Data Analysis using R

Version Control

Sven Werenbeck-Ueding

21.09.2023

Introduction

- Data analysis projects are typically conducted by a team of analysts
- Through team work in courses you may be used to storing your group work independently on your PCs and (when a group member finishes a task) send updated files to each other
- Especially when files are often revised, it becomes harder and harder to trace which file reflects the current state of your work
- The larger the group, the more work this kind of workflow means for each group member
- Version control systems such as Git help trace changes made to your files

What is Git?

Git is a version control system used in particular in software development to **trace changes to the source code**. This way of working has been adopted by scientists in order to keep track of the countless changes to the multitude of files that exist within a project.

- Manages the evolution of a set of files in a so-called **repository** (or "repo")
- Data analysis projects are made up of a many files (data, scripts, figures, reports, ...)
    - Often subject to incremental changes
    - Using Git allows to track changes (and in the worst case undo mistakes like accidentally deleting a file)
- Allows for distributed development
    - Each collaborator has a local copy of the repository
    - Able to develop locally without an internet connection

Installing Git

Step 1: Register a GitHub Account

- Register a free GitHub account on https://github.com/
- Remarks on your username (in case you want to use your GitHub account for e. g. your resume):
    1. Use some variant of your actual name
    2. Do not include your current organization/employer
    3. Keep it short
    4. Use lower case only
    5. Use a hyphen − for word separation

Step 2a: Install Git for Windows

- Git for Windows (`msysgit` or "Git Bash") can be downloaded from https://gitforwindows.org/
    - Contains Git and other useful tools such as the Bash shell
- Install Git
    - When asked about "Adjusting your PATH environment", select "Git from the command line and also from 3rd-party software"
    - Otherwise accept the defaults

Step 2b: Install Git for macOS

Option 1

- Install Xcode command line tools
- Go to the shell and enter one of these commands to elicit an offer to install developer command line tools:

```
git --version
git config
```

- Accept the offer and “Install”

Option 2

Install Git from http://git-scm.com/downloads

Step 3: Connect your GitHub account to Git

Start RStudio and connect your previously registered GitHub account to Git using the `usethis` package:

```
# If you do not have the package installed, run:
# install.packages("usethis")

# Replace user name and email with your GitHub user name and the
# email address you registered your account with!

library(usethis)
use_git_config(user.name = "YOUR_NAME",
               user.email = "YOUR_MAIL@ruhr-uni-bochum.de")
```

Step 4: Set up your GitHub credentials

- In order to interact with GitHub (e. g. `pull` a repository or `push` some changes), we have to provide credentials
- For communication with the server, we can choose between two protocols, HTTPS and SSH
- For ease of use, we set up our credentials using HTTPS which requires a **personal access token (PAT)**
- We can do this easily in `R`

Step 4: Set up your GitHub credentials

- Use the `usethis` package to generate a PAT:
- Click "Generate token", copy the generated PAT and store the PAT where you can easily find it
- Next time you are asked for your password, give this PAT

```
usethis::create_github_token()
```

Step 4: Set up your GitHub credentials

i

You can store your PAT using the `gitcreds` package. `gitcreds_set()` will open a prompt, where you can paste your PAT:

```
gitcreds::gitcreds_set()
```

