GIT Tut-tut-tut

A Quick Note: *This tutorial makes the assumption that you have already created a repository on the github.com site. If not, please go back and do that first.*

**Creating a Simple Repository**

1. Connect to GitHub (*ssh –T* [*git@github.com*](mailto:git@github.com))
2. Set up *git config –global user.name “Your Name”*
3. *Git config –global user.email [your email]*
4. Create a directory for your project (*mkdir ~/[project]*)
5. Change to your new project directory using *cd ~/[project]*
6. Use *git init* to make your new project a repository

**Adding and Committing Your Files**

1. Add/Create your files in your new project’s directory
2. Once you have your files use *git add .* to add all files to your commit queue (you can also use *git add [file]* if you want to be specific)
3. Great! Now we can commit our files! Use *git commit –m ‘[message]’* and all of your added files will now be added to your repository!
   1. You can also use *git commit –a –m ‘[message]’* to add and commit all of your modified files
4. If you are using GitHub you can now create a repository online by using *git remote add origin* [*git@github.com:[username]/[directory].git*](mailto:git@github.com:[username]/%5bdirectory%5d.git) and then *git push origin master*

Congratulations! You should now have an existing repository on GitHub. A quick workflow would be something like this:

1. *Git commit –a –m ‘[message]’*
2. and then when you are ready to show off your changes use *[git push origin master]*

**Creating Branches**

Another Quick Note: *Branching is the act of creating another commit that is an aside from the “trunk” of your repository. For example: (shamelessly stolen from* <http://www.eecs.harvard.edu/~cduan/technical/git>)*“*Say you are working on a paper. You’ve gotten a first draft out, submitted for review. You then get a new batch of data, and you’re in the process of integrating it into the paper. Halfway in, however, the review committee calls you up and tells you that you need to change some of your section headings to conform to format specifications. What do you do?

Obviously you don’t want to send them your half-baked revisions with corrected headings. What you want to do is jump back to the version you sent out, change the headings on that version, and send off that copy, all the while keeping your recent work safely stored somewhere else.*”*

*To perform the action above, please follow the steps below:*

1. *git branch [new-head-name] [reference to a specific commit]*
   1. Say we have commits A,B,C and we need to create a branch at commit B we would use this command *git branch new-branch HEAD^*
      1. The *HEAD^* references the parent of the current HEAD.

**Committing From Your New Branch**

1. To switch to your new branch use the command *got checkout [head-name]*
   1. **NOTE: You must commit your changes before you run *git checkout.* If you do not then git will start acting wonky**
2. You can now commit your changes as normal**Git Commands to Remember**

**mkdir [project]** – makes a directory

**cd [project]** – changes your location to a new directory

**git init** – use after *mkdir* to turn your directory into a git repository

**ssh –T** [**git@github.com**](mailto:git@github.com) – use to connect to github

**git add [file]** – to add a file to your commit

**git commit –m ‘[message]’** – to commit your files

**git commit –a –m ‘[message]’** – to add and commit all of your files

**git push origin master** – to push your changes to your repository on github

**git status** – see which files have changed

**git log** – to see all of your previous commits

***git branch [new-head-name] [reference to a specific commit]*** – creating a new branch

**git checkout [head-name]** – points HEAD to the commit object specified

**Git Definitions**

**HEAD** – HEAD is

**^** - the caret directs to the parent of the commit. IE: HEAD^ would point to the commit prior to the latest