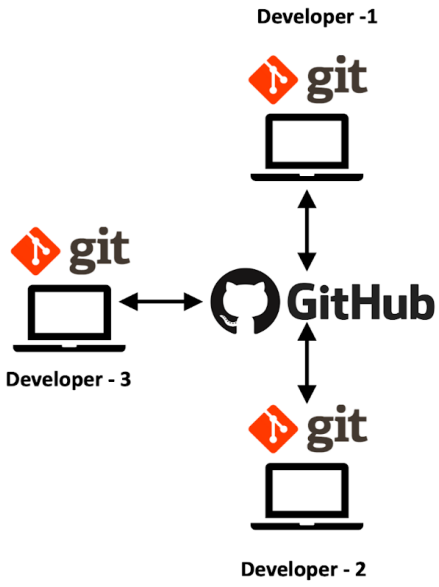
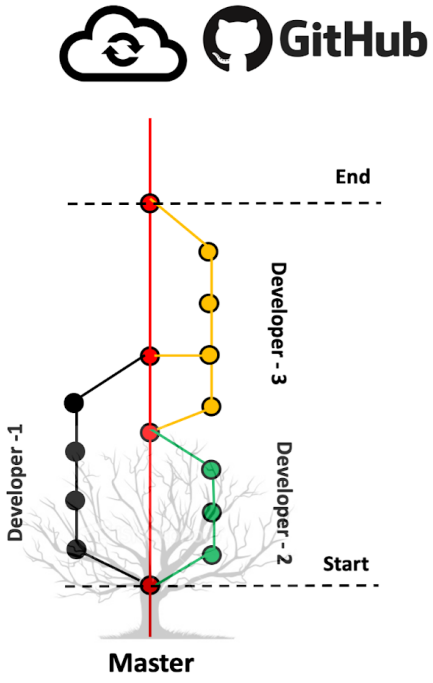
```
/**
 *
 *                                                    ,--,
 *    ,----..                    ___                ,--.'|
 *   /   /   \    ,--,    ,--,'|_     ,--,  |  :                ,---,
 *  |   :     : ,--.'|    |  | :,' ,---.'|  :  '            ,--,,---.'|
 *  .   |  ;. / |  |,     :  : ' : |   | : _' |         ,'_ /||   | :
 *  .   ; /--`  `--'_   .;__,'  /  :   : |.'  |  .--. |  | :: :   : :
 *  ;   | ;  __ ,' ,'| |  |    |  |   ' '  ; :,'_ /| :  . |: :     |,-.
 *  |   : |.' .''  | | :__,'| :  '   |  .'. ||  ' | | . .|   : '   |
 *  .   | '_.' :|  | :     '  : |__ |  | :  ||  | ' | | ||   |  / :
 *  '   ; : \  |'  : |__ |  | '.'|'  : |  ; ;:  | : ;  ; |'   : |: |
 *  '   | '/  .'|  | '.'|;  :    ;|  | '  ,/ '  :  `--'   \   | '/ :
 *  |   :    /  ;  :    ;|  ,   / ;  : ;--'   :  ,      .-./   :    |
 *   \   \ .'   |  ,   /  ---`-'  |  ,/        `--`----'   /    \  /
 *    `---`      ---`-'           '---'                     `-'----'
 *
 * @Author Luke Kvamme, Reed Shoolbred, Michael Peeler, Bryson Addison
 *
 */
```



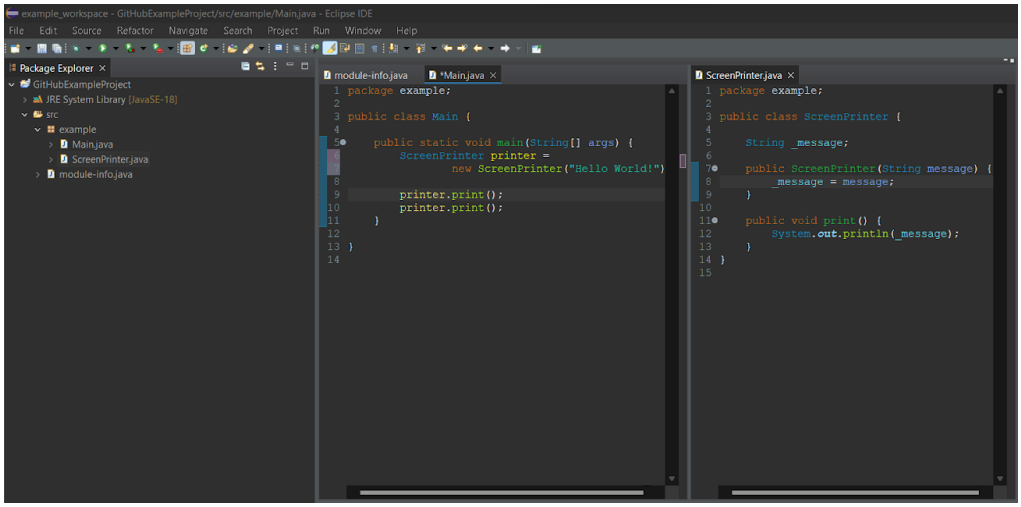



```
module versionControl {
      requires GitHub Website : github.com;
      requires GitHub Desktop : desktop.github.com;
      requires Eclipse : eclipse.org;
}
```