Creating a New Repository

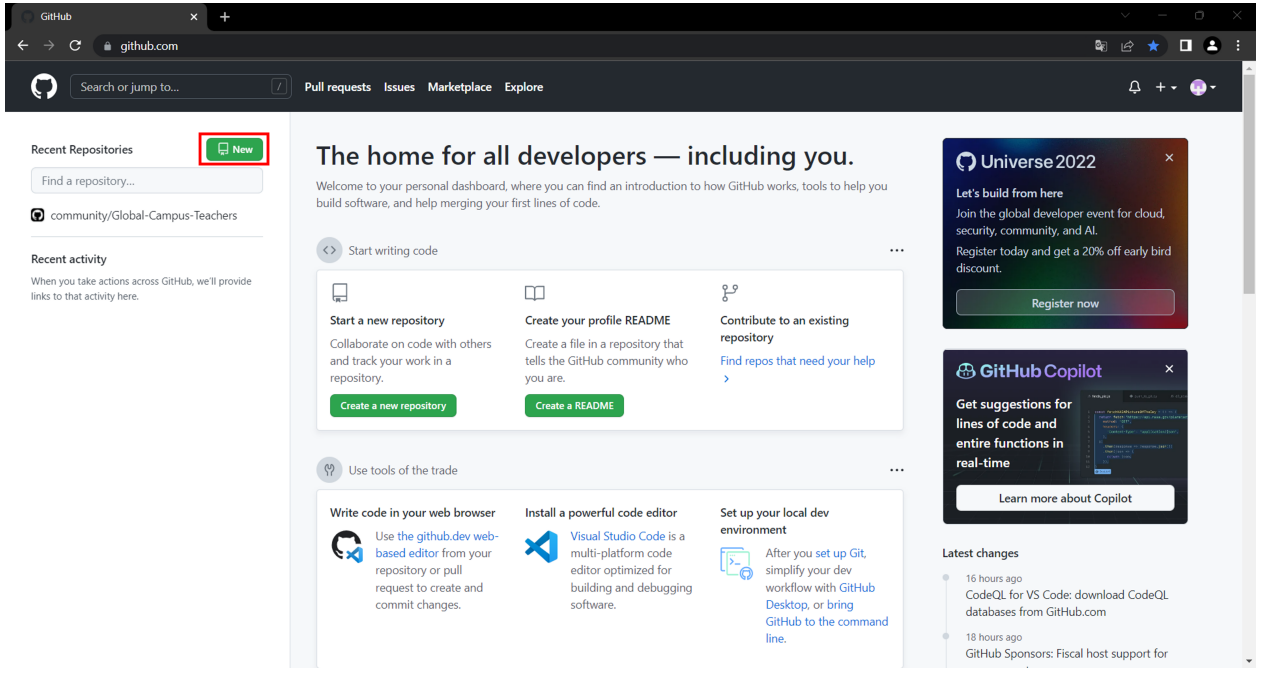
Initializing a Repository

- There are different ways to set up a repository
    - Using Git from the shell
    - Using the `usethis` package
    - Via RStudio IDE
- The easiest way is to create a GitHub repository first, then an R project with version control via the RStudio IDE

