Using ACLs with Fedora Core 2 (Linux Kernel 2.6.5)

| Back to Index |

By Van Emery

Table of Contents

- Introduction
- Assumptions
- Getting Started
- Using ACLs
- More `setfacl` Details and Examples
- Example Scenario
- The Default ACL
- Using `cp` and `mv` with ACLs
- Copying ACLs
- Archive and Restore Files with ACLs
- XFS Notes
- Final Notes
- Additional Resources

Introduction

What are ACLs and why would you want to use them?

ACLs are Access Control Lists for files and directories. They are based on the IEEE's POSIX 1003.1e draft 17, also known simply as POSIX.1e. ACLs are an addition to the standard Unix file permissions (r,w,x,-) for User, Group, and Other. ACLs give users and administrators flexibility and fine-grained control over who can read, write, and execute files. This can all be done without adding mysterious groups and pestering the system administrator.

Commercial Unix operating systems (except SCO) have all had ACL functionality for quite awhile. Microsoft's NTFS also has similar capabilities. FreeBSD 5.x supports POSIX.1e ACLs as well. The new Linux 2.6 kernel supports ACLs for EXT2, EXT3, XFS, JFS, and ReiserFS.

Fedora Core 2, Red Hat's first distribution with a 2.6 kernel, is a good vehicle for taking Linux ACLs for a test drive. This document is a basic HOWTO/tutorial on using ACLs with Fedora.

Assumptions

- You are using Fedora Core 2
- You have another partition besides /, /**boot**, and **swap** defined, or some unpartitioned free space on one of your disks
- You are using the EXT2, EXT3, or XFS filesystems
- You can login as root

If you have no free space on the disk, and all of your files and binaries are located in the root (/) partition, then **you may not want to experiment with ACLs**.

Note: Both JFS and ReiserFS *can* support ACLs under Linux, but Fedora Core 2 does not appear to support it. ReiserFS cannot be mounted with the "acl" option, and `jfs_mkfs` appears to be seriously broken. Therefore, this HOWTO will be limited to EXT2, EXT3, and XFS.

Getting Started

Assuming you have an EXT2 or EXT3 partition that you are willing to use for testing, we can get started. On my test machine, I have the following available partitions:

- `/dev/hda5 /home (ext3)`
- `/dev/hda9 /XFS (xfs)`

For the examples in this HOWTO, I will be using the home directory of user "tristan", which is `/home/tristan`. Note that this directory belongs to a separate Linux partition, *not* the the root (/) partition. If you have some extra unpartitioned space on one of your disks, this would be a good time to create a test partition. You can do this with the `fdisk` command, then you can format it with the `mke2fs` command. Make sure you read up on all the required steps before you do this, otherwise you can nuke your system, disk, or data!

If you did not install Fedora Core 2 with the "XFS" option, but you want to try ACLs on XFS, take a look at the XFS Notes section.

You will need to unmount the partitions of your choice, and then remount them with the "acl" option. First, I made a copy of my `/etc/fstab` file:

```
[root@fc2 root]# cp -v /etc/fstab /etc/fstab.org
`/etc/fstab' -> `/etc/fstab.org'
```

Then, I made the following modifications in **red** to the `/etc/fstab` config file. For clarity, I am only including hard disk entries:

```
LABEL=/                 /                       ext3    defaults        1 1
```

```
LABEL=/boot             /boot                   ext3    defaults        1 2
LABEL=/home             /home                   ext3    rw,acl          1 2
LABEL=/tmp              /tmp                    ext3    defaults        1 2
LABEL=/usr              /usr                    ext3    defaults        1 2
LABEL=/var              /var                    ext3    defaults        1 2
/dev/hda8               swap                    swap    defaults        0 0
/dev/hdd1               /Data                   ext3    ro,noatime      1 2
LABEL=/XFS              /XFS                    xfs     rw,noatime      0 2
```

