Written Guide:

Reproducible Practices Using Git, GitHub, and RStudio

Akbar Rakha Syahpradana (34096647)

1. Setting Up Your Project and Version Control (Git)

This section will guide you through creating a Git repository from existing work and setting up the remote repository.

Step 1: Create Your R Studio Project and QMD File

1. Open R Studio.
2. Create a new project.

   > How to create new project?
   >
   > Go to **File > New Project > New Directory > New Project**.

   Enter the project's name and browse the location where you want to store your project, and click create project .
3. Create a new `.qmd` file.

   > How to create new .qmd file?
   >
   > Go to **File > New File > Quarto Document**

   Enter the title and author's name. You can also choose the output (e.g. HTML or PDF or Word).
4. The section at the top of your `.qmd` file (between`---`) is the YAML header. Here, you can change the title, subtitle, author name, output format (e.g., PDF or HTML), and theme. The output for HTML would be like Figure 1
5. Don't forget to save your file and rename it.

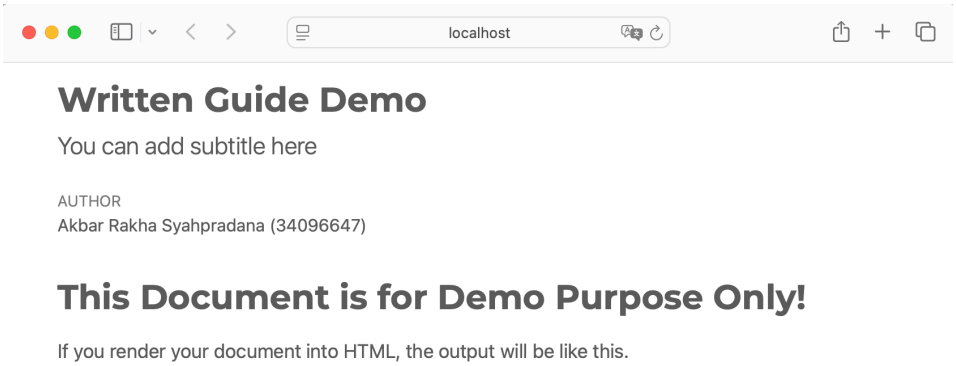


Figure 1: HTML Output.

6. Create a `.gitignore` file to prevent Git from tracking files that shouldn't be included in version control.

   For example, we don't want Git to track the .Rproj.user/ folder, which is automatically created when we make a new R project. To exclude it, we need to include that line in the text file.

   > **Creating .gitignore file**
   >
   > Go to **File > New File > Text File**
   > type .Rproj.user/ and save the file as .gitignore

> **Caution**
>
> This setup only creates a new project **without version control**. It simply creates a new project folder in your chosen location.

Step 2: Initialise The Git Repository

1. Open your terminal in R Studio
2. Make sure it has the correct path (See Figure 2). If not, use `cd` command to change the directory

```
cd path/to/your/project
```