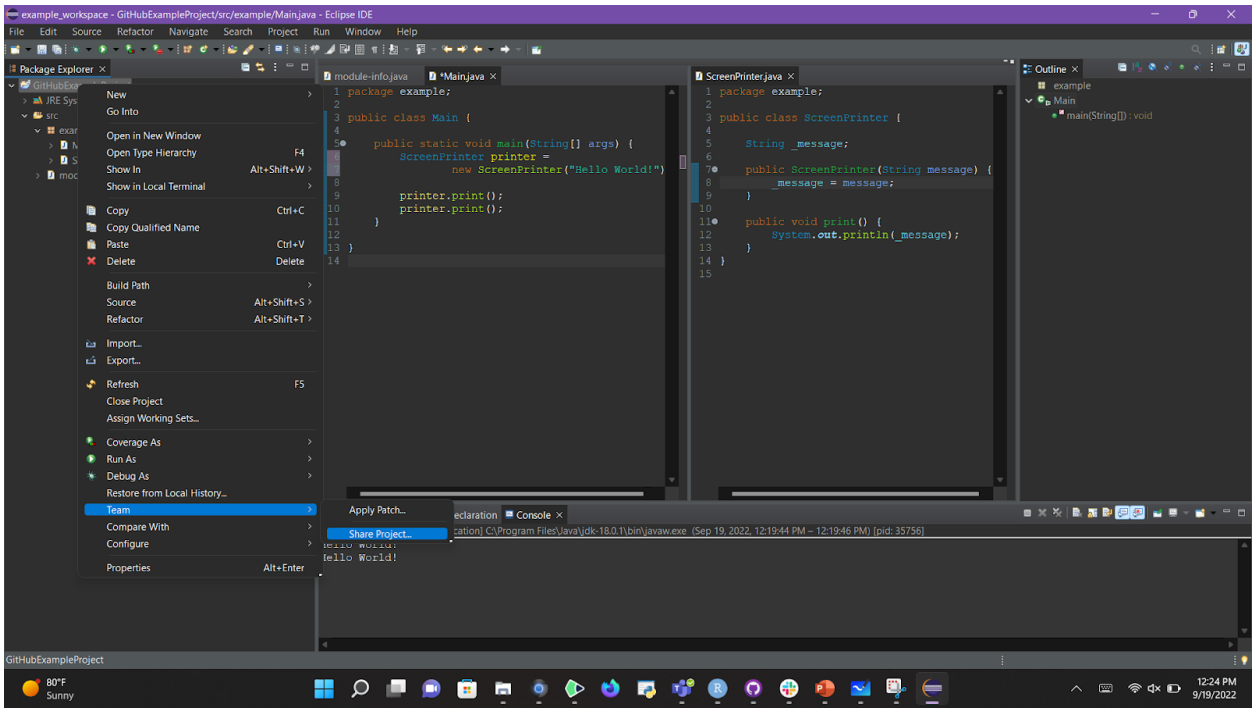
```
enumerate Terms {
      Repository {all files of the project and each file's revision
                history},
      MasterBranch {the main files},
      Branch {alternative, edited versions of the repository not yet
            merged with the master branch.},
      Fork {making a copy of a repository on GitHub, turning a branch
            into its own repository. Forks can merge back into the
            repository 'upstream' of them after edits are made, or
            can be spun into different projects. Typically used for
            projects you do not own},
      Clone {taking a branch and saving it locally},
      Fetch {gets the latest version of a branch that you already have
            locally},
      Staging {getting a file ready for a commit},
      Commit {save a change to the branch in the local repository
            class fix <fix: description> {patches a bug}
            class feat <feat: description> {introduces a new feature}
            class ! or BREAKING CHANGE <fix! description> {introduces
                a breaking change to the api}
            class Other {refactor, style, test} },
      Push {updates the non-local branch with the local version},
      PullRequest {also called "merge request," requests to merge the
            commits on the branch into the project's main repository.
            class Comment {Submit general feedback without explicitly
            approving the changes or requesting additional changes}
            class Approve {Submit feedback and approve merging the
                changes proposed in the pull request}
            class RequestChanges {Submit feedback that must be
                addressed before the pull request can be merged}},
      Merge {combines two branches into one},
      Rebase {avoids merge commits if you pull from a remote
            repository and have divergent changes and instead rebases
            local branch on the remote branch it tracks}
}
```

