Using git without the command line

Game Dev Club at SJSU

Author: Cole Pergerson

This is a guide on how to use git without touching the command line. Basically you'll be pressing buttons instead of writing commands. This guide was created to promote the use of git for version and source code control while also providing an easy alternative to the command line.

Please note that if you're planning to go into an industry that uses git, then we highly recommend that you learn how to use git in the command line as it will give you a more complete understanding of the tool.

Below is the list of tools that will be covered. This list isn't exhaustive as there are alternatives to each of these tools.

GitHub Desktop
A friendly UI that provides a visual process for most of the git commands.
Documentation: https://docs.github.com/en/desktop

Visual Code (or Atom)
A lightweight code editor that has visual merge conflict tools. If GitHub Desktop encounters a merge conflict, then you can open the conflicted file with Visual Code and resolve conflicts.
Documentation: https://code.visualstudio.com/docs/editor/versioncontrol

Setup

Install GitHub Desktop and Visual Code.

Make sure that the editor for GitHub Desktop is set to Visual Code.
Click here for help

Using GitHub Desktop

What is Git and GitHub?

Git is a version control tool, meaning that it keeps track of all the versions of your project. At any time, you can restore older parts of your project if you break your current version. GitHub is a site that will allow you to upload your git project to their cloud servers. Your code will be online and others can collaborate with you remotely. When you download GitHub Desktop, it will automatically download git on to your computer.

Git:https://git-scm.com/
GitHub: https://github.com/features

There are some git concepts that you should know. Commits are the different versions of your project. It's up to you how many updates a single commit will have, you'll know more about that later. Whenever you see "Commit your changes," that means creating a commit. Git allows other people to work together on the same project using branches. These are separate safe zones that can be created and worked in without affecting other branches. Each branch contains their own commits and to get commits from another branch, you'll need to merge the branches.

GitHub has some concepts that may be confusing. First, projects are called repositories. Pushing means uploading and Pulling means downloading. The Origin means the repository stored in the cloud. You will be pulling commits from the origin and pushing your commits to the origin.

I'm the host of the project

Whoever has started the project will need to upload the files to GitHub. You can do this through the desktop application via two ways

1. Make the local repository, then save the project file the repo file location.

2. Make the project first, then create the local repository.

Click File → New repository. A window will appear where you can name your repo and choose your local installation location. Look to see if there is a .gitignore file for your editor. There is one for Unity and Unreal. The .gitignore tells git to ignore files that don't need to be shared to other users and makes collaborating easier.

If you followed process 1, then go to your editor and save your project to the newly created repo folder. If you followed process 2, then drag your project file into the repo folder.

One those steps are created, you should see changes recorded in GitHub Desktop (Fig. 2). If you're using Unity, check "Note for Unity" below first. When you're ready, write a summary of the changes, "Uploaded project" is fine. Then press commit changes.

Note for Unity

There are two things to know.

First, make sure the whole team is using the same version of Unity. If you all have different versions, then you'll experience issues in Git syncing.

Second, the default gitignore for Unity usually doesn't work. If your project is stored in a subfolder in the repo folder, then GitHub Desktop will most likely show over a 100 changes (Fig. 1). Since this is a new project, you should have less than 30 changes (Fig. 2). This is because your .gitignore is assuming the project is in the root folder, not in a folder within the root folder. Edit the .gitignore by removing the '/' in front of the file directories listed at the top of the file. See images below

