Using GitHub a learn.sparkfun.com tutorial

Available online at: http://sfe.io/t11

Contents

What's a Repo?

Repo is short for **repository**. Think of a repo as a folder of files and all the changes made to the files are recorded. If there's ever a problem with a file you can go back in time to figure out what changes you made. The most common use for repos are for managing large code projects but repo tracking is good for a variety of applications in the hardware world including PCB layouts, firmware, datasheets and documentation.

For example, let us imagine someone has created an Arduino sketch to demonstrate how to read an analog sensor.

```
language:c
byte myValue = 0;
myValue = analogRead(A0);
```

There's a couple improvements that could be made to this code (analogRead returns an int not a byte!). If the code was just a file on someone's website you'd have to send them an email and suggest the improvements. This is a bit tedious, and when a project gets longer than a few lines of code, email is not a viable way to collaborate on projects. GitHub allows one person to manage their own projects (also called revision or version control) and it also allows lots of people to work together on large projects (source code management).

What is this git thing?

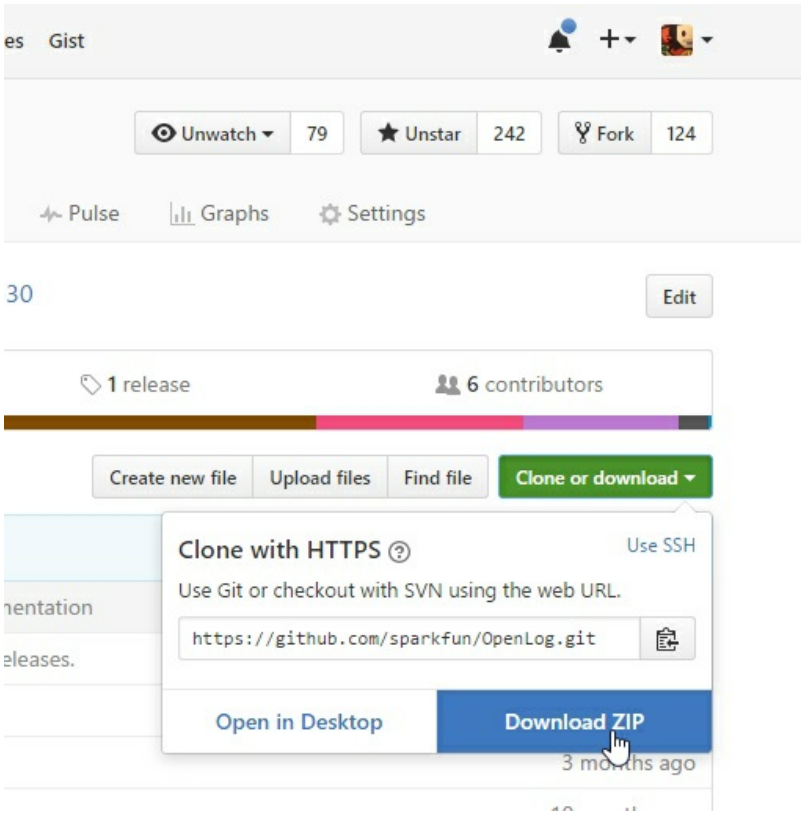
Git is a software management tool designed for extremely large coding projects (such as Linux). Because the majority of work that we do at SparkFun is on smaller projects, we use only a fraction of its capabilities. While Git uses a command line interface, GitHub was created to give Git a slicker looking web interface. Furthermore, GitHub released a GitHub Desktop GUI for Windows (graphical user interface) that makes moving repos around even easier.

We're going to cover a few things in this tutorial:

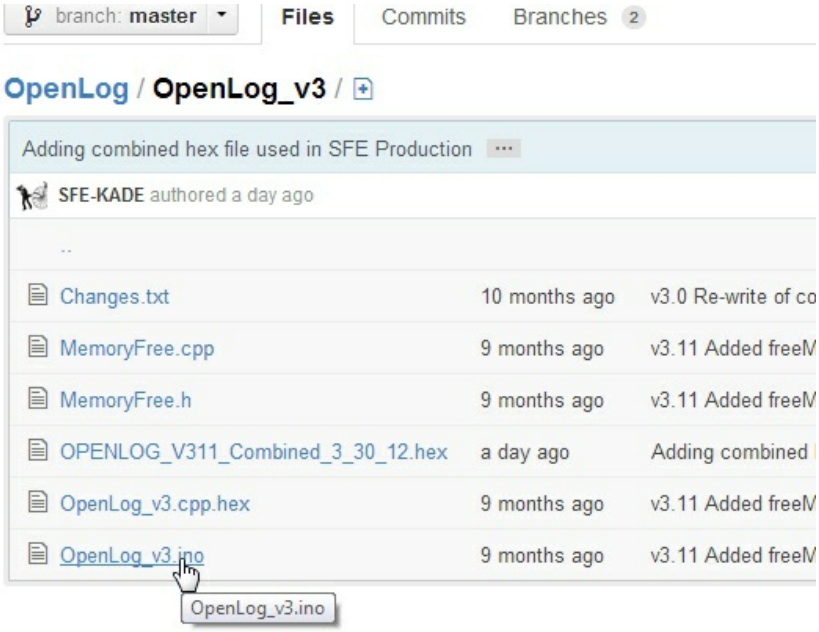
- **Download ZIP** - How to get something from GitHub
- **Manage** - How to manage your own stuff on GitHub
- **Pull Requests** - How to improve something on GitHub
- **Wiki and Issues** - There is lots more on GitHub including Wikis and Issue Tracking

Download ZIP

Just need to get some code from a public GitHub project? Here's how to get something from GitHub:



On every project there's an easy to use 'Download ZIP' button that will give you the entire contents of the project. This is useful if you just need to grab and go (you leech you). However, this is not the correct way if you plan to contribute back.

