

5513Assignment2- Git & GitHub Beginner's Guide with RStudio and CLI

Jingyi Wang 35678925

Table of contents

Introduction	1
Step 1: Create a new RStudio Project and a simple Quarto file	1
Step 2: Initialize the folder as a Git repository and push to GitHub	3
Step 3.1: Create and push a new branch testbranch	5
Step 3.2: create a folder called data	8
Step 4: Amend the previous commit	9
Step 5: Switch back to the main branch and cause a merge conflict	12
Step 6: Merge testbranch into main and resolve conflict	12
Step 7: Tag the commit as version 1.0	14
Step 8: Delete the testbranch	16
Step 9: View the commit log	16
Step 10: Add a plot and undo a commit	18
Tips	19
Conclusion	20

Introduction


This guide demonstrates how to efficiently use Git, GitHub, and the command line interface (CLI) for version control and collaboration. By following these steps, you'll master Git for Quarto projects in RStudio.

Step 1: Create a new RStudio Project and a simple Quarto file

First, open RStudio and create a new project named 5513Assignment2 (Figure 1).

New Project Wizard

Back **Create New Project**



Directory name:

Create project as subdirectory of:

☐ Create a git repository

☐ Use renv with this project

☐ Open in new session

Figure 1: Create a new R Project

Inside the project, make a new Quarto document called `example.qmd` (Figure 2). Add some simple content (Figure 3) and click the Render button to knit it into an HTML file (Figure 4).

New Quarto Document

Document
Presentation
Interactive

Title: example

Author: Jingyi Wang

☒ HTML
Recommended format for authoring (you can switch to PDF or Word output anytime)

☐ PDF
PDF output requires a LaTeX installation (e.g. <https://yihui.org/tinytex/>)

☐ Word
Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux)

Engine: Knitr

Editor: ☒ Use visual markdown editor ?

? [Learn more about Quarto](#)

Create Empty Document Create Cancel

Figure 2: Create a new Quarto Document

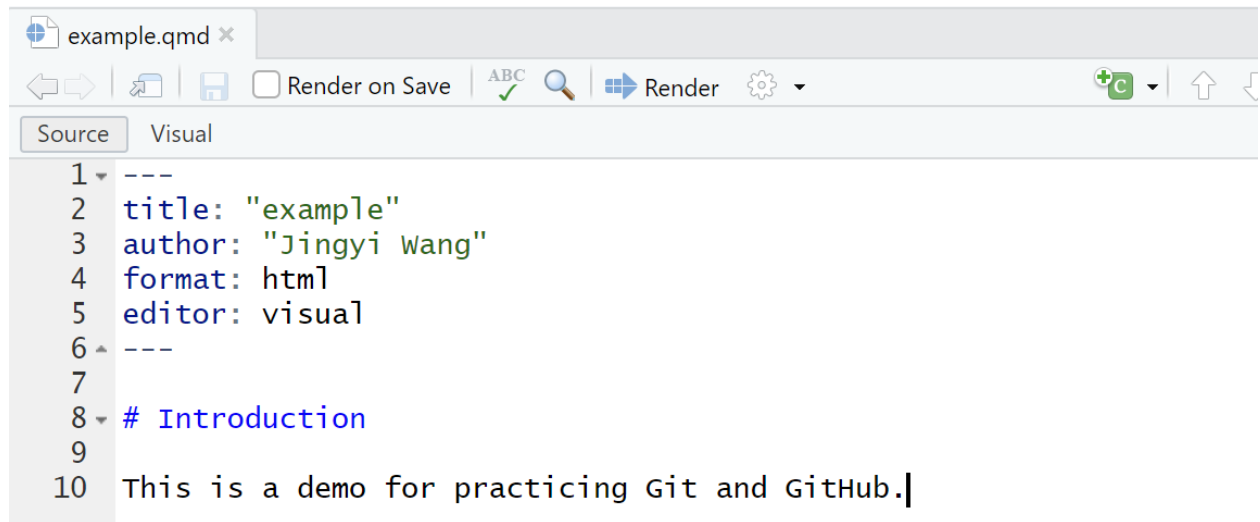
After rendering, your project folder should now contain both the `.qmd` file and the `.html` output (Figure 5).

□ Reflection:

This step is important because it sets up your working space. It connects your folder to Git so it can track changes and link to GitHub later.

