# Lab Session 8: Using Git with Visual Studio

## **Create New Git Repository**

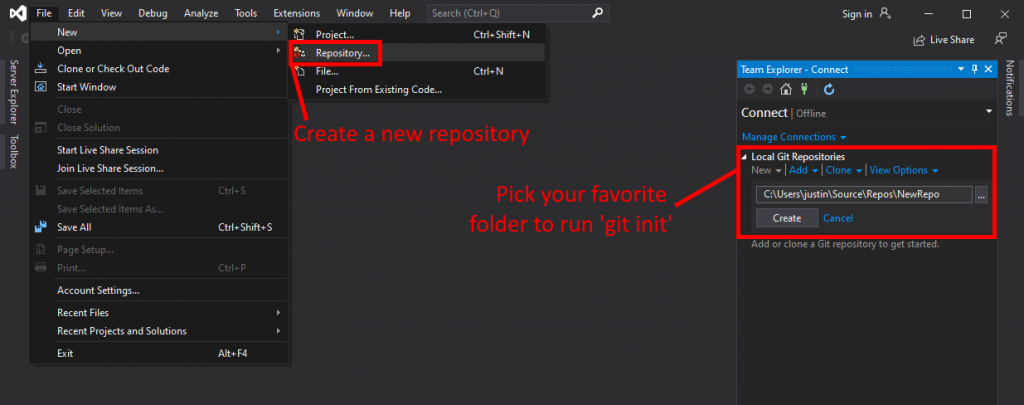
To create a new repository:

Click File –> New –> Repository…

The Team Explorer opens on the right

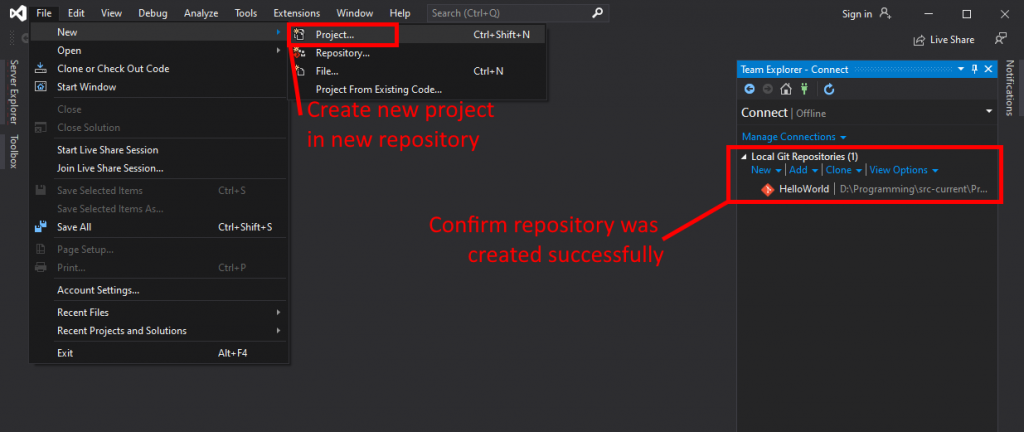
Pick the folder that you want to create the repository in

This is equivalent to running git init from the command line.