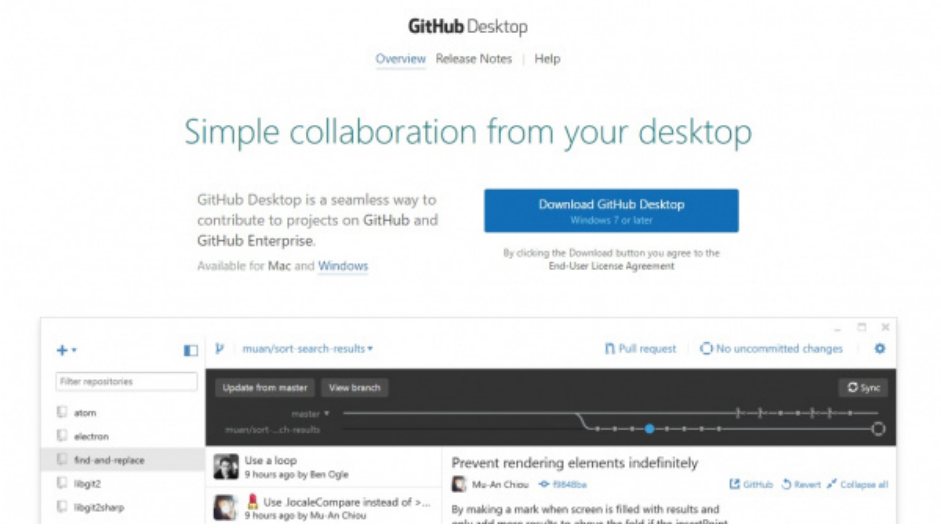


Right-clicking won't work

Note: If you're navigating around a project and see a file you'd like to grab, right clicking and selecting save-as will not get you the file. You will get an HTML file instead of the raw file you might be expecting. You should either use the ZIP download button or clone the repo to a local folder. Keep reading! We'll show you how.

Managing Repos

To start, you'll need to create an account on GitHub. Don't worry, it's free for regular users.

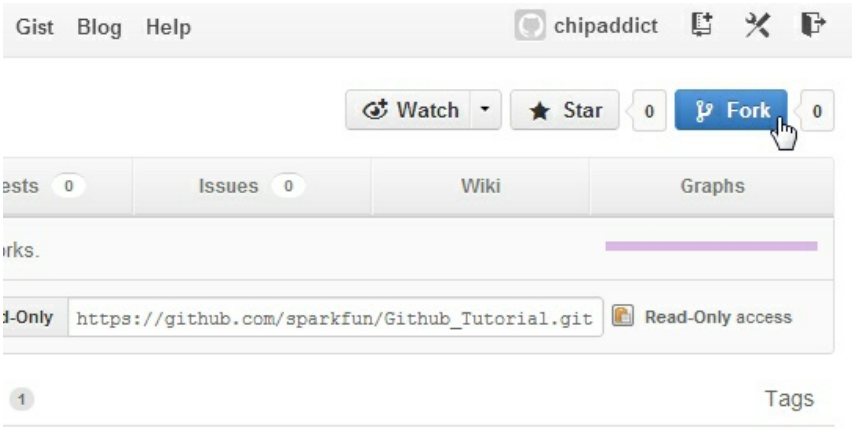


There are plenty of Git clients for Mac and Linux but this tutorial will focus on Windows. If you're a Windows or Mac user I highly recommend you try out the GitHub Desktop GUI. The following tutorial will focus on this client so please download and install the software.

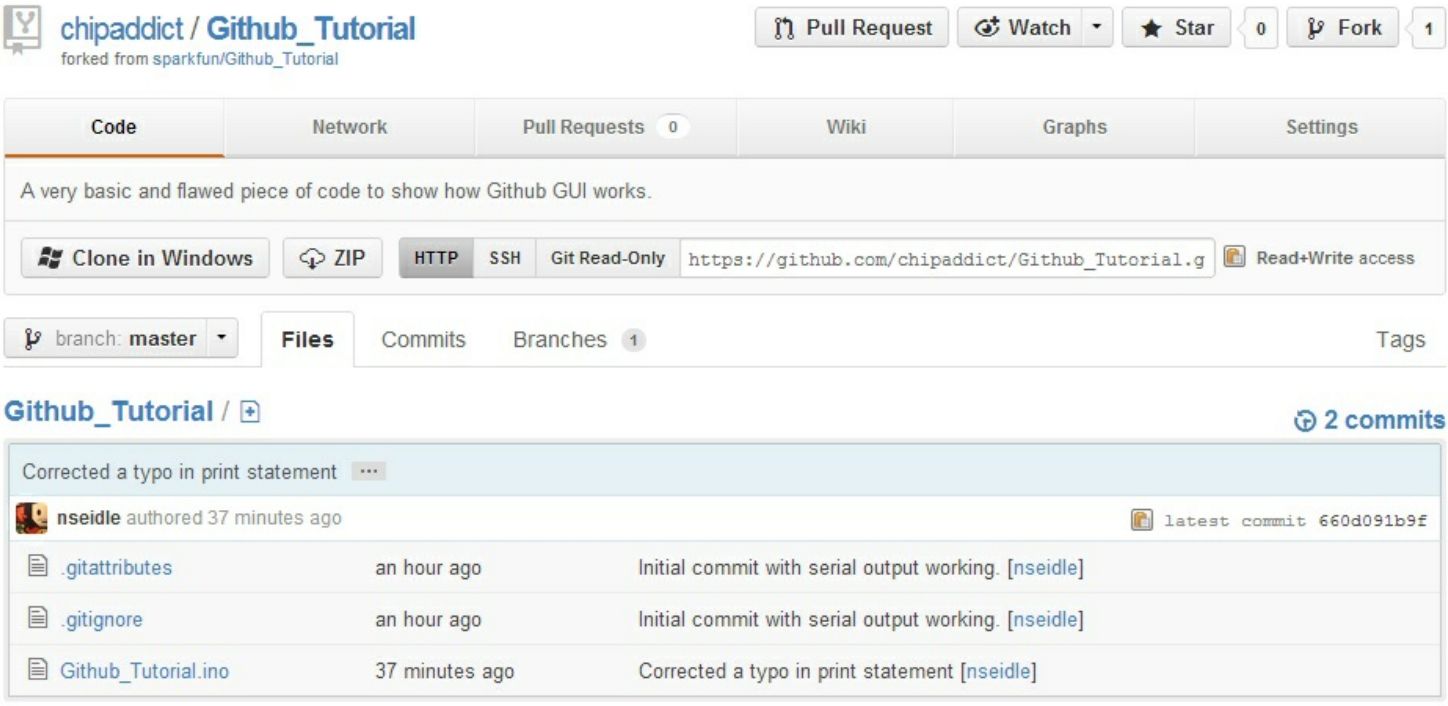
During installation, it will ask you for things such as your login information, name and email address. All of this information will be associated with the commits you make.

During installation, it may ask you to scan for repos on your computer. I recommend you *don't* do this. The scan can take a long time and because you're reading this tutorial, you probably won't have any repos.

Once the GUI is launched you'll probably not have any local repos. Let's go get one from the SparkFun GitHub account. Let's grab the GitHub Tutorial repo. Once you are logged in to GitHub, click on the 'Fork' button.

