**Introduction and Purpose**

My intent is to use version control for my projects for Arduino, Raspberry Pi and other documents for personal use. Benefits: version control, sharing between multiple computers (PCs and RPis). For free Git, I must be willing to let the public access the repositories. Git is complicated enough that several sessions and continued use of the tool seems necessary to be fluent in its use.

## References:

* [git -- fast-version-control](https://git-scm.com/) Main page introducing version control with local private git repository and remote public [GitHub](https://github.com/) repository. Links to downloads for Windows, Linux and Mac OS X.
* [BitBucket](https://www.atlassian.com/software/bitbucket) by Atlassian. Remote cloud repository that is private.
* See git above for [GUI clients](https://git-scm.com/downloads/guis). "Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience." [SourceTree](https://www.sourcetreeapp.com/) is in use with notes below. Also installed is GitHub Desktop with my raw notes found below (seems harder to learn). For Linux, [GitKracken](https://www.gitkraken.com/) has been recommended by forum and is useable on both Ubuntu Linux and Windows (unlike SourceTree which doesn't run on Linux)
* [Getting Started with Git using Source Tree](https://www.youtube.com/watch?v=UD7PV8auGLg) (5 part video series on YouTube)
* Design News CEC course *Code Sharing, Collaboration and Version Control: An Hands-on Introduction to Git and GitHub* by Charles Lord. 15Nov16. Source tree as integrated with NXP/Freescale development tools.
* Sams *Teach Yourself Python in 24 Hours*, Chapter22 "Saving your code properly through versioning."
* Command Line Interface Reference and Tutorials (Use only if GUI clients fall short!)
  + Git User Guide [ProGit-en.1084.pdf](../Git/progit-en.1084.pdf)  (Instructions use Command Line instead of GUI)
  + Several ways to learn Git exist and I find conflicting information in them. Two were used here: ProGit (Git Basics chapter) and gittutorial(7) ManualPage reached from Git CMD by typing 'git help tutorial' link to file in git *for Windows* folder: [file:///C:/Program%20Files/Git/mingw64/share/doc/git-doc/gittutorial.html](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\gittutorial.html)

**GitHub and SourceTree - Steps for Creating a Repository**

These notes were created while learning document version control using GitHub and SourceTree tools. For introductory information see Design News CEC course *Code Sharing, Collaboration and Version Control: An Hands-on Introduction to Git and GitHub* by Charles Lord. 15Nov16.

* **GitHub** is a web user interface to remote repositories owned by me. The contents are publicly searchable. For private repositories, consider using **BitBucket**.
* **SourceTree** is a user interface to the local Git repository and Push/Pull/Fetch capabilities to the remote GitHub repository.

Assumptions for these notes:

* Source Tree is loaded and Git is selected as the local repository program.
* SourceTree is not integrated with any other development tools on the local machine. Source tree is 'stand alone' and is used to perform version control on any folder on the local machine.
* I am the only user of the repository, so SourceTree license is free.
* A demonstration folder containing the project to put in version control exists, with or without any files inside.
* These instructions use files SB1.txt, SB2.txt, SB3.txt, WorkFlowForNewRepository.docx and README.md for purposes of demonstration. The docx file is this document of notes.
* File creation and revision is done in the demonstration folder (working files). (Remote repository revisions use slightly different instructions such as 'Pull' or 'Fetch' to check out remote revisions to the local repository... to be proven.)
* The demonstration repository is temporary and disposable. When appropriate, these notes will be relocated to a 'permanent' repository.

Startup

Initialize remote and local repositories as follows:

1. Create a new remote repo in GitHub using a descriptive name. Do not create Readme.md yet.

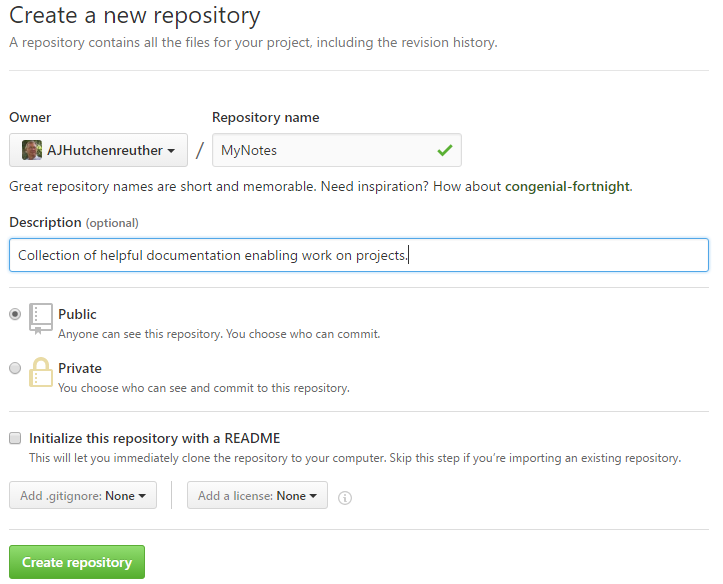


Figure - Example of creation of GitHub repository

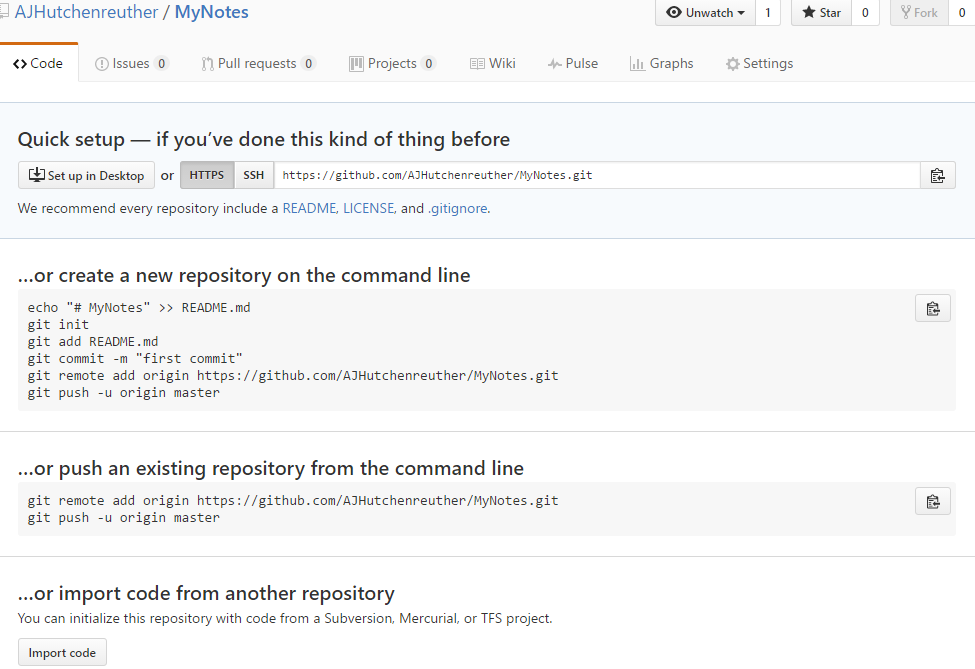
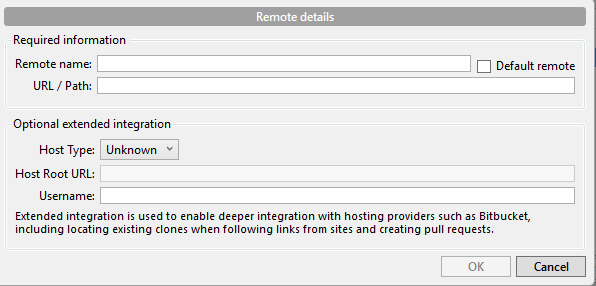
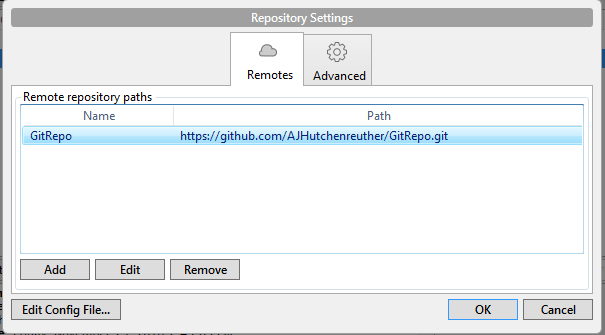
1. GitHub will present choices for populating the remote repo. Take the first one and copy the link to the remote repo to the clipboard. 

Figure - Example of capturing link to GitHub repository. (MyNotes is replaced with GitRepo in following text!)

