GIT

# Objectives

* Learn to use GIT for code version control.

# References

* GitHub account: <https://github.com/>
* Download GitHub Desktop: <https://desktop.github.com/>
* Download GIT: <https://git-scm.com/>
* GIT Tutorial: <https://try.github.io/levels/1/challenges/1>
* Short GIT Reference: <http://gitref.org/index.html>
* Useful guides: <https://guides.github.com/>

# Module Prerequisites

None.

# Description

## Git

Git is a distributed version control system that has become very popular. This module will only look at the basics of Git and will mostly direct you to online material to review to learn about Git. To get an idea of what version control is and what Git is, watch the videos "What is Version Control", "What is Git" and "Get going with Git" on <https://git-scm.com/videos>.

To understand Git and distributed version control systems go to the link <https://git-scm.com/book/en/v2> and read sections 1.1-1.3, 2.1-2.3, 2.5. Note that the repository that is referred to most of the time is a local repository on your computer. Only when it talks about remotes are the repositories located remotely.

A good tutorial to follow is <https://try.github.io/levels/1/challenges/1>. When you walk through this tutorial be sure to scroll down and examine the contents of the folders and files in the "My Octobox Repository."

## GitHub

GitHub is a place to store your repositories remotely. It allows you to collaborate with others on the same project.

To get a GitHub account go to <https://github.com/> and follow the information on the front page to create an account. The account will be a free account and your projects and information will be shared publically. If you want a private account you will have to pay a monthly fee.

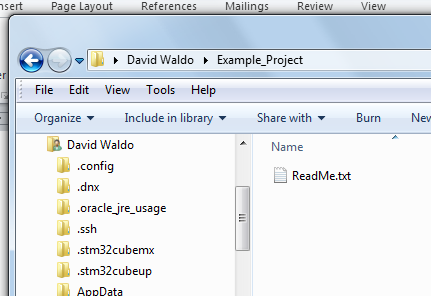
## GitHub Desktop

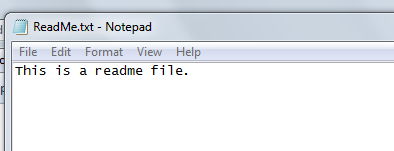
Git is written as a command line program. However, there are GUI based programs around that implement Git. GitHub has a GUI based application that interfaces with the GitHub repositories. It works well for basic project control tasks.

To install GitHub Desktop, go to <https://desktop.github.com/> and download and install it. When it starts up it will probably say there are no repositories found. It searches for repositories so it is possible it will find some.

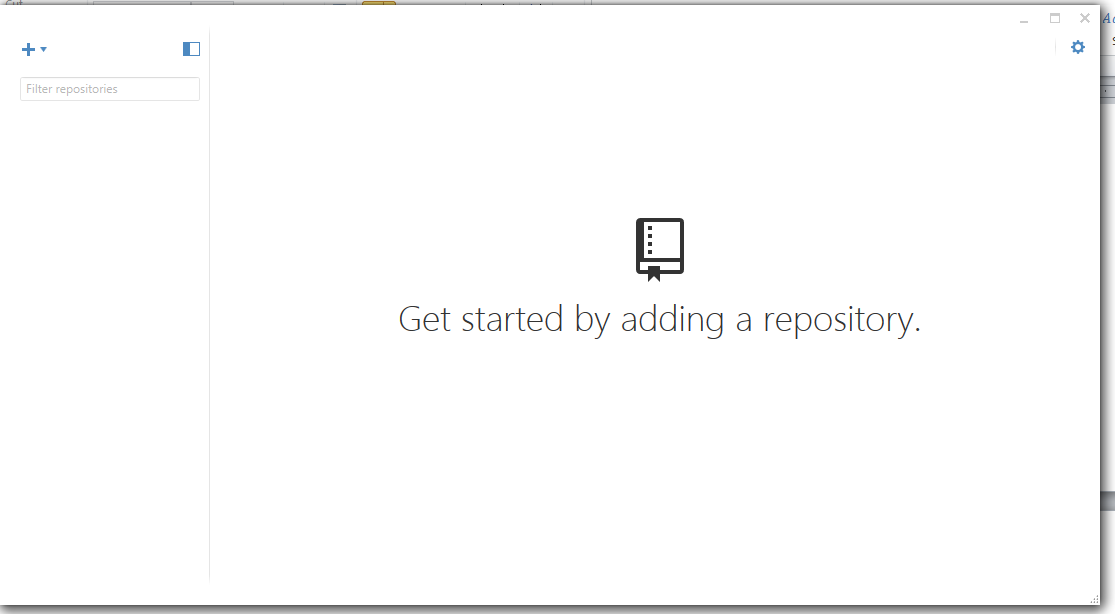
Let's walk through using GitHub Desktop to make a local repository and push it to a remote repository on GitHub.

* First, make a directory that has project files in it. I will be using a Windows computer. Make a directory C:\Users\<your name>\Example\_Project.
* Then make a text file in that project folder called ReadMe.txt.

