A Git Practical Guide for Dummies

Content Page Layout v0.1:

1. What is Git
2. Why you should use Git?
3. Understanding Git from a fundamental level
4. Setting up the Git environment
5. Basic must-know Commands in Git
6. How can Git help you to manage your projects

Content page layout v0.2:

1. Problem statement
2. Scenario simulation using Dropbox

Analogies to use for Git for Dummies Learning Lesson

* Drop Box
* Story-line based

1. What is Git?

Git in a nutshell is basically a distributed version control system that helps you manage your documents, files or any collection of information. Git allows you to:

* Track your work history as well as all its changes
* Undo mistakes that you accidentally make to your work
* Collaborate with other people on a project easily

1. Why should you use Git?

* Git is fast and entirely distributed

As compared to VCS tools like Perforce or Subversion, Git does nearly all of its operations without needing a network connection, including history viewing, difference viewing and committing.

This also means that Git is very fast compared to those systems partially due to the fact that none of these operations has any dependency on network latency.

An result of an experiment conducted by professionals indicated that the ‘log’ command took 3.7 seconds to run in subversion while Git took only 0.3 seconds! You’ll find similar differences with nearly any command comparison. Take the commit command as another example. Since we can isolate the commit from the network ‘push’ in Git, this action takes a quarter of a second in Git but 45 seconds in Subversion. Even if you need to push to a shared repository, Git still is way faster as compared to subversion at 6 seconds. Therefore with Git’s fully distributed characteristic, most commands all seem instantaneous.

Small

Content page layout v0.2

What can a single user use Git for?

Take writing a report as an example; with Git you could:

* Have a historical archive of all your report versions without having multiple files
* See the difference between any two versions of the report
* Assign tags to indicate any important report updates
* Go back in time to a work on an earlier version of the report.
* Create and work on an experimental branch of the report, keeping the main branch unaffected
* Have a backup of your historical archive on the Git Server

What can a multiple user environment use Git for?

Same example, writing a report but involving many teammates;

Git Tutorial v 0.1

1. What is Git & Git Hub

Git is a Version Control System which is used by developers to manage source codes.

Git Hub is a web-based hosting service for software development project that uses Git.

1. Getting started

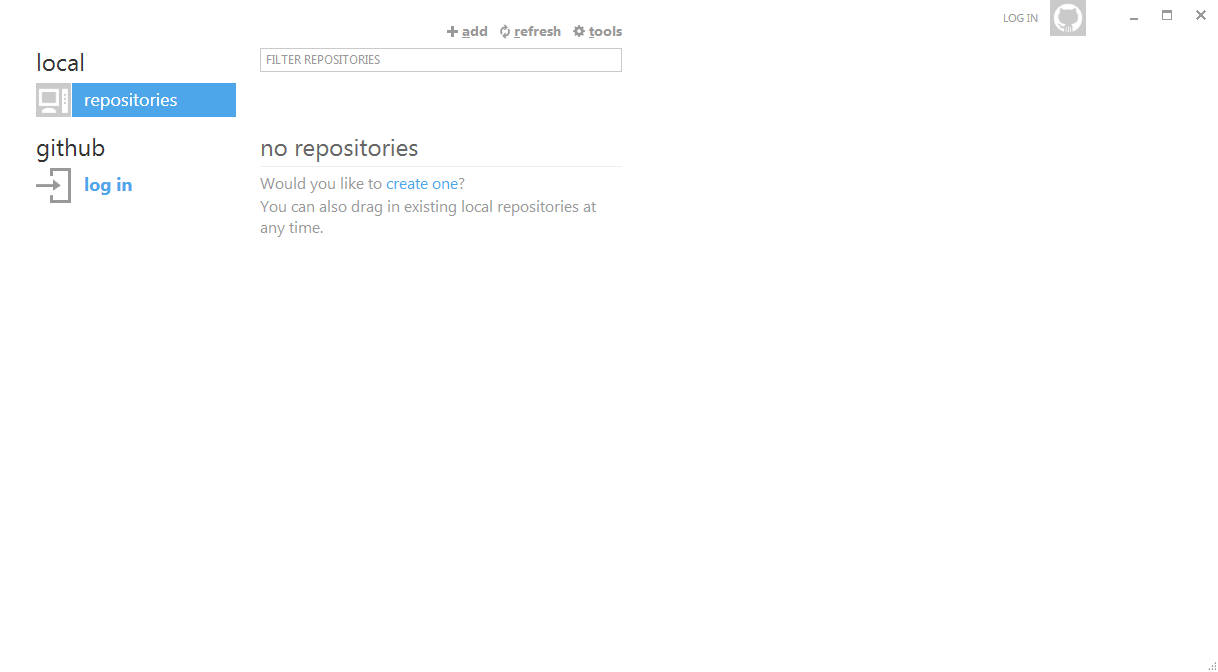
* Install Git from <http://git-scm.com/downloads>
* Install Git Hub from <http://windows.github.com/> or <http://mac.github.com/>

Download from the former if you are using Windows or the latter if you are using a Mac. Create a Git Hub account so that you can collaborate with your project mates at github.com

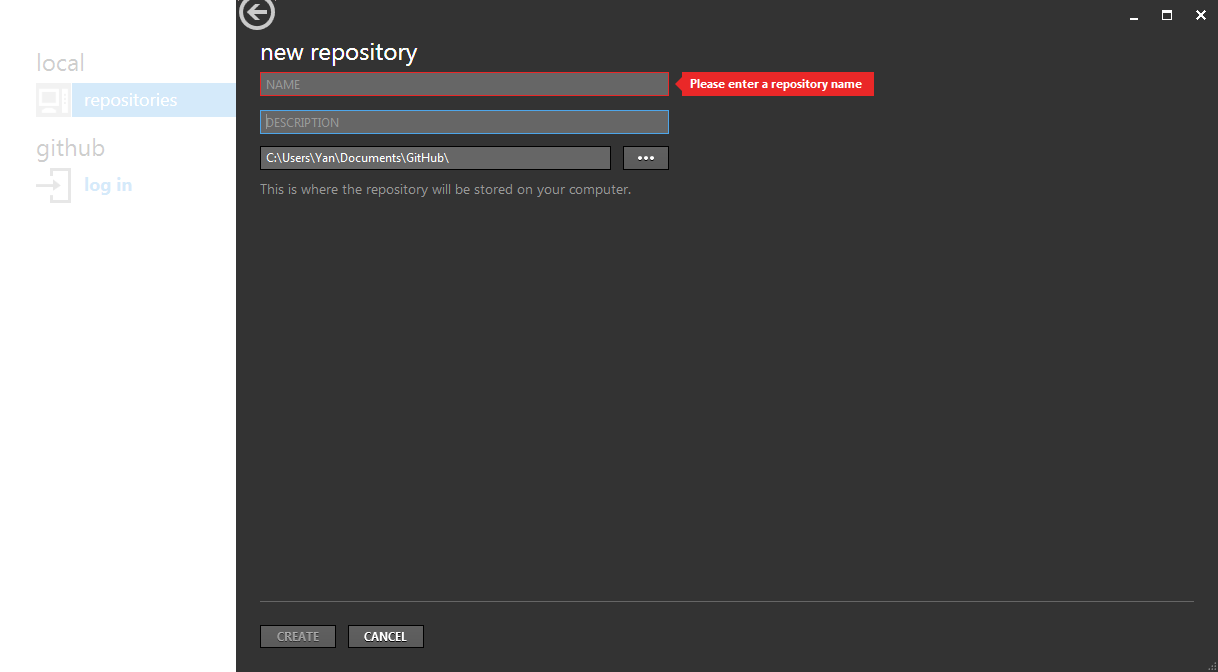
* 1. Create a Git Repository

There will be two categories of Git repositories here, local and remote. The local repository is offline (present in your laptop or desktop) while the remote repository is online (present in your GitHub account).

