Introduction

As a student, most of the time we are given a small project to work on either individually or in small groups. Version control is not an issue as with such a small group, we can transfer files to each other easily. However when the size of the project grows larger and more people are working on it ,version control becomes an issue as there would be a few people working on the same file and it is a big hassle to keep all the files up to date. This is where a version control system such as GitHub comes in.

GitHub is a web-based Git repository hosting service. It offers all of the distributed revision control and source code management(SCM) functionality of Git as well as adding its own features. When developers are creating something (an application, for example), they are making constant changes to the code and releasing new versions, up to and after the first official (non-beta) release.

Version control systems keep these revisions straight, and store the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.

Checking in + conflicts

"Checking in" in GitHub is a term used to pull files from the repository to your computer. This means that the most up to date file on GitHub is pulled onto your computer and merges itself with your current file. When the file merges, there are obviously conflicts to the file and a conflict message will appear onto the file you merged so you can edit your own file to match with the latest. So it is advised that the first thing you would do after opening GitHub is to pull the latest file.

Checking out + conflicts

"Checking out" in GitHub is a term used to deposit files to the repository from your computer. However before you deposit, it is ideal that you leave a commit message. This commit message is used to notify other contributors of your changes so that they can make changes to theirs too. After committing, all you have to do is "push" and the file will be on the repository. The same conflict message will appear when others pull your file if theirs are not up to date.

Branching

"Branching" in GitHub is basically making a backup. Instead of committing to changes onto the only file (which is also called the master file), you create branches from your master and every time you commit, you commit to only the branches and not the master file. This will allow easy changes as the master file will remain unchanged and only the changes needed will be added onto the master file by the master of the repository.

Revision numbers

"Revision numbers" in GitHub is a way for everyone to know which changes are truly the latest. The revision number in GitHub is based off unique hashes as if it was based off an incremental value, we would think that v1.21 is the latest when v1.20 was actually committed just before v1.21 was posted by another contributor.

Pull Request

"Pull request" is that when reverting a pull request on GitHub , it creates a *new* pull request that contains one revert of the merge commit from the original merged pull request.

