Capítulo 3

Permisos

3.1. Introducción

En GNU/Linux cada fichero lleva asociados unos permisos que definen quién puede acceder a él y qué operaciones puede realizar. Es decir, mediante los permisos podemos hacer que un fichero no pueda ser leído por nadie o que otro pueda ser ejecutado por todo el mundo.

Existe un usuario especial en el sistema GNU/Linux, llamado **superusuario**, que puede leer o modificar cualquier fichero en el sistema, independientemente de los permisos que tenga asociados. El nombre de *login* `root` posee privilegios de superusuario; éste lo emplean los administradores del sistema para realizar tareas de mantenimiento.

Cuando un usuario inicia su sesión, teclea un nombre y después confirma que efectivamente es esa persona tecleando una contraseña (*password*). El nombre se conoce como **login_id**. El sistema en realidad reconoce al usuario por medio de un número llamado identificador del usuario **UID**. Además del UID, al usuario se le asigna un identificador de grupo, o **GID**, que lo coloca en cierta clase de usuarios.

Esta información la podemos obtener utilizando la orden `id`, que nos muestra el nombre de usuario (login_id), el UID, y los grupos a los que éste pertenece, junto con los GID que les corresponden.

El sistema mantiene toda la información que necesita conocer sobre cada usuario en el fichero `/etc/passwd`. Se trata de un fichero de texto que contiene una línea por cada usuario autorizado a utilizar el sistema. El formato de la línea es:

```
login_id:clave_cod:UID:GID:varios:dir_de_entrada:shell
```

El primer campo contiene el nombre de *login* del usuario. El segundo está reservado para la contraseña del usuario, apareciendo ésta codificada[1]. Cuando un usuario le da su contraseña al programa `login`, éste la codifica y compara el resultado con la que tiene almacenada en el fichero `/etc/passwd`. Si ambas coinciden, se permite al usuario iniciar la sesión. Los campos tercero y cuarto contienen el identificador numérico del usuario (UID) y el del **grupo principal** al que éste pertenece (GID).

El campo `varios` puede contener cualquier cosa, el nombre completo, dirección, número de teléfono, etc. Esta información se puede obtener mediante la orden `finger`.

El sexto campo indica el directorio de entrada del usuario y el campo `shell` le dice al sistema qué *shell* deberá ejecutarse cuando el usuario inicie la sesión; si este campo se deja vacío, se ejecutará el *shell* de Bourne.

Un usuario puede pertenecer a más de un grupo. El grupo que aparece en el fichero `/etc/passwd` se llama grupo principal y el resto de los grupos a los que pertenece un usuario, **grupos secundarios**. El fichero `/etc/group` contiene la lista de los grupos que existen en el sistema con el siguiente formato:

```
nombre_grupo:x:GID:lista_de_usuarios
```

El cuarto campo de este fichero contiene la lista de los usuarios que pertenecen a un grupo como secundario. Así el sistema sabe a qué grupos pertenece un usuario.

En un sistema donde esté activado NIS (*Network Information System*) el fichero donde se almacena la información de los usarios se encuentra en el directorio `/var/yp/src`. NIS permite que un conjunto de máquinas conectadas en red compartan la misma información acerca de los usuarios, de forma que un usuario del sistema podrá acceder a cualquiera de las máquinas con un mismo nombre de usuario y contraseña, ya que esta información está centralizada en una de las máquinas. Asimismo cada máquina mantiene sus ficheros `/etc/passwd` y `/etc/group` locales que contienen información sobre usuarios y grupos locales. Estos ficheros tienen como primer campo de la última línea el carácter `+` para indicar que el resto de usuarios están listados en el fichero `/var/yp/src/passwd` y el resto de los grupos en `/var/yp/src/group` en la máquina que mantiene la base de datos NIS (servidor NIS).

[1] Esto es lo tradicional en los sistemas UNIX, sin embargo esto puede llegar a ser peligroso para la seguridad del sistema y en la actualidad algunos sistemas ocultan esta información, guardándose la contraseña codificada en un fichero aparte al cual no tienen acceso los usuarios. En el lugar de la contraseña codificada aparece entonces un `*` en el fichero `/etc/passwd`.

3.2. Permisos de ficheros y directorios

Tanto los ficheros regulares como los directorios pueden llevar asociados tres tipos de permisos: **lectura**, **escritura** y **ejecución**. La orden `ls` seguida de la opción `-l` nos permite conocer los permisos de un fichero. Veamos un ejemplo:

```
$ ls -l /etc/passwd
-rw-r--r-- 1 root root 3215 Nov 28 13:26 /etc/passwd
```

En este caso hemos obtenido información del fichero `/etc/passwd`. La cadena `rw-r--r--` nos indica los permisos asociados al fichero. El permiso de lectura se representa mediante el carácter `r`, el de escritura mediante la `w` y el de ejecución mediante una `x`. Siempre que aparece uno de estos caracteres indica que el permiso está activado, si en su lugar aparece el carácter `-`, el permiso no está activado. Además, los usuarios pueden dividirse en tres clases: el propietario, el grupo y el resto de los usuarios. A cada clase de usuarios se le asocia un patrón del tipo `rwx` que indica lo que un usuario perteneciente a esa clase puede hacer con el fichero, como se muestra en la figura 3.1.