<< Sharing Projects >>

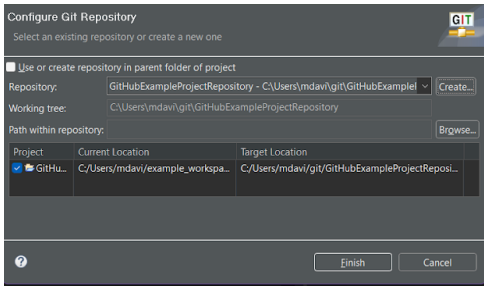
Begin with your project inside **Eclipse.**

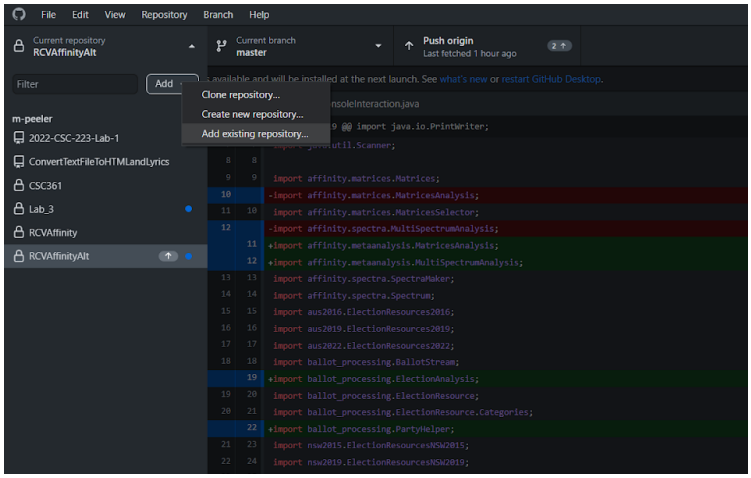


Right click on the project you want to share, then select "**Team**" → "**Share Project.**"

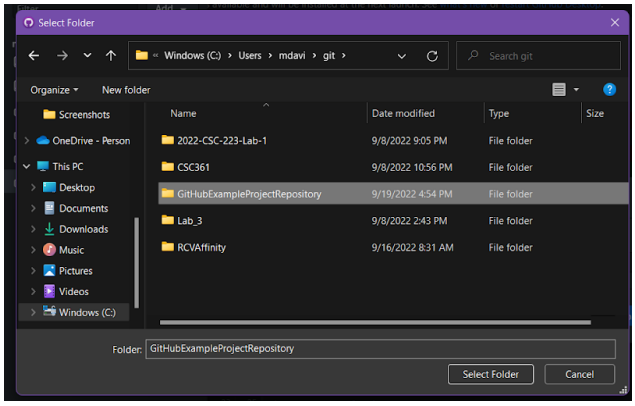


Name your repository, then click "**Finish.**" In the **GitHub Desktop** app, click on "**Add**" → "**Add existing repository.**"

