Git and Github Guide

Kaiyi Ye 35255005

Table of contents

1 Set Up RStudio Project and Create a Quarto File

1.1 Step 1: Create a New Project

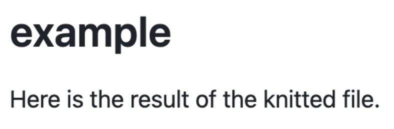
1. Open RStudio.
2. Go to the top menu and select `File` > `New Project....`
3. In the "New Project" wizard, select `New Directory`.
4. Choose `New Project`.
5. Enter a name for your project and select a location on your computer where you want to save it.
6. Click `Create Project`.

1.2 Step 2: Create a Quarto File

1. Go to the top menu and select `File` > `New file` > `Quarto Document`.
2. Save the file as `example.qmd`.
3. Modify the YAML to set the document to render as HTML:

```
title: "example"
author: "your name"
date: today
format: html
```

4. Save the file, then render to preview the document output (see Figure 1).



example

Here is the result of the knitted file.

Figure 1: Render Preview

2 Create a Git Repository from Existing work

If you have an existing project that you want to version control with Git, follow these steps:

2.1 Step 1: Initialize the Git Repository

1. Open your terminal or command prompt
2. Navigate to the root directory of your existing project folder using the `cd` command. For example: `cd path/to/your/project`
3. Run `git init` to initialize this directory as a Git repository. This command creates a new subdirectory named `.git` that contains all of your necessary repository files.

2.2 Step 2: Add Your Files to the Repository

1. Add all of your project files to the staging area: `git add .`
2. Commit the files to the repository with a descriptive message: `git commit -m "Initial commit"`

2.3 Step 3: Set Up the Remote Repository

1. Go to GitHub and create a new repository. Do not initialize it with a README, .gitignore, or license.
2. Copy the URL (SSH) of the new GitHub repository.
3. In your terminal, add the remote repository URL to your local repository:

```
git remote add origin git@github.com:your-username/your-repository.git
```

4. Push your local repository to GitHub: `git push -u origin main`

Now you should be able to see your repository on GitHub.

Tip

We use the `-u` flag to tell Git to remember the connection between our local main branch and the remote origin/main branch.

3 Create a New Branch and Make Changes

3.1 Step 1: Create a New Branch

1. In the terminal, make sure you are inside your project folder: `cd my-new-project`
2. Create and switch to a new branch called `testbranch`:

```
git branch tsetbranch
git switch testbranch
```

Or you can also do this in one command:

```
git switch -c testbranch
```

Tip

The `-c` flag is short for `--create`. It tells git to create a new branch and immediately switch to it.

3.2 Step 2: Make Changes to a File

1. Open the `example.qmd` file in RStudio.
2. Make sure you are working on branch `testbranch`: `git branch`
3. Add a new line to the bottom: `This is a change I made on the testbranch.`
4. Save the file.

3.3 Step 3: Stage and Commit the Changes

1. Check which files were changed: `git status`
2. Stage the file: `git add example.qmd`
3. Commit the change: `git commit -m "Added a line to example on testbranch"`

Your changes made to the file are now saved locally in your local branch.

4 Amend the Last Commit

4.1 Step 1: Create a Folder inside the Project

1. Inside the RStudio project, create a folder called `data`.
2. Place the data into this folder.

4.2 Step 2: Amend the Last Commit to Include the Folder

1. Stage and Commit the Changes:

```
git add .
git commit --amend
```

2. Edit the commit message at the top of the text editor. For example: "Added a line to example on testbranch and created a folder".
3. Save the commit and close the editor.
4. Push this amended commit to the remote: `git push -u origin testbranch`

You should now see the `testbranch` and the commit messages appear on GitHub.

Tip

We need the `-u` flag on the `push` command to set the remote branch as the default tracking branch. After running this, Git will remember that our local local is connected to the remote (origin). From now on, we can simply run `git push` without needing to specify the remote or branch name again.

