## Thursday June 23, 2016

**Linux Manual Pages Search**

**Articles**

- Five Minutes to a More Secure SSH
- Five Minutes to an Even More Secure SSH
- Using cURL for FTP Over SSL File Transfers
- Copying Only Directories With Rsync
- Troubleshooting SSH Connections
- Use ACK as a Better Grep
- Four Tips for Moving Around the Filesystem Quickly With the Bash Shell
- Keeping Tabs on Cron
- Learning Network Security With Linux Iptables Firewall Scripts
- Linux Process Termination Made Easy
- Monitoring and Alerting on Patterns in Linux Logfiles
- Quick and Easy Way to Monitor Process Memory Usage
- Quick Logfile Processing With Perl
- Quickly Upgrading and Reconfiguring a Source-based PHP Installation
- Running Remote Commands on Multiple Servers With SSH and Dsh
- Save Time on the Command Line With the Bash Shell
- Speeding Up Bulk File Operations
- The Forgotten Power of Unix Text Utilities
- Top 30 GUI-Replacing Linux Console Apps
- Using Install to Copy Files and Change Ownership and Permissions



# Speeding Up Bulk File Operations

It's a common sysadmin task to want to change permissions on all the files and subdirectories under a top-level directory. You could just use the '-R' switch to **chmod**, but what if your files and directories need different permissions? One scenario that comes up is with shared directories - you have a directory tree that has to be writable by users in a specific group. To do this you want to set the group ID bit on all the directories, so that files created by an individual user are always writable by the entire group (this is numeric permission mode 2775). We want regular files to just have permissions 664. Let's work through a few solutions, when we're done we'll time them to see which is faster.

## Find

We first need a way to differentiate files and directories - one easy way is with the **find** command, which as a bonus will also recurse into subdirectories for us. Here's our first crack at a solution - let's assume we have changed to the top-level directory we are interested in already:

```
find . -type f -exec chmod 664 {} \;
find . -type d -exec chmod 2775 {} \;
```

A word of warning - **don't** try something like 'find . -type f | chmod 664 *' - chmod will ignore its standard input and change the permissions on all the files in the current directory. This is easily fixable by just re-running chmod, but it would be a disaster if you were trying to delete only certain files or directories. Anyway, in the command above, the '-type f' and '-type d' output just files and just directories, respectively. The '-exec' will execute the given command on each file or directory produced by find. The special construct '{}' is a placeholder for the current argument, as output by find. These commands will work, but they are very slow on large directory trees, since the chmod is operating on one file or directory at a time. We could try to improve

```
chmod 664 $(find . -type f)
chmod 2775 $(find . -type d)
```

## Xargs

These last two commands will work fine until we have more than a few dozen files or directories in total - if we do, we'll get the error '/bin/chmod: Argument list too long'. That's a cue that we should be using **xargs**, a very useful command that will submit its input in manageable chunks to the specified command. Here is our next try:

```
find . -type f | xargs chmod 664
find . -type d | xargs chmod 2775
```

This is better - the errors about the command line being too long will go away, and this will work, most of the time. But what happens if we have directories or filenames with spaces, quotes or other special characters in them? This comes up quite a bit when you have transferred files from Windows filesystems - the end result will be that xargs will mangle its input, and the command will fail with an error like *xargs: unmatched single quote; by default quotes are special to xargs unless you use the -0 option*. The error leads us in the right direction, the solution is to use a couple of options to find and xargs that go together: **-print0** and **-0**.

**-print0**: Find option that prints the full filename to standard output, terminated by a null character instead of a

newline.

**-0**: Xargs option that says input is terminated by a null character, rather than a newline, and all characters are treated literally (even quotes or backslashes).

Here is our final attempt with find and xargs:

```
find . -type f -print0 | xargs -0 chmod 664
find . -type d -print0 | xargs -0 chmod 2775
```

This will work for us all the time, no matter what special characters comprise file or directory names.

## Perl

There are some versions of find that don't support the '-print0' switch. On these systems, you may be able to use a **Perl** solution:

```
perl -MFile::Find -e 'find sub { -f && chmod 0664, $_; \
                    -d && chmod 02775, $_ },"."'
```

The **find** procedure exported by the the **File::Find** module takes two arguments - a callback subroutine and a list of directories. It will recursively descend into the list of supplied directories (in this case just the current directory '.'), and run the callback subroutine on each file or directory found. The subroutine in this case is an anonymous one, given by 'sub { -f && chmod 0664, $_; -d && chmod 02775, $_ }'. It first tests whether the current argument is a regular file, if it is it performs the required 'chmod 664'. It then tests whether the current argument is a directory, and as you might expect, performs the required 'chmod 2775'. The variable '$_' represents the current argument, in this case whatever the current file or directory name is. Note also that the numeric permissions must always have a leading zero so that the Perl interpreter knows they are octal numbers.

This solution has the advantage of working on any Unix system that has Perl installed, since File::Find is a core Perl module.

## Which is Faster?

So how fast does each solution run? Here are the timings on a directory tree with 9105 files and 370 directories:

```
time find . -type f -exec chmod 664 {} \;

real    0m15.687s
user    0m5.676s
sys     0m9.877s

time find . -type f -print0 | xargs -0 chmod 664

real    0m0.132s
user    0m0.036s
sys     0m0.080s

time perl -MFile::Find -e 'find sub { -f && chmod 0664, $_; },"."'

real    0m0.151s
user    0m0.080s
sys     0m0.056s

time perl -MFile::Find -e 'find sub { -f && chmod 0664, $_; \
-d && chmod 02775, $_ },"."'

real    0m0.160s
user    0m0.064s
sys     0m0.076s
```

As you can see, the xargs solution is more than two orders of magnitude faster then find/exec. The Perl solution was surprisingly fast, very much comparable to the xargs solution. When you consider that the last Perl solution timed tests for both files and directories at once, it is faster than running two xargs commands in a row.