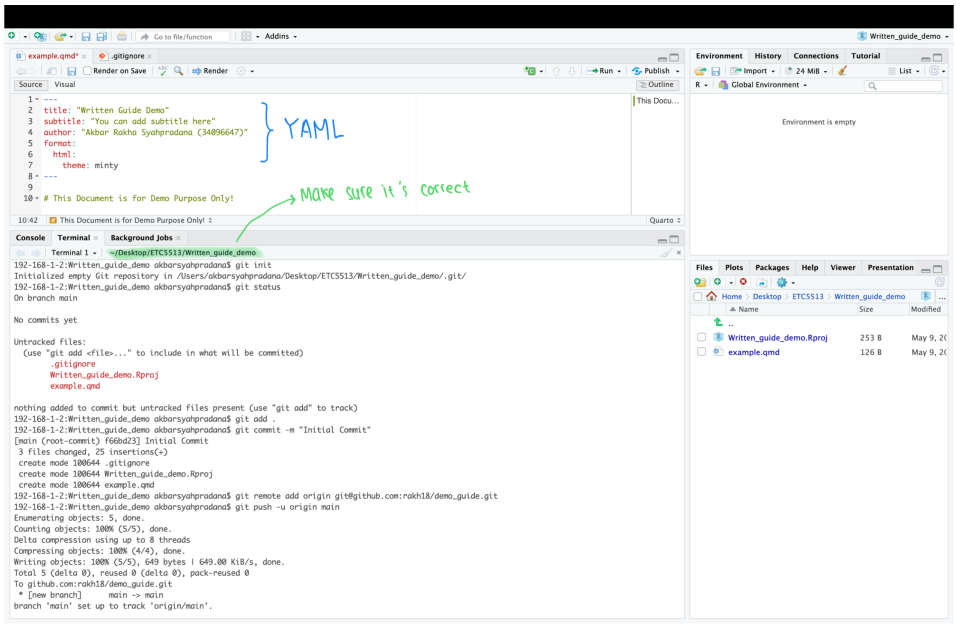


Figure 2: Correct Directory.

3. If you are in the correct directory, initialise the new Git repository using this command in your terminal:

```
git init
```

4. Add all files to staging area using `git add`. You can use `git add .` to stage all the files, or just use `git add your_file_name` for specific file.

5. Check git status and commit to the repository.

```
git status
git add .
git commit -m "Input your commit message here"
```

6. Set up your remote repository.

   Go to GitHub and create a new repository without README, .gitignore, or license (See Figure 3)

7. Copy the **SSH URL** (See Figure 3).

8. Go back to your terminal in R Studio, add the remote repository URL to your local repository and push to GitHub.

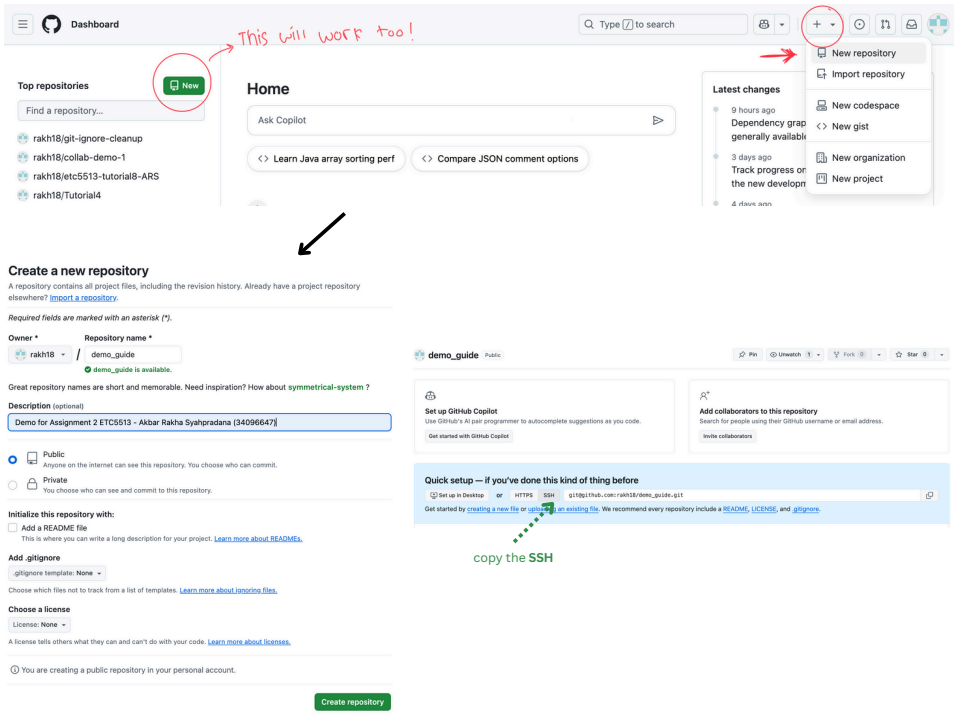


Figure 3: New Repository and Copy SSH URL from GitHub.

```
git remote add origin <paste your SSH URL here>
git push -u origin main
```

9. (Optional but recommended) Add a license to your repository.

   A license gives clear permission for others to use your work. Without a license, even if your project is public, others do not have the right to copy, use, or share it. Adding a license shows what people are allowed to do with your code and helps protect your work.

   > How to add license?
   >
   > Go to **New file > Create new file > Text File**, Copy and paste from LICENSE Don't forget to commit and push:
   >
   > ```
   > git add LICENSE
   > git commit -m "Add license file"
   > git push
   > ```

2. Working with Branches and Making Changes

This section will guide you through creating a new branch, making changes to your project, and pushing those changes to GitHub. You'll also learn how to amend a commit.

