Insider Tips: Find and Locate

Lost and Found

Modern computers with their multiple Gigabyte hard disks store thousands of files. A lost file can cause a lot of work, and it can also pose a security risk. Fortunately, Linux has some versatile tools for finding those "lost files."

BY MARC ANDRÉ SELIG

If you are looking for files in Linux, the command line is your best option for quick and reliable results. GUIs such as KDE (Figure 1) typically lack comparable functionality, flexibility, and speed.

The most important command for file searching is *find*. Without any flags specified, the tool will just find all the files below the current directory. If you want to search another directory, simply specify the directory name as the first argument. For example, *find /home* will output a list of the files and directories below */home*.

But *find* really begins to shine when you start feeding it options. The *-name* pattern argument allows users to tell *find* to output files that match the pattern. For example, *find ~ -name '*firefox*'* finds all the files in a user's home directory that include the *firefox* string in their names.

It makes sense to enclose arguments in double, or at least single, quotes. This stops the shell from expanding parameters such as **firefox**.

Options, Tests, Actions

find recognizes three types of arguments: options, tests, and actions. Options influence the behavior of *find* itself, for example, by restricting the number of subdirectory levels to search, or restricting the search to a single partition. Tests allow the user to restrict the search to specific files, for example, files that are less than a week old, *find -mtime -8*; or files that are over a week old, *find -mtime +7*.

Actions influence the way the results are output. The following action *-exec command* would pass the names of the files that *find* discovers to the program *command*.

Actions also support logical operators, allowing admins to specify complex output behavior. This said, administrators are likely to use tests more than they use the other types of arguments. The test might restrict the command to a specific type, such as *-type f* (to find only files), *-type d* (to find only directories), or *-type l* (to find only symlinks).

Responsible admins search for files that belong to non-existent users or groups in the process of cleaning up. The following command will take care of that chore: *find / -nouser -o -nogroup*. The *-o* option is the logical OR operator, which links the two expressions, *-nouser* and *-nogroup*. Another trick that is extremely useful for admins is searching for files with a set UID or set GID bit using *find / -perm +6000 -ls*. The *-ls* appended to this command runs *ls -lisd* for each file.

Executing Commands

-exec allows you to use filenames as arguments and even to create miniature shell scripts. But watch out for the pitfalls: using this functionality as root in public directories (like */tmp* or the home directories) is not a good idea. There is a short delay between finding files that match the search parameter and running the appropriate command, and crafty attackers might be able to exploit the gap. In the past there have been numerous attacks on Unix installations that use *find* to clean up the */tmp* directory.

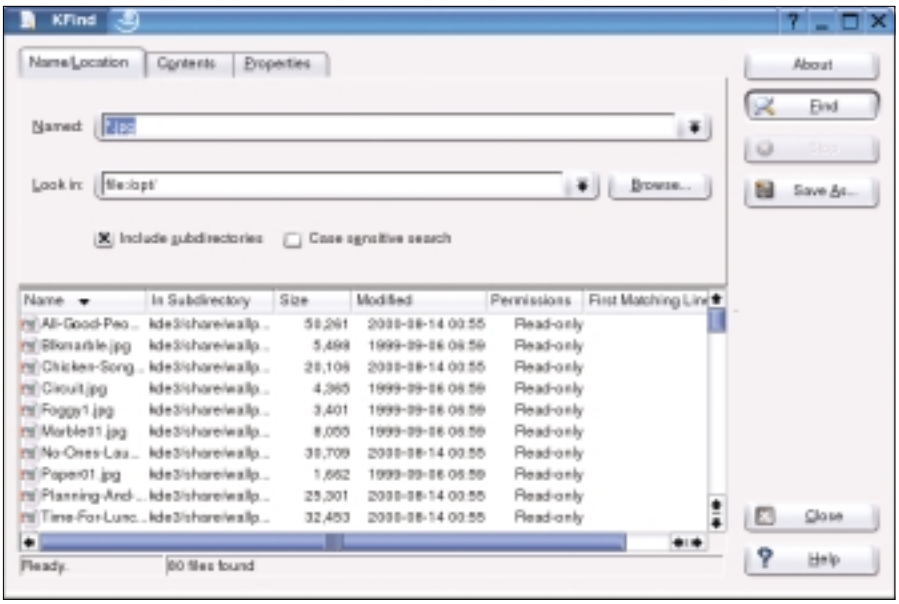

Figure 1: The KDE program KFind gives you an excellent front-end for file searching, but it cannot match the flexibility of *find*.

