A Beginner's Git and GitHub Tutorial | Udacity

Git and GitHub are two of the coolest technologies around for developers. Git, despite its complexity and rather terse beginnings, is the version control tool of choice for everyone from web designers to kernel developers. And GitHub is the social code-hosting platform used more than any other. On GitHub, you'll find everything from playful, simple experiments to the Linux kernel itself.

But despite this popularity, there's a lot to learn if you want to use these tools properly, and not just be a beginner. Both tools are sophisticated, providing a rich tapestry of functionality. Consequently, they're not for the faint of heart: they can be quite demanding if you want to use them to the fullest.

So if you're just starting out, perhaps coming across from one of the older version control tools, I want to help you make a great start, by giving you a solid working foundation from which you can grow your knowledge over time. To do so, we'll start with Git, learning how to perform the common operations you'll do every day on a local source code repository. Then we'll cover the basics of GitHub, and how to integrate the local Git repository into it, so that others have access to this project as well. Sound good? Then let's get started.

The Fundamentals

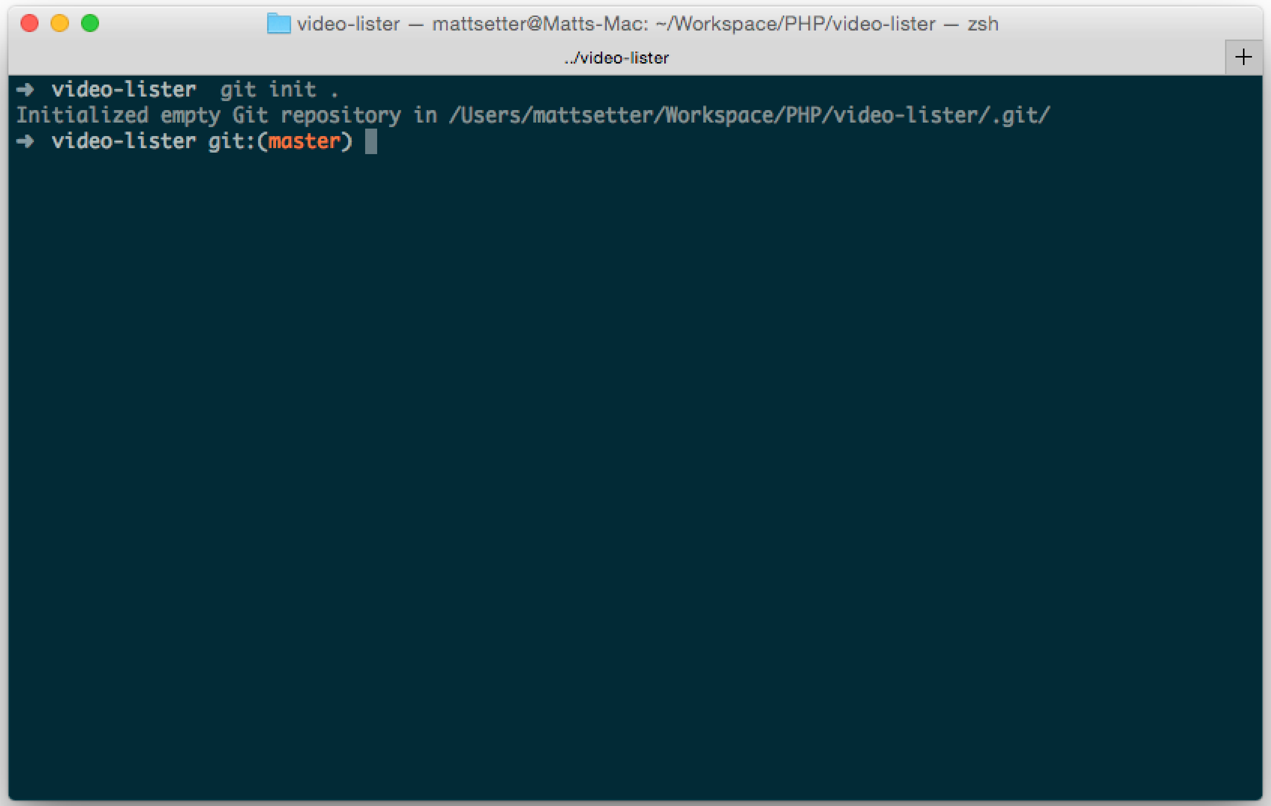
First, you'll need to have a few things installed and available. For starters, I'll assume you already have Git installed. If you don't, grab a copy of the latest version for your operating system. If you're on Linux, you can install it via your package manager instead.

Secondly, you'll need to be at least partly comfortable with using the command line. Now not everyone is, so if you're not, don't worry. This will all be quite straightforward. Nothing too complex. Thirdly, we're going to create a simple repository consisting of a code file and a README. So make sure you have a directory set aside where you can do this.

Then, with everything prepared, let's step through a standard set of actions you'll commonly use on a daily basis. Specifically, we're going to use `init`, `clone`, `add`, `commit`, `diff`, and `log`. There are a number of other, more advanced actions you can perform. But in the beginning, you won't need them.

Initializing a Repository

Before you can work with Git, you have to initialize a project repository, setting it up so that Git will manage it. Open up your terminal, and in your project directory run the command `git init .` as shown in the screenshot below.


video-lister — mattsetter@Matts-Mac: ~/Workspace/PHP/video-lister — zsh
../video-lister

```
➜ video-lister  git init .
Initialized empty Git repository in /Users/mattsetter/Workspace/PHP/video-lister/.git/
➜ video-lister git:(master)
```

A new hidden directory called `.git` will now be present in your project directory. This is where Git stores its database and configuration information, so that it can track your project.

Cloning a Repository

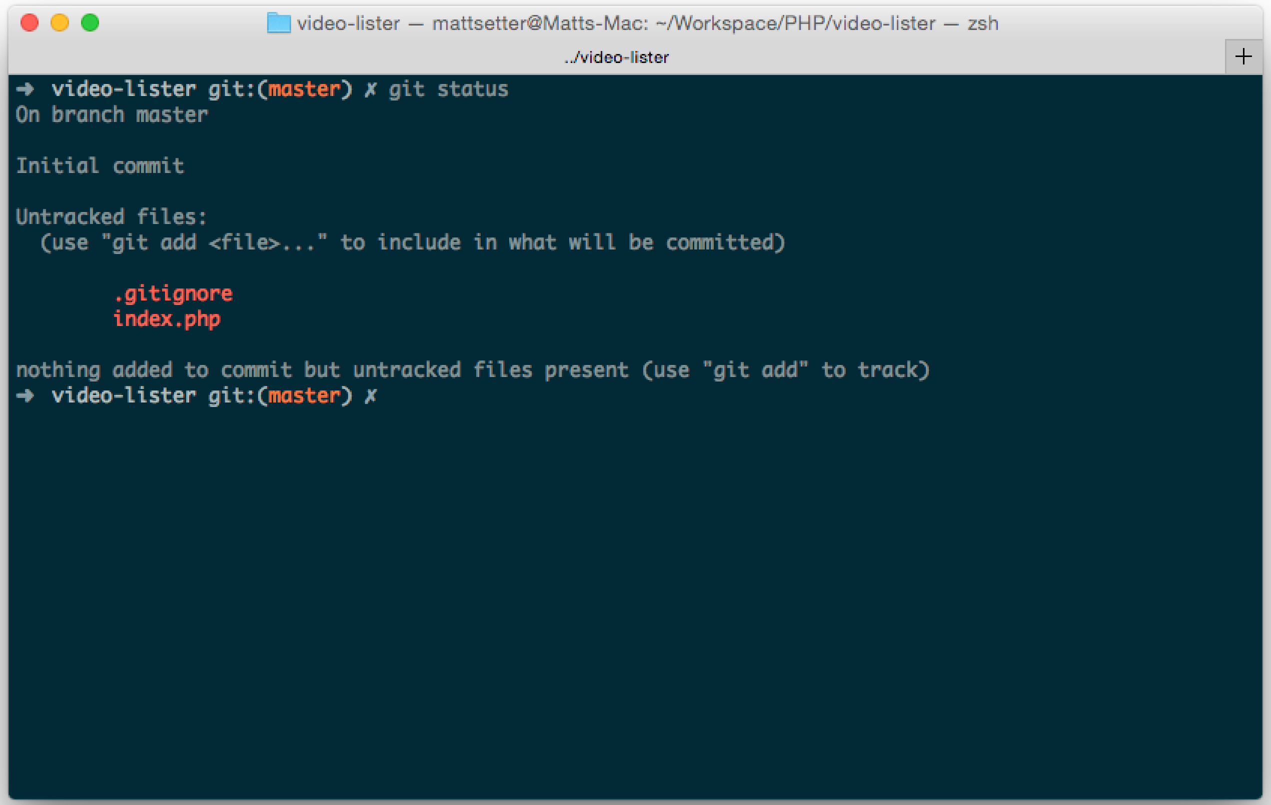
There's another way to access a repository, which is cloning. Similar to checking out a repository in other systems, running `git clone <repository URL>` will pull in a complete copy of the remote repository to your local system. Now you can work away on it, making changes, staging them, committing them, and pushing the changes back.