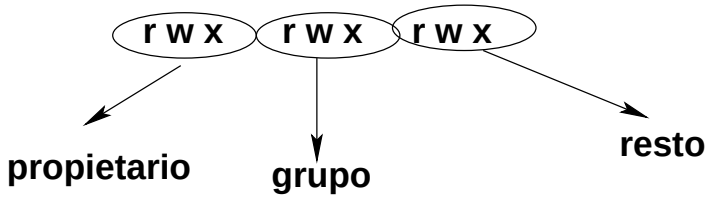


Figura 3.1: Permisos de un fichero

Para el fichero de nuestro ejemplo, el propietario (`root`) tiene permiso de lectura y escritura, los usuarios que pertenecen al grupo (`root`) sólo tienen permiso de lectura y lo mismo ocurre con el resto de los usuarios del sistema.

Si un fichero tiene el permiso de lectura activado se nos va a permitir examinar su contenido, si tiene activado el de escritura podremos modificarlo, y por último, el permiso de ejecución nos permite ejecutarlo, si se trata de un programa.

Los directorios también llevan permisos asociados. Se necesita permiso de lectura para ver el contenido de un directorio, permiso de escritura para añadir o borrar ficheros de un directorio, y permiso de ejecución para utilizarlo como parte de un camino.

Un directorio es un fichero que contiene información sobre otros ficheros. Esta información es básicamente una tabla de nombres de ficheros y su

localización en el disco (número índice).

`.`	104
`..`	200
`.profile`	2033
`bin`	130
`file1`	1025

Si pensamos que un directorio es un fichero que contiene esta información, podemos entender mejor cómo funcionan los permisos de lectura y escritura. El permiso de lectura nos permite ver el contenido del directorio; esto es justamente lo que hace la orden `ls`. El permiso de escritura permite a los programas realizar operaciones que alteran el fichero directorio. Borrar un fichero (`rm`), cambiar su nombre (`mv`) o crear uno nuevo, son operaciones que modifican el contenido del directorio y, por tanto, para realizarlas necesitamos permiso de escritura.

El permiso de ejecución de un directorio nos puede parecer un poco más extraño al principio. Se le suele llamar **permiso de búsqueda**, ya que nos permite buscar en el directorio el número de nodo índice que le corresponde al fichero, a partir de su nombre. Para ilustrar esto veamos cómo procede el sistema operativo GNU/Linux cuando se abre un fichero. Cuando se le proporciona un nombre de fichero, el sistema debe localizar a partir de éste sus bloques en el disco. Supongamos que le damos el nombre `/usr/ast/mbox`.

En primer lugar busca `usr` en el directorio raíz con el fin de hallar el nodo índice del fichero `/usr`. A partir de este nodo-i, el sistema localiza el directorio `/usr` y busca el siguiente componente en él, `ast`. Cuando ha encontrado la entrada de `ast`, ésta tiene el nodo-i del directorio `/usr/ast`. A partir de este nodo-i se puede hallar el contenido del directorio y buscar `buzon`. Con este nodo-i podemos acceder a los bloques del fichero deseado. Este proceso de búsqueda se ilustra en la figura 3.2.

Los nombres de ruta relativos se buscan de la misma forma que los absolutos, sólo que comenzando desde el directorio de trabajo y no desde el directorio raíz. Todo directorio tiene entradas para los ficheros `.` y `..` que se colocan ahí cuando se crea el directorio. La entrada `.` tiene el número de nodo-i del directorio actual y la entrada `..` tiene el número de nodo-i del directorio padre. Por tanto, un procedimiento que busca a `../fps/prog.c` simplemente busca a `..` en el directorio de trabajo, halla el número de nodo-i del directorio padre y rastrea ese directorio hasta encontrar `fps`.

Hay que hacer notar que sólo cuando hay que abrir un fichero entran en juego sus permisos. Las órdenes `rm` y `mv` sólo requieren permisos de búsqueda y escritura en el directorio; los permisos del fichero no importan. Esto es

