Git with R-studio

Jinko Graham

Nov. 14, 2020

1 Introduction

- This document provides information on resolving conflicts that can arise when two or more people work on code and/or documentation in the same GitHub repository.
- In Git, every developer maintains a local version-controlled copy of the repository called the local repository.
- Changes to one's local repository are shared with others through a remote repository on GitHub.
- Conflicts arise when your local copy is out of sync with the remote repository.
- Git can resolve many conflicts on its own, but manual resolution is required when two developers make changes to the same lines in a file.
- Our focus is on manual resolution of conflicts using RStudio.
- Document overview:
  - To understand how to resolve conflicts we need to first cover a few basics of Git and the RStudio interface to Git.
  - We then discuss what happens when there is a conflict and a method to resolve the conflict using RStudio.
- **References**
  - Hadley Wickham's chapter on Git for R package developers who use RStudio [https://r-pkgs.org/git.html]
  - The online git book [https://git-scm.com/book/en/v2]

2 Git basics

- A Git repository is a database of snapshots, or "commits" of files made by one or more developers.
  - Aside 1: Each commit is identified by a unique 40-character "secure hash algorithm" (SHA) key.
  - Aside 2: Commits contain information about the author of the snapshot, the date and time when it was made, as well as the SHA of its parent commit(s).
- GitHub houses a "master copy" of this database.
- The GitHub workflow for a developer is:
  1. Pull a copy of the master from GitHub to start your local repository.
  2. Make changes to the local repository.
  3. When done, push the local repository back to GitHub to be merged with the master. In a push, Git merges the local and master repositories.
- Git can handle most merges on its own. Conflicts arise when we try to push our local version but, since our last pull from the GitHub master, another developer has merged changes to the same lines of the same files that we've changed locally.
  - When this happens, we must manually resolve the conflict.

  - As discussed in Section 4, Git requires that we resolve such conflicts in our local copy of the repository.
  - Thus, to understand the manual conflict resolution process it is helpful to understand more about how Git maintains a local copy of a repository, as this differs from other version control systems like SVN.

2.1 Committing changes to a local Git repository

- (Much of this section is verbatim from the Getting Started chapter of the online Git book [https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F])
- As noted above, the local repository is your database of everybody's commits of files.
  - When we work on the local repository and "commit" our changes, we are committing to the **local** copy of the database. This is in contrast to other version control software like SVN. With SVN, committing changes means committing to the **master** copy of the database on the SVN server.
- In addition to local files being "committed", they can be "modified" or "staged".
  - Modified means that you have changed the file in your "working tree" (see below) but have not committed it to your local database yet.
  - Staged means that you have marked a modified file in its current version to go into the next commit snapshot in your database.
  - Committed means that the data is safely stored in your local database.
- A Git project can be divided into sections: the working tree, the staging area, and the .git directory.
  - The working tree is the file directory that we work in for the Git project. Files in the working tree can be tracked by Git for the database. Tracked files can be modified or unmodified since the last commit to the database.
  - The staging area is comprised of all the files or file modifications that you would like to add to the Git database in your next commit.
  - The .git directory is a subfolder of your working tree. This is your Git database.
- Git workflow:
  1. You modify files in your working tree.
  2. You selectively stage changes you want to be part of your next commit to the Git database.
  3. You commit, meaning that Git (i) takes a snapshot of the files in the staging area and (ii) stores that snapshot permanently in the Git database.

2.2 Working with a remote GitHub repository

- This section is just a recap of what we've learned so far.
- GitHub hosts the master copy of a repository.
  - We push our local Git database *to* GitHub and pull the master copy (comprised of the merged databases of all other developers) *from* GitHub.
- GitHub workflow:
  1. We pull a copy of the remote repository from GitHub into our working tree.
  2. We make changes to our local repository; i.e., we modify files in the working tree, stage them and commit them.
  3. We push the changes in our local repository to the remote GitHub repository.
- Conflicts arise when we try to push our local version but, since our last pull from the GitHub master, another developer has merged changes to the same lines of the same files that we've changed locally.

3 Using Git from RStudio

- The RStudio Git pane (Figure 1) is in the top-right panel of RStudio.

