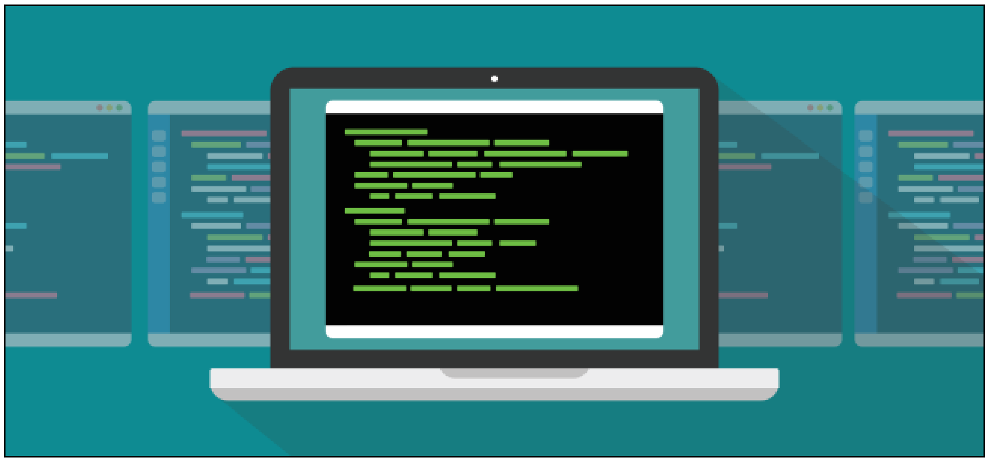
How to Use All Linux's Search Commands

DAVE MCKAY @thegurkha
JUNE 28, 2019, 9:00AM EDT



Linux offers six different ways to search, and each has its merits. We'll demonstrate how to use `find`, `locate`, `which`, `whereis`, `whatis`, and `apropos`. Each excels at different tasks; here's how to choose the right tool for the job.

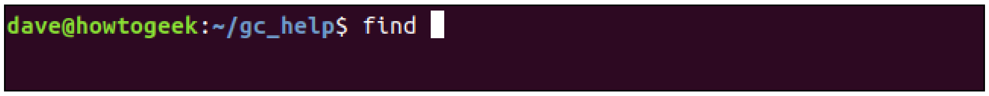
You're spoiled for choice when it comes to commands for searching and finding in Linux. Why so many? Well, they each have their specialties and perform better than the others in certain circumstances. You could think of them as a sort of Swiss-Army knife for searching. We're going to look at each blade in turn and find out its particular strengths.

The find Command

The behavior of the `find` command is difficult to determine by trial and error. Once you understand the syntax, you start to appreciate its flexibility and power.

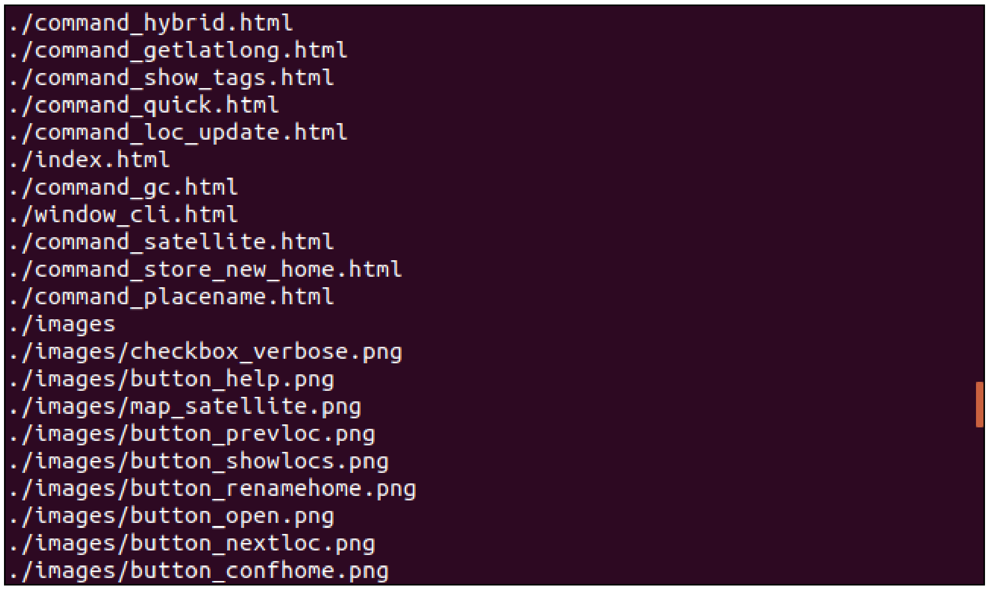
The simplest way to use `find` is to just type `find` and hit enter.

```
find
```



```
dave@howtogeek:~/gc_help$ find
```

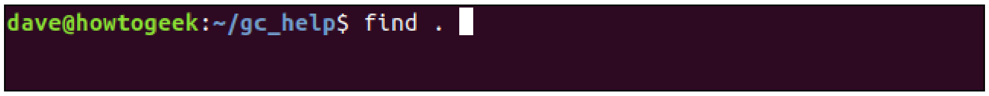
Used in this way `find` behaves like `ls`, but it lists all of the files in the current directory *and* those in subdirectories.



```
./command_hybrid.html
./command_getlatlong.html
./command_show_tags.html
./command_quick.html
./command_loc_update.html
./index.html
./command_gc.html
./window_cli.html
./command_satellite.html
./command_store_new_home.html
./command_placename.html
./images
./images/checkbox_verbose.png
./images/button_help.png
./images/map_satellite.png
./images/button_prevloc.png
./images/button_showlocs.png
./images/button_renamehome.png
./images/button_open.png
./images/button_nextloc.png
./images/button_confhome.png
```

Some implementations of `find` require you to put the `.` for the current directory. If this is the case with your version of Linux, use the following command:

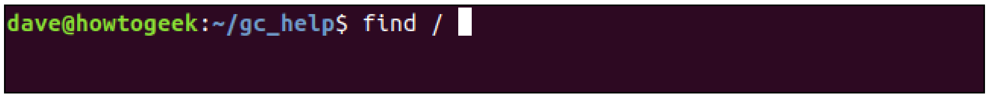
```
find .
```



```
dave@howtogeek:~/gc_help$ find .
```

To have `find` search from the root folder you'd use this command:

```
find /
```



```
dave@howtogeek:~/gc_help$ find /
```

To start the search from your home folder use this command:

```
find ~
```

```
dave@howtogeek:~/gc_help$ find ~
```

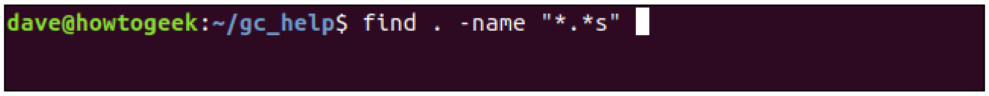
Using find With File Patterns

For `find` to be something more than an auto-recursing version of `ls`, we must provide it with something to search for. We can provide filenames or file patterns. Patterns make use of wildcards where `*` means any string of characters and `?` means any single character.

Patterns must be quoted to work correctly. It is easy to forget to do this, but if you don't quote the wildcard pattern `find` won't be able to properly carry out the command you gave it.

With this command, we're going to search in the current folder for files that match the pattern "*.*s". This means any filename that has a file extension that ends in "s". We use the `-name` option to tell `find` we're either passing in a filename or a filename pattern.

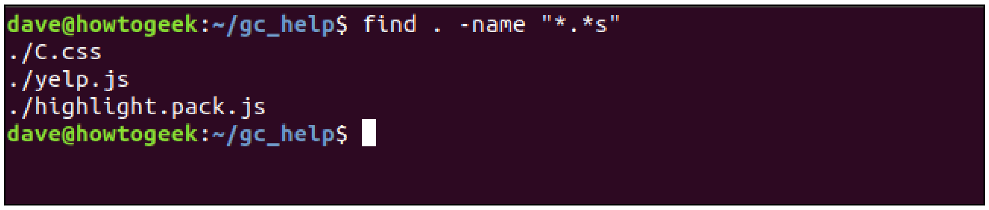
```
find . -name "*.*s"
```



```
dave@howtogeek:~/gc_help$ find . -name "*.*s"
```

`find` returns these matching files.