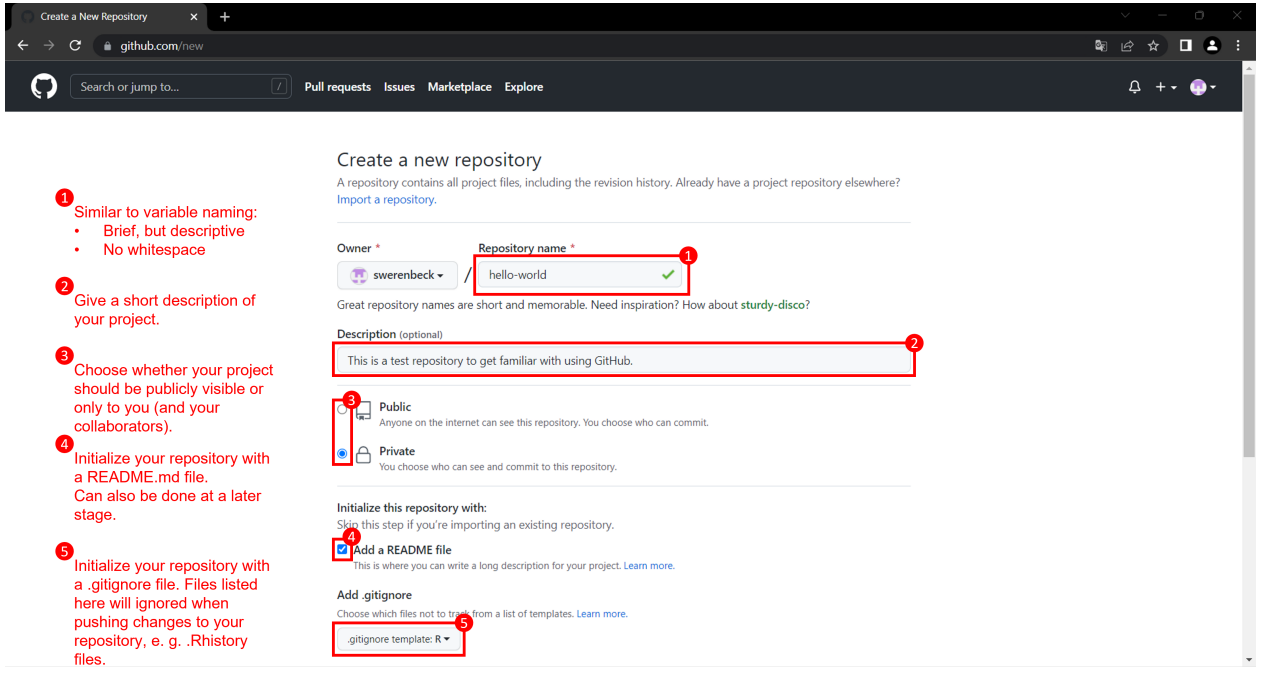
i

If you are interested in the other approaches, check out chapters 15 to 17 of Bryan (2022)!