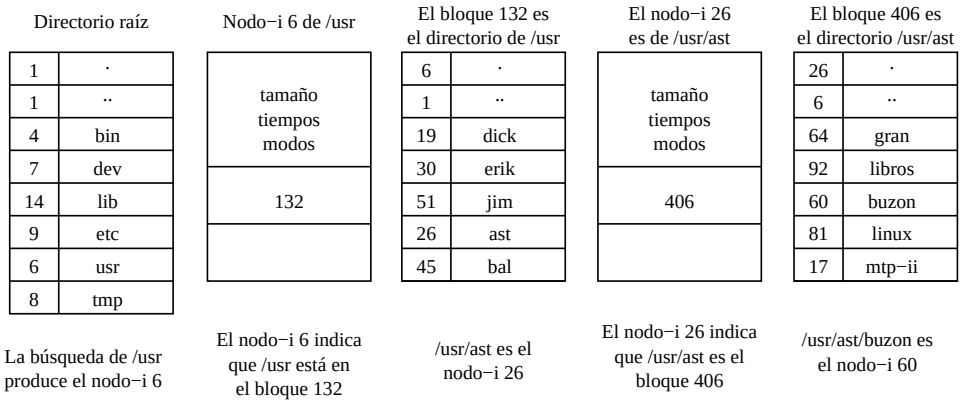


Figura 3.2: Las etapas en la búsqueda de `/usr/ast/buzon`

importante, porque alguien que no tenga acceso de lectura o escritura a nuestros ficheros puede borrarlos o reemplazarlos si tiene permiso de búsqueda y escritura en el directorio.

En el cuadro 3.1 se muestra un resumen de los permisos que deben tener activados los ficheros o directorios implicados para la realización de diferentes operaciones.

	Fichero origen	**Fichero destino**	**Directorio origen**	**Directorio destino**
`cp` (destino no existe)	r	-	x	wx
`cp` (destino existe)	r	w	x	x
`mv`	-	-	wx	wx
`ln`	-	-	x	wx
`ln -s`	-	-	-	wx
`rm`	-	-	wx	-

Cuadro 3.1: Permisos necesarios para hacer operaciones con ficheros

¿Qué permisos tiene activados un fichero cuando se crea? En principio un fichero regular tendrá activados los permisos de lectura y escritura para los tres tipos de usuarios, y los directorios tendrán activados todos los permisos (lectura, escritura y ejecución). Dado que esto no es conveniente desde el punto de vista de la seguridad del sistema, existe una orden que nos permite establecer limitaciones a los permisos de creación de los ficheros. Se trata de la orden `umask`, que estudiaremos con más detenimiento en el apartado 3.4. El administrador del sistema suele establecer mediante esta orden unos permisos adecuados para nuestros ficheros.

Ejercicios:

1. Suponga el siguiente fichero:

   ```
   ---xrwxrw- 1 juan alum 31 Jul 10 12:12 fichero
   ```

   ¿Qué operaciones puede realizar `juan` sobre `fichero`? ¿Y alguien que pertenezca al grupo `alum`?

3.3. Cambio de permisos

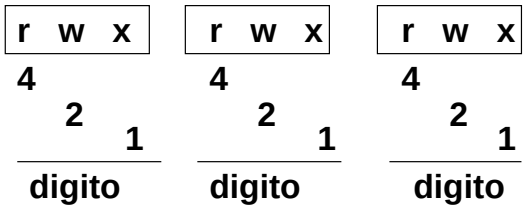
Para cambiar los permisos asociados a un fichero se utiliza la orden `chmod` cuyo formato es

`chmod` *modo fichero* ...

donde *modo* es la especificación de los permisos que se van a aplicar al *fichero*. El modo se puede especificar de dos formas, como números octales o por descripción simbólica; a continuación vamos a ver los dos métodos.

3.3.1. Números octales

El modo se representa como un número octal de tres dígitos, obteniéndose cada uno de ellos mediante la suma de los equivalentes numéricos de cada permiso.



Ejemplo:

```
$ chmod 640 prueba
```

Esta línea de órdenes hace que los permisos del fichero `prueba` *queden de la siguiente forma:* `rw-r---`

3.3.2. Descripción simbólica

El formato de la descripción simbólica del modo es:

[*quién*] *op permiso*

quién es una combinación de las letras `u` (para los permisos del propietario), `g` (grupo) y `o` (otros). Puede utilizarse una `a` para representar a los tres a la vez (`ugo`), y éste es el valor predeterminado.

op puede ser un `+` para añadir un permiso que actualmente no se tiene, un `-` para quitar un permiso que se tiene, o un `=` para asignar de forma absoluta un permiso.

permiso es una combinación de las letras `r`, `w`, `x`, `s` y `t` (`s` representa a los permisos *set user id* o *set group id* y la `t` al *sticky bit*, que estudiaremos más adelante).