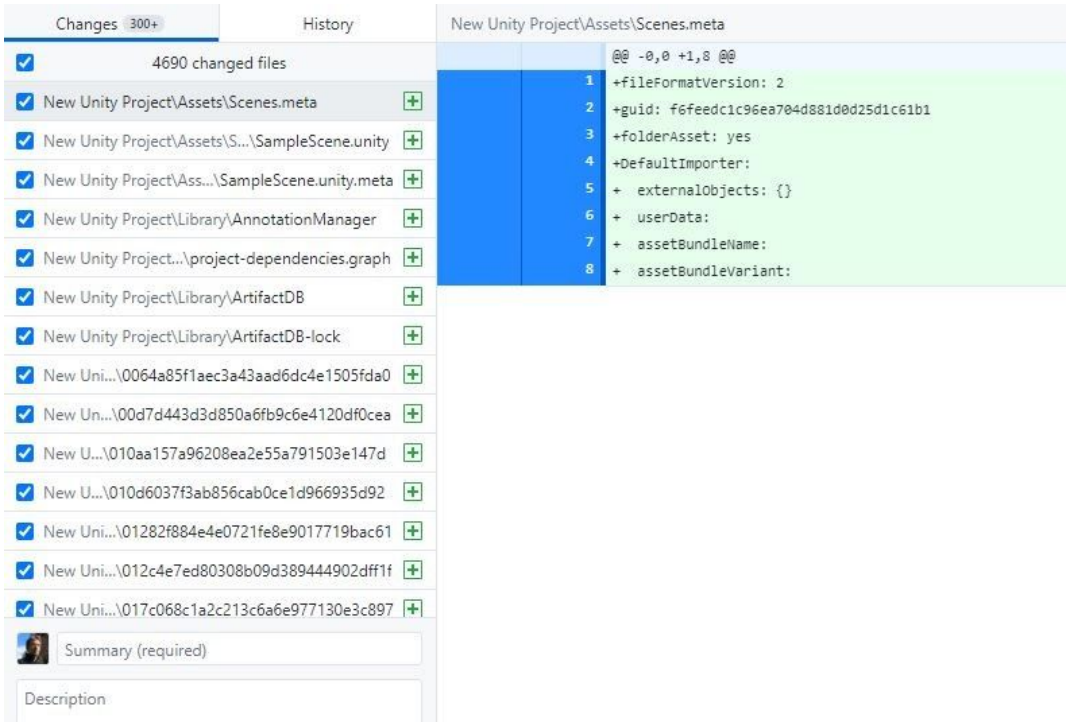


Figure 1

Default .gitignore	.gitignore with the ‘/’s removed. Everything else in the file is fine.
*.gitignore - Notepad File Edit Format View Help # This .gitignore file should be placed at # # Get latest from https://github.com/githu # /[Ll]ibrary/ /[Tt]emp/ /[Oo]bj/ /[Bb]uild/ /[Bb]uilds/ /[Ll]ogs/ /[Mm]emoryCaptures/	*.gitignore - Notepad File Edit Format View Help # This .gitignore file should b # # Get latest from https://githu # [Ll]ibrary/ [Tt]emp/ [Oo]bj/ [Bb]uild/ [Bb]uilds/ [Ll]ogs/ [Mm]emoryCaptures/

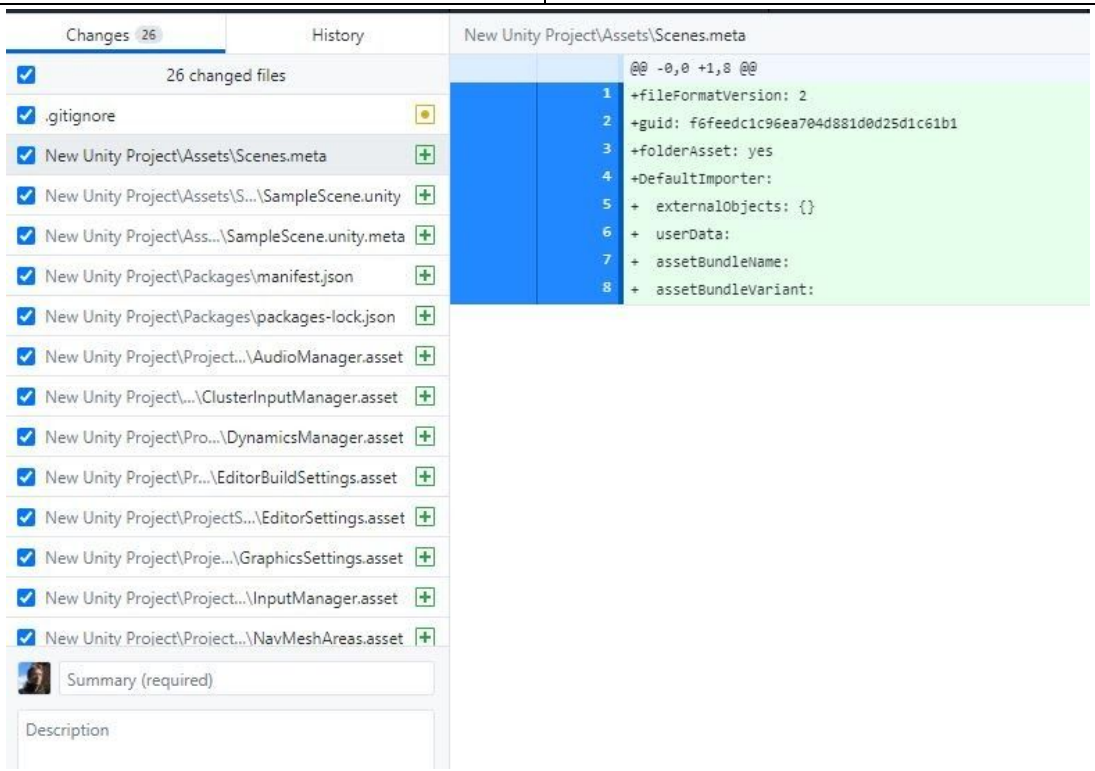
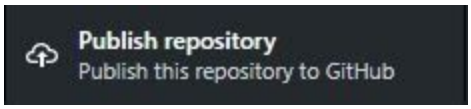


Figure 2

Publishing Repo

Now publish the repository if you haven't done it already. You should see that option as one of the buttons on the top. Once you publish the repo, you might have another option called "Push Origin." Go ahead and press that too, look at the "Uploading Changes" section below for more details on that.



