Using git:

Before we go anywhere, let me disclose that I use git almost exclusively through a shell client (bash, so the default on terminal in mac and linux). As far as I know, there's no native eclipse terminal (but then again, eclipse isn't my primary dev tool). So this may not turn to be that helpful, unless you choose to use github shell client, but that might not be worth it.

So for the most part, this is just gonna go over vocabulary of git and how it can be used, should you decide to use it in terminal. Whether or not you manage to use it in eclipse or not is an entirely different task.

This document will detail commands related specifically to using git to share work with teammates

LOCAL REPOSITORY

git init

When starting with git, keep in mind that each directory/subdirectory will have it's own git repository, which typically will look like /../workspace/[project]/.git/. If you have more subdirectories, you can create smaller repositories. If you're in a higher level directory, it will include all subdirectories in the repository as well, so /workspace/.git will include every project in workspace in that repository. Sometimes useful for developing very large projects with a lot of assets, like an app.

git branch

One extremely useful feature of git is creating branches. When you first initialise your repository, it'll create the default branch for you, master. You can rename this branch if you like, but it's standard practice to keep it as branch master. This is generally where the project will be committed when it reaches a stable build after some implementations. Creating separate branches will allow for project members to work on an implementation and be able to version control their separate code without worrying about breaking the master branch's code. You can navigate branches using the git checkout command.

git checkout [branchname] to navigate to the branch.

git checkout –b [branchname] to create a new branch and navigate to the created branch

git branch to list all branches in the repository

git add

Using the add command is how you decide which project folders/documents/files/whatever you want to commit to the repository. For the most part, in this project you'll probably just want to commit the entire directory every single time (git tracks deltas, so if there's no change to a file, it won't add it). To do that use

git add . (PUT THAT PERIOD IN!)

If you only want to add specific folders/files, you can use

```
git add [folder/file name].
```

Git add WILL NOT commit to the repository, it just prepares it for travel.

```
git commit
```

Commitment is scary. Jokes aside, this is where you take all those folders and files you added and for reals put it in the repository. Thankfully, if you've been practicing version control, even if you do break the program, you can roll it back easily. When committing, you commit to your current branch

Every commit must be paired with a changelog (please utilise this for better teamwork points). Should be quick and to the point (this changelog is also what shows up in github as the description of the latest push. Pretty snazzy huh?).

USING GIT WITH GITHUB

Okay, this is where the fun stuff happens. Firstly, you're gonna need the URL for the repo. Should just be simple, like https://github.com/[username]/[reponame]. You're ten gonna need to tell your local git that you have a remote repository. For this you use syntax

```
git remote add origin https://github.com/[username]/[reponame]
```

'origin' can be replaced with anything, that's just the name you give to your remote repository so you can refer to it in your git. I use varying names since I don't clear my terminal so all of that persists. It should also persist in your local repositories, but I can't guarantee it (but it makes sense for it to persist).

To pull from the github repository, you use syntax

```
git pull origin [branch name]
```

You can typically omit branch names when pulling master (esp if you only have a master branch). For the most part, you can even omit 'origin' unless you have multiple remotes associated with your local repo.\

Pushing to github!

Probably the thing that's gonna be used the most. Pushing to github is really easy and painless (but the steps to get to this point are pretty drawn out). Firstly, you'll have to already have committed to your local repository and all that if you want your latest changes. Then to push the repo to github, it's simply

```
git push origin
```

It'll then prompt you for your github ID and then password. Put those in, then bam. Pushed to github.

That's a really abbreviated tutorial for git/github. I hope it kinda makes sense? If you have any questions just feel free to ask, or you can google it. Either way.