Git Guide for Unity

In this guide I’ll cover the software needed and key bash commands to use Git with a collaborative Unity game.

# Software

The first thing you will need is the git software. This will set up the Bash and GUI (not very useful) to allow collaborative version control. You can download the software here:

<https://git-scm.com/downloads>

The other piece of software which isn’t necessary, but I have found to be useful is Source Tree. Source Tree allows you to do all the normal Git commands in a visual environment. It can be really useful for understanding how the version control system works but I would still recommend using the bash commands 90% of the time because ST can sometimes take 10 minutes to do something that would take 10 seconds from a terminal window. I have no idea why it is so shit sometimes. <https://www.sourcetreeapp.com/>

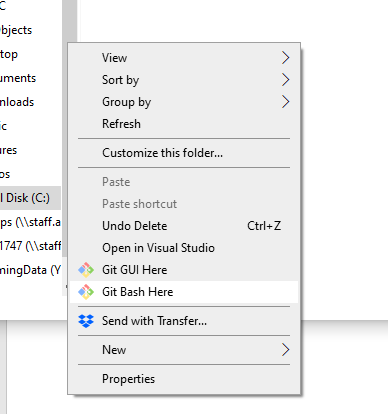
You will have to set up accounts for Github and source tree before moving on.

# Git

## Adding the Repository

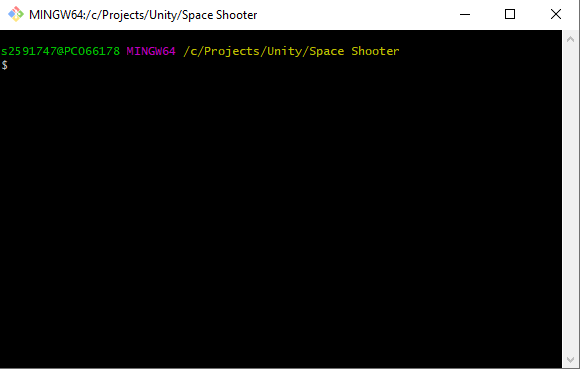
I’ve set up a repo for the project so what you will have to do is init a **local** git repo in the base folder for the unity project and then set the **remote** to look at my github repo.

First go to the Unity Projects root folder (it will have Asset, Library folder, etc). then right click the whitespace in the folder window and select “Git Bash Here”.

This will open a Bash terminal; it is going to be the main way you interact with Git. From here you will need to type the following commands to initialize the local git repo and link it to the remote.

## Initializing and setting upstream

You should see a bash window like this, and it should be pointing to your project root folder.



From here type the following commands:

### Initialize

>> git init

### Set Remote Upstream

>> git remote add origin <https://github.com/Connorrr/UnitySpaceGame.git>

### Get Remote files and folders

>> git pull origin master

*You should now see a gitignore and readme file in your project directory*

### Check Remote Branches

>> git branch -r

*You should now see the branches which exist on the remote repo*

### Setup your local branch and track the remote branch

>> git branch Ben\_Branch

>> git checkout Ben\_Branch

>> git push --set-upstream origin Ben\_Branch

*You will need to login to you GitHub account from here and I may have to add you to the project…. Not really sure about this yet but in the long run it is definitely best to set up an SSH login for Git and Source Tree so you don’t have to keep logging in.*

### Add files, first commit, and push to remote repo

>> git add .

>> git commit -m "This is the first commit but in future these commit messages will say what changes have been made in this commit"

>> git push

From here you will be able to see the work you have done on your branch reflected on the remote repo and I will be able to see it too. (<https://github.com/Connorrr/UnitySpaceGame>)

When we you are spending a lot of time programming then there are 3 main command that you will repeat a lot. They essentially add and manage file changes, commit these changes to the repo with a message that indicates the changes that have been made and then push these changes to the remote repo. They go like this:

>> git add .