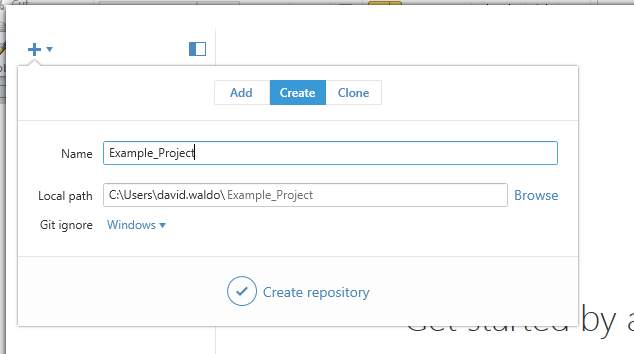




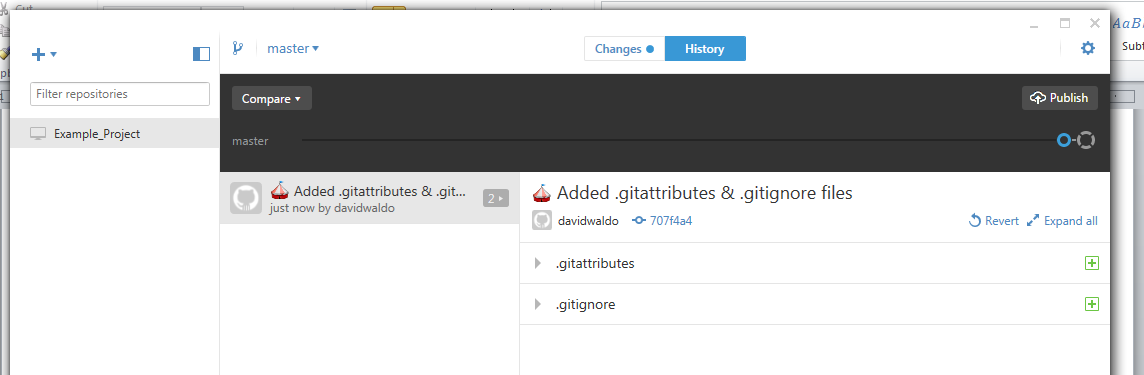
* Now open GitHub Desktop.



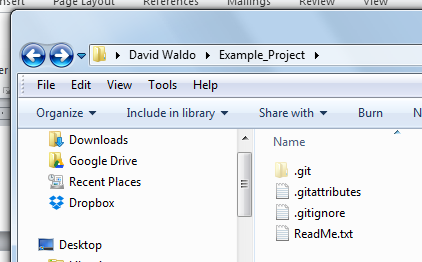
* First, make the folder into a Git repository that is empty. Do this by clicking the + button and selecting Create and filling in the information that matches the folder that you created. Notice the Local path should be the same as the directory you created.

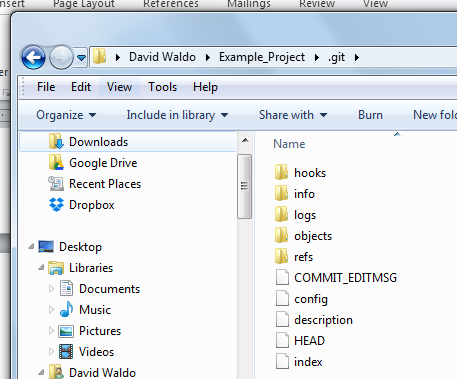


* Click Create repository at the bottom.



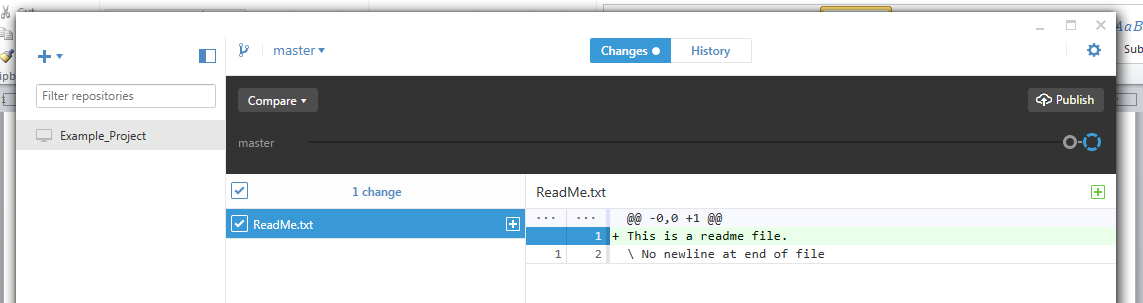
Notice that it automatically adds the .gitattributes and .gitignore files to the directory. In the directory itself you will see the .git folder and its contents.