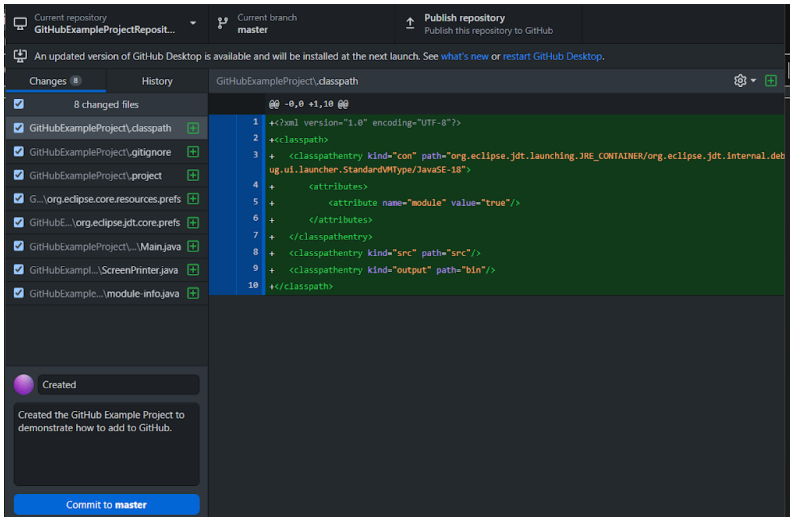




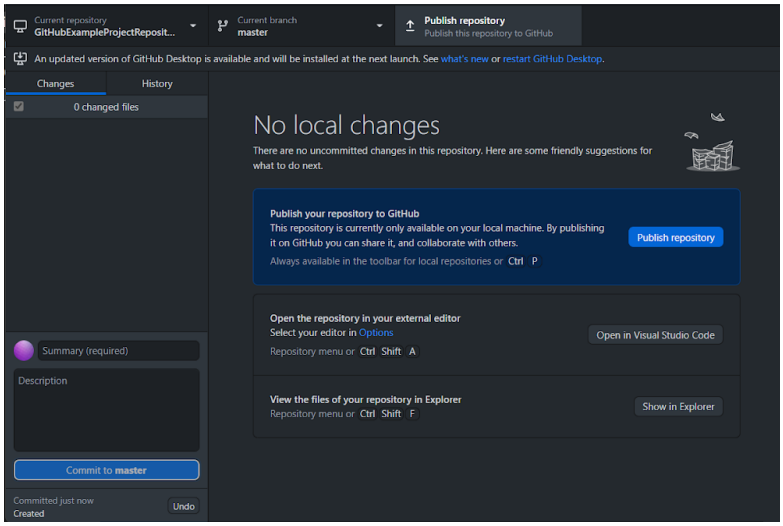
Locate the folder in your system with the repository, select the folder, and then add the repository.



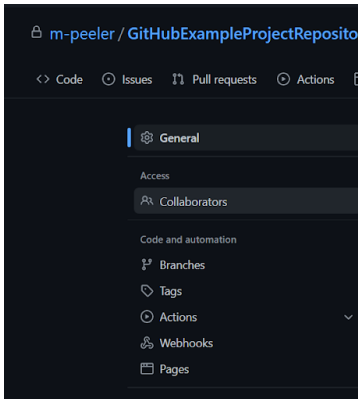
Title your change, give a description, and hit "**Commit**".

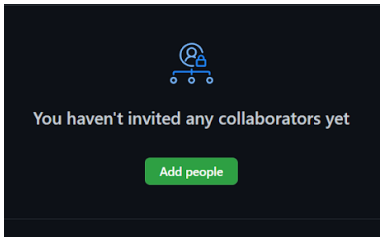


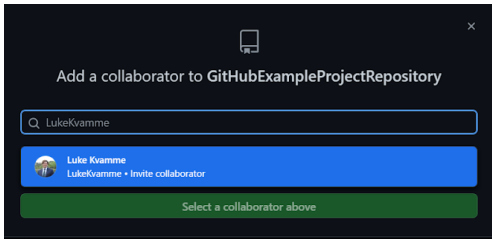
Finally, hit "**Publish repository**," and resolve any potential conflicts. You can also create a description for your repository, and determine whether or not you'd like it to be private.



If your repository is private, you must share it with any collaborators you want to have on the project! Go to the repository on the **GitHub Website**, then click on "**Settings**". Go to "**Collaborators**," and then "**Add people**," and then type in the collaborator you would like to add.

