How to use GitHub and EGit

Integrated GitHub through Eclipse

by Nicholas Berberi (with editing from Christian Gideon)

Make sure you’re using the latest version of Eclipse to avoid problems. You will also need a working internet connection.

Ctrl + Click to view the image links throughout the tutorial. There’s also a glossary at the bottom of the document.

As of now, there is a special order in doing the different tasks. Do downloading egit in eclipse first, then setting up authentication, then committing and pushing code to the repository (up to the all caps SAVE line in the section), then getting the code from the GitHub repository onto your computer, then back to the SAVE bullet point and forward from there.

NOTES (if you have errors): If you get an error on "name the project" (in the Getting the code from the GitHub repository onto your computer section) that says it's not empty, click cancel on the window and go to file, import, existing projects in workspace (under general), then pick the workspace, and momentum should be there. Import it.

If it is not connected to GitHub when you are trying to commit code, then right click on the project, go to team, and there should be an option that says something along the lines of share. Go through that section and you should be good to go!

------------------------------------------------------------------------------------------------------------------------------------------

# Downloading eGit in Eclipse

* In Eclipse: Help 🡪 Install New Software 🡪 “Work with” this link (copy and paste it in) - http://download.eclipse.org/egit/updates. Then press enter (you won’t see anything until you press enter). [See image](http://i.imgur.com/gKjBMA8.png)
* Select the Eclipse Git Team Provider. The other items are optional.
* Then click next and follow the installation process, it should be pretty self-explanatory. Don’t worry if there are any modifications to the original download request.

# Getting the code from the GitHub repository onto your computer

* In Eclipse: File 🡪 Import 🡪 Git 🡪 Projects from Git 🡪 (next) 🡪 Clone URI. 🡪 (next)
* Go to the [GitHub repository](https://github.com/ChaminadeEngineering/2016-Nugget), and copy the ssh link. [See image](http://i.imgur.com/en9G0zz.png)
* Go back to Eclipse, and paste the ssh link in as the URI. (there may or may not be “ssh://” in front of the URI link, don’t worry about it)
* In the window, set the protocol (under “connection”) to ssh. [See image](http://i.imgur.com/S7F6FOB.png)
* You don’t need to change anything else. Leave the user as git and the password blank. Click next. If it asks about the authenticity of GitHub, click yes to continue connecting. If it says “.ssh” and/or “known\_hosts” does not exist, create them.
  + If you’re getting an error when trying to contact the URI, try doing File 🡪 Import 🡪 Git 🡪 Projects from Git 🡪 (next) 🡪 GitHub 🡪 (next) 🡪 Search for the repository (2016-Nugget in this case) 🡪 Select the repository and click next.
* Select the branch you want. Hint: you probably want the master branch. Click next.
* Choose where the files from the repository will go on your computer. Make the location the workspace.
* Don’t change anything else. Click next.
* Import as general project. Click next.
* Name the project. You can just leave it as the default.
* Click finish. You should now have all of the code from the repository on your computer.

# Setting up Authentication

Note: Authentication is per computer, so if you’re using multiple computers you will have to do this again for each one.

If authentication is not set up correctly, you will get an “auth fail” when trying to push code to the repository.

* In Eclipse, go to Window 🡪 Preferences 🡪 General (click the arrows to expand) 🡪 Network Connections 🡪 SSH2 🡪 Key Management tab.
* Generate a DSA Key and copy the whole thing (It’s the box above “Fingerprint,” *not* the fingerprint). The box may be big or [only 1 line](http://i.imgur.com/XrISJmv.png), but the whole key is multiple lines. Make sure to Ctrl-A and Ctrl-C to copy the whole thing.
* Click “Save Private Key…” You don’t need passphrase protection. Save it to your computer as id\_dsa, preferably in your .ssh folder along with the known\_hosts file.
* Go to the GitHub website, and open up your account settings (found by clicking on the little icon in the top right). [Direct link here](https://github.com/settings/profile).
* Click “SSH keys” in the left bar, click “New SSH key” in the top right, title the key (with your name), and then paste the key that you copied. [Follow this map](http://i.imgur.com/pLFdINE.png)
* Now, the key should be tied to both your computer and GitHub account. This authorizes you to use your GitHub account while working locally with your computer – such as interacting with a repository through Eclipse.

# Committing and pushing code to the repository

Once you have finished editing your code, you need to commit and push it to the repository.