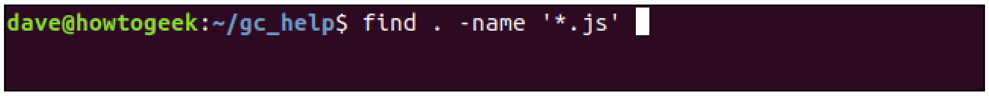
Note that two of the file extensions are two characters long and one is three characters long. This is because we used the pattern "*.*s". If we'd only wanted the two character file extensions, we would have used "*.?s".



```
dave@howtogeek:~/gc_help$ find . -name "*.*s"
./C.css
./yelp.js
./highlight.pack.js
dave@howtogeek:~/gc_help$ 
```

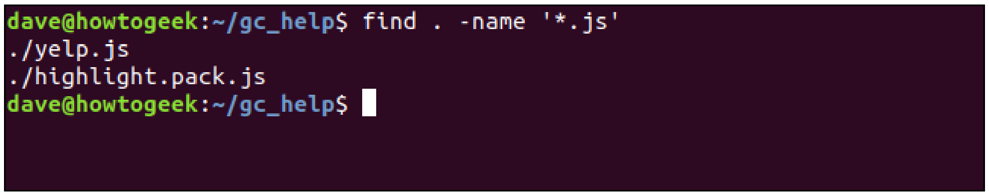
If we'd known in advance that we were looking for JavaScript ".js" files we could have been more specific in our file pattern. Also, note that you can use single quote marks to wrap the pattern if you prefer.

```
find . -name '*.js'
```



```
dave@howtogeek:~/gc_help$ find . -name '*.js' 
```

This time `find` only reports on the JavaScript files.

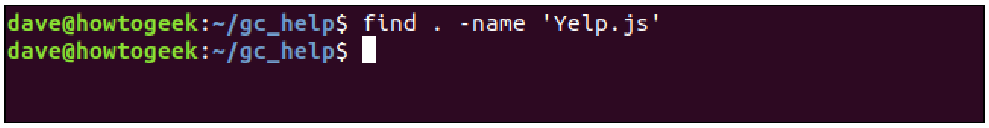


```
dave@howtogeek:~/gc_help$ find . -name '*.js'
./yelp.js
./highlight.pack.js
dave@howtogeek:~/gc_help$ 
```

Ignoring Case With find

If you know the name of the file you want `find` to locate, you can pass that to `find` instead of a pattern. You don't need to wrap the filename in quotes if there are no wildcards in it, but it is good practice to do it all the time. Doing so means you won't forget to use them when do you need them.

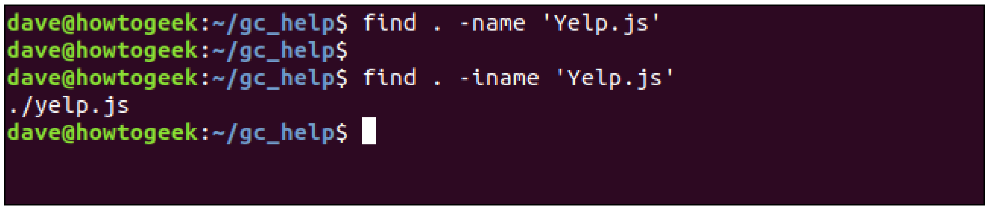
```
find . -name 'Yelp.js'
```



```
dave@howtogeek:~/gc_help$ find . -name 'Yelp.js'
dave@howtogeek:~/gc_help$ 
```

That didn't return anything. But's odd, we know that file must be there. Let's try again and tell `find` to ignore case. We do that by using the `-iname` option (ignore case name)

```
find. -iname 'Yelp.js'
```



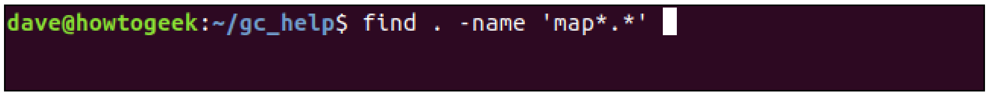
```
dave@howtogeek:~/gc_help$ find . -name 'Yelp.js'
dave@howtogeek:~/gc_help$
dave@howtogeek:~/gc_help$ find . -iname 'Yelp.js'
./yelp.js
dave@howtogeek:~/gc_help$
```

That was the problem, the filename starts with a lowercase "y", and we were searching with an uppercase "Y."

Recursing Subdirectories with find

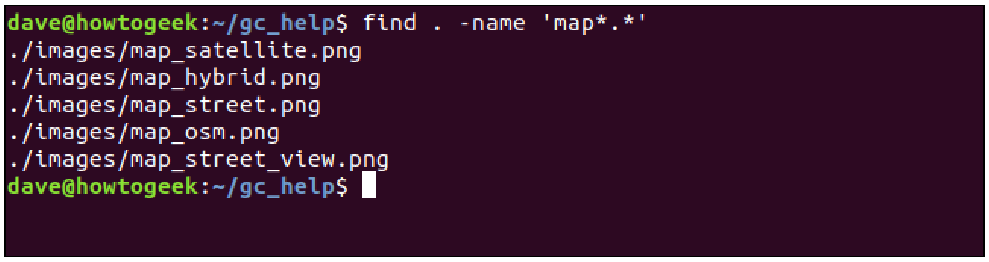
One great thing about `find` is the way it recursively searches through subdirectories. Let's search for any files that start with "map."

```
find . -name "map*.*"
```



```
dave@howtogeek:~/gc_help$ find . -name 'map*.*'
```

The matching files are listed. Note that they are all in a subdirectory.

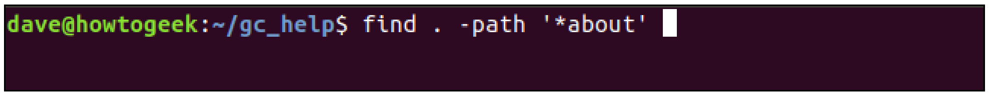


```
dave@howtogeek:~/gc_help$ find . -name 'map*.*'
./images/map_satellite.png
./images/map_hybrid.png
./images/map_street.png
./images/map_osm.png
./images/map_street_view.png
dave@howtogeek:~/gc_help$
```

Searching for Directories With find

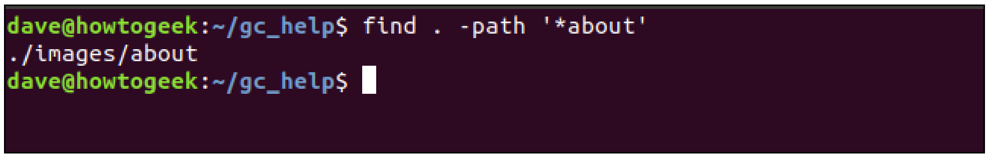
The `-path` option makes `find` look for directories. Let's look for a directory that we can't quite remember the name of, but we know it ends with the letters "about."

```
find . -path '*about'
```



```
dave@howtogeek:~/gc_help$ find . -path '*about'
```

The directory is found, it is just called “about,” and it is nested inside another directory within the current directory.



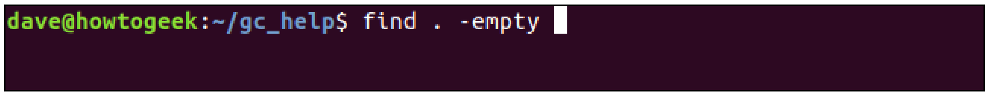
```
dave@howtogeek:~/gc_help$ find . -path '*about'
./images/about
dave@howtogeek:~/gc_help$
```

There is an `-ipath` (ignore case path) option that allows you to search for paths and to ignore case, similar to the `–iname` option discussed above.

Using File Attributes with find

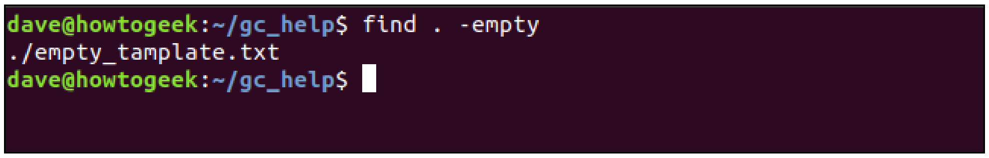
`find` can look for files that have attributes that match the search clue. For example, you can look for files that are empty using the `-empty` option, regardless of what they’re called.

```
find . -empty
```



```
dave@howtogeek:~/gc_help$ find . -empty
```

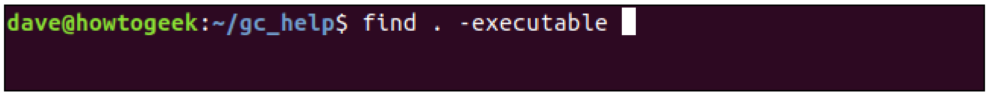
Any zero byte length files will be listed in the search results.



```
dave@howtogeek:~/gc_help$ find . -empty
./empty_tamplate.txt
dave@howtogeek:~/gc_help$
```

The `-executable` option will find any file that can be executed, such as a program or a script.