Adding a New File

I'm primarily a PHP developer, so that's what I'll be using in the sample code for this tutorial. However, if you prefer Python, Ruby, Go, or another language, feel free to substitute your language of choice. Now create a new file, called `index.php`, in your project directory, and in it, add the following code:

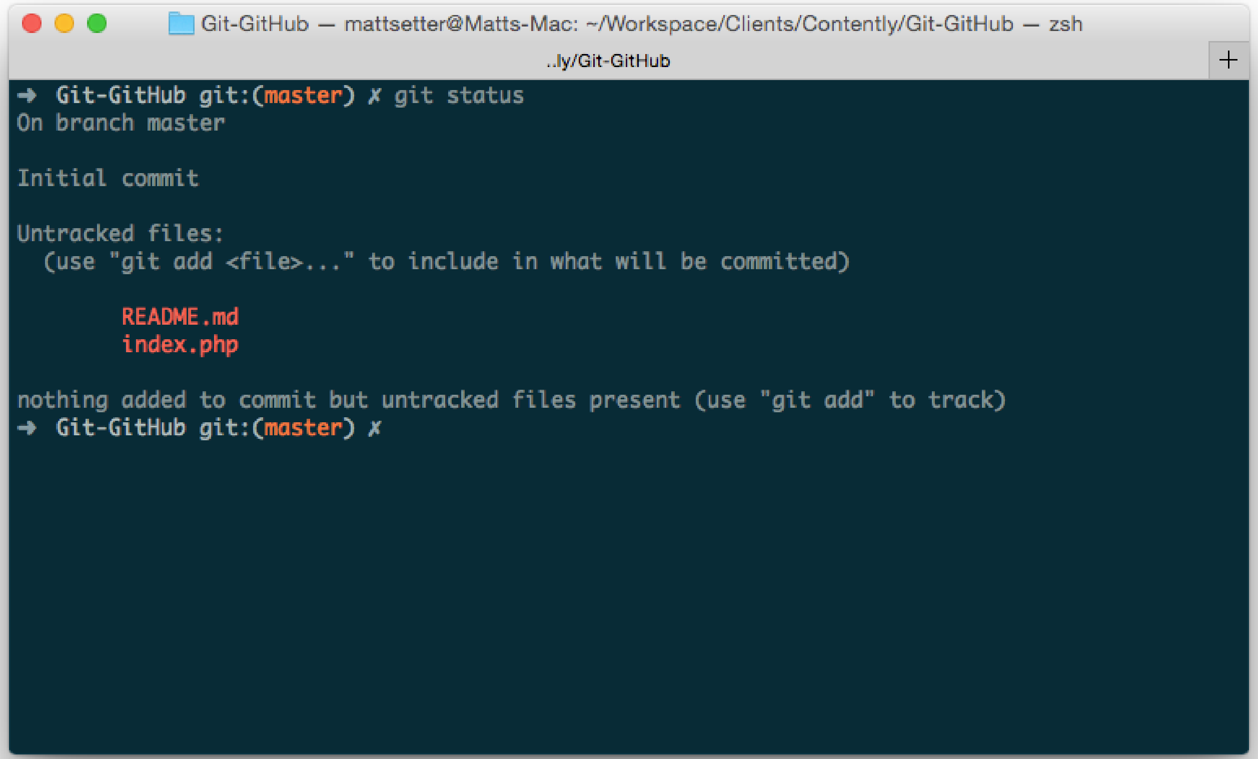
```
<?php print "Hello World";
```

After saving the file, from the terminal run the command `git status`. This will show you the current status of your working repository. It should look similar to the screenshot below, with `index.php` listed as a new, untracked file.


video-lister — mattsetter@Matts-Mac: ~/Workspace/PHP/video-lister — zsh
../video-lister

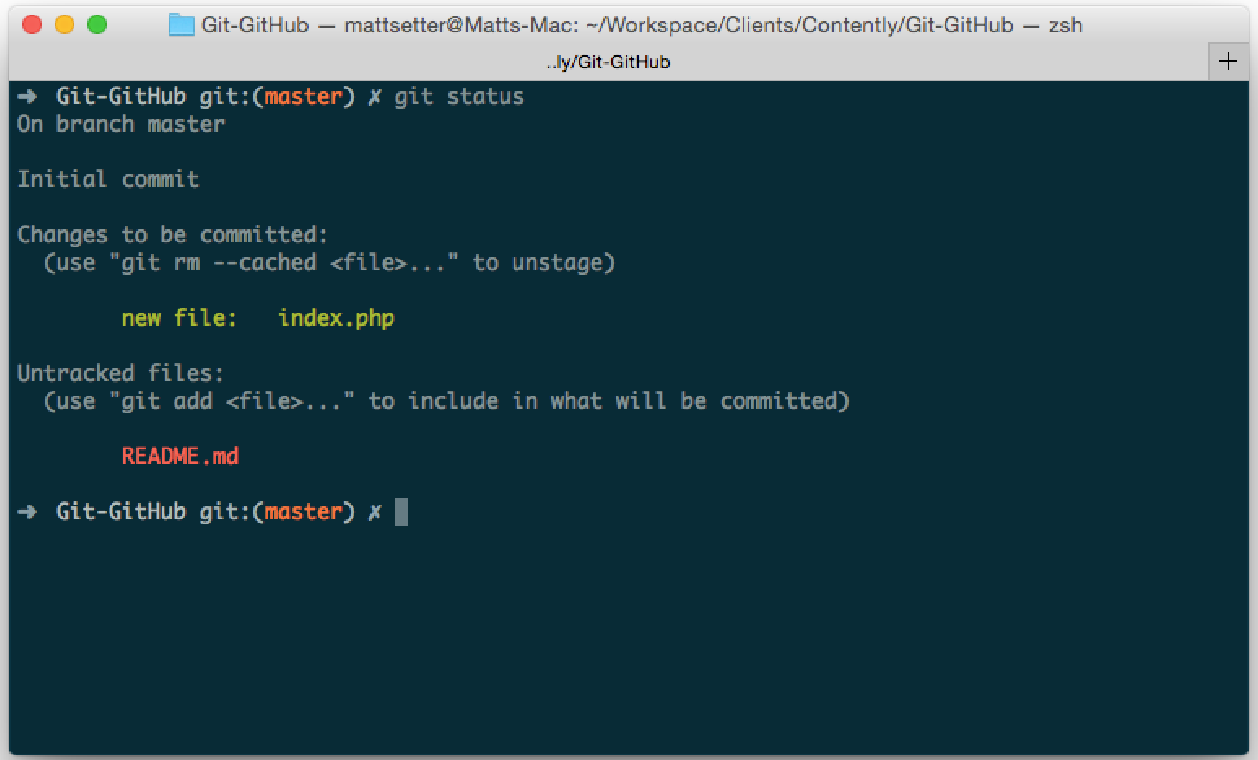
```
➜ video-lister git:(master) ✗ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore
        index.php

nothing added to commit but untracked files present (use "git add" to track)
➜ video-lister git:(master) ✗
```

Now let's see how you can work on multiple files, without having to commit all of them. Create a second file, called `README.md` (every good project has to have one, right?). In that, add a few details, such as the project name, your name, and your email address. Run `git status` again, and you'll now see the two files listed as untracked, as shown below.