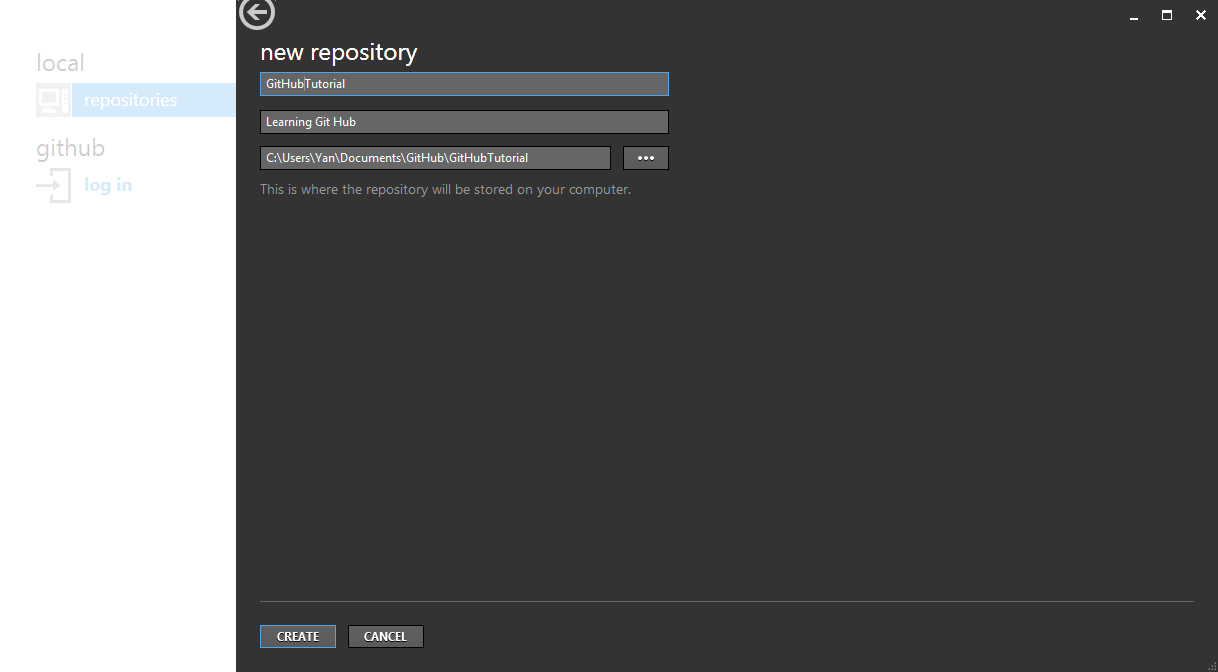
To create a local or online repo, you can use Git Hub’s GUI control which is very user friendly.



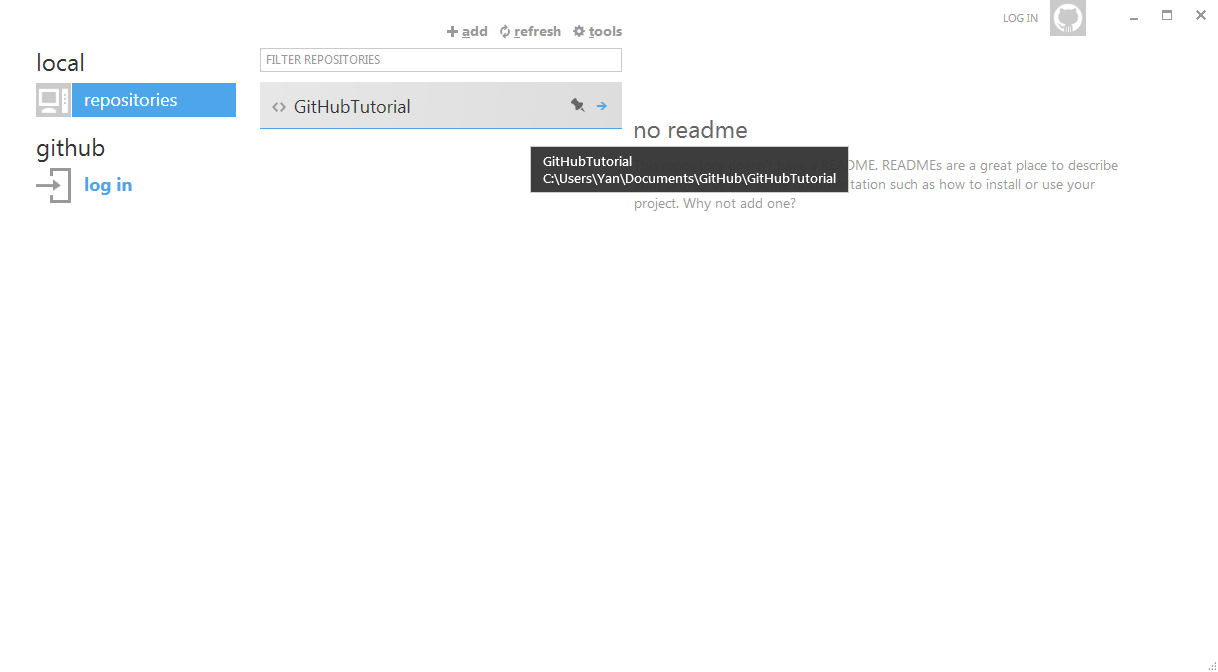
Click on the *“create one”* link and you can get started quickly.



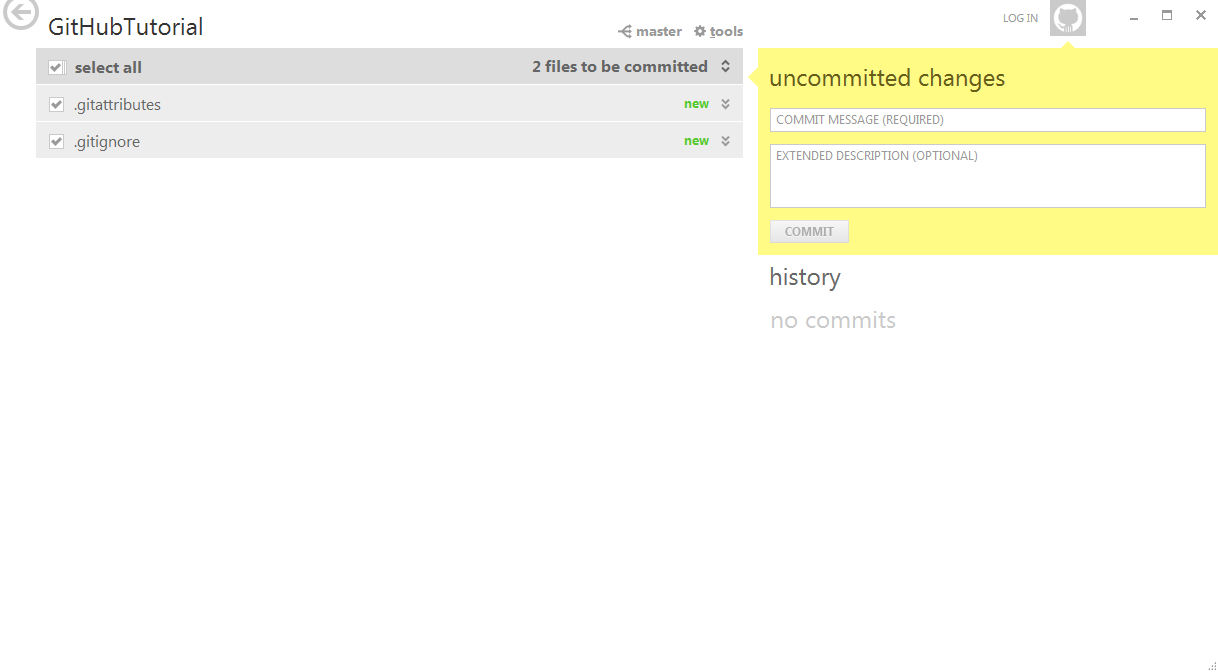
Git Hub will prompt you for the name and description of your local repo.



As you can see the name you gave for your repo will be part of the directory path for your repo location.

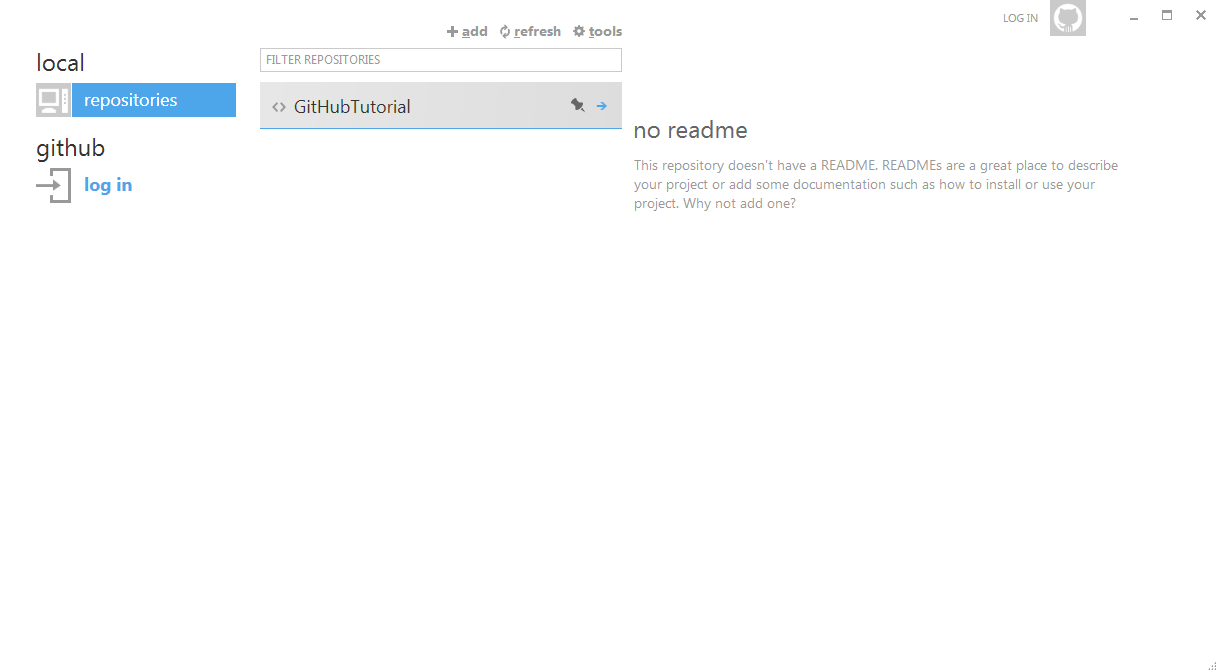


And there you have it your first local repo! If you want to stop using this repo, just simply right click the repo, click the *“stop tracking this repo”* option and delete the folder. If you have an existing directory already, you can drag and drop the folder to tell Git Hub to start tracking this directory as a local Git repo.