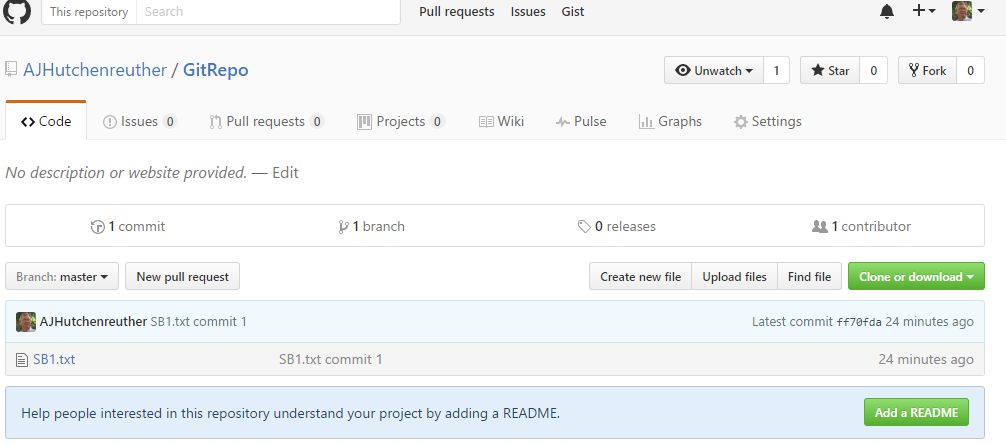
1. With SourceTree, Create a new local repository (repo) on PC. Designate the demonstration folder, GitRepo, as the location of the local repository.
2. In SourceTree, complete Remote Details to link the local repo to the remote repo. Use the command Repository.Repsitory Settings.Add and paste the remote link which was placed earlier on the clipboard in the URL/Path: field. Type in remote repo name in the Remote name field.



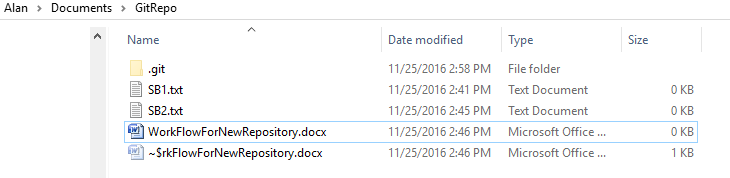
1. Optionally, specify Host Type (GitHub), Root URL: www.GitHub.com and Username.
2. Click OK to Remote Details and view this dialog:



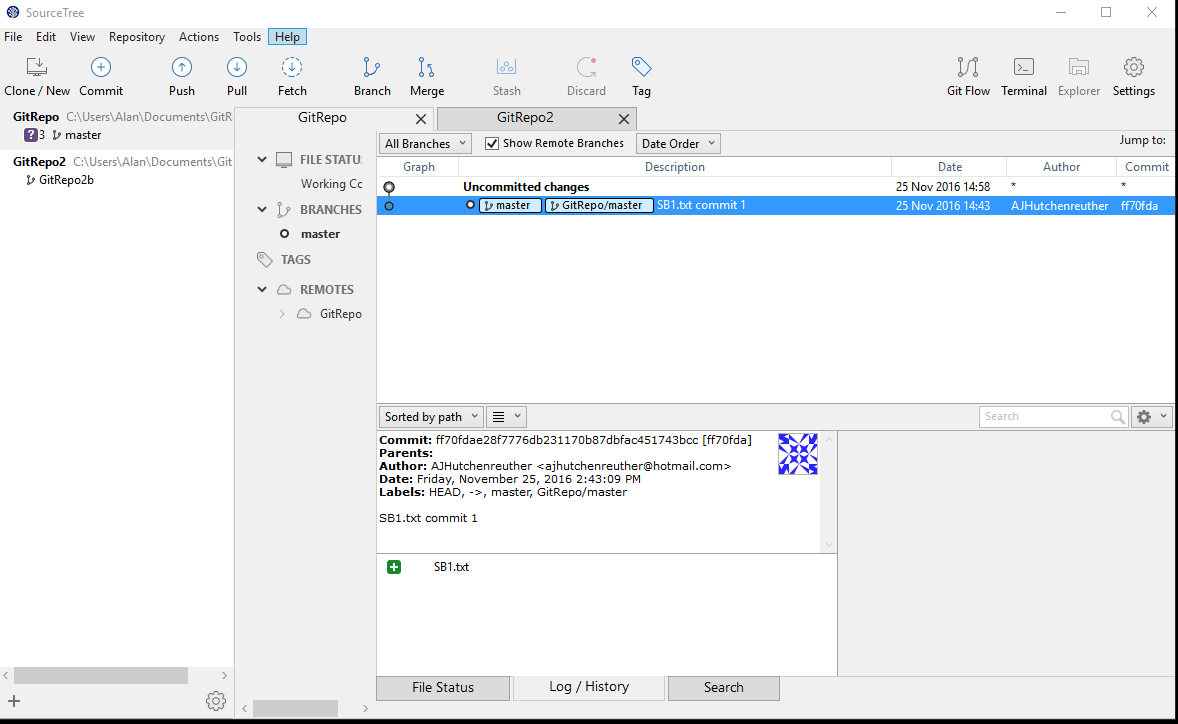
1. The repositories are now created and linked. Now add files to the local repository folder by moving, copying or Windows quickmenu 'New'. In this illustration, after adding SB1.txt, SourceTree will show 'uncommitted changes' in the repository menu, with SB1.txt in the Unstaged files panel.
2. Stage the file, and click on the Commit action on the command ribbon. Add a descriptive comment at the bottom of the SourceTree dialog and click on the Commit button. This will create a 'master' branch.
3. Use the SourceTree action 'Push' and select the desired file(s) to send to the GitHub remote repo. GitHub will display the remote repo contents when Code is selected (1 commit, 1 branch in this example):