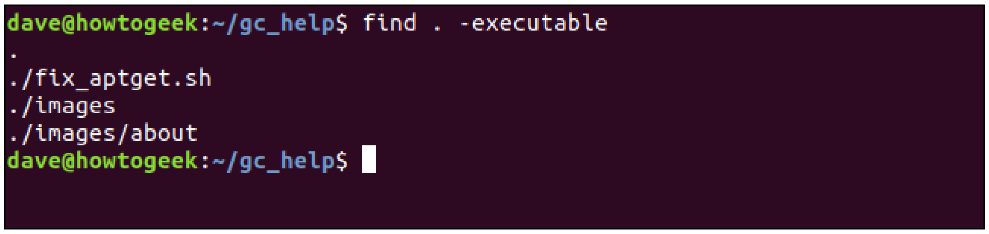
```
find . -executable
```



```
dave@howtogeek:~/gc_help$ find . -executable
```

The results list a file called “fix_aptget.sh”.

They also contain three directories, including ‘.’, the current directory. The directories are included in the results because the execute bit is set in their file permissions. Without this, you wouldn’t be able to change into (“run”) those directories.

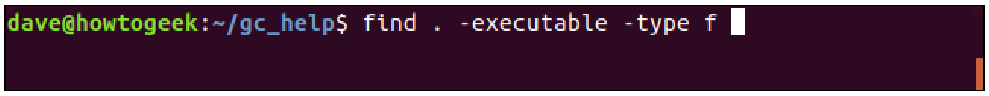


```
dave@howtogeek:~/gc_help$ find . -executable
.
./fix_aptget.sh
./images
./images/about
dave@howtogeek:~/gc_help$
```

The -type Option

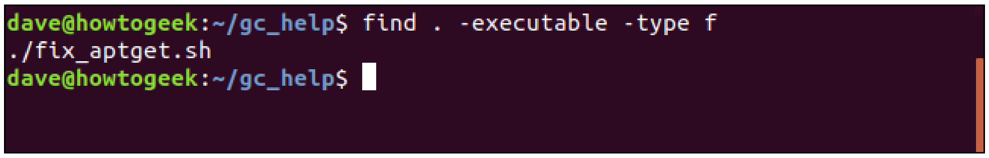
The `-type` option allows you to search for the type of object you are looking for. We’re going provide the type indicator “f” as a parameter to the `-type` option because we want `find` to search for files only.

```
find . executable -type f
```



```
dave@howtogeek:~/gc_help$ find . -executable -type f
```

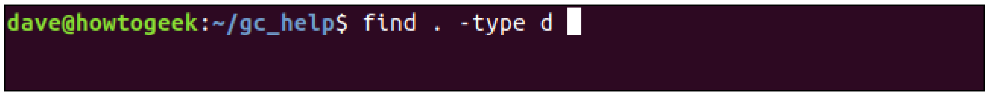
This time the subdirectories are not listed. The executable script file is the only item in the results.



```
dave@howtogeek:~/gc_help$ find . -executable -type f
./fix_aptget.sh
dave@howtogeek:~/gc_help$
```

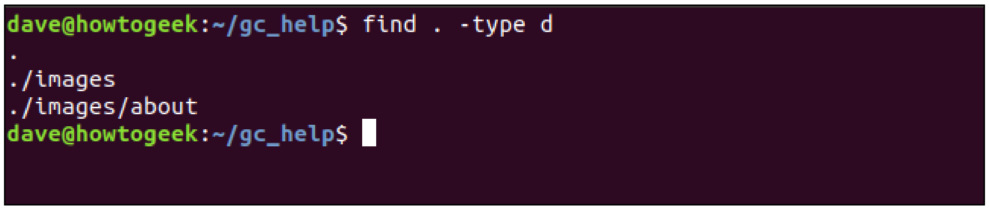
We can also ask `find` to only include directories in the results. To list all the directories, we can use the `-type` option with the type indicator "d".

```
find . type -d
```



```
dave@howtogeek:~/gc_help$ find . -type d
```

Only directories and subdirectories are listed in the results.



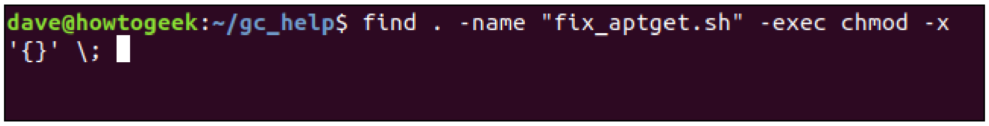
```
dave@howtogeek:~/gc_help$ find . -type d
.
./images
./images/about
dave@howtogeek:~/gc_help$
```

Using Other Commands With find

You can perform some additional action on the files that are found. You can have the files passed, in turn, to some other command.

If we need to make sure there are no executable files in the current directory and subdirectories, we could use the following command:

```
find . -name "fix_aptget.sh" -exec chmod -x '{}' \;
```



```
dave@howtogeek:~/gc_help$ find . -name "fix_aptget.sh" -exec chmod -x
'{}' \;
```

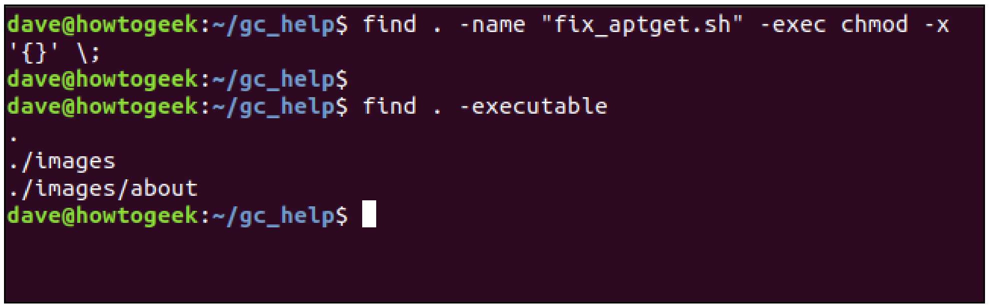
The command means:

- Search in the current directory for a named object called "fix_aptget.sh".
- If it is found execute the `chmod` command.
- The parameters that are passed to `chmod` are `-x` to remove executable permissions and `'{}'` which represents the filename

of the found file.

- The final semicolon marks the end of the parameters that are going to be passed to `chmod`. This has to be ‘escaped’ by preceding it with a ‘\’ backslash.

Once this command has been run, we can search for executable files as before, and this time there will be no files listed.



```
dave@howtogeek:~/gc_help$ find . -name "fix_aptget.sh" -exec chmod -x
'{}' \;
dave@howtogeek:~/gc_help$
dave@howtogeek:~/gc_help$ find . -executable
.
./images
./images/about
dave@howtogeek:~/gc_help$ 
```

To cast our net wider, we could use a file pattern instead of the filename we used in our example.

This flexibility allows you to search for specified file types, or with filename patterns, and have some action carried out on the matching files.

Find has many other options, including searching for files by their modified date, files owned by a user or group, files that are readable, or files that have a specific set of file permissions.

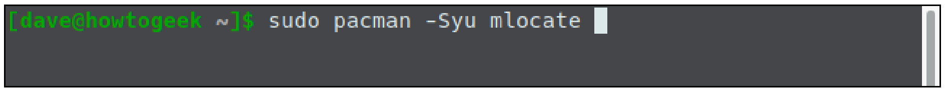
The locate And mlocate Commands

Many Linux distributions used to have a copy of `locate` included with them. This was superseded by the `mlocate` command, which was an improved and updated version of `locate`.

When `mlocate` is installed on a system it modifies the `locate` command so that you actually use `mlocate` even if you type `locate`.

Current versions of Ubuntu, Fedora, and Manjaro were checked to see whether they had versions of these commands pre-installed on them. Ubuntu and Fedora both included `mlocate.` It had to be installed on Manjaro, with this command:

```
sudo pacman -Syu mlocate
```



```
[dave@howtogeek ~]$ sudo pacman -Syu mlocate
```

On Ubuntu, you can use locate and `mlocate` interchangeably. On Fedora and Manjaro you must type `locate` , but the command is executed for you by `mlocate`.

If you use the `--version` option with `locate` you'll see that the command that responds is actually `mlocate`.

```
locate --version
```

Because `locate` works on all of the Linux distributions that were tested, we'll use `locate` in our explanations below. And it's one less letter to type.

The locate Database

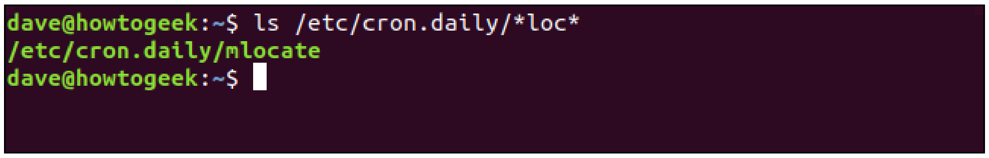
The biggest advantage that `locate` has is speed.

