

# Finding Things in Unix

Dru Lavigne

One of the most useful utilities to be found on any Unix system is the find command.

In the next two articles, I'd like to work you through the syntax of this command and provide you with some practical examples of its usage.

The command itself has a very simple syntax:

**find where\_to\_search expressions**

The expressions are the part that can look confusing the first few times you use find. They are also the part that can vary from Unix system to Unix system, so you may want to take a quick peek at find's manpage if you find yourself on a new system. The most common expressions that you can use on your FreeBSD system are as follows:

-name	must be quoted when using wildcards
-type	e.g. f=file d=directory l=link
-user	name or UID
-group	name or GID
-perm	specify permissions
-size	rounded up to next 512 byte block or use c to specify bytes
-atime	last time file was read
-ctime	last time file's owner or permissions were changed
-mtime	last time file was modified
-newer	find files newer than given file
-delete	remove files found
-ls	gives same output as ls -dgils
-print	displays results of find command
-exec command {} \;	to execute a command; note the required syntax
-ok	use instead of exec to be prompted before command is executed
-depth	starts at lowest level in directory tree rather than root of given directory
-prune	used when you want to exclude certain subdirectories

I'll be giving examples that show how to use and combine these expressions. Before doing that, what can find be used for? If you use the whatis command to see what it does, the answer may surprise you:

```
whatis find
find(1)          - walk a file hierarchy
```

In a nutshell, find is meant to recursively search directories to find any files that meet your specified expressions. This may not seem like such a big deal, but there aren't that many Unix utilities that can "walk" through a directory and all of its subdirectories. This ability proves quite useful, as not only can you find files, but you can do something with them as you find them.

# Finding Things in Unix

Dru Lavigne

Let's start with some simple examples and then work our way towards some more seemingly-complicated expressions. The simplest find you can do is to simply type this:

```
find . -print
```

Since the "." represents your current directory, this find command will find all of the files in your current directory and all of its subdirectories, and then print the results to your screen.

On your FreeBSD system, -print is assumed if you forget to type it, so this command will also give the same result:

```
find .
```

However, it's a good idea to get in the habit of using -print, in case you ever need to do a find on a system that does not assume this action.

To find all of the files in your home directory, you should first make sure you are in your home directory, then repeat that find command, like so:

```
cd  
find . -print
```

The cd command will always take you to your home directory. Since the find command can be used to do powerful things, it is always a good idea to first cd to the directory structure in which you wish to work. For the rest of this article, I will assume that you are in your home directory, so you won't inadvertently affect any files on your FreeBSD system that don't reside in your home directory.

The above examples demonstrated how easy it is to use find, but usually you are looking for something specific when you invoke the find command. This is where the other expressions come into play. Let's try to find a file with a specific name:

```
touch file1  
find . -name file1 -print  
./file1
```

Let's pick apart what I just did for a moment. I created an empty file named file1 using the touch command. I then told find to search my current directory (".") for a file named (-name) file1 and to print the results of the search to my screen. Also, I can tell that I only have one file named file1 in my home directory and all of its subdirectories, as only one result was displayed.

Often, when you need to use the find command, you are looking for more than one file. For example, you may want to find all of the files with a certain extension. I tend to download a lot of .pdf files and don't always remember to keep them in the same place. Occasionally, I like to collect them and put them in a directory I've created named pdfs. When this urge strikes, I can use the following find command to search my home directory and its subdirectories for all .pdf files:

```
find . -name "*.pdf" -print  
./pdfs/50130201a.pdf  
./pdfs/50130201b.pdf  
./pdfs/50130201c.pdf  
./pdfs/IWARLab.pdf  
./pdfs/DoS_trends.pdf
```

# Finding Things in Unix

Dru Lavigne

```
./pdfs/Firewall-Guide.pdf  
./2000_ports.pdf
```

It looks like I've been pretty good lately, as I only have one .pdf file that is not in my pdfs directory.