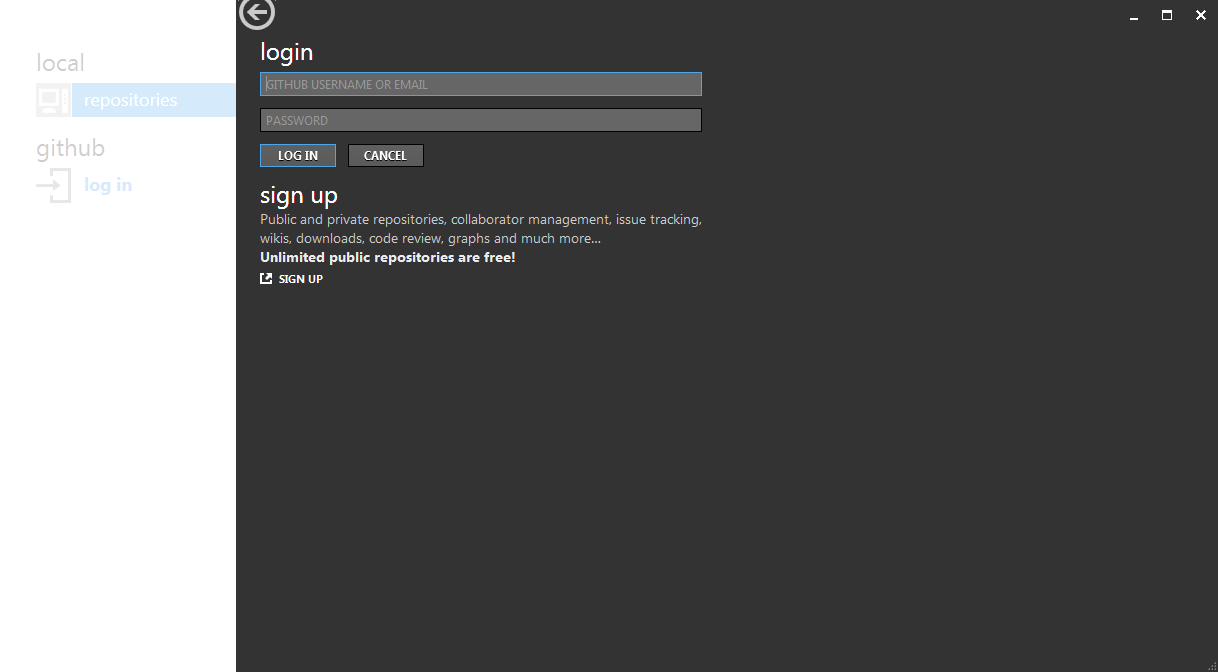


If you open your newly created *“GitHubTutorial”* repo, you can see two files inside. These two files are by default created whenever you create a new repo. But in any case you will not be touching these two files so just leave them alone. Let us now create an online repo in your Git Hub account/server.

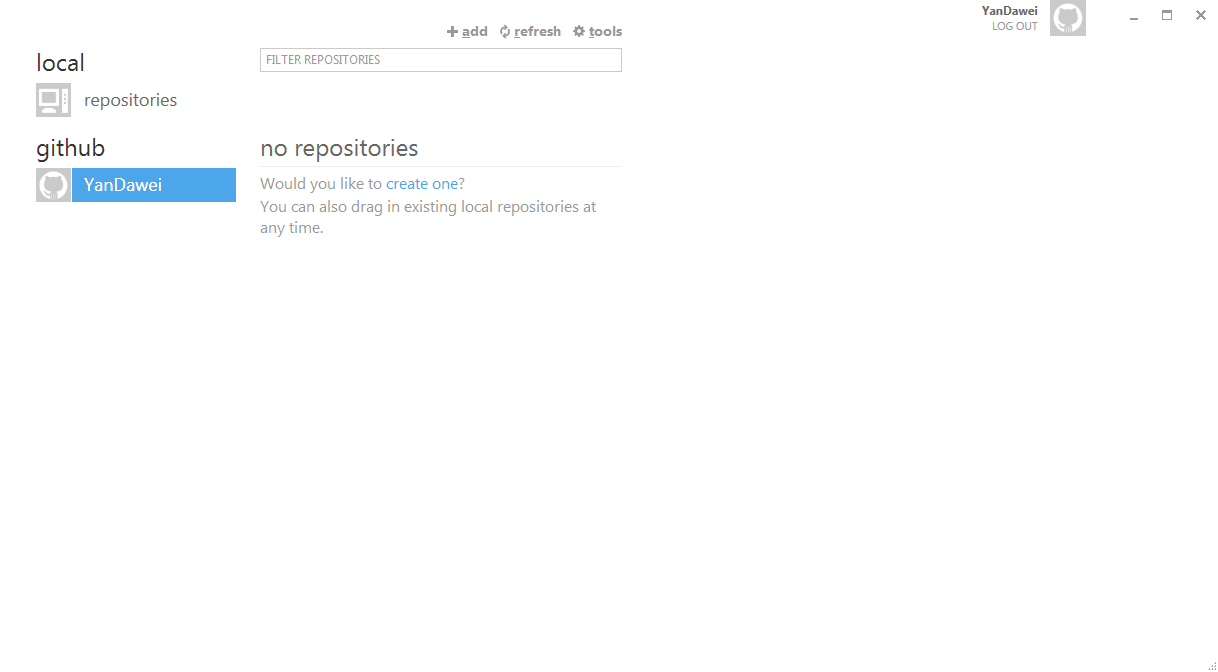
I am assuming you already created a Git Hub account through Git Hub website, if you haven’t; please go to this link github.com.



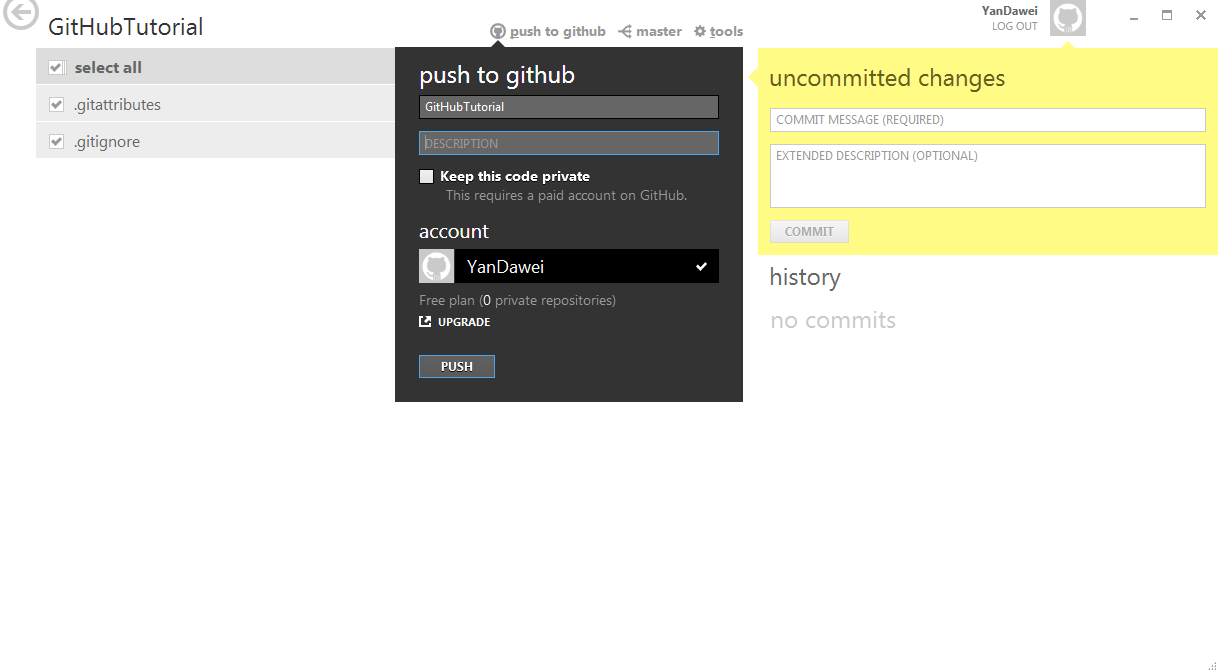
You can see a login link, click it.