Ejemplos:

1. ```
   $ chmod a=rw  prueba
   $ ls -l prueba
   -rw-rw-rw-   .............
   ```
   Se otorga a todos los usuarios los permisos de lectura y escritura.
2. ```
   $ chmod go-w prueba
   $ ls -l prueba
   -rw-r--r--   .............
   ```
   Al grupo y al resto de usuarios se le quitan el permiso de escritura.
3. ```
   $ chmod u+x,g+w prueba
   $ ls -l prueba
   -rwxrw-r-- .............
   ```
   Le damos permiso de ejecución al propietario y de escritura al grupo.

Hay que hacer notar que **los permisos de un fichero sólo pueden ser cambiados por su propietario o por el superusuario**.

Ejercicios:

1. ¿Qué significa que un fichero tenga los permisos 751? Represéntelos como una cadena de permisos. Dé los permisos de forma simbólica al fichero `prueba` para que tenga los permisos anteriores. ¿Cuáles son los requisitos para cambiar los permisos a un fichero?
   ```

2. Suponga que tenemos un fichero `permisos` sobre el que vamos realizando la siguiente secuencia de órdenes de forma consecutiva. Indique, de forma razonada, qué permisos tendrá tras la ejecución de cada una de ellas

   a) `$ chmod 365 permisos`

   b) `$ chmod g-r permisos`

   c) `$ chmod o=x permisos`

   d) `$ chmod u+r permisos`

3.4. La orden `umask`

La orden `umask` sirve para establecer los permisos con los que se crearán los ficheros y directorios a partir del momento en que la ejecutemos. El administrador del sistema puede establecer la máscara que desee poniendo la orden `umask` en un fichero que se lea al iniciar la sesión. Su formato es:

`umask` [*modo*]

donde *modo* indica la máscara predeterminada. Ésta se puede dar de forma octal, en cuyo caso indica los permisos que no se van a establecer, y en forma simbólica, que indica los permisos que se van a establecer.

Un ejemplo de máscara predeterminada podría ser `rw-r-----` para los ficheros regulares, y `rwxr-x---` para los directorios, correspondiendo esto a la orden `umask 027`. ¿A qué es debida esta diferencia? A un fichero regular no se le activa nunca el permiso de ejecución al crearlo; si queremos que lo tenga deberemos dárselo utilizando la orden `chmod`. Observe que la orden `umask` trabaja de forma opuesta a `chmod`; los valores que le damos le dicen al sistema qué permisos no se deben dar cuando se crea el fichero.

Un usuario podrá dar la orden `umask` en cualquier momento si quiere modificar su máscara por omisión, a partir de ese instante los ficheros que cree tendrán la nueva máscara de permisos. Esta orden será válida hasta que finalice la sesión. Si quiere que su máscara siempre sea esa tendrá que colocar la orden `umask` en un fichero que se lea al comienzo de la sesión.

Ejemplo: Damos la orden `umask 077` ¿Qué máscara de permisos tendrán los ficheros que se creen a continuación?

Si creamos un fichero regular tendrá los permisos: `rw----`*, y si creamos un directorio:* `rwx---`.

Ejercicios:

1. Supongamos que damos la orden `$ umask 026`, ¿qué permisos tendrán activados los ficheros ordinarios y directorios que se creen a partir de este momento?

2. ¿Qué diferencia hay entre dar la orden `umask` y `umask -S`?

3.5. Permisos especiales

Aparte de los permisos que ya hemos estudiado, existen otros sobre los que no hemos hablado todavía, los permisos SUID (*set user id*), SGID (*set group id*) y el bit *sticky*.

3.5.1. El bit SUID

Cuando se ejecuta un programa, al proceso creado se le asignan cuatro números que indican a quién pertenece el proceso. Estos son los UID y GID reales y efectivos. Normalmente, los UID y GID efectivos son los mismos que los reales. El sistema GNU/Linux utiliza los UID y GID efectivos para determinar los permisos de acceso de un proceso a los ficheros.

GNU/Linux determina los permisos de acceso de un proceso a un fichero de la siguiente forma; si el UID efectivo de un proceso es el mismo que el UID del propietario del fichero, entonces ese proceso tiene los permisos de acceso al fichero del propietario; si no es así, si el GID efectivo del proceso concuerda con el GID del grupo asociado con el fichero, entonces el proceso tiene los permisos de acceso del grupo; si no es así, el proceso tiene los permisos de acceso de los otros. Esto lo podemos ver en el algoritmo 3.1.

Algoritmo 3.1 Acceso de un proceso a un fichero

```
si UID_efectivo == UID_del_fichero
entonces
   Acceder al fichero como su propietario
si no si GID_efectivo == GID_del_fichero
entonces
   Acceder al fichero como miembro del grupo
si no
   Acceder al fichero como el resto de usuarios
fin si
```

El establecimiento del permiso SUID en un programa cambia su compor-

tamiento. Cuando este permiso está establecido, todos los procesos creados por ese programa tendrán el UID efectivo del propietario del programa y no el del usuario que lo ejecuta. Puesto que los permisos de acceso a los ficheros son determinados a partir del UID efectivo, no del real, el proceso correspondiente a un programa con el permiso SUID activado tiene los mismos permisos de acceso que el propietario del programa, sin importar quién lo ejecute.