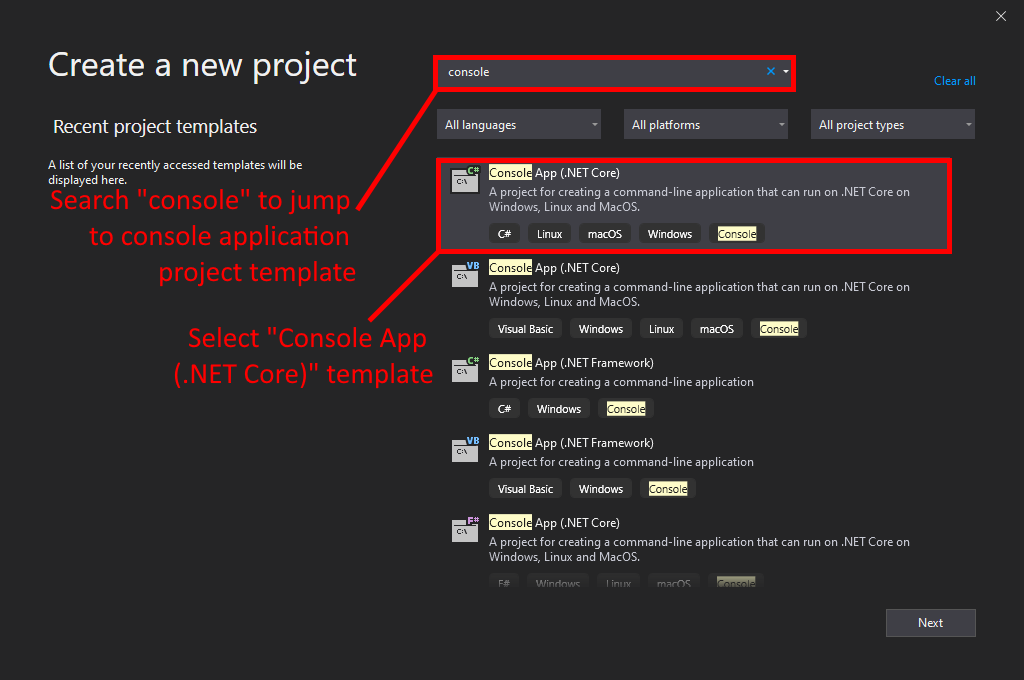


## **Create Project in New Git Repository**

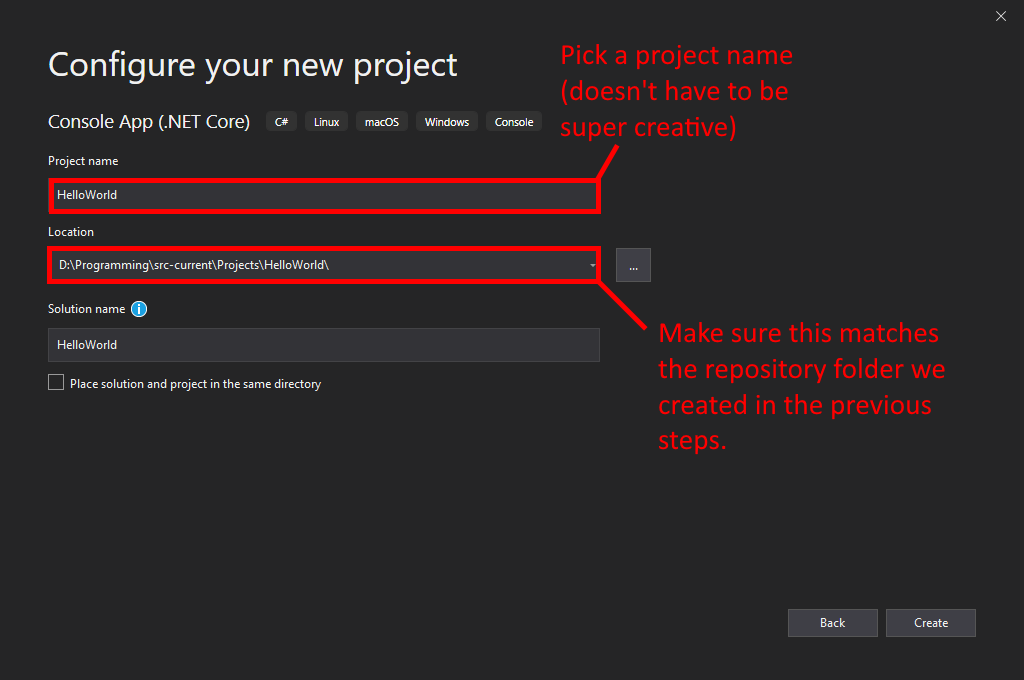
The repository is ready to go, but it’s empty. Confirm that the repository now shows in the “Local Git Repositories” section of the Team Explorer. Initiate the Project Creation Wizard by clicking File –> New –> Project.



Search for “console” to filter down to just console applications. Select “C# Console App (.NET Core)” to create a new console application based on whatever version of .NET Core SDK was installed with Visual Studio. Click Next.



Enter your project name, select the folder location in which you initialized the Git repository from previous steps. If you get this wrong, you won’t be able to follow along with the rest of the guide. Click create, and you’re good to go. Your solution, project file, and template classes will be created and placed in your repository folders.