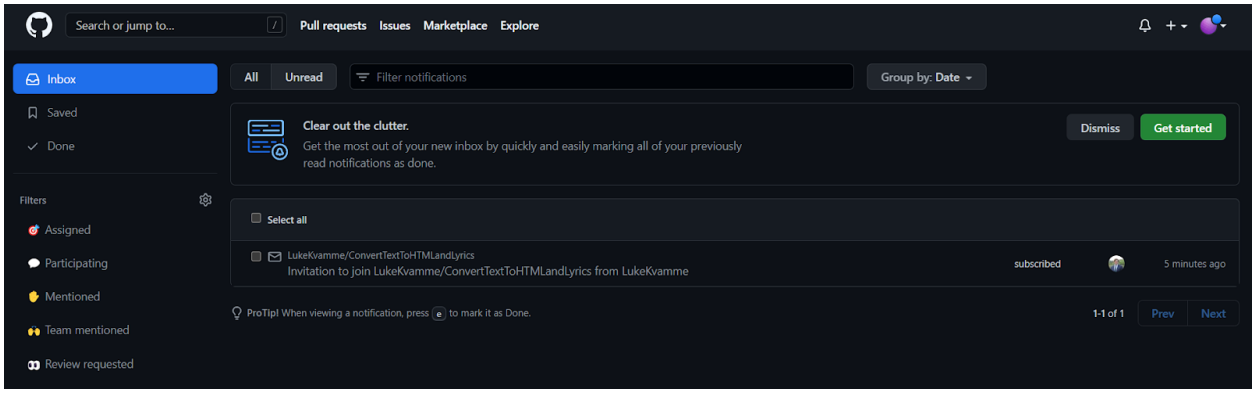




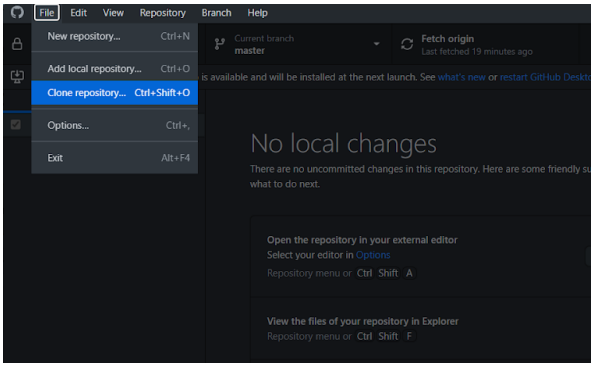


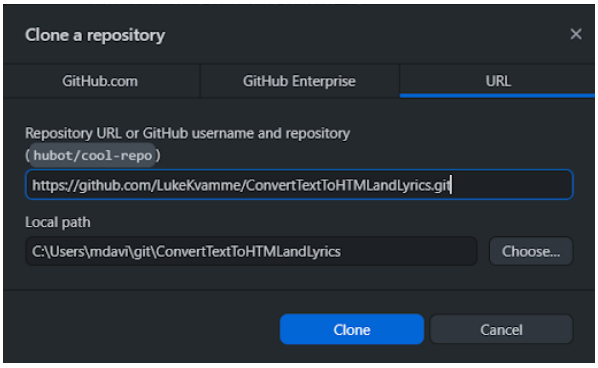
<< Joining and Downloading a Projects >>

For a private repository you have been invited to, first you must click on "**Notifications**" and then accept your invitation.



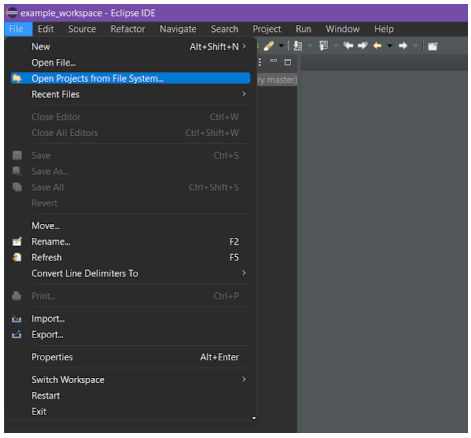
Then, get a link to the repository, and open the **GitHub Desktop** app. Click on "**Clone repository**," select "**URL**," and paste the link. Pick a path for your repository, then click "**Clone**."





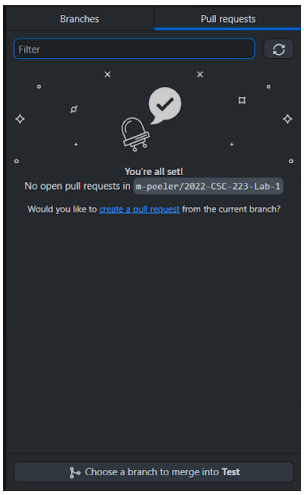
Open **Eclipse**, then select "**File**" → "**Open Projects from File System.**" Find your repository (it will probably be under "**Documents**" → "**GitHub**"), open the folder, and hit "**Finish.**"

Now your project is in **Eclipse**! If you edit it, you can go through the "**Commit**" process in the **GitHub Desktop** app, and "**Push**" it back to the repository.



<< Pull Request >>

If you wish to merge two branches, open the **GitHub Desktop** app, make sure any updates on your branch have been committed and pushed, and then click on the branch name. Under the "**Pull requests**" tab, click on "**create a pull request.**" You will be directed to the **GitHub Website**, where you can write a description of your request and then make it.