Now, you will need to remount the `/home` partition with the "acl" option. The easiest way to do this is with the "remount" option, since it will work even while the partition is in use:

```
[root@fc2 root]# mount -v -o remount /home
/dev/hda5 on /home type ext3 (rw,acl)
```

Another way to remount the partition with the "acl" option is to make sure that nobody else is on the sytem and the `/home` partition is not in use, then unmount, then mount the partition:

```
[root@fc2 root]# umount /home
[root@fc2 root]# mount /home

[root@fc2 root]# mount -l
/dev/hda2 on / type ext3 (rw) [/]
/dev/hda1 on /boot type ext3 (rw) [/boot]
/dev/hda5 on /home type ext3 (rw,acl) [/home]
/dev/hda7 on /tmp type ext3 (rw) [/tmp]
/dev/hda3 on /usr type ext3 (rw) [/usr]
/dev/hda6 on /var type ext3 (rw) [/var]
/dev/hdd1 on /Data type ext3 (ro,noatime) []
/dev/hda9 on /XFS type xfs (rw,noatime) [/XFS]
```

If you had trouble unmounting your target partitions, you may need to drop to single user mode with the `init 1` command. This should allow you to unmount the filesystems. After that, you can remount the filesystems and issue an `init 3` or `init 5` command to put you back into your regular operating environment.

Using ACLs

Now, we can actually start using ACLs. The basic commands that we are interested in are:

- `getfacl`
- `setfacl`

We will first look at the `getfacl` command. The owner of the directory we will be working with is "tristan", and the guest user will be "axel" and the guest group will be "lensmen". First, create a test file, then look at the permissions and the ACL:

```
[tristan@fc2 tristan]$ cd /home/tristan
[tristan@fc2 tristan]$ cp /etc/services pizza
```

```
[tristan@fc2 tristan]$ ls -l pizza
-rw-r--r--  1 tristan tristan 19936 May 28 16:59 pizza

[tristan@fc2 tristan]$ getfacl pizza
# file: pizza
# owner: tristan
# group: tristan
user::rw-
group::r--
other::r--
```

So far, there is nothing very exciting to see. Now, let's change the ACL so that user "axel" can read and write to the file:

```
[tristan@fc2 tristan]$ setfacl -m u:axel:rw- pizza
[tristan@fc2 tristan]$ getfacl pizza
# file: pizza
# owner: tristan
# group: tristan
user::rw-
user:axel:rw-
group::r--
mask::rw-
other::r--

[tristan@fc2 tristan]$ ls -l pizza
-rw-rw-r--+ 1 tristan tristan 19936 May 28 16:59 pizza
```

You will notice that there is now an extra user entry in the ACL, and there is a "+" next to the file in the output from the `ls` command. The "+" indicates that an ACL has been applied to the file or directory. Now, let's add a group ("lensmen") and another user ("tippy") to the ACL for `pizza`:

```
[root@fc2 tristan]# setfacl -m u:tippy:r--,g:lensmen:r-- pizza

[root@fc2 tristan]# getfacl pizza
# file: pizza
# owner: tristan
# group: tristan
user::rw-
user:axel:rw-
user:tippy:r--
group::r--
group:lensmen:r--
mask::rw-
other::r--
```

Hmmm...what's the **mask** entry? This is the effective rights mask. This entry limits the effective rights granted to all ACL groups and ACL users. The traditional Unix User, Group, and Other entries are not affected. If the mask is more restrictive than the ACL permissions that you grant, then the mask takes precedence. For example, let's change the mask to "`r--`" and give user "tippy" and group "lensmen" the permissions `rwx`, and see what happens:

```
[tristan@fc2 tristan]$ setfacl -m u:tippy:rwx,g:lensmen:rwx pizza
```

```
[tristan@fc2 tristan]$ setfacl -m mask::r-- pizza

[tristan@fc2 tristan]$ getfacl --omit-header pizza
user::rw-
user:axel:rw-                   #effective:r--
user:tippy:rwx                  #effective:r--
group::r--
group:lensmen:rwx               #effective:r--
mask::r--
other::r--
```

The ACL now shows an "effective" rights mask. Even though "tippy" has been given `rwx` permissions, he actually only has `r--` permissions because of the mask.