You'll note that in order to get this find command to work, I had to "quote" the "\*" wildcard by typing "\*.pdf" instead of just \*.pdf. There are two other ways to quote, so the following two commands will yield the same results:

```
find . -name \*.pdf -print  
find . -name '*.pdf' -print
```

Let's add to that original command and see how the output changes. What if I was only interested in seeing which .pdf files were not in the pdfs directory? Let's repeat that find command, but pipe the results to grep so only that one file will be displayed:

```
find . -name "*.pdf" -print | grep -v "^\.pdfs/" ./2000_ports.pdf
```

Well, that command worked, but that syntax looks pretty scary; we better pick it apart. Whenever you use grep -v, you are creating a reverse filter, meaning that you want it to show the opposite of what follows. In my case, I'm not interested in the files that reside in the ./pdfs/ directory; I want to find the files that aren't, so I used the reverse filter. You'll note that I also "quoted" the whole expression. I also added a bit extra, that ^\ stuff. The ^ tells grep to only search the very beginning of a line for my expression. The \ is an extra quote so that grep does not interpret the . as a special character. The whole thing put together told grep to just show me the files that don't live in the ./pdfs/ directory, so I received the desired output.

Brave enough to try something even more useful but complicated-looking? Let's get find to not only find this file, but move it to the correct directory using the following command:

```
find . -name "*.pdf" -print | grep -v "^\.pdfs/" | xargs -J X mv X ./pdfs/
```

To see if it worked, let's repeat the original find command:

```
find . -name "*.pdf" -print  
./pdfs/50130201a.pdf  
./pdfs/50130201b.pdf  
./pdfs/50130201c.pdf  
./pdfs/IWARLab.pdf  
./pdfs/DoS_trends.pdf  
./pdfs/Firewall-Guide.pdf  
./pdfs/2000_ports.pdf
```

So it worked. Let's see why. Once grep finished filtering the find output, we piped that result to the xargs command to finish the job for us. The J switch tells xargs to take all of the files it receives and assume that the file you specify with the command is to be the destination. For example, before I ran the find command, I had no idea how many files needed to be moved. There may have been one, or there may have been several. I needed to let xargs know that regardless of how many files were found, I want them all moved and I want them all moved to the pdfs directory. That bit of magic is the job of the J switch. To get the J switch to work properly, I also defined a character (X) and put that character on either side of the mv command.

## Finding Things in Unix

Dru Lavigne

Remember that Unix filenames don't necessarily have extensions, so you may want to search for a more complicated pattern. Let's say I want to find any files that have "bsd" somewhere in their name. I would do this command:

```
find . -name "*bsd*" -print
./.kde/share/icons/favicons/www.freebsd.org.png
./.kde/share/icons/favicons/www.freebsdjournal.org.png
./.kde/share/wallpapers/bsdbg1280x1024.jpg
./mnmclient-1.11/contrib/freebsd
```

We can also find a file by more than just its name. For example, to find all the files that you have not read in more than (+) 30 days:

```
find . -atime +30 -print
```

To see files you haven't modified, use -mtime instead, and to see files you haven't changed the owner or permissions of, use -ctime. The number after the + indicates how many days or 24-hour periods. To see which files were modified today, try:

```
find . -mtime -1 -print
```

This will show what files were modified during the last 24 hours. Note that this time you should use the -, as you want to find the files from less than one day ago.

The other switch that deals with time is the -newer switch. The three time switches all use 24-hour periods. If you would like to be a bit more granular in your time than that, the -newer switch will compare a file's access, modification, and change times to within a minute. For example, to see if any of your "dot" files were changed since you last changed your .cshrc file, you could execute this command:

```
find . -type f -name ".*" -newer .cshrc -print
```

You'll note that I've included some other new switches in this command. I specified a "type" of -f for files, as I don't want to see any directories, just files. I told the -name switch that I was interested in seeing files that start with a ".". Finally, I used the -newer switch to indicate that I was interested in the files that were modified since I last modified my .cshrc file.