<< Sync >>

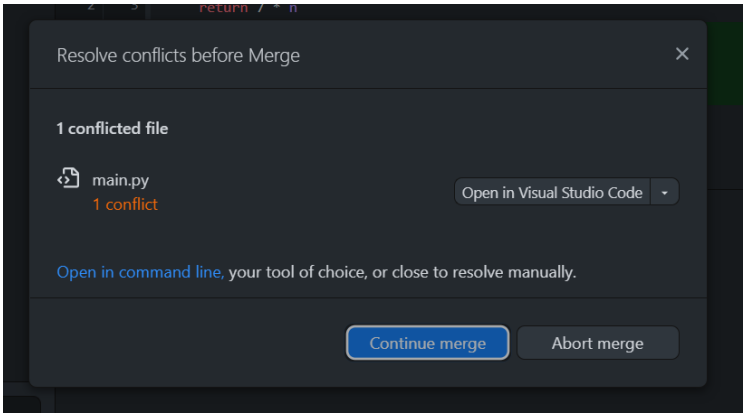
If you have forked the repository, you are working off a version based on your own **GitHub** account. This repository is independent of the original repository, but you can **merge** back to it, and can use it extremely similarly to a branch. However, to keep your fork up to date, you must sync it with the 'upstream' version. The **GitHub Website** will warn you if your fork is falling behind the 'upstream' version by telling you how many commits ahead and behind this version you are. To re-sync your fork, open the **GitHub Website**, select your fork of the repository, and click on "**Sync fork**" and then "**Update Branch.**"

<< Conflicts >>

Merge conflicts happen when you merge branches that have competing commits. Competing commits are a result of either competing-line changes, or when one branch deletes a file while another one edits it.

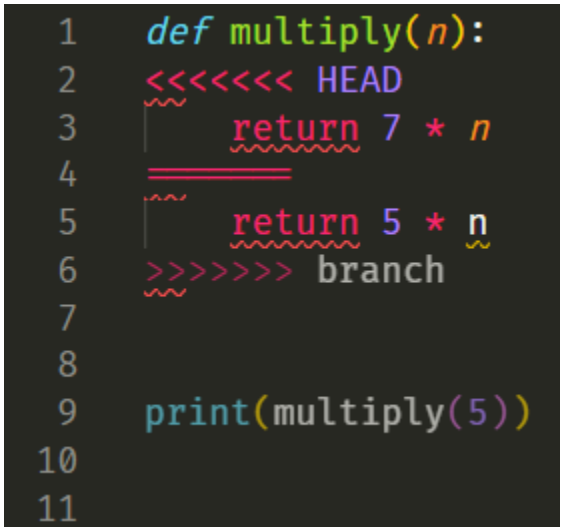
>> Conflicting Branches

When attempting to merge conflicting branches, GitHub Desktop detects all conflicts and will not let you merge the branches until all conflicts are resolved.

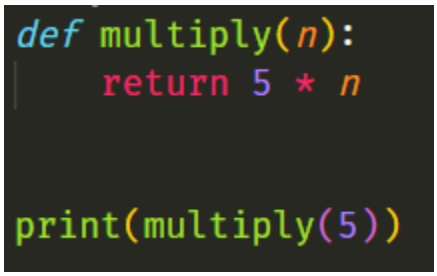


>> Resolving merge conflicts

Git places the conflict markers <<<<<<<, =======, and >>>>>>> around the conflict. The head/base branch will be labeled at the top, while the other branch will be labeled at the bottom. To resolve the conflict, delete the conflict markers and choose what code to keep.