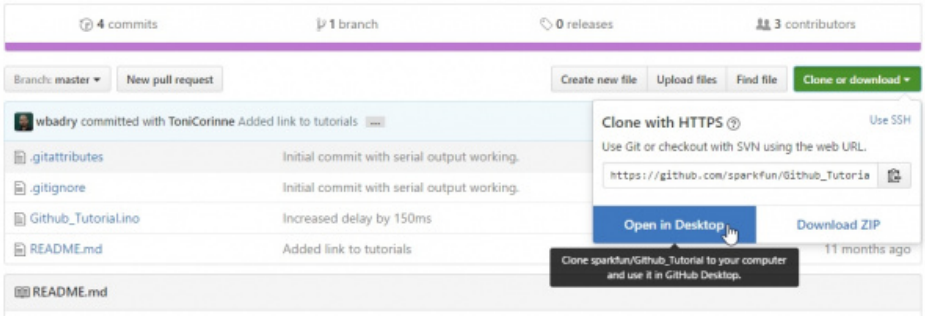


We have now created a 'fork' or a copy of this repo and is located within your GitHub account. Note the words in the upper left corner of the window "**chipaddict/Github_Tutorial**" and the words underneath "**forked from sparkfun/Github_Tutorial**". This shows that you have this project on your account (your account name will be different).



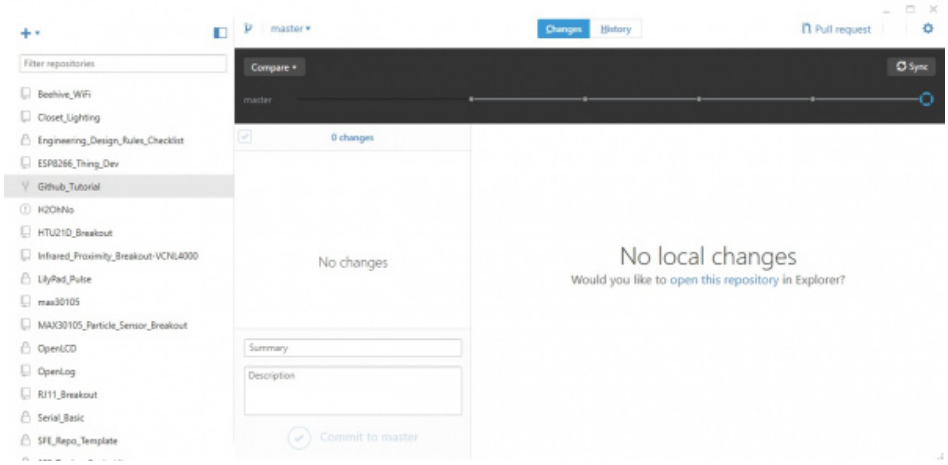
Now you can make lots of changes to this repo without affecting the original project. This is helpful if someone has some example code that is close to what you need but needs lots of modifications for your plans.

Now that you have your own copy of the project online, click on the ‘Clone or Download' button then the 'Open in Desktop' button. If you are not logged in, it may take you to the GitHub Windows page.

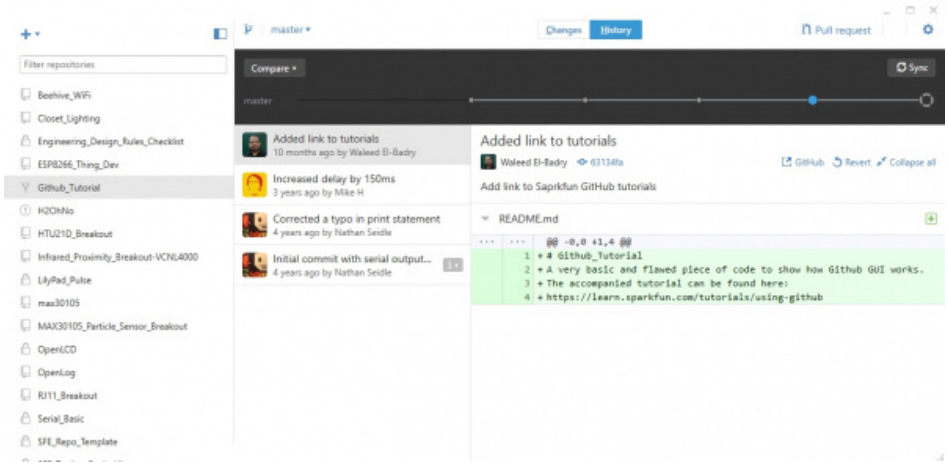


Windows may ask you for permission to allow the link to launch and use the GitHub software. This is ok. The GitHub GUI will open and a download will begin.

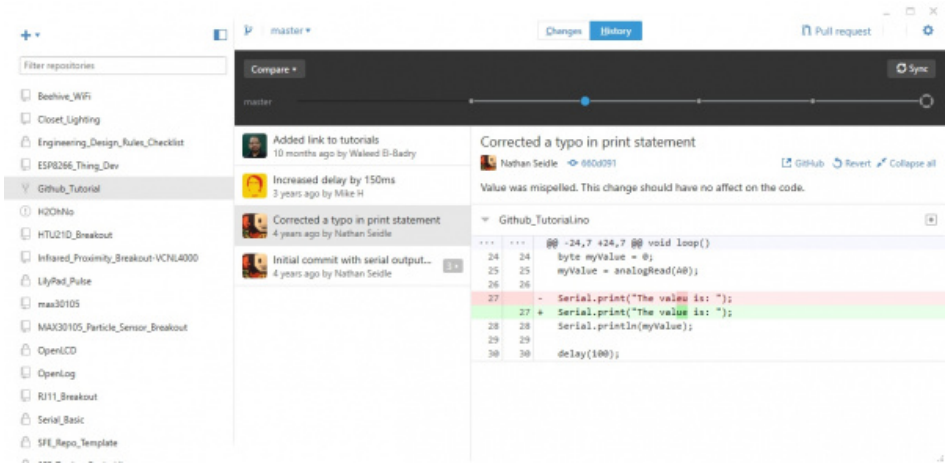
Because we just forked this repo there are no changes. Previous versions of the GitHub Desktop GUI had a timeline dot. If you click on any of the dots at the top we can go back in time and see what's changed over time. Otherwise, there will be a tab on the left hand side listing the changes and history. You can see the file that was changed and what was added (in green) to the file.