>> git commit -m “Added firing mechanic to the ship class with included sound files"

>> git push

You want to repeat these commands often. Essentially every time you add something new and it appears to be working as expected. It’s important that you don’t commit it when it is half finished as you want the commits to act as points in the timeline where you can jump back and know that at this point things were working.

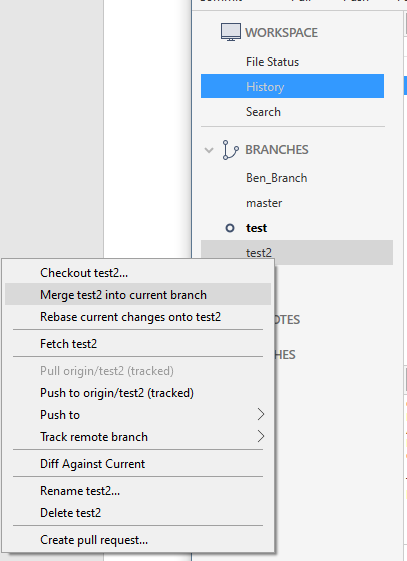
## Branches

The way we will need to handle this is by having our individual branches that we work on (Ben\_Branch, Connor\_Branch) and then merging these with the master branch. This can be slightly difficult because eventually we will change something that will affect the others branch, this is called “merge conflicts” and I have found that Source Tree is more useful when dealing with these than the standard bash window so make sure you have ST connected to your Github account and then add the local repository from inside the ST GUI. ST is pretty intuitive and should automatically manage the changes we have already made.

We are going to find that the most difficult files to manage are the .unity ones. These are the files that hold all the scene information and since managing merge conflicts requires editing the text of the file it’s really difficult to figure out what the fuck is going on. I recommend just taking the most recent version of the .unity file and making a copy of the old one before the merge. It’s a lot easier to re-add stuff to the scene than it is to resolve the merge conflicts on the scene file. This is also why it’s really important to use prefabs as much as possible and mostly generate the scene from a script file if there is going to be a bunch of changes.

