

Introduction to git and GitHub

Jennifer Lin

2022-09-06

Contents

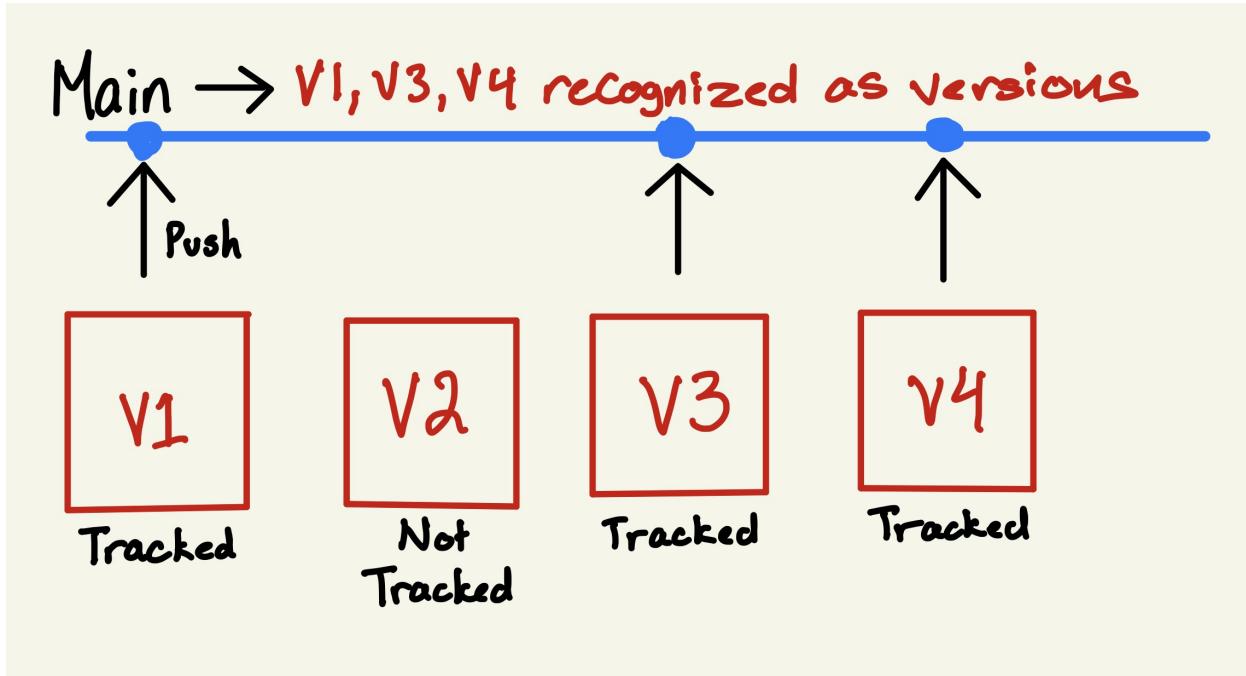
The Benefits of Version Tracking	1
Setting up git and GitHub	2
Creating a Repository	3
Adding the <code>.gitignore</code>	4
Clone Repository	6
Adding Files to the Repository	8
<code>git add</code> and <code>git status</code>	9
<code>git commit</code> and <code>git push</code>	10
Checking Commits	11
<code>git pull</code>	13
Branches	15
Create a Branch	16
Working with Branches on Local Computer	17
Pull Requests	18
Finding an Older Version	22
Git Point and Click	25
GitHub and R Studio	25
GitHub Desktop	27
Concluding Thoughts	33

The Benefits of Version Tracking

Have you ever written a document on your computer where you have one version of the file that you swear is the final version so you name it “Paper FINAL.docx” and save? Then, have you realized that you made a mistake and needed to change something and then you

save this new version as “Paper FINAL FINAL.docx” and so on? Alternatively, have you saved a file called “Paper.docx” to realize that you had a wording in a previous version that you now liked but you deleted it because it did not look good back then? Have you wished that you could get a time machine to go back to find that wording?

I apply no judgment if any of the above scenarios ring true but you and I both agree that either scenario is not ideal. What if I told you that the time machine that you wish you had in the second scenario exists and it can help you find a smarter solution to both problems at the same time? Introducing **Version Control**, specifically **git** and GitHub.



For starters, **git** is the computer software and GitHub is the online app. Both cooperate to help you manage your documents and store various versions of your files so you can retrieve them if needed. This is the essence of version control. It creates a backlog of all the changes that you have made and saved to control cloud. Saving a document on your own computer does not help save the version. You **must** add it to the repository in order for it to be tracked. But much more on all of this later. First, let's get the programs set up.

Setting up **git** and GitHub

To set up **git**, locate the terminal application on your computer. All computers, regardless of the operating system, has a terminal. From there, check to make sure that it is installed by following the [instructions here](#).

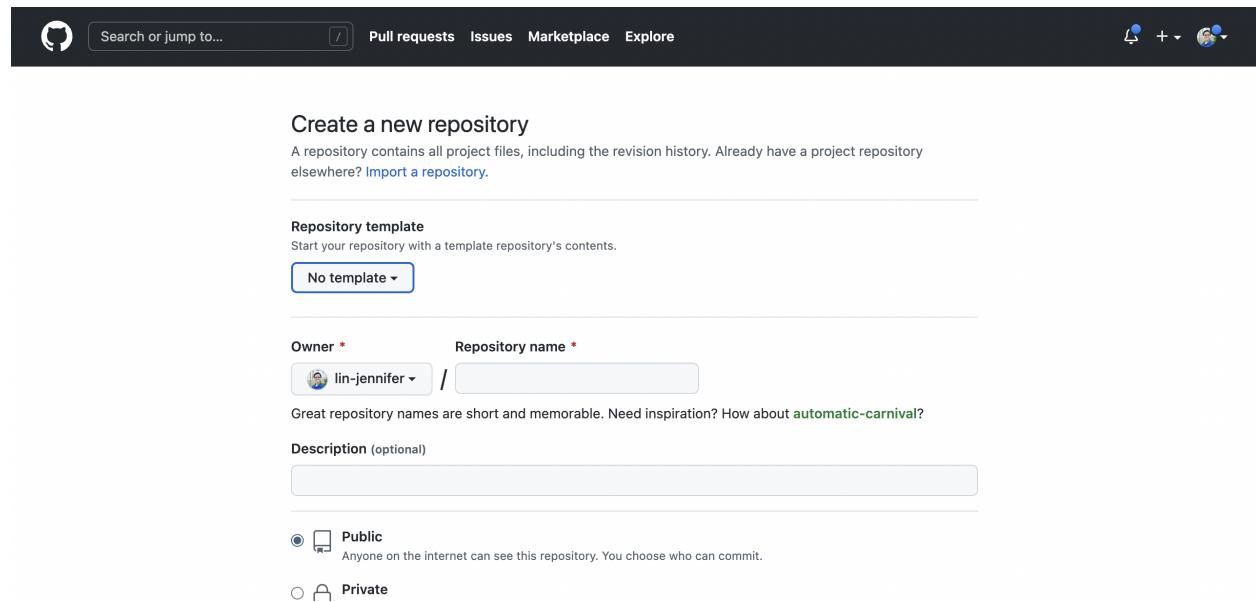
From here, you should set up a GitHub account on <https://github.com/>. Follow the instructions for signing up. When you are done, be sure to apply for the [Education Developer Pack](#) which gives you GitHub Pro so long as you are in academia as a teacher or as a student. GitHub Pro is usually \$4 a month but the Developer Pack gives you its benefits for free. This

includes unlimited private and public repositories and access to use GitHub Pages, which is a neat way to create your own academic website.¹

In addition, the Education Pack also provides neat perks for other programs. It provides you free access to things like DataCamp, NameCheap (for custom domain names, which can be useful when you are on the job market) and a bunch of other random computing tools that you may never need as a political scientist but are good to have in case you do need it. The main point is that the pack is easy to apply for, using your student ID and you should take advantage.