Create a new branch

1. Go to your R Studio terminal.
2. You can create new branch by `git branch branch_name` and switch to that branch using `git switch branch_name` See Figure 4.

   > Or do this in one command
   >
   > git switch -c branch_name

3. If you are currently on the new branch, you can make changes and commit them using:

   ```
   git add .
   git commit -m "Your Messages"
   ```

4. You can check your branches in GitHub (See Figure 5).

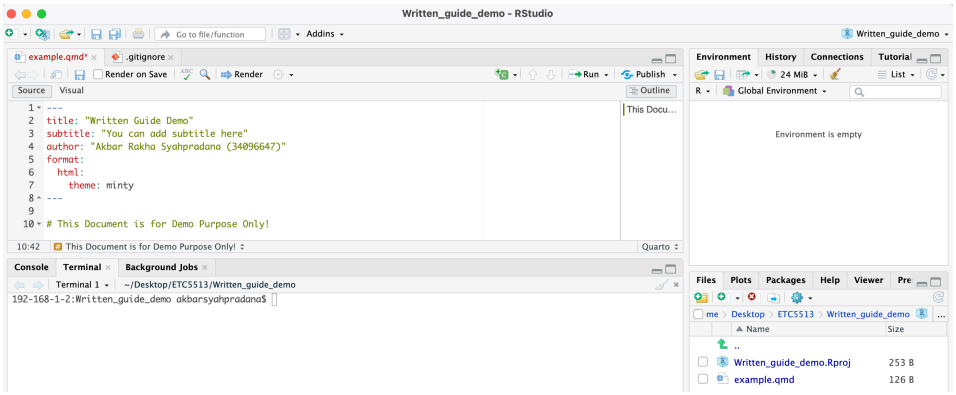


Figure 4: Creating Branch.

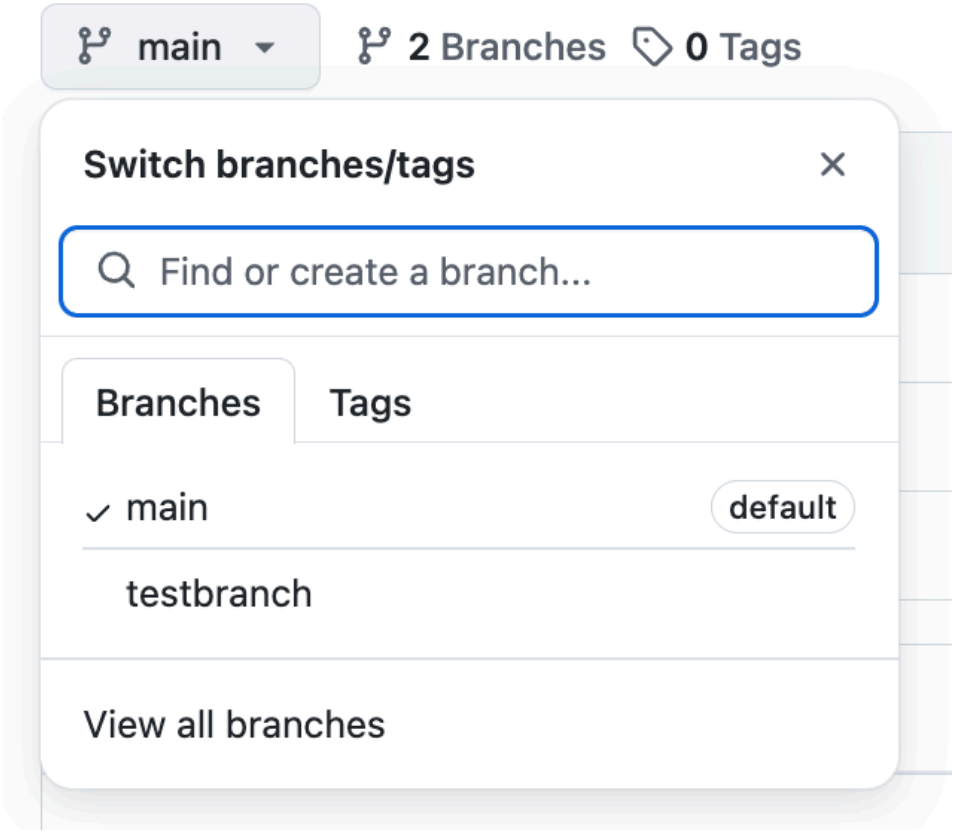


Figure 5: Check Branches in GitHub.

