Master the Linux ls command

Linux's ls command has a staggering number of options that can provide important information about your files.

The **ls** command lists files on a POSIX system. It's a simple command, often underestimated, not in what it can do (because it really does only one thing), but in how you can optimize your use of it.

Of the 10 most essential terminal commands to know, the humble **ls** command is in the top three, because **ls**doesn't *just* list files, it tells you important information about them. It tells you things like who owns a file or directory, when each file was lost or modified, and even what kind of file it is. And then there's its incidental function of giving you a sense of where you are, what nearby objects are lying around, and what you can do with them.

If your experience with **ls** is limited to whatever your distribution aliases it to in **.bashrc**, then you're probably missing out.

GNU or BSD?

Before looking at the hidden powers of **ls**, you must determine which **ls** command you're running. The two most popular versions are the GNU version, included in the GNU **coreutils** package, and the BSD version. If you're running Linux, then you probably have **ls**installed. If you're running BSD or MacOS, then you have the BSD version. There are differences, for which this article accounts.

You can find out which version is on your computer with the **--version** option:

```
$ ls --version
```

If this returns information about GNU coreutils, then you have the GNU version. If it returns an error, you're probably running the BSD version (run **man ls | head**to be sure).

You should also investigate what presets your distribution may have in place. Customizations to terminal commands are frequently placed in **$HOME/.bashrc** or **$HOME/.bash_aliases** or **$HOME/.profile**, and they're accomplished by aliasing **ls** to a more complex **ls** command. For example:

```
alias ls='ls --color'
```

The presets provided by distributions are very helpful, but they do make it difficult to discern what **ls** does on its own and what its additional options provide. Should you ever want

to run **ls** and not the alias, you can "escape" the command with a backslash:

```
$ \ls
```

Classify

Run on its own, **ls** simply lists files in as many columns as can fit into your terminal:

```
$ ls ~/example
bunko         jdk-10.0.2
chapterize    otf2ttf.ff
despacer      overtar.sh
estimate.sh   pandoc-2.7.1
fop-2.3       safe_yaml
games         tt
```

It's useful information, but all of those files look basically the same without the convenience of icons to quickly convey which is a directory, or a text file, or an image, and so on.

Use the **-F** (or **--classify** on GNU) to show indicators after each entry that identify the kind of file it is:

```
$ ls ~/example
bunko          jdk-10.0.2/
chapterize*    otf2ttf.ff*
despacer*      overtar.sh*
estimate.sh    pandoc@
fop-2.3/       pandoc-2.7.1/
games/         tt*
```

With this option, items listed in your terminal are classified by file type using this shorthand:

- A slash (**/**) denotes a directory (or "folder").
- An asterisk (*****) denotes an executable file. This includes a binary file (compiled code) as well as scripts (text files that have executable permission).
- An at sign (**@**) denotes a symbolic link (or "alias").
- An equals sign (**=**) denotes a socket.
- On BSD, a percent sign (**%**) denotes a whiteout (a method of file removal on certain file systems).
- On GNU, an angle bracket (**>**) denotes a door (inter-process communication on Illumos and Solaris).
- A vertical bar (**|**) denotes a FIFO.

A simpler version of this option is **-p**, which only differentiates a file from a directory.

Long list

Getting a "long list" from **ls** is so common that many distributions alias **ll** to **ls -l**. The long list form provides many important file attributes, such as permissions, the user who owns each file, the group to which the file belongs, the file size in bytes, and the date the file was last changed:

```
$ ls -l
-rwxrwx---. 1 seth users        455 Mar  2  2017 esti
```

```
-rwxrwxr-x. 1 seth users       662 Apr 29 22:27 fact
-rwxrwx---. 1 seth users  20697793 Jun 29  2018 fop-
-rwxrwxr-x. 1 seth users      6210 May 22 10:22 gete
-rwxrwx---. 1 seth users       177 Nov 12  2018 html
[...]
```

If you don't think in bytes, add the **-h** flag (or **--human**in GNU) to translate file sizes to more human-friendly notation:

```
$ ls --human
-rwxrwx---. 1 seth users   455 Mar  2  2017 estimate.
-rwxrwxr-x. 1 seth seth    662 Apr 29 22:27 factorial
-rwxrwx---. 1 seth users   20M Jun 29  2018 fop-2.3-b
-rwxrwxr-x. 1 seth seth  6.1K May 22 10:22 geteltori
-rwxrwx---. 1 seth users   177 Nov 12  2018 html4mutt
```

You can see just a little less information by showing only the owner column with **-o** or only the group column with **-g**:

```
$ ls -o
-rwxrwx---. 1 seth    455 Mar  2  2017 estimate.sh
-rwxrwxr-x. 1 seth    662 Apr 29 22:27 factorial
-rwxrwx---. 1 seth    20M Jun 29  2018 fop-2.3-bin.tar
-rwxrwxr-x. 1 seth   6.1K May 22 10:22 geteltorito
-rwxrwx---. 1 seth    177 Nov 12  2018 html4mutt.sh
```

Combine both options to show neither.

Time and date format

The long list format of **ls** usually looks like this:

```
-rwxrwx---. 1 seth users      455 Mar  2  2017 esti
-rwxrwxr-x. 1 seth users      662 Apr 29 22:27 fact
-rwxrwx---. 1 seth users 20697793 Jun 29  2018 fop-
-rwxrwxr-x. 1 seth users     6210 May 22 10:22 gete
-rwxrwx---. 1 seth users      177 Nov 12  2018 html
```

The names of months aren't easy to sort, both computationally or (depending on whether your brain tends to prefer strings or integers) by recognition. You can change the format of the time stamp with the **--time-style** option plus the name of a format. Available formats are:

- full-iso (1970-01-01 21:12:00)
- long-iso (1970-01-01 21:12)
- iso (01-01 21:12)
- locale (uses your locale settings)
- posix-STYLE (replace STYLE with a locale definition)

You can also create a custom style using the formal notation of the **date** command.

