

Reviewing Existing Work

When you're the sole developer on a project, it's easy to know what progress has been done on the project because you did everything yourself. Things can become a bit more complicated, though, when you're working on a team - whether that team is local in an office or if you are developing with someone just across the internet.

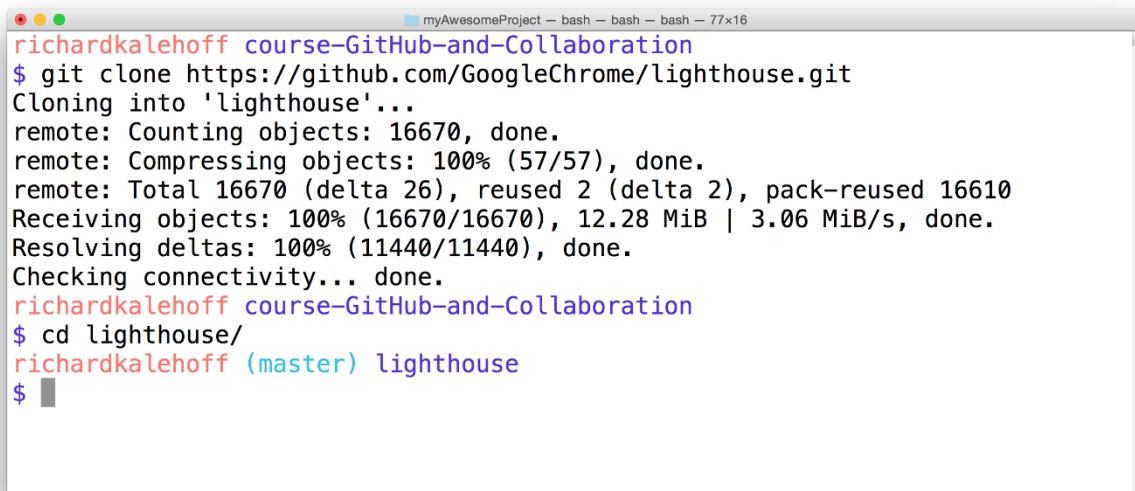
Sometimes it can be hard to see what the other developers have been doing on the project. Especially if developers are working across multiple different branches. How can I see all of the commits that Deepesh made? Or what if Christine made a change and said that her commit resolves issue 47 in our project's ticket tracking system. How can we discover the all of this information in the repository?

We can discover details about what other developers have done by using the extremely powerful `git log` command.

Clone Lighthouse Project

We first need to download a project that is being worked on by multiple different people. Let's download the Lighthouse project by Google that's an app for auditing, performance metrics, and best practices for Progressive web apps.

Here's the [Lighthouse project on GitHub](https://github.com/GoogleChrome/lighthouse).

A terminal window titled 'myAwesomeProject -- bash -- bash -- bash -- 77x16' showing the process of cloning the Lighthouse project. The user 'richardkalehoff' is in the directory 'course-GitHub-and-Collaboration'. They run the command 'git clone https://github.com/GoogleChrome/lighthouse.git'. The terminal output shows the cloning progress: 'Cloning into 'lighthouse'...', 'remote: Counting objects: 16670, done.', 'remote: Compressing objects: 100% (57/57), done.', 'remote: Total 16670 (delta 26), reused 2 (delta 2), pack-reused 16610', 'Receiving objects: 100% (16670/16670), 12.28 MiB | 3.06 MiB/s, done.', 'Resolving deltas: 100% (11440/11440), done.', and 'Checking connectivity... done.'. Finally, the user changes the directory to 'lighthouse/' and the prompt shows '(master) lighthouse'.

Cloning Google's Lighthouse project from GitHub.

Filtering Collaborator's Commits

Being able to narrow down the commits to just the ones you're looking for can be a chore. Let's look at the different ways we can discover information that our collaborators have done!