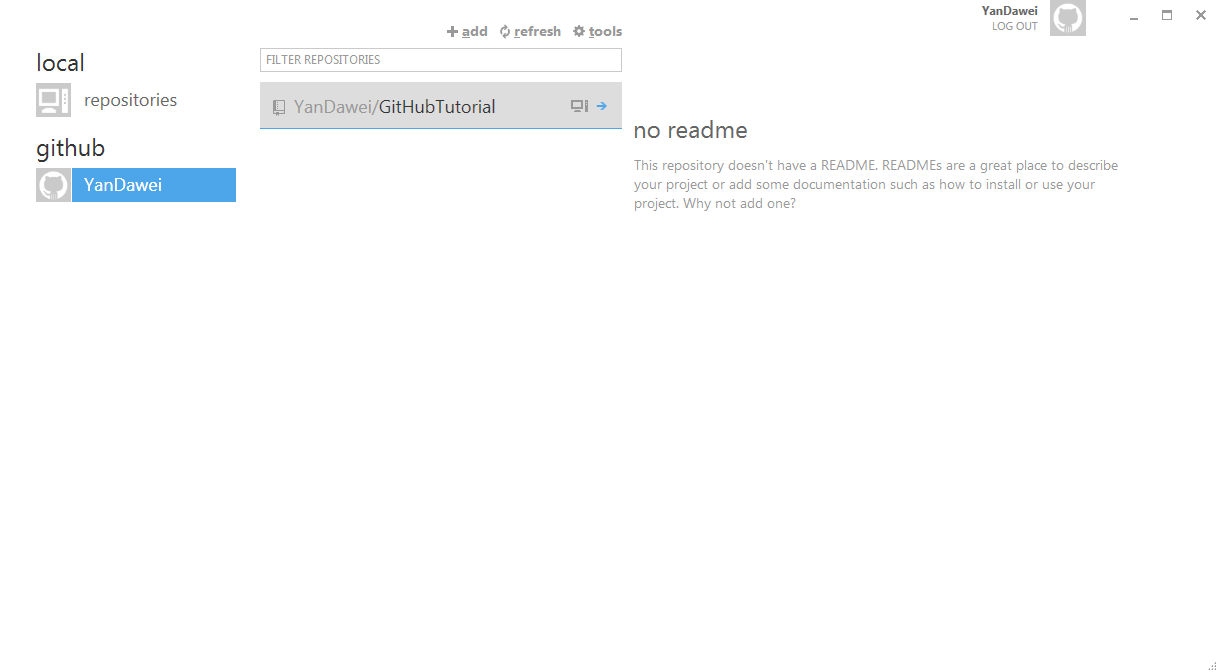
After which Git Hub will prompt you for your login credentials.



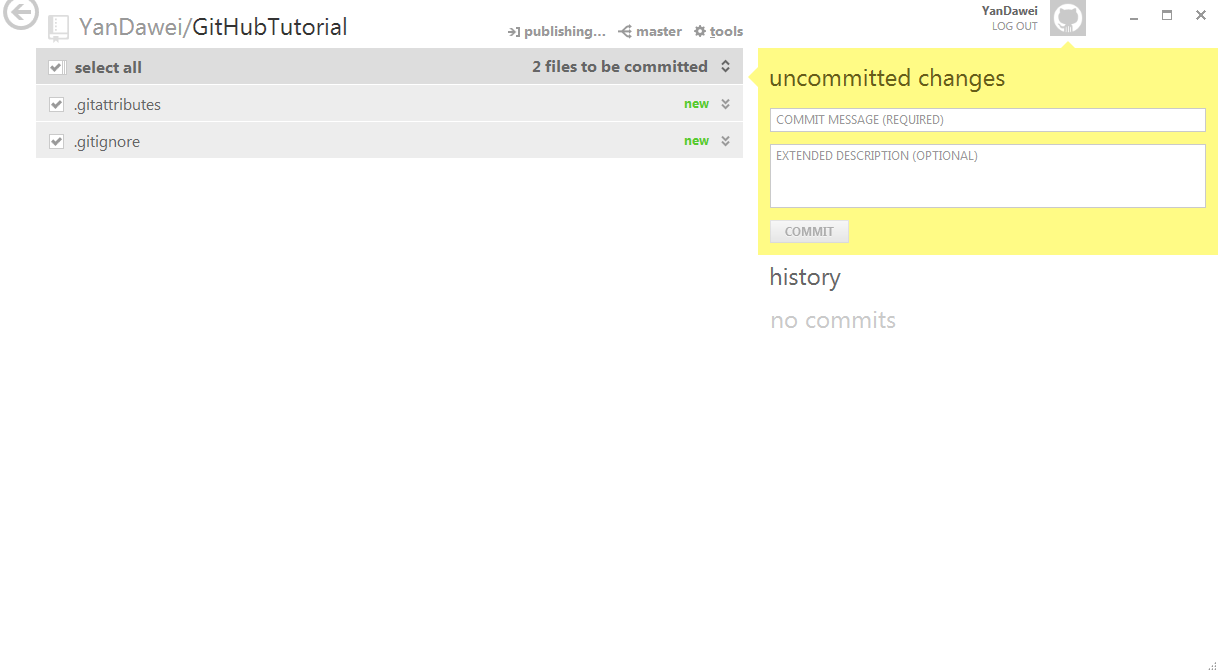
You can see that I have logged in to my Git Hub account and I have currently no online repos in my Git Hub server. Let’s create one!



Go back to your local repo, in this case *“GitHubTutorial”*. Click the push to GitHub link.



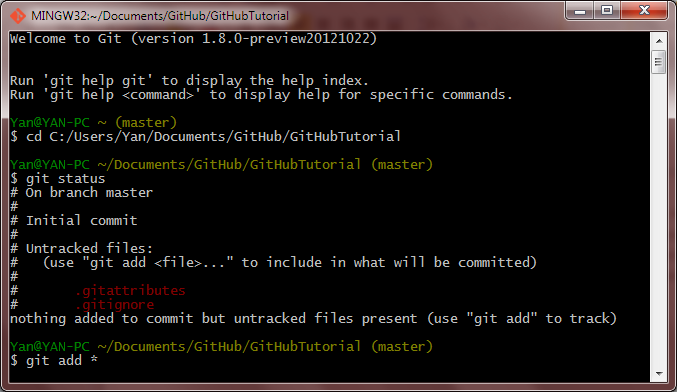
And there you have it your first online repo on Git Hub! Oh yea I forgot you still have to publish this Repo to the server to facilitate collaboration later on.



Just enter your Git Hub Repo and click on the *“publish”* link at the top. Add, Remove and committing files in the Repository

* 1. Committing, Reverting, Rolling back and pushing

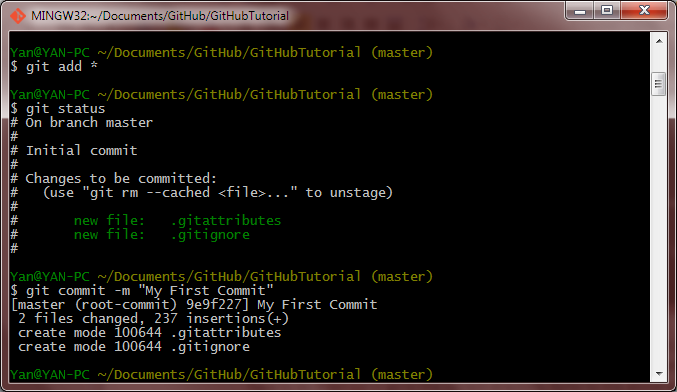
HAHAHAH, I bet you received a failed publish error message from Git Hub, that’s because you don’t have any commit yet in your Git Hub account. No Commit = No data in your Repo because Git stores snapshots of the data in the Repo only if you commit. Remember those two files that are automatically created, let us commit them.



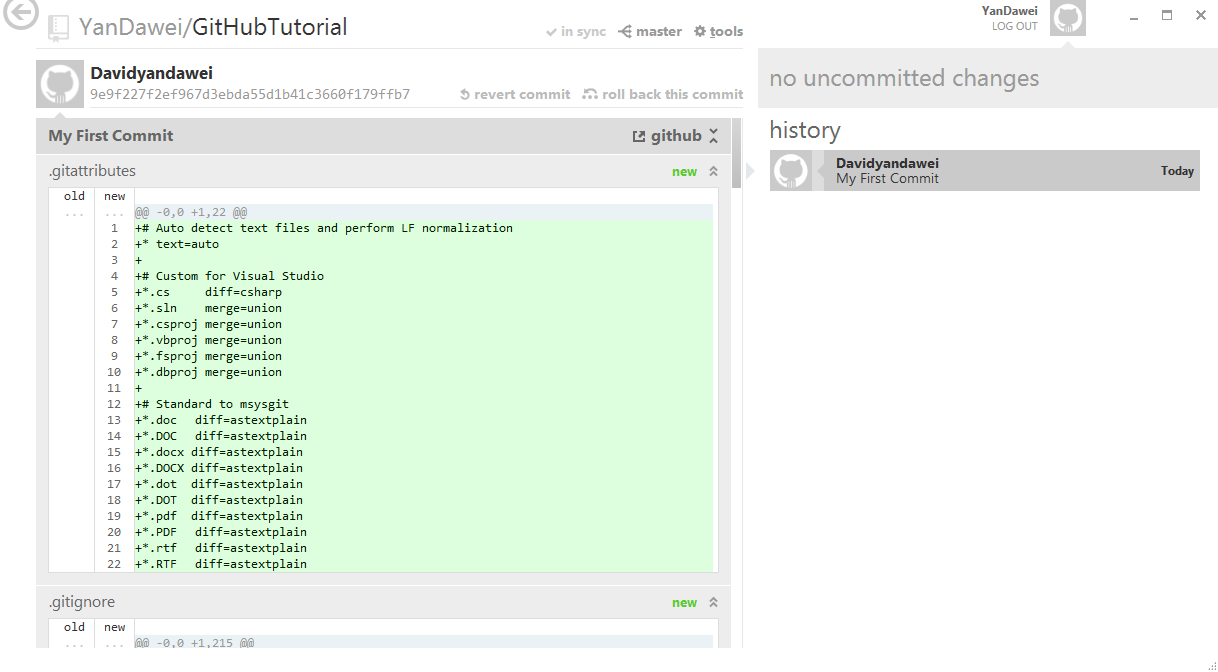
Open up the Git Bash console, it should look like the above screenshot. By default, Git goes to my directory C:/Users/Yan, it may be different on your computer but don’t worry, all we have to do is tell Git to go to the correct directory. Refer to the above screenshot for all the commands.

