Find Files and Do Stuff with GNU find

Serge Y. Stroobandt

Introduction

I can still remember how my first encounter with the `find` command infused a considerable amount of «shock and awe» in me. This had all the looks of certainly not going to be an easy command to learn! Over the years, I painstakingly gathered a number `find` recipes which eventually led to the inception of this article.

Nowadays, I find `find` hardly difficult. Writing this article helped a lot in getting a feel for `find`. Knowing that I can always fall back on this recipe collection certainly also strengthened my present day confidence.
Are you equally interested in becoming a `find` guru?
Now, do yourself a favour and have a quick look at `man find`.
Done? Let us continue then with some examples for fun and profit...

When not to use find

Gain speed with `locate`

When looking for a hard to find file or directory that already exists for some while on your drive, do not use `find` but use `locate` instead.

```
$ locate filename
```

The reason is queries with `locate` execute much faster than with `find`. Unlike `find`, `locate` uses a previously created database to perform the search. This database is updated periodically from `cron`. Nevertheless, there are many instances where the use of `find` is required, albeit because `find` offers an `-exec` clause. Numerous examples are given below.

Unleash the power of `globstar`

Here is a real-life example where I unleashed the power of `globstar`. The Xubuntu icon would not appear on the XFCE application menu after installing Xubuntu on a system which already had a `/home` directory made by a preceding XFCE distribution that lost my favour. I knew the icon file name would start with `xu`, end in `.png` and that it would be located somewhere in the "unique system resources" folder named `/usr`.

Finding this file with `find` would require a rather intricate regular expression. However, things turn out much simpler when you uncomment or add `shopt -s globstar` in `~/.bashrc`. Now you can do the following:

```
$ cd /usr
/usr $ locate **/xu*.png
/usr $ ls **/xu*.png
```

Both commands yielded precisely nine full file paths on my system, among which the desired `/usr/share/pixmaps/xubuntu-logo-menu.png`. Interestingly, the brute-force `ls` happens to be almost 36% faster than the database look-up of `locate`:

```
/usr $ time locate **/xu*.png
4.2s
/usr $ time ls **/xu*.png
2.7s
```

Rename multiple files

Multiple files can be renamed by specifying a regular expression for string substation. This requires first installing the `rename` command.

```
$ sudo apt install rename
$ rename 's/xenial/bionic/' *
```

List only hidden files & directories in the current directory

```
$ ls -d .*
```

As a long list:

```
$ ls -dl .*
```

Stricter, without `.` nor `..`:

```
$ ls -dl .!(|.)
```

List any text file containing a specific string

This can be done with a recursive `grep` command. Notice the uncharacteristic small case `-r`. A capital `-R` will also follow symbolic links. Do not add any file specification like `*.txt` because then `grep` will come up only with files in the current directory; not with those in subdirectories. If wildcard searches are required, use `find` instead (see below). Otherwise, `grep -r` will search only in the working directory!

```
$ grep -r 'class="fixed"'
```

List

List specific text files containing a specific string

If you are indifferent to the text file name pattern, use `grep -r` instead. If not, the following command lists the path and name of text files ending in `.desktop` and containing the text `searchstring`:

```
$ find . -name '*.desktop' -exec grep -l searchstring {} \;
```

> To avoid surprises, always use **single quotes** around `find` arguments containing **wildcards.**

The option `-l` suppresses the normal output of `grep` and prints the file names with matches instead. Instead, use option `-H` if both the file name and the found string in its context needs to be shown.

```
$ find . -name '*.desktop' -exec grep -H searchstring {} \;
```

Find files without descending into directories

With `-maxdepth 1`, `find` will not descend into underlying directories.

```
$ find . -maxdepth 1 -type f -name '*.md'
```

Find and list directories

The following command will find all directories called `doc`, print their name preceded by an empty line and list their contents in a detailed, human-readable format. This example also shows how multiple commands can be executed with `find` by adding an `-exec` for each command.

```
$ find . -iname doc -exec echo -e \\n{} \; -exec ls -lh {} \;
```

Count

Recursively count files except hidden ones

```
$ find . -type f -not -path '*/\.*' |wc -l
```

Count directories

Finally, a little extra about listing and counting only directories —and nothing else— in the current directory. By now, you might think you would need `find` for that. Surprisingly, `ls` and `wc` will do just fine. List only directories in the current folder with

```
$ ls -dl */
```

Piping through `wc -l` will count the number of lines printed by the preceding list command.

```
$ ls -dl */ |wc -l
```

File operations

Remove many files

As strange as it may sound, `rm *` has its limits as for the number of files that it may delete. Upon hitting this limit, the `rm` command will complain along the lines of:

```
-bash: /usr/bin: Argument list too long
```

Luckily, in situations like this, the `find` command comes to the rescue.

```
$ sudo find . -type f -delete
```