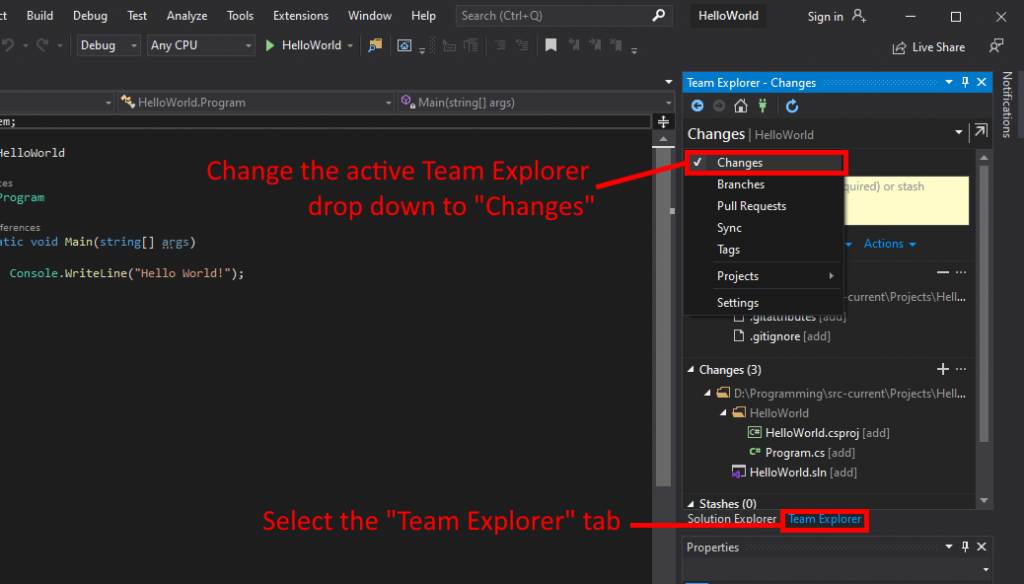


## **Commit New Project to Git**

Visual Studio has a “Team Explorer” docked sidebar (you can drag it to move it around) which enables you to manage your local Git repository and attached remotes.

Change the Team Explorer section drop down to “Changes” so that you can view changes on your active branch (which is the master branch at the moment).

Even though your project has been added to your local repository folder, you haven’t yet committed those changes. Files that Git recognizes as new are not tracked by default and require you to explicitly add the files to be tracked. If you don’t see any changes listed here, go back to the project creation step and make sure that you created the project in the same folder as the Git repository.

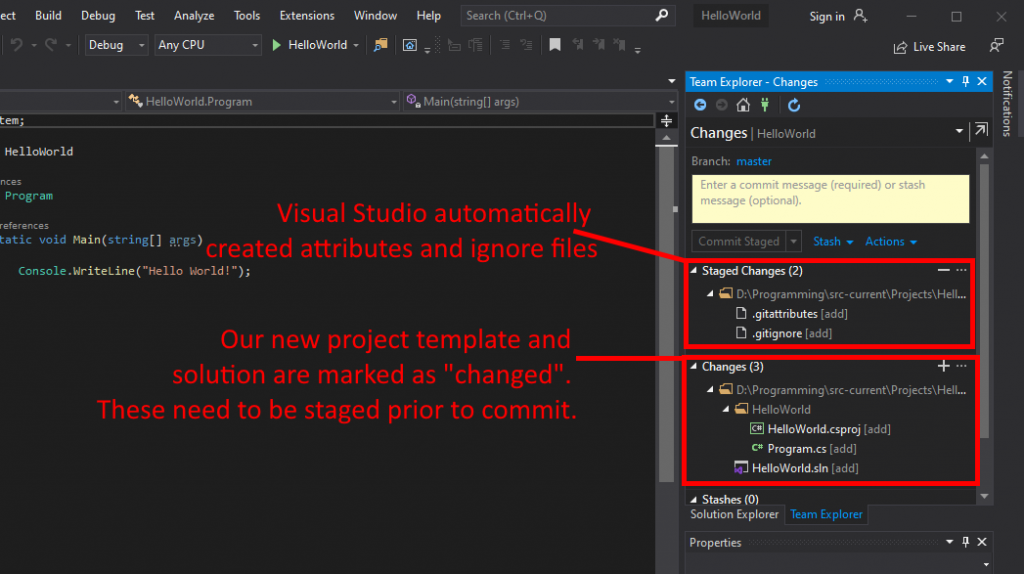


The Changes window in the Team Explorer tab breaks down changes in two ways: “Changes” and “Staged Changes”.

Changes is a list of pending file changes that you haven’t acted on. That is, you haven’t run git add on those files to stage them for a commit.