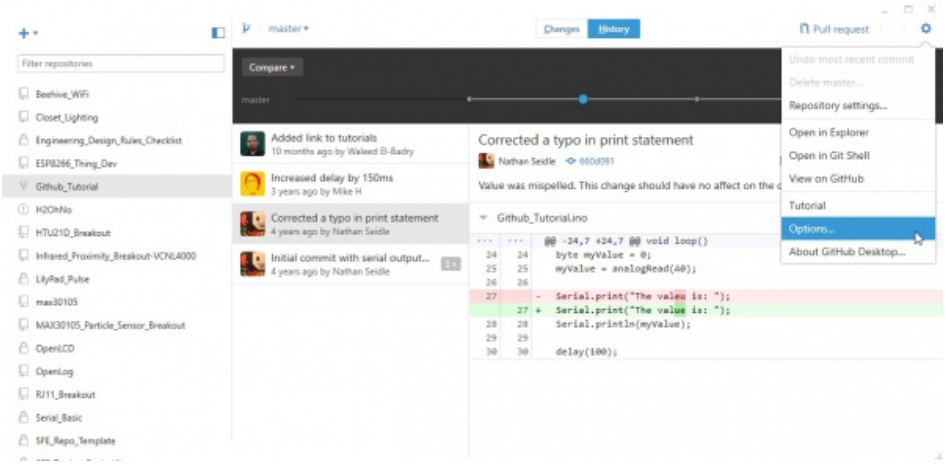
Hey thanks Waleed! He suggested a change to the README and added a link back to this tutorial. Smart.



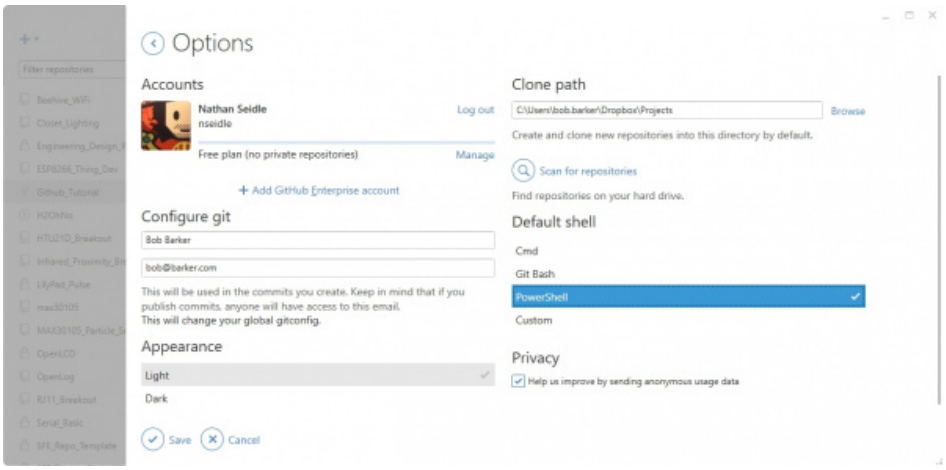
Even further back in time we can see there was a comment **Corrected a typo in print statement**' and below that we can see the lines of code that were altered - red was removed, green was added. We can do lots of fun stuff like 'revert commit' and 'roll back this commit' but for now, let's see how revision tracking works.



For older versions, click on the gear in the upper right corner and selection **Options...**'. For the latest version, this will be located '**File** > **Clone repository...**' .



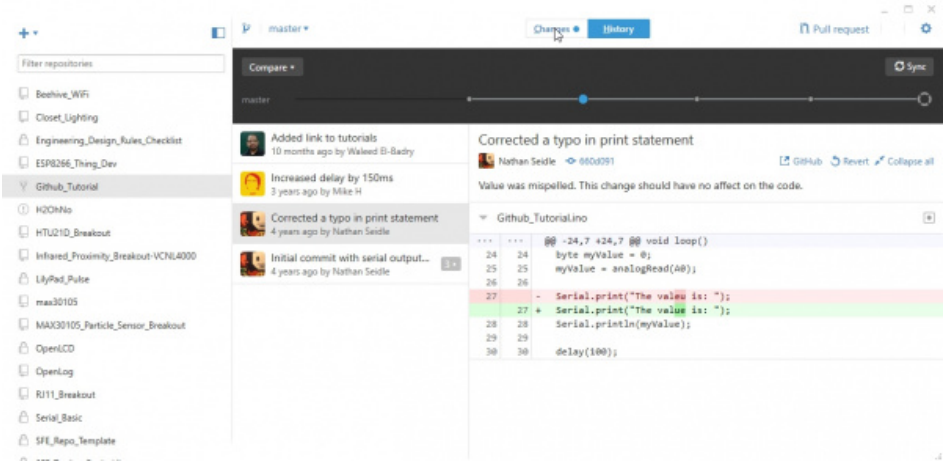
You will be able to find where your repositories are being stored under **Clone path**'. I changed this to store my repositories in a Dropbox folder. I use Dropbox liberally in conjunction with GitHub so that I can work on projects across devices and then push code up to GitHub once the project has reached a level of stability.



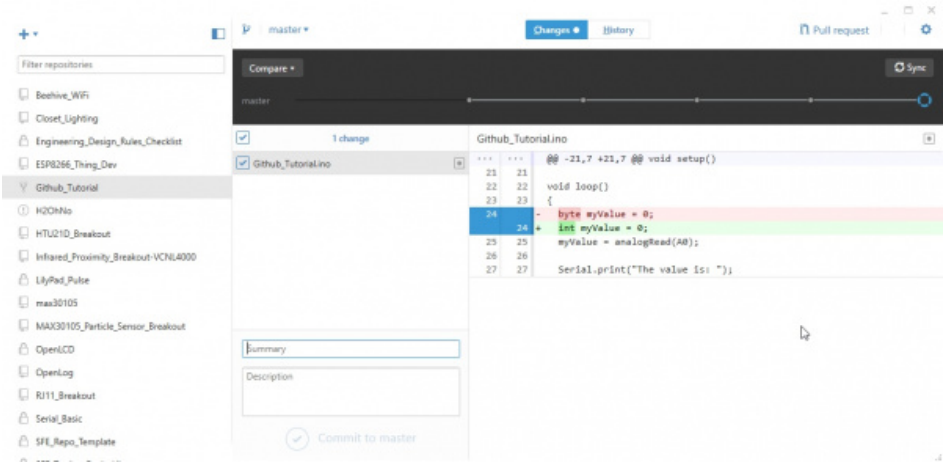
You should now know where your repos are stored. Navigate to that directory and open up the *Github_Tutorial.ino* file. From the Arduino IDE or Windows Notepad let's correct the variable declaration from **byte** to an **int**. Save the modification and return to the GitHub GUI.

```
void loop()
{
  byte myValue = 0;
  myValue = analogRead(A0);

  Serial.print("The value is: ");
  Serial.println(myValue);

  delay(100);
}
```

GitHub has noticed that a file has changed! For older versions, you should see a dot on the changes button. Otherwise, you will noticed that there are items in the **Changes** tab.