Creating a Repository

Now that you have your GitHub account set up, time to create a repository! A repository is a folder, usually for a project, that stores files. It is used similar to how you would use a folder on your own desktop. There is a command line method to create repositories, but why complicate our lives? Using your new GitHub account, locate the green **New** button on the upper left hand corner of your home screen. Click on it and you should see a form to create a new repository:

A screenshot of the GitHub 'Create a new repository' interface. At the top, there's a dark header bar with the GitHub logo, a search bar, and navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. On the right side of the header are user icons. Below the header, the main form has a title 'Create a new repository'. It includes a note about what a repository is and a link to import one. A 'Repository template' section allows selecting a template repository, with a 'No template' button. The 'Owner' field is populated with 'lin-jennifer'. The 'Repository name' field is empty and highlighted with a red border. Below these fields is a note about repository naming. The 'Description (optional)' field is empty. At the bottom, there are two radio buttons for 'Public' (selected) and 'Private' repository types, with a note explaining the difference between them.

We are going to fill this out. I am going to create a learning GitHub repository and make it public so that it can be a companion repository for anyone to practice GitHub with this tutorial. Even though I am doing this, you should almost **always** make your repositories private as Google sometimes indexes GitHub repositories for anyone to find. This is how you can find R packages in their developer version or cool datasets. It is a useful tool but perhaps not ideal if you are using GitHub to track your working papers and code. This is the benefit of GitHub Pro. The free plan will limit your number of private repositories but this is unlimited for Pro.

¹All GitHub accounts come with a custom domain for a website based in GitHub Pages. The domain is `username + github.io` so choose your username wisely!

Anyways, my learning repository will be `hello_git` and it will include the `README.md` file. This file is simply a default file that you should always include in your repositories. It is a good place to store important information about your project, especially once it is public. As you will soon see, the `README` is formatted nicely when you open the repository. It is the visitor's first look at your repository and it is also your first look at it too. So let's create this repository and see what we get.

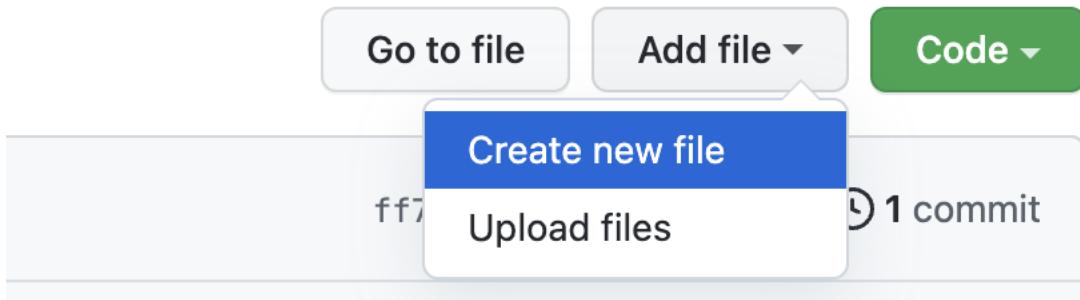
The screenshot shows a GitHub repository page for 'lin-jennifer/hello_git'. At the top, there are navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the header, the repository name 'lin-jennifer / hello_git' is shown as 'Public'. On the left, there are tabs for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings', with 'Code' being the active tab. In the center, there is a commit history showing a single commit from 'lin-jennifer' titled 'Initial commit' made 'now' with hash 'ff77c0e'. Below the commit, the 'README.md' file is displayed with the text 'hello_git'. On the right, there is an 'About' section with the message 'No description, website, or topics provided.' followed by statistics: 'Readme', '0 stars', '1 watching', and '0 forks'. Below that is a 'Releases' section stating 'No releases published' and 'Create a new release'. At the bottom right is a 'Packages' section which is currently empty.

Notice that the `README` contains the title of the repository as a default. You can always edit the `README` using the little pencil icon on the upper right hand corner of the document window.

Adding the `.gitignore`

When you created the repository, there was an option to ask if you wanted a `.gitignore` file. A `.gitignore` file is one that tells `git` to ignore some files when you are pushing to the cloud. This might be especially useful if you have large data files on your computer that should not be pushed or if you have auxiliary files (like `.Rhistory` or `.DS_Store`) that are only useful to your computer and no one else. It can be difficult to create your own `.gitignore` as there are file extensions that are out there that apply to the software that you plan on using that you perhaps never learned about. Fortunately, there is an app for that.

Before we go there, though, let's create a new file on GitHub.



In the text box for a new file, write `.gitignore`. Immediately, you will see options on the right hand side for templates. Do not use those. The default only allows you to add one program, so you pick between R, L^AT_EX, Stata, Mac and so on, but you cannot have them all.

hello_git / `.gitignore` in main

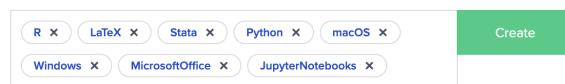
A screenshot of a GitHub file editor. The title bar shows 'hello_git / .gitignore' and 'in main'. Below the title bar are two buttons: '<> Edit new file' and 'Preview'. The main area contains the file content: '1'. There is a vertical scrollbar on the left side of the content area.

Enter gitignore.io. This is a web app that automatically creates `.gitignore` files for you, and can include all of the programs that you are planning to use. Here are all the software and programs that I like to include. **Remember**, your computer is technically a program too and it has auxiliary files that you don't want to push either!



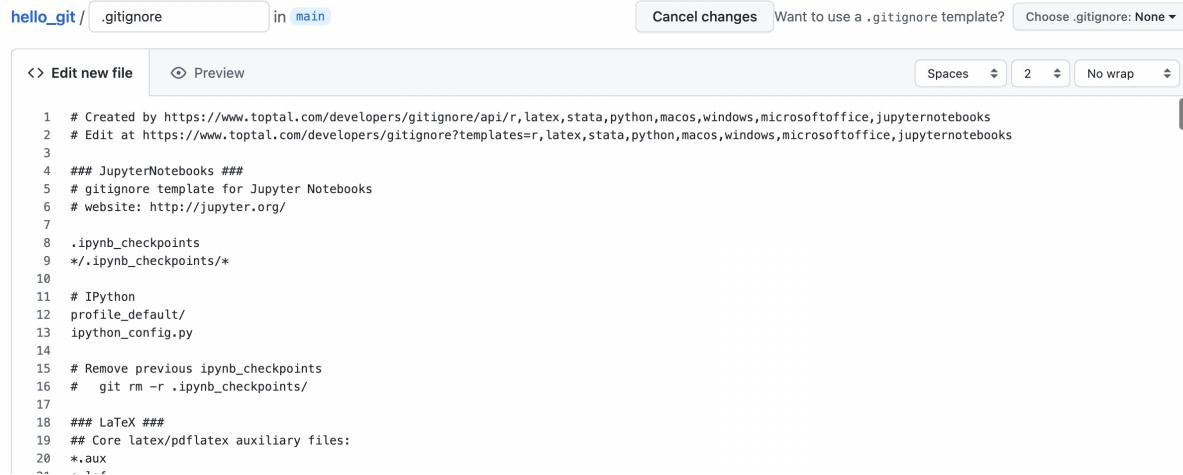
gitignore.io

Create useful `.gitignore` files for your project



[Source Code](#) | [Command Line Docs](#)

Once you click “Create”, you will get a whole list of file extensions. Select all on the screen and copy and paste it into the new file on your GitHub screen.