Git-GitHub — mattsetter@Matts-Mac: ~/Workspace/Clients/Contently/Git-GitHub — zsh
..ly/Git-GitHub

```
➜ Git-GitHub git:(master) ✗ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md
        index.php

nothing added to commit but untracked files present (use "git add" to track)
➜ Git-GitHub git:(master) ✗
```

Now let's stage `index.php`, because we're not interested in `README.md` just for the moment. To do that, run `git add index.php`. Now run `git status` again, and you'll see `index.php` listed as a new file under "Changes to be committed," and `README.md` left in the "Untracked files" area.


Git-GitHub — mattsetter@Matts-Mac: ~/Workspace/Clients/Contently/Git-GitHub — zsh
..ly/Git-GitHub

```
➜ Git-GitHub git:(master) ✗ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   index.php

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md

➜ Git-GitHub git:(master) ✗
```

Updating Your Git Configuration

Now you're ready to commit `index.php`. But before you do, I want to show you how to configure the editor, which Git will use when you write commit messages. This can be quite helpful, especially if you're not a regular command-line user.

By default, Git uses the program specified in the environment variables `$VISUAL` or `$EDITOR`, which on Linux systems is normally pico, vi, vim, or emacs. If these are new to you, you might want to change it to an application you're more familiar with, perhaps Notepad, TextEdit, or Gedit. To do that, run the following command from your terminal:

```
git config --global core.editor <your app's name>
```

There are a number of other configuration changes you can make, such as your name and email address, what the commit message looks like by default, whether to use colors, and so on. For a complete list, check out the git configuration section of the Git book. For the remainder of this tutorial, I'll be using vim, as it's my editor of choice. But don't feel you have to.

Making the First Commit

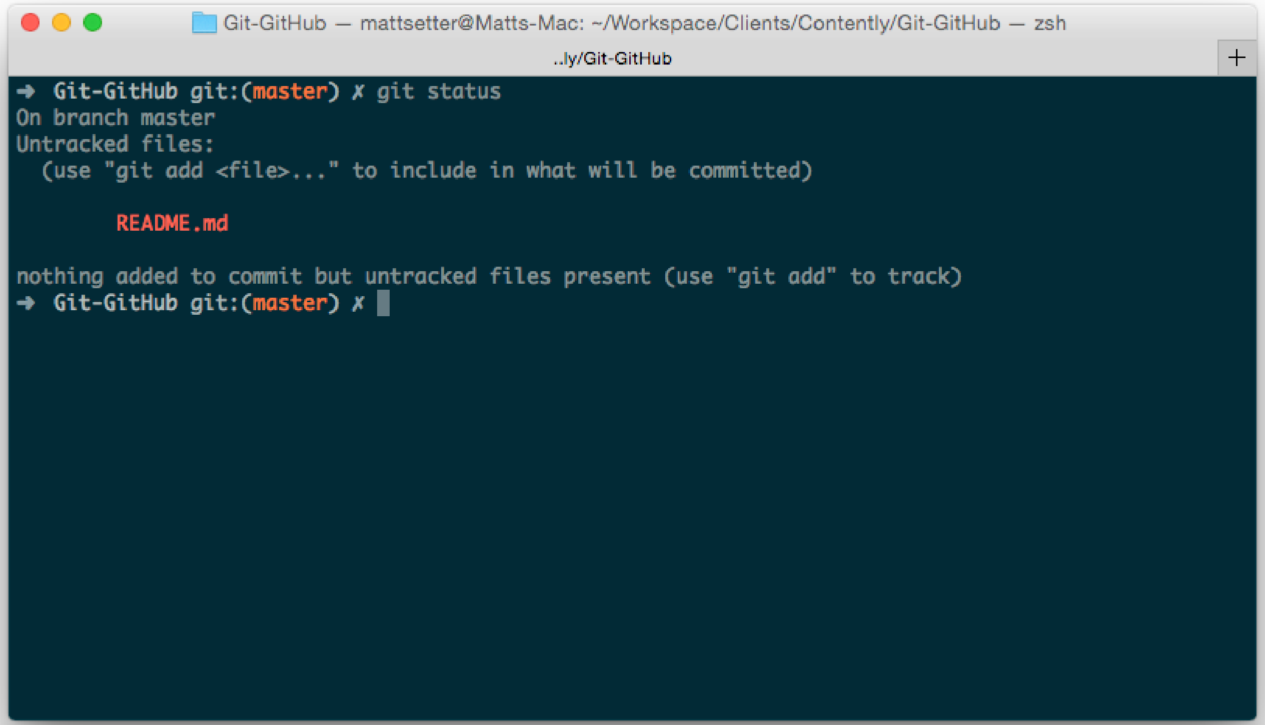
Committing in Git is a lot like committing in other version control systems, such as Subversion. You start the process, add a meaningful commit message to explain why the change was made, and that's it, the file's changed. So run `git commit`. This will automatically open up your editor and display the commit template below.

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   index.php
#
# Untracked files:
#       README.md
#
```

As with the output of `git status`, you see the state of your working repository, which makes it easy to remember what you're committing and what you're not. A good commit message is composed of two parts: a short message, less than 72 characters long, which briefly states (in active voice) the change being made; and a much longer, optional description, which is separated from the brief description by a newline.

In this case, there's no need to write anything too involved, as we're just adding the file to the repository. But if the change you were making involved a complex algorithm, perhaps in response to a bug filed against the code, you'd want to give your fellow developers a good understanding of why you made the change that you did. So add the following simple message "Adding the core script file to the repository," save it, and exit the editor.

Now that the file's committed, run `git status` again, and you'll see that `README.md` is still listed as untracked.


Git-GitHub — mattsetter@Matts-Mac: ~/Workspace/Clients/Contently/Git-GitHub — zsh
..ly/Git-GitHub

```
➜ Git-GitHub git:(master) ✗ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md