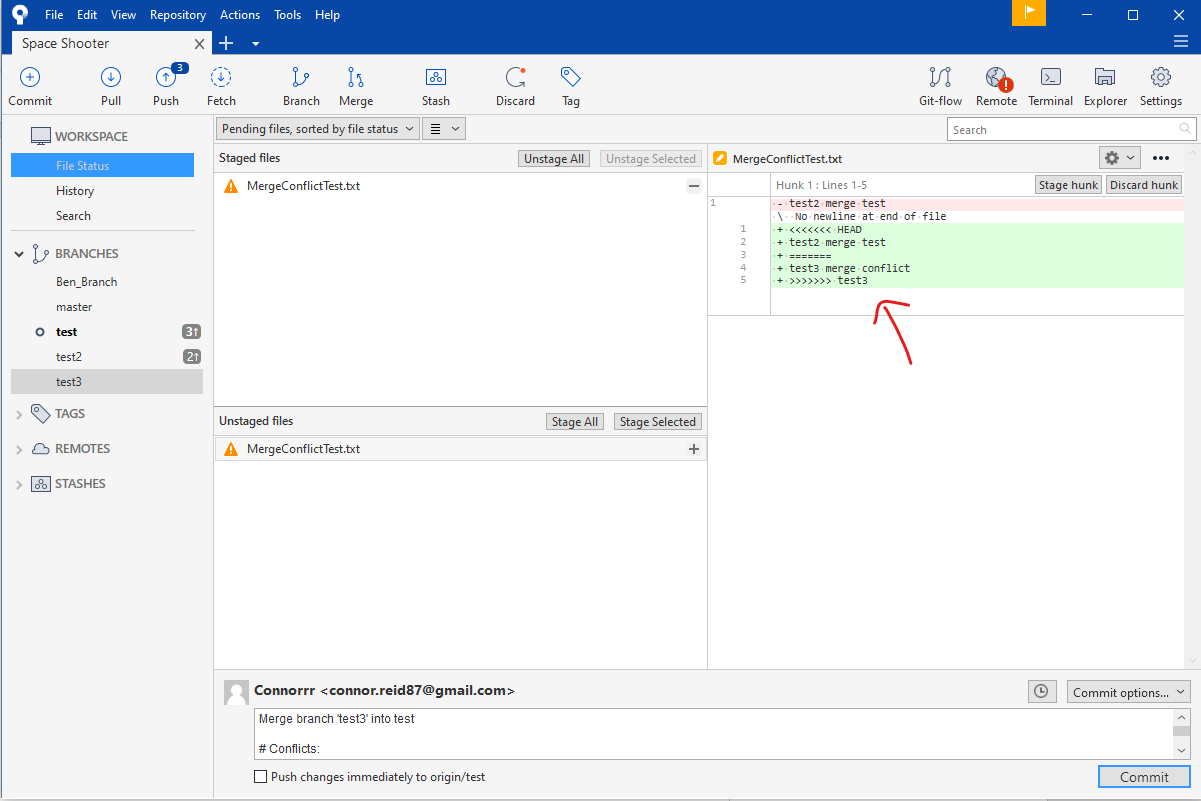
## Handling Merge Conflicts

I’ll show you here how to use Source Tree to handle merge conflicts. I have 3 branches “test”, “test2 and “test3””. In each I have a text file called “MergeConflictTest.txt” and the text inside this file is different for each. I am going to switch to the “test” branch using “git checkout test” and then go to source tree and start the merge process.

On the left panel of ST you can see the branches. I am in test right here and then I right click test2 and hit “merge test2 into the current branch”.

This merge will see that the text in test2 is newer so it will just use the test2 text and merge it into the MergeConflictTest text file but then if we also merge the test3 branch in then this will cause a merge conflict.

After the merge conflict error window pops up you can click on the “File Status” tab and this will show which files have a merge conflict and where this conflict is.



<<<<<<< HEAD

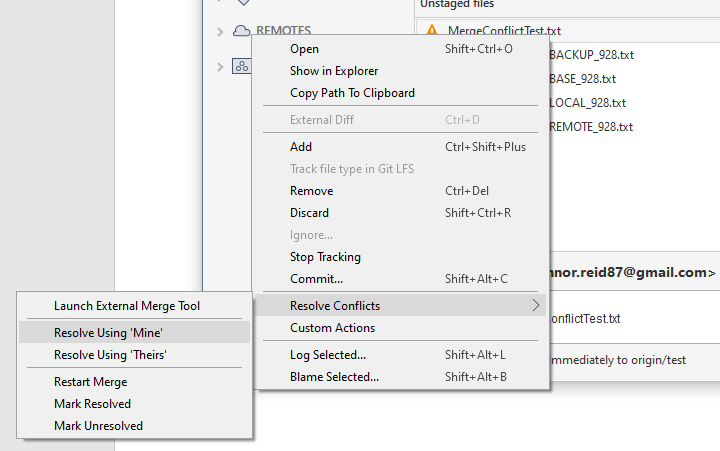
test2 merge test

=======

test3 merge conflict

>>>>>>> test3

This text essentially says that you need to go into that text and choose which one will work for the project. If you exclude the >>>> and <<<<< lines the you essentially want something that will be a safe mix of the text above the ======= or below it. Alternatively, you can simply choose one over the other by right clicking the conflicting file in the File Status window and selecting Resolve Conflicts > resolve using “mine” / “theirs”.



Once you’ve done this hit the “Stage All” button in ST and click commit. It will generate a commit message that is descriptive of the merge changes.

It’ll take a while to get used to this but essentially what we are going to want to do is push to our remote branches often and then periodically merge these branches into the **master** branch. When one of us merges into the master branch then we should notify the other because they will want to pull the changes into their local master branch and then merge the master into our local branch.

I know this is pretty complicated, but we should be able to get used to it relatively quickly.