CONTROL DE ACCESO A ARCHIVOS CON PERMISOS DEL SISTEMA DE ARCHIVOS LINUX

RH124

Capitulo 6

Descripción general	
Meta	Configurar los permisos del sistema de archivos Linux en los archivos e interpretar los efectos de seguridad de los distintos parámetros de configuración de permisos.
Objetivos	• Explicar cómo funciona el modelo de permisos de archivo Linux. • Cambiar los permisos y la propiedad de los archivos con las herramientas de línea de comando. • Configurar un directorio en el que los archivos creados recientemente puedan ser escritos en forma automática por los miembros del grupo propietario del directorio, usando permisos especiales y configuración de default umask.
Secciones	• Permisos del sistema de archivos Linux (y práctica) • Administración de los permisos del sistema de archivos desde la línea de comando (y práctica) • Administración de permisos predeterminados y acceso a archivos (y práctica)
Trabajo de laboratorio	• Control de acceso a archivos con permisos del sistema de archivos Linux

Permisos del sistema de archivos Linux

RH124

Objetivos

Tras finalizar esta sección, los estudiantes deberían poder explicar cómo funciona el modelo de permisos de archivo de Linux.

Permisos del sistema de archivos Linux

El acceso a los archivos por parte de los usuarios es controlado por *permisos de archivos*. El sistema de permisos de archivos de Linux es simple pero flexible, lo que hace que sea fácil de comprender y de aplicar, y aún así poder manejar fácilmente los casos más normales de permisos.

Los archivos tienen solo tres categorías de usuario a las que se le aplican permisos. El archivo pertenece a un *usuario*, que generalmente es quien creó el archivo. El archivo también pertenece a un solo *grupo*, generalmente el grupo primario del usuario que creó el archivo, pero esto se puede cambiar. Se pueden establecer diferentes permisos para el usuario propietario y el grupo propietario, así como para todos los *otros* usuarios en el sistema que no sean el usuario o un miembro del grupo propietario.

Se aplicarán los permisos más específicos. Por lo tanto, los permisos de *usuario* anulan los permisos de *grupo*, que anulan *otros* permisos.

En el siguiente gráfico, joshua es un integrante de los grupos joshua y web, mientras que allison integra los grupos allison, wheel y web. Cuando joshua y allison necesitan colaborar, los archivos deben asociarse con el grupo web y los permisos del grupo deben permitir el acceso deseado.

Figura 6.1: Ilustración de la membresía de grupo

También existen tres categorías de permisos que se aplican: *leer*, *escribiry ejecutar*. Estos permisos afectan el acceso a archivos y directorios de la siguiente forma:

Permiso	Efecto en los archivos	Efecto en los directorios
r (read)	Pueden leerse los contenidos del archivo.	Los contenidos del directorio (nombres de archivos) pueden detallarse.
w (write)	Los contenidos del archivo pueden cambiarse.	Cualquier archivo en el directorio puede crearse o eliminarse.

Permiso	Efecto en los archivos	Efecto en los directorios
x (exec)	Los archivos pueden ejecutarse como comandos.	Es posible acceder al contenido del directorio (según los permisos de los archivos en el directorio).

Tenga en cuenta que los usuarios normalmente poseen privilegios tanto de **read** como de **exec** en los directorios de solo lectura, por lo que pueden listar el directorio y acceder a su contenido. Si un usuario solo posee acceso de **read** en un directorio, los nombres de los archivos dentro de este pueden detallarse, pero no estará disponible otra información, incluidos permisos o marcas de tiempo, y tampoco se podrá acceder a ellos. Si un usuario solo posee acceso **exec** en un directorio, no podrá enumerar los nombres de los archivos en este, pero sí podrá acceder a su contenido en caso de que ya conozca el nombre de un archivo del que posee permiso para leer. Así, podrá acceder al contenido del archivo especificando su nombre.

Todo usuario que cuente con permisos de escritura para el directorio donde se encuentra el archivo puede quitar un archivo, sin importar la propiedad ni los permisos del archivo en sí. (Esto se puede anular con un permiso especial, el *sticky bit*, el cual abordaremos al finalizar la unidad).