nothing added to commit but untracked files present (use "git add" to track)
➜ Git-GitHub git:(master) ✗
```

Seeing Differences

Now that we've got some files under version control and have looked at the basic commands, let's see how to review file changes. I didn't complete the last commit for that reason. To review changes in a file, we use the command `git diffcommand`. Git `diff`, similar to Linux `diff` and other `diff` programs, compares two files and shows the changes the more recent file contains, if any.

Let's have a look at the differences we've staged for `README.md`. To do that, run `git diff README.md`. See something unexpected? I'm guessing you were expecting to see the difference between the most recent version and the staged copy of the file, not the unstaged. That's a little gotcha that might catch you out, at least at first.

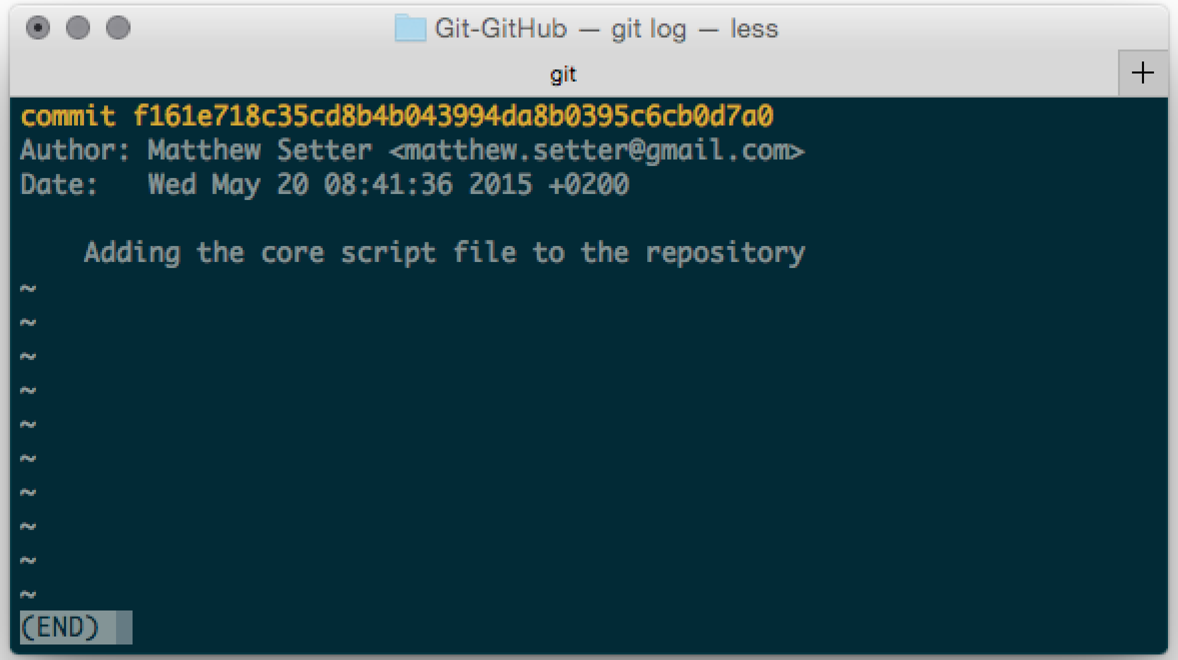
You need to bear in mind whether you're diff'ing a staged or an unstaged file. By default, with no extra arguments, `git diff` will diff the unstaged changes. If you want to see staged changes, then run `git diff --cached README.md`. That command will display something similar to this:

```
diff --git a/README.md b/README.md new file mode 100644 index 0000000..27c0a86 ---
/dev/null +++ b/README.md @@ -0,0 +1,5 @@ +# Simple Git Project + +## Authors +
+Matthew Setter <matthew@maltblue.com>
```

There you see the changes to the file. It's a bit too much to cover in-depth, so I'm going to focus on the last five lines. Before each line you see a plus sign, which indicates an addition. So all of those lines are being added to the file. If we'd removed any content, it would be preceded by a minus. There's quite a number of options which can be passed to `diff`. Have a look at the online reference.

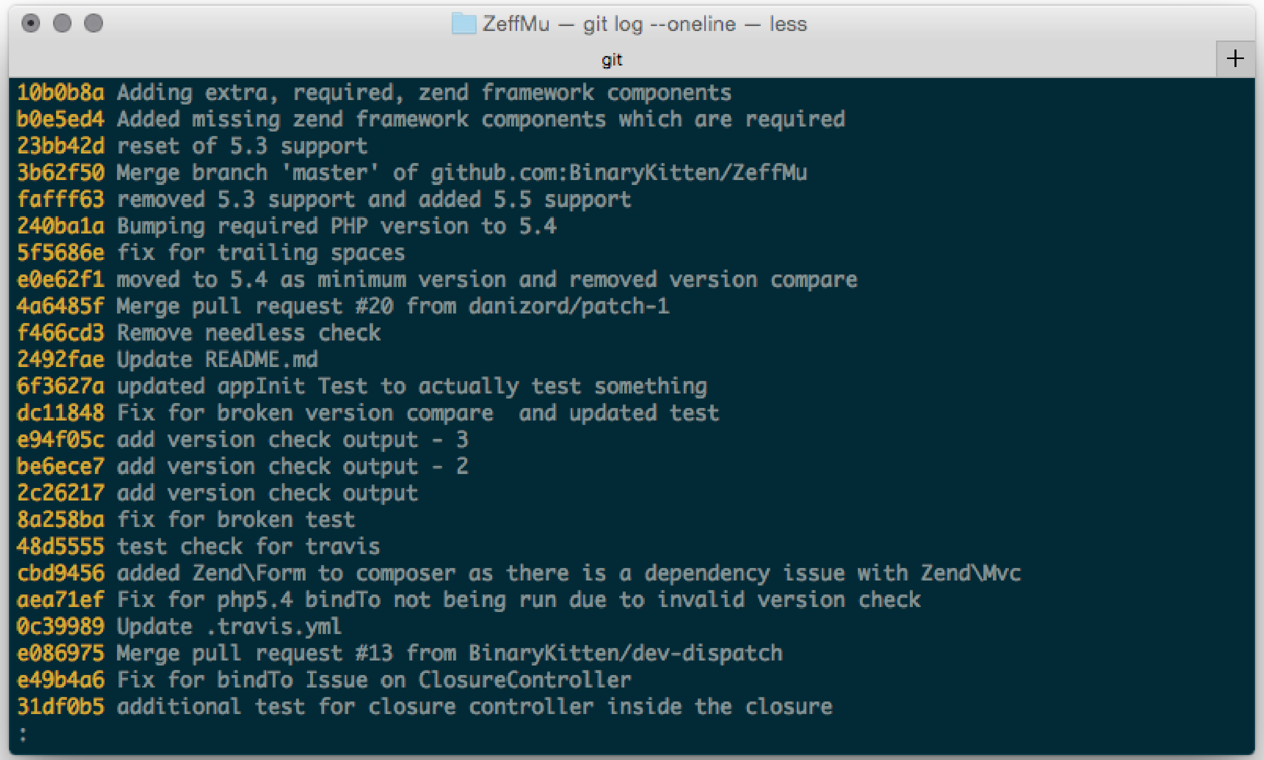
Viewing Change History

Now what if you wanted to see your repository or file history over time? To do that, you need to use git log. Just running `git log` in your project repository will show you a list of changes in reverse chronological order. With no further arguments, you'll see the commit hash, the author name and email, a timestamp for the commit, and the commit message.



```
commit f161e718c35cd8b4b043994da8b0395c6cb0d7a0
Author: Matthew Setter <matthew.setter@gmail.com>
Date:   Wed May 20 08:41:36 2015 +0200

    Adding the core script file to the repository
~
~
~
~
~
~
~
~
~
~
(END)
```

Now this is fine, but what if you want to customize what you see? What if you just want to view the commit hash and the commit message? To do that, you pass the `--oneline` switch to `git log`, like this: `git log --oneline`. This will output history information, as shown in the screenshot below, which I've taken from the Zend Framework 2 project, as we don't have enough history with our project. `--oneline` is a shortcut for `--pretty=oneline`. Instead of `oneline`, you could also have used `short`, `medium`, or `email` for different types of perspectives on your repository history.



