Access Control Lists in Linux

This chapter provides a brief summary of the background and functions of POSIX ACLs for Linux file systems. Learn how the traditional permission concept for file system objects can be expanded with the help of ACLs (*Access Control Lists*) and which advantages this concept provides.

Advantages of ACLs

Note

POSIX ACLs

The term "POSIX ACL" suggests that this is a true POSIX (*Portable Operating System Interface*) standard. However, the respective draft standards POSIX 1003.1e and POSIX 1003.2c have been withdrawn. Many UNIX-flavoured operating systems based their ACL concepts on these documents. The implementation of file system ACLs as described in this chapter is based on the contents of these two documents, which can be viewed at the following URL:

`http://wt.xpilot.org/publications/posix.1e/`

Note

Traditionally, a file object in Linux is associated with three sets of permissions. These sets assign read (`r`), write (`w`), and execute (`x`) permissions for the three user groups file owner, group, and other. Nine bits are used to determine the characteristics of all objects in a Linux file system. Additionally, the *set user id*, *set group id*, and *sticky* bits can be set for special cases. More details are revealed in section *Users and Access Permissions* in the *User Guide* manual.

This lean concept is fully adequate for most practical cases. However, for more complex scenarios or advanced applications, system administrators formerly had to use a number of tricks to circumvent the limitations of the traditional permission concept.

ACLs can be used for situations where the traditional file permission concept does not suffice. They allow the assignment of permissions to individual users or groups even if these do not correspond to the owner or the owning group.

Access Control Lists are a feature of the Linux kernel and are currently supported by ReiserFS, Ext2, Ext3, JFS, and XFS. Using ACLs, complex scenarios can be realized without implementing complex permission models on the application level.

The advantages of ACLs are clearly evident in situations such as the replacement of a Windows server by a Linux server. Some of the connected workstations may continue to run under Windows even after the migration. The Linux system offers file and print services to the Windows clients with Samba. As Samba supports ACLs, user permissions can be configured both on the Linux server and in Windows with a graphical user interface (only Windows NT and later). With winbindd, it is even possible to assign permissions to users that only exist in the Windows domain without any account on the Linux server. On the server side, edit the Access Control Lists using getfacl and setfacl.

Definitions

User class The conventional POSIX permission concept uses three *classes* of users for assigning permissions in the file system: the owner, the owning group, and other users. Three permission bits can be set for each user class, giving permission to read (`r`), write (`w`), and execute (`x`). An introduction to the user concept in Linux is provided in the *User Guide* in the section *Users and Access Permissions*.

Access ACL The user and group access permissions for all kinds of file system objects (files and directories) are determined by means of access ACLs.

Default ACL Default ACLs can only be applied to directories. They determine the permissions a file system object inherits from its parent directory when it is created.

ACL entry Each ACL consists of a set of ACL entries. An ACL entry contains a type (see Table B.1 on the following page), a qualifier for the user or group to which the entry refers, and a set of permissions. For some entry types, the qualifier for the group or users is undefined.

Handling ACLs

The following section explains the basic structure of an ACL and its various characteristics. The interrelation between ACLs and the traditional permission concept in the Linux file system is briefly demonstrated by means of several figures. Two examples show how you can create your own ACLs using the correct syntax. In conclusion, find information about the way ACLs are interpreted by the operating system.

Structure of ACL Entries

Basically, ACLs can be divided into two classes: A *minimum* ACL merely comprises the entries for the types owner, owning group, and other, which corresponds to the conventional permission bits for files and directories. An *extended* ACL exceeds this concept. It must contain a *mask* entry and may contain several entries for the *named user* and *named group* types. Table B.1 on the next page provides a brief summary of the various available types of ACL entries.

Type	Text Form
owner	`user::rwx`
named user	`user:name:rwx`
owning group	`group::rwx`
named group	`group:name:rwx`
mask	`mask::rwx`
other	`other::rwx`

Table B.1: *ACL Entry Types*

The permissions defined in the entries *owner* and *other* are always effective. Except for the *mask* entry, all other entries (*named user*, *owning group*, and *named group*) can be either effective or masked. If permissions exist in one of the above-mentioned entries as well as in the mask, they are effective. Permissions contained only in the mask or only in the actual entry are not effective. The example in Table B.2 demonstrates this mechanism.