Group By Commit Author

This is not a massive project, but it does have well over 1,000 commits. A quick way that we can see how many commits each contributor has added to the repository is to use the `git shortlog` command:

```
$ git shortlog
```

```
Abby Armada (1):
  Create .codeclimate.yml (#1708)

Addy Osmani (7):
  Merge pull request #1 from samccone/sjs/tweak-loops
  Merge pull request #68 from paullewis/manitests
  Add support for sequentially batch-running URLs (#640)
  Replace instances of 'not unfunctioning' with 'still functional' (#764)
  Temporarily disable cache contains start_url audit in config (#766)
  Update TTI scoring label to 5000ms (matches guidance) (#947)
  Fixes #1012 - typo

Adriano Caheté (1):
  Fix for tick/cross symbols on Windows.

André Cipriani Bandarra (1):
  Improve check for used JS features (#544)

:
```

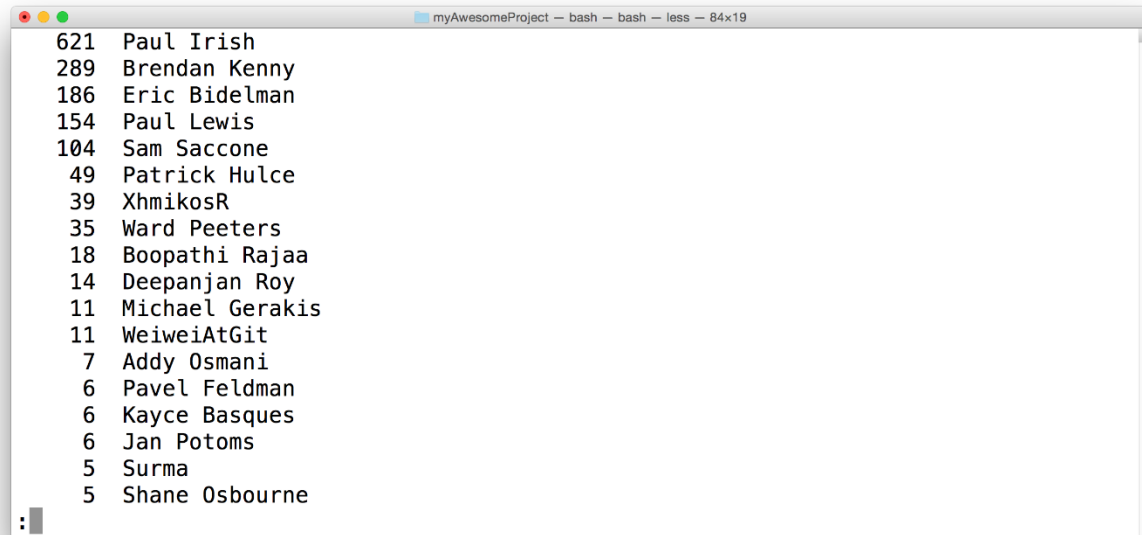
The terminal showing the results of running the `git shortlog` command. It displays all commits sorted by author.

In the screenshot above we can see that:

- Abby Armada has made one commit in the repository
- Addy Osmani has made seven commits
- Adriano Caheté has made one commit
- André Cipriani Bandarra has made one commit

`git shortlog` displays an alphabetical list of names and the commit messages that go along with them. If we just want to see just the number of commits that each developer has made, we can add a couple of flags: `-s` to show just the number of commits (rather than each commit's message) and `-n` to sort them numerically (rather than alphabetically by author name).

```
$ git shortlog -s -n
```



```
621 Paul Irish
289 Brendan Kenny
186 Eric Bidelman
154 Paul Lewis
104 Sam Saccone
49 Patrick Hulce
39 XhmikosR
35 Ward Peeters
18 Boopathi Rajaa
14 Deepanjan Roy
11 Michael Gerakis
11 WeiweiAtGit
7 Addy Osmani
6 Pavel Feldman
6 Kayce Basques
6 Jan Potoms
5 Surma
5 Shane Osbourne
```