```
10b0b8a Adding extra, required, zend framework components
b0e5ed4 Added missing zend framework components which are required
23bb42d reset of 5.3 support
3b62f50 Merge branch 'master' of github.com:BinaryKitten/ZeffMu
fafff63 removed 5.3 support and added 5.5 support
240ba1a Bumping required PHP version to 5.4
5f5686e fix for trailing spaces
e0e62f1 moved to 5.4 as minimum version and removed version compare
4a6485f Merge pull request #20 from danizord/patch-1
f466cd3 Remove needless check
2492fae Update README.md
6f3627a updated appInit Test to actually test something
dc11848 Fix for broken version compare  and updated test
e94f05c add version check output - 3
be6ece7 add version check output - 2
2c26217 add version check output
8a258ba fix for broken test
48d5555 test check for travis
cbd9456 added Zend\Form to composer as there is a dependency issue with Zend\Mvc
aea71ef Fix for php5.4 bindTo not being run due to invalid version check
0c39989 Update .travis.yml
e086975 Merge pull request #13 from BinaryKitten/dev-dispatch
e49b4a6 Fix for bindTo Issue on ClosureController
31df0b5 additional test for closure controller inside the closure
:
```

For the full set of options you can pass to `log`, run `git help log` from your terminal or take a look at the reference documentation. There is a wide variety of options available. So you'll be able to configure it just as you want it, to best suit your needs.

Branching

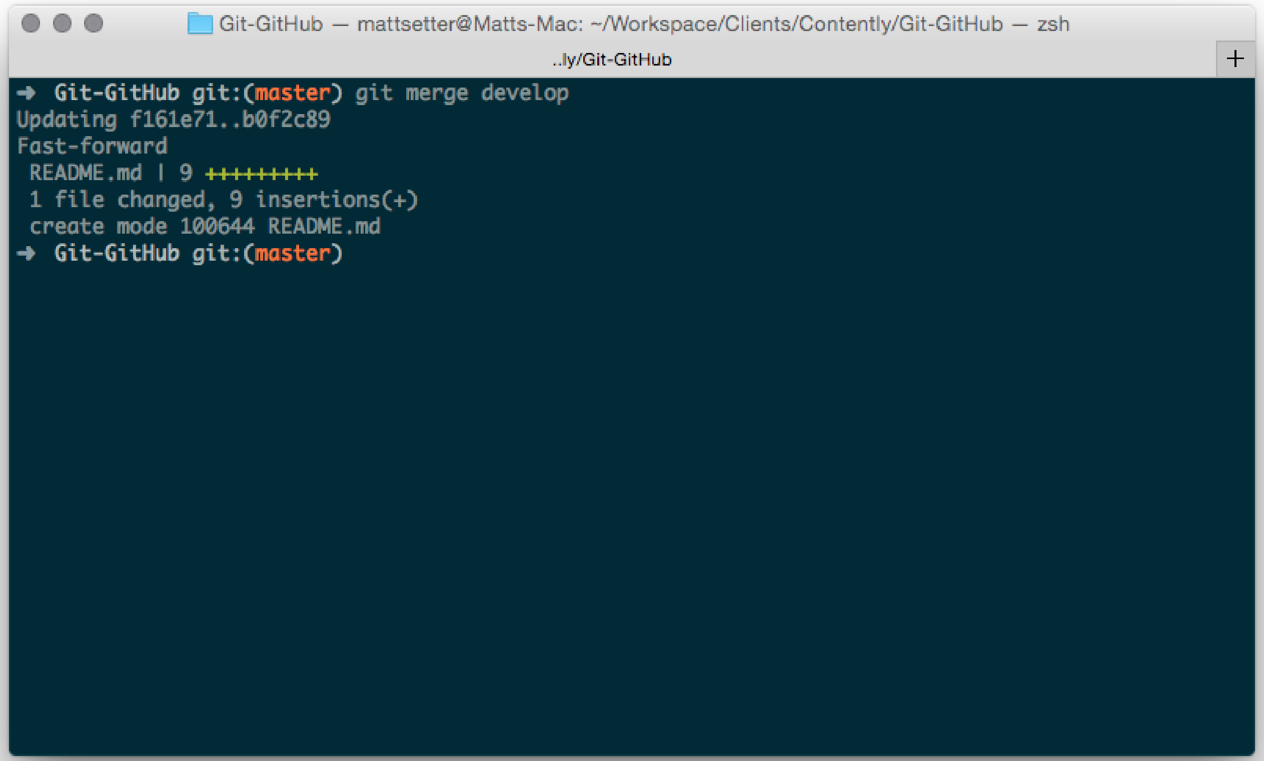
Before we finish up, let's look at branching, one of the fundamental aspects of working with Git. You may not have used branches before. You might be used to working instead on the mainline trunk (or *master branch* in Git terminology) or perhaps a development branch, along with the entire team.

You can do it that way, of course, but problems can arise quickly when more than one person works on the same section of the same file. Branches are essential for being able to safely experiment with concepts and ideas. Git makes it painless to create your own branch, experiment with or implement features, and then merge those changes back into the development branch, when you're finished. Let's see how to do that.

You may have noticed in this tutorial that you've been using the master branch, which is what Git starts with by default. Now we'll create the development branch. From your terminal, run `git checkout -b develop` to create a new branch called `develop`. Running this command will both create and check out the new branch, which at first is simply a copy of the master branch. If you run `git status`, you'll still see the two separate changes to `README.md`. Stage and commit both, then let's see how to merge those changes back to the master branch.

With those two changes staged and committed, you're ready to merge them to the master branch. First, you need to check out the branch you want to merge. To do that, run `git checkout master`. Then you need to merge the changes to the current branch from the branch you've worked on. To do that, run `git merge develop`.

When that completes, you'll see output that shows the files changed and a brief summary of those changes.

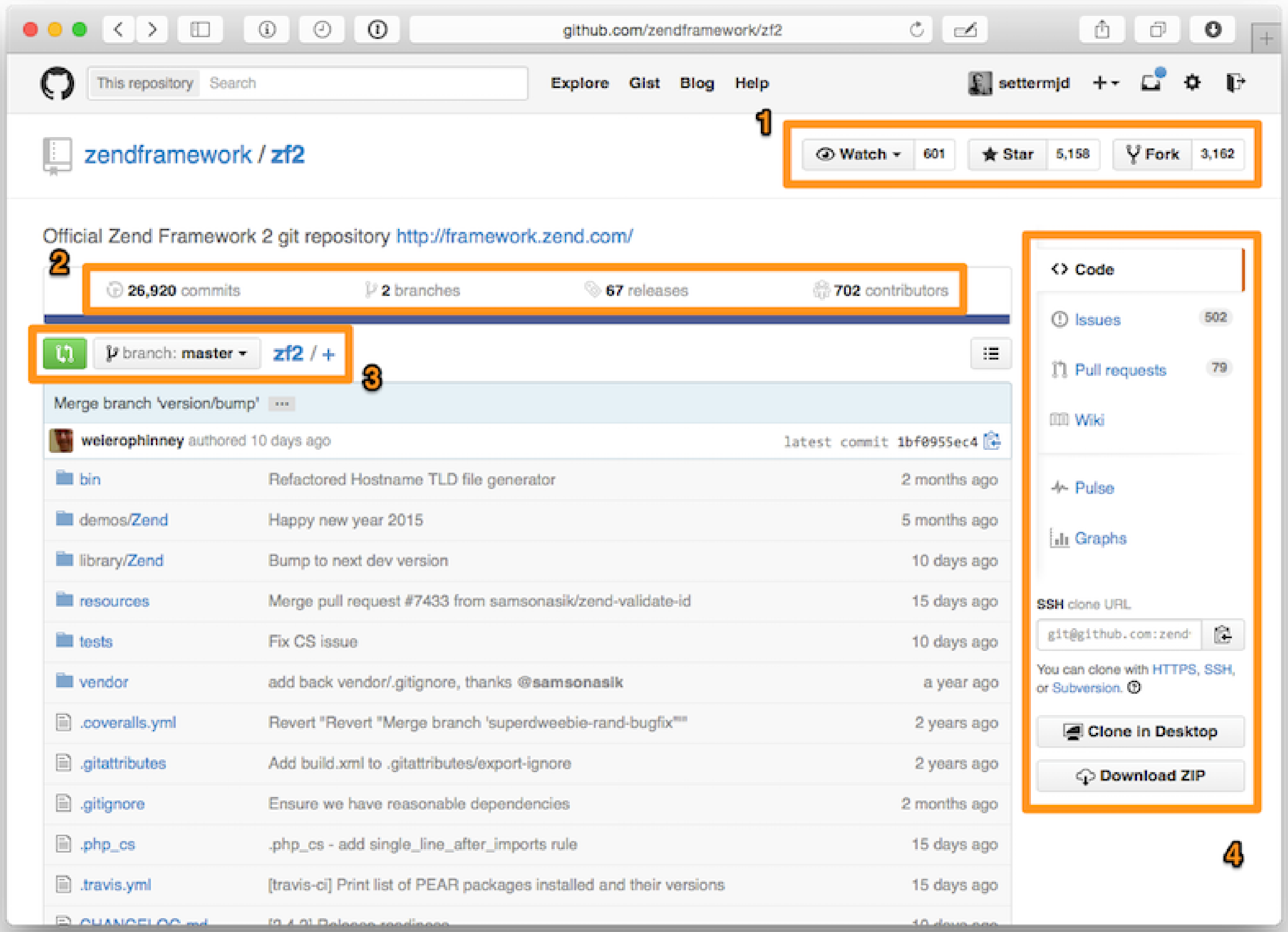

Git-GitHub — mattsetter@Matts-Mac: ~/Workspace/Clients/Contently/Git-GitHub — zsh
..ly/Git-GitHub