Entry Type	Text Form	Permissions
named user	`user:jane:r-x`	`r-x`
mask	`mask::rw-`	`rw-`
	effective permissions:	`r--`

Table B.2: *Masking Access Permissions*

ACL Entries and File Mode Permission Bits

Figure B.1 on the next page and Figure B.2 on the facing page illustrate the two cases of a minimum ACL and an extended ACL. The figures are structured in three blocks — the left block shows the type specifications of the ACL entries, the center block displays an example ACL, and the right block shows the respective permission bits as displayed by `ls -l`.

In both cases, the *owner class* permissions are mapped to the ACL entry *owner*. The mapping of *other class* permissions to the respective ACL entry is also constant. However, the mapping of the *group class* permissions varies:

- In the case of a minimum ACL — without *mask* entry — the *group class* permissions are mapped to the ACL entry *owning group*. This is shown in Figure B.1 on the next page.

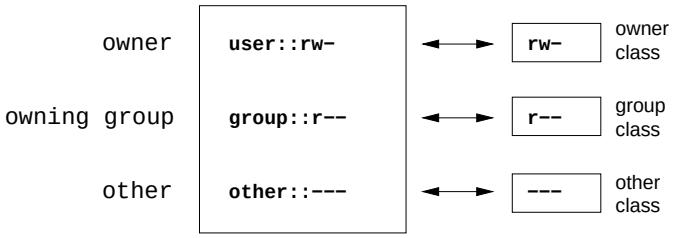

Figure B.1: *Minimum ACL: ACL Entries Compared to Permission Bits*

- In the case of an extended ACL — with *mask* entry — the *group class* permissions are mapped to the *mask* entry. This is shown in Figure B.2.

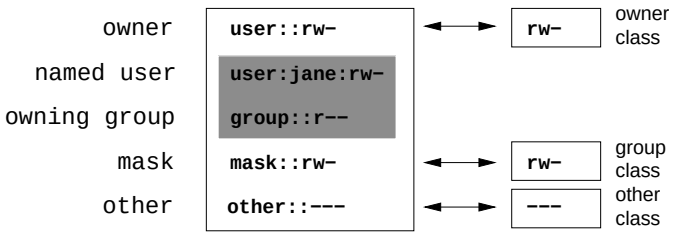

Figure B.2: *Extended ACL: ACL Entries Compared to Permission Bits*

This mapping ensures the smooth interaction of applications with ACL support and those without ACL support. The access permissions that were assigned by means of the permission bits represent the upper limit for all other "fine adjustments" made by means of ACLs. All permissions not reflected here were either not set in the ACL or are not effective. If permission bits are changed, this is also reflected in the respective ACL and vice versa.

A Directory with Access ACL

The handling of access ACLs is demonstrated in three steps by means of the following example:

- Creating a file system object (a directory in this case)
- Modifying the ACL

- Using masks

1. Before you create the directory, use the `umask` command to determine which access permissions should be masked from the outset:

   ```
   umask 027
   ```

   The owner has all access permissions (read, write, execute) (`0`) and write access is disabled for the owning group (`2`). All other users are denied all kinds of access (`7`). The `umask` value can be read as a bit mask. Refer to the manual page of `umask` for further details.

   ```
   mkdir mydir
   ```

   After creating the `mydir` directory with the permissions set by `umask`, use the following command to check if all permissions were assigned correctly:

   ```
   ls -dl mydir drwxr-x--- ... tux project3 ... mydir
   ```

2. Check the initial state of the ACL and insert a new user entry and a new group entry.

   ```
   getfacl mydir

   # file: mydir
   # owner: tux
   # group: project3
   user::rwx
   group::r-x
   other::---
   ```

   The output of `getfacl` precisely reflects the mapping of permission bits and ACL entries as described in *ACL Entries and File Mode Permission Bits* on page 504. The first three output lines display the name, owner, and owning group of the directory. The next three lines contain the three ACL entries *owner*, *owning group*, and *other*. In fact, in the case of this minimum ACL, the `getfacl` command does not produce any information you could not have obtained with `ls`.

   Your first modification of the ACL is the assignment of read, write, and execute permissions to an additional user `jane` and an additional group `djungle`.