Para comprender mejor su funcionamiento veamos un ejemplo. Consideremos los siguientes ficheros:

```
-rw-r--r--    1 root     root     /etc/passwd
-rwsr-xr-x    1 root     root     /usr/bin/passwd
```

El fichero `/usr/bin/passwd` es el programa ejecutable que nos permite cambiar la contraseña. Como ya sabemos, ésta se almacena codificada en el fichero `/etc/passwd`. Si nos fijamos en los permisos de este último (`-rw-r--r--`) podemos observar que sólo el propietario (`root`) tiene permiso de escritura. ¿Cómo es posible entonces que cualquier usuario pueda cambiar su contraseña y modifique el fichero `/etc/passwd`? Esto es posible gracias al permiso SUID que posee el fichero `/usr/bin/passwd`, que aparece como una `s` en el campo de ejecución del propietario. De este modo cuando cualquier usuario lo ejecute (tiene permiso de ejecución para todos los usuarios), su UID efectivo se hará igual al del propietario del fichero; en este caso, al del usuario `root`. Así cuando el proceso creado tenga que modificar el fichero `/etc/passwd` tendrá permiso de escritura en él.

Es importante tener en cuenta que el permiso SUID es independiente del de ejecución, de forma que el fichero ejecutable deberá tener ambos permisos. En el caso de que tengamos un fichero con el SUID activado y no tenga permiso de ejecución para el propietario, aparecería una `S` en el campo de ejecución del propietario.

Para establecer el permiso SUID se utiliza la orden `chmod` en notación simbólica:

`chmod u+s` *fichero*

o bien en notación octal[2]:

`chmod 4`*xxx* *fichero*

[2]Observe que ahora el modo se representa como un número octal de cuatro dígitos, donde el primero se emplea para los permisos especiales y el resto para el propietario, grupo y resto de usuarios.

Para quitarlo también se usa la misma orden:

`chmod u-s` *fichero*

3.5.2. El bit SGID

Al igual que hay un UID efectivo también existe un GID efectivo que se comporta de forma similar. Cuando un programa tiene el permiso SGID establecido, el proceso se ejecuta con los derechos de acceso del grupo asociado con el fichero.

El permiso SGID se establece mediante la orden `chmod` en notación simbólica:

`chmod g+s` *fichero*

o bien en notación octal:

`chmod 2`*xxx* *fichero*

El SGID se borra de la misma forma que el SUID:

`chmod g-s` *fichero*

El permiso SGID se puede activar también en los directorios, teniendo un significado diferente al que hemos visto. Esto se estudiará en el apartado 3.6.

3.5.3. El bit *sticky*

Nos queda un último permiso del que hablar: el bit *sticky*. Este bit se representa mediante una `t` en la máscara de permisos, apareciendo en el campo de ejecución de los otros. Se aplica a directorios de uso público, es decir, aquellos que tienen todos los permisos activados y por tanto, todo el mundo puede crear y borrar ficheros, esto puede ser un poco peligroso porque un usuario puede borrar los ficheros de otros; para evitarlo se activa el bit *sticky*. Con esto se consigue que cada usuario sólo pueda borrar los ficheros que son de su propiedad. Un directorio al que se le suele activar el bit *sticky* es `/tmp`.

Para establecerlo podemos dar:

`chmod 1`*xxx* *directorio*

o bien

`chmod o+t` *directorio*

De forma análoga a los permisos anteriores, para desactivarlo podemos dar:

`chmod o-t` *directorio*

3.6. Cambiar el propietario y grupo de un fichero

Mediante la orden `chgrp` se puede cambiar el identificador de grupo de uno o varios ficheros o un directorio. El formato de esta orden es el siguiente

`chgrp` [*opciones*] *grupo fichero* ...

donde *grupo* puede ser un número de identificador de grupo o un nombre de grupo y *fichero* se puede expresar como un camino absoluto o relativo.

Para poder cambiar el identificador de grupo de un fichero hay que ser el superusuario, o bien ser el propietario del fichero y pertenecer al nuevo grupo.

Cuando cambiamos el identificador de grupo de un directorio, los ficheros que están debajo de él no cambian de grupo, a menos que especifiquemos la opción `-R` de la orden `chgrp`. Los ficheros que se creen en ese directorio después de haber cambiado su grupo pertenecerán al nuevo grupo siempre que el directorio tenga activado el permiso SGID. Si no lo tiene, los nuevos ficheros creados pertenecerán al grupo principal del usuario que cree dichos ficheros.

También se puede cambiar el propietario de ficheros o directorios mediante la orden `chown`. La forma de uso de esta orden es:

`chown` [*opciones*] *usuario*[`:`*grupo*] *fichero* ...

donde *usuario* es el nuevo propietario del *fichero* y puede expresarse como nombre de *login* o UID, el *grupo* puede expresarse como nombre de grupo o GID.

Esta orden sirve para cambiar el propietario y el grupo de un fichero, pero sólo el superusuario puede cambiar el propietario. Un usuario normal sólo puede usarla para cambiar el grupo.