In most cases, I want the effective mask to allow whatever permissions I granted to named users and groups, so my mask will be `rw-` or `rwx`. I will reset it like this:

```
[tristan@fc2 tristan]$ setfacl -m m::rw- pizza

[tristan@fc2 tristan]$ getfacl --omit pizza
user::rw-
user:axel:rw-
user:tippy:rw-
group::r--
group:lensmen:rwx               #effective:rw-
mask::rw-
other::r--
```

What about using the `setfacl` command to change normal User, Group, and Other permissions? No problem! This can be used instead of `chmod`:

```
[tristan@fc2 tristan]$ setfacl -m u::rwx,g::rwx,o:rwx pizza

[tristan@fc2 tristan]$ ls -l pizza
-rwxrwxrwx+ 1 tristan tristan 19965 May 29 09:31 pizza

[tristan@fc2 tristan]$ getfacl --omit pizza
user::rwx
user:axel:rw-
user:tippy:rw-
group::rwx
group:lensmen:rwx
mask::rwx
other::rwx
```

Note that the mask changed! Whenever you change the permissions of a user or a group with `setfacl`, the mask is changed to match. Therefore, if you want a restrictive mask, it must be applied *after* the user and group permissions are modified.

Another thing to keep in mind is that the `chmod` command does not alter the file's ACL...the ACL information will remain intact, *except* that the mask entry can change as described above.

More `setfacl` Details and Examples

The `setfacl` command has many options. In this section, we will look at some of the more useful ones.

Remove Specific Entries from an ACL

You can remove specific ACL entries with the `-x` option. In this example, we will remove the entry for user "tippy" and user "axel" but leave the other entries alone:

```
[tristan@fc2 tristan]$ getfacl --omit pizza
user::rwx
user:axel:rw-
user:tippy:rw-
group::rwx
group:lensmen:rwx
mask::rwx
other::rwx

[tristan@fc2 tristan]$ setfacl -x u:tippy,u:axel pizza

[tristan@fc2 tristan]$ getfacl --omit pizza
user::rwx
group::rwx
group:lensmen:rwx
mask::rwx
other::rwx
```

Remove Entire ACL

To completely remove an ACL from a file or directory:

```
[tristan@fc2 tristan]$ setfacl -b pizza
```

You can also use:

```
[tristan@fc2 tristan]$ setfacl --remove-all pizza
```

Using the `--set` Option

If you want to explicitly set all of the file permissions on a file or a group of files, you must use the `--set` option. This is different from the `-m` option, which only modifies the existing ACL. The `--set` option replaces all permissions and ACLs with the new values. When you use the `--set` option, all of the User, Group, and Other permissions *must* be defined. Here is an example:

```
[tristan@fc2 tristan]$ setfacl --set u::rw,g::rw,o::-,u:tippy:r pizza

[tristan@fc2 tristan]$ getfacl --omit pizza
user::rw-
user:tippy:r--
group::rw-
mask::rw-
other::---
```

Using `setfacl` Recursively

If you want to apply ACLs to an entire directory and all of its subdirectories, use the `-R` option. Given the directory hierarchy `/home/tristan/Level1/Level2/Level3/Level4`, the following command will add an ACL entry for group "lensmen" to all of the `Level*` directories and their contents:

```
[tristan@fc2 tristan]$ setfacl -R -m g:lensmen:r-x /home/tristan/Level1
```

Using ACL Entries from a File:

What if you have a lengthy ACL that needs to be used frequently? Rather than typing it over and over again on the command line, you can save the ACL as a text file and use it to apply ACLs to other files. For example, we will create the ACL config file `/home/tristan/myacl`:

```
user:axel:rw-
user:tippy:rw-
group:lensmen:r--
group:marty:r--
group:fafnir:r--
mask::rw-
other::---
```