* First, open the git repositories view in Eclipse if you don’t have it open already: Window 🡪 Show view 🡪 Other 🡪 Git 🡪 Git Repositories.
  + If you’re uploading code that isn’t part of the repository yet, you can upload files directly from the repository on the GitHub website ([See image](http://i.imgur.com/9RQM1UY.png)).
    - Alternatively, to do this directly in Eclipse, make sure you see “2016-Nugget” in the git repositories view window in Eclipse. If it isn’t, you need to clone the repository. Click on “Clone a Git repository.” [See image](http://i.imgur.com/u87Ne7P.png)
    - Then add the URI of the repository and set the protocol to SSH. Copy the SSH link off of the repository on GitHub, then paste it in the window on Eclipse. Don’t forget to set the connection protocol to ssh. Click next. [Image 1](http://i.imgur.com/en9G0zz.png), [Image 2](http://i.imgur.com/YqSLWsR.png)
    - Select the branch you want. Hint: you probably want the master branch. Click next.
    - Choose where the files from the repository will go on your computer. You can leave it at the default location that it suggests as the directory, or you can change it if you want. Don’t change anything else. Click finish.
    - You should have the repository in the git repositories view now.
* Right click the file (or the whole project if you’re committing more than one file) you want to commit your changes. Go to Team 🡪 Commit.
  + If you only see apply patch and share project as options, click on (Team) 🡪 “Share project…”
  + The repository type is Git. Click next.
  + For the repository, select the repository you want to commit to from the dropdown list (you should have it cloned locally already – if you don’t see it on the dropdown list, see the previous instructions for how to clone the repository). Click finish. [See image](http://i.imgur.com/OtFKcfG.png)
  + You should be able to commit and push files from that project now (go to Team 🡪 Commit to start committing).
* If you’re asked to identify yourself, add your name and email. If you’re using one of the programming computers or any non-personal computer, make sure you uncheck “Don’t show this dialog again.” [See image](http://i.imgur.com/4d1qEe9.png)
* Write a commit message that describes the changes you made. Be as descriptive as possible. Note that you won’t be able to commit without a commit message.
* Make sure the author and committer info is correct; you can change it if you want.
* Select the files that you are committing (the files that you made changes to) at the bottom. [See image](http://i.imgur.com/FdqeFRx.png)
* Click commit and push, or you can just commit them now and push them to the repository later with Team 🡪 Push to Upstream.
* If you’re asked for a username and password to push, the username is breedenpt@gmail.com and the password is Eagles1138.

# Working with branches

When you're working on a project, you're going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not. Branching exists to help you manage this workflow.

When you create a branch in your project, you're creating an environment where you can try out new ideas. Changes you make on a branch don't affect the master branch, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.

* Create a branch in Eclipse by opening the git repositories view if you haven’t already (Window 🡪 Show view 🡪 Other 🡪 Git 🡪 Git Repositories). Then right click Branches 🡪 Switch To 🡪 New Branch… [See image](http://i.imgur.com/khWPE63.png)
  + You can also do this on the GitHub website (not recommended) by typing in a name for it on the on the dropdown box that says “Branch: master ▼” on the online repository. [See image](http://i.imgur.com/jpUGZL8.png)
  + You can navigate between all existing branches from the same dropdown box.
  + However, it is recommended that you make the branch in Eclipse, so that it starts locally first and then you can push it to the repository, rather than vice-versa which can be more complicated.
* Name your branch, and don’t touch any of the other options. [See image](http://i.imgur.com/WV8NBOY.png)
* Notice the small black and white check mark icon on the new branch. This designates the branch you are currently working on (AKA the branch you have “checked out”). You can checkout another branch by double clicking on it.
* When you’ve finished work on your branch, you can create a pull request on the GitHub website repository. Click on the “Pull requests” tab, then click “New pull request.” [See image](http://i.imgur.com/BsbUO3r.png)
* Compare the two branches (make sure the base branch is the branch you want to merge to – this is usually the master branch, and then the branch you are comparing is the new branch that you want to merge). Compare their differences and make sure everything is correct, then click “Create pull request.” Describe the changes that your branch is making, then click “Create pull request” (again). [See image](http://i.imgur.com/11gfgiN.png)
* The pull request should show up in the “Pull requests” tab now. When you open the pull request, you can discuss it with the other people if necessary (have people look at it, suggest changes, etc). When everything is in order, you can click “Merge pull request” to finally merge the two branches. Alternatively, you can click “Close pull request” if you want to cancel it.
* Once the changes have been merged (or the pull request is closed), you can delete the branch since the changes have been applied to the main branch. You should see this option in the pull request page; alternatively you can delete them through the branches tab. [See image](http://i.imgur.com/cM6YOcO.png)