Cuando cambiamos el grupo o el propietario de un fichero mediante estas órdenes, se elimina automáticamente el permiso SUID si estaba activado.

Ejercicios:

1. Al ejecutar la orden `chmod 6621 ejercicio`, ¿qué permisos tendrá el fichero `ejercicio`?

2. Supongamos que tenemos la siguiente situación:

```
drwxrwxrwt  root  system  /usr/publico
-rw-------  pepe  system  /usr/publico/examen
-rw-rw-rw-  juan  system  /usr/publico/apuntes
-rw-r--r--  luis  system  /usr/publico/memoria
```

   Conteste a las siguientes preguntas, razonando la respuesta:

   a) ¿Podría borrar `juan` el fichero `/usr/publico/examen`?

   b) ¿Podría borrar `luis` `/usr/publico/apuntes`?

   `juan` y `luis` pertenecen al grupo `alumnos`.

3.7. Los permisos y las órdenes cp, mv y rm

`cp` Para poder copiar un fichero, éste debe tener el permiso de lectura activado. El comportamiento de la orden depende de la existencia del fichero de destino. Así, cuando se usa `cp` para copiar un fichero, y el fichero de destino no existe, los permisos del fichero fuente se copian también. Esto incluye al permiso SUID, sin embargo el SGID se elimina. Es decir, los permisos del fichero de destino serán iguales a los del fichero fuente, salvo si la máscara por omisión que tenemos definida para nuestros ficheros es diferente (más restrictiva), en cuyo caso le quitará aquellos permisos que se indique en esta máscara predeterminada, y el SGID. El propietario del nuevo fichero será el usuario que da la orden.

Resulta particularmente peligroso copiarnos un programa perteneciente a otro usuario, que tenga activado el bit SUID, puesto que la copia también tendrá activado este bit y nos pertenecerá.

Cuando el fichero de destino existe, tanto sus permisos como el propietario no cambian; el contenido del fichero fuente se copia en el de destino (suponiendo, por supuesto, que el fichero tiene permiso de escritura para el usuario que da la orden `cp`).

`mv` Para poder mover un fichero debemos tener permiso de escritura en el directorio donde está catalogado, los permisos del fichero no influyen en la operación. Su comportamiento depende de si los ficheros fuente y destino residen o no en el mismo sistema de ficheros.

En el primer caso, la orden `mv` simplemente cambia el nombre del fichero, no su contenido, permisos o propietario.

En el segundo, si la orden la da un usuario distinto del propietario del fichero fuente, el de destino tendrá como propietario el que da la orden. Los permisos del fichero de destino serán los mismos que los del fuente.

`rm` Para borrar un fichero no influyen los permisos de éste, debemos tener permiso de escritura en el directorio donde esté catalogado.

Cuando se borra un fichero que tiene el permiso SUID activado, es recomendable asegurarse de que sólo tenga un enlace, ya que cuando se borra el fichero lo que se van borrando son los distintos enlaces. Si borramos un fichero con SUID y existe un enlace al mismo en otro directorio, el fichero sigue existiendo; para asegurarnos de que nadie lo pueda usar lo que podemos hacer es cambiarle los permisos (quitarle SUID) antes de borrarlo, ya que al cambiar los permisos de un fichero cambian los de todos sus enlaces.

Ejercicios:

1. Supongamos que tenemos un fichero `enunciado`. A continuación damos la orden `chmod 751 enunciado`, y posteriormente la orden `cp enunciado solucion`. ¿Cómo afectaría la orden `umask 222` si se intercala entre las dos anteriores? Razone la respuesta.

3.8. Ejercicios

1. ¿Cómo puede saber cuál es su UID y GID? ¿Y los permisos asociados a un fichero?

2. ¿Podría ejecutar un programa que está en un directorio para el cual no tiene permiso de lectura? Justifique la respuesta.

3. Diga qué permisos son necesarios para borrar un fichero de un directorio. Razone su respuesta.

4. Estructura del fichero `/etc/passwd`. Importancia de este fichero en el sistema UNIX. ¿Sería muy grave que el fichero `/etc/passwd` tuviera como máscara de protección la siguiente: `rw-rw-rw-`? Explíquelo.

5. Suponga que está en su directorio principal y da la orden:

   ```
   $ chmod 000 .
   ```

   ¿Qué ocurriría durante la sesión? ¿Afectará esta orden al establecimiento de sesiones posteriores?

6. Considerando los permisos asociados al fichero `calc` ¿ve algún inconveniente en los permisos que se le han dado?

   ```
   -rws-wx-wx user1 grupo1 ....  calc
   ```

7. Existe en UNIX una orden llamada `chgrp`. El fichero `chgrp` se encuentra en el directorio `/usr/bin` y al hacer un listado largo del mismo se obtiene:

   ```
   -rwsr-xr-x 1 root 38912 Oct 19 1988 chgrp
   ```

   Explique la máscara de protección de este fichero.