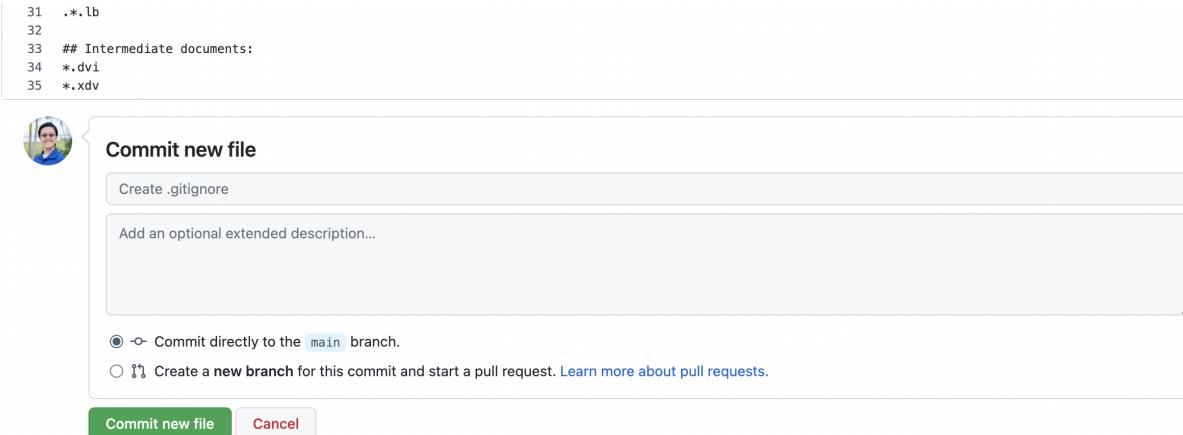


The screenshot shows the GitHub 'Edit new file' interface. The file path is 'hello_git/.gitignore' and the branch is 'main'. The code editor contains a .gitignore template for Jupyter Notebooks, listing various file types and directories to ignore. The interface includes standard text editor controls like 'Edit new file', 'Preview', and 'Cancel changes'.

```

1 # Created by https://www.toptal.com/developers/gitignore/api/r,latex,stata,python,macos,windows,microsoftoffice,jupyternotebooks
2 # Edit at https://www.toptal.com/developers/gitignore?templates=r,latex,stata,python,macos,windows,microsoftoffice,jupyternotebooks
3
4 ### JupyterNotebooks ###
5 # gitignore template for Jupyter Notebooks
6 # website: http://jupyter.org/
7
8 .ipynb_checkpoints
9 */.ipynb_checkpoints/*
10
11 # IPython
12 profile_default/
13 ipython_config.py
14
15 # Remove previous ipynb_checkpoints
16 # git rm -r .ipynb_checkpoints/
17
18 ### LaTeX ###
19 ## Core latex/pdflatex auxiliary files:
20 *.aux
~~~
```

Now that this very long file of mostly extensions are copied into the editor on GitHub, you can scroll all the way to the bottom where it says “Commit New File”. Here, you can write a description of what you did, but the default text of “Create .gitignore” will do. We will talk more about commit messages later. Click “Create New File” and you are set!



Clone Repository

Now, lets get the repository off the cloud and onto the computer. Here, we will use the terminal app that you located earlier. Going forward, I will make my demonstrations on a Mac, but Windows and Linux operates quite similarly.

Looking on the homepage on the repository on GitHub, there is a green “Code” drop down menu. Click on it. You will see three options, one for a URL, one for opening on GitHub Desktop (more later) and one for downloading file as a Zip file. NEVER download it as a Zip file because this option does not come with the invisible .git file that git needs to track versions. Instead, we want the URL. Copy the entire URL.

Let's open up terminal and find out where we are. Use `ls` to see what folders are in the folder you are currently in. Usually, the default folder is your User folder.

```
Last login: Sun Sep  4 13:33:20 on ttys001
(base) jenniferlin@Jennifers-iMac ~ % ls
Applications   Downloads   Movies       Public
Desktop        Google Drive  Music       Zotero
Documents      Library    Pictures
```

We can then use the command `cd`, or Change Directory, to locate a folder on one of these sub-folders on the computer to place the repository. I am going to put it on my desktop.

```
Last login: Sun Sep  4 13:33:20 on ttys001
(base) jenniferlin@Jennifers-iMac ~ % ls
Applications   Downloads   Movies       Public
Desktop        Google Drive  Music       Zotero
Documents      Library    Pictures
(base) jenniferlin@Jennifers-iMac ~ % cd Desktop/
```

Once I am on my Desktop, I can use the function `git clone + "URL"` to get the repository to my Desktop. This is where the URL that you copied earlier comes in handy.

```
[base] jenniferlin@Jennifers-iMac Desktop % git clone "https://github.com/lin-je
nnifer/hello_git.git"
Cloning into 'hello_git'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), 5.24 KiB | 5.24 MiB/s, done.
(base) jenniferlin@Jennifers-iMac Desktop %
```

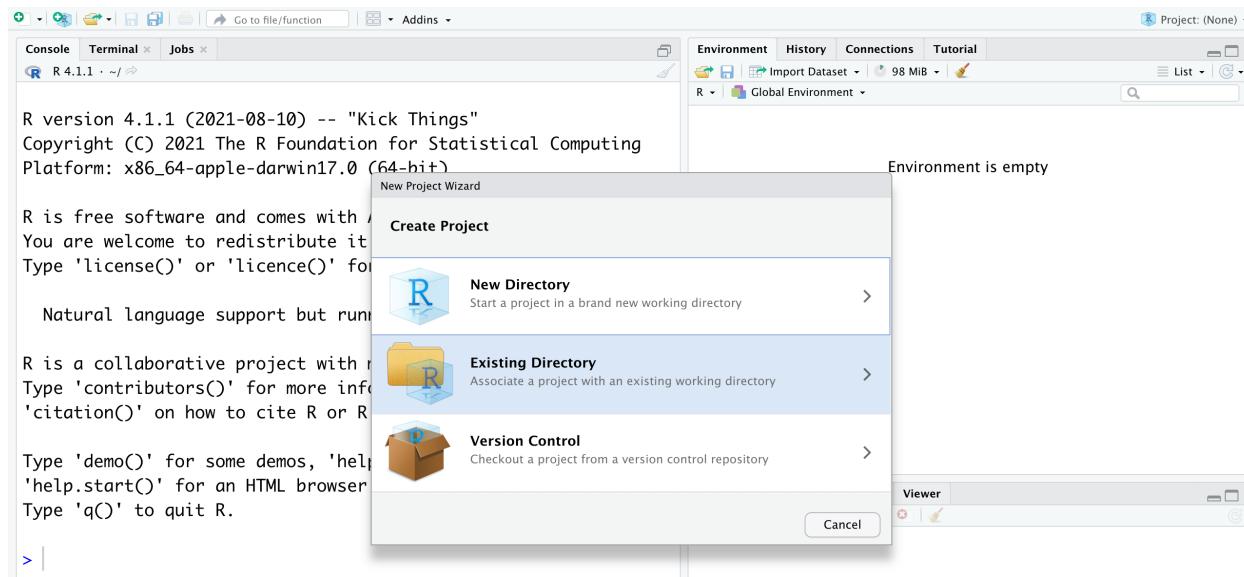
Notice that after we clone the repository, we are still on the Desktop in the terminal, but the folder is already in the destination. We need to navigate to the folder, using the handy `cd` command.