Now, we can easily apply these ACL modifications to files:

```
[tristan@fc2 tristan]$ setfacl -M myacl test*

[tristan@fc2 tristan]$ ls -l test*
-rw-rw----+ 1 tristan tristan 168 May 30 09:41 test1
-rw-rw----+ 1 tristan tristan 168 May 30 09:42 test2
-rw-rw----+ 1 tristan tristan 168 May 30 09:42 test3

[tristan@fc2 tristan]$ getfacl test1
# file: test1
# owner: tristan
# group: tristan
user::rw-
user:axel:rw-
user:tippy:rw-
group::rw-
group:marty:r--
group:lensmen:r--
group:fafnir:r--
mask::rw-
other::---
```

Note on UID, GID, and Permissions

When you are using `setfacl`, you can use numeric UIDs and GIDs instead of the actual names. The UIDs and GIDs do not have to exist yet. If you use names, then they *must* exist or you will get an error. You can use the

```
getfacl --numeric filename
```

command to view the numeric values.

Also, when you are specifying permissions, you can use octal permissions (0-7) instead of (r,w,x,-).

Example Scenario

Now that we have seen basic command usage, let's use a practical example to learn some more about ACLs. Tippy is working with Tristan on a project. He needs to be able to read, write, create, and delete files related to the project, which are located in Tristan's home directory. Tristan wants to do this without bothering the system administrator with requests for new groups and group membership changes. When the project is over, Tristan will remove the permissions for user "tippy" without bothering the sysadmin.

All of the project files are located in `/home/tristan/Project`. Here is how Tristan will handle the situation:

```
[tristan@fc2 tristan]$ setfacl -m user:tippy:--x /home/tristan
[tristan@fc2 tristan]$ getfacl /home/tristan
getfacl: Removing leading '/' from absolute path names
# file: home/tristan
# owner: tristan
# group: tristan
user::rwx
user:tippy:--x
group::---
mask::--x
other::---

[tristan@fc2 tristan]$ setfacl -R -m u:tippy:rwx,o::--- Project
[tristan@fc2 tristan]$ getfacl Project
# file: Project
# owner: tristan
# group: tristan
user::rwx
user:tippy:rwx
group::rwx
mask::rwx
other::---

[tristan@fc2 tristan]$ cd Project
[tristan@fc2 Project]$ ls -l
total 1560
-rwxrwx---+ 1 tristan tristan  86532 May 29 14:02 libgssapi_krb5.so
-rwxrwx---+ 1 tristan tristan  86532 May 29 14:02 libgssapi_krb5.so.2
-rwxrwx---+ 1 tristan tristan  86532 May 29 14:02 libgssapi_krb5.so.2.2
-rwxrwx---+ 1 tristan tristan 423572 May 29 14:02 libkrb5.so
-rwxrwx---+ 1 tristan tristan 423572 May 29 14:02 libkrb5.so.3
-rwxrwx---+ 1 tristan tristan 423572 May 29 14:02 libkrb5.so.3.2
[tristan@fc2 Project]$ getfacl --omit libkrb5.so
user::rwx
user:tippy:rwx
```

```
group::r-x
mask::rwx
other::---
```