Step 2: Initialize the folder as a Git repository and push to GitHub

In RStudio, click the “Terminal” tab at the bottom panel, and run the following commands:



The screenshot shows a code editor window titled 'example.qmd'. The editor has a 'Source' tab selected. The code is as follows:

```
1 ---
2 title: "example"
3 author: "Jingyi Wang"
4 format: html
5 editor: visual
6 ---
7
8 # Introduction
9
10 This is a demo for practicing Git and GitHub.
```

Figure 3: Write some content

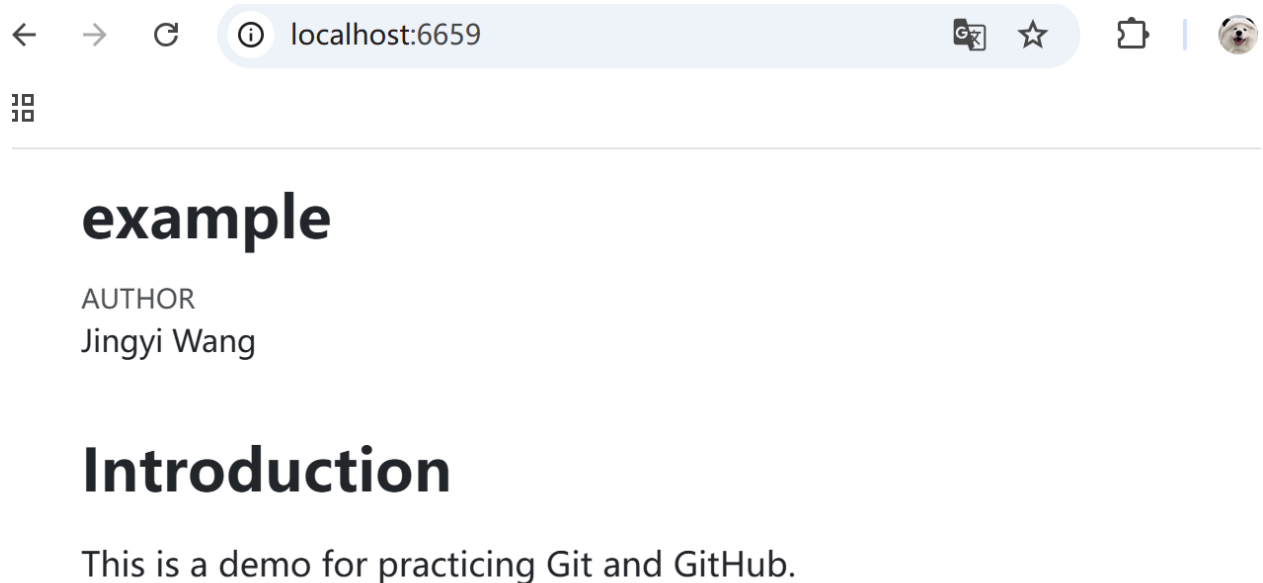


Figure 4: Knit and Create a HTML File

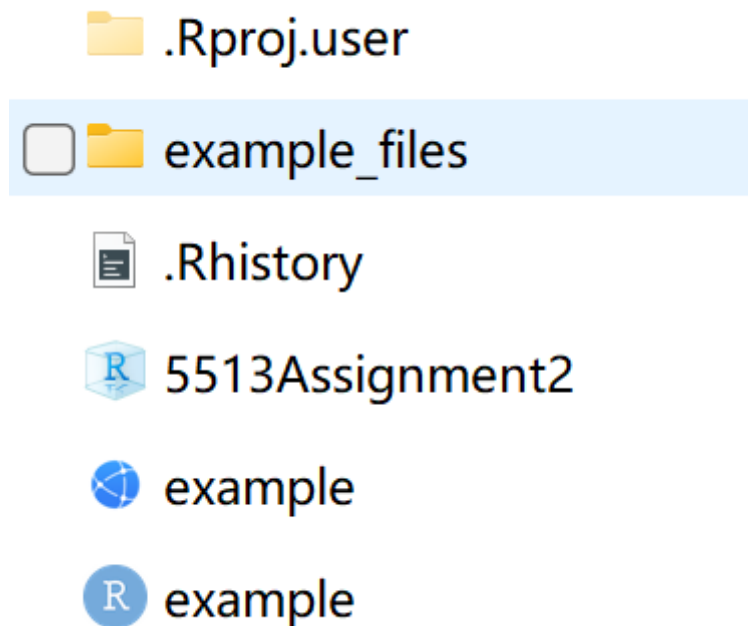


Figure 5: Step1 Final

```
git init
git add .
git commit -m "Initial commit with example.qmd and knitted HTML"
```

Then, on GitHub, create a new repository (Figure 6) and copy the SSH address(Figure 7).