```
[base] jenniferlin@Jennifers-iMac Desktop % cd hello_git
(base) jenniferlin@Jennifers-iMac hello_git %
```

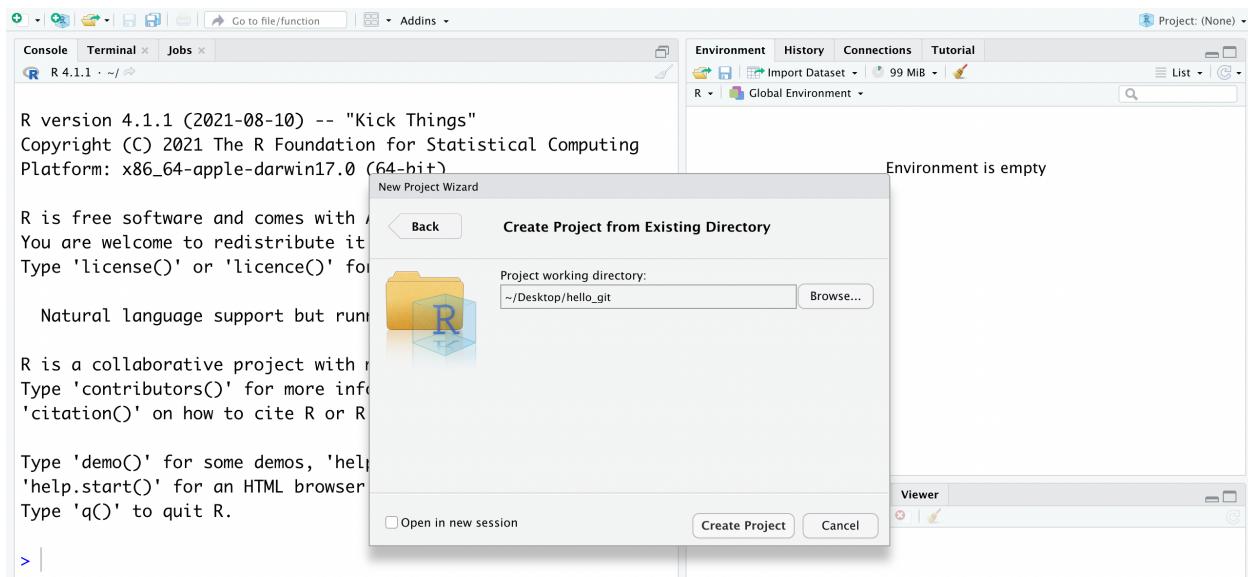
Now, open the repository folder on the Desktop, or wherever you cloned it. Notice how the only file here is the `README`. There was a `.gitignore` but it is not in the folder because it is one of those invisible `git` files. To prove that it is there, let's create a file that contains one of the ignore extensions and add it to the repository. We are going to use R as an example.

Adding Files to the Repository

We can open a new R session and go to File > New Project. We then select the option that allows us to create a new project from an existing repository.



From there, add the file path, using the Browse button, for the repository. Then hit “Create Project”.



Within this project, I am going to add an R script called `Hello.R`. The contents of this script are irrelevant. Save an R script, R markdown file, or really anything to your repository. In the Files section in the lower right hand corner of R Studio, you will see the contents of your repository.

	Name	Size	Modified
<input type="checkbox"/>	..		
<input type="checkbox"/>	.gitignore	8.9 KB	Sep 4, 2022, 1:38 PM
<input type="checkbox"/>	.Rhistory	45 B	Sep 4, 2022, 2:03 PM
<input type="checkbox"/>	hello_git.Rproj	205 B	Sep 4, 2022, 2:03 PM
<input type="checkbox"/>	Hello.R	396 B	Sep 4, 2022, 2:03 PM
<input type="checkbox"/>	README.md	11 B	Sep 4, 2022, 1:38 PM

Notice that the `.gitignore` is there and so is this things called `.Rhistory`. The `.Rhistory` file is something for R to track what commands you have ran. You do not need this when you share the repository to your co-authors or to the world. The `.gitignore` file is programmed to recognize this file and omit it when adding files to the repository.

git add and git status

And speaking of adding files, let's add the R project and the script to the repository. Go back out to the terminal and type `git status` into the command line. At some point, you will be prompted to add a username and password so that you can actually add the files. The system

should prompt you to do that. Since my system is already set up, I cannot demonstrate that here.

```
(base) jenniferlin@Jennifers-iMac Desktop % cd hello_git
(base) jenniferlin@Jennifers-iMac hello_git % ls
Hello.R          README.md      hello_git.Rproj
(base) jenniferlin@Jennifers-iMac hello_git % git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Hello.R
    hello_git.Rproj

nothing added to commit but untracked files present (use "git add" to track)
(base) jenniferlin@Jennifers-iMac hello_git %
```

Notice that there are two untracked files, the R project and Hello.R. Let's get git to start tracking those.

Using `git add`, we can add all the files that we want to track. `git add + "Filename"` adds the particular file specified. Let's add the R project.

git commit and git push

We can then use `git commit -m + "MESSAGE"` to insert a commit message about what we did. This captures an image of the file and git recognizes it as a new version. The message is a note to yourself about what you changed so you can go back and see the previous versions.

```
On branch main
Your branch is up to date with 'origin/main'.
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Hello.R
    hello_git.Rproj