Visualizar permisos y propiedades de archivos o directorios

La opción **-l** del comando **ls** expandirá los detalles de los archivos para incluir tanto los permisos de un archivo como su propiedad:

```
[student@desktopX ~]$ ls -l test
-rw-rw-r--. 1 student student 0 Feb  8 17:36 test
```

El comando **ls -l directoryname** mostrará el listado ampliado de todos los archivos que están dentro del directorio. Si desea evitar el descenso al directorio y ver los detalles expandidos de dicho directorio, agregue la opción **-d** a ls:

```
[student@desktopX ~]$ ls -ld /home
drwxr-xr-x. 5 root root 4096 Jan 31 22:00 /home
```

nota

A diferencia de los permisos NTFS, los permisos de Linux solo se aplican al directorio o archivo en el que están establecidos. Los permisos en un directorio no se heredan de forma automática por los subdirectorios o los archivos que se encuentran en él. (No obstante, los permisos en un directorio *pueden* efectivamente bloquear el acceso a su contenido). Todos los permisos en Linux se establecen directamente en cada archivo o directorio.

El permiso para **leer** en un directorio de Linux es casi equivalente a **List folder contents** en Windows.

El permiso para **escribir** en un directorio de Linux es equivalente a **Modify** en Windows. Esto implica la posibilidad de eliminar archivos y subdirectorios. En Linux, si **write** y el **sticky bit** están establecidos en un directorio, solo el usuario que sea propietario de un archivo o de un subdirectorio del directorio puede eliminarlo, lo que se asemeja al comportamiento del permiso **Write** de Windows.

El usuario root posee los permisos equivalentes a **Full Control** de Windows en todos los archivos de Linux. Sin embargo, el usuario root aún puede tener acceso restringido por la política de SELinux del sistema y el contexto de seguridad del proceso y de los archivos en cuestión. SELinux se analizará en un curso posterior.

Ejemplos: usuario, grupo y otros conceptos de Linux

```
Users and their groups:

    lucy      lucy,ricardo
    ricky     ricky,ricardo
    ethel     ethel,mertz
    fred      fred,mertz

File attributes (permissions, user & group ownership, name):

    drwxrwxr-x   ricky   ricardo   dir (which contains the following files)
      -rw-rw-r--   lucy     lucy      lfile1
      -rw-r--rw-   lucy     ricardo   lfile2
      -rw-rw-r--   ricky    ricardo   rfile1
      -rw-r-----   ricky    ricardo   rfile2
```

Comportamiento permitido o denegado	Control de permisos
lucy es la única persona que puede cambiar el contenido de **lfile1**.	**lucy** tiene permisos de escritura en el archivo **lfile1** como propietaria. No hay personas registradas como miembros del grupo de **lucy**. Los permisos para *otros* no incluyen permisos de escritura.
ricky puede ver el contenido de **lfile2**, pero no puede modificar el contenido de **lfile2**.	**ricky** es miembro del grupo **ricardo** y ese grupo posee permisos de solo lectura para **lfile2**. Si bien *otro* tiene permisos de escritura, los permisos del grupo tienen prioridad.

Comportamiento permitido o denegado	Control de permisos
ricky puede eliminar **lfile1** y **lfile2**.	**ricky** tiene permisos de escritura en el directorio que contiene ambos archivos y, como tal, puede eliminar cualquier archivo de ese directorio.
ethel puede cambiar el contenido de **lfile2**.	Dado que **ethel** no es **lucy** y no es miembro del grupo **ricardo**, los permisos relacionados con *otros* la rigen, y estos incluyen permiso de escritura.
lucy puede cambiar el contenido de **rfile1**.	**lucy** es miembro del grupo **ricardo** y ese grupo tiene permisos de lectura y escritura respecto de **rfile1**.
ricky puede ver y modificar el contenido de **rfile2**. **lucy** puede ver, pero no modificar el contenido de **rfile2**. **ethel** y **fred** no tienen ningún acceso al contenido de **rfile2**.	**ricky** es propietario del archivo y tiene acceso de lectura y de escritura a **rfile2**. **lucy** es miembro del grupo **ricardo** y ese grupo tiene acceso de solo lectura a **rfile2**. Rigen *otros* permisos para **ethel** y **fred**, y dichos permisos no incluyen permiso de lectura ni escritura.