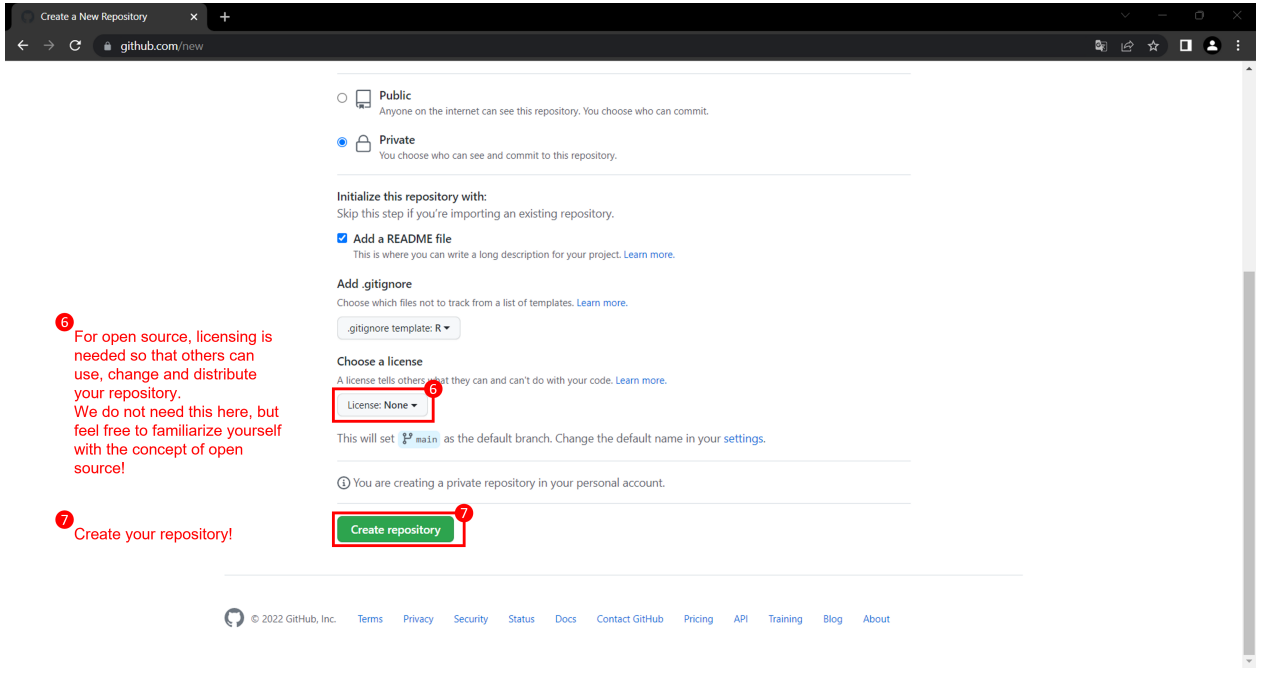
Step 1: Create a new GitHub repository



Step 1: Create a new GitHub repository



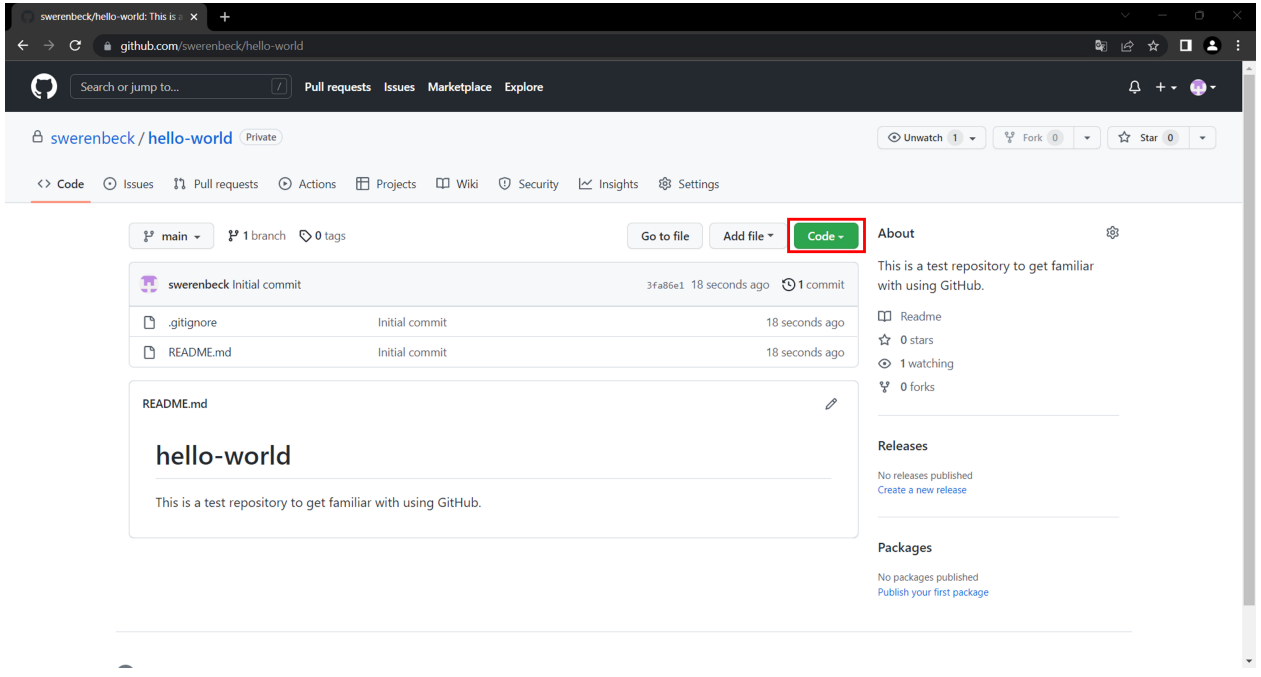
Step 1: Create a new GitHub repository



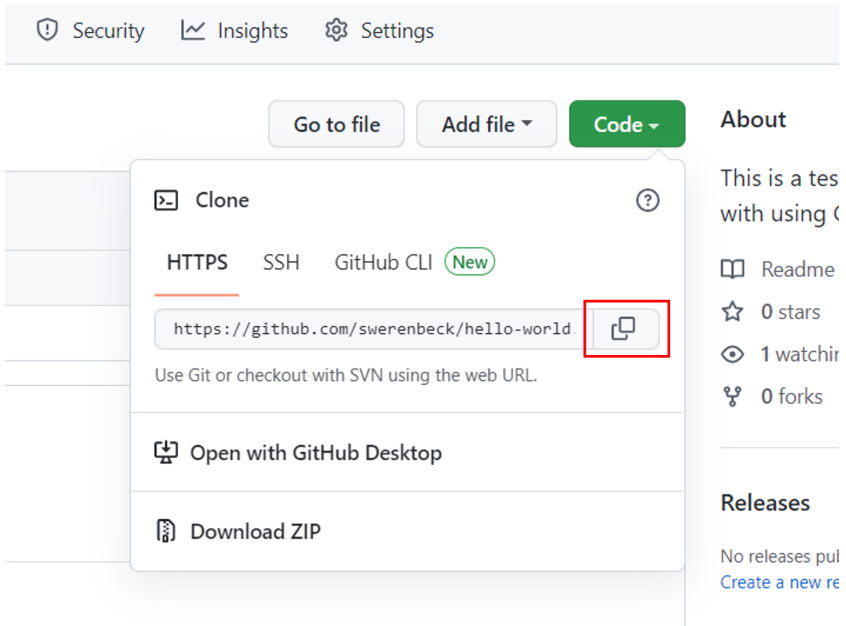
6 For open source, licensing is needed so that others can use, change and distribute your repository.
We do not need this here, but feel free to familiarize yourself with the concept of open source!