8. A continuación se da una lista de ficheros con los permisos que llevan asociados, el nombre de su propietario y el grupo al que pertenecen los ficheros

   ```
   drwxr-xr--  root  als /home/als
   drwxr-xr-x  luis  als /home/als/luis
   drwxr-x--x  luis  als /home/als/luis/texto
   -r--rw-rw-  luis  als /home/als/luis/texto/notas
   ---xrwxrw-  luis  als /home/als/luis/texto/exe
   ```

`luis` y `pepe` son dos usuarios pertenecientes al grupo `als`.

Conteste de forma razonada si se puede hacer lo siguiente:

a) ¿Podrá `luis` desde el directorio `/home/als/luis/texto` hacer las siguientes acciones?

I. `cat nota?`
II. `rm notas`
III. Ejecutar el fichero `exe`
IV. Añadir una línea al fichero `notas`

b) ¿Y el usuario `pepe` desde `/home/als/luis/texto`?

I. `rm notas`
II. `cp /dev/null notas`
III. ejecutar el fichero `exe`
IV. `cat notas`

c) ¿Encuentra algún problema a la máscara de permisos del fichero `exe`?

9. Considere la siguiente información:

```
drwxr-x---  ppp  alum  /home/alum/ppp
drwxr-x---  ppp  alum  /home/alum/ppp/juegos
-rwsr-x--x  ppp  alum  /home/alum/ppp/juegos/tetris
-rw-r-----  ppp  alum  /home/alum/ppp/juegos/puntos
drwxr-x---  jjj  alum  /home/alum/jjj
```

a) ¿Podrá el usuario `jjj` del grupo `alum` copiar sin problemas el fichero `/home/alum/ppp/juegos/tetris` a su directorio de entrada? Justifique su respuesta. Si no lo pudiera hacer, ¿qué permisos habría que modificar para poder hacerlo?

b) Suponiendo que ya se ha copiado el fichero, ¿qué permisos tendrá y cuál será el propietario del nuevo fichero? ¿Cambiaría algo si hubiera dado antes la orden `umask 420`?

c) El programa `tetris` intenta escribir la puntuación obtenida en `/home/alum/ppp/juegos/puntos`. ¿Tendrá problemas el usuario `jjj` cuando ejecute el programa que acaba de copiarse para acceder a dicho fichero?

10. A continuación se da una lista de ficheros junto con los permisos que llevan asociados, el nombre de su propietario y el grupo al que pertenecen los ficheros:

```
drwxr-xr-x  root  alum  /home
dr-xr-x---  korn  alum  /home/korn
-r-xr-----  korn  alum  /home/korn/.profile
drwxr-----  korn  alum  /home/korn/C
-rw-r--r--  korn  alum  /home/korn/C/cal.c
-rwx--x--x  korn  alum  /home/korn/C/cal
```

`korn` es del grupo `alum`.

`ritchie` es del grupo `alum`.

`root` es del grupo `system`.

`thompson` es del grupo `system`.

¿Es posible lo siguiente?

a) Para el usuario `korn` en el directorio `/home/korn`

I. `rm -f .profile`

II. `mv .profile .kshrc`

b) Para el usuario `ritchie` en el directorio `/home/ritchie`

I. ejecutar el fichero `~korn/C/cal`

II. `more ~korn/.profile`

c) Para el usuario `root` en el directorio `/usr`

I. ejecutar el fichero `~korn/C/cal`

11. Justifique la necesidad de que un sistema como UNIX tenga un permiso del tipo SUID con ejemplos que conozca.

12. A continuación se da una lista de ficheros junto con los permisos que llevan asociados, el nombre de su propietario y el grupo al que pertenecen.

```
drwxr-xr-- root  alf /home/alf
drwxr-xr-x perez alf /home/alf/perez
drwx------ fdez  alf /home/alf/fdez
drwxr--r-- perez alf /home/alf/perez/Cobol
-rws------ perez alf /home/alf/perez/Cobol/nomina
dr-x--x--- fdez  alf /home/alf/fdez/Cprog
-rwx--x--x fdez  alf /home/alf/fdez/Cprog/juego
-rw-r----- fdez  alf /home/alf/fdez/Cprog/juego.c
```

Responda razonadamente:

a) ¿Podría el usuario `perez` en su directorio `Cobol` hacer lo siguiente?

I. ejecutar el fichero `nomina`
II. `rm nomina`
III. `ls -l /home/alf/fdez/Cprog`

b) ¿Podría el usuario `fdez` en su directorio `Cprog` hacer lo siguiente?