Referencias

Página del manual (1) **`ls`**

`info coreutils` (*GNU Coreutils*)

- Sección 13: Cambiar atributos de archivos

Práctica: Interpretación de permisos de archivos y directorios

Administración de permisos del sistema de archivos desde la línea de comandos

RH124

Objetivos

Tras finalizar esta sección, los estudiantes deberían poder cambiar los permisos y la propiedad de los archivos usando herramientas de la línea de comandos.

Cambio de permisos de archivo o directorio

El comando usado para cambiar los permisos desde la línea de comandos es **chmod**, que significa "change mode" (cambiar modo) (los permisos también se conocen como el *mode* de un archivo). El comando **chmod** tiene una instrucción de permiso seguida de una lista de archivos o directorios para cambio. La instrucción de permiso puede ser emitida simbólicamente (el método simbólico) o numéricamente (el método numérico).

Palabras clave de métodos simbólicos:

```
chmod WhoWhatWhich file|directory
```

- *Who* es u, g, o, a *(para usuario, grupo, otros, todos)*
- *What* es +, -, = *(para agregar, eliminar, establecer exactamente)*
- *Which* es br, w, x *(para leer, escribir, ejecutar)*

El método *simbólico* de cambiar los permisos del archivo usa letras para representar los distintos grupos de permisos: **u** para usuario, **g** para grupo, **o** para otros y **a** para todos.

Con el método simbólico, no es necesario establecer un grupo completamente nuevo de permisos. En su lugar, se puede cambiar uno o más permisos existentes. Para lograrlo, puede usar tres símbolos: **+** para agregar permisos a un conjunto, **-** para eliminar permisos de un conjunto e **=** para reemplazar el conjunto completo por un grupo de permisos.

Método numérico:

```
chmod ### file|directory
```

- Cada dígito representa un nivel de acceso: usuario, grupo, otros.
- # es la suma de r = 4, w = 2 y x = 1.

Al utilizar el método *numérico* , los permisos son representados por un número *octal* de tres dígitos (o cuatro, al establecer permisos avanzados). Un único dígito **octal** puede

Para realizar conversiones entre una representación simbólica y numérica de permisos, debemos saber cómo se realiza la asignación. En la representación octal (numérica) de tres dígitos, cada dígito representa un grupo de permisos, de izquierda a derecha: usuario, grupo y otros. En cada uno de estos grupos, se comienza con **0**. Si se encuentra el permiso de lectura, agregue **4**. Agregue **2** si se encuentra el permiso de escritura y **1** para ejecutar.

Los permisos numéricos a menudo son usados por administradores avanzados, ya que son más breves para escribir y pronunciar, y al mismo tiempo, le proporcionan el control total de todos los permisos.

Examinar los permisos **-rwxr-x---**. Para usuario, **rwx** se calcula como **4+2+1=7**. Para grupo, **r-x** se calcula como **4+0+1=5**, y para otros usuarios, **---** se representa con **0**. Con estos tres en conjunto, la representación numérica de dichos permisos es **750**.

Ejemplos

- Elimine el permiso de lectura y escritura para el grupo y otros respecto de **file1**:

```
[student@desktopX ~]$ chmod go-rw file1
```

- Agregue un permiso de ejecución para todos respecto de **file2**:

```
[student@desktopX ~]$ chmod a+x file2
```