Amend Previous Commit

Assuming you have already committed your changes but forgot to include the data folder, you can amend your previous commit to add it.

> **When and Why to use —amend?**
>
> We use `--amend` when we want to **update the most recent commit**.
> For example, to include a file we forgot to add, or to fix a typo in the commit message. It lets us keep the commit history clean without creating a second, separate commit for something small.

1. Add the data folder in your project folder.
2. Use this command to amend previous commit, and push it (See Figure 6):

```
git add data/
git commit --amend -m "Your Messages"
git push -u origin (branch_name)
```

> **Tip**
>
> The `-u` flag tells Git to remember the connection between your local branch and the remote one. Next time, you can simply run `git push`, and Git will remember the connection.

```
192-168-1-2:Written_guide_demo akbarsyahpradana$ git add Data/
192-168-1-2:Written_guide_demo akbarsyahpradana$ git commit --amend -m "Amend previous commit to add the data folder"
[testbranch 198e2fe] Amend previous commit to add the data folder
 Date: Fri May 9 15:02:12 2025 +1000
 3 files changed, 6565 insertions(+), 1 deletion(-)
 create mode 100644 Data/example_data.csv
 create mode 100644 Data/ghg-emissions-by-sector.csv
192-168-1-2:Written_guide_demo akbarsyahpradana$ git push -u origin testbranch
```

Figure 6: Command for Amend.

3. Handling Merge Conflicts

If you make changes to the same line in both your new branch and the main branch, Git will detect a conflict and show a merge conflict when you try to combine them.

1. Assuming you have made changes to the same line in both the main branch and the new branch, you can now merge the new branch into main:

```
git switch main
git merge (branch_name)
```

2. Git will show the merge conflict (See Figure 7)

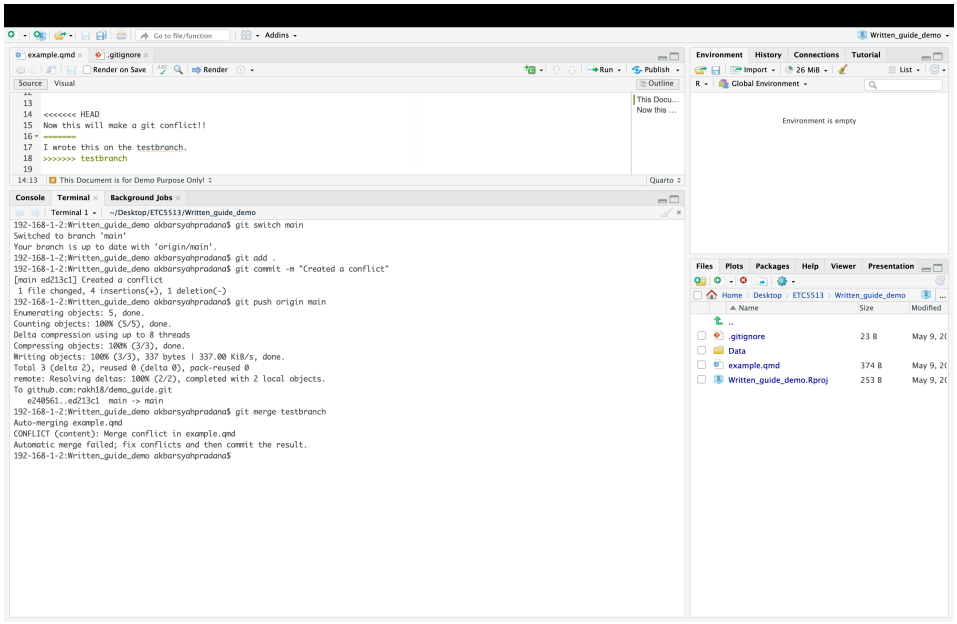


Figure 7: Merge Conflict.

! Important

```
<<<<<<< HEAD

This is the version from the main branch.

=======

This is the version from the other branch (e.g., testbranch).

>>>>>>> testbranch
```

This means:

- Everything between `<<<<<<< HEAD` and `=======` is from the **main** branch.
- Everything after `=======` until `>>>>>>> branch-name` is from the **branch you're merging in** (like `testbranch`).

