

# Chapter X -

## Contents

<b>Git Log</b>	<b>1</b>
Project set up so-far . . . . .	1
Looking at the small picture . . . . .	1
Basic git log . . . . .	2
Flag oneline . . . . .	2
Flag graph . . . . .	3
Looking at the big picture . . . . .	3
Flag all . . . . .	3
Wrapping up . . . . .	4

## Git Log

Now that you know how to do the basics in git and started working on the repo, a few commits in and you might ask yourself, how do we visualize these commits and changes we have made in the repository?

That is where this chapter, and the subsequent, **Git Diffs** come in. These commands give you a way to visualize the commits and changes made in your git repository.

To explain simply, the command `git log` and its corresponding optional flags lets you look at history of commits and how the link up in your repository. At the end of the chapter there will be links to additional resources, as well as more visual GUI tools you can use to look at the log of the repository, but I still suggest reading and learning this chapter.

This is because these tools still do rely on the git commands to generate their graphs, and understanding this command kind of gives you the ‘feel’ for it in general. Lastly, as this command comes with git, using the command does not require any additional setup other than having git.

Note: This chapter serves to show you and explain the common use cases for the command. For a more detailed explanation, look at the additional resources or the documentation of the command.

## Project set up so-far

Let’s say the project you are working on is a full-stack web application, with a frontend and backend. And you have decided to split the work to frontend and backend work, with respective features branches being worked on.

It may look something like this:

```
Github
origin/master, master
```

```
Teammate A
frontend/landing
```

```
Teammate B
backend/start-server
```