- Establezca un permiso de lectura, escritura y ejecución para usuario, lectura y escritura para grupo y ningún permiso para otros respecto de **sampledir**:

```
[student@desktopX ~]$ chmod 750 sampledir
```

nota

El comando **chmod** admite la opción **-R** para establecer permisos de manera recursiva en los archivos, en todo el árbol de directorios. Cuando utiliza la opción **-R**, puede ser útil establecer permisos de manera simbólica mediante el uso del indicador **X**. Esto permitirá ejecutar (buscar) permisos para establecer en los directorios de modo que se pueda acceder a su contenido, sin cambiar los permisos en la mayoría de los archivos. Pero tenga cuidado. Si un archivo tiene un permiso de ejecución establecido, **X** establecerá el permiso de ejecución especificado en ese archivo también. Por ejemplo, el siguiente comando establecerá de manera recursiva el acceso de lectura y de escritura en **demodir** y todos sus procesos secundarios para el propietario del grupo, pero solo aplicará permisos de ejecución de grupo a directorios y archivos que ya tienen permisos de ejecución establecidos para usuario, grupo u otros.

```
[student@desktopX ~]# chmod -R g+rwX demodir
```

Cambio de la propiedad de grupo o de usuario de un archivo o directorio

Un archivo creado recientemente es propiedad del usuario que lo crea. De manera predeterminada, el archivo nuevo es propiedad del grupo, que es el grupo principal del usuario que crea el archivo. Dado que Red Hat Enterprise Linux utiliza grupos privados de usuarios, este grupo a menudo es un grupo con ese único usuario como miembro. Para garantizar el acceso basado en membresía de grupo, es posible que se deban cambiar el propietario o el grupo de un archivo.

La propiedad del archivo se puede cambiar con el comando **chown** (change owner). Por ejemplo, para otorgarle propiedad del archivo **`foofile`** a **`student`**, se podría usar el siguiente comando:

La propiedad del archivo se puede cambiar con el comando **chown** (change owner). Por ejemplo, para otorgarle propiedad del archivo **foofile** a **student**, se podría usar el siguiente comando:

```
[root@desktopX ~]# chown student foofile
```

Se puede utilizar **chown** con la opción **-R** para cambiar recursivamente la propiedad de un árbol de directorios completo. El siguiente comando otorgaría propiedad de **foodir** y de todos los archivos y subdirectorios incluidos dentro a **student**:

```
[root@desktopX ~]# chown -R student foodir
```

El comando **chown** también se puede utilizar para cambiar el propietario del grupo de un archivo, anteponiendo el nombre del grupo con dos puntos (**:**). Por ejemplo, el siguiente comando cambiará el grupo de **foodir** a **admins**:

```
[root@desktopX ~]# chown :admins foodir
```

nota

En lugar de usar **chown**, algunos usuarios cambian el propietario del grupo con el comando **chgrp**; este comando realiza exactamente lo mismo que cambiar la propiedad con **chown**, e incluye el uso de **-R** para que afecte la totalidad de árboles de directorio.

Referencias

Páginas del manual: **ls**(1), **chmod**(1), **chown**(1) y **chgrp**(1)

Práctica: Administrar la seguridad de los archivos desde la línea de comandos

Administración de permisos predeterminados y acceso a archivos

RH124

Objetivos

Tras finalizar esta sección, los estudiantes deberían poder configurar un directorio en el que los miembros del grupo que posee el directorio puedan escribir automáticamente los archivos creados recientemente, mediante el uso de permisos especiales y configuraciones de umask predeterminadas.

Permisos especiales

El permiso **setuid** (o **setgid**) en un archivo ejecutable significa que el comando se ejecutará como **usuario** (o **grupo**) del archivo, no como el usuario que ejecutó el comando. Un ejemplo de este caso es el comando **passwd**:

```
[student@desktopX ~]$ ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 35504 Jul 16  2010 /usr/bin/passwd
```

En una larga lista, puede detectar los permisos **setuid** con una **s** minúscula, donde normalmente esperaría ver la **x** (permisos de ejecución del propietario). Si el propietario no posee permisos de ejecución, será reemplazada por una **S** mayúscula.

