Finding files and searching for text

FINDERS KEEPERS

With Linux, you can keep track of your files using a variety of tools; we examine some of the most useful utilities. We also show you how to search for text patterns in files using grep.

BY DMITRI POPOV AND JOE CASAD

When it comes to finding and identifying files on your system, you are spoiled for choice. Linux offers a variety of tools that can help you locate files and programs, including *find*, *locate*, *whereis*, and *which*. These tools are not particularly difficult in use, and mastering them can help you use your Linux system more efficiently.

Finding Files with find

The *find* tool lets you search for files by name or a part of the name. By default, *find* searches recursively, meaning it looks for files through the entire directory tree. At the very minimum, *find* requires two options: a path to the directory where the search should start and the name of the file to look for. The name of the file is specified with the *-name* switch. For example, the following command will search for files whose names start with *Lin* in the *foo* directory and its subdirectories:

```
find /home/foo -name "Lin*"
```

As shown in this example, you can use wildcards in the search string to broaden the search. Because the *find* command is case sensitive, the previous command line initiates a search for all file names that start with *Lin*, but not those that begin with *lin*. However, you can instruct *find* to ignore case with the use of the *-iname* switch:

```
find /home/user -iname "Lin*"
```

The *find* command lets you specify multiple starting directories. The following command will search through the */usr*, */home*, and */tmp* directories to look for all *.bin* files:

```
find /usr /home /tmp -name "*.bin"
```

If you don't have the appropriate permissions to search in the system directories, *find* will display error messages. To avoid cluttering up the search results, you can send all error messages to the *null* file (i.e., discard them):

```
find /usr /home /tmp ➥
   -name "*.bin" 2>/dev/null
```

The *find* tool also supports the AND, OR, and NOT Boolean operators, which let you construct complex search strings. For example, you can use the *-size* parameter to limit the search to files that are larger than the specified limit:

```
find /photos ➥
   -iname "*.NEF" -and -size +7M
```

The command line above searches for *.NEF* files (Nikon raw files) that are larger than 7MB. In a similar manner, you can use the *!* (NOT) operator to find files that are larger than 7MB but are not *.NEF* photos:

```
find /downloads -size +7M ! ➥
   -iname "*.NEF"
```

The OR operator also can come in handy when you need to find files that match either of the specified criteria:

```
find /downloads -size +7M ➥
   -or -iname "*.NEF"
```

Instead of searching for files by name, you can use *find* to search for files by owner. For example, if you want to find all files owned by root, you can use the following command:

```
find . -user root
```

In a similar manner, you can use *find* to search for files owned by a specific group:

```
find . -group www
```

The *-type* option is useful for specifying the type of object to search for, such as *f* (regular file), *d* (directory), *l* (symbolic link), and a few others. Do you want to find the directory of photos from Berlin? Here is the command for that:

```
find berlin/ -type d
```

The *find* tool also offers several options that can be used to find files by time, including *-mmin* (last modified time in minutes), *-amin* (last accessed time in minutes), *-mtime* (last modified time in hours), and *-atime* (last accessed time in hours). So, if you want to find photos that were modified 10 minutes ago, you can use the following command:

```
find /photos -mmin -10 -name "*.NEF"
```

The *-exec* option is another rather useful option that allows you to execute a command on every search. For example, the following command searches for **.NEF* files in the *photos* directory and renames the found file with the *exiv2* tool:

```
find /photos -iname "*NEF" ➥
   -exec exiv2 mv ➥
   -r "%Y%m%d-%H%M%S" ➥
   *.NEF {} \;
```

Photo by Nadjib BR on Unsplash

Note the {} \; at the end of the command. The {} symbol is a placeholder for the name of the file that has been found, whereas \; indicates the end of the command. Instead of *-exec*, you can also use the *-ok* option, which asks you for confirmation before the command is executed.

Finally, you can use the *-fprint* option

```
find /home/user -name "Lin*" ➥
  -fprint search_results.txt
```

to print the search results to a text file.

Searching for Files with locate and updatedb

Similar to *find*, the *locate* tool lets you find files by their names. But instead of searching the system in real time, *locate* searches the database of file names, which is updated daily. The key advantage of this approach is speed; finding files with *locate* is much faster than with *find*. The use of *locate* is easy: Just run the *locate* command with the name of the file you want to find:

```
locate backup.sh
```

To ignore the case, you can use the *-i* option:

```
locate -i backup
```

As with *find*, you can use wildcards in your searches:

```
locate "*.jpg"
```

If you want to see only a limited number of results, you can do so by using the *-n* option followed by the number of your choice:

```
locate "*.jpg" -n 5
```

As mentioned before, *locate* performs searches by querying the database of file names, which is automatically updated every day, so if you have just downloaded a batch of photos from your camera, the *locate* command won't see them until the database is updated.

Fortunately, you don't have to wait until the system updates the database; with the *updatedb* command, you can manually update the database at any time. Just execute the *updatedb* command as root to force the system to update the database.

whereis and which

If you need to find the path to an executable program, its sources, and man pages, the *whereis* tool can help. The following command, for example, returns paths to binary, source, and man pages for the Rawstudio application:

```
whereis rawstudio
```

Using the available options, you can limit your search to specific types. To search only binaries, you can use the *-b* option, or use *-m* to search for man pages and *-s* to search for source files.

Whereas the *whereis* tool lets you locate program files and man pages, *which* tells you which version of a command will run if you just type its name in the terminal. For example, the *which soffice* command returns the */usr/bin/soffice* path. This means that the *soffice* command runs the application in the */usr/bin* directory. If you want to find all the locations of the command, you can use the *-a* option:

```
which -a soffice
```

With just these few, simple commands, you can locate your files quickly and easily.

grep

The Bash command shell also has tools that will let you search for a text string inside of a file. The most popular command for finding a search string is *grep*.

In its most basic form, *grep* searches a file for text matching a specified pattern and outputs every line of the file that contains the string.

The syntax for the *grep* command is:

```
grep [options] pattern file_name(s)
```

You can specify the search pattern explicitly or use a regular expression. (See the article elsewhere in this issue on regular expressions.)

Several options help to refine the search (see Table 1 for some examples). For example, if you don't want to output all the lines that match the search string but only want to know the number of matching lines, use the *-c* option.

To specify more than one pattern, use the *-e* option once for each pattern:

```
grep -e pattern1 -e pattern2 ➥
filename.txt
```

Alternatively, you can use the *-f* option to specify a pattern file that can contain multiple patterns.

Although most modern text editors and word processors have built-in search features, *grep* is still very useful for searching across a group of several files or for expressing complex search patterns that would be cumbersome in a GUI tool. System administrators often use *grep* to hunt for errors, warnings, devices names, and other information in system logs. See the following articles on "Regular Expressions" and "Pipes and Redirection" for more *grep* examples. ■

Table 1: Examples of grep Options

Option	Description
-c	Prints only a number representing the number of lines matching the pattern
-e	Specifies an expression as a search pattern (you can specify multiple expressions in one command – use the *-e* option with each expression)
-E	Use extended regular expressions (ERE)
-f file_name	Take patterns from a pattern file
-i	Ignore case
-l	Prints a list of file names containing the search string
-o	Only prints matched parts of matching line
-v	Prints all the lines that do NOT match the search pattern
-w	Match a whole word
-A n	Prints the matched line and *n* lines after the matched line
-B n	Prints the matched line and *n* lines before the matched line
-C n	Prints the matched line with *n* lines before and *n* lines after