If, for example, more than one file extension is targeted, use the or flag `-o` between escaped parentheses `\(` and `\)`:

```
$ sudo find . -type f \( -iname '*.jar' -o -iname '*.zip' \) -delete
```

Copy found files to another directory

Here is an example of searching files with `hf` in the file name and copying those to a directory called `hf/`. Matching file names in both the current and subdirectories will be found.

```
$ find . -type f -iname '*hf*' -print -exec cp {} hf/ \;
```

Rename files recursively

Here is a convoluted example which finds all files ending in `_a4.pdf` and renames them to ending in `.a4.pdf`. Note that using multiple `-exec` additions would not work here. Instead, one resorts to a single `-exec sh -c '...$1...' _ {} \;`.

```
$ find . -name '*_a4.pdf' -exec sh -c 'mv "$1" "$(echo "$1" |sed
s/_a4.pdf\$/.a4.pdf/)"' _ {} \;
```

Add an `echo` in front of the `mv` to test out the command first:

```
$ find . -name '*_a4.pdf' -exec sh -c 'echo mv "$1" "$(echo "$1" |sed
s/_a4.pdf\$/.a4.pdf/)"' _ {} \;
```

Replace a string in text files recursively

Example: Ansible

In this example, the text sting `include:` needs to be replaced with `import_tasks:` recursively in all YAML files of an Ansible directory structure.

A test with `grep` is performed first to see what would be captured:

```
$ find . -type f -name '*.yml' -readable -writable -exec grep include: {} \;
```

Then, proceed with a `sed` dry run. The `sed` option `-n` is a synonym for `--quiet` and the `p` at the very end of the `sed` expression will print the current pattern space.

```
$ find . -type f -name '*.yml' -readable -writable -exec sed -n
's/include:/import_tasks:/gp' {} \;
```

If everything looks fine, execute the definitive command by replacing `sed` option `-n` by `-i` for "in place." Furthermore, the `p` at the end should be removed. Failing to do so would result in lines being replaced twice.

```
$ find . -type f -name '*.yml' -readable -writable -exec sed -i
's/include:/import_tasks:/g' {} \;
```

Example: M3U playlists

The DeaDBeeF music player, saves `.m3u` playlist files with absolute file path references. This renders such playlists useless on any other device with a differing file system. Below command will recursively find all `.m3u` playlist files and render their file path references relative:

```
$ find . -type f -name '*.m3u' -readable -writable -exec sed -i 's|.*/||' {}
\;
```

Note that the vertical bar `|` is merely used as a delimiter here, replacing the usual forward slash `/`. The latter cannot be readily employed because it constitutes the target.

Example: ATX-style Markdown headers

Since the advent of `pandoc` version 2, ATX-style headers require an additional space to be inserted right after hashes signalling a Markdown header. The Markdown source files of this very site were automatically adapted, all at once, by executing the following command:

```
$ find . -type f -name '*.md' -readable -writable -exec sed -i
's|\(^##*\)\([^# \.]\)|\1 \2|' {} \;
```

The `sed` subcommand is explained as follows:

- `-i` edits the markdown file 'in place.'
- `s|…|…|` is a single substitution per line.
- Each `\(…\)` denotes a part in the search expression.
- `\1` and `\2` refer to the first, respectively, second part of the search expression.
- `^##*` means that the line should start `^` with one hash `#`, followed by zero or more hashes `#*`.
- The second search sequence part should start neither with a hash, space nor period `[^# \.]`.

Note that a simpler `sed -i 's|^##*|& |'` would still insert a space, even when there is already a space behind the starting hash sequence.

Example: Ubuntu PPAs

In this example, the distribution name in custom Ubuntu private package archive (PPA) repositories needs to be updated from `xenial` to `bionic`. This needs to be performed with `sudo` privileges in the directory `/etc/apt/sources.list.d/`.

A test with `grep` is performed first to see what would be captured:

```
$ cd /etc/apt/sources.list.d/
$ sudo find . -type f -readable -writable -exec grep xenial {} \;
```

Now, proceed with a `sed` dry run. The `sed` option `-n` is a synonym for `--quiet` and the `p` at the very end of the `sed` expression will print the current pattern space.

```
$ sudo find . -type f -readable -writable -exec sed -n 's/xenial/bionic/gp'
{} \;
```

If everything looks fine, execute the definitive command by replacing `sed` option `-n` by `-i` for "in place." Furthermore, the `p` at the end should be removed. Failing to do so would result in lines being replaced twice.

```
$ sudo find . -type f -readable -writable -exec sed -i 's/xenial/bionic/g'
{} \;
```

Finally, use the `rename` command (as previously explained) to rename the multiple `/etc/apt/sources.list.d/` files.

```
$ sudo apt install rename
$ sudo rename 's/xenial/bionic/' *
```

Permissions

Change ownership recursively

If not too many files and folders are involved this can be done using `chown -R` recursively:

```
$ sudo chown -R serge:family foldername
```

Otherwise, use:

```
$ sudo find . -print -exec chown serge:family {} \;
```

Change top-level non-hidden directory permissions

```
$ find . -maxdepth 1 -type d -name '[!.]*' -print -exec chmod 770 {} \;
```

Change directory permissions recursively

From time to time, I am still tempted to make the naive mistake of recursively changing the permissions of entire directory trees with `chmod -R 770 *`. This is plain wrong of course, as files within the sub-directories will also be made executable. The end result is a major security issue.