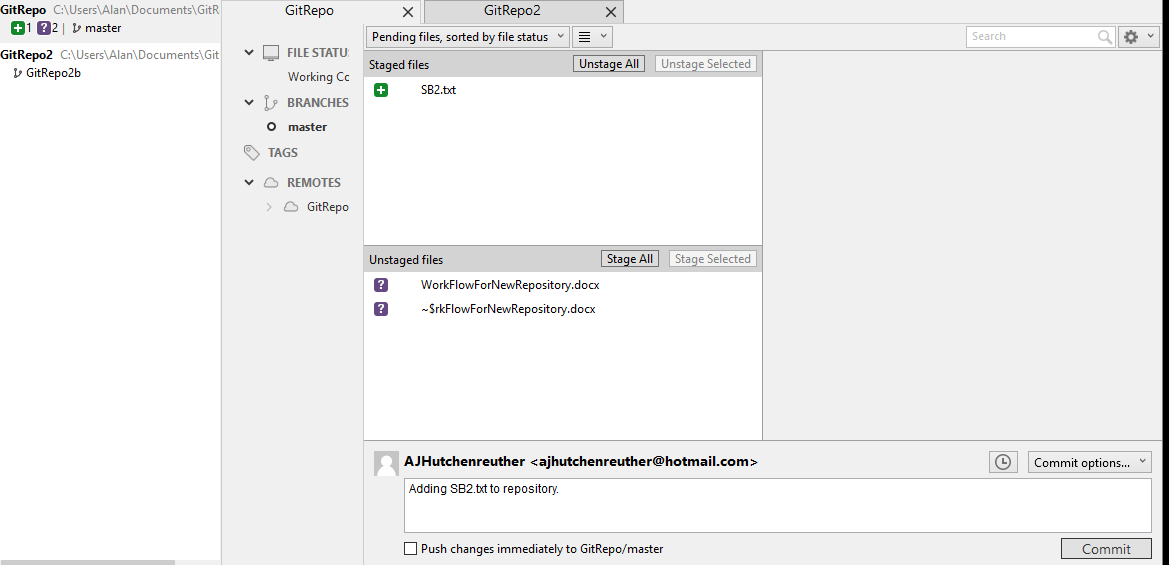
1. Add a couple additional files to the local repo folder. For example SB2.txt and WorkFlowForNewRepository.docx. Windows Explorer will display this for the example:



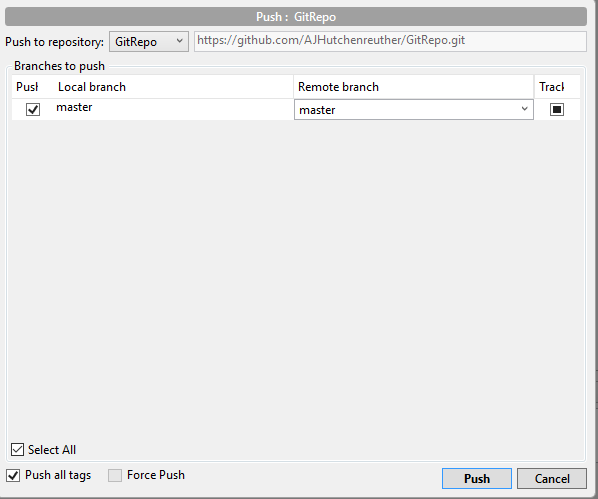
1. SourceTree will display **Uncommitted changes** like this:



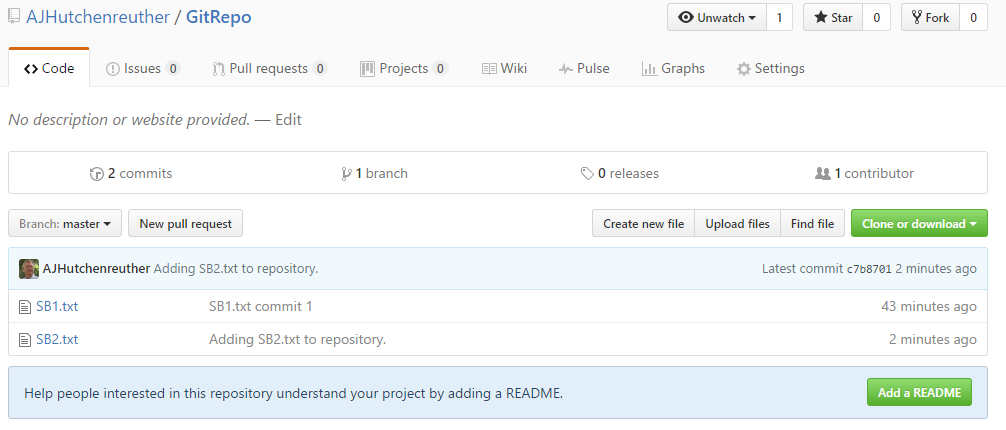
1. To add one of the two new files to the remote repository, do SourceTree Commit on the command ribbon, select file to Stage and click on Stage Selected. Finally add comment to commit comment field, and click the Commit button. This is what will be seen:



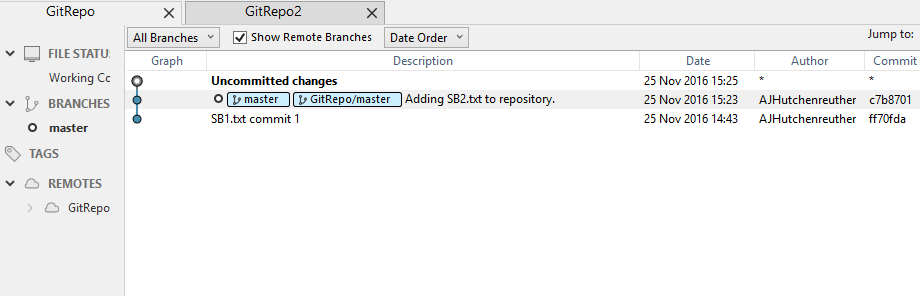
1. Now, in SourceTree:
   1. the list of changes displays all comments I entered, branch and tag names, date/time.
   2. The lower half of the window lists what files were committed, and Labels.
   3. The file list allows file to be selected and differences displayed.
   4. Quick menu allows open of current or selected version of the file (among other actions)
   5. Git saves all changes somewhere. Previous versions of a file can be opened for review.
2. Click on SourceTree Push action to check the newly committed file into the remote repo. SourceTree will give you a choice of target remote branch, pictured here:



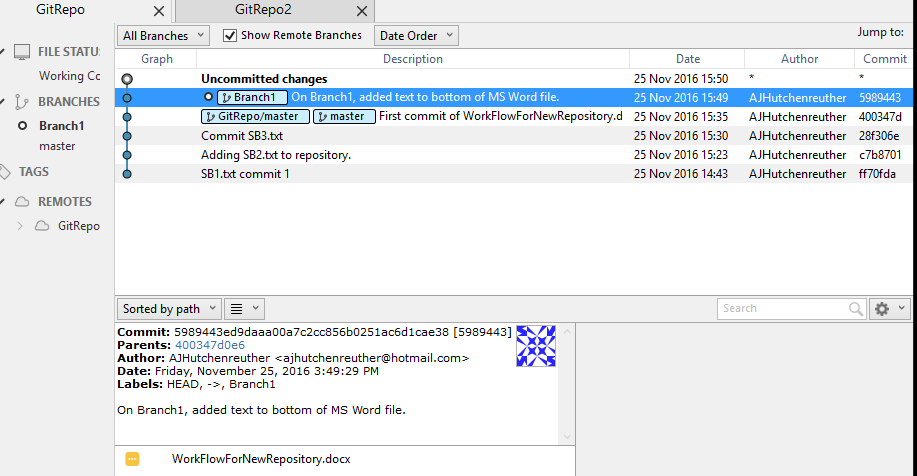
1. On GitHub, this should be seen after forcing a display update by clicking on 'commits' and then 'Code':



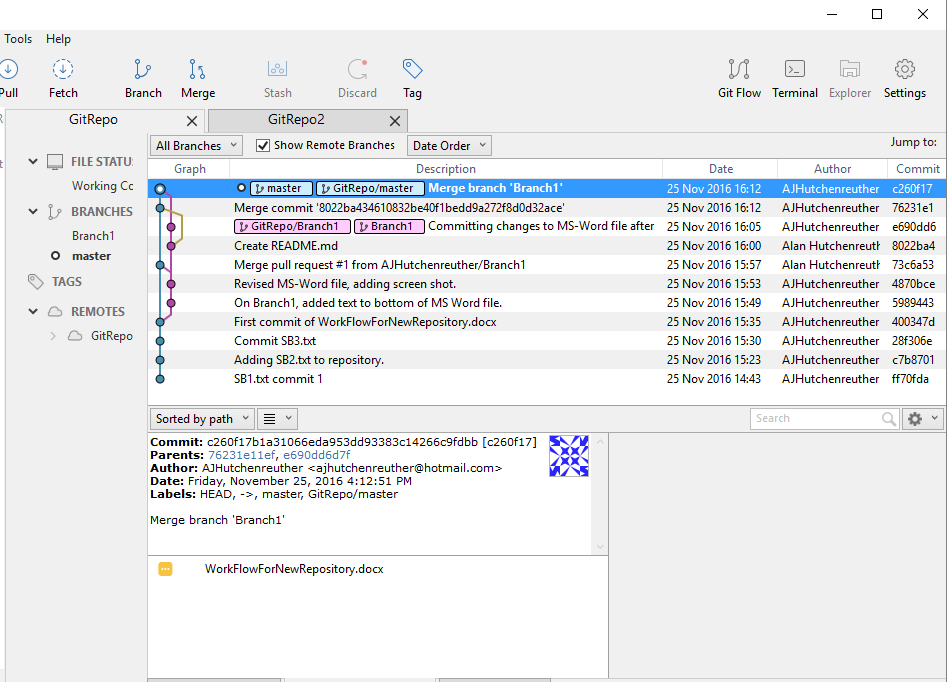
1. SourceTree displays this:



1. Repeat with new file SB3.txt but don't show all the steps!
2. Finally, commit this MS Word file. In GitHub, click on 'commit' and 'Code' to see the Word file displayed. Further revisions here will be done on a branch that will be merged into master.
3. Revised MS Word. Commit to Branch1 (local only, did not Push to GitHub). Re-open MSWord and see this in SourceTree: 'Branch1' in left of window... this should be selected for commit/push/branch/merge operations on Branch1 documents.



1. Using GitHub, selected Branch1, created Readme.md and modified the MS-Word file. Conflict with changes in local and remote repositories?
2. Pulled ReadMe.md from remote to local repositories using SourceTree Pull action button.
3. No conflicts created. Did a bunch of merges, pushes without error. Did not make changes directly to the MS-Word document in the remote repository. SourceTree looks like this:



## Miscellaneous usage notes:

1. GitHub: To delete a repository, open it, go to Settings, and at bottom of page select 'Delete this Repository' . Beware: deleting a remote repository may also delete the local repository! (To be tested.)
2. To understand the difference between Fork and Branch, Google search GitHub fork vs. branch!
3. GitHub: Use of Github.io web server to create a web page. Introduced in Day 4 of the Charles Lord's course.
4. Local repositories may be created where ever document folder happens to be located. Not all documents in a folder need be tracked. Some can be 'ignored'.
5. Version control on code with common heritage can be done on a project with branches or on multiple projects. For example, see Arduino Datalogger\_BaseCode application and derivatives. Currently, I'm thinking of keeping separate projects for current derivatives.

After installing git on Windows (follow the second link above), find on the Windows start menu: Git CMD. I copied this to a Windows 'tile'. In Windows All Programs, find Git GUI and Git Bash.

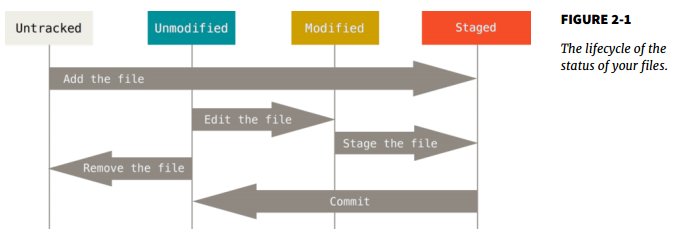
My learning initially:

I think of learning git as analogous as 'eating an elephant'. It must be done one bite at a time. This environment is different than SVN and Polygon version control software.

Git vs.GitHub: **GitHub** is a Git repository hosting service, but it adds many of its own features. While Git is a command line tool, **GitHub** provides a Web-based graphical interface. It also provides access control and several collaboration features, such as a wikis and basic task management tools for every project. *(I recommend learning and using Git installed on the local computer, NOT from the web version in GitHub. )*

Pro Git is good reference. A PDF version is stored on my PC in Documents/Git.

Be familiar with the states of a file as represented here:



A repository holds a project on the GitHub server. It is the best path for syncing or transferring a project or library between machines. I have a free GitHub account. In free accounts, all repositories are public... careful what goes in them!

*A repository should be created for each project, within the local project folder*.

The local project folder holds the working file(s) and a .git directory to hold staged status.

If a working file is modified with respect to the respository version it moves from Unmodified to Modified.

If you Stage the file, it becomes Staged and awaits your Commit command to put it in the repository.

Libraries stored on GitHub can be *downloaded* or *cloned*. Example on the Adafruit website, the RTC library allows this. *Cloned* gives the user the ability to Commit his changes to the public Git repository. (This may be dangerous!) I think I will stick with downloaded libraries for now until I understand how easy it is to corrupt someone else's repository.