Then link your local repo to GitHub and push it:

```
git remote add origin git\@github.com:Jing0922/5513Assignment2.git
git branch -M main
git push -u origin main
```

□ What this step does:

Now your local project is connected to GitHub. You can store your work online and work with others more easily.

Step 3.1: Create and push a new branch testbranch

There are two different ways to create a new branch and move the HEAD to the new branch:

1. The first method is:

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

2. Another is:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).



Owner * Repository name *

 Jing0922 / 5513Assignment2

✔ 5513Assignment2 is available.

Great repository names are short and memorable. Need inspiration? How about **cautious-funicular** ?

Description (optional)

- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

- ☐ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

 You are creating a public repository in your personal account.

Create repository

Figure 6: Create a New Repository on GitHub

Quick setup — if you've done this kind of thing before

 Set up in Desktop or **HTTPS** **SSH** `git@github.com:Jing0922/5513Assignment2.git`

Figure 7: Copy the SSH

```


10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (master)
$ git remote add origin git@github.com:Jing0922/5513Assignment2.git

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (master)
$ git branch -M main

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git push -u origin main
Enumerating objects: 57, done.
Counting objects: 100% (57/57), done.
Delta compression using up to 22 threads
Compressing objects: 100% (43/43), done.
Writing objects: 100% (57/57), 301.39 KiB | 3.01 MiB/s, done.
Total 57 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To github.com:Jing0922/5513Assignment2.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

```

Figure 8: Connect Local Repo to GitHub in Terminal


5513Assignment2
Public
Pin
Unwatch 1

main
Go to file
+
Code









 Jing0922	Initial commit with example.qmd and knitted HTML	dad8f1e · 15 hours ago	
	.Rproj.user	Initial commit with example.qmd and k...	15 hours ago
	example_files/libs	Initial commit with example.qmd and k...	15 hours ago
	.Rhistory	Initial commit with example.qmd and k...	15 hours ago
	5513Assignment2.Rproj	Initial commit with example.qmd and k...	15 hours ago
	example.html	Initial commit with example.qmd and k...	15 hours ago
	example.qmd	Initial commit with example.qmd and k...	15 hours ago

Figure 9: Connect Local Repo to GitHub

```
git switch -c testbranch
```

After any of the above steps, you can check all local branches, use:

```
git branch
```

You can see there are two branches locally (Figure 10).

```
10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git switch -c testbranch
Switched to a new branch 'testbranch'

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (testbranch)
$ git branch
main
* testbranch
```

Figure 10: Create testbranch Locally

Now open `example.qmd`, add a small change (adding a new sentence on line 11, like Figure 11), and save it.

```
7
8 # Introduction
9
10 This is a demo for practicing Git and GitHub.
11 This is a change after creating a new branch.
```

Figure 11: Edit `example.qmd` in testbranch

Then run these commands:

```
git add example.qmd
git commit -m "Edit example.qmd in testbranch"
git push origin testbranch
```

The result is shown in Figure 12.

☐ Why use branches?

Branches let you test new ideas without changing your main files. This keeps your main project safe.

Step 3.2: create a folder called data

Now still in `testbranch`, we want to create a new folder and add the data from Assignment 1 to it (Figure 13). In the terminal, you can use this command to make a folder:

```
mkdir data
```

To copy the data file, you have two options:


```

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (testbranch)
$ git add example.qmd

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (testbranch)
$ git commit -m "Edit example.qmd in testbranch"
[testbranch 3ac8ef5] Edit example.qmd in testbranch
1 file changed, 2 insertions(+), 1 deletion(-)

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (testbranch)$ git push origin testbranch
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 22 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 352 bytes | 352.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'testbranch' on GitHub by visiting:
remote:   https://github.com/Jing0922/5513Assignment2/pull/new/testbranch
remote:
To github.com:Jing0922/5513Assignment2.git
 * [new branch]      testbranch -> testbranch