```
➜  Git-GitHub git:(master) git merge develop
Updating f161e71..b0f2c89
Fast-forward
 README.md | 9 +++++++++
 1 file changed, 9 insertions(+)
 create mode 100644 README.md
➜  Git-GitHub git:(master)
```

That's how simple it is to branch and merge. Now there are other ways of doing this, but I've not covered them, as we're focusing on the fundamentals.

Using GitHub

Now that you've got a good handle on Git, let's look at GitHub. I'm keen not to overwhelm you, so I've made an annotated screenshot of a GitHub project, so that you can quickly become familiar with the most common features. Yes, GitHub is more than simply a project repository, but that's where you're likely going to spend most of your time on the site.



Official Zend Framework 2 git repository http://framework.zend.com/

26,920 commits 2 branches 67 releases 702 contributors

branch: master zf2 / +

Merge branch 'version/bump'

weierophinney authored 10 days ago latest commit 1bf0955ec4

bin	Refactored Hostname TLD file generator	2 months ago
demos/Zend	Happy new year 2015	5 months ago
library/Zend	Bump to next dev version	10 days ago
resources	Merge pull request #7433 from samsonasik/zend-validate-id	15 days ago
tests	Fix CS issue	10 days ago
vendor	add back vendor/.gitignore, thanks @samsonasik	a year ago
.coveralls.yml	Revert "Revert "Merge branch 'superdweebie-rand-bugfix""	2 years ago
.gitattributes	Add build.xml to .gitattributes/export-ignore	2 years ago
.gitignore	Ensure we have reasonable dependencies	2 months ago
.php_cs	.php_cs - add single_line_after_imports rule	15 days ago
.travis.yml	[travis-ci] Print list of PEAR packages installed and their versions	15 days ago

You can clone with HTTPS, SSH, or Subversion.

What you see is a project homepage. Across the top, in point one, are listed the project name, how many people are watching it, how many people have given it a vote of confidence by starring it, and how many people have forked it, perhaps to make changes of their own to it and contribute to it. Then, in point two, there's the number of commits to the current branch, the number of branches, the number of releases, and the number of contributors. Next, in point three, there's the branch picker, then below that there's a listing of the top-level files in the project, and when the last commit was.

Over on the righthand side, in point four, you have the key navigation options. These are:

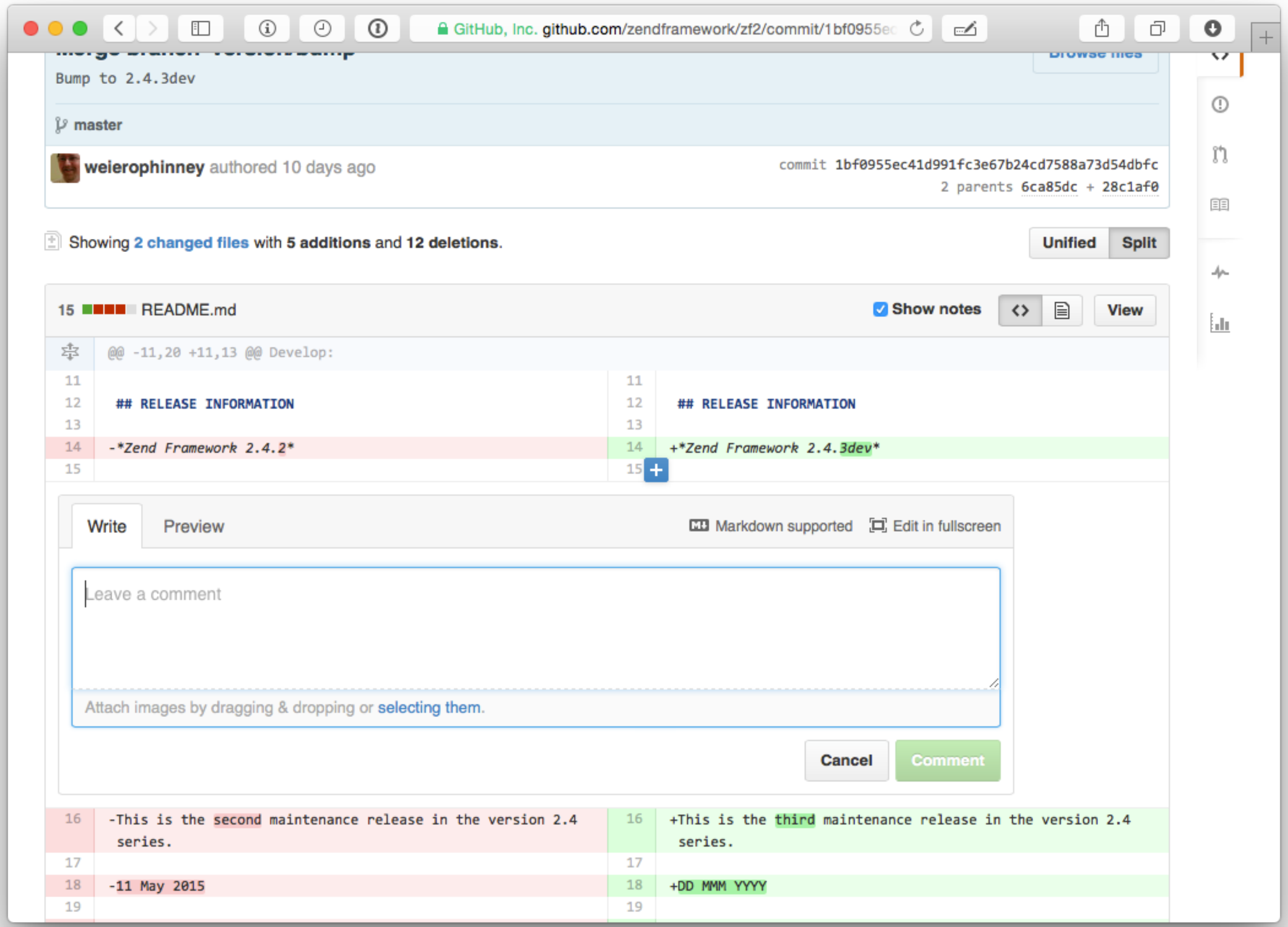
- **Code**: The view you're on by default, showing the files in the project.
- **Issues**: A simple but effective issue tracker, whether you and the team want to report bugs and problems, make requests for new features, or other such tasks.
- **Wiki**: A simple but effective wiki for documenting the project in more detail than a standard `README` file allows.
- **Pulse**: A summary of statistics about the project, including open and closed issues. Here is where you find out how active the project is.
- **Graphs**: A timeline of commits, followed by a breakdown of commits by individual contributor. You can then use the available tabs to look at the project activity in detail, based on a series of key metrics including code frequency, least to most active days for contributions, and so on.

Finally, also on the righthand side, there's the link to the repository URL. If you want to clone this project, this is the

URL you pass to `git clone`.

Now let's look at the commit history, by clicking **commits**. There you see the commits in reverse chronological order. On the left, you see the commit short description, the author's username, and when the commit was made. On the right, the short version of the commit hash, and a link to the commit.

Click the commit hash, so you can see the changes it contains. In this example, we see a side-by-side diff of the project's `README` file and a second file, `library/Zend/Version/Version.php`. You can see on the left what was removed from the previous version, and on the right what was added to this version. Above each commit, on the lefthand side, you see a short summary of the changes, which shows both the total changes (in this case 15) and a visual representation. Now here's some fun.


GitHub, Inc. github.com/zendframework/zf2/commit/1bf0955e

Bump to 2.4.3dev

master

weierophinney authored 10 days ago

commit 1bf0955ec41d991fc3e67b24cd7588a73d54dbfc
2 parents 6ca85dc + 28c1af0

Showing 2 changed files with 5 additions and 12 deletions.

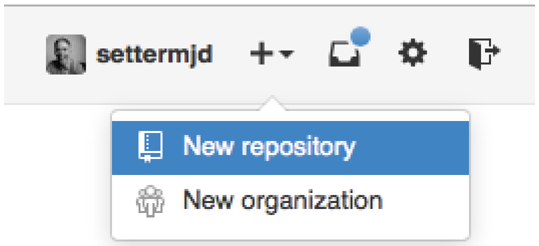
15 README.md