El **sticky bit** para un directorio establece una restricción especial sobre la eliminación de archivos: solo el propietario del archivo (y **root**) puede eliminar archivos del directorio. Un ejemplo es **/tmp**:

```
[student@desktopX ~]$ ls -ld /tmp
drwxrwxrwt. 39 root root 4096 Feb  8 20:52 /tmp
```

Por último, **setgid** en un directorio significa que los archivos creados en el directorio heredarán la afiliación de grupos del directorio, en lugar de heredarla del usuario que la creó. Esto generalmente se usa en directorios colaborativos grupales para poder cambiar automáticamente un archivo del grupo privado predeterminado al grupo compartido.

En una larga lista, puede detectar los permisos **setgid** con una **s** minúscula, donde normalmente esperaría ver la **x** (permisos de ejecución del grupo). Si el grupo no posee permisos de ejecución, será reemplazada por una **S** mayúscula.

Efectos de los permisos especiales en archivos y directorios

Permiso especial	Efecto en los archivos	Efecto en los directorios
u+s (suid)	El archivo se ejecuta como el usuario propietario, no como el usuario que lo ejecutó.	No hay efectos.
g+s (sgid)	El archivo se ejecuta como el grupo propietario.	Los archivos creados recientemente en el directorio han establecido al propietario del grupo para que coincida con el propietario del grupo del directorio.
o+t (sticky)	No hay efectos.	Los usuarios con **escribir** en el directorio solo pueden eliminar los archivos de los que son propietarios, pero no pueden eliminar ni forzar el guardado de archivos cuyos propietarios sean otros usuarios.

Establecer permisos especiales

- Simbólicamente: setuid = **u+s**; setgid = **g+s**; sticky = **o+t**
- Numéricamente (cuarto dígito precedente): setuid = 4 ; setgid = 2 ; sticky = 1

Ejemplos

- Agregue el setgid bit en **directory**:

```
[root@desktopX ~]# chmod g+s directory
```

- Establezca el setgid bit y los permisos de lectura/escritura/ejecución para el usuario y el grupo en **directory**:

```
[root@desktopX ~]# chmod 2770 directory
```

Permisos de archivos predeterminados

Los permisos predeterminados para archivos se establecen mediante el proceso que los crea. Por ejemplo, los editores de texto crean archivos para que sean de lectura y escritura, pero no ejecutables para cualquiera. Lo mismo ocurre con el redireccionamiento de shell. Además, son los compiladores quienes crean los ejecutables binarios como ejecutables. El comando **`mkdir`** crea directorios nuevos con todos los permisos establecidos: de lectura, escritura y ejecución.

La experiencia indica que estos permisos por lo general no se establecen cuando se crean los directorios y archivos nuevos. Esto ocurre porque algunos permisos son borrados por el umask del proceso de shell. El comando **`umask`** sin argumentos mostrará el valor actual del umask de shell:

```
[student@desktopX ~]$ umask
0002
```

Cada proceso en el sistema tiene un umask, que es una máscara de bits octal utilizada para borrar los permisos de archivos y directorios nuevos creados por el proceso. Si se establece un bit en el umask, el permiso correspondiente se elimina en los archivos nuevos. Por ejemplo, el umask anterior, 0002, borra el bit de escritura para otros usuarios. Los ceros iniciales indican que los permisos especiales, de usuario y de grupo no están borrados. Un umask de 077 borra los permisos de todo el grupo y de otros de los archivos creados recientemente.

Utilice el comando **umask** con un argumento numérico único para cambiar el umask de la shell actual. El argumento numérico debe ser un valor octal que se corresponda con el valor del umask nuevo. Si tiene menos de 3 dígitos, se suponen ceros iniciales.

Los valores de umask predeterminados del sistema para usuarios de shell Bash se definen en los archivos **/etc/profile** y **/etc/bashrc**. Los usuarios pueden omitir los valores predeterminados del sistema en sus archivos **.bash_profile** y **.bashrc**.

En este ejemplo, siga los pasos a continuación mientras el instructor demuestra los efectos de **umask** en directorios y archivos nuevos.