Now, Tippy can access the `/home/tristan/Project` directory. He can read, modify, add, and delete files. However, he cannot delete the `Project` directory, nor can he view any other files in Tristan's home directory. This is good, because Tippy likes to test his limits. Let's see what he can and can't do:

```
[tippy@fc2 tippy]$ cd /home/tristan
[tippy@fc2 tristan]$ ls
ls: .: Permission denied
[tippy@fc2 tristan]$ rm -rf Project
rm: cannot remove `Project': Permission denied
[tippy@fc2 tristan]$ cd Project
[tippy@fc2 Project]$ ls -l
total 1560
-rwxrwx---+ 1 tristan tristan  86532 May 29 14:02 libgssapi_krb5.so
-rwxrwx---+ 1 tristan tristan  86532 May 29 14:02 libgssapi_krb5.so.2
-rwxrwx---+ 1 tristan tristan  86532 May 29 14:02 libgssapi_krb5.so.2.2
-rwxrwx---+ 1 tristan tristan 423572 May 29 14:02 libkrb5.so
-rwxrwx---+ 1 tristan tristan 423572 May 29 14:02 libkrb5.so.3
-rwxrwx---+ 1 tristan tristan 423572 May 29 14:02 libkrb5.so.3.2
[tippy@fc2 Project]$ touch status-report.txt

[tippy@fc2 Project]$ date >> libkrb5.so.3
[tippy@fc2 Project]$ rm libkrb5.so.3
[tippy@fc2 Project]$ ls -l
total 1136
-rwxrwx---+ 1 tristan tristan  86532 May 29 14:02 libgssapi_krb5.so
-rwxrwx---+ 1 tristan tristan  86532 May 29 14:02 libgssapi_krb5.so.2
-rwxrwx---+ 1 tristan tristan  86532 May 29 14:02 libgssapi_krb5.so.2.2
-rwxrwx---+ 1 tristan tristan 423572 May 29 14:02 libkrb5.so
-rwxrwx---+ 1 tristan tristan 423572 May 29 14:02 libkrb5.so.3.2
-rw-rw-r--  1 tippy   tippy        0 May 29 16:06 status-report.txt
```

Now, after the project is complete, it is a simple matter for user Tristan to revoke Tippy's access to `/home/tristan`:

```
[tristan@fc2 tristan]$ setfacl -x u:tippy: /home/tristan
[tristan@fc2 tristan]$ getfacl /home/tristan
getfacl: Removing leading '/' from absolute path names
# file: home/tristan
# owner: tristan
# group: tristan
user::rwx
group::---
mask::---
other::---
```

If user "tippy" decides to snoop around in `/home/tristan/Project` again, he will not be able to:

```
[tippy@fc2 tippy]$ cd /home/tristan
-bash: cd: /home/tristan: Permission denied
[tippy@fc2 tippy]$ ls /home/tristan/Project
ls: /home/tristan/Project: Permission denied
```

Note that this entire example was done without having to involve the system administrator!

The Default ACL

Up until now, we have been looking at the *access* ACL. There is also another type of ACL, called the *default* ACL. The default ACL is only applied to directories, and it defines the permissions that a newly created file or directory *inherits* from its parent directory.

When you create a new directory inside a directory that already has a default ACL, the new directory inherits the default ACL both as its access ACL *and* its default ACL.

Here is an example of defining a default ACL for a directory, and what happens when files and directories are created underneath that directory:

```
[tristan@fc2 tristan]$ mkdir Plato

[tristan@fc2 tristan]$ setfacl --set u::rwx,g::r-x,o::- Plato

[tristan@fc2 tristan]$ setfacl -d --set u::rwx,u:tippy:rwx,u:axel:rx,g::rx,g:lensmen:rx,o::- Plato
[tristan@fc2 tristan]$ getfacl Plato
# file: Plato
# owner: tristan
# group: tristan
user::rwx
group::r-x
other::---
default:user::rwx
default:user:axel:r-x
default:user:tippy:rwx
default:group::r-x
default:group:lensmen:r-x
default:mask::rwx
default:other::---

[tristan@fc2 tristan]$ cd Plato
[tristan@fc2 Plato]$ touch guitar
[tristan@fc2 Plato]$ getfacl guitar
# file: guitar
# owner: tristan
# group: tristan
user::rw-
user:axel:r-x                   #effective:r--
user:tippy:rwx                  #effective:rw-
group::r-x                      #effective:r--
group:lensmen:r-x               #effective:r--
mask::rw-
other::---

[tristan@fc2 Plato]$ mkdir Zep
[tristan@fc2 Plato]$ getfacl Zep
# file: Zep
# owner: tristan
# group: tristan
user::rwx
user:axel:r-x
```

```
user:tippy:rwx
group::r-x
group:lensmen:r-x
mask::rwx
other::---
default:user::rwx
default:user:axel:r-x
default:user:tippy:rwx
default:group::r-x
default:group:lensmen:r-x
default:mask::rwx
default:other::---

[tristan@fc2 Plato]$ cd Zep
[tristan@fc2 Zep]$ touch airship
[tristan@fc2 Zep]$ getfacl airship
# file: airship
# owner: tristan
# group: tristan
user::rw-
user:axel:r-x                #effective:r--
user:tippy:rwx               #effective:rw-
group::r-x                   #effective:r--
group:lensmen:r-x            #effective:r--
mask::rw-
other::---
```