The *-exec* parameter is followed by a command including a *{}* placeholder. The program replaces the brackets with the name of the file that matches the search. A semicolon terminates the command. You need to escape the semicolon with a backslash to stop the shell from interpreting it. The following command is a variant on the DOS *ren *.txt *.bak* command, which renames files with the **.txt* extension to **.txt.bak*:

```
$ find . -name '*.txt' ➥
-exec mv {} {}.bak \;
```

If your requirements are more complex, you might prefer to use the output from *find* in a shell script. The *-printf* command can help you. The following command creates a file like the one shown in Listing 1:

```
$ find /home/mas -type f ➥
-name '*.txt' -printf "mkdir ➥
-p /export/backup/%h\ncp ➥
-p %p /export/backup/%h/➥
%f.copy\n\n"
```

Locate for Quicker File Finding

In addition to the features mentioned here, *find* has a whole range of options that we cannot look into. However, the program has one major drawback: it is slow. *find* parses the search directories one at a time, and it may also need to parse the file inodes to perform the tests you specify. If you only need to search a subsection of your home directory with a few hundred files, you will not notice a delay. But if you are looking for a file that is hidden somewhere on your system, *find* may need to search through thousands and thousands of inodes – understandably, this can take some time.

Locate solves this problem by creating an index of the files stored on a machine and storing their names in a central database. The tool does not need to read inodes to perform the search; instead it just searches the database file. This search typically returns results within a fraction of a second, removing the need to wait for minutes and also avoiding any possible effect on performance for users and services.

Locate has a few limitations, so in many cases, it makes sense to keep on using *find*. Instead of the complex command line syntax that *find* offers, locate only supports primitive searching for file name components (this is all the database stores, in fact). The locate tool supports wildcards, like the question mark for a single character and the asterisk for any number of characters. These locate wildcards can also represent the slash character (in contrast to shell wildcards):

```
$ locate /home/mas/*mail*
/home/mas/.fetchmailrc
/home/mas/.procmaildata-bulk
/home/mas/.procmaildata-inbox
/home/mas/.procmailrc
```

There are two trivial requirements for using locate, but they can be troublesome nonetheless. First of all, not all Linux distributions install this GNU utility. Secondly, the database is typically updated by cron [1]. If you switch off your computer at night, and if your distribution does not use Anacron [1] or a similar tool to catch up with cronjobs that it has missed, users will need to update the database manually using the *updatedb* command. ■

Listing 1: Find-generated script

```
01 mkdir -p /export/backup//home/mas
02 cp -p /home/mas/.emacs
   /export/backup//home/mas/.emacs.copy
03
04 mkdir -p /export/backup//home/mas
05 cp -p /home/mas/.fetchmailrc /export/
   backup//home/mas/.fetchmailrc.copy
```

Other People's Files

Locate uses a central database, which can lead to a security issue in some environments. If a user wants to hide a file, that user typically sets the permissions on the superordinate directory (*chmod go-rwx*). This makes the files invisible to commands like *find* and *ls*, except for the owner and root, of course.

However, if the file is in the locate database, because *updatedb* was launched with *root* privileges (via the crontab or by root), it becomes visible to everyone. Although other users only get to see the filename, even this could be secret in some circumstances.

Genuine multiuser systems can use the *--localuser* and *--netuser* parameters when calling *updatedb*. The tool expects a user name as an argument. The user name should be *nobody*, for example, a system user with very limited privileges. This means that the *locate* database will only have a minimum of visible files.

INFO

[1] Marc André Selig, "Admin Workshop: Cron, At, Anacron", Linux Magazine #49, p 60