If you get a “non-fast-forward” error when trying to push:

1. In Eclipse, open the git repositories view if you don’t have it open already (Window 🡪 Show view 🡪 Other 🡪 Git 🡪 Git Repositories)
2. Ensure you see your local repository and can see the remote repository as a subfolder. In my version, it's called Remotes, and then I can see the remote project (origin) within that.
3. Look for the green arrow pointing up, this is the 'fetch' arrow. Right click and select 'Configure Fetch'. [See image](http://i.imgur.com/302HIgt.png)
4. You should see the URI, ensure that it points to the remote repository.
5. Look in the ref mappings section of the pop-up. Mine was empty. This will indicate which remote references you want to fetch. Click 'Add'.
6. Type in the branch name you need to fetch from the remote repository. Mine was 'master'. Continue through the pop-up, eventually clicking 'Finish'.
7. Click 'Save and Fetch'. This will fetch that remote reference.
8. Look in the 'Branches' folder of your local repository. You should now see that remote branch in the remote folder. Again, I see 'master'.
9. Right-Click on the local branch in the 'Local' folder of 'Branches', which is named 'master'. Select 'Merge', and then select the remote branch, which is named 'origin/master'.
10. Process through the merge.
11. Commit any changes to your local repository.
12. Push your changes to the remote repository.
13. Go have a tasty beverage, congratulating yourself. Take the rest of the day off.

When updating code, make sure you fetch and merge the changes from the repository before you start working to avoid conflicts/errors. (Git repositories 🡪 Remotes 🡪 right click origin 🡪 Fetch)

Resolving conflicts:

* Select the top level resource showing the red conflict label decorator
* Click **Team > Merge Tool**
* Edit the content of the conflicting resource. Try making the two files identical.
* Tell EGit that the conflict is resolved with **Team > Add to index**
* Commit the conflict resolution with **Team > Commit**

# Glossary

**Branch** - A branch is a parallel version of a repository. It is contained within the repository, but does not affect the primary or master branch allowing you to work freely without disrupting the "live" version. When you've made the changes you want to make, you can merge your branch back into the master branch to publish your changes.

**Commit** - A commit, or "revision", is an individual change to a file (or set of files). It's like when you save a file, except with Git, every time you save it creates a unique ID (a.k.a. the "SHA" or "hash") that allows you to keep record of what changes were made when and by who. Commits usually contain a commit message which is a brief description of what changes were made.

**Conflict** - Sometimes, a commit you get from the upstream branch on the remote repository (repository on GitHub) changes the same line in a file you're editing in your project branch (local repository). This is called a merge conflict.

**Fetch** - Fetching refers to getting the latest changes from an online repository (GitHub.com) without merging them in. Once these changes are fetched you can compare them to your local branches (the code residing on your local machine).

**Merge** - Merging takes the changes from one branch (in the same repository or from a fork), and applies them into another. This often happens as a Pull Request (which can be thought of as a request to merge). A merge can be done automatically via a Pull Request via the GitHub.com web interface if there are no conflicting changes. See [Merging a pull request](https://help.github.com/articles/merging-a-pull-request).

**Non-fast-forward** – an error in which the remote repository you’re pushing to has one or more commits that you don’t have in your local repository. This can generally be solved by fetching and merging changes from the remote repository.

**Push** - Pushing refers to sending your committed changes to a remote repository (such as GitHub.com). For instance, if you change something locally, you'd want to then push those changes so that others may access them.

**Remote** - This is the version of the repository that is on the GitHub website. It can be connected to local clones so that changes can be synced.

**Repository** (also, local repositories) - A repository is the most basic element of GitHub. They're easiest to imagine as a project's folder. A repository contains all of the project files (including documentation), and stores each file's revision history. A local repository is the version of the repository that is locally stored on your computer. You make changes to the local repository from your computer, and then synchronize it with the remote repository on GitHub through pushing and fetching changes.

**SSH** - SSH keys are a way to identify yourself to an online server, using an encrypted message. It's as if your computer has its own unique password to another service. GitHub uses SSH keys to securely transfer information from GitHub.com to your computer.

**URI** – A Uniform Resource Identifier, which can either be a Uniform Resource Name (URN) that names a resource, designating a globally unique, permanent identifier for that resource, or a Uniform Resource Locator (URL) which locates a resource, designating a protocol to use when looking up the resource and provide an "address" (AKA URL link) for finding the resource within that scheme.