The umask has no effect if a default ACL exists. In the following example, the umask is honored when a file is created in the `/home/tristan` directory, which has no default ACL. When a file is created under `/home/tristan/Plato`, which has a default ACL, you can see that the umask is ignored:

```
[tristan@fc2 tristan]$ umask ugo=
[tristan@fc2 tristan]$ umask
0777
[tristan@fc2 tristan]$ touch button
[tristan@fc2 tristan]$ ls -l button
----------  1 tristan tristan 0 Jun  1 00:47 button

[tristan@fc2 tristan]$ cd Plato
[tristan@fc2 Plato]$ touch switch
[tristan@fc2 Plato]$ ls -l switch
-rw-rw----+ 1 tristan tristan 0 Jun  1 00:47 switch
```

You can also modify and create default ACLs with another syntax, prefixing the u, g, or o entries with a "`d`" :

```
[tristan@fc2 tristan]$ setfacl -m d:u:axel:rwx,d:g:lensmen:rwx Plato
[tristan@fc2 tristan]$ getfacl Plato
# file: Plato
# owner: tristan
# group: tristan
user::rwx
group::r-x
other::---
default:user::rwx
default:user:axel:rwx
default:user:tippy:rwx
default:group::r-x
```

```
default:group:lensmen:rwx
default:mask::rwx
default:other::---
```

Using `cp` and `mv` with ACLs

Three major file utilities, `ls`, `cp`, and `mv` have been updated to handle ACLs. The `mv` command will always preserve ACLs if it is possible. If it is not possible, it will issue a warning. The `cp` command will only preserve ACLs if used with the `-p` or `-a` options.

In both cases, if you are trying to copy/move from a filesystem that supports ACLs to a filesystem that does not, only the standard Unix permissions will be retained. In the example below, you can see that using the `cp -p` command within the ACL-enabled `/home` filesystem worked, and using the same command to copy the file to the `/root` directory (which is *not* ACL-enabled) resulted in an error message. As root, do the following:

```
[root@fc2 root]# cd /home/tristan
[root@fc2 tristan]# mkdir ACL
[root@fc2 tristan]# cp -p pizza ACL/pizza
[root@fc2 tristan]# ls -l ACL/pizza
-rw-rwx---+ 1 tristan tristan 19965 May 29 09:31 ACL/pizza

[root@fc2 tristan]# cp -p pizza /root
cp: preserving permissions for `/root/pizza': Operation not supported
[root@fc2 tristan]# ls -l /root/pizza
-rw-rwx---  1 tristan tristan 19965 May 29 09:31 /root/pizza
```