```
@@ -11,20 +11,13 @@ Develop:

 ## RELEASE INFORMATION                          ## RELEASE INFORMATION

-*Zend Framework 2.4.2*                         +*Zend Framework 2.4.3dev*

-This is the second maintenance release in the version 2.4 series.    +This is the third maintenance release in the version 2.4 series.

-11 May 2015                                    +DD MMM YYYY
```

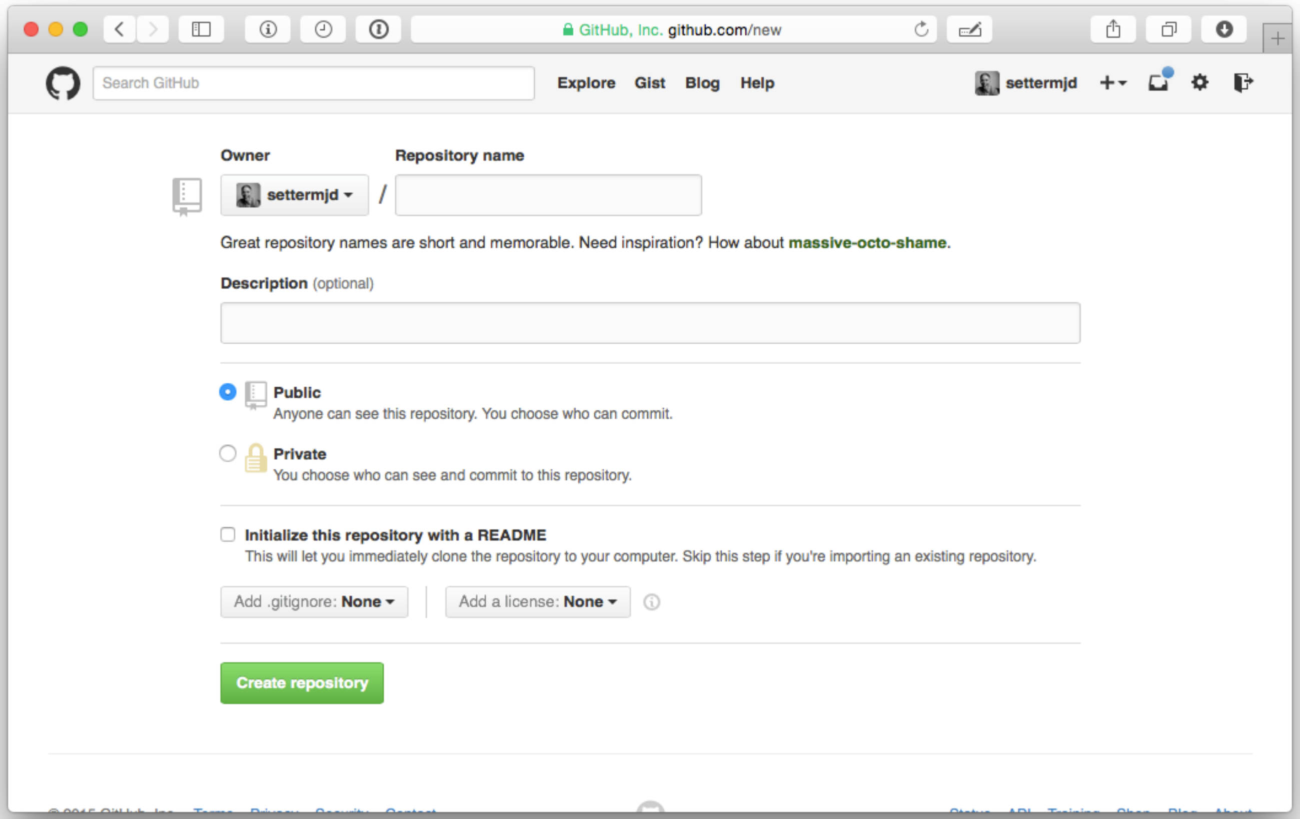
Mouse over either the left or right sides. Notice a blue plus icon appears? If you click on it, you can comment on the code at that point in time. This is an excellent feature, as it makes GitHub a truly collaborative coding experience. One last thing, want to comment on the commit as a whole? There's a comment box at the bottom of every commit. Click through the other tabs, and see what you find in each.

Adding Our Project to GitHub

Now let's get the simple project we've been working on into GitHub. To do that, after you've logged in to your account, click the plus symbol in the upper righthand corner, and click `New repository` from the dropdown. There you'll see the new project creation form.



In the `Repository name` field, add a name. It needn't be special: "first-project" works. Give it a description if you want. Perhaps "My first GitHub project." Then, leave the project with the default of public. That way anyone can find it if they search for it. Finally, click the `Initialize this repository with a README` checkbox, and leave the two select boxes set to `None`. Now click `Create repository`.



Owner: settermjd / Repository name:

Great repository names are short and memorable. Need inspiration? How about **massive-octo-shame**.

Description (optional)

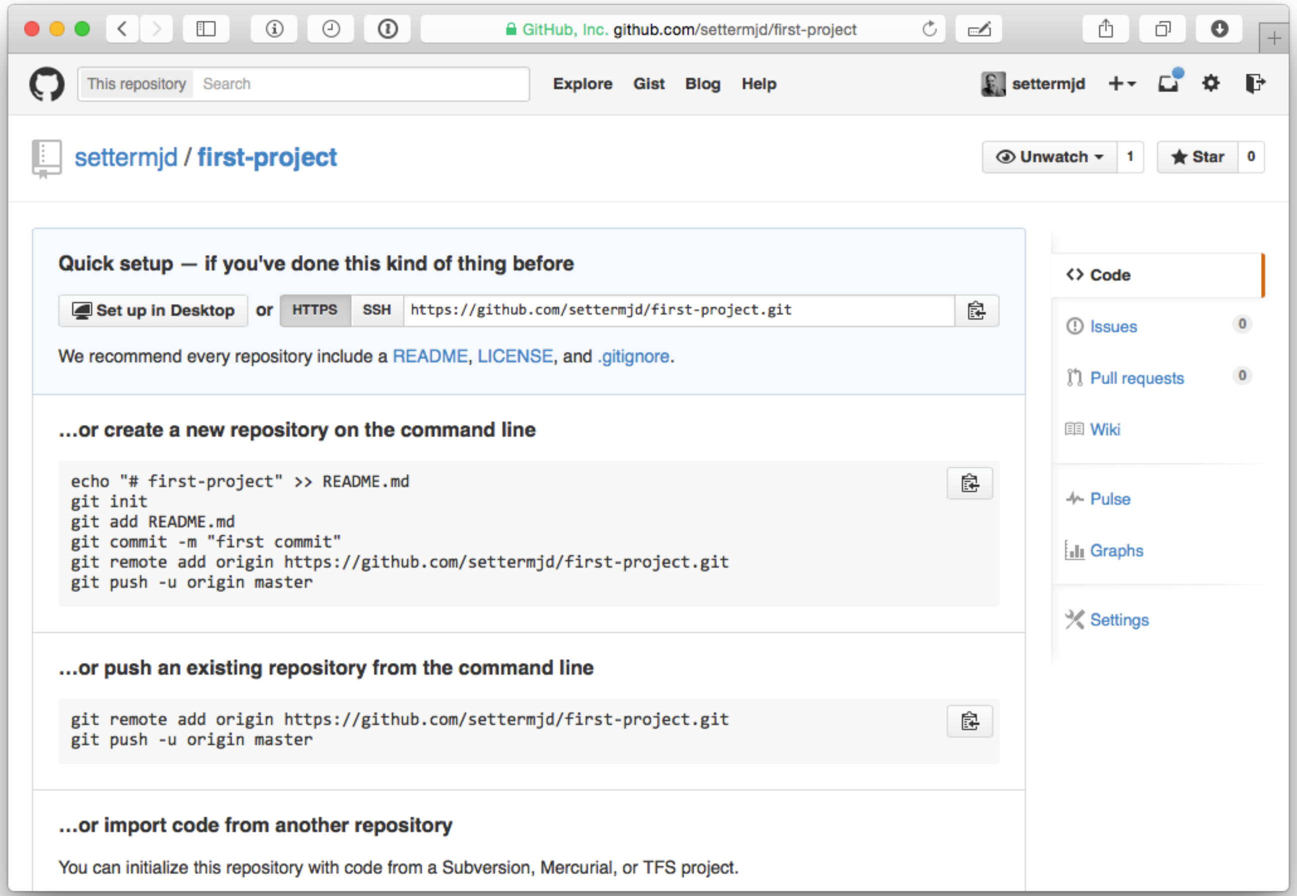
● **Public**
Anyone can see this repository. You choose who can commit.

○ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None**

You'll now be taken to the quick setup page. This page gives you a host of post-setup information about integrating your new GitHub project with your existing local repository, which we'll do, or cloning it fresh. We're going to add GitHub as a remote for our project. To do that, copy the first line under `…or push an existing repository from the command line`, and paste it in the terminal where you've been working up till now.



Quick setup — if you've done this kind of thing before