You will see the main screen change as well showing the changes that git has detected:

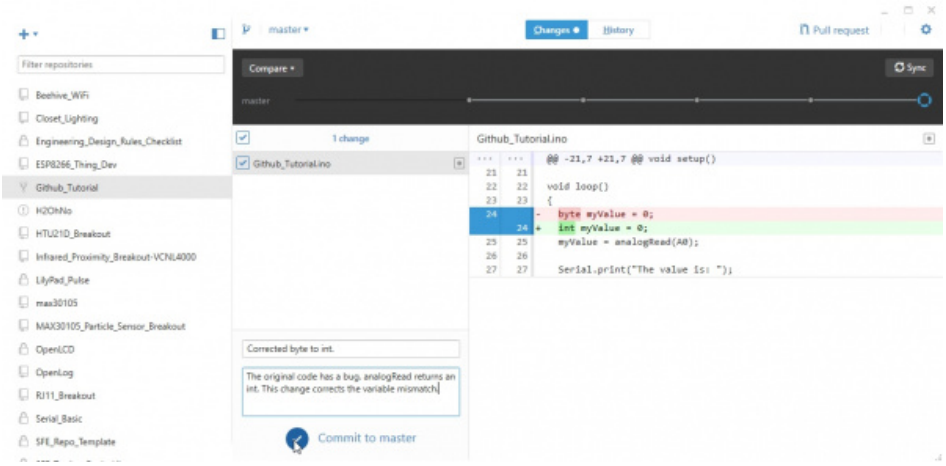


Wow! The small change we made is now nicely highlighted.

Now let's talk about how repos work. You have a**local working copy**, a **local repo**, and a **global repo**.

Local Working Copy

Local working copy: You generally write code, layout PCBs, and hack on documentation on your local computer on a local copy. Throughout the day you would use the GitHub window to 'commit' these changes to the a local repo. The changes you've made throughout the day are *not* known to the world, only to your local repo on your local computer.



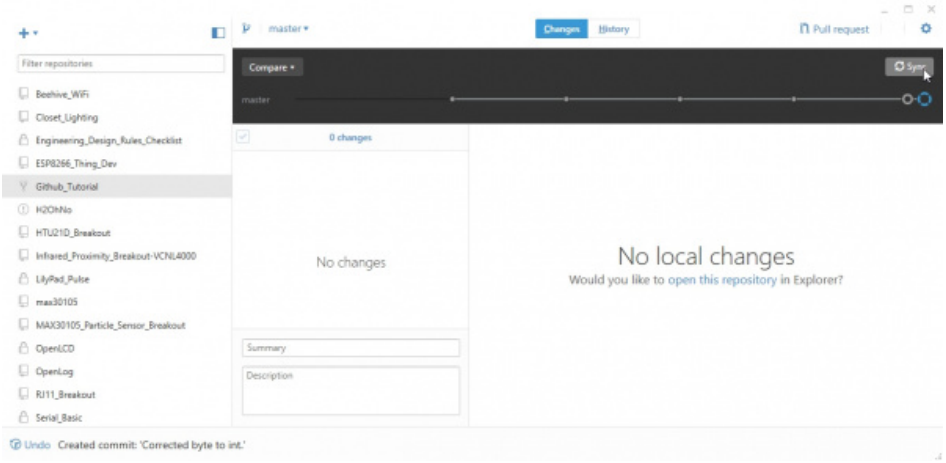
Local Repo

Local repo: Now let's commit the change we've made to our local repo. We must comment about what we changed before we can

commit it. Thoroughly describe what you did and then hit '**Commit to master**'.

As a general rule, try to commit small changes, frequently. If you wait 4 months between commits it is going to be very difficult for you to remember why you changed 5 lines of a subfunction.

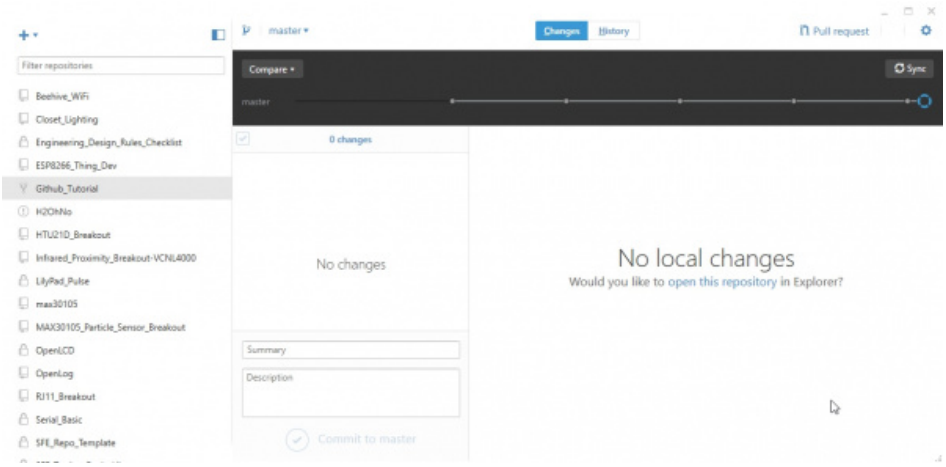
After pressing commit, the Sync button appears.



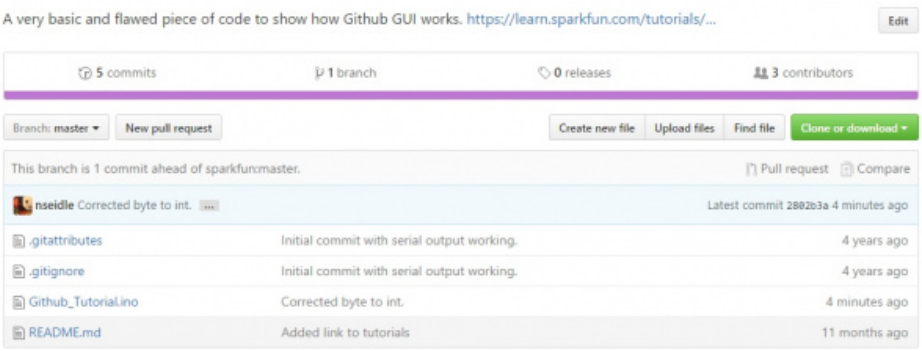
We now have *no uncommitted changes* but we **do** have *unsynced commits*. This means we've committed the changes to our local repo but we have not yet pushed (synchronized) the changes to our global repo.

Global Repo

Global repo: Press 'Sync'. This will push the changes that we made within our local repo up to GitHub and to our account.