When you use the `find` command, it dashes off and performs a search across your filesystem. The `locate` command works very differently. It does a database lookup to determine whether what you are looking for is on your computer. That makes the search much faster.

Of course, it does raise an obvious question about the database. What ensures the database is up to date? When `mlocate` is installed it (usually) places an entry in `cron.daily`. This runs each day (very early in the morning) and updates the database.

To check whether this entry exists, use this command:

```
ls /etc/cron.daily/*loc*
```



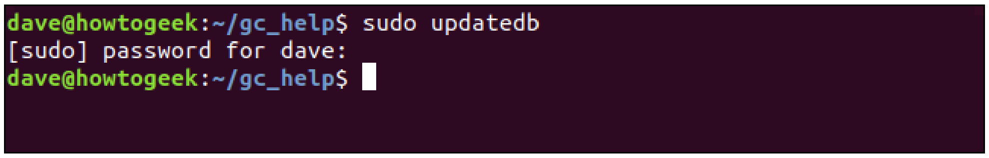
```
dave@howtogeek:~$ ls /etc/cron.daily/*loc*
/etc/cron.daily/mlocate
dave@howtogeek:~$
```

If you don’t find an entry there, you could set up an automated task to do this for you at the time you choose.

RELATED: ***How to Schedule Tasks on Linux: An Introduction to Crontab Files***

What if your computer isn’t on at the time when the database is supposed to be updated? You can manually run the database update process with the following command:

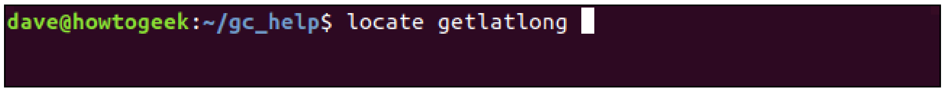
```
sudo updatedb
```


```
dave@howtogeek:~/gc_help$ sudo updatedb
[sudo] password for dave:
dave@howtogeek:~/gc_help$
```

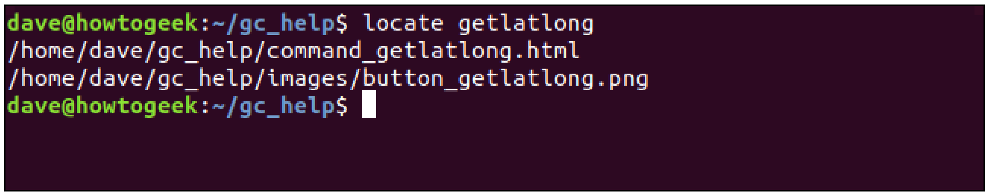
Using locate

Let’s look for files that contain the string “getlatlong”. With locate, the search automatically looks for any matches that contain the search term anywhere in the filename, so there is no need to use wildcards.

```
locate getlatlong
```


```
dave@howtogeek:~/gc_help$ locate getlatlong
```

It’s hard to convey speed in a screenshot, but almost immediately the matching files are listed for us.

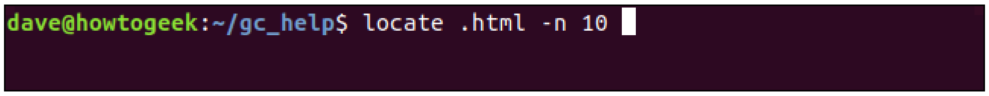

```
dave@howtogeek:~/gc_help$ locate getlatlong
/home/dave/gc_help/command_getlatlong.html
/home/dave/gc_help/images/button_getlatlong.png
dave@howtogeek:~/gc_help$
```

Telling locate How Many Results You Want

Sometimes you may know there are lots of files of the type your searching for. You only need to see the first few of them. Perhaps you just want to be reminded which directory they are in, and you don’t need to see all of the filenames.

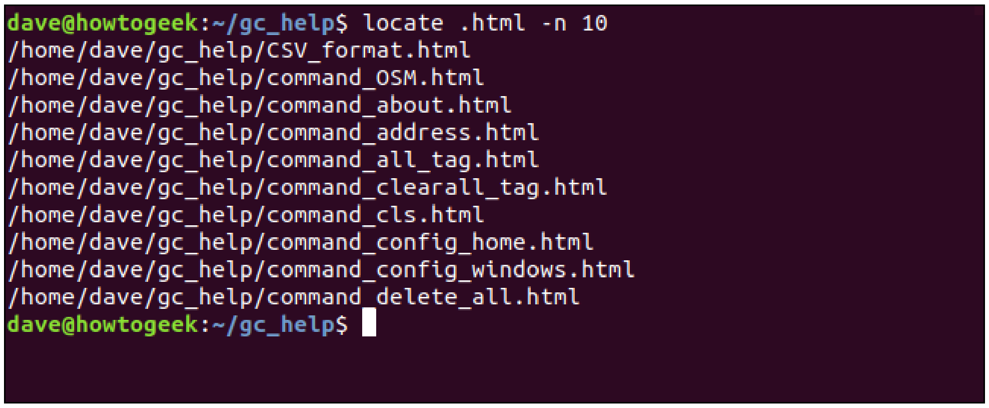
Using the `-n` (number) option you can limit the number of results that `locate` will return to you. In this command, we’ve set a limit of 10 results.

```
locate .html -n 10
```



```
dave@howtogeek:~/gc_help$ locate .html -n 10
```

`locate` responds by listing the first 10 matching file names it retrieves from the database.

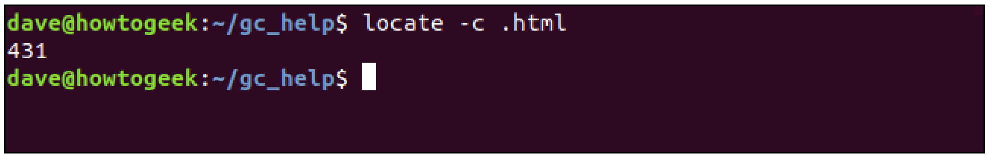


```
dave@howtogeek:~/gc_help$ locate .html -n 10
/home/dave/gc_help/CSV_format.html
/home/dave/gc_help/command_OSM.html
/home/dave/gc_help/command_about.html
/home/dave/gc_help/command_address.html
/home/dave/gc_help/command_all_tag.html
/home/dave/gc_help/command_clearall_tag.html
/home/dave/gc_help/command_cls.html
/home/dave/gc_help/command_config_home.html
/home/dave/gc_help/command_config_windows.html
/home/dave/gc_help/command_delete_all.html
dave@howtogeek:~/gc_help$
```

Counting Matching Files

If you only want to know the number of matching files and you don't need to know what they are called or where they are on your hard drive, use the -c (count) option.

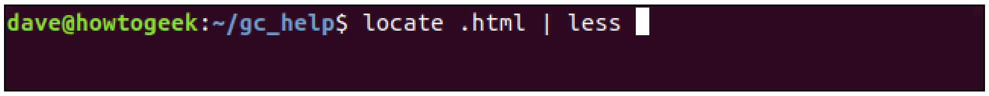
```
locate -c .html
```



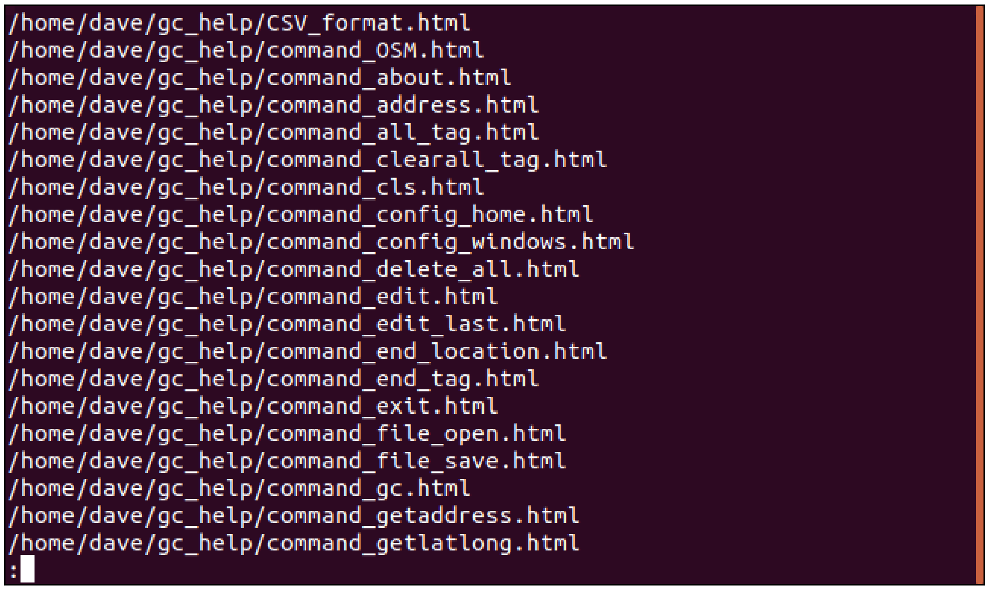
```
dave@howtogeek:~/gc_help$ locate -c .html
431
dave@howtogeek:~/gc_help$
```