```
https://github.com/settermjd/first-project.git
```

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# first-project" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/settermjd/first-project.git
git push -u origin master
```

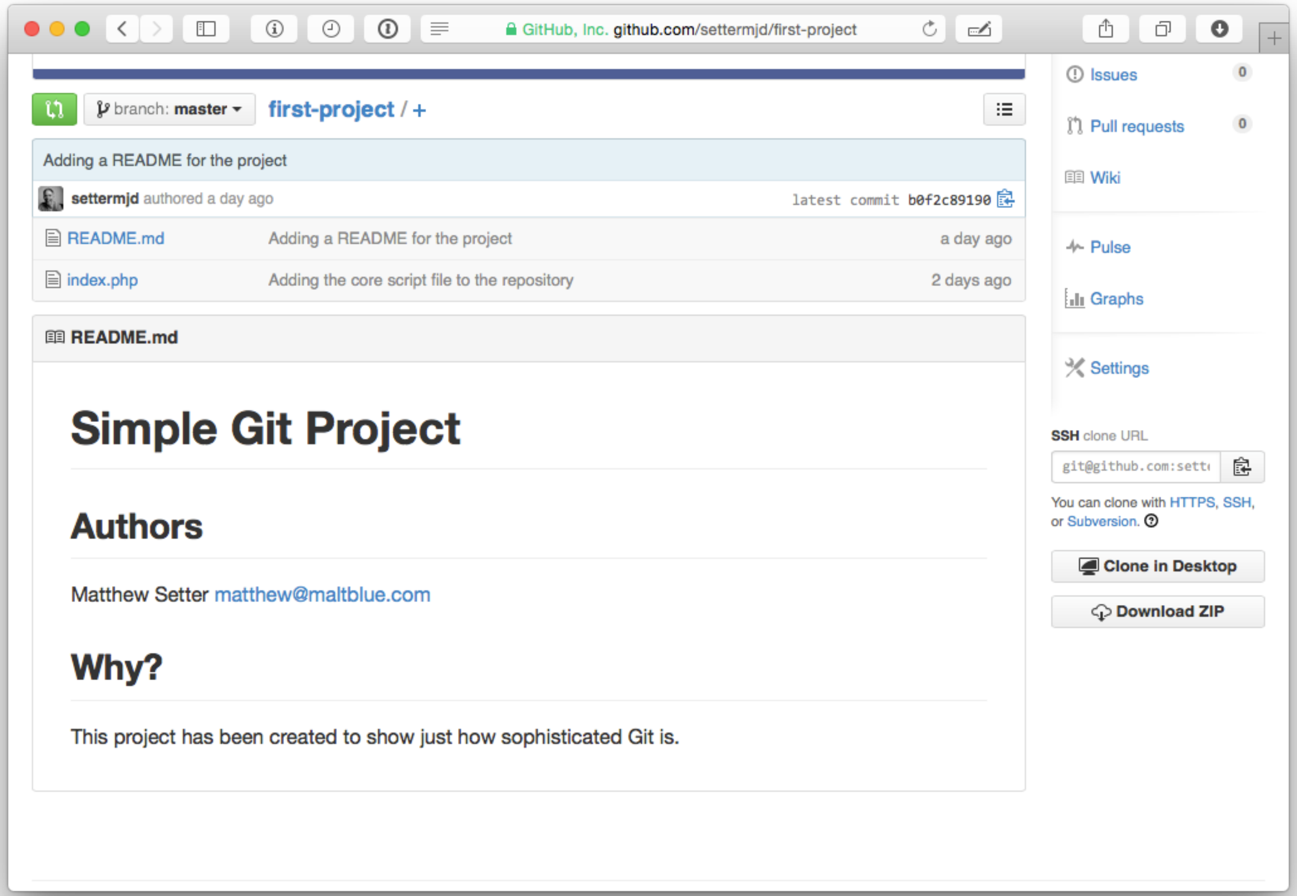
...or push an existing repository from the command line

```
git remote add origin https://github.com/settermjd/first-project.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Doing so won't display any output. Now copy the second line and paste it into your terminal. This will push our changes to GitHub. You'll see output, something like that shown in the screenshot below. Now go back to GitHub in your browser and refresh the page. You'll see our `README.md` and `index.php` displayed in the files list, and the contents of the `README.md` rendered at the bottom of the page.



Adding a README for the project

settermjd authored a day ago latest commit b0f2c89190

README.md	Adding a README for the project	a day ago
index.php	Adding the core script file to the repository	2 days ago

README.md

Simple Git Project

Authors

Matthew Setter matthew@maltblue.com

Why?

This project has been created to show just how sophisticated Git is.

The Bottom Line

And that's the basics of how to work with Git and GitHub. I hope you've seen that, whilst there are quite a lot of concepts to take in, once you've gotten a handle on the basics, you can pick it up reasonably quickly, if with some patience.

There's so much more to cover than there's the opportunity to do here. But one thing about both Git and GitHub—the community behind them is extremely supportive and generous in the sheer volume of information contributed. You'll be in good hands.

For more learning, try the Udacity How to Use Git and GitHub course for free, or look through the selection of links in the further reading section. You'll find just about all you need to become a Git and GitHub master in next to no time.