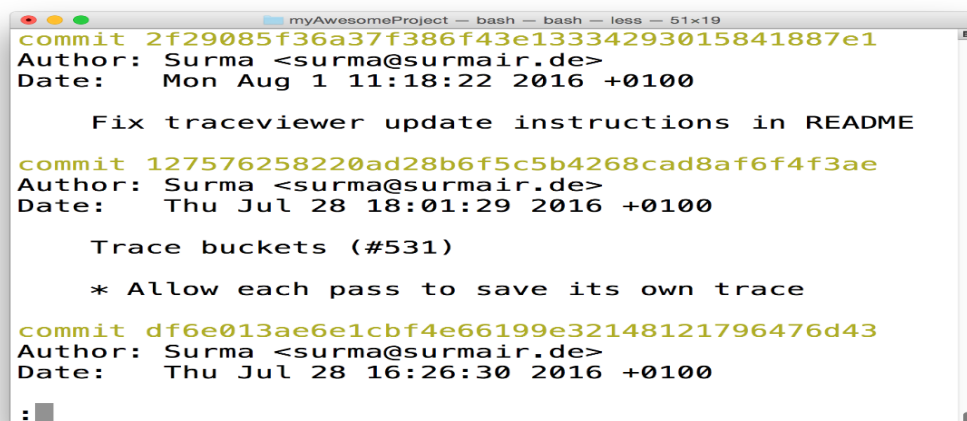
git shortlog with the -s -n flags to show only the number of commits each author has made, sorted numerically.

We can see in the image above that Surma has added five commits to the Lighthouse project. These five commits are spread out throughout the repository. What if we want to see *just* these five commits by Surma, though?

Filter By Author

Another way that we can display all of the commits by an author is to use the regular git log command but include the --author flag to filter the commits to the provided author.

```
$ git log --author=Surma
```



```
commit 2f29085f36a37f386f43e13334293015841887e1
Author: Surma <surma@surmair.de>
Date: Mon Aug 1 11:18:22 2016 +0100

    Fix traceviewer update instructions in README

commit 127576258220ad28b6f5c5b4268cad8af6f4f3ae
Author: Surma <surma@surmair.de>
Date: Thu Jul 28 18:01:29 2016 +0100

    Trace buckets (#531)

    * Allow each pass to save its own trace

commit df6e013ae6e1cbf4e66199e32148121796476d43
Author: Surma <surma@surmair.de>
Date: Thu Jul 28 16:26:30 2016 +0100
```

The terminal application showing the result of running git log --author=Surma. The output displays only the commits that Surma made.

Question 1 of 3

If you run `git shortlog -s -n`, again, you'll see that there is a "Paul Irish" and a "Paul Lewis". If the following command were run:

```
$ git log --author=Paul
```

What would it display?

Answer: commits by both Paul Irish and Paul Lewis

If we wanted to see only the commits by Paul Lewis we have to run:

```
$ git log --author="Paul Lewis"
```

⚠ Quotes Are Important ⚠

Pay attention to the use of the quotes in the previous command. If it were written *without* the quotes like this `git log --author=Paul Lewis`, it would not work. If it's formatted this way *without* the quotes, Git would think that Lewis is not part of the "author" flag, and it would cause an error.

What are the first seven characters of the SHA for Paul Lewis' *first* commit in the Lighthouse project?