nothing added to commit but untracked files present (use "git add" to track)
(base) jenniferlin@Jennifers-iMac hello_git % git add "hello_git.Rproj"
(base) jenniferlin@Jennifers-iMac hello_git % git commit -m "Initiate R Project
for Repo"
[main a56f5f9] Initiate R Project for Repo
 1 file changed, 13 insertions(+)
  create mode 100644 hello_git.Rproj
(base) jenniferlin@Jennifers-iMac hello_git %
```

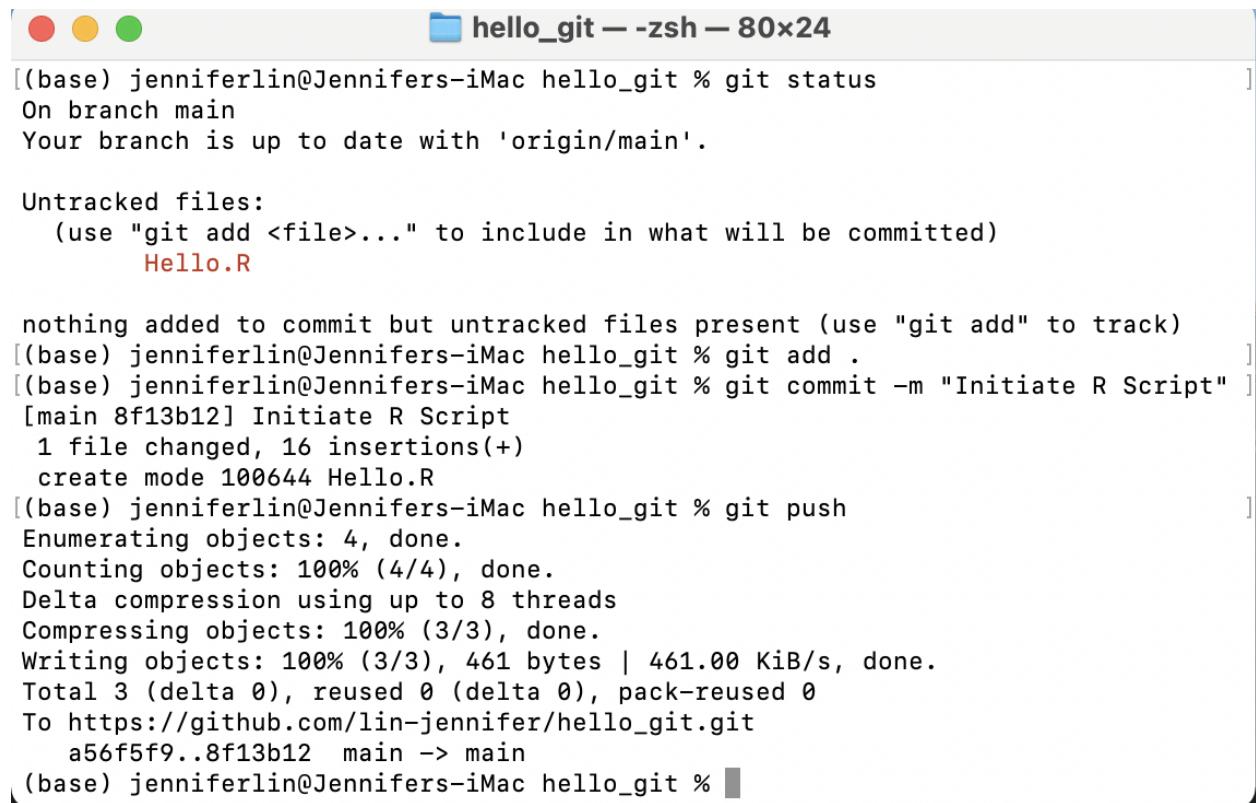
To add it to the cloud, we can use `git push`.

```

(base) jenniferlin@Jennifers-iMac hello_git % git commit -m "Initiate R Project ] for Repo"
[main a56f5f9] Initiate R Project for Repo
 1 file changed, 13 insertions(+)
  create mode 100644 hello_git.Rproj
(base) jenniferlin@Jennifers-iMac hello_git % git push
] Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 475 bytes | 475.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/lin-jennifer/hello_git.git
  9c05b11..a56f5f9  main -> main
(base) jenniferlin@Jennifers-iMac hello_git %

```

Are we done? Let's use `git status` to find out. It appears that we still have the `Hello.R` file to push. Let's add that with `git add`. But this time, instead of using `git add + "Filename"`, let's do `git add ..`. This is a shorthand to add all the files that need a commit message. This is assigning one commit message to all of the files added, so be careful when using this shorthand!



```

[(base) jenniferlin@Jennifers-iMac hello_git % git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Hello.R

nothing added to commit but untracked files present (use "git add" to track)
[(base) jenniferlin@Jennifers-iMac hello_git % git add .
[(base) jenniferlin@Jennifers-iMac hello_git % git commit -m "Initiate R Script"
[main 8f13b12] Initiate R Script
 1 file changed, 16 insertions(+)
  create mode 100644 Hello.R
[(base) jenniferlin@Jennifers-iMac hello_git % git push
] Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 461 bytes | 461.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/lin-jennifer/hello_git.git
  a56f5f9..8f13b12  main -> main
(base) jenniferlin@Jennifers-iMac hello_git %

```

Checking Commits

Now, let's go back to the GitHub web app and see our repository. Notice how all of the files that we added, along with our commit messages, are in the repository.

The screenshot shows a GitHub repository page for 'lin-jennifer/hello_git'. The repository is public. The main branch is 'main' with 1 branch and 0 tags. There are 4 commits by 'lin-jennifer' listed:

File	Commit Message	Time Ago
.gitignore	Create .gitignore	1 hour ago
Hello.R	Initiate R Script	1 minute ago
README.md	Initial commit	3 hours ago
hello_git.Rproj	Initiate R Project for Repo	6 minutes ago

Below the commits, there is a file named 'README.md' containing the text 'hello_git'.

The one file that is missing is `.Rhistory` and that because it was ignored in the push process thanks to the `.gitignore`.

I can also click on the text that says “N Commits” under the green “Code” drop down to see my commits.

The screenshot shows the same GitHub repository page, but the 'Code' dropdown is expanded to show the commit history for September 4, 2022. The commits are:

- Initiate R Script (commit `8f13b12`)
- Initiate R Project for Repo (commit `a56f5f9`)
- Create .gitignore (commit `9c05b11`)
- Initial commit (commit `ff77c0e`)

This does not look too bad! Let's click into the first one and see what happens.

The screenshot shows a GitHub repository page for 'lin-jennifer/hello_git'. The 'Code' tab is selected. A commit message 'Initiate R Script' is shown, along with a timestamp 'lin-jennifer committed 21 minutes ago'. The commit details show 1 parent commit 'a56f5f9' and a commit hash '8f13b12e56999b037eb946e0aabedf45c773d8f3'. Below this, it says 'Showing 1 changed file with 16 additions and 0 deletions.' The file 'Hello.R' is expanded, showing the following code:

```

16 Hello.R
...
@@ -0,0 +1,16 @@
+ # ****
+ # OVERVIEW #####
+ # ****
+ # Hello.R
+ # Author:
+ # Created On: YYYY MM DD
+
+ # ****
+ # DAYA #####
+ # ****
+ hello <- c("hello", "hi", "hola", "bonjour")

```

We see that GitHub is telling us what was added in green, and if applicable, what is deleted is in red.

git pull

`git` is a push and pull process, especially when collaborating. Let me pretend to be a different user and make edits to the `README` using the tiny pencil on the upper right on the home screen. I will add some notes about what I did and overall, make it look nicer. Before I do, notice the current `README`. It is just the name of the repository in big letters.

When you use the pencil tool to update the `README`, it will ask you to commit changes, like what happens when we did the `.gitignore`. Simply commit and add. Now, look at the new `README` on the cloud.

The screenshot shows a GitHub repository page for 'lin-jennifer/hello_git'. At the top, there are navigation links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the header, it shows 'main' branch, 1 branch, 0 tags, and buttons for Go to file, Add file, and Code. A list of commits is shown, with the most recent being 'lin-jennifer Update README.md' (df4dee1, 2 minutes ago). Below the commits is the content of the README.md file, which contains the text 'An Introduction to GitHub'.

Let's look at the local copy of the README.md file on the computer.



So we just made an edit on the web app, why did it not transfer to the desktop version? To get it to transfer, we need `git pull` in the terminal. **You should ALWAYS pull before you push!** This is especially true if you know that you have made a change on the cloud, on a different computer, or if a co-author has made changes.

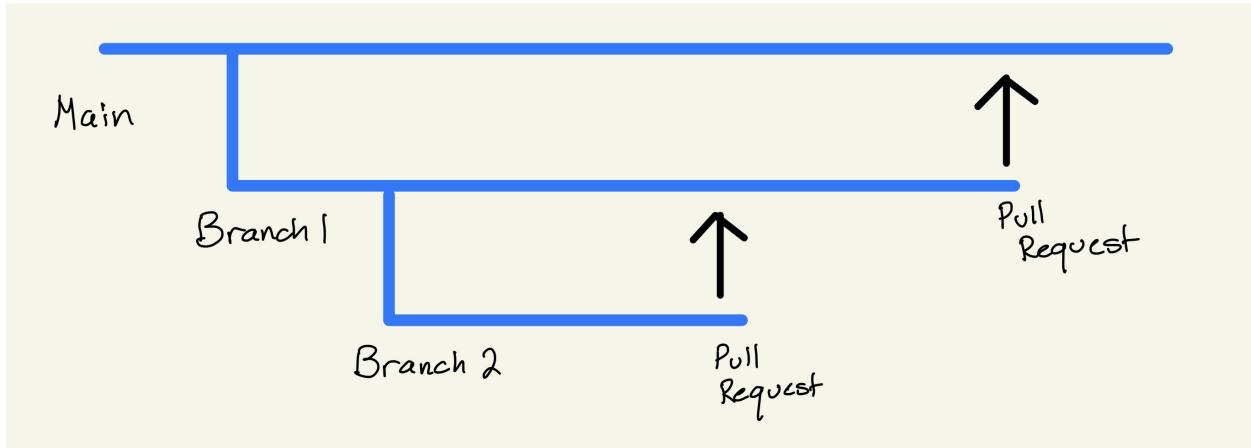
To fetch the changes, simply type `git pull` into the terminal.

```
(base) jenniferlin@Jennifers-iMac hello_git % git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 728 bytes | 242.00 KiB/s, done.
From https://github.com/lin-jennifer/hello_git
 8f13b12..df4dee1  main      -> origin/main
Updating 8f13b12..df4dee1
Fast-forward
 README.md | 4 +---+
 1 file changed, 3 insertions(+), 1 deletion(-)
(base) jenniferlin@Jennifers-iMac hello_git %
```

As you can see, the results in the terminal will let you know what files changed and how much was changed.

Branches

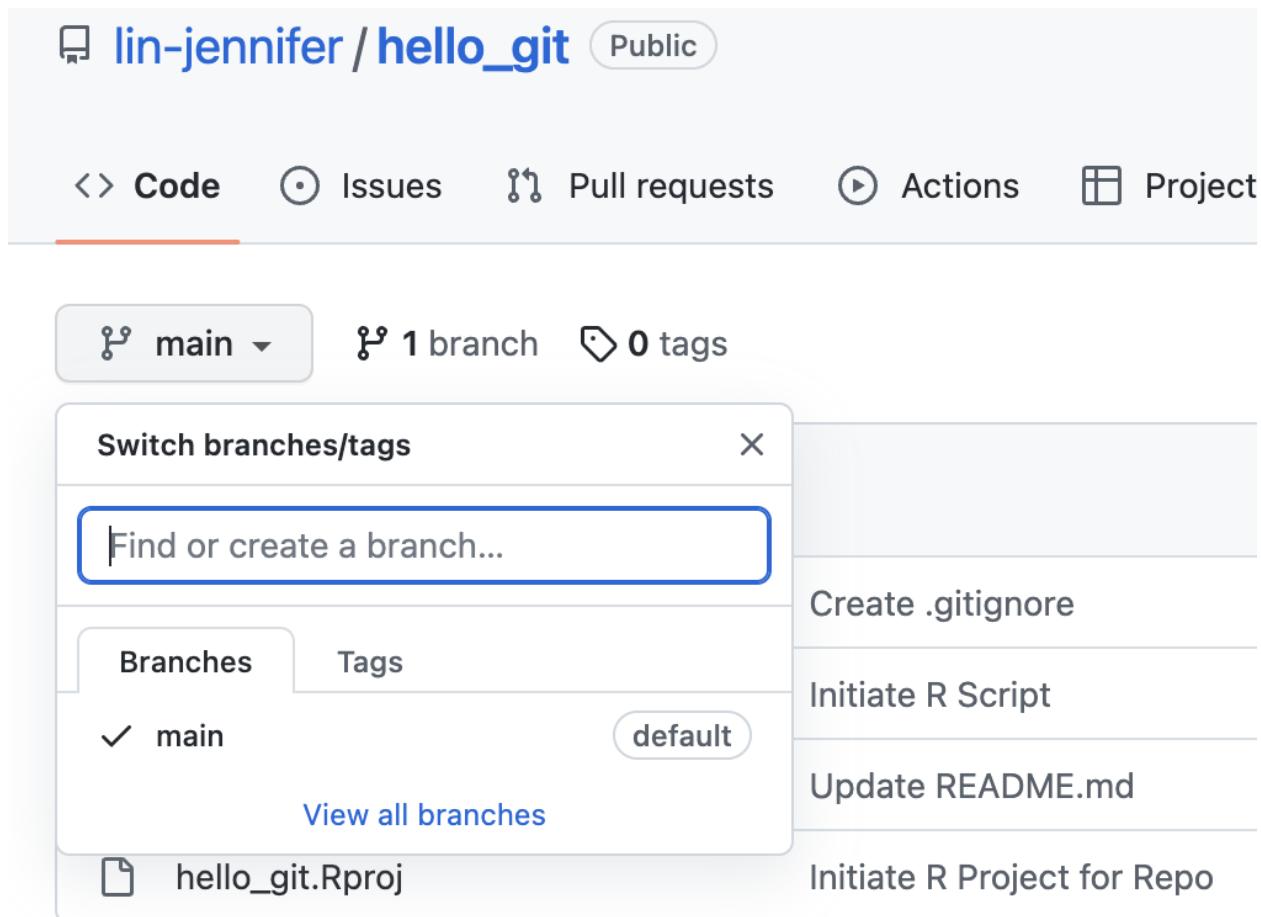
A useful feature of GitHub is in branches. You can work on various parts of a project on the side without showing it to your co-authors on the main branch and merge it once you are ready. To understand branches, you need to first understand how git works in theory.



When you create a repository, you are working in the `main` branch.² You can create branches off of the main branch to write drafts, troubleshoot code, and more, before making it official on the main branch. When you are ready to bring work from a branch to the main, you can use a pull request.

So let's take a look at the branches that we have for the `hello_git` repository

²The `main` branch was once called `master` so older git tutorials and documents might have `master`. But both reference the same thing.



Create a Branch

Suppose that I want to work on the draft of a paper on a different branch because it is just me putting my ideas down. It is super preliminary and my co-author, much less the public, should see it. Let's create a branch called `paper` by typing that into the dialog box. Once we hit "Create", the app jumps to the branch and we can see that because the dropdown that we just used is now named `paper` rather than `main`.

This branch is up to date with main.

lin-jennifer Update README.md

.gitignore	Create .gitignore
Hello.R	Initiate R Script
README.md	Update README.md
hello_git.Rproj	Initiate R Project for Repo

Beneath the menu of branches, we see a note that says “This branch is up to date with main.” Excellent. Now, lets go back out to the terminal and open up this branch on the desktop version of this folder.

Working with Branches on Local Computer

First, we need to run a `git pull` to pull the information about the new branch. Then, we use the command `git checkout + branchname` to access the branch.