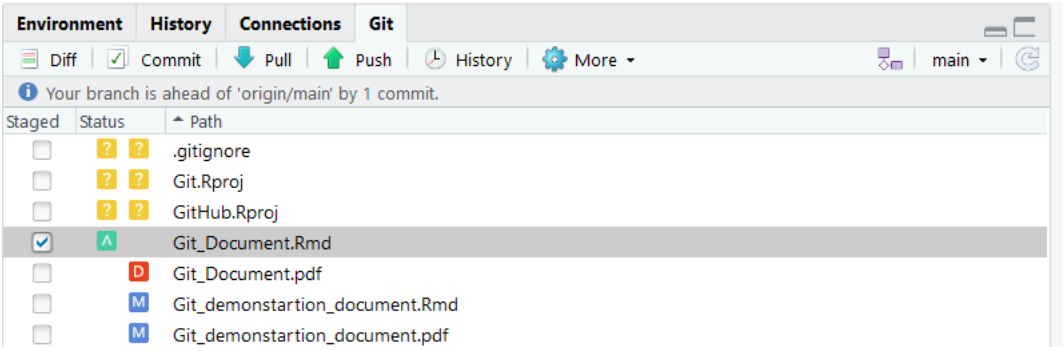

Figure 1: Git pane in RStudio

3.1 Git pane toolbar

- The toolbar across the top of the Git pane includes buttons that execute different Git commands. Of these buttons, Diff, Pull (down-arrow) and Push (up-arrow) are relevant to our discussion of conflicts. There is less to say about Pull and Push so we cover them first.
- Pull copies the remote repository on GitHub to your working tree.
- Push merges your local repository with the master copy on GitHub. Pushing will cause an error if there is a conflict between the local and remote repositories (i.e., databases). These errors are displayed in a pop-up window that is launched when you click Push.
- The Diff button also launches a pop-up window, shown in Figure 2. The Diff window can be used to select files and see all differences between that file's current state and its state in the local repository. This is a good way to remind yourself of your recent changes, to provide context for any conflicts that you may have created. Green background shading indicates additions and red background shading indicates deletions.
  - Aside: If you check the "Staged" button on the Diff pop-up window, as in Figure 2 (see the toolbar just below the list of files), and click on the name of a staged file you see all the modifications of the file as "chunks". At the top-right of each chunk is a button labelled "Unstage chunk" that you can use to selectively unstage changes to a file. Such chunks would not be included in the next commit.

