n'
  ```

  Observa que, de nuevo, hemos usado comillas simples para evitar que Bash interprete el carácter % como un carácter especial.

8.3. Opciones

Además de tests, acciones y operadores, una expresión también puede contener opciones. Dos que pueden ser útiles son:

[2]Como ya veremos en la práctica «Introducción a la programación de *shell scripts* en Linux», cuando una orden termina su ejecución, devuelve un código de retorno o de salida a través del cual indica si su ejecución tuvo éxito o no. En `find`, la acción `-exec` devuelve «verdadero» cuando este código es 0 (lo que indica que la ejecución de la orden fue exitosa) y «falso» para cualquier valor distinto de 0 (lo que indica que la ejecución de la orden falló por algún motivo).

- `-depth`: procesa el contenido de cada directorio antes del directorio en sí. Esto supone que, si un directorio tiene subdirectorios, estos serán procesados antes que el resto de ficheros del directorio.
- `-maxdepth N`: desciende un máximo de `N` niveles de subdirectorios a partir de los directorios indicados como argumentos. Observa que `-maxdepth 1` hace que no se procesen los subdirectorios y `-maxdeph 0` hace que solo se procesen los ficheros dados como argumentos a `find` en la línea de órdenes, sean estos del tipo que sean.

9. Buscando ayuda: páginas de manual y otras fuentes de información

Tenemos diferentes formas de obtener ayuda en el uso del intérprete de órdenes:

- `orden --help`: muestra una ayuda breve sobre la orden dada. Es necesario que `orden` reconozca como válida la opción `--help`. De lo contrario, nos mostrará un error (o nada, dependiendo de la orden). Ejemplo:

```
$ date --help

Modo de empleo: date [OPCIÓN]... [+FORMATO]
       o bien:  date [-u|--utc|--universal] [MMDDhhmm[[SS]AA][.ss]]
Muestra la hora actual en el FORMATO dato, o establece la fecha del sistema.
...
```

- `help orden`: muestra una ayuda breve sobre una orden interna de Bash. Por ejemplo:

```
$ help pwd
pwd: pwd [-LP]
    Muestra el nombre del directorio de trabajo actual.

    Opciones:
      -L        muestra el valor de $PWD si nombra al directorio de
        trabajo actual
      -P        muestra el directorio físico, sin enlaces simbólicos

    Por defecto, `pwd' se comporta como si se especificara `-L'.

    Estado de Salida:
    Devuelve 0 a menos que se de una opción inválida o no se pueda leer
    el directorio actual.
```

- `man orden`: muestra un manual en línea de una orden (en general, de un tema). Por ejemplo:

```
$ man ps
NAME
       ps - report a snapshot of the current processes.

SYNOPSIS
       ps [options]

DESCRIPTION
       ps displays information about a selection of the active processes.
       If you want a repetitive update of the selection and the displayed
       information, use top(1) instead.
...
```

Dejamos de visualizar una página de manual pulsando la tecla «q».

Las páginas de manual están ordenadas en secciones que se identifican mediante números. Existe secciones para órdenes en general (1), para llamadas al sistema de Linux (2), para funciones de biblioteca como las que se utilizan en un programa en C (3), para ficheros de configuración (5), para órdenes de administración del sistema (8), etc.

Modos típicos de uso:

- `man <tema>`: cuando el tema que queremos consultar suele aparecer en una única sección. Ejemplos: `man ls`; `man strcpy`; `man stdio`.
- `man -a <tema>`: mostrará todas las páginas que hay sobre ese tema. Para avanzar de una a otra hay que salir primero de la página de manual actual pulsando la tecla «q». Por ejemplo, `man -a mount` mostrará todas las páginas de manual existentes del tema `mount`.
- `man -S<sección> <tema>`: para consultar la página de manual de un tema y una sección concreta. El número de sección aparece en la primera línea de la página entre paréntesis:

  ```
  MOUNT(8)                Linux Programmer's Manual                MOUNT(8)
  ```

  Un ejemplo de uso sería `man -S8 mount`.
- `man -k <cadena>` (o `apropos <cadena>`): sirve para buscar todas las entradas que contienen la cadena «cadena».
- `man -f <tema>` (o `whatis <tema>`): muestra un listado de todas las páginas de manual que hay para ese tema, indicando la sección en la que está y una breve descripción de la misma.

Además de estas herramientas, que incorpora el propio sistema operativo, en Internet podemos acudir a diversos medios para obtener ayuda:

- Documentación de las distribuciones:
  - Fedora: http://docs.fedoraproject.org
  - Ubuntu: https://help.ubuntu.com/
  - Debian: http://www.debian.org/doc/
  - etc.
- Sitios comunitarios (FAQ, foros, . . .).
- Proyectos de documentación libre, como el proyecto TLDP (http://www.tldp.org/).
- Etc.

Finalmente, no debemos olvidar las fuentes tradicionales de documentación y ayuda impresa que se detallan en la sección de bibliografía de este boletín.

Bibliografía

- Páginas de manual de las diferentes órdenes descritas y del intérprete de órdenes Bash.
- *Shell & Utilities: Detailed Toc.* The Open Group Base Specifications. http://pubs.opengroup.org/onlinepubs/9699919799/utilities/contents.html.
- *Linux: Domine la administración del sistema*, 2ª edición. Sébastien Rohaut. ISBN 9782746073425. Eni, 2012.

10. Ejercicios propuestos

1. Muestra la ruta absoluta del directorio en el que te encuentras actualmente.

2. Cámbiate al directorio `/etc` y comprueba que, efectivamente, tu directorio actual es ese.

3. Usando la opción `-l` de `ls`, lista todas las entradas del directorio `/etc` que empiezan por «a». Si alguna de esas entradas es un directorio, se debe mostrar información de la entrada como tal, no de las entradas que contiene el directorio.

4. Regresa a tu directorio personal y, sin moverte de él y usando una única orden, lista todas las entradas de los directorios `/usr/bin` y `/usr/sbin` cuyo nombre tenga exactamente 4 caracteres cualesquiera. Usa al menos una ruta relativa para uno de los dos directorios indicados.

5. Crea en tu directorio personal el directorio `configuracion` y, sin moverte de tu directorio personal, copia en él todo el directorio `/etc`, conservando todos los atributos posibles para cada fichero (tiempos, permisos, propietarios, etc.). Nota: observa que se producen algunos errores ya que hay ficheros que no se pueden copiar por falta de permisos.

6. Muévete a la copia del directorio `/etc` y, una vez en ella, busca recursivamente todas las entradas que sean directorios.

7. Borra recursivamente de la copia de `/etc` todos los ficheros regulares con un tamaño inferior a 100 KiB. Si usas la orden `rm`, evita que esta pida confirmación (podría hacerlo por diversos motivos en algunos casos).

8. Solo para los ficheros regulares del directorio actual (es decir, sin considerar subdirectorios), muestra su nombre y tamaño, siguiendo un formato similar al que se muestra a continuación:

   ```
   Nombre: ./services, tamaño: 692241 bytes
   Nombre: ./ld.so.cache, tamaño: 146569 bytes
   ...
   ```

9. Sin moverte del directorio en el que estás y usando rutas relativas, cambia el nombre del directorio `configuracion` que has creado anteriormente a `copia_etc`.

10. Vuelve a tu directorio personal e intenta borrar el directorio `copia_etc` usando la orden `rmdir`. ¿Puedes? Si no, usa la orden adecuada para hacer el borrado.

11. Muestra los nombres de los ficheros y directorios ocultos de tu directorio personal, sin que aparezcan ficheros o directorios que no estén ocultos. Pista: no uses la opción `-a` de la orden `ls` sino un patrón adecuado. Además, ten cuidado con los directorios ocultos. No nos interesa su contenido, sino solo su nombre.

12. Busca recursivamente a partir del directorio `/usr` todas las entradas cuyo nombre empiece por una letra mayúscula y acaben con un número.

13. Busca recursivamente a partir del directorio `/usr` todas las entradas cuyo nombre no acabe con un número.

14. Busca recursivamente a partir de los directorios `/etc`, `/usr/bin` y `/usr/sbin` todos los ficheros regulares que hayan sido accedidos hace más de 7 días.

15. Busca recursivamente a partir de tu directorio personal todos los ficheros que han sido modificados más recientemente que el fichero `~/.bash_profile` (recuerda que el carácter «~» representa a la ruta absoluta de tu directorio personal).

16. Busca recursivamente a partir del directorio `/etc` todos los ficheros regulares con un tamaño mayor que 5 KiB y menor que 30 KiB cuyo nombre comience por la letra «g». Los ficheros encontrados se tienen que copiar al directorio `temp` de tu directorio personal (crea dicho directorio previamente). Además, para cada fichero copiado, debe aparecer por pantalla un mensaje del tipo «`Copiado fichero de tamaño bytes`», siendo `fichero` un nombre de fichero copiado y `tamaño` su tamaño correspondiente en bytes.

17. Busca recursivamente a partir de tu directorio personal todos los ficheros regulares en los que haya algún permiso activo (`r`, `w` o `x`) para el grupo o para el resto de usuarios. En esos ficheros, deja los permisos adecuados para que el propietario solo pueda leer y escribir, y todos los demás usuarios queden sin permisos de cualquier tipo.

18. Busca recursivamente a partir de los directorios `/usr/bin` y `/usr/sbin` todos los ficheros regulares que pertenezcan al usuario `root` y para los que, sin embargo, dicho usuario no tenga permiso de escritura. Se debe mostrar un listado siguiendo un formato similar al que se muestra a continuación (busca en la página de manual de `find` como mostrar los permisos de un fichero):

```
Fichero: /usr/bin/staprun, permisos: ---s--x---
Fichero: /usr/bin/sudoreplay, permisos: ---x--x--x
...
```