3. To resolve the merge conflict, edit the file to keep which changes you want (or combine them), and make sure you remove the conflict markers («««< HEAD, ======= and »»»> branch_name).
4. Save the changes and push it with meaningful commit message.
5. Check the git status in GitHub (See Figure 8)

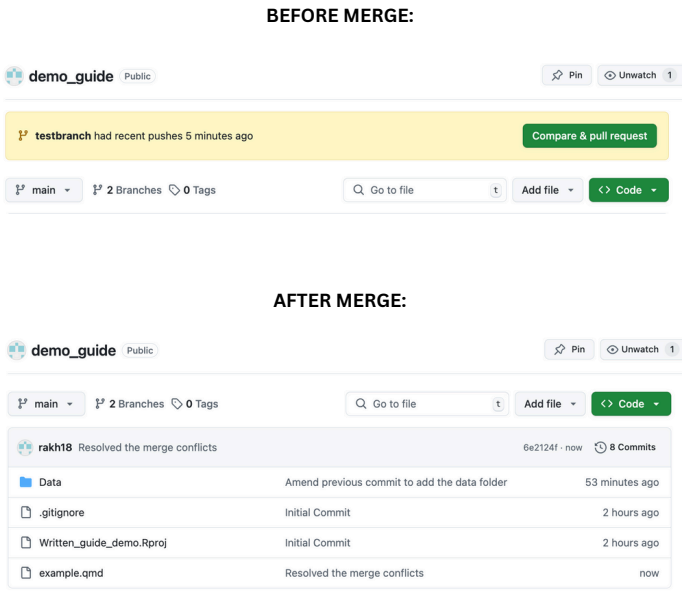


Figure 8: Git Status in GitHub.

4. Tags, Clean-up, and History

Now that your changes have been merged, this section covers how to tag your project version, clean up branches you no longer need, and view your commit history in a condensed format to keep your Git workflow organized and clear.

Create a Tag

Tags let you mark a milestone in your project's history, such as v1.0 after a successful merge. In this step, you will create a tag for your latest commit so you can easily refer to that version later.

1. Go to your terminal in R Studio.
2. Make sure you are on the main branch (or the branch with the commit you want to tag).
3. Use this command to tag the most recent commit as v1.0, and push the tag:

```
git tag -a v1.0 -m "Write a message to describe your milestone"
git push origin v1.0
```

> What if I want to tag other commit (not the most recent)?
>
> You can also tag a previous commit by first checking the history with `git log --oneline`, which will be covered in the Section .
> Then, use `git tag -a v0.9 <commit-code> -m "Describe the tag"` to tag that commit.
> You can replace `v0.9` with any tag name that suits your project. This is useful if you forgot to tag an earlier milestone.

4. Check the tag. You can check it from GitHub (See Figure 9). Or, you can also check it using:

```
git tag
```

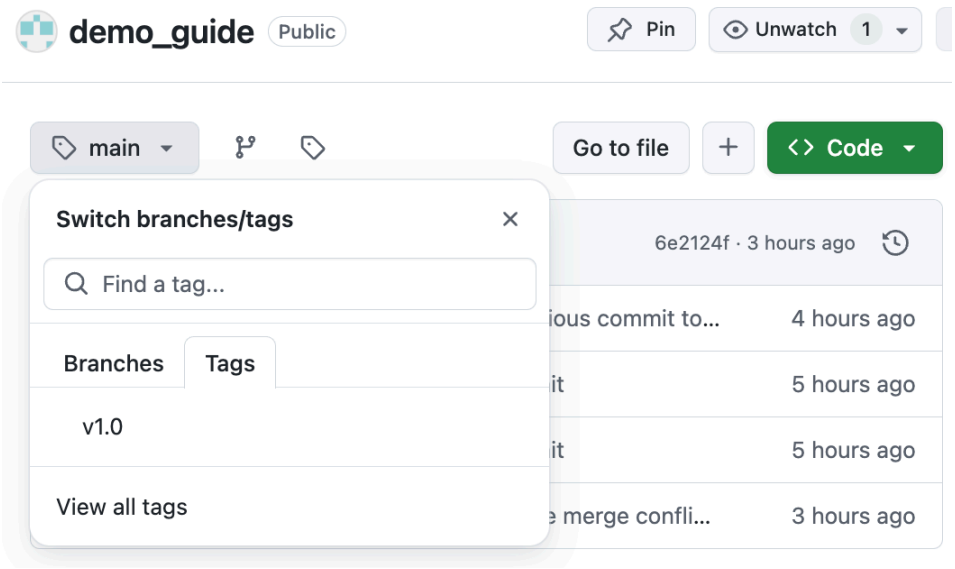


Figure 9: Check the Tag.