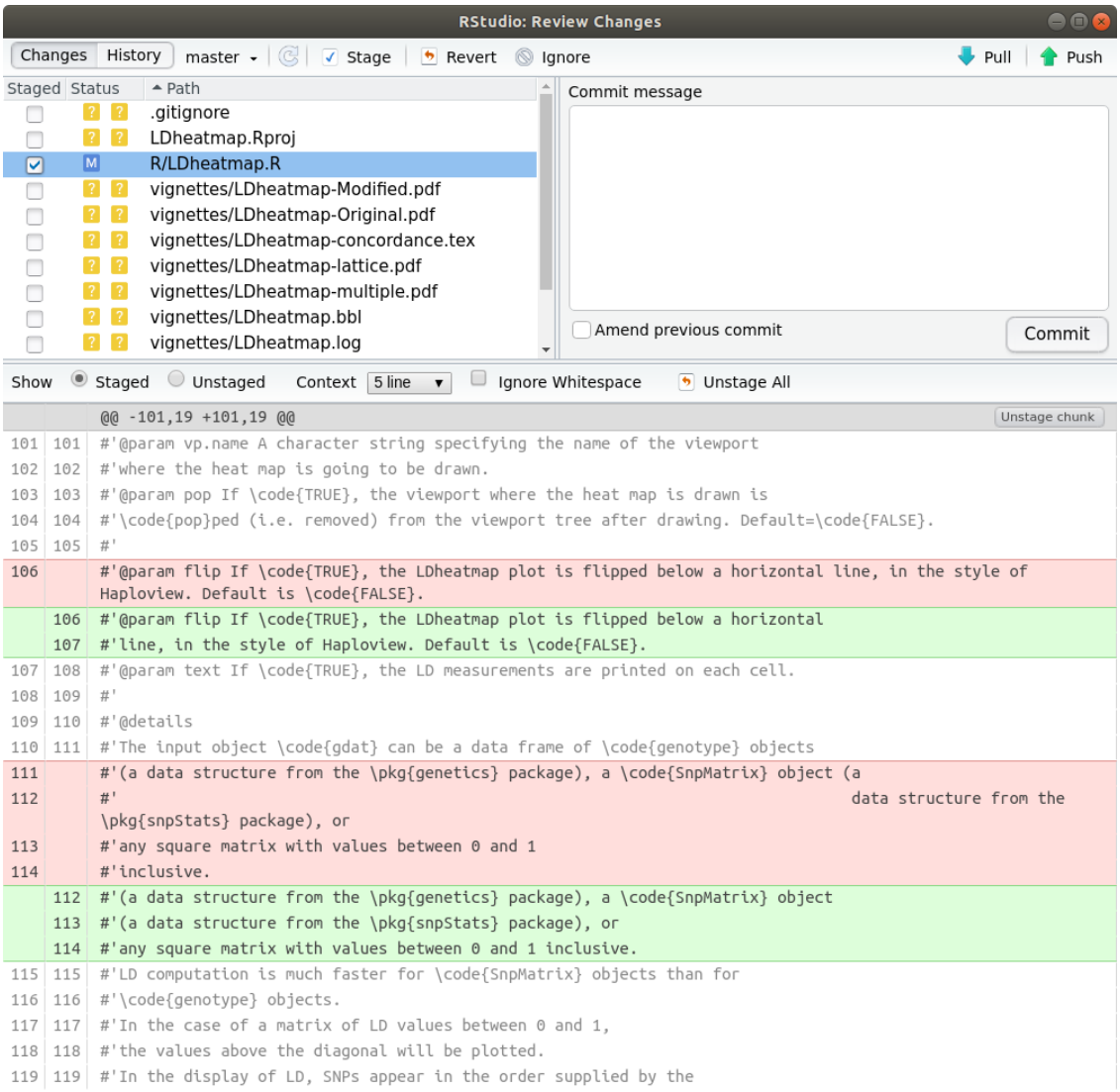

Figure 2: The Diff pop-up window, showing additions and deletions to a file.

3.2 Files list

- Below the toolbar in Figure 1 is a list of files in your working tree that may need to be staged, either because they are untracked and need to be added to the local repository, or because they are tracked and recently modified. When conflicts arise, this area and its check-boxes are part of the conflict resolution process, so we need to briefly describe what we see here.
- The check-box to the left of a file name under "Staged" is checked if the file is staged, and unchecked if not. To stage a file check its box.
- The pair of icons to the left of a file name under "Status" indicate file status in the staging area and working tree, respectively.
  - The left pair member is the status of the file in the staging area, and can be ? for unknown (because it is not part of the local Git database), A if it is to be added to the local Git database, or M if it is tracked and has been modified and staged since the last commit to the local Git database.
  - The right pair member is the status of the file in the working tree and can be ? for unknown (because it is not in the local Git database), M if it is tracked and includes unstaged modifications,

or D if it is tracked and is to be deleted from the database.
  - One exception to the above is that the pair of staging area/working tree icons can be "U U" (always a pair of U's), where U stands for unmerged with the **remote GitHub database**. To my understanding, "U U" here doesn't make sense, because the status of a file in the staging area and working tree is relative to the **local database**, while unmerged is relative to the **remote GitHub database**. I'm guessing that labelling files "U U" in the staging and working tree columns is an exception that the RStudio developers decided to make because they had no other place to denote a file's status as unmerged with the GitHub database.
  - Aside: When you see a pair of M's, this means that you have a first round of modifications that were staged, plus a second round of modifications that have not yet been staged. Clicking the Staged check-box will stage the second round of changes for you. If you don't stage this second round, they will not be included in the commit. If you do, both rounds of changes will be committed to your local repository.

4 Resolving conflicts

- Recall that a conflict occurs between your local repository and the GitHub repository when the same lines of a file differ in the two databases.
- You will be alerted to conflicts when you try to push your local repository to the GitHub repository.
- The Git push pop-up window will show an error message similar to the following:

```
>>> /usr/bin/git push origin HEAD:refs/heads/master
To https://github.com/SFUStatgen/LDheatmap
 ! [rejected]        HEAD -> master (non-fast-forward)
error: failed to push some refs to 'https://github.com/SFUStatgen/LDheatmap'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

- You must resolve the conflicts in your local repository to merge the two databases.
  - Git requires that we resolve conflicts locally. Presumably the idea is to resolve the conflicts in a sort of "sandbox", rather than changing the master copy directly.
- Initiate a pull request, by clicking the Pull button on the Git pane. When you pull you will see an error message in the Git pull pop-up window similar to the following:

```
>>> /usr/bin/git pull
From https://github.com/SFUStatgen/LDheatmap
   dac9e94..6998a8d  master     -> origin/master
Auto-merging R/LDheatmap.R
CONFLICT (content): Merge conflict in R/LDheatmap.R
Automatic merge failed; fix conflicts and then commit the result.
```

- The conflict will cause Git to modify the conflicted file in your working tree but not in your local .git database; that has to wait for you to stage and commit the resulting changes locally (see below).
- The modifications Git makes to the local copy of the conflicted file will be shown as a file diff in a second pop-up window. The text inserted into the conflicted file will look something like the following:

```
<<<<<<< HEAD
 code from your local repo is shown here
=======
 conflicting code in GitHub repo is shown here
>>>>>>> ID
```

- ID will be the 40-character ID (SHA key) of the commit in the GitHub repository that your local repository is in conflict with.

- In the RStudio file editor, go to each conflict and edit the file so that it includes the version of the code that you want.
  - For example, refering to Figure 3, if I wanted to keep my local version, shown on lines 113 and 114, I would delete lines 116 and 117 from the GitHub version, and also delete lines 112, 115 and 118 that Git inserted to mark the conflict.

```
111 #'@details
112 <<<<<<< HEAD
113 #'The input object \code{gdat} can be a data frame of \code{genotype} objects
114 #'(a data structure from the \pkg{genetics} package), a \code{SnpMatrix} object
115 =======
116 #'The input object \code{gdat} can be a  \code{genotype} object,
117 #'a \code{SnpMatrix} object
118 >>>>>>> 8dc2a85052c51f3ed3385e0675b66adaaabf135d
119 #'(a data structure from the \pkg{snpStats} package), or
```

Figure 3: RStudio editor showing text inserted by Git into a conflicted file.

- Now use the RStudio editor to save the changes.
- Over in the Git pane you will notice that the status of the conflicted file in the Git pane is "U U", and that the Staged check-box beside the filename is solid blue (Figure 4).

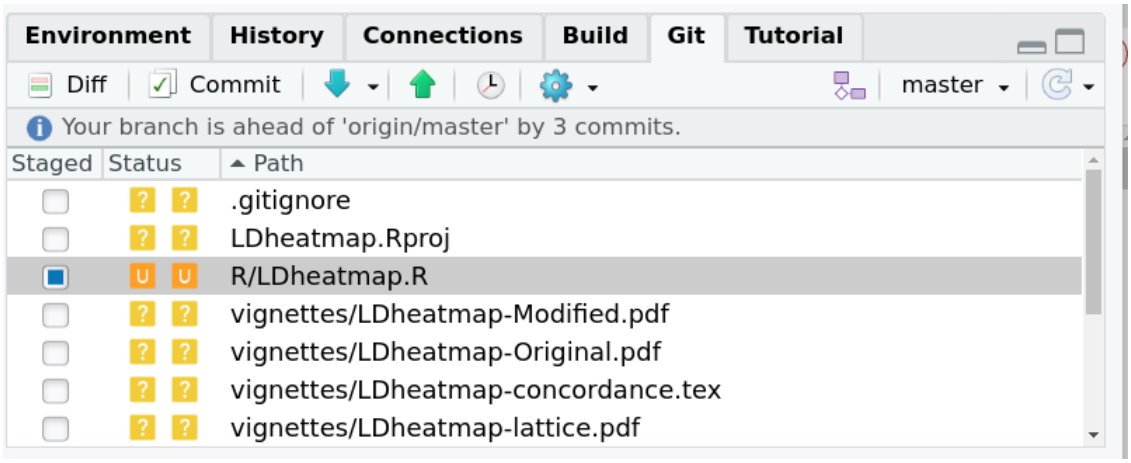

Figure 4: Git pane showing a file with status U U due to a conflict with the GitHub repository.

- From the Git pane:
  1. Click on the Staged checkbox beside the conflicted file so that it changes from solid blue to a blue check-mark. This will stage your changes.
  2. Click the commit icon to commit the changes to the local repository,
  3. Click the push icon to push your local repository to GitHub.
- Aside: There is one special case of the above instructions. If your edits to the conflicted file were to **always** choose the version of the code that is in your local repository (rather than the version in the remote GitHub repository), then when you click the Staged button in step 1, the file will disappear from the Git pane.
  - The reason for the disappearance is that there are no changes to be made to what is already in the local repository, and so there is nothing to commit; i.e., you can skip step 2. However, you still need to push the local version as in step 3.