Extended ACLs (Access Control Lists)

Sometimes one would require different permissions for different users and groups. The problem is that a file may only have 1 owning user and 1 owning group. So if the requirement is to allocate different permissions for these entities, you could make use of ACL's.

ACL's may be implemented by root or the owning user of a file provided that the filesystem is mounted with the acl option. This is achieved by using the acl option in the options block (column 4) in /etc/fstab but a much better way to do this would be to implement the acl option as a mount option in the filesystem metadata using the following command:

```
# tune2fs -o acl /dev/sda2
```

Viewing an ACL

Any file which has a + sign to the right of the permissions block has been allocated an ACL which is viewable with the getfacl command.

Setting an ACL

The setfacl command can be used to set an ACL. Here is an example of adding and ACL for a user:

```
$ ll
total 0
-rw-rw-r--. 1 ricardo ricardo 0 Feb 20 00:42 foo
```

As you can see, there is no ACL attached to the file called foo.
Our objective is to implement an ACL on the file foo with the following properties:

- ricardo remains the owning user with rw permissions
- wheel is the owning group with r permissions
- others have all their access revoked
- members of the group 3stooges are to have rwx permissions
- members of the group 7dwarves are to have r-x permissions
- the user sleepy who is a member of the group 7dwarves must not have any permissions

To achieve upon this goal we would run the following commands:

```
# chown .wheel foo
```

The above command sets the owning group for the file foo to wheel.

```
# chmod 640 foo
```

The above command sets the permissions to rw- for the owning user, r-- for the owning group and others have no permissions allocated.

```
# setfacl -m g:3stooges:rwx,g:7dwarves:rx,u:sleepy:- foo
```

The final command insures that additionally, members of the group 3stooges are allocated rwx, members of the group 7dwarves are allocated rx and the user sleepy has no permissions.

The result would be:

```
$ ll foo
-rw-rwxr--+ 1 ricardo ricardo 0 Feb 20 00:42 foo
```

```
$ getfacl foo
# file: foo
# owner: ricardo
# group: ricardo
user::rw-
user:sleepy:---
group::rw-
group:3stooges:rwx
group:7dwarves:r-x
mask::rwx
other::r--
```

ACL's are always processed in the following order and processing is stopped at the first match:

1. owning user
2. named user(s)
3. owning group
4. named group(s)
5. others

So given that the user sleepy is logged in and is a member of the group 7dwarves, we would ask the following questions:

1. Is the user in question the owning user? Our answer is no

2. Is the user in question one of the named users? Here we have a match so we do not process any further.

So the permissions which would be effective for sleepy would --- (none)

If you thought that sleepy would derive permissions from his membership through the group 7dwarves then you processed beyond the match we had for the owning user which is not allowed.

> Note 1: ACLs are a quick and easy way to solve permission issues, but the more you have the more there are to maintain.
>
> Note 2: ACLs are not implemented in the index node of a file which means that when files are backed up using tools like tar, then the corresponding ACL is not backed up with it. We need to get creative and develop a script which backs up the ACLs with the files, but more about that later.

ACL's and inheritance

When an ACL is set on a directory, any newly created children will not inherit that ACL. To ensure this behavior, one would have to create a separate ACL called a default ACL.

```
# mkdir /data

# setfacl -m u:fred:rwx,u:betty:rw,g:wheel:rx,g:flintstones:rwx,g:rubbles:rwx /data
```

fred is a member of the group flintstones
wilma is a member of the group flintstones
barney is a member of the group rubbles and wheel
betty is a member of the group rubbles

```
# getfacl /data
getfacl: Removing leading '/' from absolute path names
# file: data
# owner: root
# group: root
user::rwx
user:betty:rw-
user:fred:rwx
group::r-x
group:wheel:r-x
group:flintstones:rwx
```

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

Let's determine what permissions the users actually have to the /data directory:

fred has rwx (match to a named user)
wilma has rwx (match to a named group)
barney has rwx (match to multiple named groups in which case the permissions are combined)
betty has rw (match to a named user)

When children are created below the directory /data, those ACL's are not passed on.

```
# touch foo.1
# ll
total 0
-rw-r--r--. 1 root root 0 May 19 09:56 foo.1
```

There's no + sign next to the permissions, therefore the ACL was not passed on.

To ensure this one would create a default ACL.

!!! NOTE !!! A default ACL affects newly created children only.

!!! NOTE !!! A default ACL can be different to the actual ACL which is set on a directory.

```
# setfacl -d -m u:fred:rwx,u:dino:- /data
```

This ensures that the user fred always has rwx to newly created children of /data and the user dino always has no permissions to newly children created of the same directory.

The mask

A great way to prevent permissions from being inherited to a child is to implement a mask.

The mask specifies which permissions, should you already have it, you're allowed to use on that file.

Let's say you have apples, bananas and pears and the mask says that you may only eat apples and pears, while you have been given bananas, you may not consume them.

```
# getfacl foo.2
# file: foo.2
# owner: root
# group: root
user::---
user:fred:rwx                   #effective:rw-
user:dino:---
group::r-x                      #effective:r--
mask::rw-
other::r--
```

The mask above is set to rw- and the user fred has been given rwx. Therefore his effective permissions are rw-

To change the mask for that file, we use the command:

```
# setfacl -m m:rwx /data/foo.2
```

```
# getfacl /data/foo.2
getfacl: Removing leading '/' from absolute path names
# file: data/foo.2
# owner: root
# group: root
user::---
user:fred:rwx
user:dino:---
group::r-x
mask::rwx
other::r--
```

As you can now see, the mask allows all permissions which have been allocated.

!!! NOTE !!! Many filesystems support these extended ACL's including NTFS and NFS so one can implement a really robust file server

!!! WARNING !!! Many GUI programs (like Nautilus) do not support and recognize ACL's

---

Lab Activity

Format your flash drive with the ext4 filesystem and mount it to /data.

Make sure that the /data filesystem supports extended ACL's and do not use /etc/fstab to implement this.

Tweak /data so that any newly created files and directories will be readable and writable for members of the group wheel. Also ensure that the wbs group can create files and directories in /data/wbs.

Create a directory called /data/flintstones and make sure that members of the flintstones group have read, write and execute permissions to that directory and any newly created files and directories. Set it so that the user wilma does not have the write permission.

The user barney should always be able to write to files below /data