## Looking at the small picture

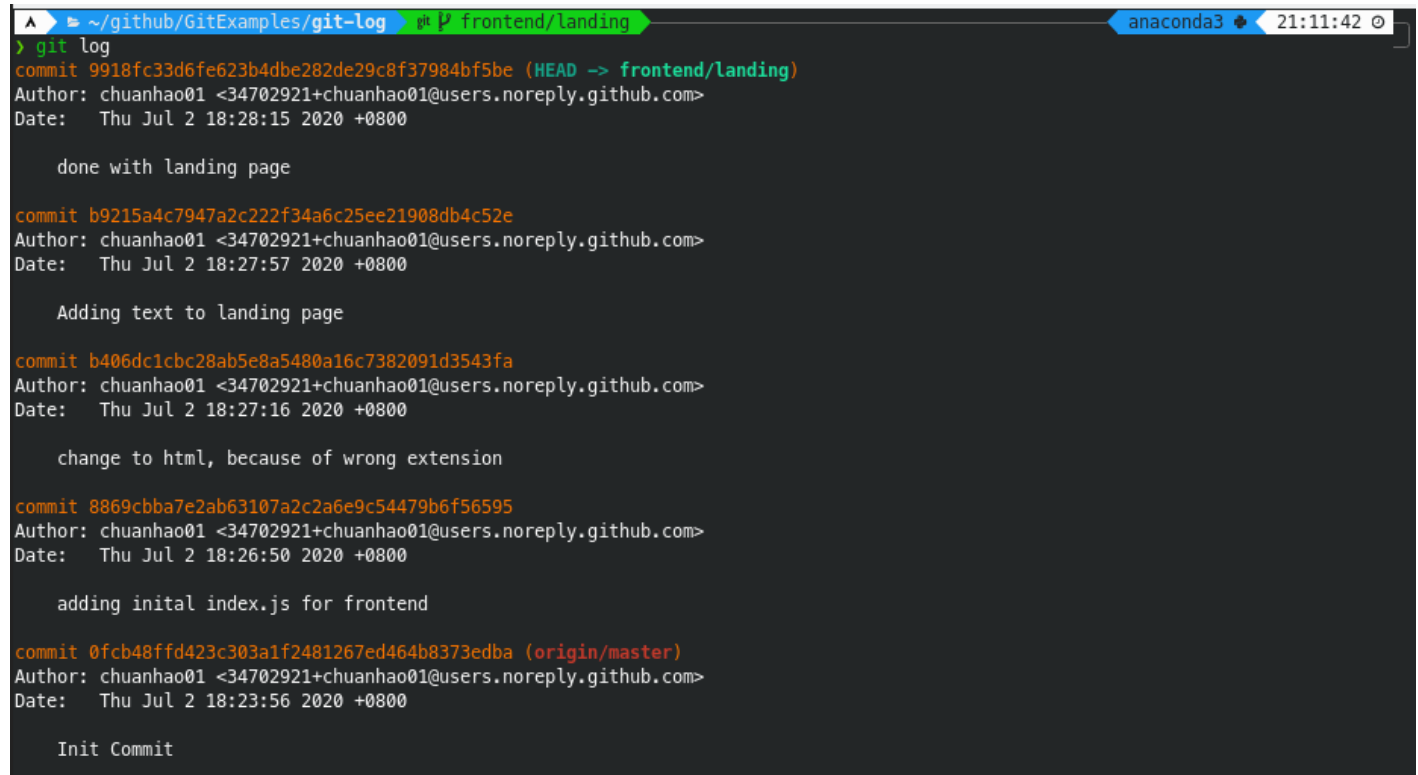
**Note:** Assume you are teammate A

Let's say at this point you have added some code and commits to the project and want to see what commits you had done. This could be because, you wanted to see where you left off yesterday or how features have or have not been done. (The later reason is when you are working on a larger feature)

## Basic git log

Well to look at all the commits made on your current branch you can do:

```
git log
```



```
> git log
commit 9918fc33d6fe623b4dbe282de29c8f37984bf5be (HEAD -> frontend/landing)
Author: chuanhao01 <34702921+chuanhao01@users.noreply.github.com>
Date: Thu Jul 2 18:28:15 2020 +0800

    done with landing page

commit b9215a4c7947a2c222f34a6c25ee21908db4c52e
Author: chuanhao01 <34702921+chuanhao01@users.noreply.github.com>
Date: Thu Jul 2 18:27:57 2020 +0800

    Adding text to landing page

commit b406dc1cbc28ab5e8a5480a16c7382091d3543fa
Author: chuanhao01 <34702921+chuanhao01@users.noreply.github.com>
Date: Thu Jul 2 18:27:16 2020 +0800

    change to html, because of wrong extension

commit 8869cbb7e2ab63107a2c2a6e9c54479b6f56595
Author: chuanhao01 <34702921+chuanhao01@users.noreply.github.com>
Date: Thu Jul 2 18:26:50 2020 +0800

    adding inital index.js for frontend

commit 0fcb48ffd423c303a1f2481267ed464b8373edba (origin/master)
Author: chuanhao01 <34702921+chuanhao01@users.noreply.github.com>
Date: Thu Jul 2 18:23:56 2020 +0800

    Init Commit
```

Figure 1: Basic git log of local branch

From this screenshot you can see that although we can see all the commits linked to our current branch, there is a bit too much information at once.

For example information like when the commit was made or who made it is not really important when trying to look at the bigger picture.

This is also not practical when looking at projects with 10s or 100s of commits, as we would be lost in the information

## Flag oneline

Luckily there are special flags we can use to format the output.

If we add the flag, `--oneline`, git will shorten each commit to oneline, only preserving the important information.

Looking at the man page for this command:

Documentation:

`--oneline`

This is a shorthand for `"--pretty=oneline --abbrev-commit"` used together.

`--abbrev-commit`

Instead of showing the full 40-byte hexadecimal commit object name, show only a partial prefix. Non default

Thus when running the command.

```
git log --oneline
```

And boom, this was what we wanted at the start. Now we can see that we have added the landing page and also changed the extension of the landing page.

```
^ | ~/github/GitExamples/git-log | frontend/landing | anaconda3 | 21:11:45 |
> git log --oneline
9918fc3 (HEAD -> frontend/landing) done with landing page
b9215a4 Adding text to landing page
b406dc1 change to html, because of wrong extension
8869cbb adding initial index.js for frontend
0fcb48f (origin/master) Init Commit
```

Figure 2: Git log with only oneline

But there is something weird here, we can also see that the `origin/master` commit together with the commits of our branch. This is because as we know, branches have to start off (be based off) somewhere. As such, we can make a pretty good guess that this `frontend/landing` was made off the master branch.

Luckily, we don't have to guess as there is a flag which will show us the relationship between these commits.

### Flag graph

To make git log show the relationships between commits, we just have to add the `--graph` flag. This flag shows a 'graph' of the commits, which for us means the relationship the commits have.