```

Figure 12: Add, Commit and Push example.qmd in testbranch

1. Using Git bash, may not always work:

```
cp "/c/Users/10354/5513/etc5513-assignment-1-Jing0922/Data/Meat_con.csv" "./data/"
```

2. Safer way: Just manually copy the file into the data folder using File Explorer.

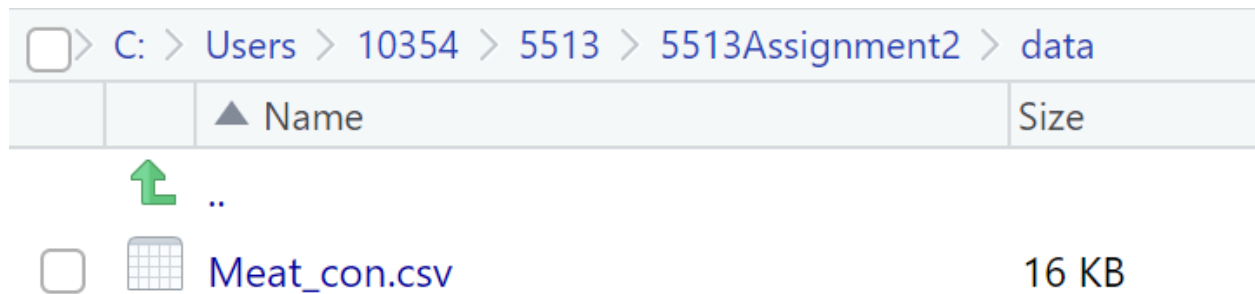


Figure 13: Create a New Folder and Copy the Data

Now add and commit the changes:

```

git add data/
git commit -m "Add data folder with assignment 1 data"

```

The final result is shown in Figure 14.

□ What did we do?

We saved useful data in the repo and tracked it with Git.

Step 4: Amend the previous commit

Sometimes we forget to include something in the last commit. We can fix that with `--amend`.

```

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (testbranch)
$ mkdir data

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (testbranch)
$ cp "/c/Users/10354/5513/etc5513-assignment-1-Jing0922/Data/Meat_con.csv" "./data/"

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (testbranch)
$ git add data/

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (testbranch)
$ git commit -m "Add data folder with assignment 1 data"
[testbranch 41f2149] Add data folder with assignment 1 data
1 file changed, 208 insertions(+)
create mode 100644 data/Meat_con.csv

```

Figure 14: Add and Commit the Changes

First, check the commit history:

```
git log --oneline
```

You will get three commits now(Figure 15).

```

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (testbranch)
$ git log --oneline
41f2149 (HEAD -> testbranch) Add data folder with assignment 1 data
3ac8ef5 (origin/testbranch) Edit example.qmd in testbranch
dad8f1e (origin/main, main) Initial commit with example.qmd and knitted HTML

```

Figure 15: Check the Commit History

Then:

```
git commit --amend
```

This will open the Vim editor (Figure 16) showing the last commit message.

□ **Vim Tip:** If you're new to Vim, remember:

- Press i to enter Insert mode (cursor starts blinking)
- Modify the message as needed
- Press Esc to return to Command mode
- Type :wq then press Enter to: write (save) + quit (exit Vim)

After editing, run:

```
git push origin testbranch --force
```

Now you check the commit history again:

```
git log --oneline
```

You can see that the last commit has been modified(Figure 18).

□ **Why use --amend?**

It lets us fix the last commit instead of making a new one.

```
Terminal 1 MINGW64:/c/Users/10354/5513/5513Assignment2
dd data folder with assignment 1 data

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sun May 4 15:37:41 2025 +1000
#
# On branch testbranch
# Changes to be committed:
#   new file:   data/Meat_con.csv
#
# Changes not staged for commit:
#   modified:   .Rproj.user/181EC6B6/pcs/files-pane.pper
#   modified:   .Rproj.user/181EC6B6/sources/prop/601BDFC2
#   modified:   .Rproj.user/181EC6B6/sources/session-alea1774/B4F7873C
#   modified:   .Rproj.user/181EC6B6/sources/session-alea1774/B4F7873C-contents
#
```

Figure 16: Vim Editor

```
10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (testbranch)
$ git commit --amend
[testbranch 0dffd00] create a data folder and add assignment 1 data
Date: Sun May 4 15:37:41 2025 +1000
1 file changed, 208 insertions(+)
create mode 100644 data/Meat_con.csv
```

Figure 17: Amend the Last Commit Message

```
10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (testbranch)
$ git log --oneline
0dffd00 (HEAD -> testbranch, origin/testbranch) create a data folder and add assignmen
t 1 data
3ac8ef5 Edit example.qmd in testbranch
dad8f1e (origin/main, main) Initial commit with example.qmd and knitted HTML
```

Figure 18: Check the Commit History after Amending

Command	Use Case	Safety
<code>git commit --amend</code>	Edit the most recent commit	Medium
<code>git reset</code>	Rewind to any past commit	High

□ Why use `--force`?