Nice job Bob! We have successfully pushed this corrected code up to the global repo on GitHub. Now go online and look at your repo.

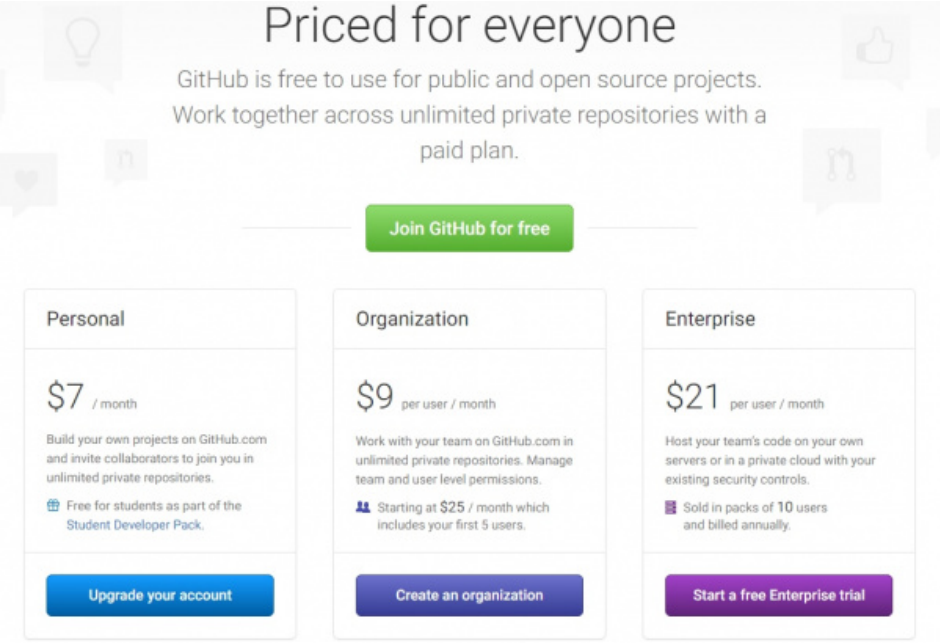


We can see the notes from the commit we made as well as the status of this fork vs the original sparkfun repo: "*This branch is 1 commit ahead of sparkfun:master*".

The GitHub web interface is similar to the Windows GUI but adds many advanced options. Use the web for changing properties of the project; use the GUI for routine commits to the local repo and global syncs.

Beyond Free

GitHub has a variety of pricing models but there's a free version that has all the power and as many public repositories as you want (yay Open Source Hardware!) but if you want private repos, you have to pay. SparkFun pays for the Organizational level because we love GitHub, use them extensively for our web development and use GitHub for our public hardware projects. We generally create a private repo for a new project and turn it public as we near the release date for the product.



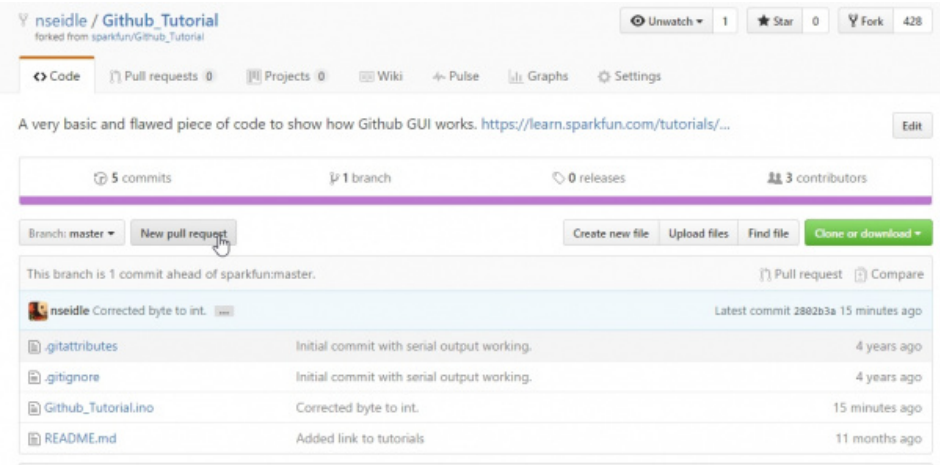
That's it for this section. These steps of forking a repo or creating your repo should allow you to create code projects, PCB layouts (we push Eagle files up to GitHub all the time), documents, images and even binary files.

In this section we found a bug and corrected it, but we have not yet let the original project know about the error. The next section will cover how to send corrections and improvements back to the original project through pull-requests.

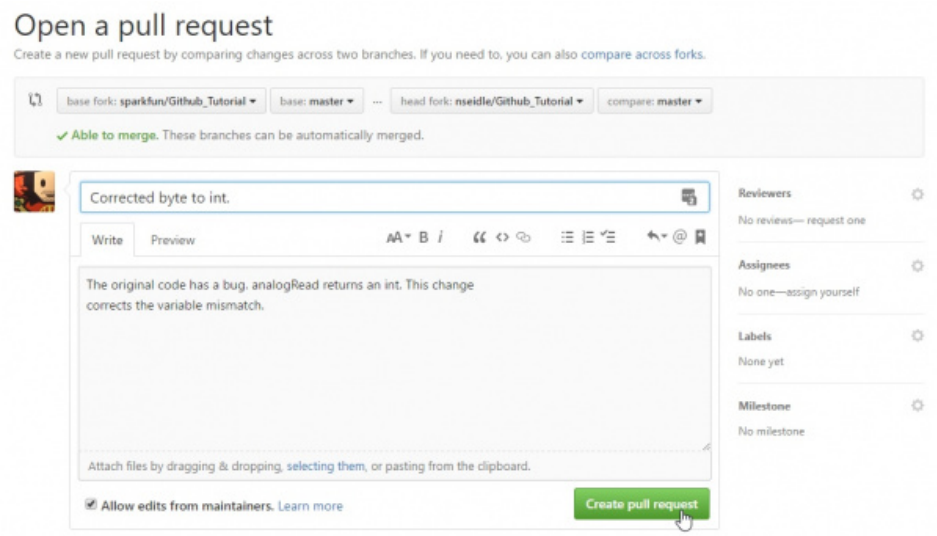
Pull Requests

Repositories are great for managing and tracking changes made to code over time. But the real power comes into play when you collaborate with multiple people on a project. When people have multiple improvements, how do we combine them? Pull requests allow contributors to give back to the main project.