Screenshots

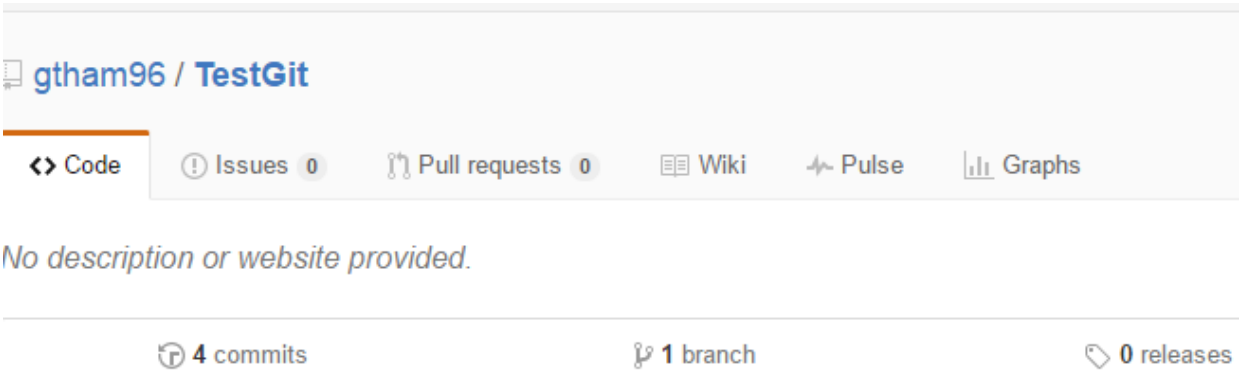

Figure 1: repository name

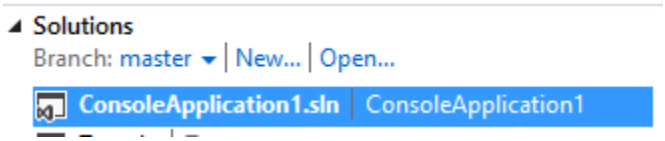

Figure 2: Solution uploaded

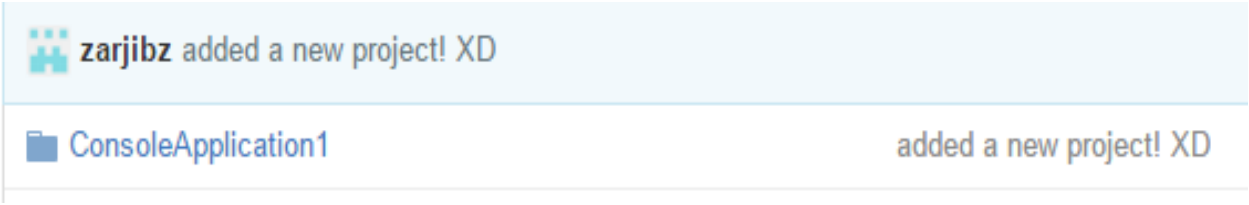

Figure 3: project added with commit message

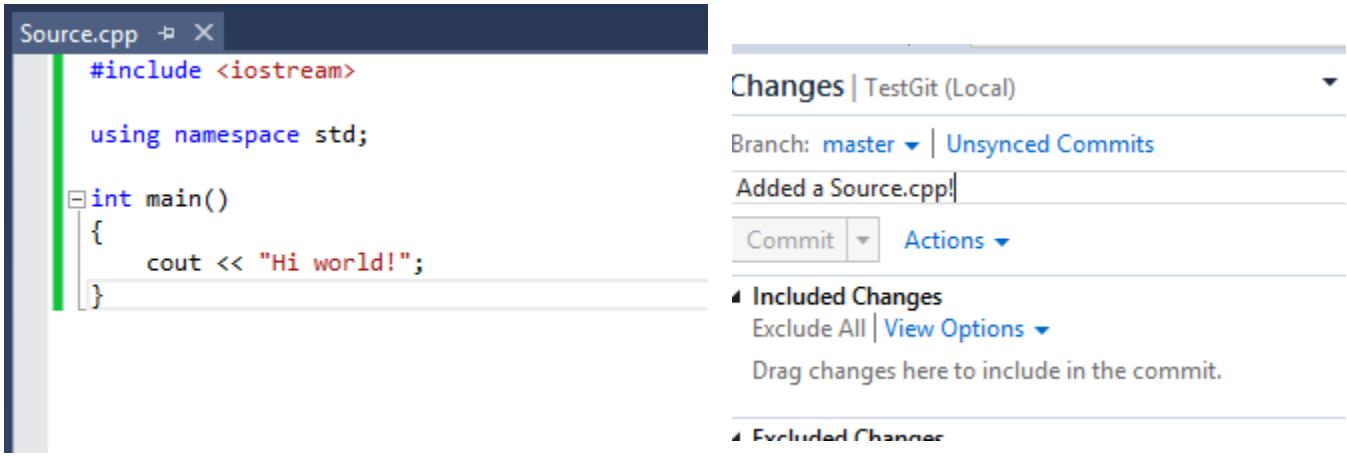

Figure 4:Code was added + committed and sync'd

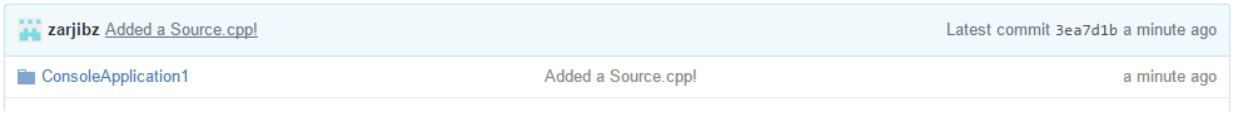

Figure 5 Committed onto GitHub

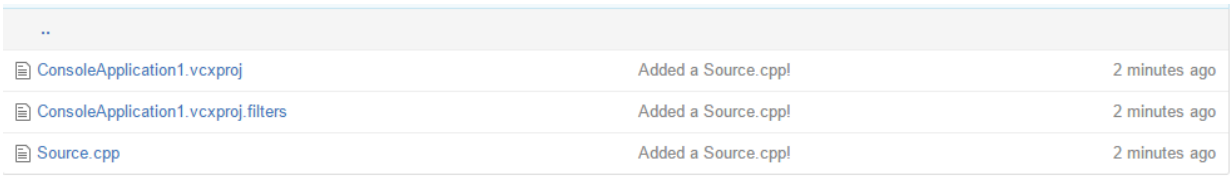

Figure 6:Source.cpp on GitHub