This requires that you have a GitHub account. Click here for help

Inviting your team to collaborate

It's important that everyone who will be collaborating on this project has a GitHub account. The repo that you have created is hosted on GitHub which takes care of the online syncing. The next important thing is that each member is invited to collaborate on the project. Anyone can download the repository you have uploaded, but they don't have permission to make edits until they are official collaborators

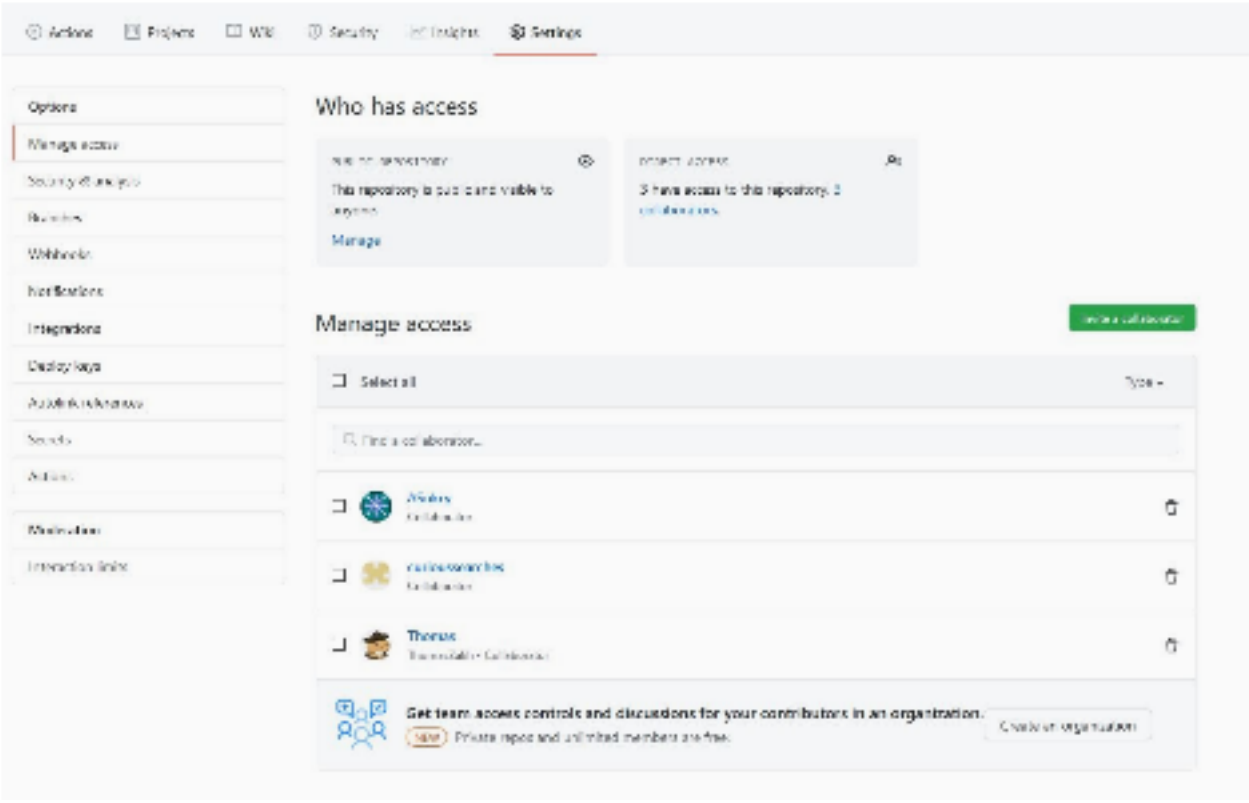


Figure 3

This is the setting page of a GitHub repo (Fig. 3). Under ‘Manage access,’ you can see there are three collaborators for this project, and only these three people and the repo owner can make edits.

A **common problem** is that the repo owner will forget to invite someone and then a team member will try to add their changes. They will be prompted with a window asking if they would like to create a fork which is basically an exact, but separate, clone of the repo. This is super confusing for anyone not familiar with GitHub, but luckily it’s an easy problem to fix.

