Configuring Shared Development Areas

Daniel Lawrence

AlphaZed Ltd
⟨`http://www.alphazed.co.uk`⟩

Although shared development areas are a Bad Idea in most cases, sometimes they are necessary. But unless a shared area is configured properly, users will have permissions problems and they won't be able to get their jobs done. This document explains how to set up a shared area under UNIX without a "777" in sight.

1. Group Membership

In UNIX, each user is a member of one or more groups. The intent is that members of a common group can work together on a project. The UNIX permissions system has been designed with this in mind.

The modern trend is to assign users to their own groups and then to add them to any additional groups as needed. Although this is a good idea from a security perspective, it does have implications when development areas need to be shared.

Group membership is specified in both the system password file and in `/etc/group`. BSD and System V versions of UNIX treat group membership slightly differently. More on this later.

The `/etc/group` file consists of a series of lines of the following format:

```
name:password:gid:members
```

name
: Group name.

password
: Group password. Never used. Place an asterisk in this field.

gid
: Numeric group id. This must be unique on the local system. If NIS is being used, this must be unique across all systems within the NIS domain.

members
: Group members. Comma-separated list of usernames.

2. File Creation Semantics

BSD and System V have different file creation semantics with regard to group membership. For shared directories, the BSD method works best.

2.1. System V

Under System V derived UNIX systems (Solaris, Linux), there is the concept of "primary group membership" and "secondary group membership". A user's primary group is the group specified in the gid field in the password file. If a user is a member of any additional groups, these groups are listed in `/etc/group`.

By default, when a file is created, it is owned by the user creating the file, and is assigned to that user's primary group.

Here is an example. The directory `example` has the following permissions:

```
$ ls -ld example
drwxrwxr-x   2 daniell  webgroup     106 Sep 15 10:35 example
```

Here *root* creates a file in the `example` directory. *root*'s primary group is `other`. Notice the new file's group membership:

```
# cd example
# touch file
# ls -l
total 0
-rw-rw-r--   1 root     other          0 Sep 15 10:50 file
```

An alternative way of assigning a new file's group is to use the group of the parent directory rather than the user's primary group. This can be configured by setting the *setgid* bit in the parent directory:

```
# cd ..
# chmod g+s example
# ls -ld example
drwxrwsr-x   2 root     webgroup      69 Sep 15 10:58 example
```

Now when a file is created in that directory, it inherits the group of the parent directory:

```
# cd example
# touch file2
# ls -l
total 0
-rw-rw-r--   1 root     other          0 Sep 15 10:50 file
-rw-rw-r--   1 root     webgroup       0 Sep 15 11:00 file2
```

Note: Directories created under a *setgid* parent inherit the *setgid* bit also.

2.2. BSD

In BSD, there is less distinction between primary and secondary group membership, and new files always take on the group of the parent directory.

2.3. NFS mounts

When a Solaris host NFS mounts a filesystem from a BSD server, you should specify the `grpid` mount option. This will cause file creation to obey the BSD semantics without requiring the *setgid* bits to be set.

3. How to Set Up a Shared Area

Considering the above file creation sematics and the way in which UNIX file permissions work, we can now see how a shared area should be set up.

Note: In the examples below, the `-exec` option to `find(1)` is used. Normally we would not use this option. We would instead pipe the output of `find(1)` into `xargs(1)`, which is more efficient. However, with the prevalence of filenames containing space characters, using `-exec` may help avoid shell quoting issues.

3.1. Create Group for Shared Area

There should be a generic group for the shared area. Do not use an individual user's group. You can either create a group specifically for this area, or you can use some generic group for several shared areas. Project-specific groups are better for security, but have slightly more administrative overhead. Somebody has to maintain group memberships in `/etc/group`.

3.2. Add Users to Group

You should make sure that all developers in the project are members of the project's group. This is done by adding their usernames to the group's entry in `/etc/group`.

3.3. Set project directory permissions

File and directories in the shared area should now be set. Most files and directories in the shared area should be part of the project group, and permissions should be set so that developers can read and write within the shared area. The following instructions assume all developers can read and write everything in the shared area. If there are any exceptions, these can be dealt with individually.

3.3.1. Set Group Membership

Make sure all the files and directories in the shared area are part of the project group.

```
# cd project-directory
# find . -exec chgrp project-group '{}' \;
```

3.3.2. Set Directory Permissions

Make sure all the directories in the shared area allow project users to create and delete files.

```
# find . -type d -exec chmod g+rwx '{}' \;
```

An additional step must be taken on Solaris hosts to set the *setgid* bit:

```
# find . -type d -exec chmod g+s '{}' \;
```

3.3.3. Set File Permissions

Make sure all the files in the shared area can be read and written by project users. Do this by making sure the group read/write bits are set.

```
# find . -type f -exec chmod g+rw '{}' ;
```

4. Set Up User Environments

Once the project directory ownerships and permissions have been set, the rest of the development environment must maintain these settings. The most important setting is a user's *umask*. This should almost always be `002` for users of shared areas. The following sections show how to set the umask in common applications.

4.1. Shells

The shell umask can be set using the internal `umask` command:

```
$ umask 002
```

This has the effect of changing the umask in the current shell session only. To set a specific umask whenever you log in, the following files may have to be changed. There may be other files, depending on your shell.

- `/etc/login.conf` (BSD)
- `/etc/profile` or `/etc/cshrc`
- `~/.profile` or `~/.cshrc`

To display the current umask setting from the shell, just type `umask`.

```
$ umask
0002
```

4.2. FTP

You can change the umask for WU-FTP sessions using the `defumask` directive in `ftpaccess`:

```
        defumask 0002
```

The FreeBSD FTP daemon lets you set the file creation umask using the `-u` *`umask`* command line option.

ProFTPD has a `Umask` directive in `proftpd.conf` which allows you to set both the file creation umask and directory creation umask:

```
        Umask 0002 0002
```

4.3. Samba

The following directives in Samba's `smb.conf` file will cause file permissions to be set correctly when users access shares with Windows machines:

```
        create mask = 774
        directory mode = 775
```

Note: The above create mask of 774 will probably cause files to be created with the owner execute bit set. This is because this bit is mapped to the DOS/Windows *archive* bit. This behaviour is controlled by the `map archive` directive, which is set by default to `yes`. The book "Using Samba" (O'Reilly) says that some DOS/Windows applications don't work properly if this bit isn't stored correctly. So we go ahead and leave it.

4.4. Netatalk

I don't know of a way to set the umask for Mac sessions. Mac users may have to resort to FTP when using a shared area if their file permissions don't get set right.