Deleting Branch

Deleting a branch after merging helps avoid clutter and confusion, especially when working on multiple features or versions.

1. Make sure you are on the **main** branch (or any branch other than the one you want to delete). Git won't let you delete the branch you're currently on.
2. Delete the branch locally:
   ```
   git branch -d branch_name
   ```
3. Delete the branch on GitHub (remote):
   ```
   git push origin --delete branch_name
   ```

Commit History

Sometimes, you just want a quick overview of what has been committed without all the details. You can use the following command to see a short version of your Git commit history:

```
git log --oneline
```

Then it will show you the output like Figure 10. This shows each commit as a single line, with the commit hash and its message. It's useful for finding commit IDs, reviewing your work at a glance, or tagging a past commit.

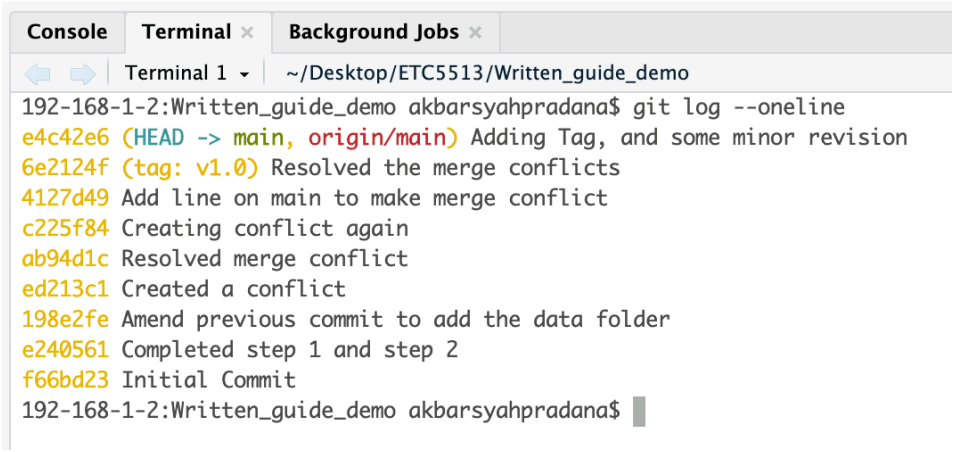


Figure 10: git log –oneline output.

> **Note**
>
> **HEAD** shows your current location in the repository. It points to the latest commit on the branch you are working on.

5. Creating a Plot and Undoing Commit

In this section, you will create a simple plot using ggplot2, commit the change, then learn how to undo the commit while keeping your changes safe.

Create a Section and Plot

1. In your `.qmd` file, you can add headings using `#` For example, to create a section for your visualisation:

```
## Visualisations (Headings 2)
### Headings 3
```

2. Create a code chunk by pressing `Command + Option + I` (Mac) or using the `Insert Chunk` button in the toolbar. See Figure 11 for a visual reference.

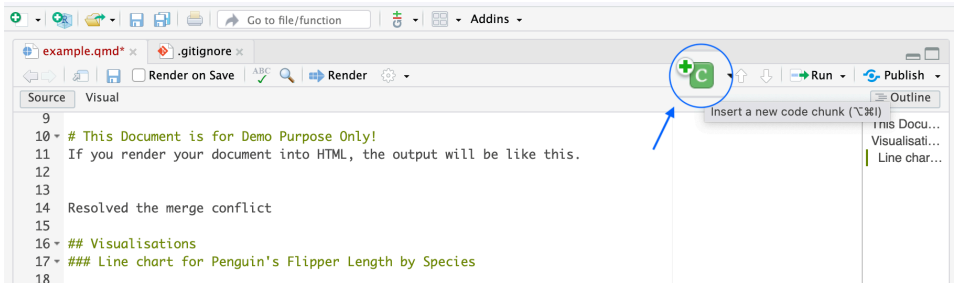


Figure 11: Create R Chunk.

