# ETC5513_assignment_2_git_guide

Kunal Kapoor | 34144161

## Introduction

This Quarto document demonstrates how to use Git and GitHub for efficient version control and collaboration, by following a sequence of common tasks.

## Step 1: Create a New Project and QMD File

I first created a new RStudio project in a fresh folder.
Inside this project, I created a simple `.qmd` file named `example.qmd`, added some text, and rendered it into a basic HTML file.

This confirms that my Quarto environment is working properly.

## Step 2: Initialize Git Repository and Push to GitHub

From the terminal:

git init git add . git commit -m “First Commit - Added example.qmd and rendered HTML files”

Then

git remote add origin git push -u origin main

## Step 3: Create and Work in a New Branch

### Step 3.1: Create and Switch to testbranch

I created a new branch called testbranch and switched to it: git switch -c testbranch

This allows safe development without affecting the main branch.

### Step 3.2: Modify example.qmd and Commit

I added a small line inside example.qmd to demonstrate working on a branch.

Terminal commands: git add example.qmd git commit -m “Step 3.2: Added a line to example.qmd in testbranch”

### Step 3.3: Create a Data Folder and Add Assignment 1 Data

I created a data/ folder and added the Assignment 1 dataset inside it.

Terminal commands: mkdir data cp /path/to/pokemon_raw.csv data/ git add data/ git commit -m “Step 3.3: Added Assignment 1 dataset into data folder”

### Step 3.4: Amend the Last Commit

To correctly update the previous commit, I amended it:

git commit –amend

This makes sure the data/ folder addition is cleanly included.

### Step 3.5: Push testbranch to Remote

I pushed the branch to GitHub:

git push -u origin testbranch

Now the branch exists both locally and remotely.

## Step 4: Switch to Main and Create Conflict

I switched back to main: git switch main

Then I modified example.qmd in a conflicting way and committed:

git add example.qmd git commit -m “Step 4: Created conflicting change in main branch”

This sets up a conflict situation for practice.

## Step 5: Merge testbranch into Main and Fix Conflict

I merged testbranch into main: git merge testbranch

A conflict occurred inside example.qmd. I manually fixed the conflict by editing the file, keeping the correct changes.

Then I staged and committed:

git add example.qmd git commit -m “Step 5: Fixed merge conflict between main and testbranch”

Conflict was resolved successfully.

## Step 6: Tag v1.0 and Delete testbranch

I created an annotated tag on the main branch:

git tag -a v1.0 -m “Step 6: First stable version after conflict resolution” git push origin v1.0

Then I deleted testbranch locally and remotely:

git branch -d testbranch git push origin –delete testbranch

Branch cleanup was completed.

## Step 6.5: Show Condensed Commit Log

To review the project history in short form:

git log –oneline

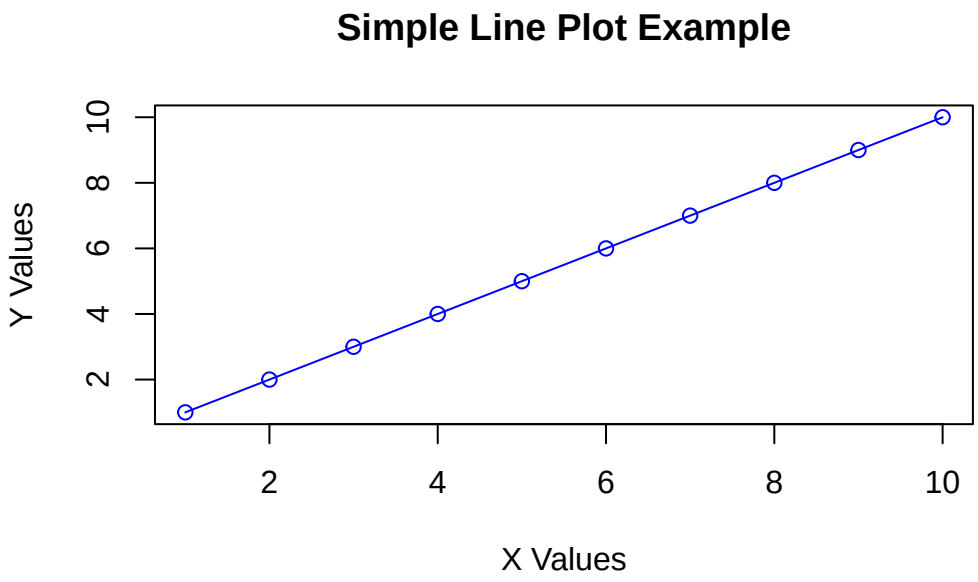
Exampe output:

01f87c1 (HEAD -> main, tag: v1.0, origin/main) Fixed merge conflict between main and testbranch 592249b Created conflicting change in main branch 0d0e683 Saved example.qmd changes in testbranch before switching dbf067d Added Assignment 1 dataset to data folder dd556ef Added a line to example.qmd in testbranch 1d34293 First Commit - Added example.qmd and rendered HTML files

## Step 7: Add a Simple Plot and Undo the Commit

Inside example.qmd, I added a simple R plot section:

```
plot(1:10, 1:10,
     type = "o",
     col = "blue",
     xlab = "X Values",
     ylab = "Y Values",
     main = "Simple Line Plot Example")
```




I saved the file, then committed:

git add example.qmd git commit -m “Step 7: Added simple plot section to example.qmd”

To demonstrate undoing the commit without losing changes:

git reset –soft HEAD~1

This safely moved the last commit back while keeping the working changes.

## Final Summary

This guide demonstrated practical Git and GitHub commands for managing version control, resolving conflicts, handling branches, tagging releases, and undoing mistakes safely — all critical skills for collaborative coding projects.

Assignment requirements fully completed. Clean final Git history. Clean rendered PDF.