   ```
   setfacl -m user:jane:rwx,group:djungle:rwx mydir
   ```

The option `-m` prompts `setfacl` to modify the existing ACL. The following argument indicates the ACL entries to modify (several entries are separated by commas). The final part specifies the name of the directory to which these modifications should be applied.

Use the `getfacl` command to take a look at the resulting ACL.

```
getfacl mydir

# file: mydir
# owner: tux
# group: project3
user::rwx
user:jane:rwx
group::r-x
group:djungle:rwx
mask::rwx
other::---
```

In addition to the entries initiated for the user `jane` and the group `djungle`, a *mask* entry has been generated. This *mask* entry is set automatically to reduce all entries in the *group class* to a common denominator. Furthermore, `setfacl` automatically adapts existing *mask* entries to the settings modified, provided you do not deactivate this feature with `-n`. *mask* defines the maximum effective access permissions for all entries in the *group class*. This includes: *named user*, *named group*, and *owning group*. Thus, the *mask* entry corresponds to the *group class* permission bits that would be displayed by `ls -dl mydir` as described in *ACL Entries and File Mode Permission Bits* on page 504.

```
ls -dl mydir drwxrwx---+ ... tux project3 ... mydir
```

As expected, the *group class* permission bits now reflect the *mask* entry. Additionally, the first column of the output contains a `+`, which points to an *extended* ACL.

3. According to the output of the `ls` command, the permissions for the *mask* entry include write access. Traditionally, permission bits of this kind would indicate that the *owning group* (here: `project3`) also has write access to the directory `mydir`. However, the effective access permissions for the *owning group* are defined as the intersection of the permissions defined for the *owning group* and *mask* — `r-x` in our example (see Table B.2 on page 504). Nothing changed here even after the addition of the ACL entries.

Edit the *mask* entry with `setfacl` or `chmod`:

```
chmod g-w mydir
ls -dl mydir

drwxr-x---+ ... tux project3 ... mydir

getfacl mydir

# file: mydir
# owner: tux
# group: project3
user::rwx
user:jane:rwx          # effective: r-x
group::r-x
group:djungle:rwx      # effective: r-x
mask::r-x
other::---
```

After having used the `chmod` command to disable the write access from the *group class* bits, the output of the `ls` command is sufficient to see that the *mask* bits were adjusted with `chmod`. This is even more evident from the output of `getfacl`. `getfacl` adds comments for all entries whose effective permission bits do not correspond to those originally set, as they are filtered by the *mask* entry. Of course, you can use `chmod` to restore the original state at any time:

```
chmod g+w mydir
ls -dl mydir

drwxrwx---+ ... tux project3 ... mydir

getfacl mydir

# file: mydir
# owner: tux
# group: project3
user::rwx
user:jane:rwx
group::r-x
group:djungle:rwx
mask::rwx
other::---
```

A Directory with a Default ACL

Directories can be equipped with a special kind of ACL — a default ACL. The default ACL defines the access permissions all objects under this directory inherit when they are created. A default ACL affects subdirectories as well as files.

Effects of a Default ACL

Basically, the permissions in a default ACL are handed down in two ways:

- A subdirectory inherits the default ACL of the parent directory both as its own default ACL and as an access ACL.
- A file inherits the default ACL as its own access ACL

All system calls that create file system objects use a `mode` parameter that defines the access permissions for the newly created file system object:

- If the parent directory does not have a default ACL, an intersection of the permissions defined in the `mode` parameter and those in the current `umask` is formed and assigned to the object.
- If a default ACL exists for the parent directory, the permission bits are determined according to the intersection of the value of the `mode` parameter and the permissions defined in the default ACL and assigned to the object.

Application of Default ACLs

The following three examples show the main operations for directories and default ACLs:

- Creating a default ACL for an existing directory
- Creating a subdirectory in a directory with default ACL
- Creating a file in a directory with default ACL

1. Add a default ACL to the existing directory `mydir`:

   ```
   setfacl -d -m group:djungle:r-x mydir
   ```

   The option `-d` of the `setfacl` command prompts `setfacl` to perform the following modifications (option `-m`) in the default ACL.