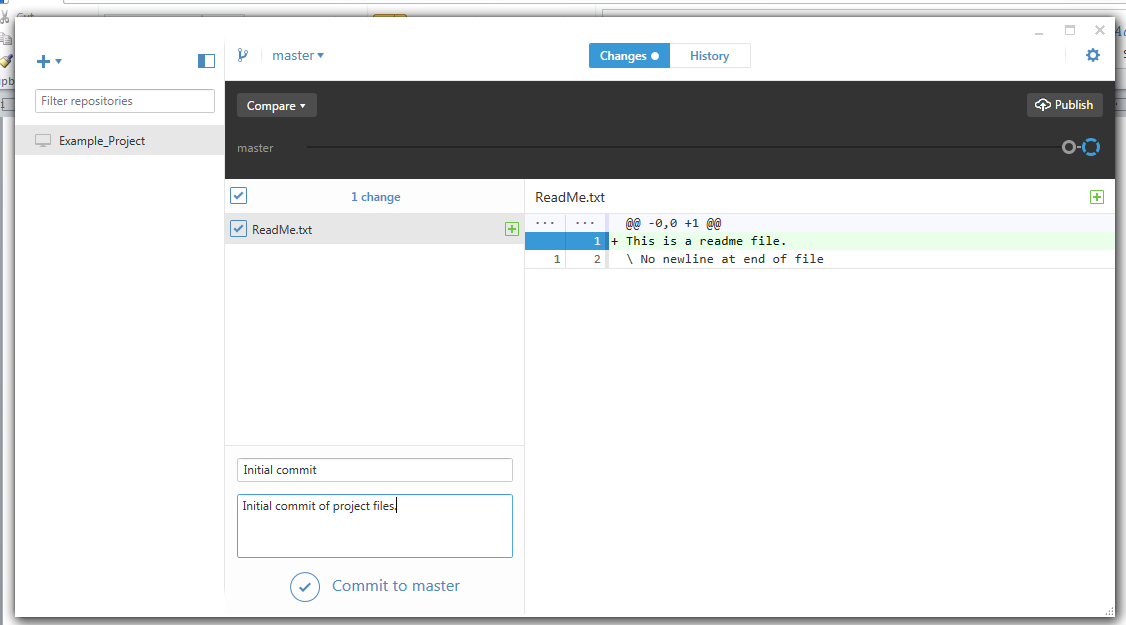


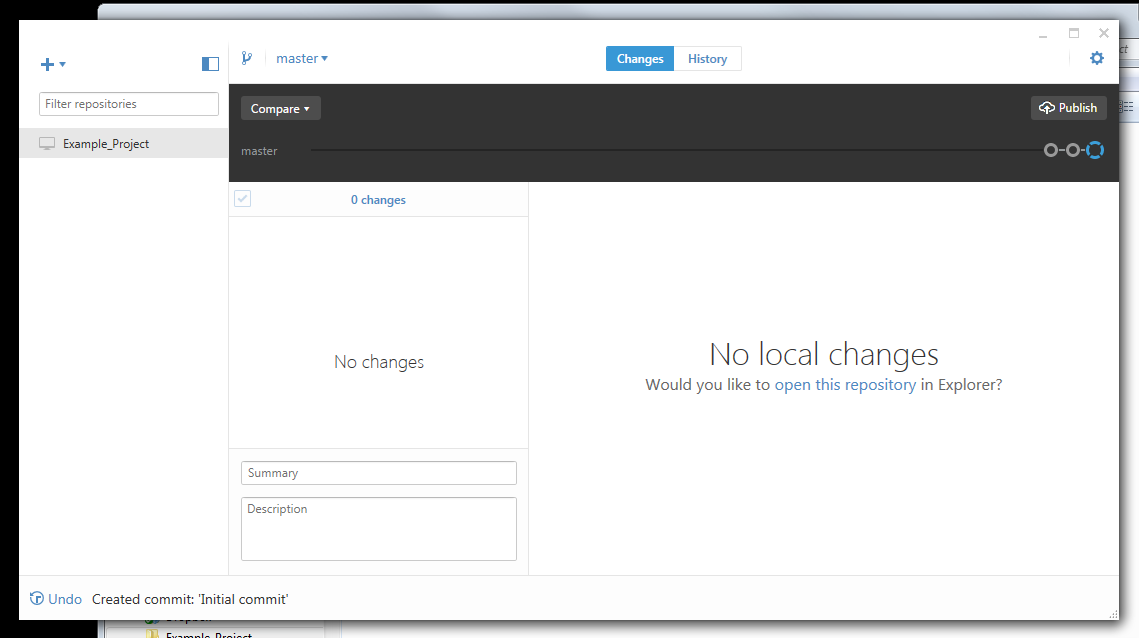
In the GitHub Desktop interface the green + sign means they have been added to the project. If you hover the cursor over the + sign it will say Added.

Notice the timeline for the master branch. It has a blue circle and a circle on the end. The blue circle shows a point in time where a command was executed. In this case just the two files shown were added to the project. If you click on the circle at the end you will see the current state of the project. Notice that the ReadMe.txt file has been added to the project and it is showing a line has been added to the file (in light green).