7 Create your repository!

Step 2: Inspect your repo and copy its link



Step 2: Inspect your repo and copy its link



Step 3: Create an R project in RStudio

In RStudio, create a new project by clicking on `File > New Project...`

Step 3: Create an R project in RStudio

Select "Version Control"

Step 3: Create an R project in RStudio

Select "Git"

Step 3: Create an R project in RStudio

Paste the repository link into "Repository URL" and click on "Create Project"

Step 3: Create an R project in RStudio

An RStudio Project in a local directory on your computer was created and linked that directory as a Git repository to your remote GitHub repository

Creating Commits

Make Changes to Your Repository

- Your repository is now set up and running. Time to fill it with content!
- Let's do some regression analysis on the `mtcars` data set included in R:

    1. Create a new folder `analyses` inside your project directory
    2. Create an `.R` file with some content, e. g. regressing the gas consumption of cars (`mpg`) on the logarithm of their horsepower (`hp`) and the number of their cylinders (`cyl`)
    3. Save the content of the code on the right to an R file `regression_analysis.R` inside the `analyses` folder
    4. Commit your changes to your Git repository

```
# Create a linear regression model
model <- lm(mpg ~ log(hp) + cyl,
            data = mtcars)

# Print a model summary for "model"
summary(model)
```

What is a commit?

- A commit can be thought of as saving changes to your git repository
- Takes a snapshot of all modified files in the repository at the moment
- The difference in files between two commits is called a "diff"
- Each commit contains a commit message
  - Looking at the commit history makes it possible to track changes
- Commits are initially only local changes
  - To make commits available to collaborators, commits have to be "pushed" to the GitHub repository

Creating a Commit

RStudio's Git pane shows files that were modified, added or deleted. To save these changes to your repository, follow these steps:

- Select files to commit

Creating a Commit

RStudio's Git pane shows files that were modified, added or deleted. To save these changes to your repository, follow these steps:

- Click on "Commit" in the Git pane

Creating a Commit

RStudio's Git pane shows files that were modified, added or deleted. To save these changes to your repository, follow these steps:

- Type a brief commit message and click on "Commit"

Commit Messages

- In software development, a commit is often done when finishing a unit of work (e. g. writing a function that lets you retrieve some data)
- Not feasible for data analysis projects
    - What would a unit of work be in a data analysis?
- Best to think of commit messages as logging work done
    - Does not have to be finished
    - It helps to log work-in-progress (e. g. when finishing for today)

Best Practices

There is no "one way" of writing commit messages but they have to be **consistent** and **informative**!

1. Separate subject from body with a blank line
2. Keep the subject line as short as possible
3. Do not end the subject line with a period
4. Use imperatives for the subject line
5. Use the body to explain *what* and *why* vs. *how*

Examples

Bad

```
Regression analysis on the mtcars data set
```

Good

Pushing Commits

We have changed our local directory by adding files and made a commit to our Git repository. Let's share our code by pushing our commit to the GitHub repository!

Pushing Commits

We have changed our local directory by adding files and made a commit to our Git repository. Let's share our code by pushing our commit to the GitHub repository!

- "Pull" to check whether the local repository is up to date
- While you were working on problem X, your collaborators might have already pushed their solution to problem Y

Pushing Commits

We have changed our local directory by adding files and made a commit to our Git repository. Let's share our code by pushing our commit to the GitHub repository!

- "Push" your commit(s) to the remote GitHub repository

Pushing Commits

We have changed our local directory by adding files and made a commit to our Git repository. Let's share our code by pushing our commit to the GitHub repository!

- Go to your GitHub repository
- The new files should now be visible

Working with Feature Branches

Branches

- Creating a branch means detouring from the main development branch and working on feature without changing the state of the main stream
    - Allows for **parallel development** streams because collaborators don't overwrite each other's code
    - Useful for single-authored repositories to experiment with different problem solving strategies
- Working directly on the main branch is generally considered bad practice
    - Would require to constantly incorporate code changes by your collaborators
    - Leads to conflicting code when working on the same branch

Issues

- The scope of development on a feature branch should always be outlined beforehand (and maybe adjusted later on)
  - Think about *what*, *why* and (to some extent) *how* you want to achieve something before writing your code
  - Don't develop something for the sake of writing code
- GitHub offers an easy to use feature for documenting the scope of a feature branch: Issues
- Issues are shown in the "Issues" tab of your repository and can be assigned to GitHub projects