Since I've started to combine switches that indicate which files I'm interested in finding, I should mention that all switches are logically "anded" unless you use the -o or logical "or". Since the switches are logically anded, I really told the find utility that I was only interested in files that were of a certain type and had a certain name and were newer than my .cshrc file.

Let's look at an example that shows the difference between a logical "and" and a logical "or". If I wanted to see all of the files in my home directory that had not been accessed in the last seven days "and" are larger than 10 Mb, I would use this command:

```
find . -atime +7 -size +20480 -print
```

However, if I wanted to see any files that either had not been accessed in the last seven days "or" that were over 10 MB in size, I would use this command instead:

```
find . -atime +7 -o -size +20480 -print
```

## Finding Things in Unix

Dru Lavigne

You'll note that I had to do some math to come up with the number to give to the `-size` expression, since `-size` is looking for the number of 512-byte blocks. However, I could have used the `expr` command to do the math for me, like so:

```
find . -atime +7 -o -size +`expr 10 \* 1024 \* 2` -print
```

Note that in this example, everything between the backquotes (the ``` on the far left of your keyboard) is what will do the required math. We still need the `+` in front of the first back quote, as we want to see files greater than 10 MB. You could also test what the results of the math will be by adding the `echo` command to the beginning of the command:

```
echo find . -atime +7 -o -size +`expr 10 \* 1024 \* 2` -print find . -atime +7 -size +20480 -print
```

It is a good idea to echo complex commands first, to ensure that the stuff you've quoted will do what you expect before asking the `find` command to execute it.

That should get you started for this week. In next week's article, I'll continue through the rest of the expressions and give some more practical examples for using the `find` command.

Let's continue with this example:

```
find . -atime +7 -o -size +`expr 10 \* 1024 \* 2` -print
```

As a recap, this command was looking for any files in the current directory and its subdirectories (represented by `.`) that have not been accessed for more than 7 days (`-atime +7`) or (`-o`) that were greater than a certain size (`-size +`). I used the `expr` command to calculate the size for me. Since I was aiming for 10MB and `find` thinks in terms of 512 bytes, I needed to calculate 10 times 1024 times 2 (as 2 times 512 is 1024).

Notice that I used the ``` or "backquote" (the key on the far left of your PC keyboard). In Unix, whenever you want the output of one command passed to another command, put the command that will give the output between backquotes; this is known as command substitution. By putting the math that I wanted calculated between backquotes, the resulting calculation was passed to the `-size` switch and used by the `find` command.

The last thing I want you to notice is that I also had to quote the two `*` in the command using the `\` character. When calculating math, `*` represents multiply; however, to the shell it represents a wildcard. By placing a `\` before the `*`, the shell won't interpret it as a wildcard, so `expr` receives the `*` and will know that I want it to perform a multiplication.

Let's try some more examples. Let's say I have a large directory structure and I wish to search for a certain pattern and remove all of the files that match this pattern. There are several ways to do this with the `find` command, so let's compare some of these methods.

In my home directory, I have a directory called `tmp` that contains a subdirectory named `tst`. This `tst` directory has a lot of files and subdirectories, and some of these files end with a `.old` extension. Let's start by seeing just how many files live in my `tst` directory:

```
cd ~/tmp/tst
find . -print | wc -l
```

# Finding Things in Unix

Dru Lavigne

269

Notice that when the find command ran, it printed each file found on a separate line. I could then pipe that result to the word count (wc) command using the switch that counted the lines (-l). This told me that I have 269 files (including directories, since to Unix, directories are really files) in my tst directory. Let's see how many of these files have a .old extension:

```
find . -name "*.old" -print | wc -l
67
```

Now, how can I go about removing these \*.old files? One way is to use the -exec switch and have it call the rm command like so:

```
find . -name "*.old" -exec rm {} \;
```

Once that is finished, I can repeat this command to see if there are any remaining \*.old files:

```
find . -name "*.old" -print | wc -l
0
```

This command works, but it may not always be the best way to remove a large number of files. Whenever you use the -exec switch, a separate process is created for every file that find finds. This may not be an issue if you are only finding a small amount of files on your home computer. It may be an issue if you are finding hundreds or thousands of files on a production system. Regardless, this method does consume more resources and is slower than other methods.