So, now we know there are 431 files with the ".html" extension on this computer. Maybe we do want to have a look at them, but we thought we'd take a peek and see how many there were first. Armed with that knowledge we know we'll need to pipe the output through `less`.

```
locate .html | less
```


```
dave@howtogeek:~/gc_help$ locate .html | less
```

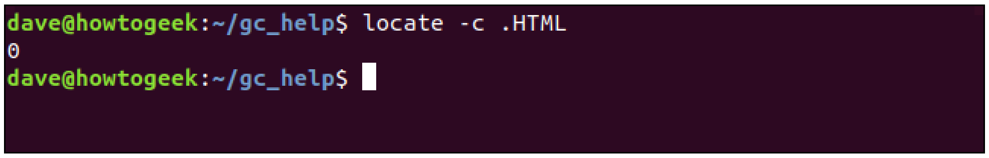
And here they all are, or at least, here's the top of the long list of them.


```
/home/dave/gc_help/CSV_format.html
/home/dave/gc_help/command_OSM.html
/home/dave/gc_help/command_about.html
/home/dave/gc_help/command_address.html
/home/dave/gc_help/command_all_tag.html
/home/dave/gc_help/command_clearall_tag.html
/home/dave/gc_help/command_cls.html
/home/dave/gc_help/command_config_home.html
/home/dave/gc_help/command_config_windows.html
/home/dave/gc_help/command_delete_all.html
/home/dave/gc_help/command_edit.html
/home/dave/gc_help/command_edit_last.html
/home/dave/gc_help/command_end_location.html
/home/dave/gc_help/command_end_tag.html
/home/dave/gc_help/command_exit.html
/home/dave/gc_help/command_file_open.html
/home/dave/gc_help/command_file_save.html
/home/dave/gc_help/command_gc.html
/home/dave/gc_help/command_getaddress.html
/home/dave/gc_help/command_getlatlong.html
:
```

Ignoring Case With locate

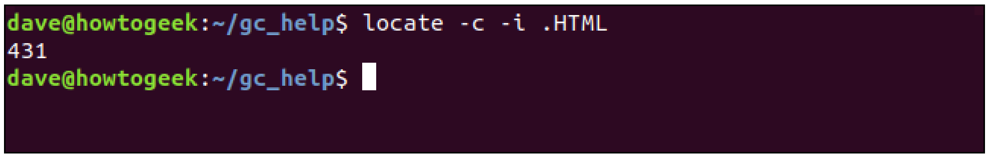
The `-i` (ignore case) causes `locate` to do just that, it ignores uppercase and lowercase differences between the search term and the filenames in the database. If we try and count the HTML files again, but mistakenly provide the search term in uppercase we'll get zero results.

```
locate -c .HTML
```


```
dave@howtogeek:~/gc_help$ locate -c .HTML
0
dave@howtogeek:~/gc_help$
```

By including the `-i` option we can make `locate` ignore the difference in case, and return our expected answer for this machine, which is 431.

```
locate -c -i .HTML
```

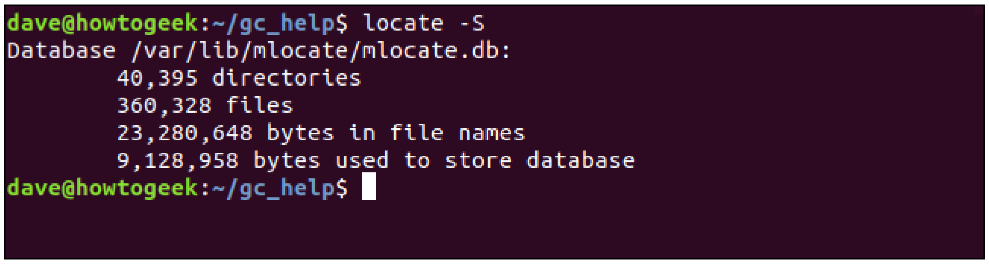


```
dave@howtogeek:~/gc_help$ locate -c -i .HTML
431
dave@howtogeek:~/gc_help$
```

The locate Database Status

To see the status of the database, use the `-s` (status) option. This causes `locate` to return some statistics about the size and contents of the database.

```
locate -s
```



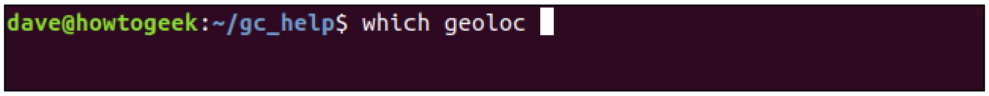
```
dave@howtogeek:~/gc_help$ locate -S
Database /var/lib/mlocate/mlocate.db:
        40,395 directories
        360,328 files
        23,280,648 bytes in file names
        9,128,958 bytes used to store database
dave@howtogeek:~/gc_help$
```

The which Command

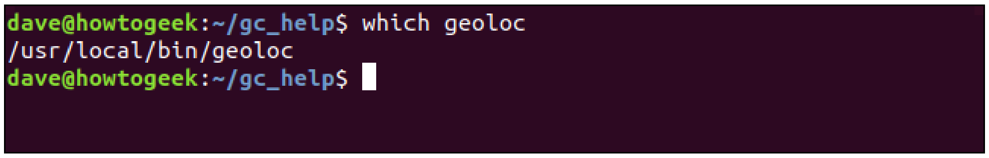
The `which` command searches through the directories in your path, and tries to locate the command you are searching for. It allows you to determine which version of a *program* or *command* will run when you type its name on the command line.

Imagine we had a program called `geoloc`. We know it is installed on the computer, but we don't know where it is located. It must be in the path somewhere because when we type its name, it runs. We can use `which` to locate it with this command:

```
which geoloc
```

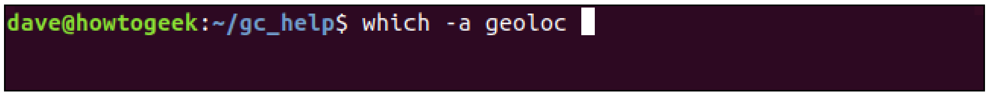

```
dave@howtogeek:~/gc_help$ which geoloc
```

`which` reports that the program is located in `/usr/local/bin`.

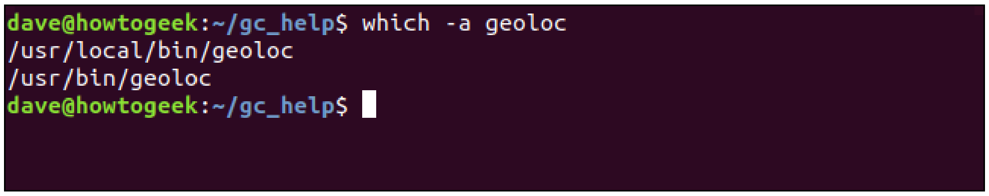

```
dave@howtogeek:~/gc_help$ which geoloc
/usr/local/bin/geoloc
dave@howtogeek:~/gc_help$
```

We can check whether there are any other copies of the program in other locations within the path by using the `-a` (all) option.

```
which -a geoloc
```


```
dave@howtogeek:~/gc_help$ which -a geoloc
```

This shows us that we have the `geoloc` program in two places.


```
dave@howtogeek:~/gc_help$ which -a geoloc
/usr/local/bin/geoloc
/usr/bin/geoloc
dave@howtogeek:~/gc_help$
```

Of course, the copy in `/usr/local/bin` is going to be found first by the Bash shell every time, so having the program in two places is meaningless.

Removing the version in `/usr/bin/geoloc` will save you a bit of hard drive capacity. More importantly, it will also avoid issues created by someone manually updating the program, and doing it in the wrong place. Then wondering why they don't see the new updates when they run the program.

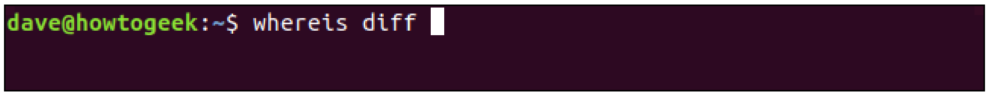
The whereis Command

The `whereis` command is similar to the `which` command, but it is more informative.