GitHub Projects

- Another great feature of GitHub is the project management tool
- You can organize and manage your repository using GitHub projects
- Create a project for your repository in the "Projects" tabs
- Give your project a concise and descriptive name
- The "Team backlog" template is best suited for your needs

Managing Your Project

- The default team backlog has the following cards representing different statuses:

    1. **New:** Shows all newly created issues in your project
    2. **Backlog:** Collect all your issues that are not yet ready for development in the backlog
    3. **Ready:** Drag issues that are ready to develop from "Backlog" to "Ready"
    4. **In progress:** When you begin the development of a feature described in an issue, drag the corresponding issue to the "In progress" card
    5. **In review:** When you are done developing the feature, it is good practice to let others review your code
    6. **Done:** Feature that have gone through the review process are set to the status "Done"

Linking Feature Branches and Issues

- Issues can be linked to feature branches by creating a branch on the issue page
- Branches can also be created directly in RStudio by clicking on the purple symbol on the right of the Git pane
- Linking issues to branches is the better approach when you are already documenting the scope of a feature branch in issues and use GitHub projects to manage your repsository

Checkout Branches

- To change the branch you are working on in RStudio, click on the dropdown right to the symbol for creating a new branch in the Git pane
- You have to pull (blue arrow) the remote repository before the newly created feature branch appears in RStudio

Commit and Push

- Let's implement what we outlined in the issue and estimate two specifications of our linear regression model:

  1. $mpg = \alpha + \beta_1 hp + \beta_2 cyl + \varepsilon$
  2. $mpg = \alpha + \beta_1 \log hp + \beta_2 cyl + \varepsilon$

- Commit your changes and push the commit to the repository

```
# Create list of specifications
formulas <- list(
  model_1 = formula(mpg ~ hp + cyl),
  model_2 = formula(mpg ~ log(hp) + cyl)
)

# Estimate each formula in formulas
models <- lapply(formulas,
                 lm,
                 data = mtcars)

# Print model summaries
lapply(models, summary)
```

Pull Requests

Creating a Pull Request

- The process of merging the feature branch into the main branch is called **pull request**
- After pushing your commit to the repository, GitHub conveniantly asks whether you want to create a pull request

Creating a Pull Request

- Open a pull request and assign it to you
- Assign a reviewer and specify the project
- If there is only one commit on the feature branch, GitHub automatically fills subject line and body for you
    - Otherwise give a concise but descriptive subjet line and note what was done
    - Use closing keywords to automatically close the issue when the branch is merged (e. g. "Closes #1" for issue number 1)
- Don't fortget to change the issue status to "In review"

Inspect Pull Requests

- Once the pull request was created, it will be displayed in the "Pull requests" tab of your repository until the feature branch was successfully merged into the main branch
- Creating a pull request automatically creates a task in your project that you can track
- Your reviewers can inspect your commits and write comments as well as make suggestions for code changes
- If all requirements for the feature were met, the reviewer can approve the pull request
- The feature branch can then be merged into the main branch by clicking on "Merge pull request"

Setting Issues on "Done"

- After merging the feature branch into the main branch, the feature branch can be safely deleted
- Because closing keywords were used, both, the issue and the pull request, were automatically set to "Done" in your repository after merging the feature

Resolving Merge Conflicts

Merge Conflicts

- Sometimes, Git identifies competing code lines, e. g. when two persons edit the same lane on two different branches
- Merge conflicts are often a hassle but (if you know what to do) are fairly easy to resolve

Just keep a cool head and do not start doing anything in hope of magically resolving the merge conflict!

Resolving Merge Conflicts Using the Terminal

1. Navigate to the terminal in RStudio (the tab next to the console)
2. Get a list of files responsible for the merge conflict by running `git status` in the terminal
3. Open the files listed as "both modified". Conflicting lines are marked by `<<<<<<<` and separated by `=======`.
    - The base branch is marked as `<<<<<<< HEAD`
    - The code line on the other branch is followed by `>>>>>>> BRANCH-NAME`
    - Adjust the code according to your needs. This may involve combining both code chunks.
4. Add your changes to the commit by running `git add .` in the terminal
5. Commit your changes with a comment:

```
git commit -m "Resolved merge conflict by deleting base code."
```

i

References

Bryan, J. (2022). *Happy Git and GitHub for the useR*. http://www.https://happygitwithr.com/, Last accessed on 2022-09-21.