Combining this with `--oneline` flag we showed above, makes for a very clean and concise git log.

(I will leave it to the reader to try out what `--graph` flag does on its own)

```
git log --oneline --graph
```

```
^ | ~/github/GitExamples/git-log | frontend/landing | anaconda3 | 21:24:54 |
> git log --oneline --graph
* 9918fc3 (HEAD -> frontend/landing) done with landing page
* b9215a4 Adding text to landing page
* b406dc1 change to html, because of wrong extension
* 8869cbb adding initial index.js for frontend
* 0fcb48f (origin/master) Init Commit
```

Figure 3: Git log with oneline and graph

### Looking at the big picture

Now that you know how to look at commits tagged to your branch, you might also want to see how the entire repository might look like.

Luckily we have a flag to look at the log of the entire repository.

### Flag all

Using the `--all` flag, this will show the git log, including all the commits from every branch you might have in the repository.

We will also be combining this flag with all previous flags to get the best overview of a project. (In my opinion)

```
git log --oneline --graph --all
```

```
^ | ~/github/GitExamples/git-log | frontend/landing | anaconda3 | 17:11:00 |
> git log --oneline --graph --all
* bf27841 (origin/backend/start-server) Started working on server code, app.js
* cde4cd8 adding packagejson and gitignore
* 2b35546 (backend/start-server) added app.js file
| * 198c121 (master) Merging down front end landing page
|/|
| * 9918fc3 (HEAD -> frontend/landing) done with landing page
| * b9215a4 Adding text to landing page
| * b406dc1 change to html, because of wrong extension
| * 8869cbb adding initial index.js for frontend
|/
* 0fcb48f (origin/master) Init Commit
```

Figure 4: Git log with oneline graph and all

Now we can see that our frontend landing page feature has already been merged down to master. We can also see that after `git fetch`(To fetch any changes from github), our teammate B had already made more commits on his feature and our local `backend/start-server` branch is behind.

## Wrapping up

To end off, the `git log` command actually has a lot more features and flags which are a bit more nuanced and situationally. What was shown here are the most common and applicable features and flags which I hope will be usefully to you.

As mentioned above, below are additional resources:

[More features of git log git log documentation](#)

Additional GUI applications that can help you visualise the git log:

Branches: Show All

☒ Show Remote Branches

Graph	Description	Date	Author
	<code>upstream/live</code> Merge pull request #1419...	3 Sep 2019 23:27	Maira Wer
	<code>ardalis/scaling-scenarios</code> <code>origin</code> <code>upda...</code>	3 Sep 2019 21:37	Steve Smi
	<code>upstream/latex</code> Test double escape to re...	3 Sep 2019 20:47	Maira Wer
	<code>upstream/mairaw-patch-1</code> update version...	3 Sep 2019 20:27	Maira Wer
	Updating scaling chapter	3 Sep 2019 20:04	Steve Smi
	<code>upstream/master</code> move article to tutorials...	3 Sep 2019 19:27	Maira Wer
	turn off feedback for a few areas (#14187)	3 Sep 2019 19:26	Maira Wer
	US 1583733 Add missing language IDs - 09 (...)	3 Sep 2019 19:24	Tom Pratt
	Update errorreport-compiler-option.md (#14...	3 Sep 2019 19:19	Youssef Vi
	Update bc30456.md (#14186)	3 Sep 2019 19:14	Youssef Vi
	"instance method call" --> "static method call...	3 Sep 2019 19:11	Ron Petrus
	Converting <code> tag to code block to avoi...	3 Sep 2019 19:09	Mauricio c
	<code>upstream/broken-links</code> fix broken link	3 Sep 2019 19:04	Maira Wer
	initial checkin (#14139)	3 Sep 2019 19:04	Terry Kim
	fix broken link	3 Sep 2019 19:02	Maira Wer
	fix broken link	3 Sep 2019 19:01	Maira Wer
	<code>upstream/hub-page</code> fix links to not redire...	3 Sep 2019 18:45	Maira Wer
	Add platform clarification (#14189)	3 Sep 2019 18:17	Tom Dykst

VSCode extension (Git Graph)

On Github: Actually there is a feature on github that also allows you to visualise the graph of your repository. It is under **Insights** → **Network** in your respective github repository.