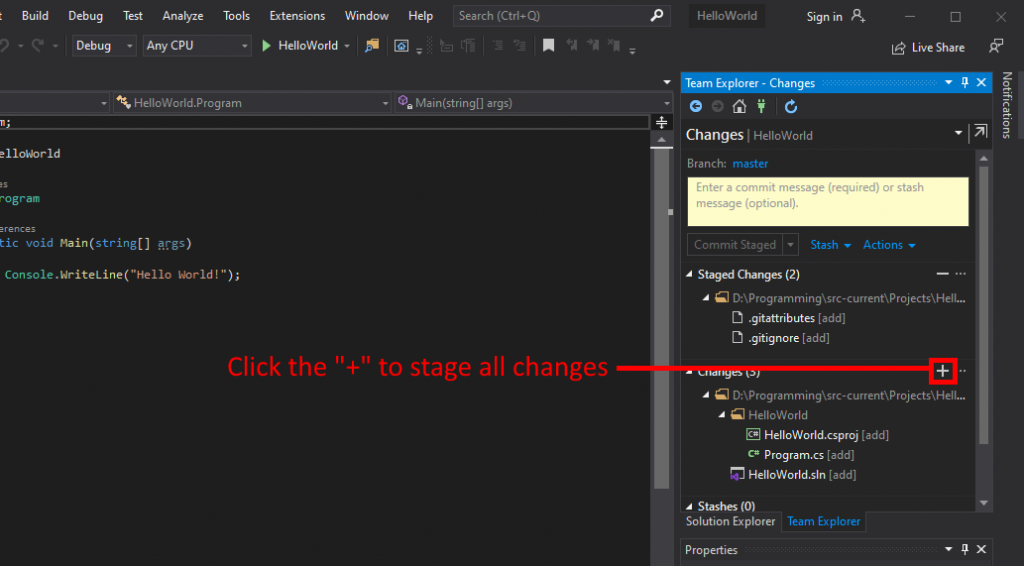
Staged Changes is a list of files that have been added to Git’s staged tracking list to be committed. In this step, Visual Studio’s repository creation wizard has automatically staged .gitattributes and. gitignore files based on built in templates.

Specifically, the. gitignore file was sourced from the GitHub gitignore repository.



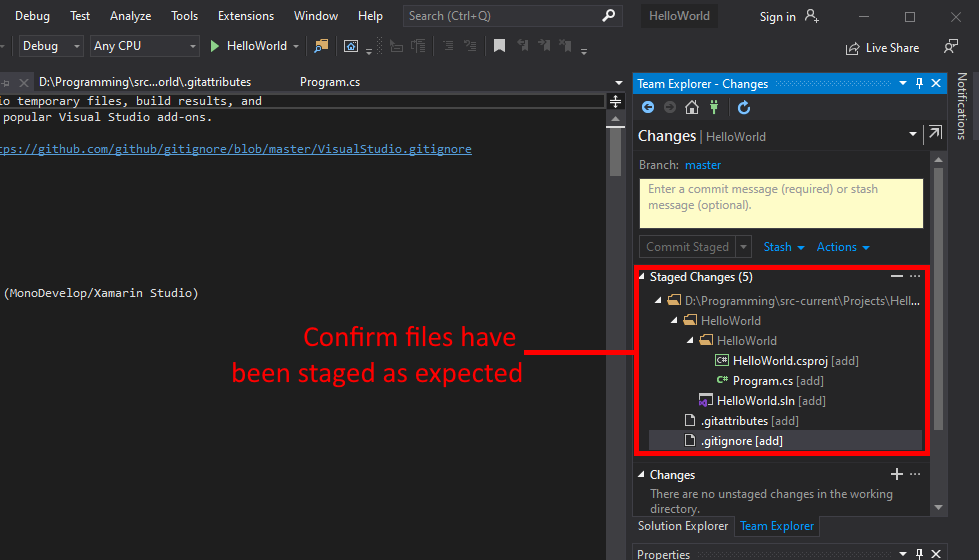
Click the + to stage all changes. In this case, staging everything is fine, but in the real world, you might want to pick and choose exactly which changes to stage. It’s common for developers to make local changes specifically for debugging or testing purposes without the intention of committing those to the repository.

Do yourself and your team a favor by always double checking your changes prior to staging and definitely prior to commitment.



The Team Explorer UI will update with all changes in the Staged Changes section. Confirm that these are as you expect prior to commitment.

If anything looks wrong at this step, you can Unstage by right clicking the file and selecting “Unstage”.