Step 1: To tell Git to switch to your desired directory type “$ cd <input the directory path here>”

From the screenshot above, I have typed in “$ git status”. To commit, I do not need this command however this command will come in handy as it allows you to check for any untracked files in the Repo. The files highlighted in red are the untracked files or files that are currently in the Repo but not monitored by Git. To tell Git to monitor them, type the following in Step 2: “$ git add <your file names here>”. A “\*” sign means you want to track every non-tracked file in the Repo which is what I did.



Now Git has started tracking the files and you can proceed to commit them. You can always check for any untracked, tracked, uncommitted files through the “$ git status” command. And yes, our files are tracked if you refer to the screenshot right above. To commit, type the following in Step 3 “$ git commit –m <your commit message>”. The “-m” part of the command is for you to give a name for the particular commit, this will allow ease of reference later on. Let’s get back to the Git Hub console and check it out.

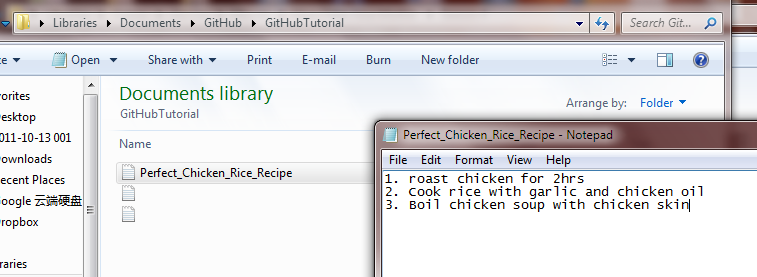


Ah, we have it, our first commit! Now you can proceed on to publishing.

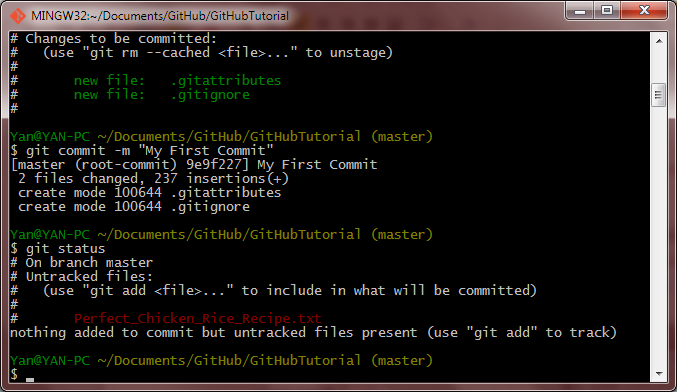
So let’s review what we have done now:

1. We have created our personal account in Git Hub
2. We have created both a local and an online repo that are synced
3. We have done our first commit

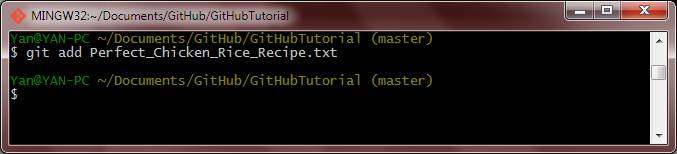
Now let us assume we have a project called “The perfect chicken ricerecipe” project. We will create a text file named “recipe for the perfect chicken rice”.

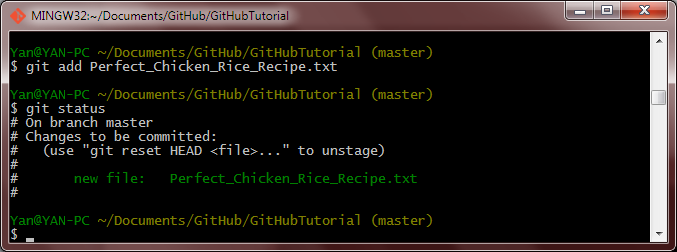


Now as you can see I created the file inside the repo and inputted some comments. Let us now check the status of the repo.

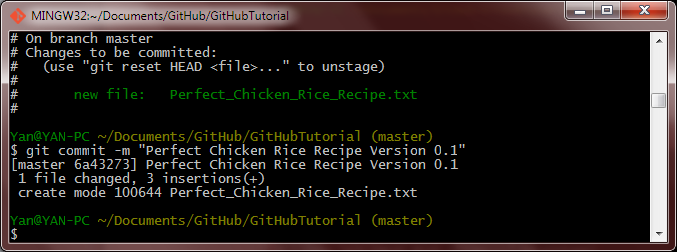


Git immediately recognizes this file! let us now track this file.

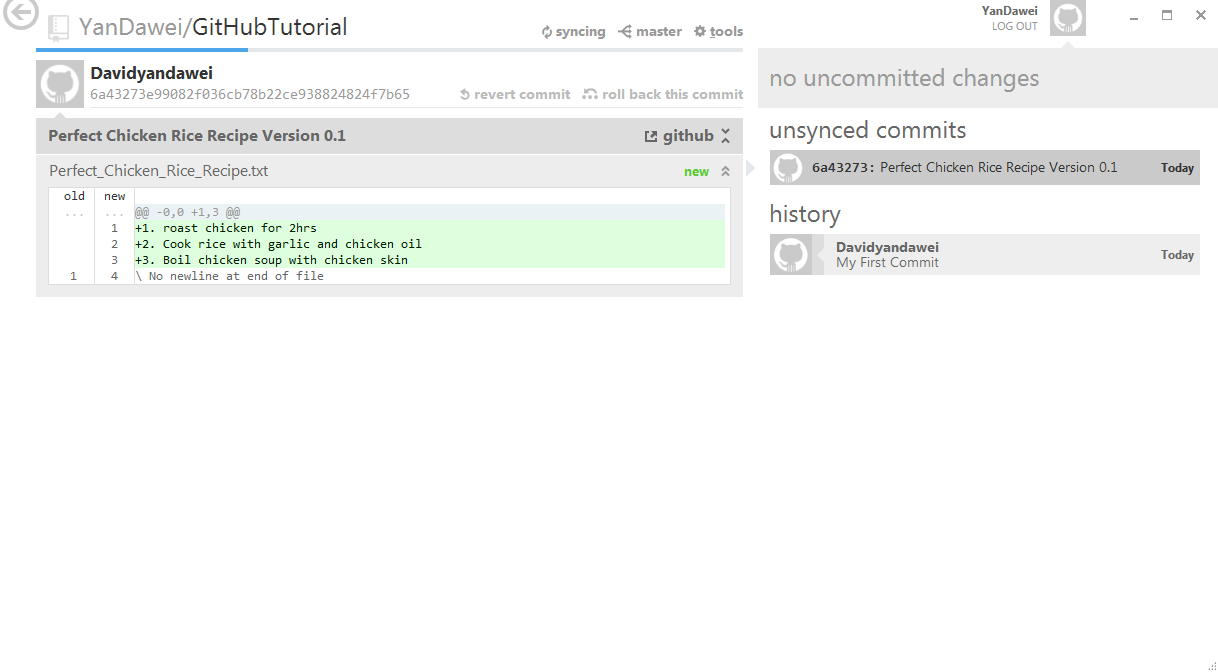


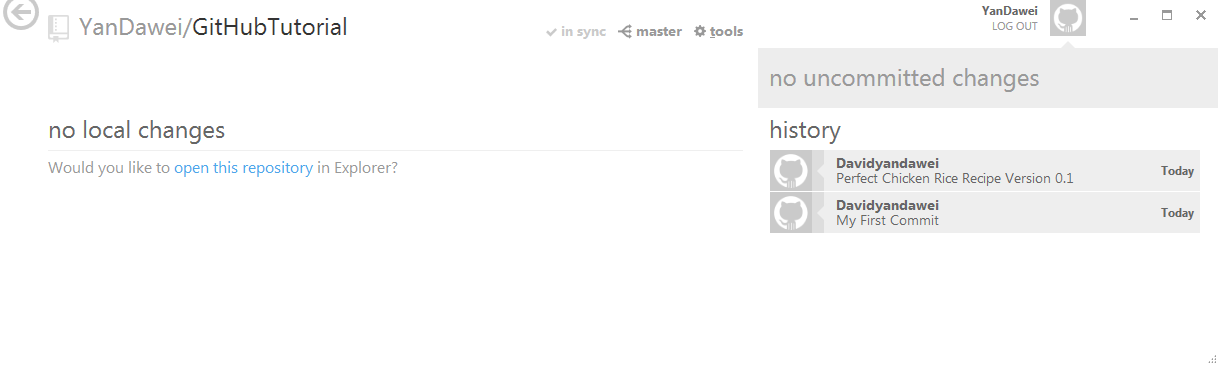


Now we make a commit in the repo again.