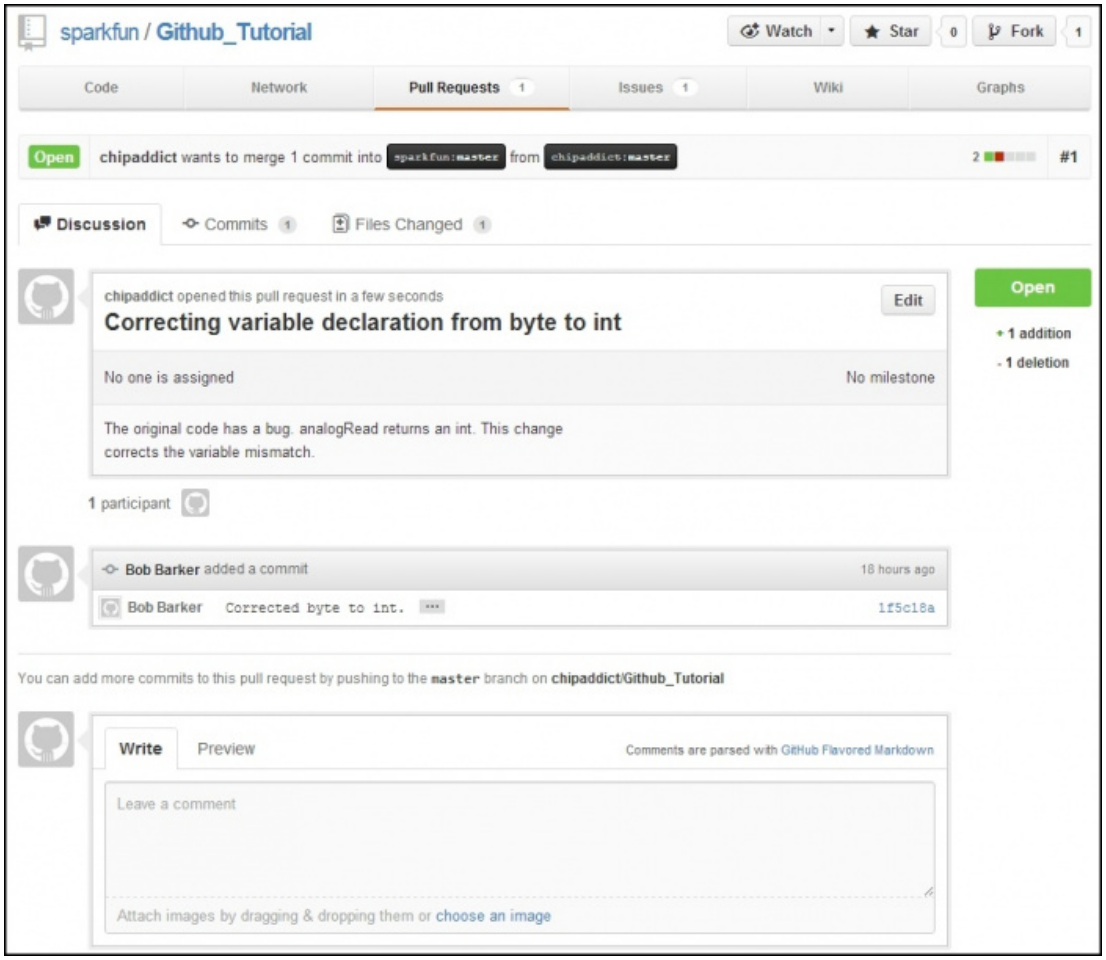
We've made some improvements to our version of the*Github_Tutorial* project. Now let's click on the Pull Request button to let the owner of the project know about the improvements we've made.



Here's where we describe the changes that we've made so that the owner of the main project (SparkFun is the owner in this example) knows what to expect. As with most comments, be as verbose as possible. The changes you've made are obvious to you but to a project owner with thousands of lines of codes and dozens of pull requests it can become confusing.

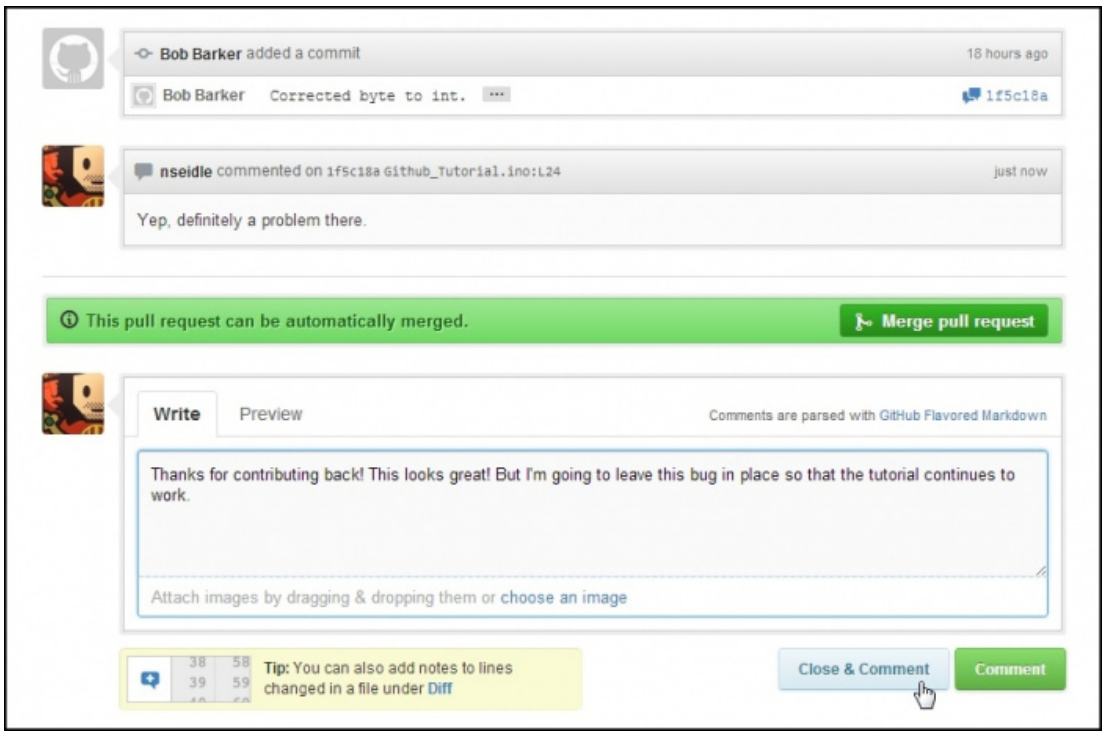


Once you've written a note about the changes you're proposing click on **Create pull request**'. Once we've sent the pull request, the owner of the main project is notified. Please don't hesitate to send a pull request to SparkFun for this tutorial. We'd love to hear from you!