Let's look at a second way to delete these files, this time using xargs:

```
find . -name "*.old" -print | xargs rm
```

You'll note that I didn't have to include the \; string at the end of this command, as that string is used to terminate commands that are passed to exec. By using xargs in this command, I will still remove all of the files that end in .old, but instead of creating a separate process for each file that is found, only one process is started through xargs. As find finds each file, it creates a list with each file on its own line. This list is passed to xargs, which takes all of the lines of the file and places them onto one line with a space to separate each file; it then passes this argument list of files to the rm command.

There is actually a third way to remove these files, using the -delete switch with find:

```
find . -name "*.old" -delete
```

This command has the easiest syntax to use and is actually the most efficient way of removing files. The -delete switch doesn't even need to open a separate process: all of the files are removed by the find process. Also, this command should always work, whereas the xargs command may fail if find finds more files that can be passed to a command as an argument list. If you are searching a deep directory structure or have very long filenames, you may reach this limit. If you are curious as to the actual limit, there is a sysctl value that has been set for you:

```
sysctl -a | grep kern.argmax
kern.argmax: 65536
```

The 65536 represents the maximum number of bytes (or characters) in an argument list.

# Finding Things in Unix

Dru Lavigne

Before moving on to some other switches, I should mention that you may want to verify which files find will find before removing them. In my examples, I was just removing old files in one of my test directories. If you are concerned that find may find some files you don't want deleted, run your command like this first:

```
find . -name "*.old" -print
```

This will give you a list of all the matching files. If the list looks good, use the -delete switch to remove the files as in the example mentioned above.

Or, you can do the above in just one find command by using -ok like so:

```
find . -name "*.old" -ok rm {} \;
```

The -ok will prompt for verification before executing the command that follows it. You'll note that I do have to use the rm command; I can't use the -delete switch. And, as with using -exec, I have to use the {} \; syntax in order for -ok to work.

Let's take a look at some more switches. The -ls switch will give the inode number, number of blocks, permissions, number of hard links, owner, group, size in bytes, last modification time, and name for each file that is found. For example, the following command will show me the first ten directories in my home directory; you'll note that I specified that I only wanted to see directories by using the -type d switch.

```
cd
find . -type d -ls | head
976142  8 drwxr-xr-x  39 genesis  wheel    4096 Mar  3 17:52 .
1413099  2 drwxr-xr-x   2 genesis  wheel    512 Mar  3 13:38 ./pdfs
373539  2 drwxr-xr-x   2 genesis  wheel    512 Feb  6 12:38 ./tst
1087249  2 drwxr-xr-x   2 genesis  wheel    512 Oct  4 07:29 ./perlscripts
650764  2 drwx-----   2 genesis  wheel    512 Mar  3 17:52 ./mail
706616  2 drwx-----   4 genesis  wheel    512 Sep 22 14:29 ../kde
706635  2 drwx-----  11 genesis  wheel    512 Nov  7 12:36 ../kde/share
706636  4 drwx-----   3 genesis  wheel   1536 Mar  2 18:38 ../kde/share/config
785986  2 drwx-----   2 genesis  wheel    512 Sep 22
14:36 ../kde/share/config/colors
706682  2 drwx-----   3 genesis  wheel    512 Mar  2 18:36 ../kde/share/fonts
```

Let's get a little fancier with the -ls switch. Earlier in the article, we piped some output to wc -l to see how many files contained a certain expression. Let's be a bit more particular and see how many subdirectories are in my home directory:

```
find . -type d -print | wc -l
256
```