```
(base) jenniferlin@Jennifers-iMac hello_git % git pull
From https://github.com/lin-jennifer/hello_git
 * [new branch]      paper      -> origin/paper
Already up to date.
(base) jenniferlin@Jennifers-iMac hello_git % git checkout paper
Branch 'paper' set up to track remote branch 'paper' from 'origin'.
Switched to a new branch 'paper'
(base) jenniferlin@Jennifers-iMac hello_git %
```

Go ahead and open up the folder on your computer, from where you first stored it. You will

notice that things look the same. Let's change that. Let's make a few changes, like create a folder for the paper and insert a L^AT_EX document using this [Basic Template from Overleaf](#). Once you have a folder and a document, let's add and commit it to the `paper` branch using the same steps as above.

```
[base] jenniferlin@Jennifers-iMac hello_git % git checkout paper
Switched to branch 'paper'
Your branch is up to date with 'origin/paper'.
[base] jenniferlin@Jennifers-iMac hello_git % git status
On branch paper
Your branch is up to date with 'origin/paper'.


Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Paper/


nothing added to commit but untracked files present (use "git add" to track)
[base] jenniferlin@Jennifers-iMac hello_git % git add .
[base] jenniferlin@Jennifers-iMac hello_git % git commit -m "Add a paper template"
[paper 9bb91c0] Add a paper template
  1 file changed, 39 insertions(+)
   create mode 100644 Paper/Paper.tex
[base] jenniferlin@Jennifers-iMac hello_git % git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 592 bytes | 592.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/lin-jennifer/hello_git.git
  df4dee1..9bb91c0  paper -> paper
[base] jenniferlin@Jennifers-iMac hello_git %
```

Pull Requests

When we go back to the GitHub page, we get a message telling us that the `paper` branch is one commit ahead of `main`. That is true, because we just made a commit. We then get this option for “Compare & Pull Request”.

Suppose that you are happy with the `paper` and you want to bring it to the `main` branch so that your co-authors can see the brilliant work that you have been doing on the side. To do this, we will click on that green “Compare & Pull Request” button. When we do, we get the following:

The screenshot shows a GitHub repository page for 'lin-jennifer/hello_git'. The 'Code' tab is selected. A prominent message at the top says 'Able to merge. These branches can be automatically merged.' Below this, there's a 'Compare' dropdown set to 'paper' and a 'base' dropdown set to 'main'. A large text area titled 'Add a paper template' contains the commit message: 'Add a paper template'. It includes tabs for 'Write' and 'Preview' and a toolbar with various rich text icons. A smaller text area below it says 'Leave a comment'.

It tells us that we are comparing the `paper` branch to the `main` branch and that we can merge. But there is nothing that shows what changed. Instead, we see a dialog box with the commit message that I used to make the commit of the `paper` folder to the branch. For now, this is all fine and good. Let's scroll down.

Further down on the page, we can see the version tracking at work. We see that one file has been changed and the number of commits. We also can see the changes in the file itself.

The screenshot shows the commit history for the `paper` branch. It displays one commit made by 'lin-jennifer' on Sep 4, 2022, with the message 'Add a paper template'. The commit resulted in 1 file changed and 1 contributor. Below the commit, a diff view shows the changes made to the file `Paper.tex`. The diff shows 39 additions and 0 deletions. The code changes are as follows:

```
@@ -0,0 +1,39 @@
1 + \documentclass[11pt]{article}
2 +
3 + \usepackage{sectsty}
4 + \usepackage{graphicx}
5 +
6 + % Margins
7 + \topmargin=-0.45in
8 + \evensidemargin=0in
9 + \oddsidemargin=0in
10 + \textwidth=6.5in
11 + \textheight=9.0in
12 + \headsep=0.25in
13 +
```

Great. This is essentially what we want. Let's go back up to the comment box on top. It provides you a space to leave a comment, formatted in markdown, so I will. I type in markdown and here is what it looks like in Preview.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

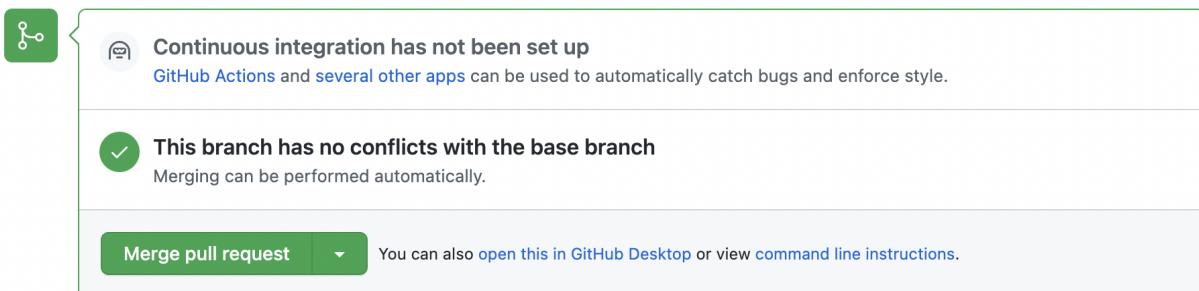
The screenshot shows the GitHub interface for creating a pull request. At the top, there are dropdown menus for 'base: main' and 'compare: paper'. A green checkmark indicates 'Able to merge. These branches can be automatically merged.' Below this, a user profile picture is shown next to a text input field labeled 'Add a paper template'. There are 'Write' and 'Preview' tabs. The main content area is titled 'Draft Paper Template' and contains the text 'Explored ways to integrate ideas to *LATEX* formatting. Ready to share!'. At the bottom right is a green 'Create pull request' button.

Ok. Now we are ready to click the “Create Pull Request” button. Once we do, you can see that there is now an open pull request and the comments that were inserted from the draft stage are now on top.

The screenshot shows the GitHub pull request page for the repository 'lin-jennifer/hello_git'. The repository is public. The navigation bar includes links for Code, Issues, Pull requests (1), Actions, Projects, Wiki, Security, Insights, and Settings. The 'Pull requests' tab is active. The main content shows a pull request titled 'Add a paper template #1'. It has an 'Open' status and a comment from 'lin-jennifer' stating 'lin-jennifer wants to merge 1 commit into main from paper'. Below the pull request are sections for Conversation (0), Commits (1), Checks (0), and Files changed (1). The commit details show a comment from 'lin-jennifer' 30 seconds ago: 'Draft Paper Template'. The commit message is 'Explored ways to integrate ideas to *LATEX* formatting. Ready to share!'. The commit hash is 9bb91c0. A note at the bottom says 'Add more commits by pushing to the paper branch on lin-jennifer/hello_git.'

When we scroll down a little, we can see that GitHub has automatically told us whether we are good to go with merging the `paper` branch and the `main` branch.

Add more commits by pushing to the `paper` branch on [lin-jennifer/hello_git](#).



A screenshot of a GitHub pull request merge interface. It shows two sections: one indicating 'Continuous integration has not been set up' (with a note about GitHub Actions and other apps) and another stating 'This branch has no conflicts with the base branch' (with a note about automatic merging). A green 'Merge pull request' button is at the bottom left, and a note says 'You can also open this in GitHub Desktop or view command line instructions.'

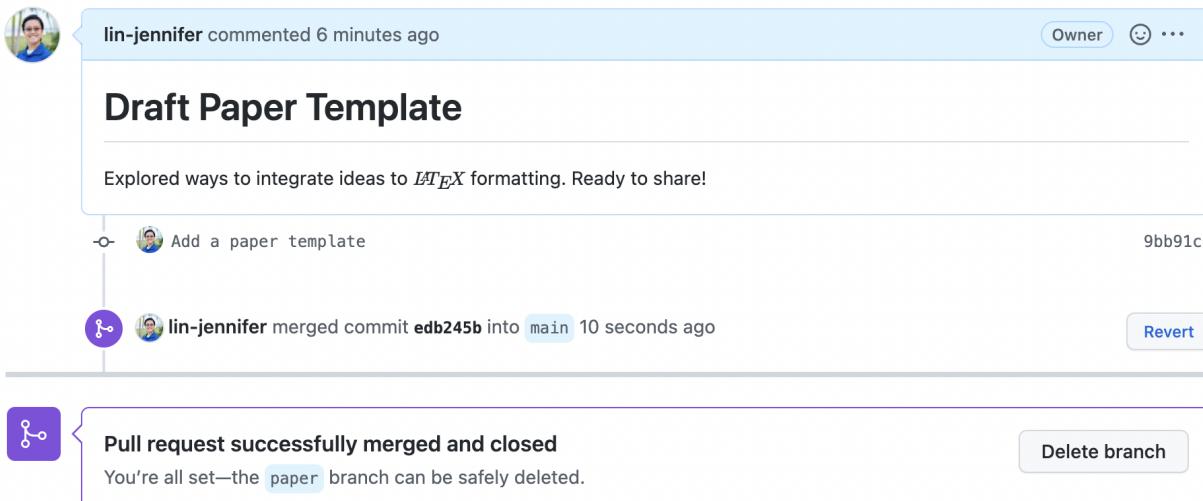
Since we have been given the green light, let's go ahead and do the merge. It will then prompt us to make sure that we are making the right choice. We are! So click "Confirm merge".



A screenshot of a GitHub merge confirmation dialog. It shows the title 'Merge pull request #1 from lin-jennifer/paper', a placeholder for 'Add a paper template', and a note that the commit will be authored by 42876762+lin-jennifer@users.noreply.github.com. At the bottom are 'Confirm merge' and 'Cancel' buttons.

Now that the branch has been merged successfully, we are given the option to delete the branch. I will leave that decision to you. You can save or delete, its a personal choice.

But notice that once you confirm the merge, you just made a commit to the `main` branch. Remember, this means that the `paper` folder is in the `main` branch online but it is not on your `main` branch on your computer.



A screenshot of a GitHub repository page. It shows a comment from 'lin-jennifer' that reads 'Draft Paper Template'. Below it is a commit message: 'Explored ways to integrate ideas to *LATEX* formatting. Ready to share!' with a link to 'Add a paper template'. Another commit message follows: 'lin-jennifer merged commit `edb245b` into `main` 10 seconds ago'. At the bottom, a purple icon indicates the pull request was merged, with a message: 'Pull request successfully merged and closed' and 'You're all set—the `paper` branch can be safely deleted.' A 'Delete branch' button is also visible.

Let's go out to the terminal to fix this. Using `git checkout main` and `git pull` we can get the entire repository up to date on the local version.