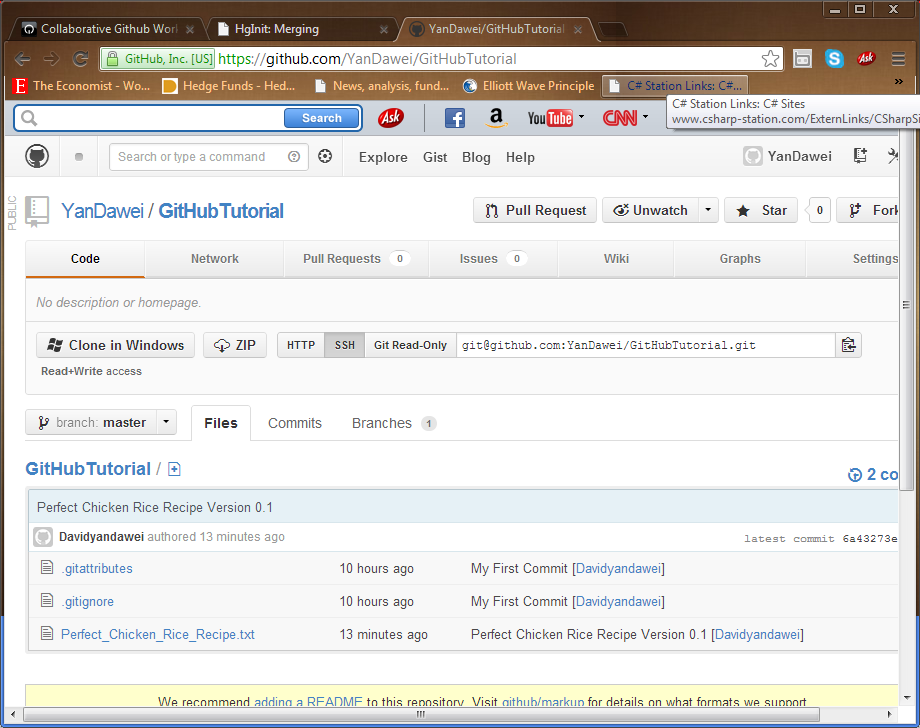


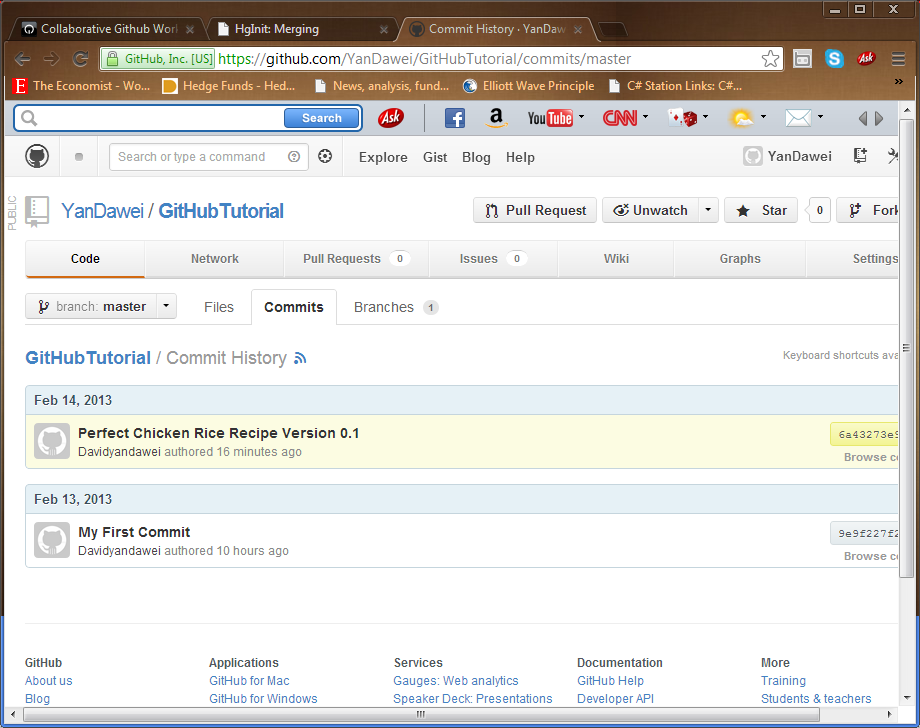
And I give it a sensible message. Now let us sync this commit to our Git Hub Repo.



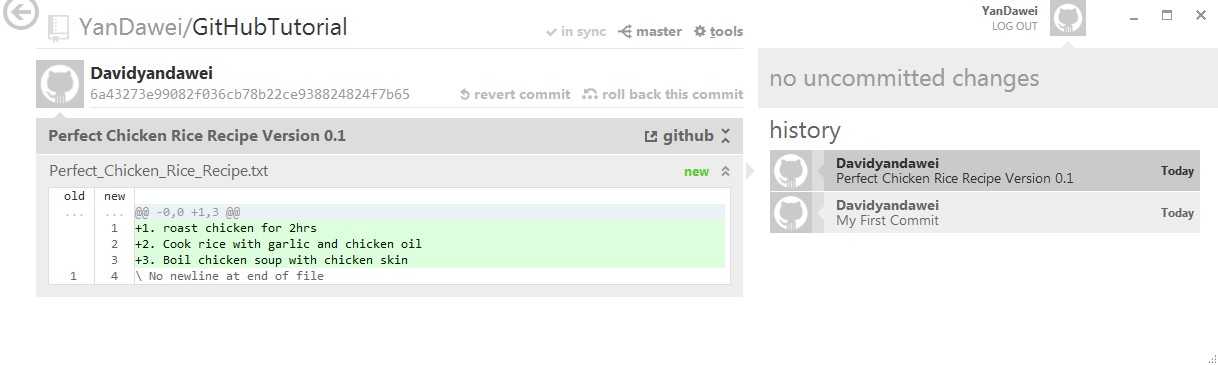


If you go to the Git Hub webpage and sign in to your account you can see your repo and all your commits as well.



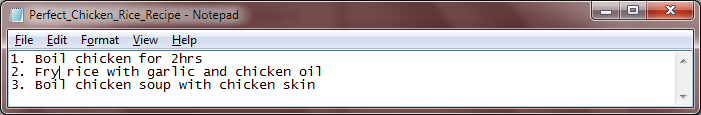
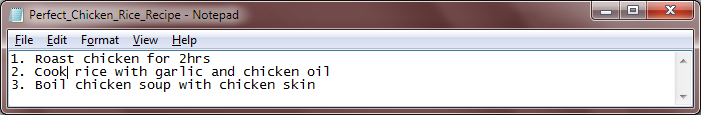


We will get back to using Git Hub web tools later.



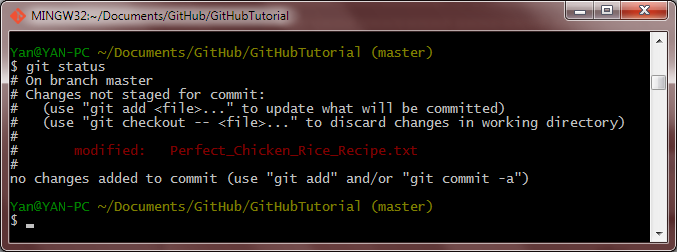
Now can you spot the “revert commit” and “roll back this commit” link? We will now learn what they are and how can we use them. Let us now change the file contents inside our perfect chicken rice recipe.

Before



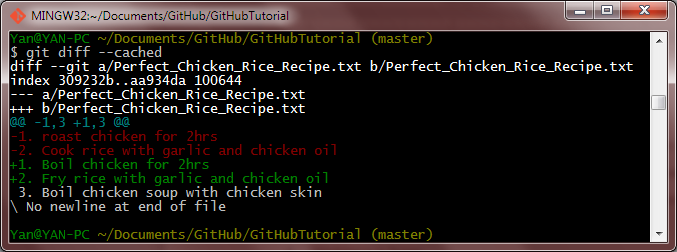
After

We have changed Roast to Boil and Cool to Fry, which makes no sense at all but this is just for experimental purposes. Let’s check the repo status now.



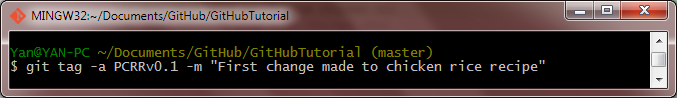
Git recognizes the modified file and gave us options to either discard the modification or tell us to track the modified file. We will assume we want the changes and to track this file.