Actually, there are only 255 subdirectories, as one of them is my current directory. Now, let's get a better idea of how this directory structure is laid out using this command:

```
find . -type d -ls | awk '{print $4 - 2, $NF}' | sort -rn | head
37 .
26 ../kde/share/apps/kio_http/cache
18 ../kde/share/apps
15 ../gimp-1.2
```

# Finding Things in Unix

Dru Lavigne

```
9 ./tmp/tst
9 ../kde/share
8 ./tmp/tst/h
8 ./tmp/tst/g
8 ./tmp/tst/f
8 ./tmp/tst/e
```

Wow, that's pretty cool. It looks like there are 37 subdirectories in my home directory (.), 26 subdirectories in the .kde/share/apps/kio\_http/cache subdirectory, etc. Now, let's see how this find command worked. I started by using the `-ls` switch, which gave a fair bit of information regarding each directory as it was found. This information was piped to the `awk` utility, which is used to extract the data from certain fields. You'll note that in the original `-ls` output, the results were in certain fields: inode number, number of blocks, permissions, number of links, etc. I told `awk` to take the information from column 4 (which contains the number of links and is `$4` to `awk`) and subtract 2 from that value (as I'm not interested in the directories `.` or `..`). I also wanted to know the name of each directory; since this was the very last column, I used `$NF` to represent that field. By placing these instructions within the curly braces `{}`, I told `awk` to do this to every file that it received from the `find` command. I then piped the results from `awk` to the `sort` command; by using `-rn`, I told `sort` to sort the numerical output from largest to smallest so I could see which directories had the most subdirectories. I didn't want to bore you with all the output, so I also piped the final results to the `head` command so it would only display the first ten.

Another `find` switch is the `-perm` switch. An example is to search for any files that have their permissions set to `777`, that is, set to read, write, and execute for everyone. This can be easily done with this command:

```
find . -perm 777 -print
```

The above command searches for files with the exact permissions of `777`. If you are concerned with only a certain bit, rather than all the permission bits, you can do something like this:

```
find . -perm -4000 -print
```

This example will only yield files that have the SUID bit set. (Read more about permissions in a previous article.) Another handy `find` command is this one:

```
find . -perm -0002 -print
```

It will find all files that are writable by others. Note that you can use `-0002`, `-002`, `-02`, or `-2` and receive the same result as leading 0s are assumed.

The last two switches I want to cover in today's article are useful when backing up or replicating directory structures. Let's start with `-depth`. Let's say I want to back up my entire home directory to a mounted directory named `/backup`. I can do this:

```
find . -depth -print | cpio -dump /backup
```

This command may also work without the `-depth` switch, but not always. By default, `find` lists the files it finds by starting at the point mentioned in the `find` command, in my case `.` or my home directory. That is, it lists first the directory, and then the contents of that directory. If it encounters a directory that has read-only permissions, `find` will still provide a list of the contents of that directory to the `cpio` command, but the `cpio` command won't have permission to replicate the files in that subdirectory. It is interesting

## Finding Things in Unix

Dru Lavigne

to note that `cpio` will still be able to create the directory, but as it does, it will set the permissions to read only, so it won't be able to create any files below that directory.

However, if you remember to use `-depth`, `find` will instead start its search at the lowest level, meaning it will list the contents of directories before it lists the directories themselves. This means that the files will already have been replicated by `cpio` before it sets the read-only permissions on the parent directory.

What if I don't want to replicate my entire home directory, just portions of it? This is where the `-prune` switch comes into play. Let's say I want to back up everything in my home directory except my `tmp` directory. I could do this:

```
find . -type d -name tmp -prune -o -print | cpio -dump /backup
```

You'll note that the syntax seems a little bit backwards. I used the `-name` switch to find any directories (`-type d`) named `tmp` and pass that list to `-prune`. I then used the logical or `-o` so that everything else will be printed and piped to `cpio`.

I hope the examples provided in the last two articles have helped you to become more comfortable with the `find` command and its syntax.