Because the history changed, GitHub needs a forced update to accept it.

□ Warning:

Amending rewrites Git history. After amend, you must use `git push -f`. This is **dangerous** if others are working on the same branch! Always check with your team first.

Step 5: Switch back to the main branch and cause a merge conflict

Now we go back to the main branch:

```
git checkout main
```

Notice that the changes of `example.qmd` made in `testbranch` (the line 11 modification shown in Figure 11) are now absent. **Don't be worried!** This is expected behavior when switching branches.

Now, we deliberately create a conflict. Open `example.qmd` and edit the same line 11 (but with different content than in `testbranch`). And add new content to other lines (as shown in Figure 19). Save it.

```

7
8 # Introduction
9
10 This is a demo for practicing Git and GitHub. I hope you can learn something
   useful!
11 This is a change in origin main.
12 Have a nice day!
```

Figure 19: Edit `example.qmd` in main

Then run:

```
git add example.qmd
git commit -m "Conflicting change in main branch"
git push origin main
```

□ Why do this?

We're preparing for a merge conflict by changing the same line in two branches.

Step 6: Merge `testbranch` into main and resolve conflict

Now we try to combine the two branches:

```
git merge testbranch
```

```

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git add example.qmd

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git commit -m "Conflicting change in main branch"
[main 42cfb0] conflicting change in main branch
1 file changed, 3 insertions(+), 1 deletion(-)

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 22 threads
Compressing objects: 100% (3/3), done.
writing objects: 100% (3/3), 446 bytes | 446.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:Jing0922/5513Assignment2.git
dad8f1e..42cfb0 main -> main

```

Figure 20: Add, Commit and Push example.qmd in main

❑ Critical Check:

You *must* be on the main branch before merging. If you accidentally run `git merge main` while on `testbranch`, the merge direction will be reversed!

A conflict will appear in `example.qmd`. It looks like Figure 21. There are a lot of markers, such as `<<<<<<`, `=====` and `>>>>>>`.

```

7
8 # Introduction
9
10 <<<<<< HEAD
11 This is a demo for practicing Git and GitHub. I hope you can learn something
12   useful!
13 This is a change in origin main.
14 Have a nice day!
15 =====
16 This is a demo for practicing Git and GitHub.
17 This is a change after creating a new branch.
18 >>>>>> testbranch
19

```

Figure 21: A Conflict when Combine Two Branch

Open the file and manually fix the content. You can keep both lines, delete one, or rewrite(Figure 22). Save it.

Then:

```

git add example.qmd
git commit -m "Resolve merge conflict"
git push origin main

```

```

7
8 # Introduction
9
10 This is a demo for practicing Git and GitHub. I hope you can learn something
    useful!
11 This is a change in origin main & a new branch--testbranch.
12 Have a nice day!|

```

Figure 22: Resolve Conflict

To check status:

```
git status
```

The result is Figure 23. In GitHub, the `example.qmd` looks like Figure 24.

```

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git status
on branch main
Your branch is up to date with 'origin/main'.

changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   .Rproj.user/181EC6B6/pcs/files-pane.pper
    modified:   .Rproj.user/181EC6B6/pcs/windowlayoutstate.pper
    modified:   .Rproj.user/181EC6B6/sources/prop/601BDFC2
    modified:   .Rproj.user/181EC6B6/sources/session-a1ea1774/B4F7873C
    modified:   .Rproj.user/181EC6B6/sources/session-a1ea1774/B4F7873C-contents

no changes added to commit (use "git add" and/or "git commit -a")

```

Figure 23: The Status after Resolving Conflict Locally

☐ Why do conflicts happen?

Conflicts happen when both branches edit the same lines. Git needs your help to resolve it.

☐ Why we do this:

Merging brings together updates from different branches. It's a key part of collaboration and ensures everyone's work fits together.