I’m trying to join the project

If you haven’t already, create a GitHub account. Send your GitHub username to whoever is the host of the project. It’s very important that you’re a collaborator of the GitHub repository. GitHub is a site that hosts projects that are using git for version and source code control. A repository is the name of a project hosted on GitHub. There are millions of public repositories on GitHub that you can view and download, but you can’t make edits to them unless you're a collaborator from them. The repo host must invite you to be a collaborator, otherwise, when you try to make edits, GitHub desktop will say you don’t have permission and will ask you to create

a fork. **Don't create a fork**, a fork is an exact, but separate, verison of the repo**.** Basically, you'll be making changes to the wrong project!

To download a version of the project, you'll need to clone the project. When you see or hear clone, just think of downloading the project.

There are two ways to do this.
Clone from GitHub desktop
Clone from GitHub.com

Clone the project, choose where you want to store it, and then open the project.

Uploading Changes

Git automatically your records changes and GitHub Desktop will list them under the Changes tab. In Figure 4, there are 26 changes that git sees. You can simply commit these changes, which is git's way of saying that these changes are good and ready to be sent to everyone else. To commit, simply add a comment and press the blue button at the button. Click here for a more detailed explanation. Then you have one more step, which is to push those changes.

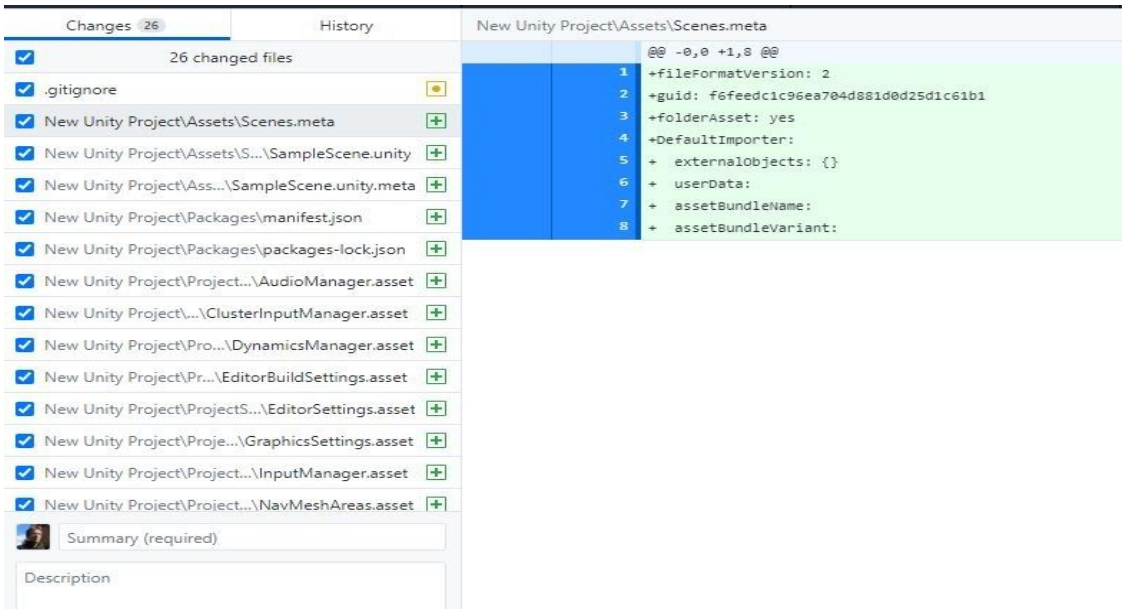
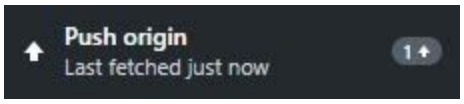


Figure 4

Pushing changes will upload them to the GitHub repo for everyone else to download. Once you committed, you should see a button at the top that will say “Push Origin.” There will be number in that button which will tell you how many commits you’ll be pushing.



Simply push the Push button to share your changes.

Working with others

You now know how to upload changes, but how do you upload changes that don’t conflict with your teammate’s work? This scenario is definitely possible and could lead to loss of work and time trying to resolve git conflicts. That is why you must use branches!

Setting up a branch

Branches are like seperate spaces with the repo that you can freely edit from the main project. In fact, the main project is a branch! It is called master and that is the root of the project. It’s a best practice to have everyone one there own branches and then merge those changes to master. This way each member doesn’t have to worry about messing with other people's changes and if there is a conflict, it happens at one place independent of all the other branches. This way you won’t lose work by bringing changes together.

[Click here for a detailed guide to branches]

Merging changes

Once everyone is working on their own branch, there will be a point where all those changes need to come together to update the game. You can bring changes from one branch into another by merging them.