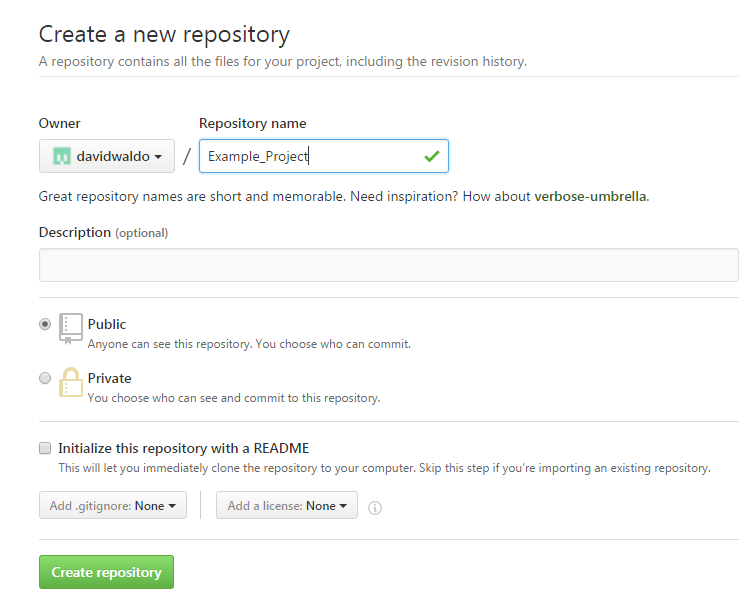
* Once you have made changes and want to save a version of the files, you need to commit the files. At the bottom type in a summary and description of the changes and then select Commit to master.

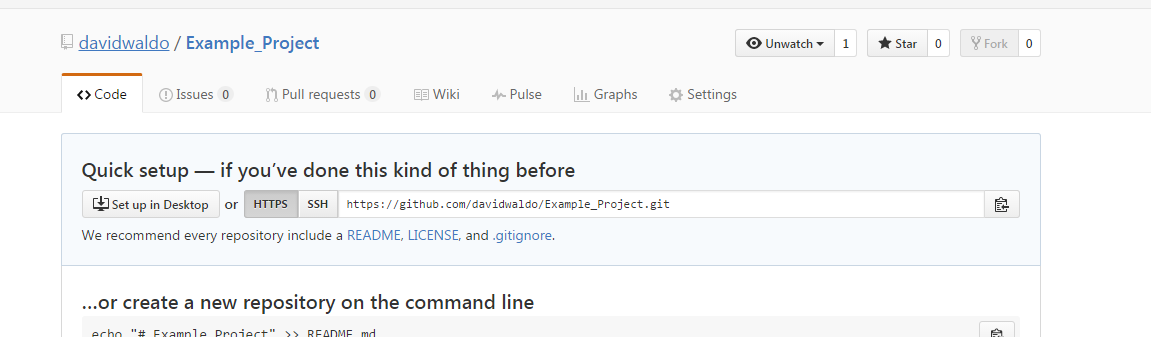




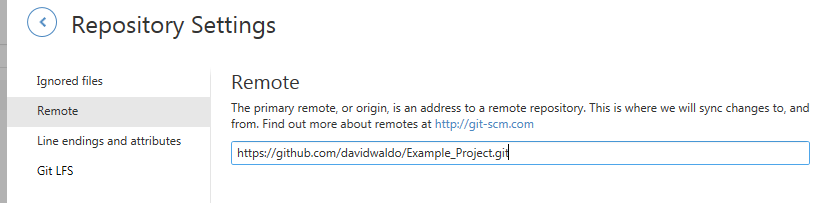
Now it shows that no changes have been made to the project since the last commit. You can click on previous circles and see what changes to the project have been made.

* All of this so far is on the local computer. Now it is time to push it to a project on GitHub. Login to GitHub and create a new repository. You may want to name it the same as the project folder name.