Step 7: Tag the commit as version 1.0


Now we want to mark the current commit as a version.


Use this command to create an annotated tag:

```
git tag -a v1.0 -m "Release version 1.0"
git push origin v1.0
```

This creates a tag named "v1.0", which will appear on GitHub(Figure 25).






☐ Reminder:

 main 5513Assignment2 / example.qmd Go to file t ...

 **Jing0922** Resolve merge conflict f547656 · 4 minutes ago 

12 lines (10 loc) · 253 Bytes



Code Blame

 Raw     


```
1  ---
2  title: "example"
3  author: "Jingyi Wang"
4  format: html
5  editor: visual
6  ---
7
8  # Introduction
9
10 This is a demo for practicing Git and GitHub. I hope you can learn something useful!
11 This is a change in origin main & a new branch--testbranch.
12 Have a nice day!
```

Figure 24: The example.qmd after Resolving Conflict in GitHub

Releases Tags Create release from tag Delete


 v1.0
 f547656
Compare

v1.0

 Jing0922 tagged this 1 minute ago

Release version 1.0

▼ Assets 2

 [Source code \(zip\)](#) 1 minute ago


 [Source code \(tar.gz\)](#) 1 minute ago

Figure 25: Create an Annotated Tag

Tags are like bookmarks. They help you find key points in your project history. Annotated tags (-a) store extra metadata including who created them and when.

Step 8: Delete the testbranch

Now that testbranch has been successfully merged into main, we can safely remove it to keep our repository organized.

❑ Critical Check:

You can not delete your currently active branch, which means you *can not* be on the testbranch.

Delete the branch locally:

```
git branch -d testbranch
```

Check branch, use:

```
git branch
```

Delete the branch from GitHub (remote):

```
git push origin --delete testbranch
```

Now, only the main branch will be left locally(Figure 26) and in your GitHub repo(Figure 27).

```
10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git branch
* main
  testbranch

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git branch -d testbranch
Deleted branch testbranch (was 0dffd00).

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git branch
* main
```

Figure 26: Delete the Branch Locally

❑ Why delete branches?

Deleting old branches keeps your project tidy and organized. Just remember: Git won't let you delete the branch you're currently using.

Step 9: View the commit log

You can check the history of your changes by using:

```
git log --oneline
```


Branches

Overview

Yours

Active

Stale

All

Q

Search branches...

Default

Branch	Updated
<div>main</div> <div></div>	<div></div> 15 minutes ago

Figure 27: Delete the Branch from GitHub

```
10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git log --oneline
f547656 (HEAD -> main, tag: v1.0, origin/main) Resolve merge conflict
42cfeb0 conflicting change in main branch
0dffd00 create a data folder and add assignment 1 data
3ac8ef5 Edit example.qmd in testbranch
dad8f1e Initial commit with example.qmd and knitted HTML
```

Figure 28: Check the Commit History

This shows a short list of all your commits with commit IDs and messages(Figure 28).

❑ Why do this?

Viewing the commit history shows who made changes, when, and what was done. It's especially helpful for tracking team work or reviewing your progress.

Step 10: Add a plot and undo a commit

Add a new section to `example.qmd` and draw a plot, like Figure 29.