Merging branches

Select the branch that you want to *bring changes into*. Goto Branch → ‘Merge into current branch…’ A window will appear

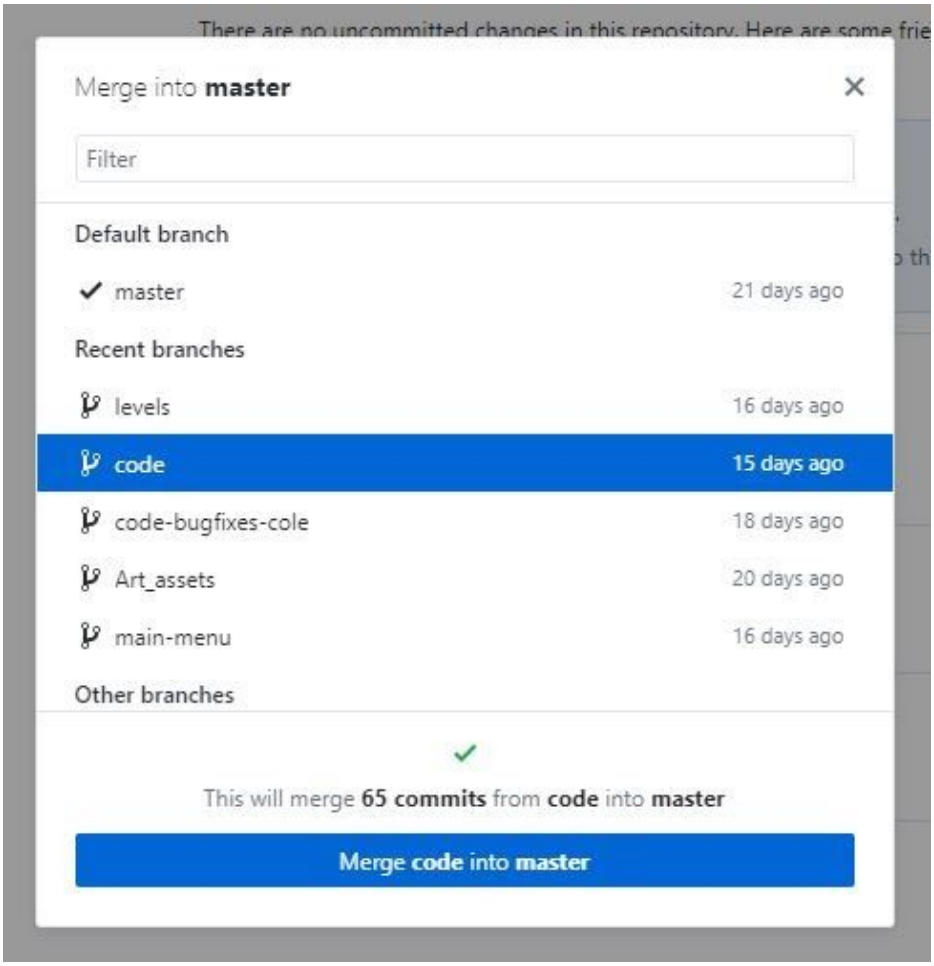


Figure 5

In figure 5, my selected branch is master, the main project, which has a check mark next to it. Here I want to bring the changes from the coding branch into master to update the game. As you can see at the bottom, GitHub Desktop is telling me exactly what this operation is going to do. "This will merge 65 commits from code into master." It is really important that you read and trust what is said before merging. Click merge, it is that easy!

Click here for a, somewhat, detailed guide to merging

Resolving merge conflicts in GitHub Desktop

Sometimes it isn't very easy because there may be merge conflicts. You'll know because GitHub Desktop will tell you beforehand (Fig. 6).