Answer: c09a442

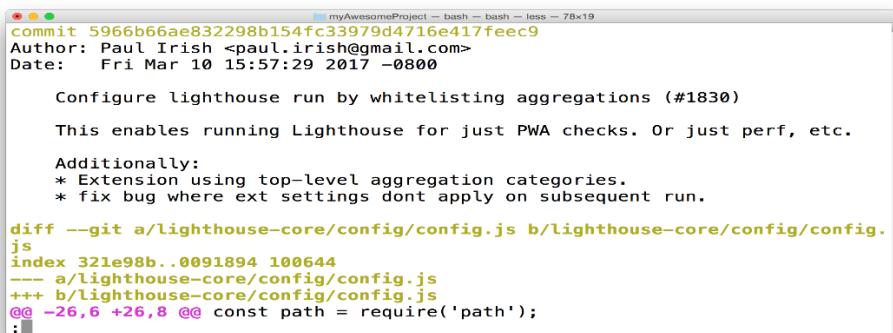
Filter Commits By Search

Before going through this section on filtering by searching, I feel like I need to stress how important it is to write good, *descriptive* commit messages. If you write a descriptive commit message, then it's so much easier to search through the commit messages, later, to find exactly what you're looking for.

And remember, if the commit message is not enough for you to explain what the commit is for, you can provide a detailed description of exactly why the commit is needed in the description area.

Let see an example of extra details in a commit in the lighthouse project by looking at commit `5966b66`:

```
$ git show 5966b66
```



```
commit 5966b66ae832298b154fc33979d4716e417feec9
Author: Paul Irish <paul.irish@gmail.com>
Date:   Fri Mar 10 15:57:29 2017 -0800

    Configure lighthouse run by whitelisting aggregations (#1830)

    This enables running Lighthouse for just PWA checks. Or just perf, etc.

    Additionally:
    * Extension using top-level aggregation categories.
    * fix bug where ext settings dont apply on subsequent run.

diff --git a/lighthouse-core/config/config.js b/lighthouse-core/config/config.js
index 321e98b..0091894 100644
--- a/lighthouse-core/config/config.js
+++ b/lighthouse-core/config/config.js
@@ -26,6 +26,8 @@ const path = require('path');
```

The terminal application showing commit 5966b66 from the Lighthouse project. The author has provided additional information about the commit.

The commit message is `Configure Lighthouse run by whitelisting aggregations (#1830)`. But there's a lot more text than just that. Beneath the commit message, you'll find a couple of lines with additional information about the commit. This section provides further information on the *why* this commit was needed.

So why do we care about all of this detail? For one thing, it's easier for you to go back and review the changes made to the repository, and it's easier for others to review the changes to. Another thing is filtering commits by information in the current message or description area.

We can filter commits with the `--grep` flag.

How about we filter down to just the commits that reference the word "bug". We can do that with either of the following commands:

```
$ git log --grep=bug
```

```
$ git log --grep bug
```

Watch Out For Spacing

Remember that spacing is an issue, here, too. If you're trying to search for something that is multiple words and has spaces between the words, you need to wrap everything in quotes. For example, to search for unit tests you would need to use the following command, `git log --grep="unit tests"`.

More On grep

If you don't know what `grep` is then the `--grep` flag might not seem like a logical choice for the flag's name. `Grep` is a pattern matching tool. It is way beyond the scope of this course to cover `grep`. But as a brief intro, if you were to run `git log --grep "fort"`, then Git will display only the commits that have the character `f` followed by the character `o` followed by `r` followed by `t`.

For more info on `Grep`, check out our [Shell Workshop course](#).

Question 3 of 3

One of the following browsers had a CSS bug that was fixed by a commit. Use `git log` and the `--grep` flag to figure out which browser had the bug.

Answer: Firefox

Recap

The `git log` command is extremely powerful, and you can use it to discover a lot about a repository. But it can be especially helpful to discover information about a repository that you're collaborating on with others. You can use `git log` to:

- group commits by author with `git shortlog`

```
$ git shortlog
```

- filter commits with the `--author` flag

```
$ git log --author="Richard Kalehoff"
```

- filter commits using the `--grep` flag

```
$ git log --grep="border radius issue in Safari"
```

grep is a complicated topic and you can find out more about it [here on the Wiki page](#) or in our [Shell Workshop course](#).