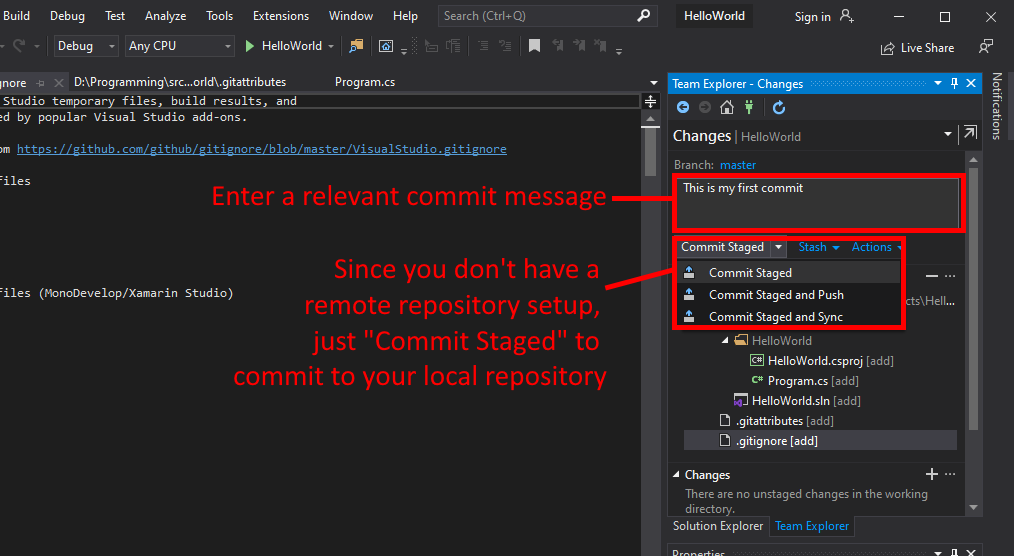


Good commits have good commit messages. Good commit messages aren’t too short and nor too long. That’s vague, but the point is that you should be descriptive enough for those who will come after you but without requiring the reader to struggle to the end.

You have three options here:

* Commit Staged to commit staged changes to the local repository
* Commit Staged and Push to commit staged changes to the local repository and push to a remote repository (like GitHub or Bitbucket)
* Commit Staged and Sync to commit staged changes to the local repository, pull changes from a remote repository, and then push your changes to that remote repository

We only care about the first one for this guide because we have no remote repositories setup.

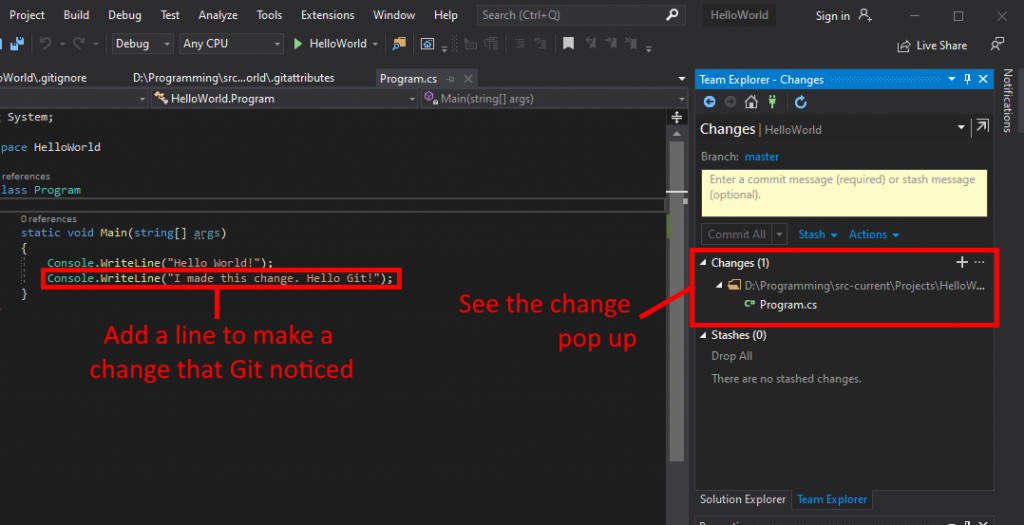


At this point the project is committed to Git and enshrined in the history forever. Let’s work on incremental changes now.

## **Commit a Change to Git**

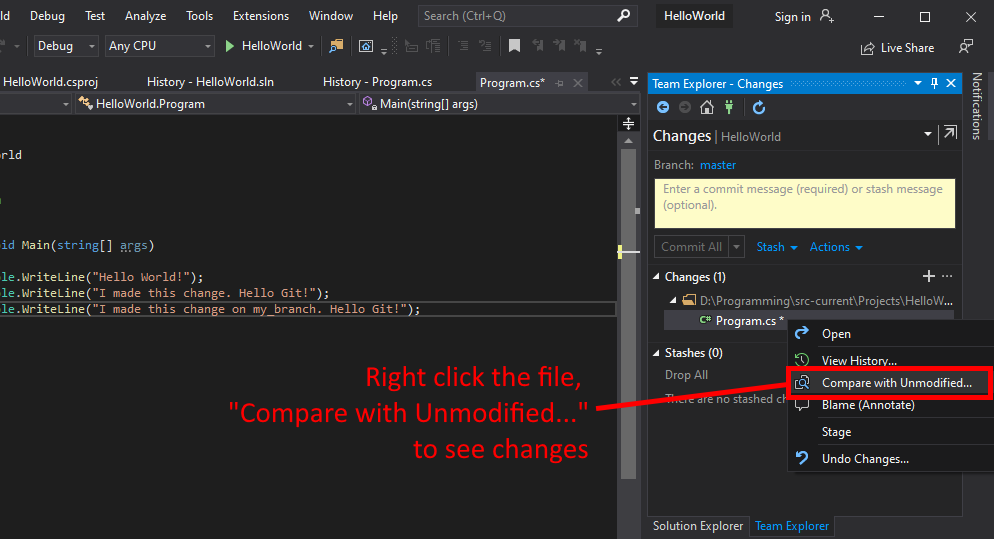
Any time you add or change a file to the tracked repository, the Visual Studio Team Explorer will automatically detect and display the changes in the Changes tab.