I. `rm juego`
II. `ls -lg /home/alf/perez/Cobol`

13. Dados los siguientes ficheros y directorios:

```
drwxr-xr-x  manuel  al93  /home/a93/manuel
drw-r-x---  manuel  al93  /home/a93/manuel/dir1
-r--r-----  manuel  al93  /home/a93/manuel/dir1/fich1
drwxr-xr-x  manuel  al93  /home/a93/manuel/dir2
```

Conteste razonadamente a las siguientes preguntas: ¿Podría el usuario `manuel` hacer las siguientes acciones estando situado en su directorio de entrada?

a) `cp dir1/fich1 dir2/fich2`
b) `rm dir1/fich1`
c) `rm fich1`, el usuario se encuentra ahora en el directorio `dir1`
d) editar el fichero `fich1` para añadir una línea
e) editar el fichero `fich1` para añadir una línea, suponiendo que el usuario `manuel` se encuentra en su directorio `dir1`

14. Suponga que usted tiene por nombre de usuario `user1`, un compañero suyo tiene de nombre de usuario `user2`, y posee un fichero llamado `fich1` que tiene activado el permiso de escritura para el grupo y para los otros, pero no tiene permiso de lectura para ninguno de esos grupos. ¿Qué línea de órdenes daría usted para cambiar los permisos de dicho fichero, para que tenga permiso de lectura?

15. Suponga que tiene un fichero llamado `units` cuya máscara de permisos es: `rw-r--r--`. Su máscara por omisión para la creación de ficheros es 027. Si da la orden `cp units unidades` ¿qué permisos tendrá el nuevo fichero `unidades`?

16. Créese un enlace simbólico a un fichero ya existente. ¿Qué permisos tiene el nuevo fichero? Si utiliza la orden `chmod` para cambiar los permisos de este nuevo fichero, ¿qué ocurre?. Explique la respuesta.

17. Un usuario de UID=7 y GID=9 sitúa en su directorio `$HOME/bin` un programa llamado `cpal`. Este programa accede durante su ejecución a un fichero llamado `tpal` para modificarlo, que está situado en el mismo directorio. Los permisos asociados a dichos ficheros y al directorio son los siguientes:

```
rwx--x--x $HOME/bin
rwx--s--x $HOME/bin/cpal
rwx---r-- $HOME/bin/tpal
```

    a) ¿Podría un usuario de UID=18 y GID=9 ejecutar el fichero `cpal` sin problemas?
    b) ¿Qué ocurriría si el usuario de UID=21 y GID=15 intentara ejecutar el programa `cpal`?

18. Dados los ficheros:

```
drwxr-xr-x  root  al93  /home/al93
drwxr-xr-x  juan  al93  /home/al93/juan
drwxr--r--  juan  al93  /home/al93/juan/docs
-rw-rw-rw-  juan  al93  /home/al93/juan/docs/apuntes
```

    ¿Podría el usuario `pepe` perteneciente al grupo `al92` dar las órdenes siguientes?

    a) `ls ~juan`
    b) `ls ~juan/docs`
    c) `rm ~juan/docs/apuntes`

19. ¿Qué orden hay que dar para ver la máscara de permisos de forma simbólica?

20. Existen en nuestro sistema dos ficheros cuyos permisos, propietario y grupo se dan a continuación:

```
-?????????  daemon system /usr/games/rogue
-rw-r--r--  daemon system /usr/games/lib/rogue_roll
```

    El fichero `/usr/games/rogue` contiene el código ejecutable de un juego que se desea pueda ser ejecutado por cualquier usuario del sistema. Durante su ejecución, este programa intenta acceder al fichero `/usr/games/lib/rogue_roll` para escribir en él la puntuación que ha obtenido el usuario que ha jugado. Dé la línea de órdenes necesaria para establecer permisos estrictamente necesarios que debe tener el fichero `/usr/games/rogue` para que lo anterior sea posible. Explique por qué se dan cada uno de los permisos otorgados.

21. Considere los siguientes ficheros y los permisos asociados a los mismos:

```
fichero  propietario  grupo  permisos
prog     pepe         alum   rwxr-x--x
datos    pepe         alum   rw-rw-r--
```

Teniendo en cuenta que el programa `prog` durante su ejecución abre el fichero `datos` para lectura/escritura, conteste a las siguientes preguntas:

¿Podrá la usuaria `ana` ejecutar `prog` sin ningún tipo de problemas?

a) Si `ana` pertenece al grupo `alum`

b) Si `ana` pertenece a otro grupo distinto

Si existe algún problema ¿sería posible solucionarlo sin alterar los bits de protección del fichero `datos`?

22. Suponga que da la siguiente secuencia de órdenes:

```
$ mkdir temp
$ cp /etc/passwd temp
$ chmod 400 temp
$ ls temp
$ ls -l temp
```

¿Cuál será la salida de las dos últimas órdenes? Explique la razón.

23. ¿Qué significa la máscara de creación de ficheros 024?

24. ¿Qué contiene el fichero `/etc/group`? ¿Cómo podemos cambiar el grupo al que pertenece un fichero? Requisitos necesarios. ¿Qué ocurre si el fichero al que cambiamos de grupo es un directorio?