

Appendix for Chapter 16: Access Control Lists

If all you had to go on was written in Chapter 16, you’d think that Terminal offers no way to assign file permissions to *multiple* owners or groups, as you can with the Finder’s Get Info window.

In fact, you can do anything to permissions with Terminal that you can in the Finder—and more. Once you learn how to manage Leopard’s new *access control lists* (ACLs), you can keep one particular person from deleting one particular file, or allow only one group to add files but not subdirectories to a parent directory.

You don’t need any special new commands to manage ACLs, either; you can do it all with the old standby commands in Terminal, *ls* and *chmod*. To examine your home directory’s files for ACLs, for example, use *ls -l* as before:

```
MacChris:~ chris$ ls -l
total 0
drwx-----@ 3 chris  staff  102 Nov  4 10:59 Desktop
drwx-----+ 5 chris  staff  170 Nov 13 11:45 Documents
drwx-----+ 4 chris  staff  136 Nov  4 11:03 Downloads
drwx-----+ 27 chris  staff  918 Nov  4 11:32 Library
drwx-----+ 3 chris  staff  102 Nov  4 10:59 Movies
```

That + at the end of the permission codes means that this directory has an ACL. But if an item also has an *extended attribute*, you’ll get an @ sign instead of the +, as shown here on the Desktop directory. You won’t know for sure if it *also* has an ACL.

To zoom in on any ACLs, then, just add the *-e* flag to *ls -l*:

```
MacChris:~ chris$ ls -le
total 0
drwx-----@ 3 chris  staff  102 Nov  4 10:59 Desktop
0: group:everyone deny delete
drwx-----+ 5 chris  staff  170 Nov 13 11:45 Documents
0: group:everyone deny delete
1: user:msilver allow list,add_file
drwx-----+ 4 chris  staff  136 Nov  4 11:03 Downloads
```

```
0: group:everyone deny delete
```

You still see the @ sign, but below every other item with an ACL, there’s an additional line: *0: group:everyone deny delete*. This is the actual permissions rule or Access Control Entry (“ACE”) contained by the ACL. An ACL can have many ACEs, and they’re numbered beginning with 0.

Take a closer look at the ACE “*0: group:everyone deny delete*”; it reads like this:

- **0.** In other words, this is ACE number 0.
- **group:everyone.** The ACE controls the access of a group, which is named “everyone”. (The *everyone* group is created by the system and means what it says.)
- **deny delete.** Every ACE either denies or allows some kind of access. This one denies the access (that is, the permission) to delete the directory.

In short, this complete rule says: “Don’t let anyone delete this directory.” And *that’s* why you’re prevented from deleting these folders from your very own home directory.

If you don’t like the thought of being barred from mucking with your own Home folder, there are several ways to get around it. (You’ll want to experiment on a new test account, however, so you don’t hang yourself with the rope you’re about to receive.)

The easiest way to remove the directory is to do it as the *root user*. This very special account holder has free reign, unencumbered by either permissions system. Just *sudo rm -rf* the directory, and it’s gone.

Instead of actually removing the directory right away, you could use *chmod* to add another ACE, one that provides *allow* delete permissions for a specific account (or group). You’d then be able to remove the directory anytime, using Terminal or the Finder, but still keep any other account from deleting it.

To give Chris’s account permission to delete his own Downloads directory, the command would be:

```
chmod +a# 0 'chris allow delete' Downloads
```

The command works like this: *chmod +a* creates a new ACL (or ACE if there’s already an ACL). Numbering the new ACE is optional, but in this case it’s needed to override the existing deny rule. Therefore, the # sign and the number for the new ACE, 0, come after the *a+*. (The existing ACE 0 gets bumped to number 1.) The actual access definition comes next, surrounded in single quotes, followed, finally, by name of the item getting ACed. The result would look like this:

```
drwx-----+ 2 chris  staff   68 Nov 13 21:14 Downloads
0: user:chris allow delete
1: group:everyone deny delete
```

Chris can now trash the folder either in the Finder or with *rm -rf* in Terminal.

A third way to get around an ACL is to just remove it completely. Do this by giving *chmod* the -N flag and specifying the target. In this case, *chmod -N Downloads* would do the trick.

You’re not likely to run into many permissions requirements that ACLs can’t solve, especially when you need to share computers with lots of people. If fiddling with ACLs sounds like your cup of tea, don’t forget to have a good look at the ACL section of the *chmod* manpage.

Note: When you create an account, it gets added to at least one group—its *primary* group. The primary group then becomes the *group owner* for any files created by that account.

If you inspect the icons in a Home directory, you’ll find that they belong to a group called *staff*, the default primary group for all new user accounts on Leopard.

Mac OS X versions before Panther used this arrangement, too, but Mac OS X 10.3 and 10.4 (Panther and Tiger) used a different scheme. In those versions, when you created an account named *chris*, Mac OS X created a *chris* group with only one member; that became the account’s *primary* group. In fact, if you’ve upgraded to Leopard from Panther or Tiger, you’ll find that your old Home folder icons are still owned by a group named for your account.

Why Apple’s about-faces? ACLs let the system once again use the more flexible shared primary group *staff*, while keeping the improved security provided in Panther and Tiger by the single-member primary groups.