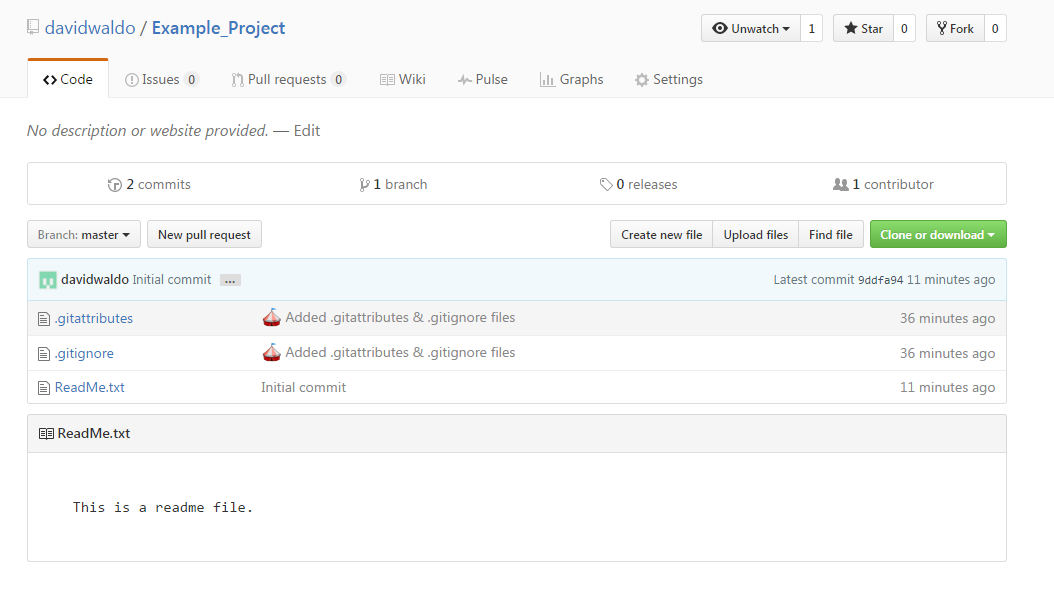




* Now go back to GitHub Desktop and make changes to the repository settings. Select the gear shape in the upper right and Repository Settings. It will initially show the ignored files (.gitignore). Click on the Remote tab and put in the name of the remote repository. For me it was <https://github.com/davidwaldo/Example_Project.git>. You can get the link from the Quick setup box. DON'T FORGET THE ".git" ON THE END OF THE LINK. Click Ok at the bottom.



* To push the changes to the remote repository click the Publish button. It will say Syncing for a little while and then change to just Sync.
* Check the remote repository by refreshing the remote page. Notice it says 2 commits. That was the initial create when the two files were added and the commit we did when we added the ReadMe.txt file.



Now the project is versioned locally in a local repository and stored in a remote repository.

## .gitignore

Git can be used to control Keil MDK projects. If you want to use Git you should install a Git command line program and then read the application note "Using Git for Project Management with uVision" <http://www.keil.com/appnotes/docs/apnt_279.asp>.

You can also use GitHub Desktop to control Keil MDK projects. You would use GitHub Desktop the same way that is described above. However, you would probably not want to track changes to certain kinds of files like object files and backup files. This is where the .gitignore files comes in.

The file .gitignore can be used to specify file names to ignore by Git. You can use wildcards in the specifications to indicate the types of files. Here is a listing of .gitignore that can be a starting place for the Keil MDK projects.

\*.axf

\*.htm

\*.Inp

\*.map

\*.tra

\*.dep

\*.\_\_i

\*.crf

\*.d

\*.o

\*.lst

\*.lnp

\*.bak

\*.Vishwas

\*.bin

\*.fed

\*.\_2i

\*.l1p

\*.l2p

\*.iex

.#\*

\*.uvgui.\*

\*.uvopt

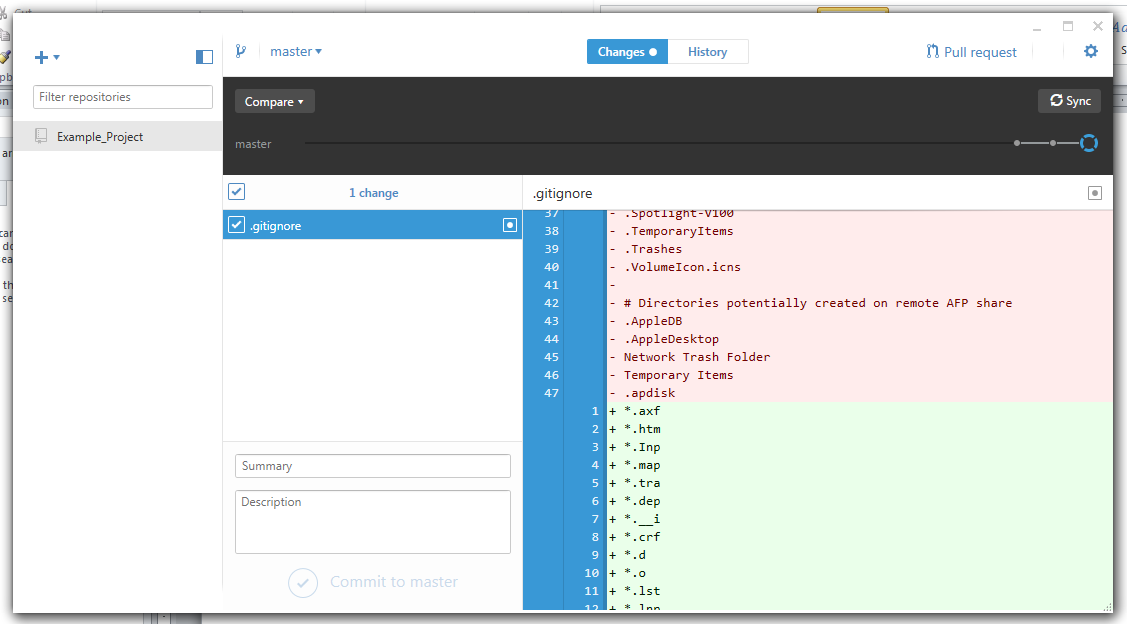
\*.c.\*

\*.o.\*

\*.s.\*

For other types of projects you can use a standard .gitignore file or start from scratch and add your own types.