1. Cree un archivo y un directorio nuevos para ver cómo el umask predeterminado afecta los permisos.

```
[student@desktopX ~]$ touch newfile1
[student@desktopX ~]$ ls -l newfile1
-rw-rw-r--. 1 student student 0 May  9 01:54 newfile1
[student@desktopX ~]$ mkdir newdir1
[student@desktopX ~]$ ls -ld newdir1
drwxrwxr-x. 2 student student 0 May  9 01:54 newdir1
```

2. Establezca el valor de unmask en 0. Esta configuración no enmascarará ninguno de los permisos de los archivos nuevos. Cree un archivo y un directorio nuevos para ver cómo este umask nuevo afecta los permisos.

```
[student@desktopX ~]$ umask 0
[student@desktopX ~]$ touch newfile2
[student@desktopX ~]$ ls -l newfile2
-rw-rw-rw-. 1 student student 0 May  9 01:54 newfile2
[student@desktopX ~]$ mkdir newdir2
[student@desktopX ~]$ ls -ld newdir2
drwxrwxrwx. 2 student student 0 May  9 01:54 newdir2
```

3. Establezca el valor del umask en 007. Esta configuración enmascarará todos los "otros" permisos de los archivos nuevos.

```
[student@desktopX ~]$ umask 007
[student@desktopX ~]$ touch newfile3
[student@desktopX ~]$ ls -l newfile3
-rw-rw----. 1 student student 0 May  9 01:55 newfile3
[student@desktopX ~]$ mkdir newdir3
[student@desktopX ~]$ ls -ld newdir3
drwxrwx---. 2 student student 0 May  9 01:54 newdir3
```

4. Establezca el valor del umask en 027. Esta configuración enmascarará el acceso de escritura para miembros del grupo y todos los "otros" permisos de los archivos nuevos.

```
[student@desktopX ~]$ umask 027
[student@desktopX ~]$ touch newfile4
[student@desktopX ~]$ ls -l newfile4
-rw-r-----. 1 student student 0 May  9 01:55 newfile4
[student@desktopX ~]$ mkdir newdir4
[student@desktopX ~]$ ls -ld newdir4
drwxr-x---. 2 student student 0 May  9 01:54 newdir4
```

5. Inicie sesión como **root** para cambiar el umask predeterminado para usuarios sin privilegios a fin de evitar todo acceso de usuarios que no estén en su grupo.

   Modifique **/etc/bashrc** y **/etc/profile** para cambiar el umask predeterminado para los usuarios de shell Bash. Dado que el unmask predeterminado para usuarios sin privilegios es 0002, busque el comando **umask** en estos archivos que establezca el umask en ese valor. Cámbielos para establecer el umask en 007.

```
[root@desktopX ~]# less /etc/bashrc
    # You could check uidgid reservation validity in
    # /usr/share/doc/setup-*/uidgid file
    if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
       umask 002
    else
       umask 022
    fi

    # Only display echos from profile.d scripts if we are no login shell
[root@desktopX ~]# vim /etc/bashrc
[root@desktopX ~]# less /etc/bashrc
    # You could check uidgid reservation validity in
    # /usr/share/doc/setup-*/uidgid file
    if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
       umask 007
    else
       umask 022
    fi

    # Only display echos from profile.d scripts if we are no login shell
[root@desktopX ~]# less /etc/profile
    # You could check uidgid reservation validity in
    # /usr/share/doc/setup-*/uidgid file
    if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
        umask 002
    else
        umask 022
    fi

    for i in /etc/profile.d/*.sh ; do
[root@desktopX ~]# vim /etc/profile
[root@desktopX ~]# less /etc/profile
    # You could check uidgid reservation validity in
    # /usr/share/doc/setup-*/uidgid file
    if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
        umask 007
    else
        umask 022
    fi

    for i in /etc/profile.d/*.sh ; do
```

6. Vuelva a iniciar sesión como **student** y confirme que los cambios de umask que realizó sean persistentes.

```
[student@desktopX ~]$ umask
0007
```

nota

Es posible que otros shells, como **tcsh**, tengan distintos archivos de inicialización predeterminados del sistema en **/etc** y directorios principales de los usuarios.

Referencias

Páginas del manual: **bash**(1), **ls**(1), **chmod**(1) y **umask**(1)

Práctica: Control de permisos y propiedad de archivos nuevos