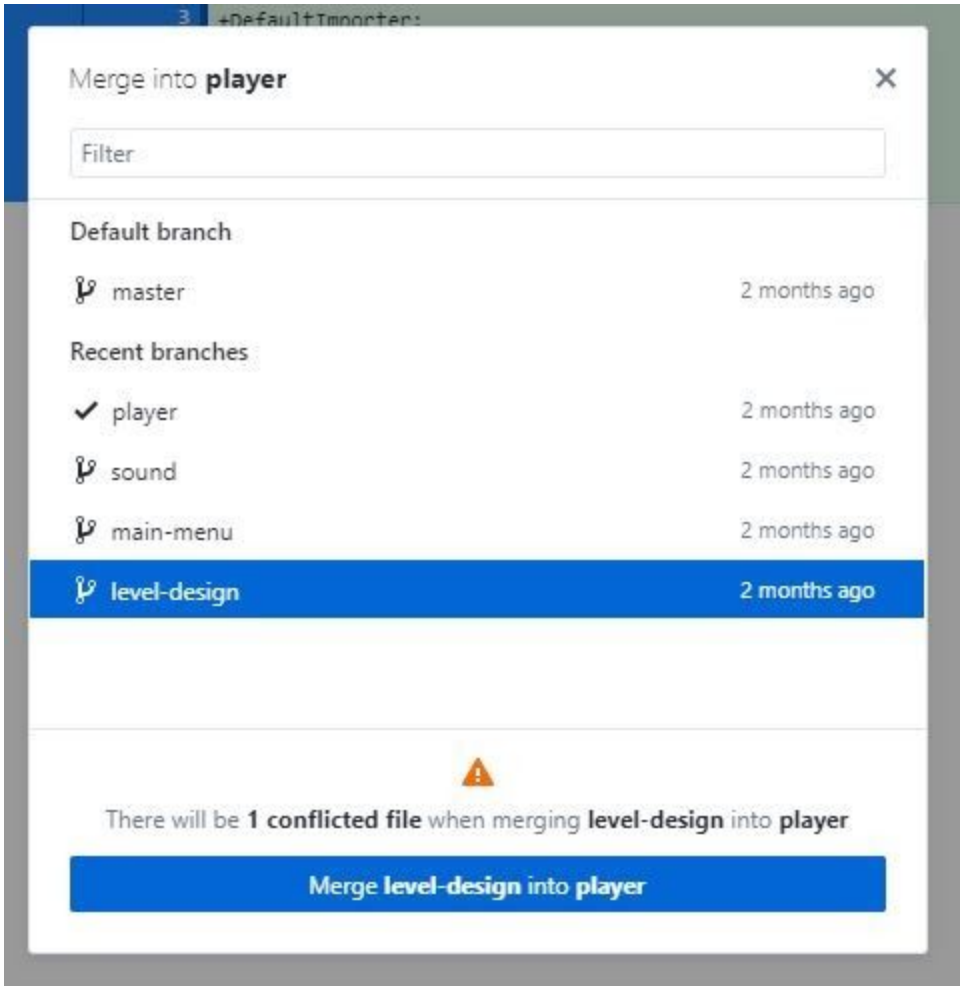


Figure 6

Everything is fine..O_O

Click merge the button, don't worry, you can abort the merge at any time. Another window will appear detailing all the merge conflicts (Fig. 7). As you can see, the commit merge button is greyed out but the abort merge is readily available to be pressed. Merge conflicts are conflicts in the file that git has decided that it is best for us, humans, to resolve. In this case, there is only one issue that needs to be fixed. There is an option to open in Visual Studio, but that will be explained later in this document. Instead, we will solve this using GitHub desktop by clicking the arrow point down. There will be some options, but the two bottom ones are what we will be focusing on. Use the modified changes from X or you use the modified changes from Y. X and Y are the two branches involved in this conflict. Basically, do you want to override the conflict with *your* changes or with *their* changes. It's one or the other, and in most cases, this is the

easiest and best choice to make. Of course you have to sacrifice someone's work, which in most cases, isn't too bad.