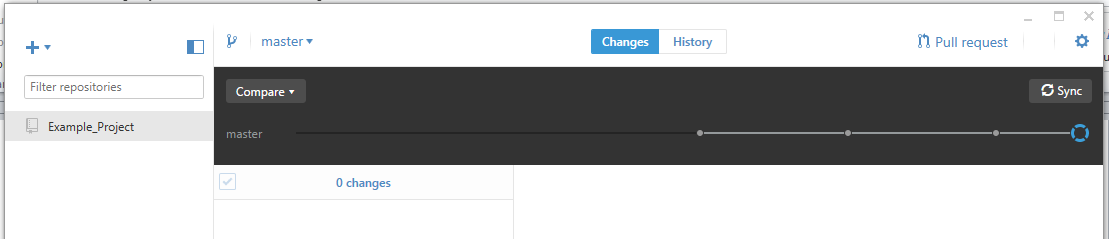
In GitHub Desktop with a repository open, select the gear shape in the upper right and Repository Settings and the Ignored files tab and paste in the above list or enter your own list. Click Ok. This will change the .gitignore file. Notice now the Changes tab and the .gitignore changes.



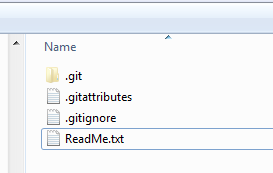
## Branching and Merging

Read the section 3.1 in the documentation for Git about [Branching Basics](https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell). Also read 3.2 in the documentation for Git about [Basic Branching and Merging](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging). These sections will be used to demonstrate how to use GitHub Desktop for branching an merging.

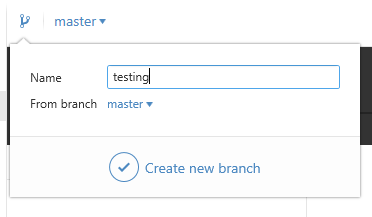
This picture shows the state of the repository Example\_Project on the computer:



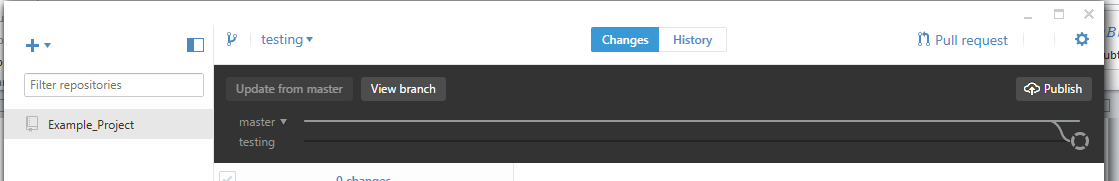
Note that the branch is master and there have been a few commits. This directory just contains the Readme.txt file and the Git files.



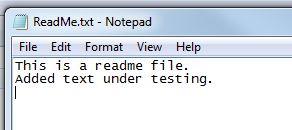
* Add a new branch called testing.



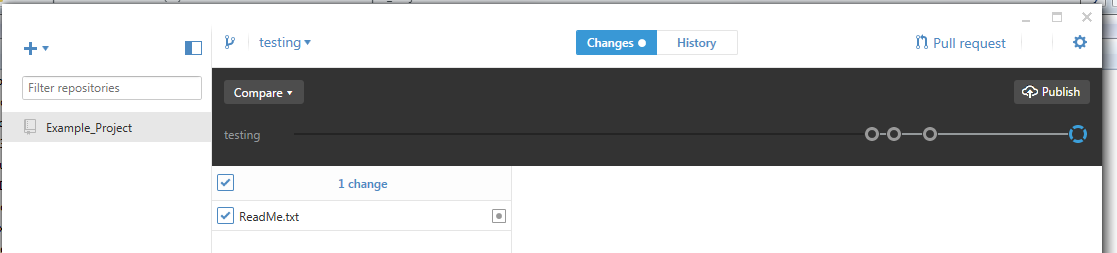
You can see the branch on the time line. Note that testing is selected as the current branch.