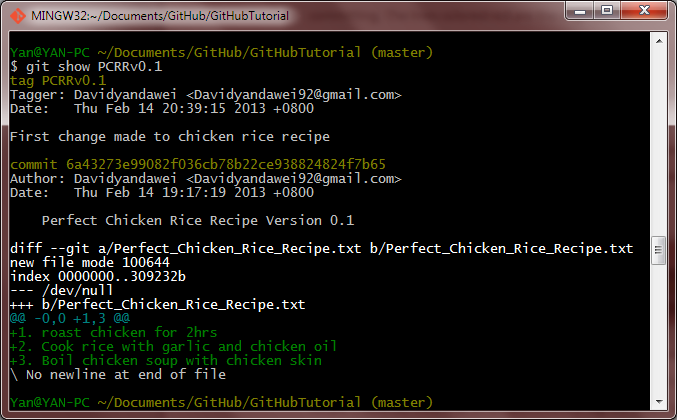
If we wish to see the difference between the previous file and the changed file we can run a git diff command.



Now there are a lot of different types of git diff commands pertaining different situations. The one I used above $ git diff –cached is to see the difference between the last commit and the changes that I have not yet committed. Ultimately it is to see what changes I would be committing. The lines colored red are the originals while the ones colored green are the changes.

Now you know that we can input a message alongside the commit to provide easier historical reference in the future. Right now I’m going to teach you another method to do that, tagging. Tagging can be used to mark and important point in history which you may want to reference back to in the future. It can also be used alongside commit messages. Let us now create a tag.

 Observe the above screenshot, I have created a new tag called PCRRv0.1 (Perfect Chicken Rice Recipe version 0.1) and assigned a message called “First change made to chicken rice recipe”. Now do note that when you create a tag like this, it automatically tags the most recent commit. So if we want to take a look at what commit is being tagged right now, just simply do a show tag command.



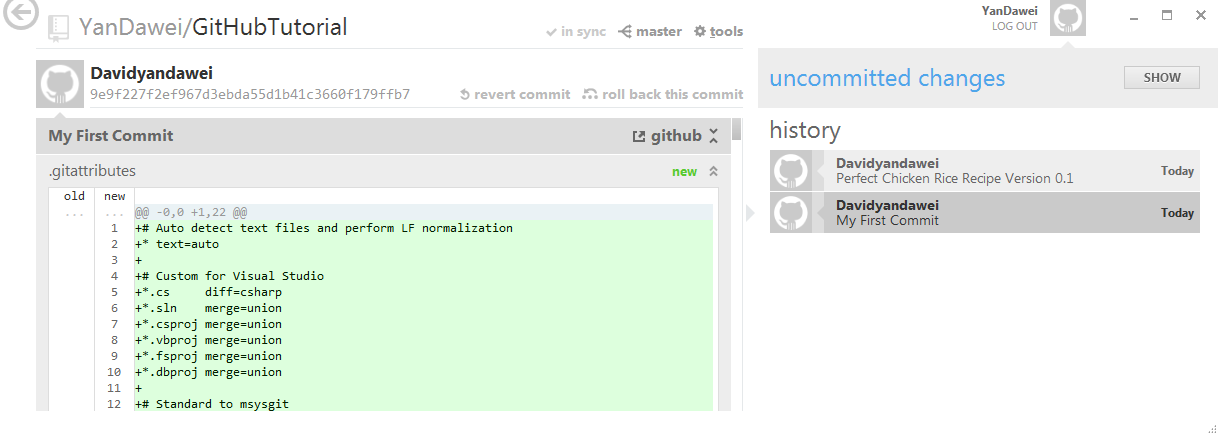
You can see that it automatically tags the most recent commit.

If you wish to tag a particular commit in the past, in my case I want to tag the commit before I created the chicken rice file, you have to first identify the commit you wish to tag by its commit checksum.

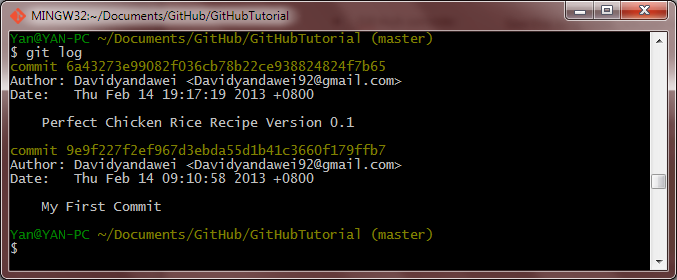
You can do that in a variety of ways:

See this long code; we just need the first 7 elements

* Git Hub console:

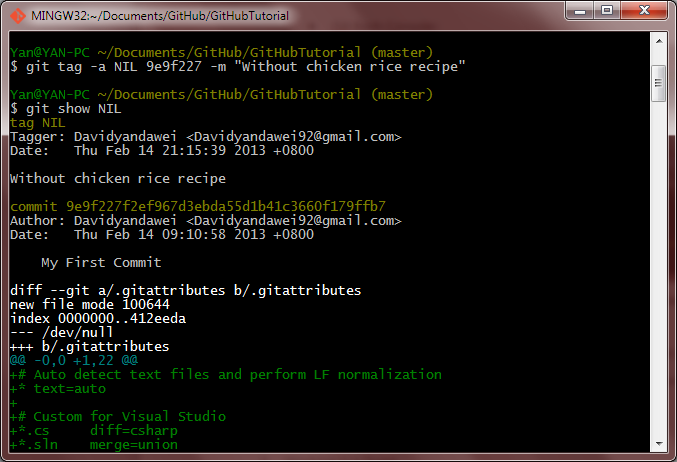


* Git Bash Console:



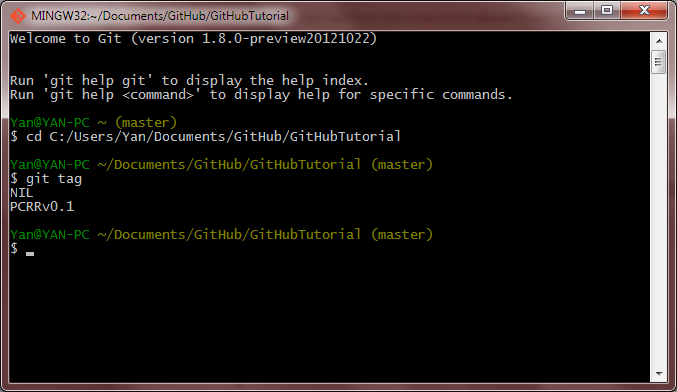
Just run a git log command and take note of the first 7 elements of the commit checksum

Now that we know that the commit we want to tag has a commit checksum of 9e9f227, we will proceed on to tagging that commit. We will first create a new tag then tag that tag to that commit.

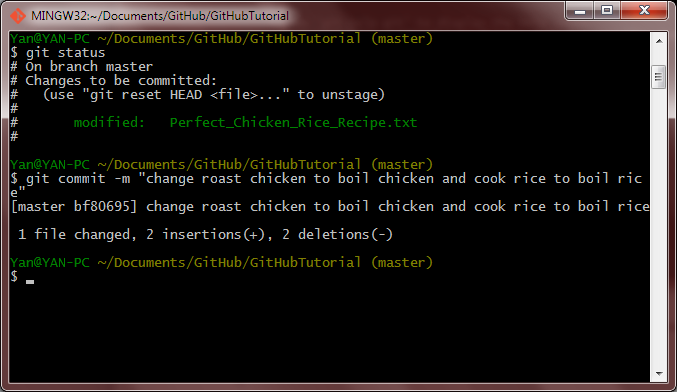


As you can see from the screenshot above I have created a tag called NIL with a tag message of “without chicken rice recipe and tagged it to my very first commit.

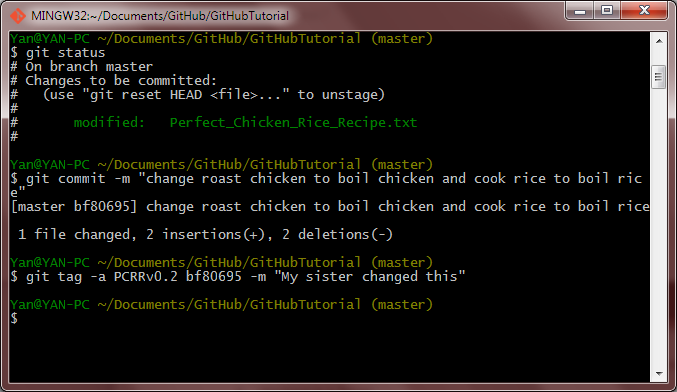
Now let’s see the list of tags that are currently in the repo,



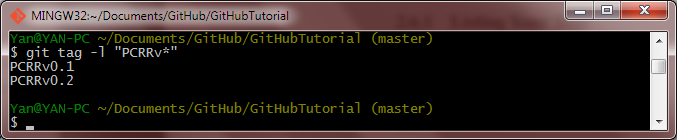
Ok enough of tagging; let us now continue where we ended off previously.