The correct way of tackling this problem is:

```
$ sudo find . -type d -print -exec chmod 770 {} \;
```

Change file permissions recursively

Similar to the previous code snippet, but then for files. Again, one cannot simply do `chmod -R 640 *`, as this would render directories unbrowsable because of a failing execution bit.

```
$ sudo find . -type f -print -exec chmod 640 {} \;
```

Prevent others from deleting files in your folder

Sticky bit

If on a shared drive, a directory get its *sticky bit* set, it becomes an append-only directory. Putting it more accurately; it becomes a directory in which files may only be removed or renamed by a user if the user has write permission for the directory *and* the user is also the owner of the file, the owner of the directory, or root.

Without the directory *sticky bit*, any user with group write and execute permissions for the directory can rename or delete files, even when that user is not the owner of those files. With the directory *sticky bit* set, only the owner can delete or rename his or her files. However, other users with write permissions are still allowed to edit files they do not own residing in *sticky bit* folders.

A *sticky bit* on files is of no relevance to the Linux kernel.

The *sticky bit* is set either with `chmod +t` or by adding the octal value `1000` to the usual octal absolute mode number:

```
$ sudo find . -type d -print -exec chmod 1770 {} \;
```

A directory which is executable by the `others` class, and which has the *sticky bit* set, will have the final `x` changed to a lower case `t` when listed by `ls -l`:

```
drwxrwxrwx   →   drwxrwxrwt
```

A directory which is not executable by the `others` class, but with the *sticky bit* set, will have the final `-` changed to a capital `T` when listed by `ls -l`:

```
drwxrwx---   →   drwxrwx--T
```

Let new subdirectories & files inherit the group ID

Setting the sticky bit to a directory will become even more useful when newly created files and subdirectories inside that directory would automatically inherent the same group ownership of this parent directory. This can be achieved by *setting the group ID upon execution* (`setgid` or SGID) to any such parent directory. Simply add the octal value `2000` to the usual octal absolute mode number of any parent directory or use `chmod g+s`:

```
$ sudo find . -type d -print -exec chmod 2770 {} \;
```

Setting the *sticky bit* and `setgid` can be combined by adding the octal value `3000` instead:

```
$ sudo find . -type d -print -exec chmod 3770 {} \;
```

However, inheritance of group ownership will only occur when the creator of any new file or subdirectory is also a member of that group. Furthermore, inheriting the sticky bit of a parent directory is unfortunately not possible.

> **Caveat: One does not want to set the group ID upon execution for files,** because that might render these executable by the `others` class; i.e. the whole world.

A directory which is executable by the `others` class, and which has both the *sticky bit* and the *SGID* set, will have the group `x` changed to a lower case `s` and the final `x` changed to a lower case `t` when listed by `ls -l`:

```
drwxrwxrwx   →   drwxrwsrwt
```

A directory which is not executable by the `others` class, but with both the *sticky bit* and the *SGID* set, will have the group `x` changed to a lower case `s` and the final `-` changed to a capital `T` when listed by `ls -l`:

```
drwxrwx---   →   drwxrws--T
```

> **Caveat:** As outlined above, newly created subdirectories will inherit the group ID through `setgid`. However, **newly created subdirectories cannot inherit the sticky bit from their parent directory.** Personally, I consider this a GNU/Linux deficiency.

Directories without the sticky bit can be found with below statement whereby the hyphen `-` in front of the value `1000` should be considered as a prefix which will match this and higher permission values.

```
$ find . -type d \! -perm -1000
```

The goal is to set the sticky bit and `setgid` of those directories which parent directory has the sticky bit set. That would call for a nested `find` statement, if it were not that `find` statements cannot be nested. This is why rather a pipe with a recursive conditional will be employed on the directories found without a sticky bit. The `-k` test returns true if the sticky bit —in this case, of the parent directory— is set.

```
$ find . -type d \! -perm -1000 |while read d; do if [[ -k "$d/.." ]]; then
chmod +t,g+s "$d"; fi; done
```

It might also be handy to overwrite the `mkcd` command.