Sort by time

Usually, the **ls** command sorts alphabetically. You can make it sort according to which file was most recently changed (the newest is listed first) with the **-t** option.

For example:

```
$ touch foo bar baz
$ ls
bar  baz  foo
$ touch foo
$ ls -t
foo bar baz
```

List type

The standard output of **ls** balances readability with space efficiency, but sometimes you want your file list in a specific arrangement.

For a comma-separated list of files, use **-m**:

```
ls -m ~/example
bar, baz, foo
```

To force one file per line, use the **-1** option (that's the number one, not a lowercase L):

```
$ ls -1 ~/bin/
bar
baz
foo
```

To sort entries by file extension rather than the filename, use **-X** (that's a capital X):

```
$ ls
bar.xfc  baz.txt  foo.asc
$ ls -X
foo.asc  baz.txt  bar.xfc
```

Hide the clutter

There are a few entries in some **ls** listings that you may not care about. For instance, the metacharacters **.** and **..** represent "here" and "back one level," respectively. If you're familiar with navigating in a terminal, you probably already know that each directory refers to itself as **.** and to its parent as **..**, so you don't need to be constantly reminded of it when you use the **-a** option to show hidden files.

To show almost all hidden files (the **.** and **..** excluded), use the **-A** option:

```
$ ls -a
.
..
.android
.atom
.bash_aliases
[...]
$ ls -A
.android
.atom
.bash_aliases
[...]
```

With many good Unix tools, there's a tradition of saving backup files by appending some special character to the name of the file being saved. For instance, in Vim, backups get saved with the **~**character appended to the name.

These kinds of backup files have saved me from stupid mistakes on several occasions, but after years of enjoying the sense of security they provide, I don't feel the need to have visual evidence that they exist. I trust Linux applications to generate backup files (if they claim to do so), and I'm happy to take it on faith that they exist.

To hide backup files from view, use **-B** or **--ignore-backups** to conceal common backup formats (this option is not available in BSD **ls**):

```
$ ls
bar.xfc  baz.txt  foo.asc~  foo.asc
$ ls -B
bar.xfc  baz.txt  foo.asc
```

Of course, the backup file still exists; it's just filtered out so that you don't have to look at it.

GNU Emacs saves backup files (unless otherwise configured) with a hash character (**#**) at the start and end of the file name (**#file#**). Other applications may use a different style. It doesn't matter what pattern is used, because you can create your own exclusions with the **--hide** option:

```
$ ls
bar.xfc  baz.txt  #foo.asc#  foo.asc
$ ls --hide="#*#"
bar.xfc  baz.txt  foo.asc
```

List directories with recursion

The contents of directories are not listed with the **ls**command unless you run **ls** on that directory specifically:

```
$ ls -F
example/  quux*  xyz.txt
$ ls -R
quux  xyz.txt

./example:
bar.xfc  baz.txt  #foo.asc#  foo.asc
```

Make it permanent with an alias

The **ls** command is probably the command used most often during any given shell session. It's your eyes and ears, providing you with context and confirming the results of commands. While it's useful to have lots of options, part of the beauty of **ls** is its brevity: two characters and the Return key, and you know exactly where you are and what's nearby. If you have to stop to think about (much less type) several different options, it becomes less convenient, so typically even the most useful options are left off.

The solution is to alias your **ls** command so that when you use it, you get the information you care about the most.

To create an alias for a command in the Bash shell, create a file in your home directory called **.bash_aliases** (you must include the dot at the beginning). In this file, list the command you want to create an alias for and then the alias you want to create. For example:

```
alias ls='ls -A -F -B --human --color'
```

This line causes your Bash shell to interpret the **ls**command as **ls -A -F -B --human --color**.

You aren't limited to redefining existing commands. You can create your own aliases:

```
alias ll='ls -l'
alias la='ls -A'
alias lh='ls -h'
```

For aliases to work, your shell must know that the **.bash_aliases** configuration file exists. Open the **.bashrc** file in an editor (or create it, if it doesn't exist), and include this block of code:

```
if [ -e $HOME/.bash_aliases ]; then
    source $HOME/.bash_aliases
fi
```

Each time **.bashrc** is loaded (which is any time a new Bash shell is launched), Bash will load **.bash_aliases**into your environment. You can close and relaunch your Bash session or just force it to do that now:

```
$ source ~/.bashrc
```

If you forget whether you have aliased a command, the **which** command tells you:

```
$ which ls
alias ls='ls -A -F -B --human --color'
        /usr/bin/ls
```

If you've aliased the **ls** command to itself with options, you can override your own alias at any time by prefacing **ls** with a backslash. For instance, in the example alias, backup files are hidden using the **-B**option, which means there's no way to back up files with the **ls** command. Override the alias to see the backup files:

```
$ ls
bar  baz  foo
$ \ls
bar  baz  baz~  foo
```

Do one thing and do it well

The **ls** command has a staggering number of options, many of which are niche or highly dependent upon the terminal you

use. Take a look at **info ls** on GNU systems or **man ls** on GNU or BSD systems for more options.

You might find it strange that a system famous for the premise that each tool "does one thing and does it well" would weigh down its most common command with 50 options. But **ls** does only one thing: it lists files. And with 50 options to allow you to control how you receive that list, **ls** does its one job very, *very* well.