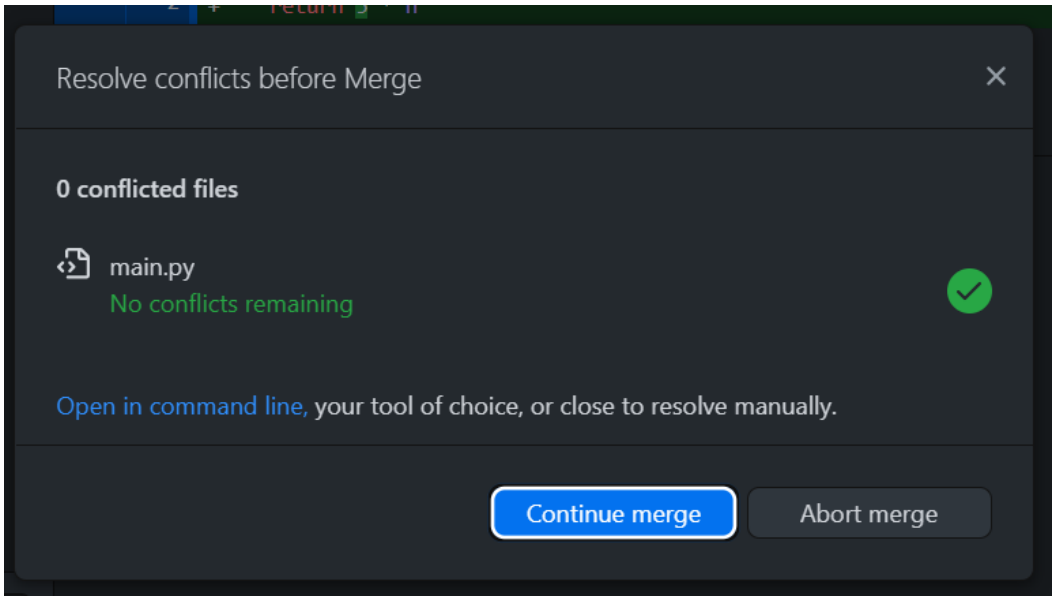


Example A: Merge conflict



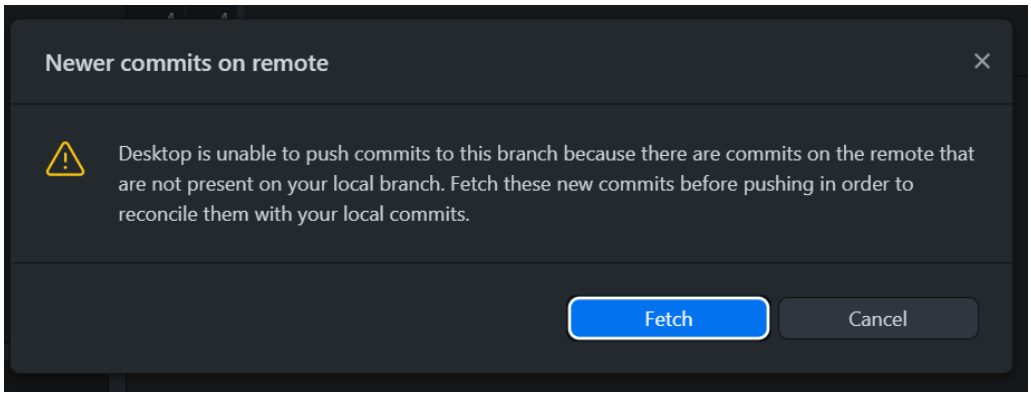
Example B: Resolved!

When all conflicts are resolved, return to GitHub Desktop and you should find no conflicts remaining. At this point, you can merge the branches.