Add a line to the Program.cs file and watch as the change displays automatically.



Before you commit stage and commit changes, always check to make sure the changes are what you expected. Long development sessions can leave behind unwanted comments and code that you intended to remove prior to commitment.

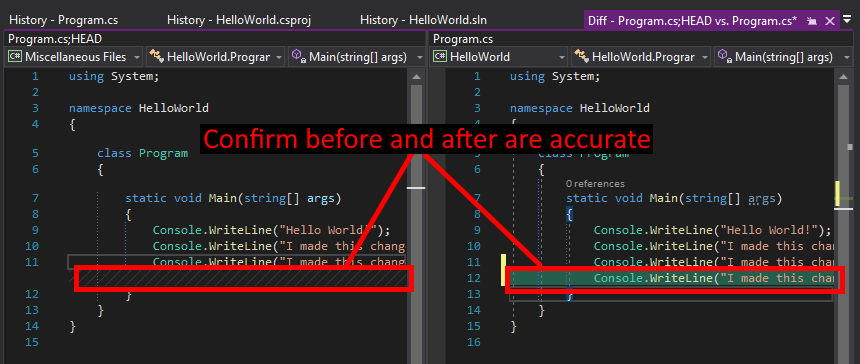
Right click the file or folder that changed and click “Compare with Unmodified…” to see what changes are pending to be staged or committed.



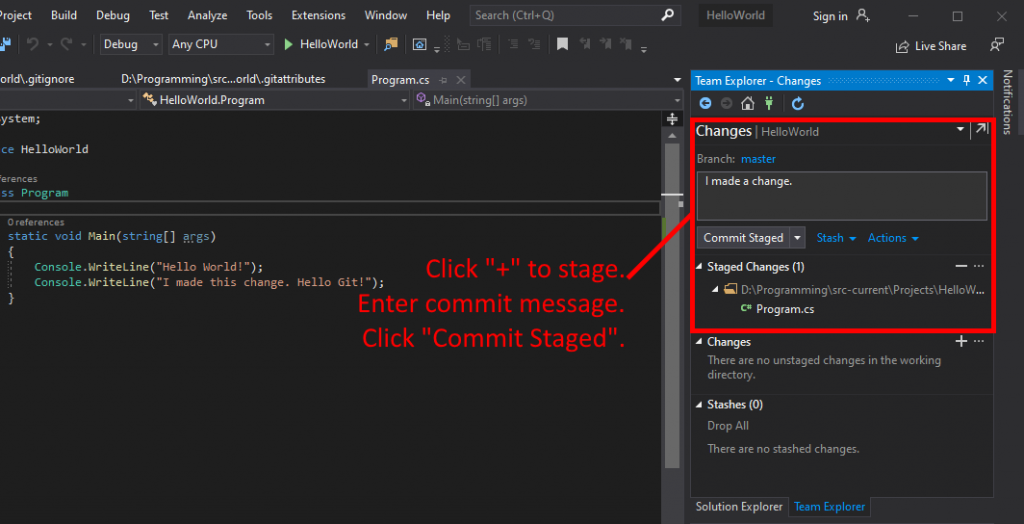
The previously chosen option will launch the Visual Studio Diff Viewer, which displays changes between the HEAD (left, unmodified state of your branch) and the current state (right, pending changes that haven’t yet been committed).

Red lines on the left indicate the previous state. Green lines on the right indicate the current state.

Always confirm that these changes are exactly as you expect.



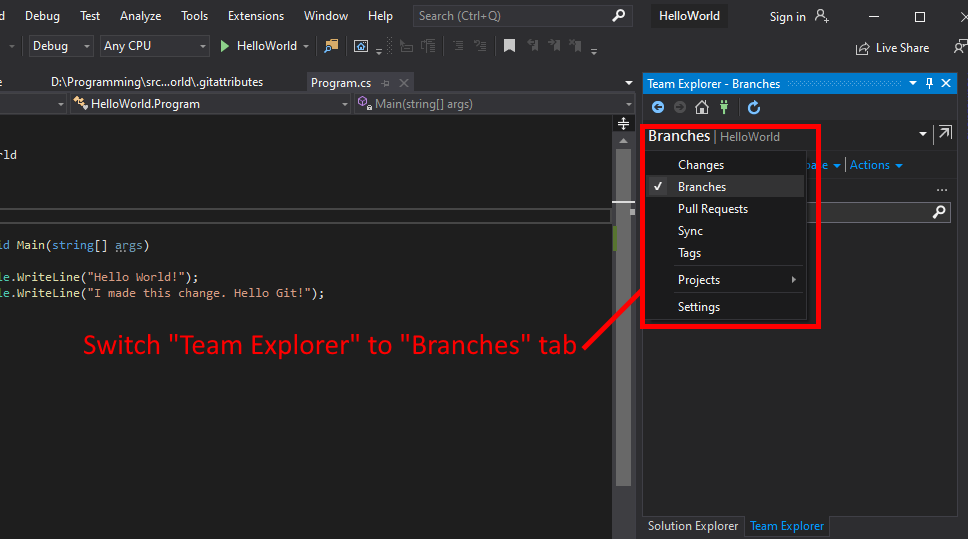
If everything is as expected, just like before, click the + to stage the changes. Enter the commit message (a good one). Click “Commit Staged” to commit to the local repository.



