This might differ for non-Windows, if anything is different (or if anything goes wrong), just ask over Facebook. This is also for the perspective of uploading work to your own fork or branch rather than the project’s master branch directly!

You can download and install Git from here: <https://git-scm.com/>

**Setting up your local repository (local = on your computer)**

* Make a folder where you want to store your copy of the project. You can name it what you want.
* Go inside the folder and right click. You should see two options such as Git GUI Here and Git Bash Here. This doc focuses on Bash which is the command line variant of Git. If you don’t have these options then manually start up your Git from where you installed it and path to here, if you want a point of reference for use it’s very much Minix so remember you can do cd [pathname] and similar commands.
* You should be inside the folder now, use the command **git clone [https.git]**, where you replace the bracketed part with your fork. For example I used:

git clone https://github.com/CosmicScientist/TanfieldRailway.git

It is very much the URL of your fork with .git on the end.

* Now you should see you’re downloading everything from your fork and since our project isn’t large, this shouldn’t take long at all.

**Setting the upstream**

* The upstream is the project you will have forked from, once you set this you shouldn’t need to set it again.
* Do **git remote –v**, this should display your remote links where your *origin* is your fork that you send your work to (this should be listed twice as fetch and push), what we will add should show up as the *upstream* (again listed twice as fetch and push).
* For our project you will only need to do:

**git remote add upstream https://github.com/JoshuaNCL/TanfieldRailway.git**

* Remember to check this worked with **git remote –v**. This lets Git know the project you want to keep in synch with (useful for when it inevitably gets updated!).

**Branches**

* You have a local repository but you can’t just shove work in there yet! You need to make separate *branches* to keep your work separated. Each branch is a different version of the project in your local repository.
* Use **git branch** to check what branches you have, if you just made the repository there should just be *master* listed.
* Use **git branch [name]** to create a new branch with a name relevant to what you’re going to be doing, e.g. git branch addsChangelog, git branch adds\_changelog
* To move to any branch you can use **git checkout [branch]**, remember to use **git branch** to check the names and spelling.
* Any work you do should be made on branches separate to your master branch. This next bit is a little confusing but comes with practise. When adding work to the folder in your choice of file explorer (how you look at files normally), it is added to the branch *that is open in Git*, so you need to (in Git) checkout the branch you want to add to or edit files of. This is tricky, ask for help on Facebook if you’re not sure.

**Pushing and Pulling**

* To send changes to GitHub, you will need to “push” them there. To do this you need to go through a few steps.
* Move onto the correct branch you have work you want to put onto your fork.
* **git status** tells you what files you have edited or created newly for your branch in red.
* Use **git add [file]** for each of these files if you want to send specific ones or if you want to send them all, you can perform **git add -A** which is a shortcut for add all new and updated files to the changes. Use **git status** to check you’ve added the right ones or them all, they will be in green. You can use **git reset** if you make a mistake.
* Use **git commit -m “[message]”** to add a message that is used to label each new load of work you put onto GitHub for everyone else to see. A little humour from xkcd:



* Now you can “push” using **git push** which sends your new work to GitHub, you will need to give your username and password unless you set up a SSH key (more on that later, it isn’t necessary), Git will either display a popup for each or ask directly in the command line. If the branch doesn’t exist on GitHub, Git will tell it to make a new one and you can go and look at it immediately after the files have uploaded.
* You may have noticed at some point Git will complain about lacking an “origin”, if it does, you will have to set your origin (your fork) for each new branch you create. This is a sad fact of life unless your Git is more up to date than mine with a time saving feature but if you find this issue, you can easily get around it (and it should prompt you with the command to do it too) by inputting **git push -set-upstream origin [branch name]**, this only needs to be done once per branch as it essentially sets the origin to being your fork.
* What about pulling? If you want to download an existing branch or edits to the branch that you have on GitHub but not on your local repository, you can use **git pull** to update your branch to be the same as the online version.
* If you want to have the same version as the upstream (the main project), you can do **git pull upstream master** to take from the main project and send it to your master. You can then do a **git push** to send it to your fork’s master branch.

**Making sure your fork and branches are up-to-date (synched with the main project)**

* You should check each time you’re about to start producing or push (send it to GitHub) your work whether or not you’re in sync with the main project (something may have happened, usually it’s that other people have merged changes to the main project).
* If you have your work safely outside of your repository (you shouldn’t keep large changes inside of your local repository, make backups!), you can use **git fetch --all**, which grabs all changes from the upstream (our master project). If you want to be advanced you can use **git fetch --all --prune**, which also deletes any local branches that have been deleted on GitHub (you would delete them on GitHub for when the branch was merged into the master project, which makes their purpose fulfilled and their required existence at an end).

**SSH key? Why would I need that?**

* This helps get around some of the hassle of entering your username and password in every time you push, pull or fetch. I recommend looking this up for yourself (you don’t have to!) by reading up this page (the useful links are in the SSH section) <https://help.github.com/articles/which-remote-url-should-i-use/>

If anything is wrong in this, mention it right away!

If I didn’t cover that one thing you want to know about, mention it right away!

If you found something you want everyone else to know about, mention it right away!