Copying ACLs

If you already have a file with a complex ACL, you can easily copy that ACL to other files by piping the output of a `getfacl` command into the `setfacl` command. Here is an example of copying the ACL from `bingo.txt` to all of the files starting with "test":

```
[tristan@fc2 Compaq]$ ls -l
total 4
-rw-rw----+ 1 tristan tristan 0 Jun  2 09:52 bingo.txt
-rw-rw----  1 tristan tristan 0 Jun  2 09:53 testa1
-rw-rw----  1 tristan tristan 0 Jun  2 09:53 testa2
-rw-rw----  1 tristan tristan 0 Jun  2 09:55 testa3
-rw-rw----  1 tristan tristan 0 Jun  2 09:53 testa4
-rw-rw----  1 tristan tristan 0 Jun  2 09:55 testa5

[tristan@fc2 Compaq]$ getfacl bingo.txt | setfacl --set-file=- test*

[tristan@fc2 Compaq]$ ls -l
total 24
-rw-rw----+ 1 tristan tristan 0 Jun  2 09:52 bingo.txt
-rw-rw----+ 1 tristan tristan 0 Jun  2 09:53 testa1
-rw-rw----+ 1 tristan tristan 0 Jun  2 09:53 testa2
-rw-rw----+ 1 tristan tristan 0 Jun  2 09:55 testa3
-rw-rw----+ 1 tristan tristan 0 Jun  2 09:53 testa4
-rw-rw----+ 1 tristan tristan 0 Jun  2 09:55 testa5
```

```
[tristan@fc2 Compaq]$ getfacl --omit testa5
user::rw-
user:axel:rw-
user:tippy:rw-
group::rw-
group:marty:r--
group:lensmen:r--
group:fafnir:r--
mask::rw-
other::---
```

You can also archive all of the ACLs from an entire directory tree, then restore them later. You might want to do this if you are recovering files from backup media that does not support ACLs, like CD-ROM. Here is an example of archiving/saving all of the ACLs in the `/home/tristan/Tree` directory tree, and restoring them.

There are 898 files in this tree:

```
[tristan@fc2 tristan]$ du -h Tree
9.5M    Tree/A/B/C/D
19M     Tree/A/B/C
29M     Tree/A/B
38M     Tree/A
9.5M    Tree/AA/BB/CC/DD
19M     Tree/AA/BB/CC
29M     Tree/AA/BB
38M     Tree/AA
86M     Tree
```

Now, let's archive the ACLs into a file in our home directory:

```
[tristan@fc2 tristan]$ getfacl -R Tree > Tree.facl
[tristan@fc2 tristan]$ ls -l Tree.facl
-rw-rw-r--  1 tristan tristan 120550 Jun  2 12:08 Tree.facl
```

Now, we will simulate restoring the files from CD without ACLs by stripping all of the ACLs off:

```
[tristan@fc2 tristan]$ setfacl -R -b Tree
```

Now we can restore all of the ACL entries with one command:

```
[tristan@fc2 tristan]$ setfacl --restore Tree.facl
```

Archive and Restore Files with ACLs

What if you want to archive/backup files or directories with ACLs? Besides `cp`, what is there? Unfortunately, `tar`, `cpio`, `pax`, and `dump` will not save and restore ACL information. You can use the `setfacl --restore` mechanism in conjunction with a standard archiving/ backup system, but that is far from ideal. The answer is `star`, a TAR-like utility that is included in the Fedora Core 2 distribution.

Quotes from Star's author:

Star is the fastest known implementation of a tar archiver.
Star is even faster than ufsdump in nearly all cases.

Sounds interesting, doesn't it?

We can archive the entire `/home/tristan/Tree` directory tree from our previous example. We have to use the `acl` and `-Hexustar` options in order to archive and restore the ACL data. Here we go:

```
[tristan@fc2 /]$ cd /home/tristan

[tristan@fc2 tristan]$ star -Hexustar -acl -c f=Tree.star Tree
star: 8201 blocks + 0 bytes (total of 83978240 bytes = 82010.00k).
```

Now we will simulate losing our files and restoring them from a Star archive:

```
[tristan@fc2 tristan]$ rm -rf Tree

[tristan@fc2 tristan]$ star -acl -x f=Tree.star
star: 8201 blocks + 0 bytes (total of 83978240 bytes = 82010.00k).
```

When you check out the `/home/tristan/Tree` directory tree, you will find that it has been restored along with all the ACLs!

rdiff-backup

If you want to use a disk-to-disk backup instead of a tape archiver, consider using rdiff-backup. The stable branch now supports ACLs.