## **Create and Commit to a Local Branch**

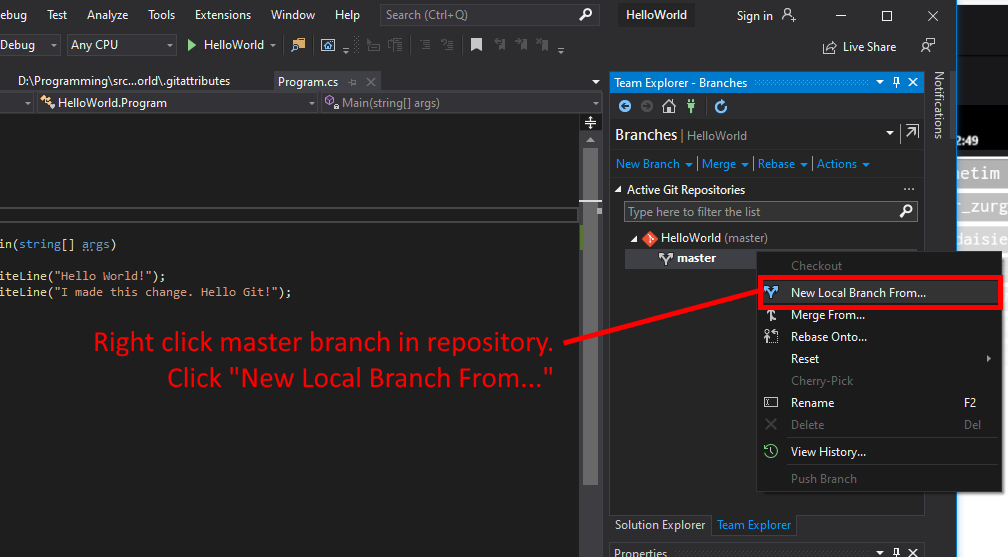
Without the ability to create independent branches of code, team members would be conflicting with each other every time a developer made a commit. Since we also advocate for committing early and committing often, such conflicts would be an immediate deal breaker.

Change the Team Explorer section drop down to “Branches” so that you can view all branches in the local repository.



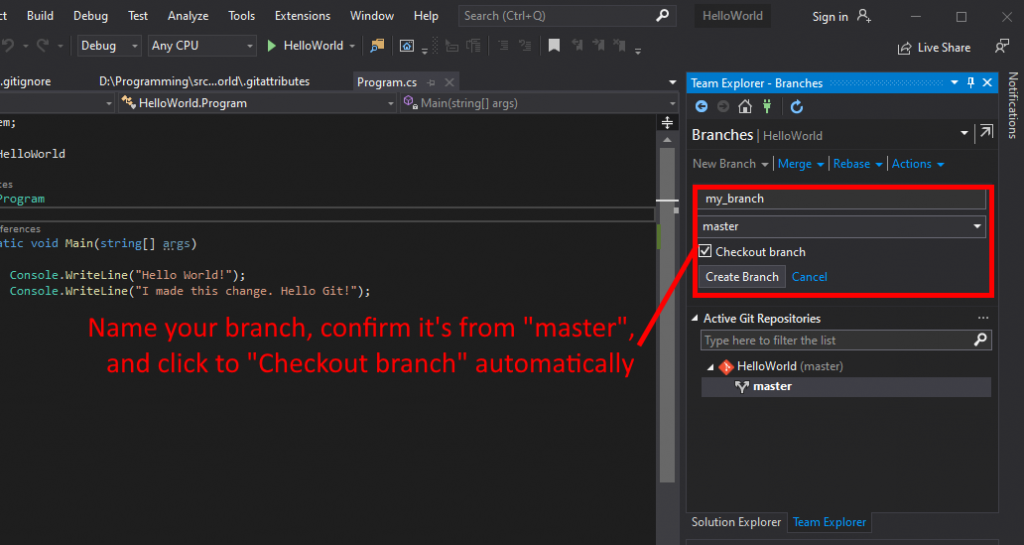
The only branch in a new repository is the default branch known as master. Until now, all changes have been committed against that branch.