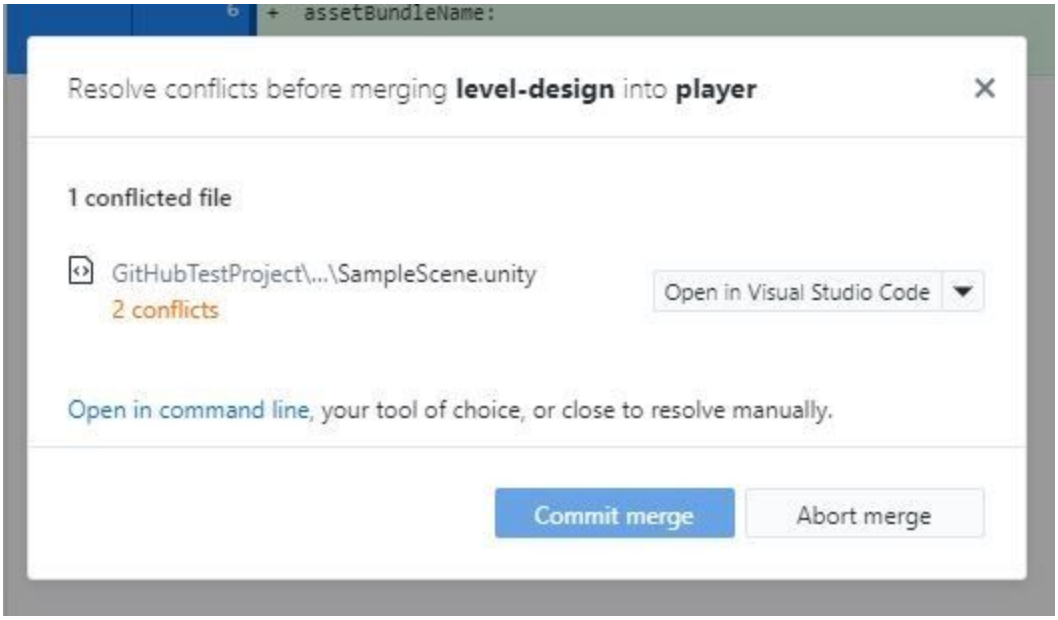


Figure 7

Using Visual Code to solve merge conflicts

Let's come back to this example. As you can see (Fig. 8), one of the options is to "Open in Visual Studio Code." Go ahead and press the button, Visual Studio Code will open up. Usually Visual Code will detect git in the system and open the conflicted file.

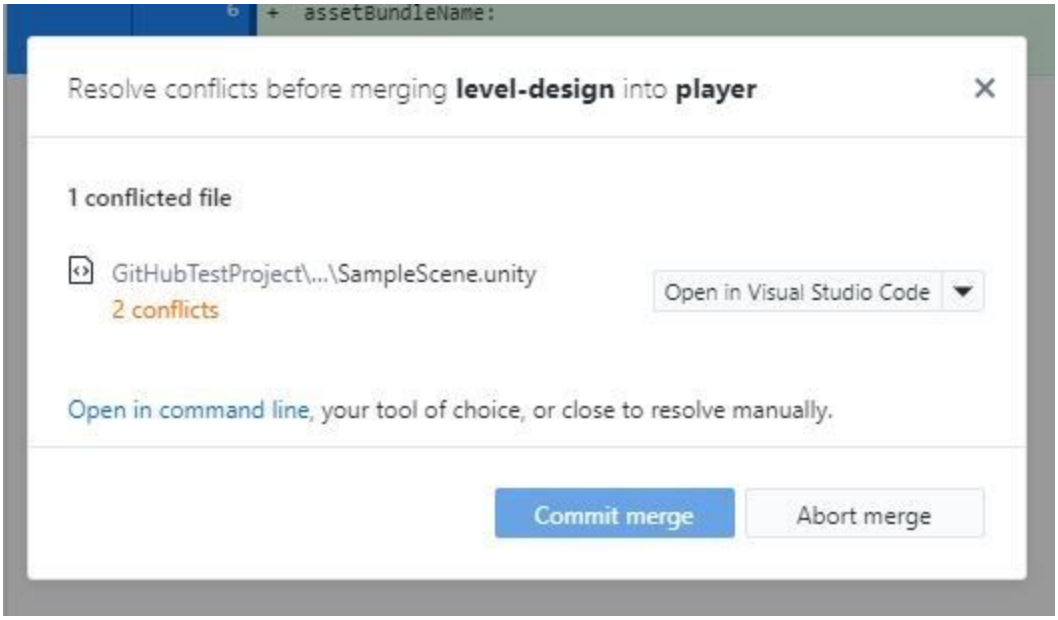


Figure 8

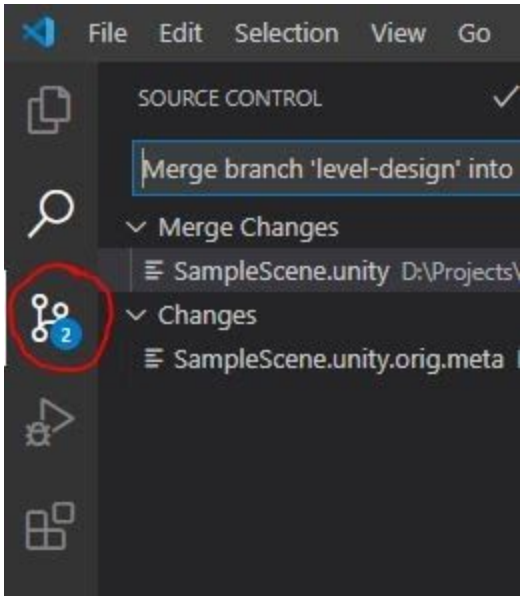


Figure 9

You should see the conflicted file once it opens and on the left, you'll see a branch symbol (Fig. 9). Click that and the source control window will open up.

In Visual Studio Code, you'll have all the same options in GitHub Desktop plus some more helpful tools which I will show you.