In addition to the location of the command or program file, `whereis` also reports where the man (manual) pages and source code files are located. In most cases, the source code files won't be on your computer, but if they are, `whereis` will report on them.

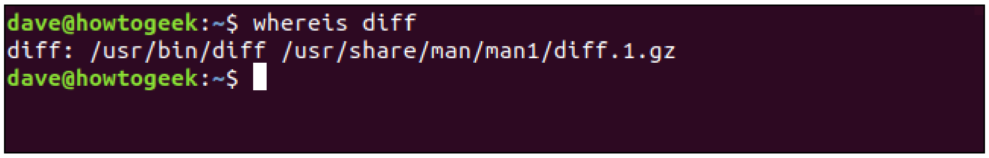
The binary executable, the man pages and the source code are often referred to as the "package" for that command. If you want to know where the various components of the package for the `diff` command are located, use the following command:

```
whereis diff
```



```
dave@howtogeek:~$ whereis diff
```

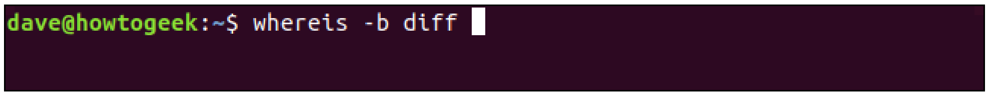
`whereis` responds by listing the location of the `diff` man pages and the `diff` binary file.



```
dave@howtogeek:~$ whereis diff
diff: /usr/bin/diff /usr/share/man/man1/diff.1.gz
dave@howtogeek:~$
```

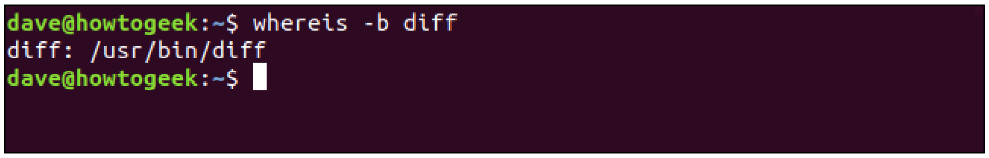
To restrict the results to only show the location of the binary (in effect, make `whereis` work like `which`) use the `-b` (binary) option.

```
whereis -b diff
```



```
dave@howtogeek:~$ whereis -b diff
```

`whereis` only reports on the location of the executable file.

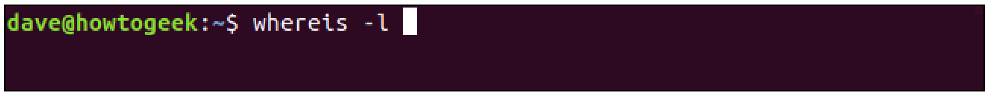


```
dave@howtogeek:~$ whereis -b diff
diff: /usr/bin/diff
dave@howtogeek:~$
```

To restrict the search to report only on the man pages use the `-m` (manual) option. To restrict the search to report only on the source code files use the `-s` (source) option.

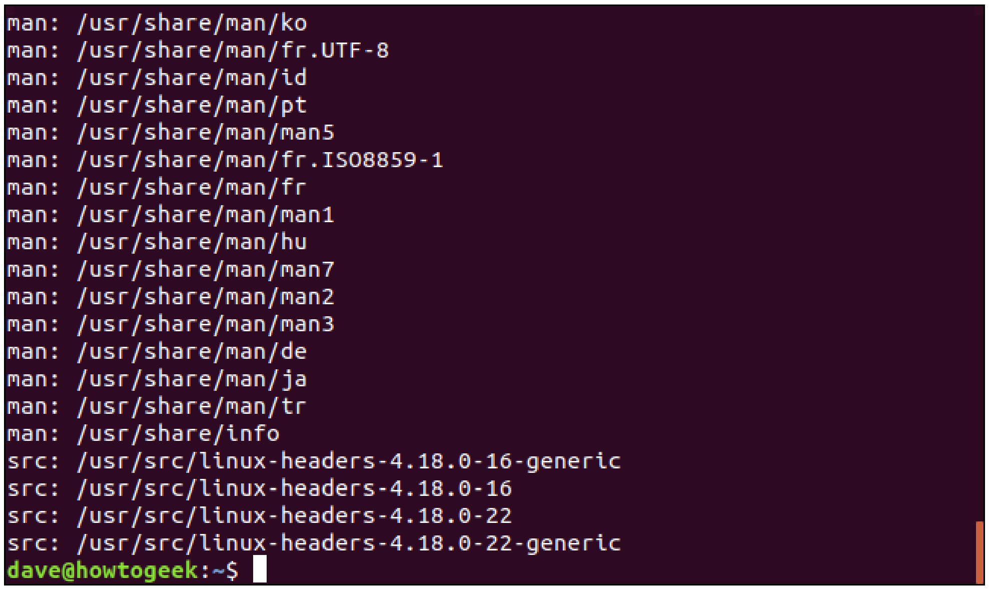
To see the locations that `whereis` searches through, use the `-l` (locations) option.

```
whereis -l
```



```
dave@howtogeek:~$ whereis -l
```

The locations are listed for you.

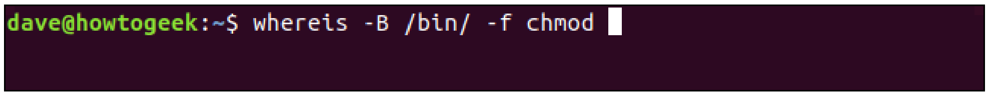


```
man: /usr/share/man/ko
man: /usr/share/man/fr.UTF-8
man: /usr/share/man/id
man: /usr/share/man/pt
man: /usr/share/man/man5
man: /usr/share/man/fr.ISO8859-1
man: /usr/share/man/fr
man: /usr/share/man/man1
man: /usr/share/man/hu
man: /usr/share/man/man7
man: /usr/share/man/man2
man: /usr/share/man/man3
man: /usr/share/man/de
man: /usr/share/man/ja
man: /usr/share/man/tr
man: /usr/share/info
src: /usr/src/linux-headers-4.18.0-16-generic
src: /usr/src/linux-headers-4.18.0-16
src: /usr/src/linux-headers-4.18.0-22
src: /usr/src/linux-headers-4.18.0-22-generic
dave@howtogeek:~$
```

Now that we know the locations `whereis` will search in, we can, should we choose, restrict the search to a particular location or group of locations.

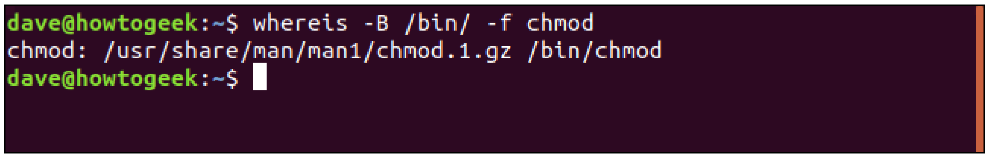
The `-B` (binary list) option restricts the search for executable files to the list of paths provided on the command line. You must provide at least one location for `whereis` to search through. The `-f` (file) option is used to signal the end of the location last the start of the filename.

```
whereis -B /bin/ -f chmod
```



```
dave@howtogeek:~$ whereis -B /bin/ -f chmod
```

whereis looks in the single place we asked to search through. That happens to be where the file is located.



```
dave@howtogeek:~$ whereis -B /bin/ -f chmod
chmod: /usr/share/man/man1/chmod.1.gz /bin/chmod
dave@howtogeek:~$
```

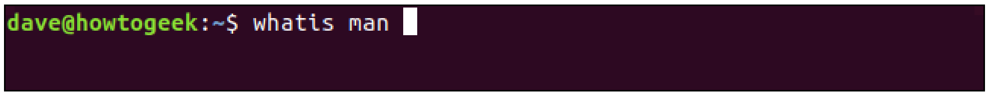
You can also use the -M (manual list) option to restrict searches for man pages to the paths you provide on the command line. The -S (source list) option allows you to restrict the search for source code files in the same way.

The whatis Command

The whatis command is used to quickly search through the man (manual) pages. It provides one-line summary descriptions of the term you've asked it to search for.