5 Create a Conflict

1. Switch back to branch `main`: `git switch main`
2. Double check which branch you are currently working on: `git branch`
3. Open the `example.qmd` file and edit the exact same line in `main` that you previously changed in `testbranch` to trigger a conflict. For example: "This line was edited on main."
4. Save the file, then stage and commit the change:

```
git add .
git commit -m"Added the same line on main to make a conflict"
git push origin main
```

6 Fix the Merge Conflict

6.1 Step 1: Merge the `testbranch` into `main`

Try to merge the changes in `testbranch` onto `main` : `git merge testbranch`

You'll see a merge conflict message that Git can't automatically merge the file.

```
<<<<<<< HEAD
This line was edited on main.
=======
This is a change I made on the testbranch.
>>>>>>> testbranch
```

This means:

- Everything above `=======` is from `main`
- Everything after `=======` is from `testbranch`

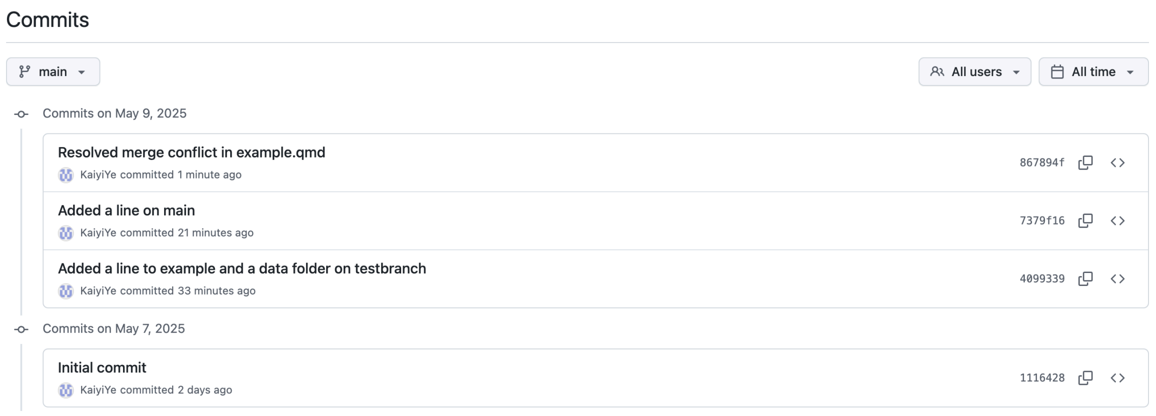
6.2 Step 2: Resolve the Conflict

1. Edit the file to keep only one version — or combine them: "This line includes changes from both branches."
2. Make sure you delete all the conflict markers: `<<<<<<< HEAD` , `=======` and `>>>>>>> testbranch`
3. Save the file.

6.3 Step 3: Finalize the Merge

1. Stage the fixed file: `git add .`
2. Complete the merge with a commit: `git commit -m "Resolved merge conflict in example.qmd"`
3. Push the updated `main` branch to GitHub: `git push origin main`

Now you should see that the GitHub commit history on the `main` branch displays all commit messages we've made before (see Figure 2).


Commits

main

All users

All time

Commits on May 9, 2025

Resolved merge conflict in example.qmd
KaiyiYe committed 1 minute ago
867894f

Added a line on main
KaiyiYe committed 21 minutes ago
7379f16

Added a line to example and a data folder on testbranch
KaiyiYe committed 33 minutes ago
4099339

Commits on May 7, 2025

Initial commit
KaiyiYe committed 2 days ago
1116428

Figure 2: Commit History View in Git

7 Make an Annotated Tag

Annotated tags stores extra metadata, such as the tagger name, email and date.

1. Create a new annotated tag identified with v1.0: `git tag -a v1.0`
2. Write a message for the tag in the text editor. For example: "Initial release version"
3. Save the editor and close it.
4. Push the tag to the remote: `git push origin v1.0`

Now you can view your tags on GitHub.

Tip

By default, `git tag` creates a tag on `HEAD` (i.e., the latest commit). To tag an earlier commit, simply add the commit hash at the end of the command:

```
git tag <tag-name> <commit-hash>
```

8 Delete a Branch

1. Switch to branch `main` : `git switch main`. Remember that you can use `git branch` to double check which branch you are working on.
2. Delete the branch `testbranch` locally: `git branch -d testbranch`
3. Delete the branch from the remote: `git push origin --delete testbranch`

Now you can see that there is only one branch `main` in GitHub.

❗ Important

You cannot delete a branch if your `HEAD` is on that branch.

9 Show the Commit Log

Use `git log --oneline` to show the commit log in the terminal.