You can right click any of the files and resolve the conflict by selecting the incoming changes or the current ones (Fig. 10)

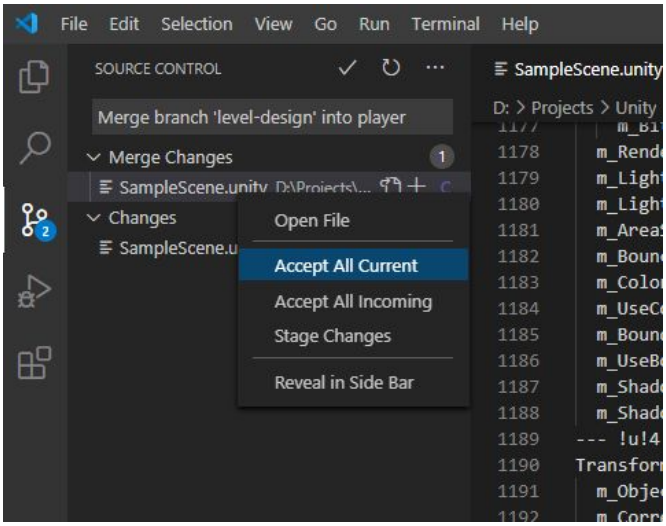


Figure 10

Another way to resolve conflicts is by going directly in the code. In this example, I have a conflict at the variable speed (Fig. 11).

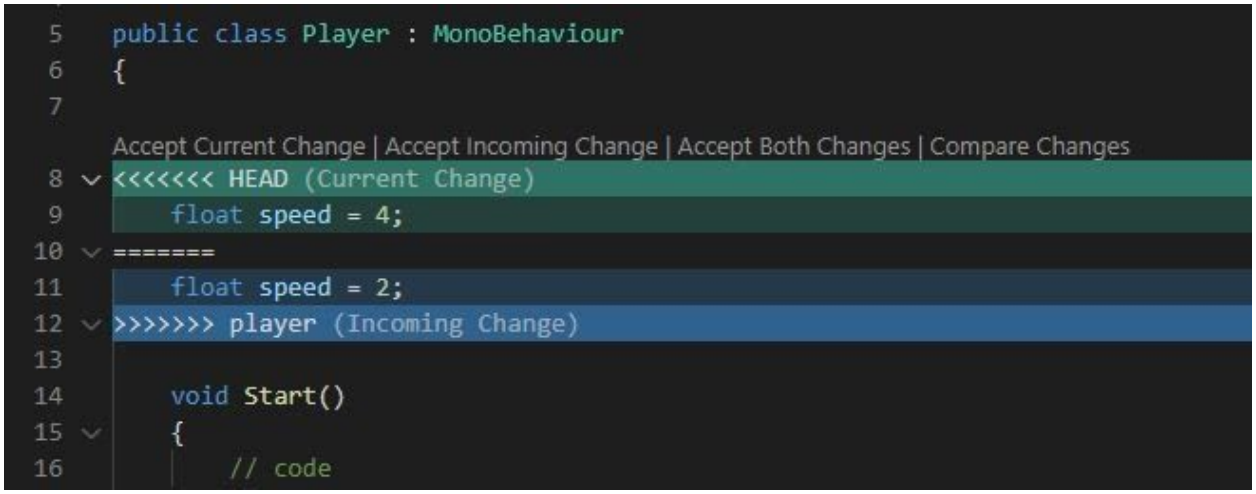


Figure 11

The green code is the current changes, the changes on the current branch. The blue code is the incoming change, the changes from the other branch. I can either click the buttons on the top, or I can directly edit the code. When there is a merge conflict, Git adds the code from both branches into the conflicted files and separates them with the “<<<” and “===” symbols. Visual Studio is handy because it will highlight them to make them easier to find. In some cases, you might want both branch changes so directly editing the code would be the best option.

Our best practices for git

Learn git command line :)

I highly recommend learning git command line because that will give you a better understanding of everything I went over. If not, you always will be limited to the option of GitHub Desktop and you'll be hopeless in any deveasting merge conflicts (this is rare, but happens). Assuming you're a student at San Jose State University, you have free access to Lynda.com with your library card. There are some great courses on that site that cover git command line and you'll be more confident in using Git after watching them.

Branches

Work in separate branches! You can either name your branches after your teammate names or discipline. It's best, however, to name them after features or parts of your game. Usually I use a mixture of these. I might have the player branch, which is for player related code, and then I'll have the art branch, where my artist can upload all of their assets to. It really depends on what is best for your team.

Always keep the master branch working! When it comes to merging everyone's changes together, don't merge them into master yet. Have a separate branch for that where you can merge all the changes at once, resolve any conflicts, and then merge that branch into master. By keeping the master branch clean. You'll always have, ideally, a playable game that you could send off for playtesting.