Take a closer look at the result of this command:

```
getfacl mydir

# file: mydir
# owner: tux
# group: project3
user::rwx
user:jane:rwx
group::r-x
group:djungle:rwx
mask::rwx
other::---
default:user::rwx
default:group::r-x
default:group:djungle:r-x
default:mask::r-x
default:other::---
```

`getfacl` returns both the access ACL and the default ACL. The lines that begin with `default` form the default ACL. Although you merely executed the `setfacl` command with an entry for the `djungle` group for the default ACL, `setfacl` automatically copied all other entries from the access ACL to form a valid default ACL. Default ACLs do not have a direct effect on access permissions, they only come into play when file system objects are created. When handing down the permissions, only the default ACL of the parent directory is taken into consideration.

2. In the next example, use `mkdir` to create a subdirectory in `mydir`, which will "inherit" the default ACL.

```
mkdir mydir/mysubdir
getfacl mydir/mysubdir

# file: mydir/mysubdir
# owner: tux
# group: project3
user::rwx
group::r-x
group:djungle:r-x
mask::r-x
other::---
default:user::rwx
default:group::r-x
default:group:djungle:r-x
```

```
default:mask::r-x
default:other::---
```

As expected, the newly-created subdirectory `mysubdir` has the permissions from the default ACL of the parent directory. The access ACL of `mysubdir` is an exact reflection of the default ACL of `mydir`, just as the default ACL that this directory will hand down to its subordinate objects.

3. Use `touch` to create a file in the `mydir` directory:

```
touch mydir/myfile
ls -l mydir/myfile

-rw-r-----+ ... tux project3 ... mydir/myfile

getfacl mydir/myfile

# file: mydir/myfile
# owner: tux
# group: project3
user::rw-
group::r-x          # effective:r--
group:djungle:r-x   # effective:r--
mask::r--
other::---
```

Important in this example: `touch` passes on `mode` with the value `0666`, which means that new files are created with read and write permissions for all user classes, provided no other restrictions exist in `umask` or in the default ACL (see *Effects of a Default ACL* on page 509).

If effect, this means that all access permissions not contained in the `mode` value are removed from the respective ACL entries. Although no permissions were removed from the ACL entry of the *group class*, the *mask* entry was modified to mask permissions not set via `mode`.

This approach ensures the smooth interaction of applications, such as compilers, with ACLs. The compiler can create files with restricted access permissions and subsequently mark them as executable simply by changing the file mode permission bits with `chmod`. The `mask` mechanism makes sure that the respective users and groups are assigned the permissions they are granted in the default ACL.

The ACL Check Algorithm

The following section provides brief information on the check algorithm applied to all processes or applications before they are granted access to an ACL-protected file system object. As a basic rule, the ACL entries are examined in the following sequence: *owner*, *named user*, *owning group* or *named group*, and *other*. The access is handled in accordance with the entry that best suits the process; permissions do not accumulate.

Things are more complicated if a process belongs to more than one group and would potentially suit several *group* entries. An entry is randomly selected from the suitable entries with the required permissions. It is irrelevant which of the entries triggers the final result "access granted". Likewise, if none of the suitable *group* entries contains the correct permissions, a randomly selected entry triggers the final result "access denied".

Outlook

As described in the preceding sections, ACLs can be used to implement very complex permission scenarios that fully meet modern applications. The traditional permission concept and ACLs can be combined in a smart manner.

However, some important applications still lack ACL support. Except for the star archiver, there are no backup applications that guarantee the full preservation of ACLs during a backup.

Basic file commands (`cp`, `mv`, `ls`, etc.) already support ACLs. Editors and file managers (such as Konqueror) do not yet offer any ACL support. ACLs keep getting lost when files are copied with the file manager Konqueror. If, for example, a file with an access ACL is modified in an editor, the backup mode of the editor used will determine whether or not the access ACL is preserved following the modification:

- If the editor writes the changes to the original file, the access ACL will be preserved.
- If the editor saves the updated contents to a new file that is subsequently renamed to the old file name, the ACLs may be lost, unless the editor supports ACLs.

The more applications support ACLs, the more it will be possible to exploit the great potential of this feature.

Tip

Additional information

Detailed information about ACLs is available at the following URLs:

`http://sdb.suse.de/en/sdb/html/81_acl.html`

`http://acl.bestbits.at/`

and in the man page for `getfacl` (`man 1 getfacl`), the man page for `acl` (`man 5 acl`), and the man page for `setfacl` (`man 1 setfacl`).

Tip