XFS Notes - Setting Up an XFS Filesystem with ACLs

XFS natively supports POSIX.1e ACLs. Unless you installed Fedora Core 2 with the XFS option, you will need to install the XFS RPM packages in order to use XFS. They are located on FC-2 ISO disk #4, in the `Fedora/RPMS` directory. You will need to install the following packages:

- `xfsprogs-2.6.13-1.i386.rpm`
- `xfsprogs-devel-2.6.13-1.i386.rpm`

Use the `rpm -ivh xfsprogs*rpm` command and you will soon be ready to go.

You will need a spare partition for your XFS filesystem. In my case, I created a spare partition as `/dev/hda9`. You must now create an XFS filesystem:

```
[root@fc2 root]# mkfs.xfs -i size=512 -f -L "/XFS" /dev/hda9
meta-data=/dev/hda9              isize=512    agcount=8, agsize=61046 blks
         =                       sectsz=512
```

```
data     =                       bsize=4096   blocks=488368, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=1
naming   =version 2              bsize=4096
log      =internal log           bsize=4096   blocks=2560, version=1
         =                       sectsz=512   sunit=0 blks
realtime =none                   extsz=65536  blocks=0, rtextents=0
```

The `-i` option is used to specify the size of the inodes. 256 is the default, but 512 bytes per inode significantly increases the speed of ACL lookups.

Now, create a directory to act as the mountpoint:

```
[root@fc2 root]# mkdir /XFS
```

Now, we have to actually mount the new filesystem. Unlike EXT2 and EXT3, no "acl" option is necessary. XFS assumes that you want ACLs. Example:

```
[root@fc2 root]# mount -v -t xfs /dev/hda9 /XFS
/dev/hda9 on /XFS type xfs (rw)

[root@fc2 root]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda2             4.2G  197M  3.8G   5% /
/dev/hda1              97M  5.9M   86M   7% /boot
/dev/hda5             9.7G  128M  9.0G   2% /home
/dev/hda7             985M   17M  919M   2% /tmp
/dev/hda3              15G  3.2G   11G  23% /usr
/dev/hda6             4.9G  184M  4.4G   4% /var
/dev/hdd1              29G   11G   17G  38% /Data
/dev/hda9             1.9G  160K  1.9G   1% /XFS
```

Alternatively, you could have used the mount command with the disk label "/XFS" that was added when you created the XFS filesystem. Example:

```
[root@fc2 root]# mount -v -t xfs -L "/XFS" /XFS
mount: mounting /dev/hda9
/dev/hda9 on /XFS type xfs (rw)
```

The last step is to add an entry to `/etc/fstab` so that the filesystem/partition will be mounted automatically during system boot. Here is a sample entry:

```
LABEL=/XFS          /XFS            xfs     rw,noatime    0 2
```

You can now start using the filesystem with ACLs.

Final Notes

There are limits to how many ACL entries can be applied to each file or directory. The number is filesystem dependent. EXT2 and EXT3 can have up to 32 entries, and XFS can have up to 25. Reiser and JFS can have over 8,000.

Enabling and using ACLs on a filesystem can reduce performance. It does not make sense to use ACLs for the root partition (/), `/boot`, `/usr`, `/var`, etc. I can see ACLs being

very useful in `/home` and other user data partitions.

The only way to get familiar with Linux ACLs is to practice using them. Have fun with it!

Additional Resources

- Andreas Grünbacher's "POSIX Access Control Lists on Linux" whitepaper (local copy)
- Excellent article on Linux ACLs by Carla Schroeder
- Linux ACL Homepage
- rdiff-backup page
- `man getfacl`
- `man setfacl`
- `man acl`
- `man star`
- `/usr/share/doc/star-1.5a25` documentation
- Star ACL README file

Last updated: 2005-09-22

Using ACLs with Fedora Core 2 (Linux Kernel 2.6.5)

| Back to Index |