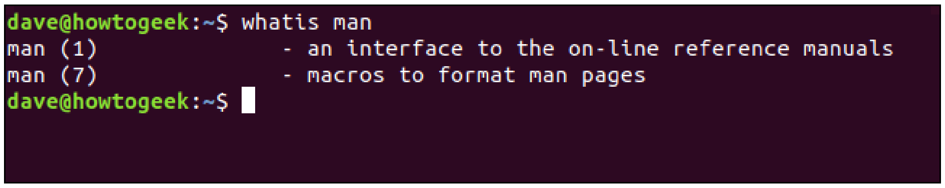
Let's start with a simple example. Although it looks like the starting point of deep philosophical debate, we're just asking whatis to tell us what the term "man" means.

```
whatis man
```



```
dave@howtogeek:~$ whatis man
```

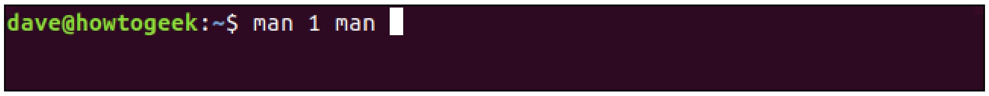
whatis finds two matching descriptions. It prints a short description for each match. It also lists the numbered section of the manual that contains each full description.



```
dave@howtogeek:~$ whatis man
man (1)              - an interface to the on-line reference manuals
man (7)              - macros to format man pages
dave@howtogeek:~$
```

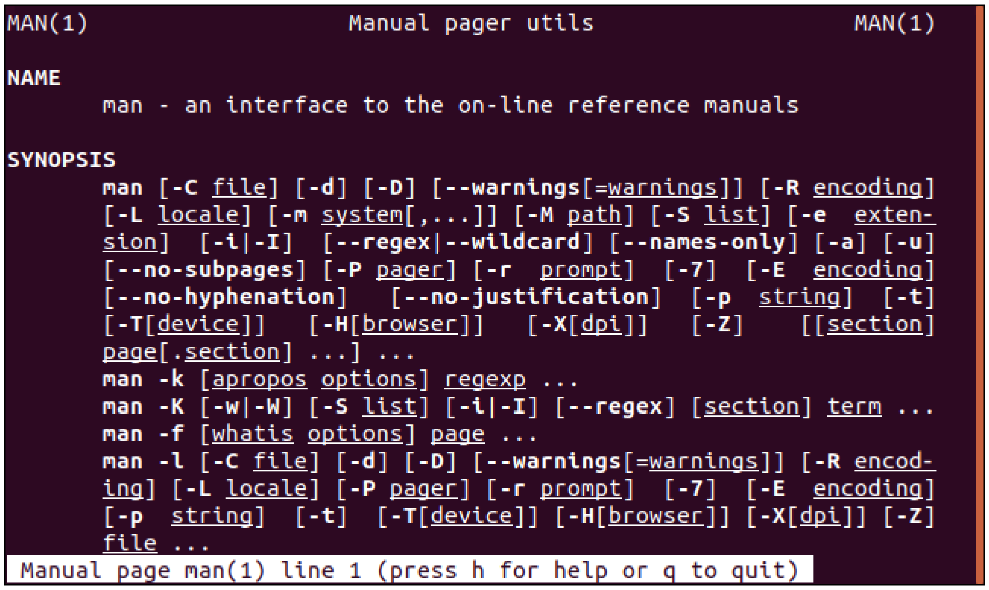
To open the manual at the section that describes the `man` command, use the following command:

```
man 1 man
```



```
dave@howtogeek:~$ man 1 man
```

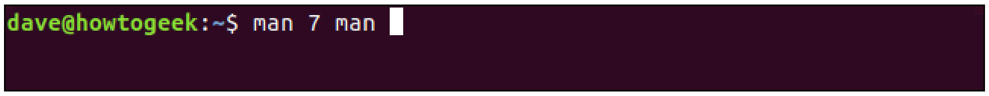
The manual opens at section man(1), at the page for `man`.



```
MAN(1)                    Manual pager utils                    MAN(1)

NAME
       man - an interface to the on-line reference manuals

SYNOPSIS
       man [-C file] [-d] [-D] [--warnings[=warnings]] [-R encoding]
       [-L locale] [-m system[,...]] [-M path] [-S list] [-e  exten-
       sion]  [-i|-I]  [--regex|--wildcard] [--names-only] [-a] [-u]
       [--no-subpages] [-P pager] [-r  prompt]  [-7]  [-E  encoding]
       [--no-hyphenation]  [--no-justification]  [-p  string]  [-t]
       [-T[device]]  [-H[browser]]  [-X[dpi]]  [-Z]    [[section]
       page[.section] ...] ...
       man -k [apropos options] regexp ...
       man -K [-w|-W] [-S list] [-i|-I] [--regex] [section] term ...
       man -f [whatis options] page ...
       man -l [-C file] [-d] [-D] [--warnings[=warnings]] [-R encod-
       ing] [-L locale] [-P pager] [-r prompt]  [-7]  [-E  encoding]
       [-p  string]  [-t]  [-T[device]] [-H[browser]] [-X[dpi]] [-Z]
       file ...
 Manual page man(1) line 1 (press h for help or q to quit)
```

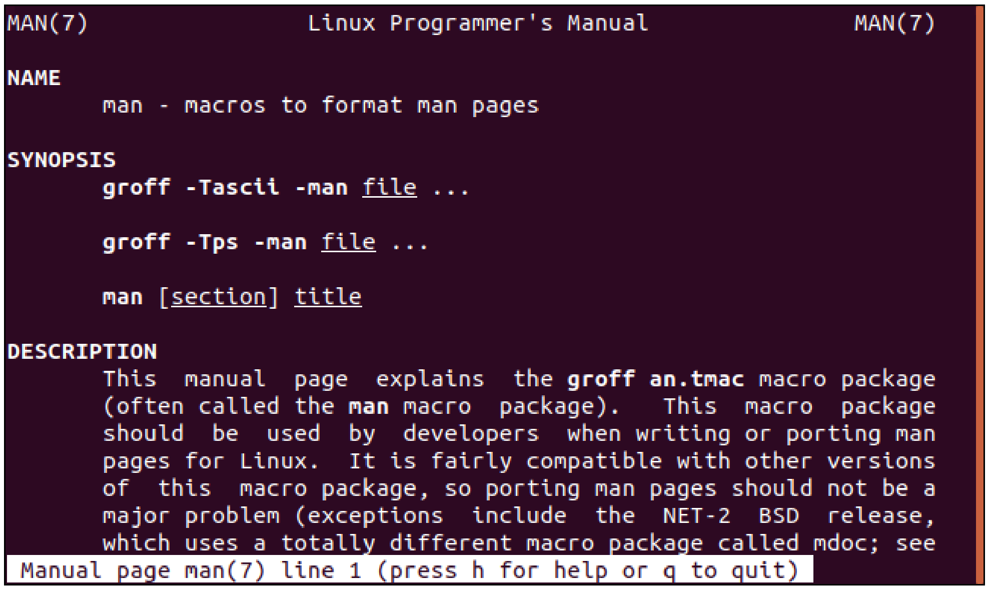
To open the manual at section 7, at the page that discusses the macros you can use to generate man pages, use this command:

```
man 7 man
```



```
dave@howtogeek:~$ man 7 man
```

The man page for the man macros is displayed for you.

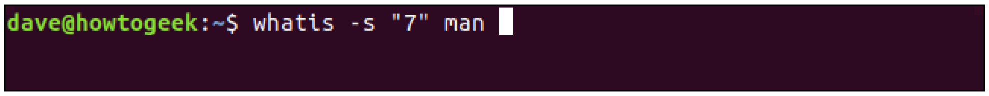


```
MAN(7)                  Linux Programmer's Manual                 MAN(7)

NAME
       man - macros to format man pages

SYNOPSIS
       groff -Tascii -man file ...

       groff -Tps -man file ...

       man [section] title

DESCRIPTION
       This  manual  page  explains  the groff an.tmac macro package
       (often called the man macro  package).   This  macro  package
       should  be  used  by  developers  when writing or porting man
       pages for Linux.  It is fairly compatible with other versions
       of  this  macro package, so porting man pages should not be a
       major problem (exceptions  include  the  NET-2  BSD  release,
       which uses a totally different macro package called mdoc; see
 Manual page man(7) line 1 (press h for help or q to quit)
```

Searching In Specific Sections of the Manual

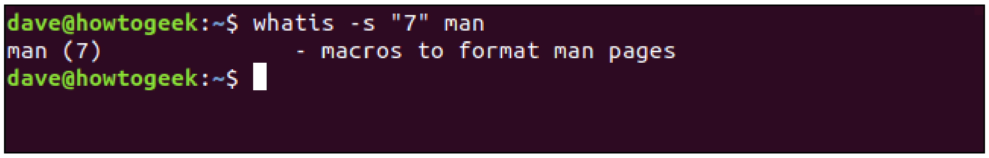
The `-s` (section) option is used to limit the search to sections of the manual you are interested in. To have the `whatis` search restricted to section 7 of the manual, use the following command. Note the quote marks around the section number:

```
whatis -s "7" man
```



```
dave@howtogeek:~$ whatis -s "7" man
```

The results only refer to section 7 of the manual.