**Some Git CMD commands to know:**

git : brings up Help list.

git help tutorial

git help

git help everyday - useful minimum set of commands opened in browser.

git config - (looks complicated. Use of Git GUI has a more visual configuration screen)

cd {project directory} - change directory as expected.

git init - creates a blank repository directory in the project folder. 'index'

git add - see git help everyday

git status

git checkout

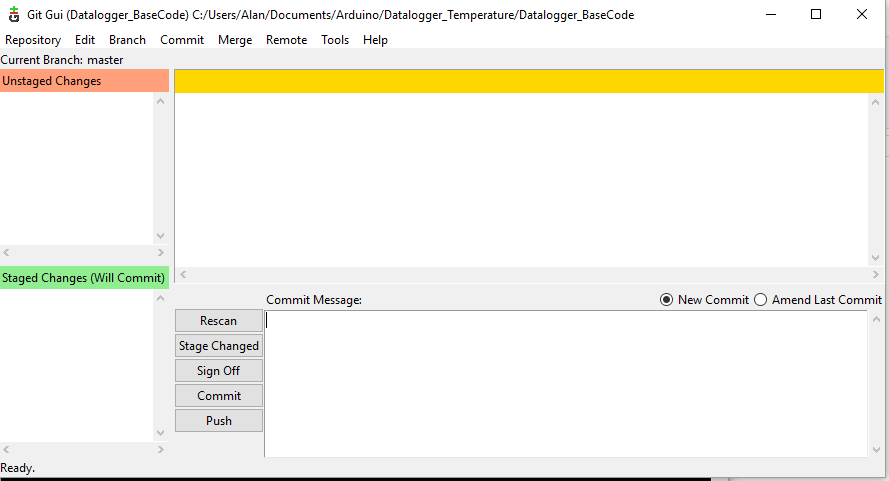
git diff

For **standalone individual developers** these Git CMD commands are most important: (Taken from [giteveryday](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\giteveryday.html#STANDALONE) in ProGit.) Ignore hyphens "-" between git and command!

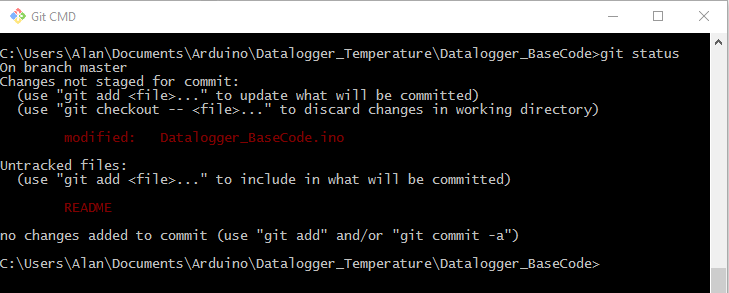
* [git-init](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-init.html)(1) to create a new repository.
* [git-log](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-log.html)(1) to see what happened.
* [git-checkout](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-checkout.html)(1) and [git-branch](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-branch.html)(1) to switch branches.
* [git-add](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-add.html)(1) to manage the index file.
* [git-diff](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-diff.html)(1) and [git-status](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-status.html)(1) to see what you are in the middle of doing.
* [git-commit](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-commit.html)(1) to advance the current branch.
* [git-reset](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-reset.html)(1) and [git-checkout](file:///C:\\Program%20Files\\Git\\mingw64\\share\\doc\\git-doc\\git-checkout.html)(1) (with pathname parameters) to undo changes.
* [git-merge](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-merge.html)(1) to merge between local branches.
* [git-rebase](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-rebase.html)(1) to maintain topic branches.
* [git-tag](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-tag.html)(1) to mark a known point.

Git GUI

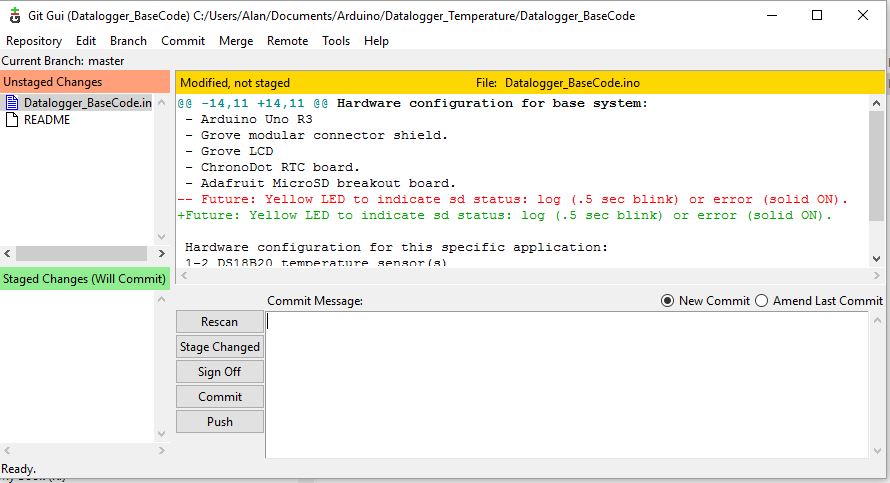
Typical GUI presentation for a project where there are no modified files.