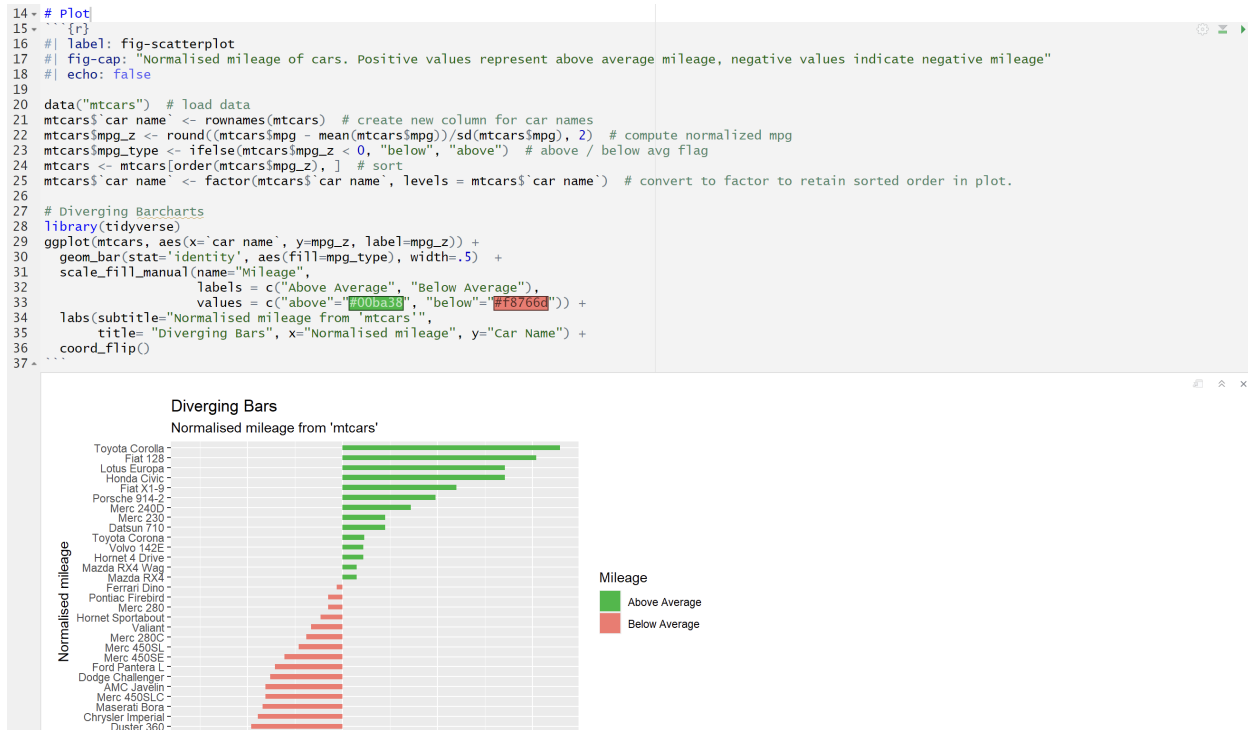


Figure 29: Add the Plot

Then add and commit it:

```
git add example.qmd
git commit -m "Add simple plot"
```

To undo the most recent commit but keep all changes in your working directory, use a **soft reset**:

```
git reset --soft HEAD~1
```

This command removes the last commit but keeps the changes in the staging area.

To verify the changes:

```
git status
```

You can see that all previously committed changes now appear as “Changes to be committed” (shown in green)(Figure 30)

```

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git reset --soft HEAD~1

10354@LAPTOP-RTKEIEJS MINGW64 ~/5513/5513Assignment2 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   example.qmd

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .Rproj.user/181EC6B6/pcs/files-pane.pper
        modified:   .Rproj.user/181EC6B6/pcs/windowlayoutstate.pper
        modified:   .Rproj.user/181EC6B6/sources/prop/601BDFC2
        modified:   .Rproj.user/181EC6B6/sources/session-a1ea1774/B4F7873C
        modified:   .Rproj.user/181EC6B6/sources/session-a1ea1774/B4F7873C-contents
        modified:   .Rproj.user/shared/notebooks/C26CF219-example/1/181EC6B6a1ea1774/chunks.json
        modified:   .Rproj.user/shared/notebooks/C26CF219-example/1/s/chunks.json

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .Rproj.user/shared/notebooks/C26CF219-example/1/s/cpsqa3xqeqjha/

```

Figure 30: The Status after Soft Reset

□ Why do this?

Sometimes we want to fix a commit message or add more things before pushing it. Soft reset is useful when you want to change the commit message or add more to the commit.

Extended Knowledge: Other Reset Modes

Mixed Reset(Default)

```
git reset HEAD~1
```

This command unstages changes but keeps them in working directory. If you verify with git status, changes show as unstaged in red.

Hard Reset(□ Dangerous!)

```
git reset --hard HEAD~1
```

This command completely discards the commit and **all** local changes. No recovery unless you used git reflog.

□ Be careful:

1. Soft reset (--soft) → Keeps changes staged.
2. Mixed reset (--mixed) → Keeps changes but unstages them.
3. Hard reset (--hard) → Deletes everything in the commit.

Tips

There are some tips for you:

- Use clear commit messages like “Add plot section to qmd” or “Fix conflict between branches”.
- Keep commits small and focused.

- Always use branches when testing or adding new features.

Conclusion

This guide gives you a strong start with Git, GitHub, RStudio and CLI. By learning these steps, you can manage your work more clearly and work better with others.