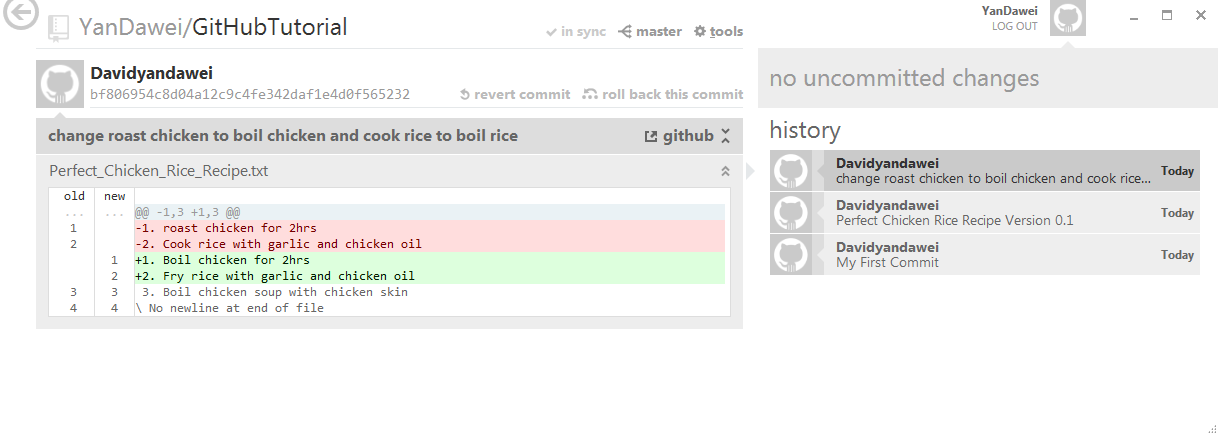
Now that I have committed the weird changes, let us assign a tag to this.



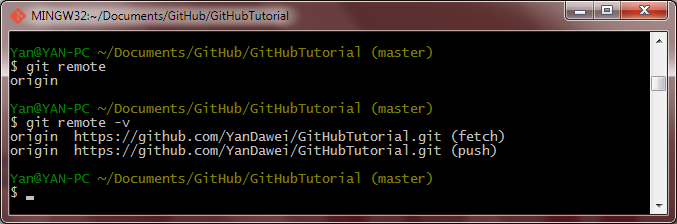
Since the changes are under the project Chicken Rice Recipe, the tag will be named PCRRv0.2. Suppose we wish to see all the versions under the chicken rice recipe tag, we can do the following:



The list tag command $ git tag –l <tag> will do the job. Now we can proceed to publish the new commit to our Git Hub server using the Git Hub console.

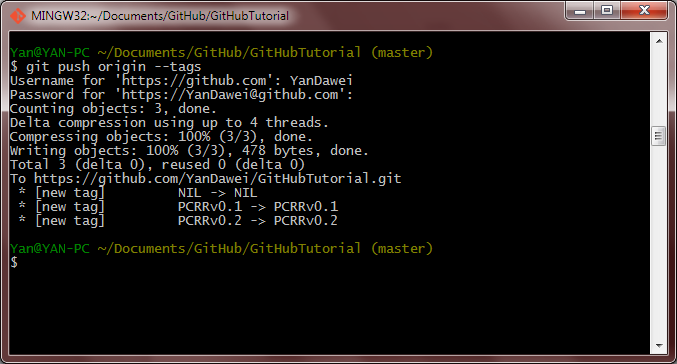


Yes its published to the server! Ok theres something that I missed out about tags. Tags that you created are not automatically pushed to Git Hub. So if you wish to share your tags with your collaborators, you have to push your tags to your git hub server. The command to do that is $ git push <remote repo name> <tag name>. In the following screenshot I will show you how to inspect the list of remote repos that are synced to the local repo.



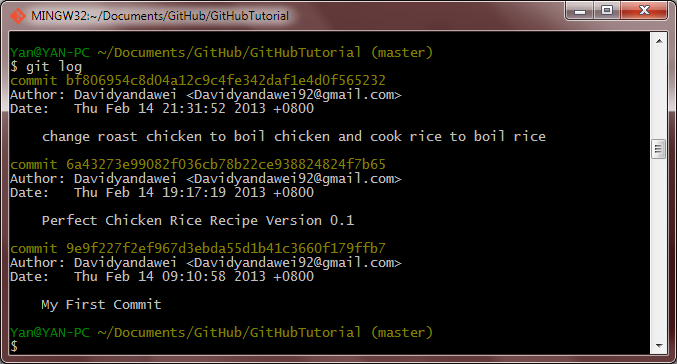
To see the names of your remote repo only, input the first command above, the added –v behind will provide you the URL of the remote repo as well. Now you might be thinking, does that mean I can sync multiple remote repos to my local repo or vice versa? Yes, you can and I will tell you how in the later sections.

Let’s push the tags now to the remote repo.

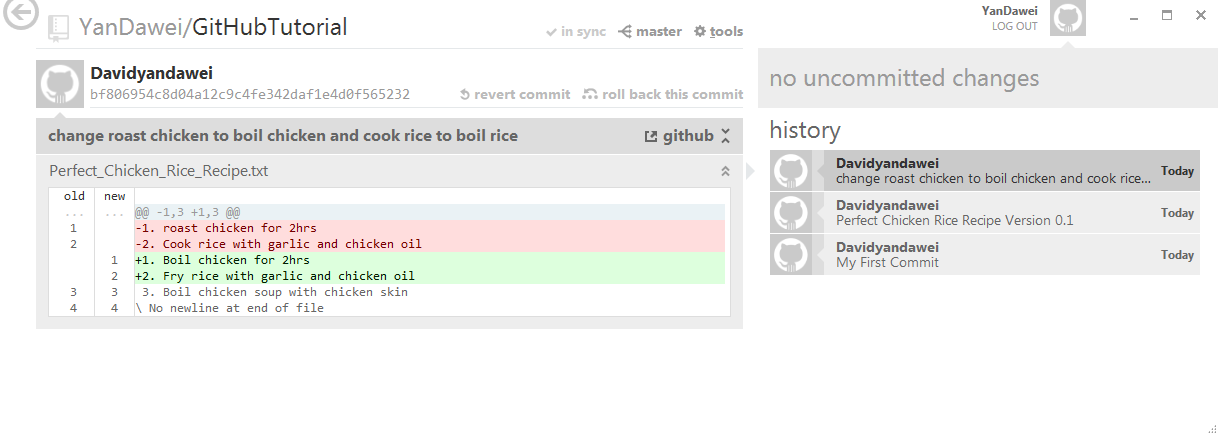


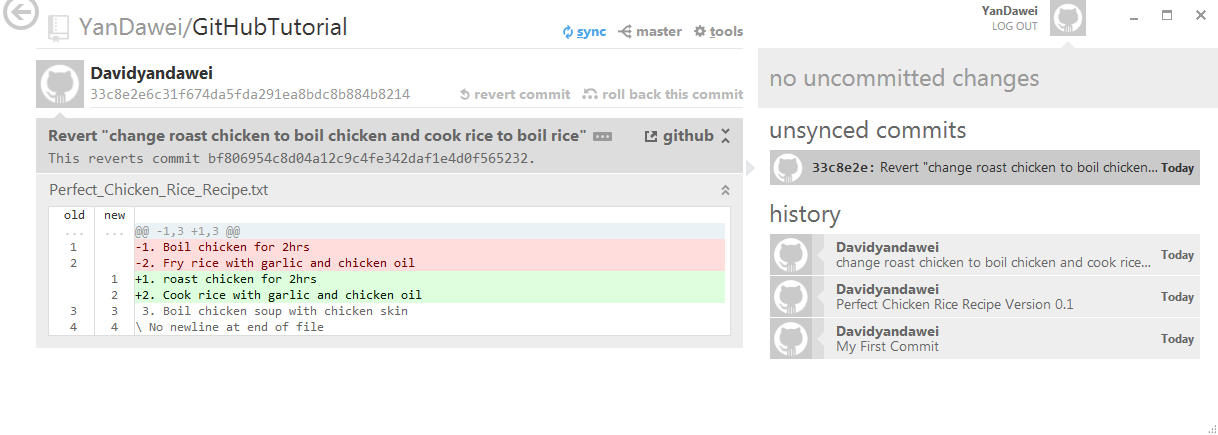
Yes, I guess I forgot to mention that you will need to create a SSH key for you to push items in your local repo to your remote repo, my bad. Because generating the SSH key is boring, I will not touch it in this document and anyway Git Hub already has a help section on it. Just go to this link: <https://help.github.com/articles/generating-ssh-keys>

Let us now take a look at our repo:



We have three commits and the most recent commit is really retarded. Suppose I want to remove the most recent commit using revert commit.





You can see the revert commit option creates a new commit that reverts my changes in the most recent commit. Let’s try the rollback this commit option.

1. Collaborating with others
   1. Set up a central repository
   2. Push changes to the central repository
   3. Pull changes from the central repository
   4. Merge changes from different contributors