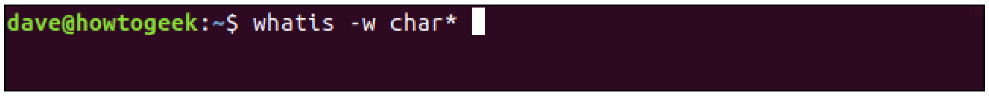


```
dave@howtogeek:~$ whatis -s "7" man
man (7)              - macros to format man pages
dave@howtogeek:~$
```

Using whatis With Wildcards

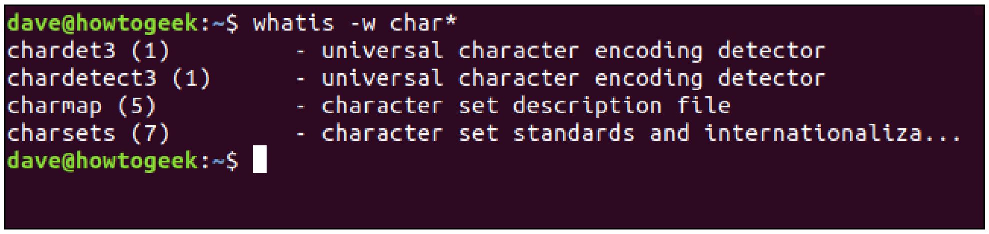
You can use wildcards with `whatis`. You must use the `-w` (wildcard) option to do so.

```
whatis -w char*
```



```
dave@howtogeek:~$ whatis -w char*
```

The matching results are listed in the terminal window.



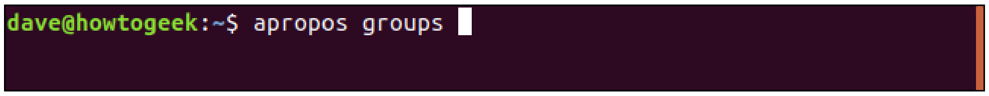
```
dave@howtogeek:~$ whatis -w char*
chardet3 (1)         - universal character encoding detector
chardetect3 (1)      - universal character encoding detector
charmap (5)          - character set description file
charsets (7)         - character set standards and internationaliza...
dave@howtogeek:~$
```

The apropos Command

The `apropos` command is similar to `whatis`, but it has a few more bells and whistles. It searches through the man page titles and one line descriptions looking for the search term. It lists the matching man page descriptions in the terminal window.

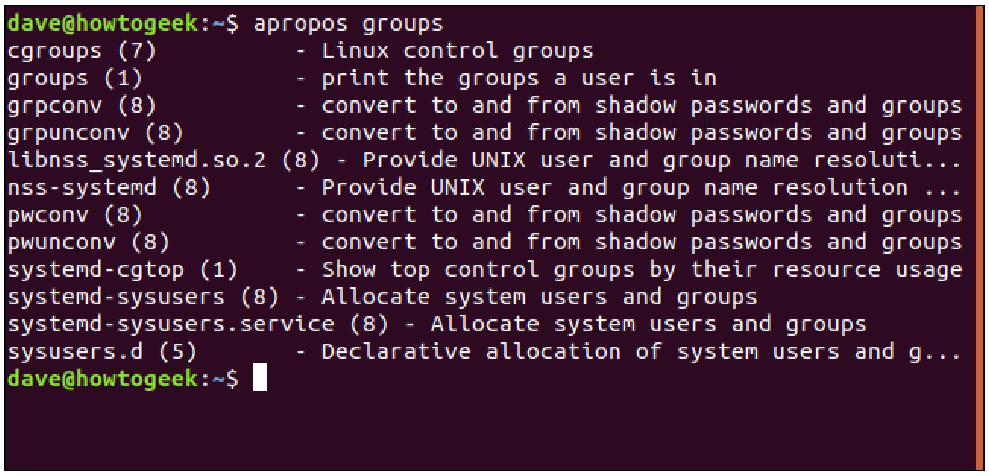
The word apropos means “related to” or “concerning,” and the command `apropos` took its name from this. To search for anything related to the `groups` command, we can use this command:

```
apropos groups
```



```
dave@howtogeek:~$ apropos groups
```

`apropos` lists the results to the terminal window.



```
dave@howtogeek:~$ apropos groups
cgroups (7)          - Linux control groups
groups (1)           - print the groups a user is in
grpconv (8)          - convert to and from shadow passwords and groups
grpunconv (8)        - convert to and from shadow passwords and groups
libnss_systemd.so.2 (8) - Provide UNIX user and group name resoluti...
nss-systemd (8)      - Provide UNIX user and group name resolution ...
pwconv (8)           - convert to and from shadow passwords and groups
pwunconv (8)         - convert to and from shadow passwords and groups
systemd-cgtop (1)    - Show top control groups by their resource usage
systemd-sysusers (8) - Allocate system users and groups
systemd-sysusers.service (8) - Allocate system users and groups
sysusers.d (5)       - Declarative allocation of system users and g...
dave@howtogeek:~$ 
```

Using More Than One Search Term

You can use more than one search term on the command line. `apropos` will search for man pages that contain *either* of the search terms.

```
apropos chown chmod
```

The results are listed as before. In this case, there is a single entry for each of the search terms.

Using Exact Matches

`apropos` will return man pages that contain the search term even if the term is in the middle of another word. To make `apropos` return only exact matches for the search term, use the `-e` (exact) option.

To illustrate this, we'll use `apropos` with `grep` as the search term.

```
apropos grep
```

There are many results returned for this, including many where `grep` is incorporated in another word, such as `bzfgrep`.

Let’s try that again and use the `-e` (exact) option.

```
apropos -e grep
```

We have a single result this time, for what we were actually searching for.

Matching All Search Terms

As we saw earlier if you provide more than one search term `apropos` will search for man pages that contain *either* search term. We can change that behavior by using the `-a` (and) option. This makes `apropos` only select matches that have all of the search times in them.

Let’s try the command without the `-a` option so that we can see what results `apropos` gives.

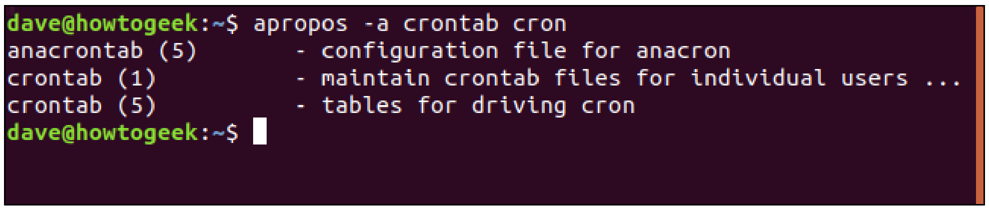
```
apropos crontab cron
```

The results include man pages that match one or the other of the search terms.

Now we'll use the `-a` option.

```
apropos -a crontab cron
```

This time the results are narrowed down to those containing both search terms.



```
dave@howtogeek:~$ apropos -a crontab cron
anacrontab (5)       - configuration file for anacron
crontab (1)          - maintain crontab files for individual users ...
crontab (5)          - tables for driving cron
dave@howtogeek:~$ 
```

Yet More Options

All of these commands have more options—some of them many more options—and you are encouraged to read the man pages for the commands we've discussed in this article.

Here's a quick summary for each command:

- *find*: Provides a feature rich and granular search capability to look for files and directories.

- *locate*: Provides a fast database-driven search for programs and commands.
- *which*: Searches the $PATH looking for executable files
- *whereis*: Searches the $PATH looking for executable files, man pages, and source code files.
- *whatis*: Searches the man one-line descriptions for matches to the search term.
- *apropos*: Searches the man page with more fidelity than whatis, for matches to the search term or terms.

Looking for more Linux terminal information? Here are 37 commands you should know.

RELATED: ***37 Important Linux Commands You Should Know***

READ NEXT

› What Should You Do If You Receive a Phishing Email?

› Why Do You Have to Log In to Your Home PC, Anyway?

› The Best New Features in Android 10, Available Now

› How to Use the chown Command on Linux

› What's New in Windows 10's 20H1 Update, Arriving Spring 2020

DAVE MCKAY

Dave McKay first used computers when punched paper tape was in vogue, and he has been programming ever since. He is now a Data Protection Officer and has worked as a freelance programmer, manager of an international software development team, and an IT services project manager. Dave is a Linux evangelist and open source advocate. **READ FULL BIO »**