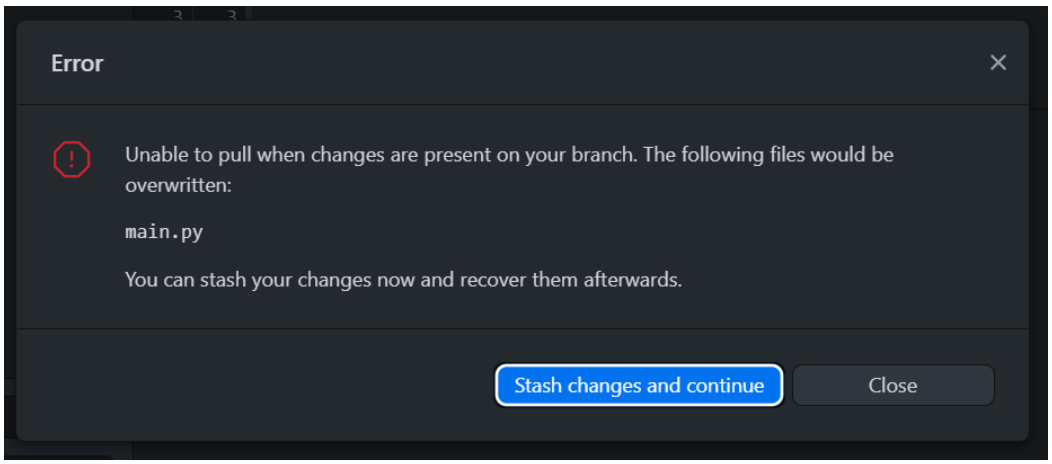


>> "Updated Upstream" Conflicts

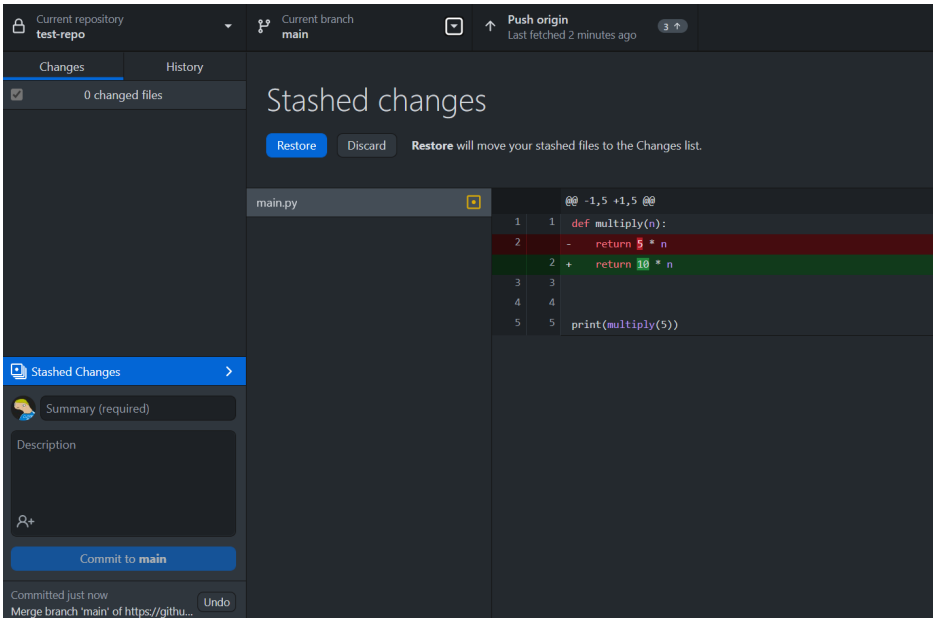
If multiple people are trying to work on the same file, you may run into conflicts. When a file you have changes on gets updated, and you go to push your changes, GitHub Desktop will alert you.



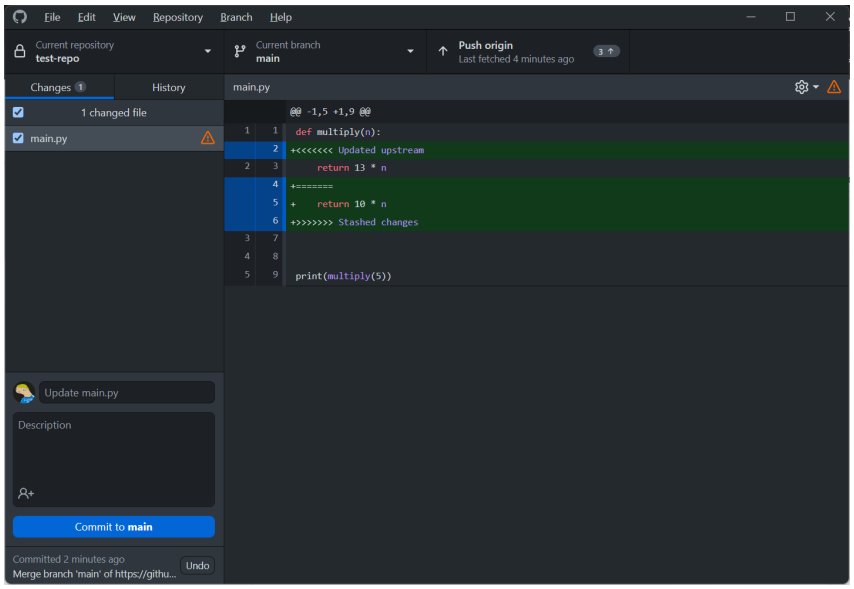
After fetching, you will be asked to stash your changes. BE AWARE! When you stash your changes, your changes ARE NOT DELETED. They are just stored on a different local branch.



To access them, click on **Stashed Changes** in GitHub Desktop. To merge the stashed changes, click the **Restore** button.



After restoring, merge markers will be placed and you can solve the conflicts.



>> Avoiding merge conflicts

While merge conflicts are a normal part of using Git, you can make efforts to run into them a lot less often:

- <-> ***Commit frequently***! The more often you commit, the less conflicts you can run into. There will also be a smaller window of time for conflicts to occur.
- <-> **Don't** work on the same file ***simultaneously***. If you can split up work into multiple files, do that if possible. Multiple people working on the same file simultaneously will usually end up with a lot of conflicts.
- <-> ***Use branches***. This is good git practice anyway, you can work on new features in a branch and then **rebase** to merge the commits to the main branch when finished.
- <-> ***Use consistent formatting***. Try to agree on how files should be formatted, otherwise you may run into a lot of problems when two or more people have different formatting guidelines set in their editor.