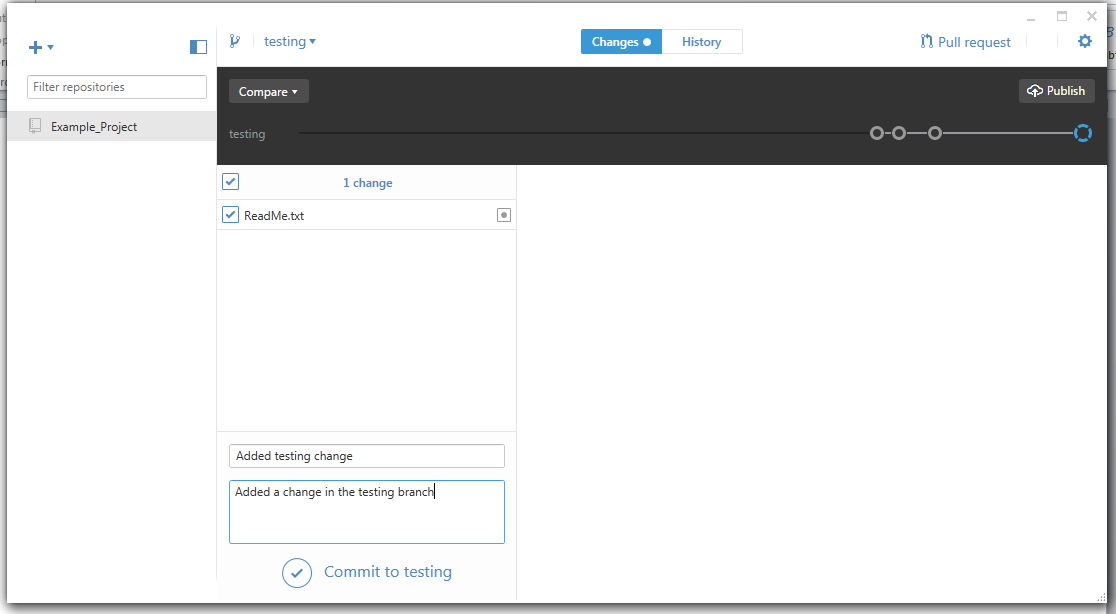
* Add some text to the ReadMe.txt file while in the testing branch.



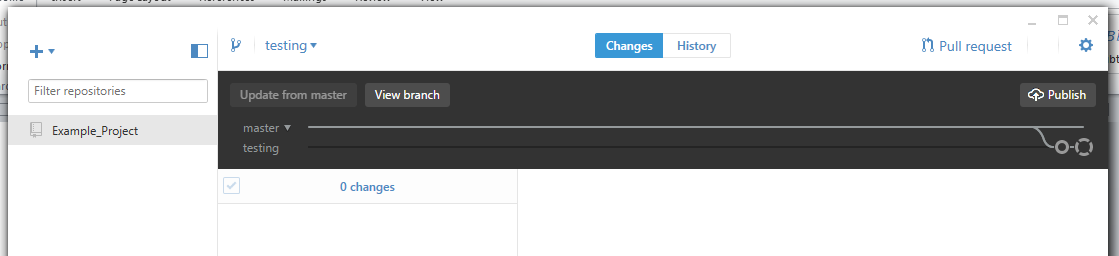
Now GitHub Desktop shows ReadMe.txt has been changed.



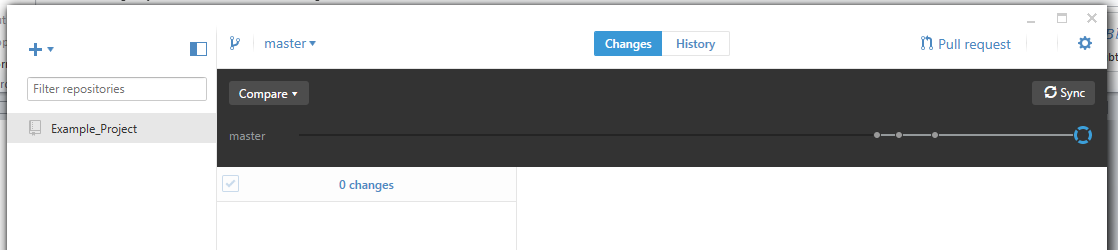
* Commit the change to the testing branch.



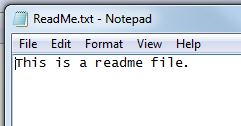
This shows the comparison with the master branch.



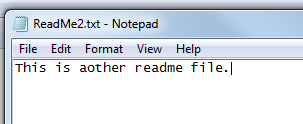
* Change to master branch by selecting the dropdown menu.