Right click the master branch, click “New Local Branch From…” to begin creating a branch based on the current state of master.



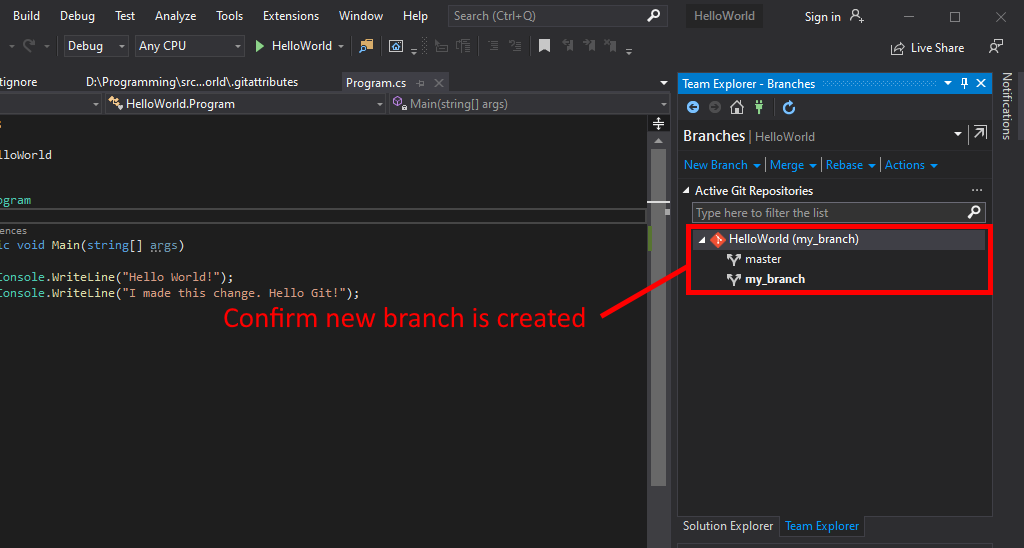
The new branch UI will display in the Team Explorer. Name the new branch, confirm that you are branching from master, and select to checkout the branch so that you don’t have to do that manually after creation. It’s just an option for convenience.

Ultimately the naming convention is up to you and your team to decide. Whatever works best for your flow is fine. Just make sure the names make sense.



Confirm the branch was created and that it is now the active branch (indicated by bold lettering) in the Branches tab of Team Explorer.

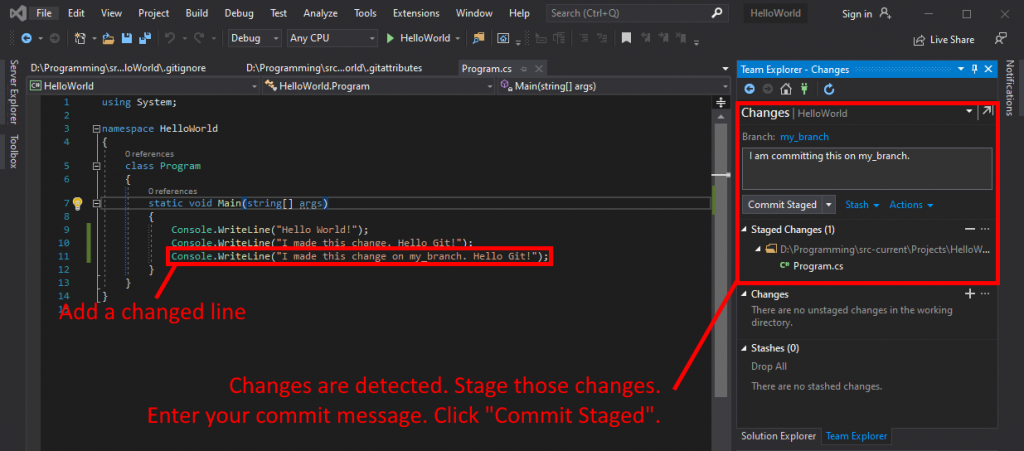
If you didn’t click to checkout the branch upon creation, you’ll have to double-click the branch now to switch to it.



After confirming that the new branch is the active branch, it’s time to add some changes and commit to that branch.

Just like in the previous sections, add a new line to Program.cs. You should see changes automatically detected and displayed in the Changes tab.

Again, just like before, use your new skills to confirm your changes, stage those changes, enter a good commit message, and then commit the changes to the new branch.



There is no difference in the process of committing to a separate branch. In fact, master is considered a branch as well, so the process is exactly the same.

But how do we get our changes back to the master branch so that other coworkers can use our changes? Surely, we don’t want our changes isolated in a separate branch forever.