3. You can make plots using the ggplot2 package. If you haven't used it before, install and load it with these in your R code chunk:

```
install.packages("ggplot2")
library(ggplot2)
```

4. Load your dataset using `read.csv()` or depends on the file's format.

5. Use ggplot2 to create a chart. In this example, I will use a line chart to visualise the relationship between two variables (See Figure 12).

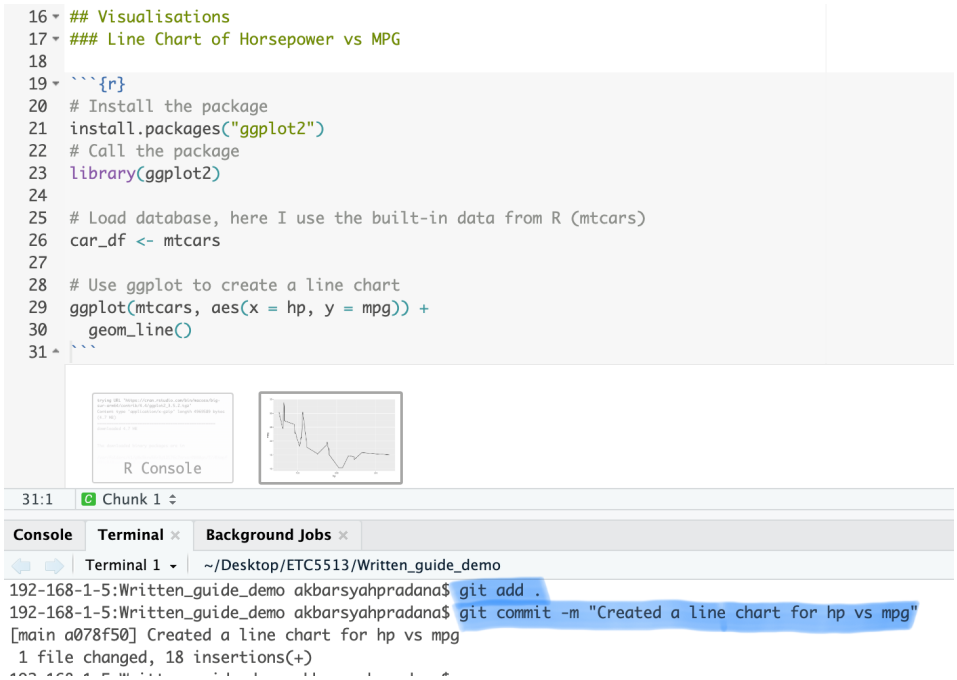


Figure 12: Create a line chart.

> **Learn More**
>
> For more information about `ggplot2`, click here.

6. Save and commit the changes as shown in Figure 12.

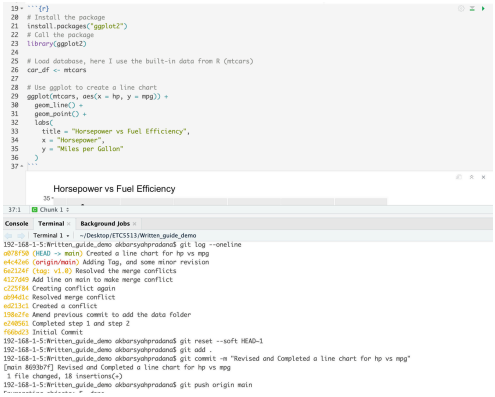
Undoing a Commit

Assume you have committed your plot, but then realise you made a mistake. For example, you forgot to include a title or other details (See Figure 13). You can undo the last commit without losing your changes by running:

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

- HEAD~1 refers to the commit before the latest one.
- This command removes the last commit but keeps your changes staged so you can fix them and recommit.

Use git reset --soft HEAD~1 :



Final commit history:

```
192-168-1-5:Written_guide_demo akbarsyahpradana$ git log --oneline
8693b7f (HEAD -> main, origin/main) Revised and Completed a line chart for hp vs mpg
e4c42e6 Adding Tag, and some minor revision
6e2124f (tag: v1.0) Resolved the merge conflicts
4127d49 Add line on main to make merge conflict
c225f84 Creating conflict again
ab94d1c Resolved merge conflict
ed213c1 Created a conflict
198e2fe Amend previous commit to add the data folder
e240561 Completed step 1 and step 2
f66bd23 Initial Commit
```

Figure 13: Undoing a commit.