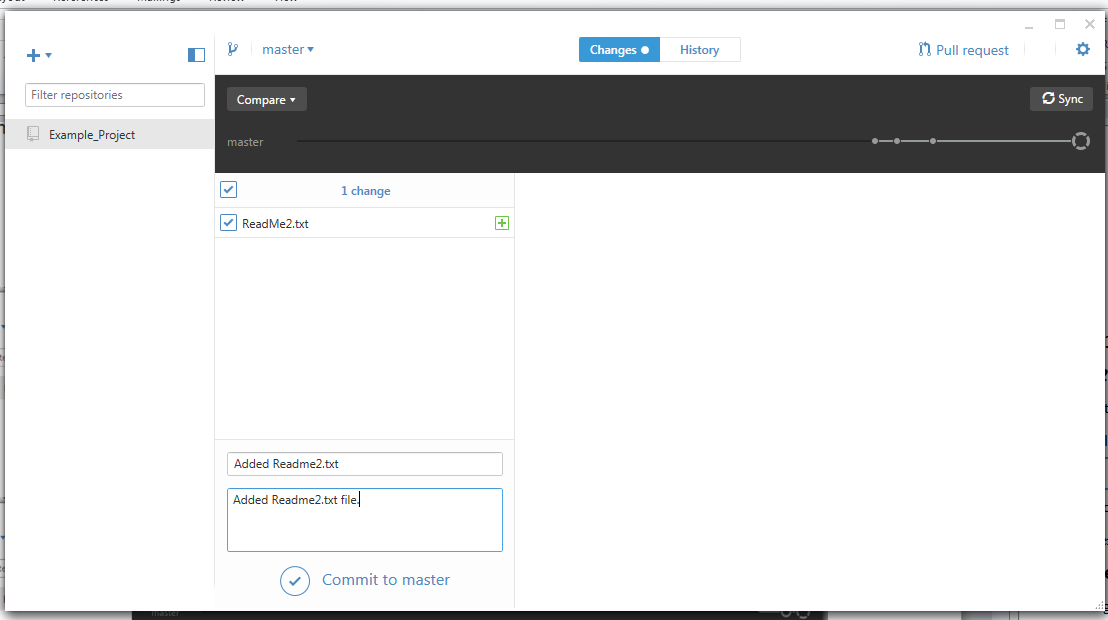
* Notice that if you open the ReadMe.txt file the change that was made on the testing branch is not there (as expected). Close the file without changing it.

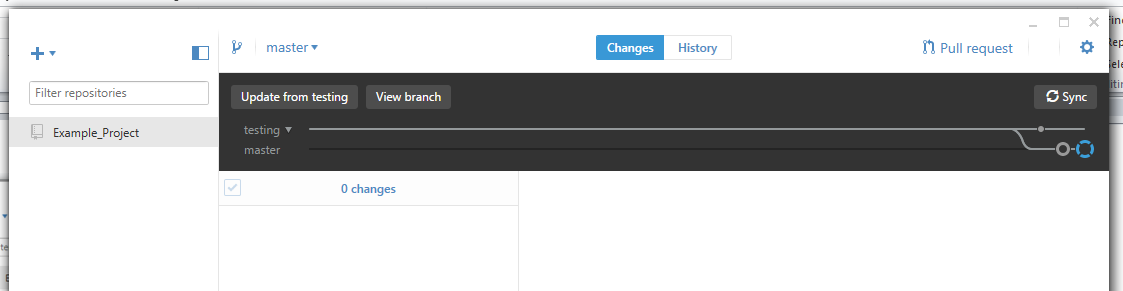


* Add another file to the directory. Call it Readme2.txt and put some text in it.

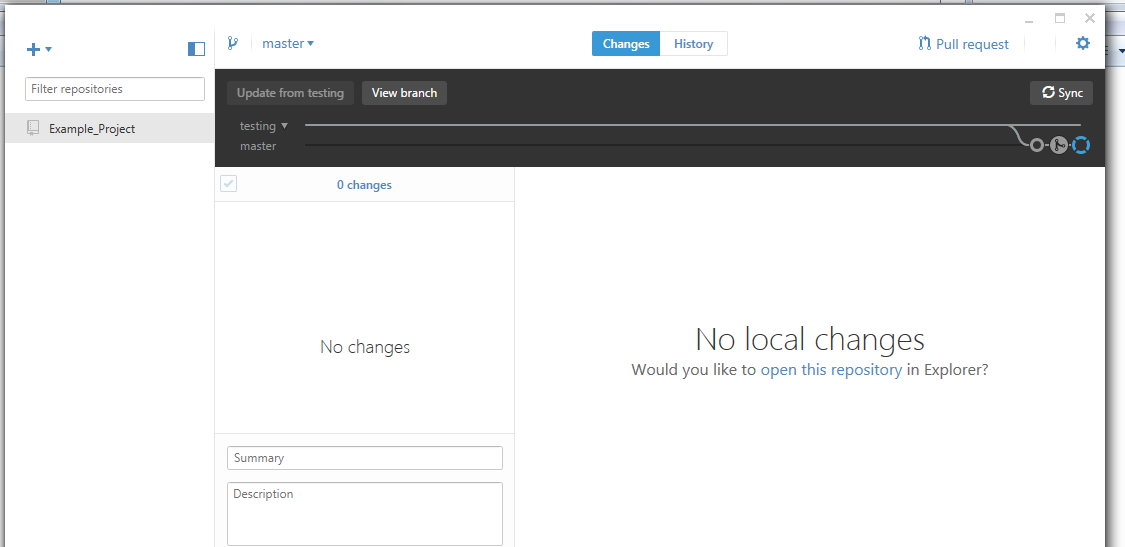


* Commit the change to the master branch.





* Merge the branches into master branch. This is done by having the master branch selected and comparing to the testing branch. Then select the Update from testing button. This will merge the testing branch changes into the master branch.



Now ReadMe.txt has changes from testing and the ReadMe2.txt file is also in the project.

* To sync the master branch with GitHub click the Sync button. Only the master branch is synced. To get testing branch on GitHub you would need to Publish it by selecting the testing branch and then clicking Publish.