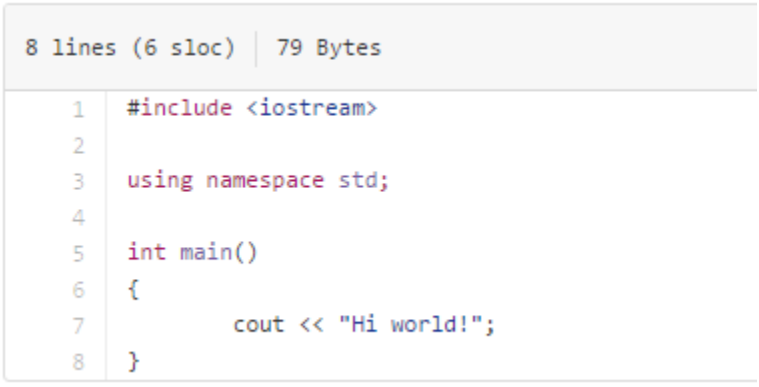

Figure 7: Source.cpp raw

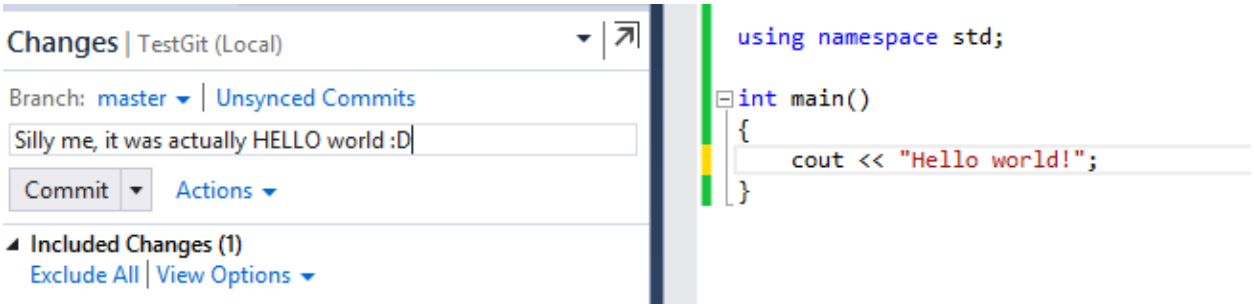

Figure 8: Changes were made

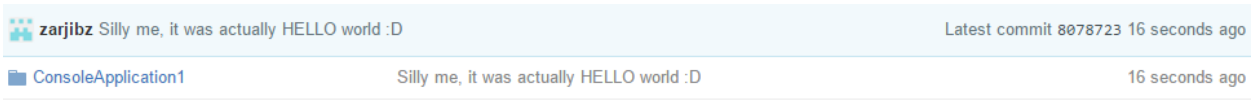

Figure 9: On GitHub

8 lines (6 sloc) | 82 Bytes

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7          cout << "Hello world!";
8  }
```

Figure 10:new source.cpp raw

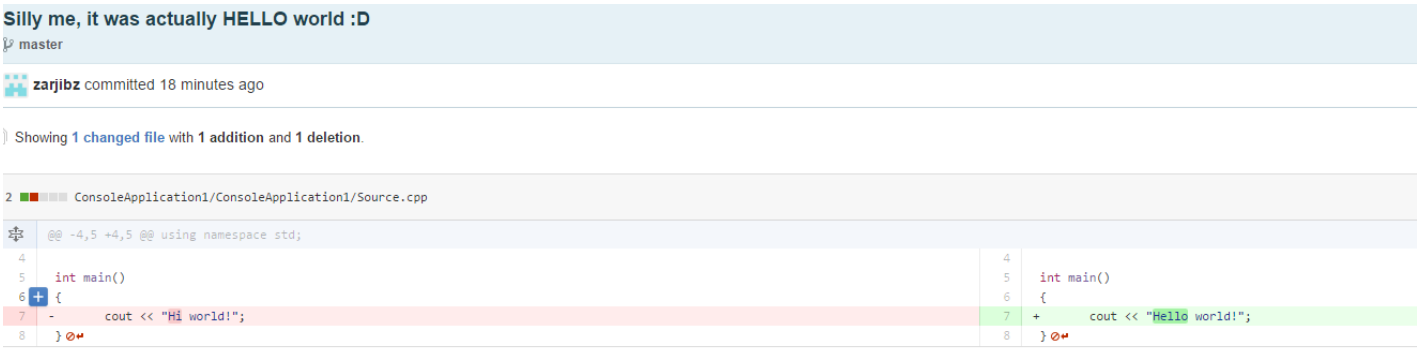

Figure 11: Using Git blame to trace changes in a file