Modification date

Find files modified after a date

```
$ find . -type f -newermt 2017-06-19
```

Find recently modified files recursively

Hidden paths (i.e. directories or files) are ignored using `-not -path '*/\.*'`. If only hidden files but no hidden directories are to be ignored, use `-name '[!.]*'` instead. More files are returned by changing the number at the end of the `|head -n 10` pipe.

```
$ find . -type f -not -path '*/\.*' -printf '%TY.%Tm.%Td %THh%TM %Ta %p\n'
|sort -nr |head -n 10
2017.01.28 07h27 Sat ./find/en/index.html
2017.01.28 07h27 Sat ./find/en/find.md
2017.01.28 07h00 Sat ./propagation/en/propagation.md
2017.01.27 11h26 Fri ./propagation/images/owf.svg
2017.01.27 11h24 Fri ./propagation/en/index.html
```

Find recently modified directories recursively

`Permission denied` error messages are filtered out. More directories are returned by changing the number at the end of the `|head -n 10` pipe.

```
$ { find . -type d -not -path '*/\.*' -printf '%TY.%Tm.%Td %THh%TM %Ta %p\n'
2> >(grep -v 'Permission denied' >&2); } |sort -nr |head -n 10
```

Find more recent files

Calling `find` with the `-newer now` will return objects that have been more recently modified than the file called `now`.

```
$ touch now
$ find . -type f -newer now
```

Empty

Find empty files

There are two ways to go about this:

```
$ find . -type f -name '*.bib' -empty
$ find . -type f -name '*.bib' -size 0
```

Find and delete empty files

Again, use either the `-empty` or `-size 0` command option.

```
$ find . -type f -name '*.bib' -empty -delete
$ find . -type f -name '*.bib' -size 0 -delete
```

Add the `-ok` option to approve every deletion on a file by file basis.

Symbolic links

Find symbolic links

```
$ find . -type l
```

Detailed listing of what find found

Continuing from the previous example:

```
$ find . -type l -ls
```

This will print all details about the found symbolic links, including their target.

Find symbolic links to a specific target

```
$ find . -lname link_target
```

Note that `link_target` may contain wildcard characters.

Find broken symbolic links

```
$ find -L . -type l -ls
```

The `-L` option instructs find to follow symbolic links, unless when broken.

Find & replace broken symbolic links

```
$ find -L . -type l -delete -exec ln -s new_target {} \;
```

Create a site map

To promote a website, it is a good idea to creating a site map. This will inform web search engines about the URLs that are available for web crawling. In it its simplest form, such a site map may take the form of a plain text file.

At first instance, I am only interested in mentioning my English language pages which, in my particular case, have a file path `'*/en/*.html'`. Hence, `find` is required to search on full paths. This is achieved with the command line option `-wholename`.

Furthermore, `find` should also output full paths. For this to happen, `find`'s first argument should be a full path. Lazy as we are, the command `$(pwd)` is used as the first argument.

To finish off, a piped `sed` command will convert the full paths to fully qualified URLs. The piped `sort` command is merely to keep humans happy.

```
$ cd /home/serge/public_html/hamwaves.com/
find $(pwd) -wholename '*/en/*.html' |sed
's|^/home/serge/public_html/|https://|' |sort > sitemap.txt
```

Find directories with a makefile link

On rare occasions, all pages on this website need to be rebuilt. This happens normally only for structural maintenance purposes; e.g. when the text at the bottom of each page changes. The following command finds all directories containing a symbolic link called `makefile` and runs the `make` command there.

```
$ find . -type l -name makefile -exec sh -c 'make --always-make --
directory=$(dirname {})' \;
```

Find Hg Mercurial repositories to update

The initial `-exec echo \;` is only there to keep the output clean.

```
$ find . -type d -iname .hg -exec echo \; -exec hg pull -u -R {}/.. \;
```

Character encoding

List the character encoding of text files

```
$ find . -type f -iname '*.txt' -exec file -i {} \;
```

Change the character encoding of text files to UTF-8

The character encoding of all matching text files gets detected automatically and all matching text files are converted to `utf-8` encoding:

```
$ sudo apt install dos2unix
$ find . -type f -iname '*.txt' -exec sh -c 'iconv -f $(file -bi "$1" |sed
-e "s/.*[ ]charset=//") -t utf-8 -o /tmp/converted "$1" && mv /tmp/converted
"$1" && dos2unix "$1"' -- {} \;
```

First, ensure that `dos2unix` is installed on the system. In the `find` command, a sub shell `sh` is used with `-exec`, running a one-liner with the `-c` flag. Evoking `-- {}` at the end of the `find` command passes the filename as the positional argument `"$1"`. In between, the `utf-8` output file is temporarily saved as `/tmp/converted`. Finally, `dos2unix` removes any DOS `^M` carriage returns, leaving only line feeds for Unix-like systems. The whole one-liner can be used as it is in a shell script.