The `--oneline` flag condenses each commit to a single line. By default, it displays only the commit ID and the first line of the commit message. Your typical `git log--oneline` output will look something like this:

```
867894f (HEAD -> main, tag: v1.0, origin/main) Resolved merge conflict in example.qmd
7379f16 Added a line on main
4099339 Added a line to example and a data folder on testbranch
1116428 Initial commit
```

10 Make a Plot and Undo the Commit

10.1 Step 1: Make a Plot Using ggplot2

1. Create a new section called "Make a Plot" by using the `#` character.
2. Install the ggplot2 package in the Console then load it:

```r
# load the package
library(ggplot2)

# view the dataset
msleep
```

3. Try to make a plot by using a built-in dataset in ggplot2:

```r
# plot for total amount of sleep for mammals
ggplot(data = msleep, mapping = aes(x = sleep_total)) +
  geom_density() +
  labs(title = "Distribution of Total Sleep Time in Mammals",
       x = "Total Amount of Sleep (hours)",
       y = "Density") +
  theme_classic()
```

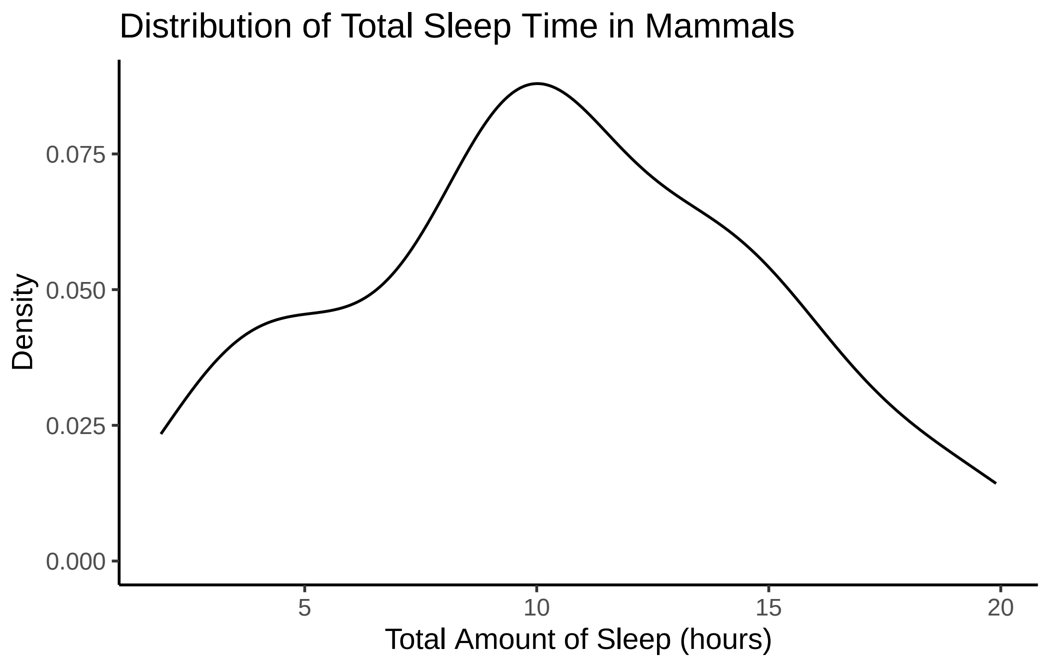


Figure 3: Distribution of Total Sleep Time in Mammals

10.2 Step 2: Stage and Commit the Changes

```
git add .
git commit -m"Add a new section and make a plot"
```

You can use `git log --oneline` to view the commit history. You'll get something like this:

```
bd86707 (HEAD -> main) Add a new section and make a plot
867894f (tag: v1.0, origin/main) Resolved merge conflict in example.qmd
7379f16 Added a line on main
4099339 Added a line to example and a data folder on testbranch
1116428 Initial commit
```

10.3 Step 3: Undo the Commit

If you've just made a commit and realize you want to undo it, you can use the following command — as long as you haven't pushed the commit yet:

```
git reset HEAD~1
```

Now try:

```
git log --oneline
```

You'll see that the latest commit has been removed — `HEAD` has moved one commit backward.

This command undoes the most recent commit but keeps all the changes in your working directory, just no longer staged. You can confirm this by running:

```
git status
```

From here, you can make any adjustments, stage the correct files, and commit again with the proper message or content.

Tip

If you prefer to keep all changes staged, use `git reset --soft` instead.