This is where the owner of the project can review the submitted code (sometimes called a patch). GitHub provides a great discussion system so that the patch can be discussed. You can even comment on individual lines of code.

Here's what the pull request looks like from the owner's point of view. The owner has the option to merge this pull request or discuss it.



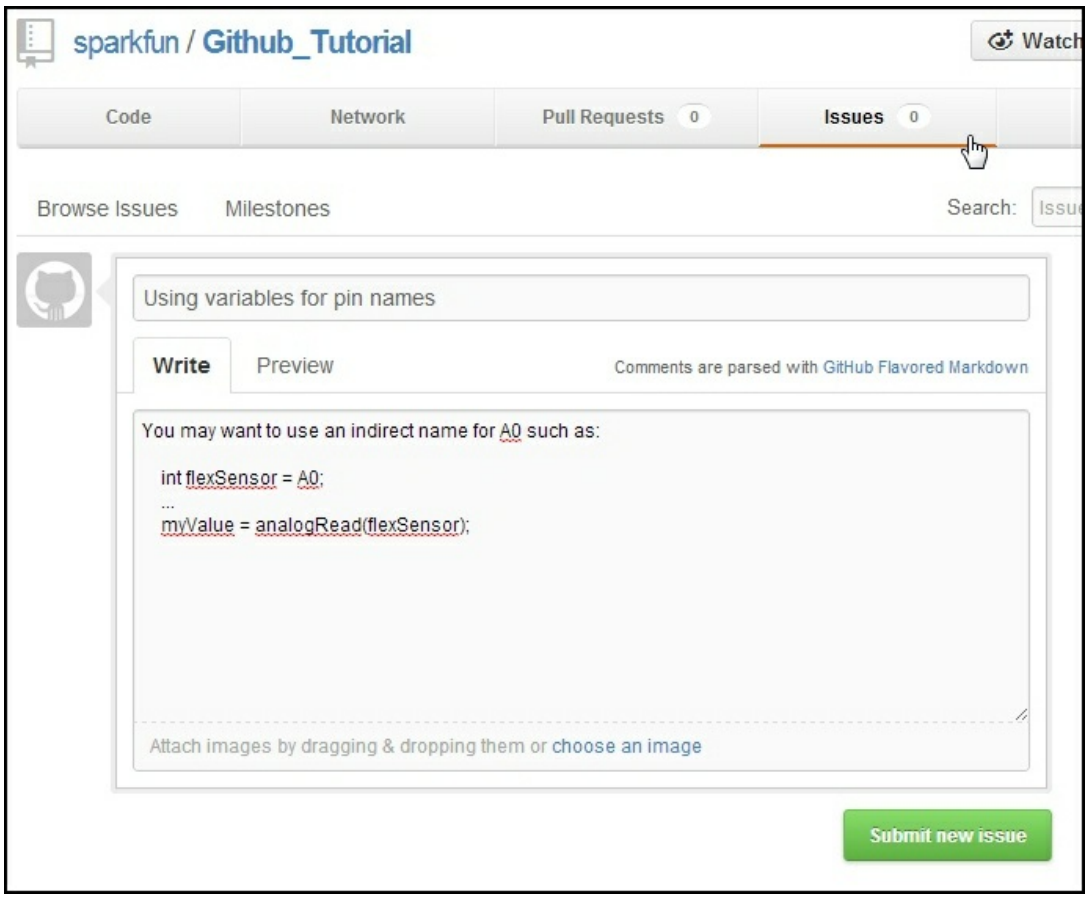
In general, create pull requests that are smaller and more simple in nature. This will make it easier for the project owner to wrap their head around. It's easier to accept pull requests that contain 5 or 10 changes, but a monumental task if you've completely rewritten 400 lines of code.

Issues and Wiki

Filing an Issue

There are some additional tools built into GitHub as well.**Issue Tracking** allows folks to post problems or issues with a given project. It's kind of like a ticketing system or tech support but with the ability to comment on a specific line of code.

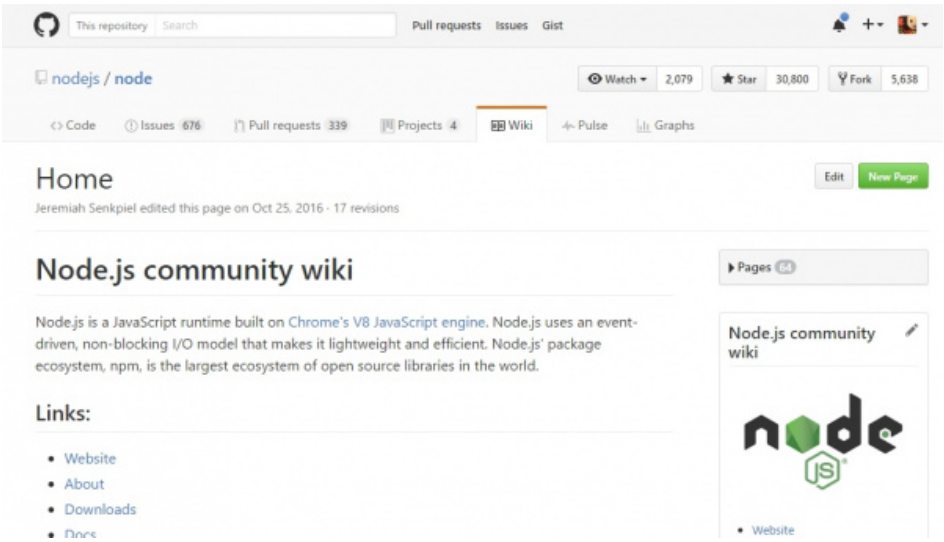
Here's is an example of creating an issue. Nothing too extraordinary but it allows for a good dialog between collaborators. You can see all the open issue on the **Github_Tutorial** project here.



GitHub Wiki

Every repo also has a Wiki available for use. This is handy for documentation, FAQs about your project, etc.

Node.js has a good example of using a wiki along side their repo.



At SparkFun, we don't often use the GitHub wiki and instead focus on hookup guides utilizing our own tutorial system. That said, for your personal projects the GitHub Wiki is a great, flexible place to have documentation for a given project or product. Collaborators can also help maintain and improve the documentation.

Try using Git and GitHub for your next project. There's an undeniable learning curve but it will make it much easier to collaborate with people.

Resources and Going Further

Now that you've got repos under control we recommend you check out these tutorials:

- Ready to up your game? Help us make the world better! Read how to use GitHub to help us improve our libraries and hardware.
- Checkout the MAX30105 repo to see how we set it up. There's also a separate repo for the Arduino library.