Result of Git CMD command *git status* after modifying a file and adding a new file.

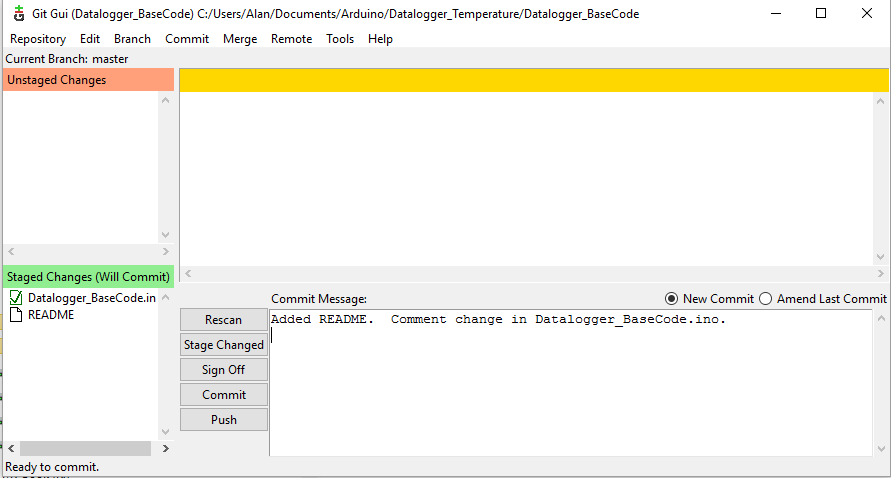


Display of this information using *Git GUI*:

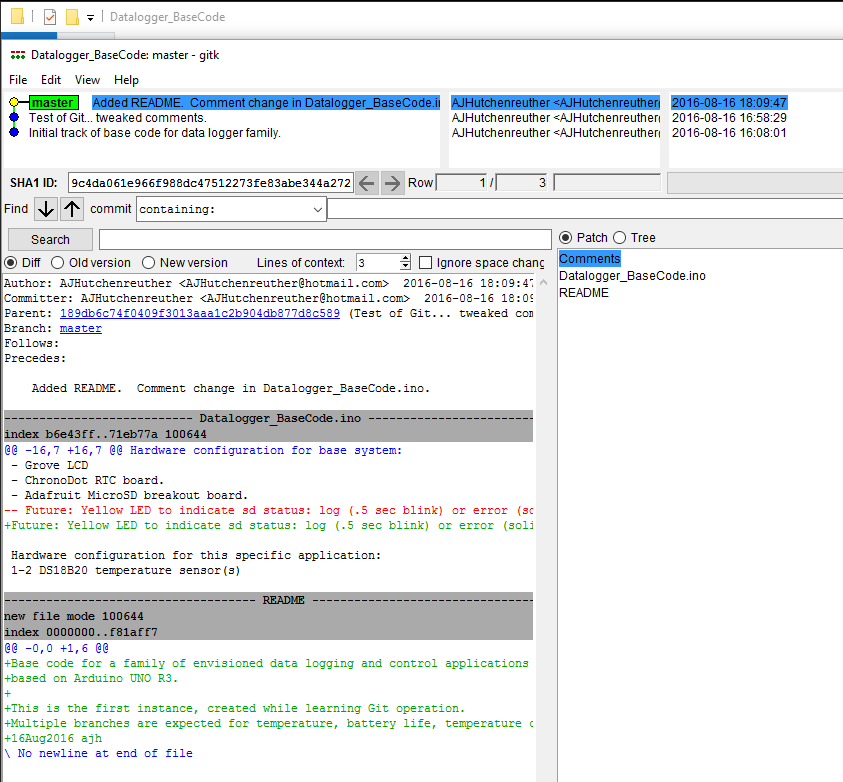


Selecting a file in the unstaged changes list will result in a display of the differences on the right.

Rescan and Stage Changed button will move the unstaged files to the staged changes list. Also README status will change from Untracked to Tracked. Add a *Commit Message* and press Commit to put in repository.



After commit, if you select git GUI, Repository.Visualize master's history, the following window displays: (Pretty cool!)



Right clicking on a version in the upper left brings up a number of commands for recovery of prior versions including 'cherry-picking'.

1. Test Git on laptop and RPi for syncing applications.
2. Do Repository for yogurt maker on GitHub.
3. Create separate repositories for data logging applications.
4. Identify key Arduino applications.
   1. Git init on key applications.
   2. Git Add
   3. Git Commit
5. Learn more about collaborating: *commit* vs *push.* Also *Fork*.