```
(base) jenniferlin@Jennifers-iMac hello_git % git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
(base) jenniferlin@Jennifers-iMac hello_git % git pull
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 634 bytes | 634.00 KiB/s, done.
From https://github.com/lin-jennifer/hello_git
  df4dee1..edb245b  main      -> origin/main
Updating df4dee1..edb245b
Fast-forward
  Paper/Paper.tex | 39 ++++++++++++++++++++++++++++++
  1 file changed, 39 insertions(+)
   create mode 100644 Paper/Paper.tex
(base) jenniferlin@Jennifers-iMac hello_git %
```

Finding an Older Version

Now that we merged the paper to the `main` branch, everyone on the team can see it as the official draft. Now, suppose one of your co-authors goes in to make some edits but you don't want those changes. You and the team decide to revert to a previous version of the file. Here is how we do it.

NOTE that reverting to a different version of a file reverts the ENTIRE repository. So if you commit multiple files under one commit, you WILL revert on ALL of those files. *Use this step WISELY!!!*

To start, first, let's change the paper, compile it and commit the revisions. As an aside, remember that when you compile L^AT_EX documents, there are a host of auxiliary files that you do not need. But there is the PDF file that you need. Notice how `.gitignore` treats it.

```

(base) jenniferlin@Jennifers-iMac hello_git % 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:   Paper/Paper.tex

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Paper/Paper.pdf

no changes added to commit (use "git add" and/or "git commit -a")
(base) jenniferlin@Jennifers-iMac hello_git % git add .
(base) jenniferlin@Jennifers-iMac hello_git % git commit -m "Modified paper template"
[main 29b0fd8] Modified paper template
 2 files changed, 4 insertions(+), 5 deletions(-)
 create mode 100644 Paper/Paper.pdf
(base) jenniferlin@Jennifers-iMac hello_git % git push
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 42.68 KiB | 42.68 MiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/lin-jennifer/hello_git.git
  edb245b..29b0fd8  main -> main
(base) jenniferlin@Jennifers-iMac hello_git %

```

Here, we are only interested in the .tex file and the .pdf output and those are the only two files that we are committing. Excellent.

Now, let's make another change to the .tex file and commit that.

```

[(base) jenniferlin@Jennifers-iMac hello_git % 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:   Paper/Paper.tex

no changes added to commit (use "git add" and/or "git commit -a")
[(base) jenniferlin@Jennifers-iMac hello_git % git add .
[(base) jenniferlin@Jennifers-iMac hello_git % git commit -m "Change Title"
[main 1254ebc] Change Title
  1 file changed, 1 insertion(+), 1 deletion(-)
[(base) jenniferlin@Jennifers-iMac hello_git % git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 387 bytes | 387.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/lin-jennifer/hello_git.git
  29b0fd8..1254ebc  main -> main
(base) jenniferlin@Jennifers-iMac hello_git %

```

But suppose that the group wants to go back to the old version, which is the one before we made this title change. Let's find where that version is, in the grand scheme of things. We can use `git log` for that.

```

[(base) jenniferlin@Jennifers-iMac hello_git % git log
commit 1254ebc7c44114115bdc551b1e17822ebc845f1a (HEAD -> main, origin/main, orig
in/HEAD)
Author: lin-jennifer <jennifer.lin16@ncf.edu>
Date:   Sun Sep 4 20:41:38 2022 -0500

  Change Title

commit 29b0fd851239cf478cf44905cccd1f27894642a8e
Author: lin-jennifer <jennifer.lin16@ncf.edu>
Date:   Sun Sep 4 20:25:18 2022 -0500

  Modified paper template

commit edb245bd6b4c222ba03d6917cabdefd1d7986b7d
Merge: df4dee1 9bb91c0
Author: Jennifer Lin <42876762+lin-jennifer@users.noreply.github.com>
Date:   Sun Sep 4 20:10:07 2022 -0500

```

Here is the one we want to revert to:

```
commit 29b0fd851239cf478cf44905ccd1f27894642a8e
Author: lin-jennifer <jennifer.lin16@ncf.edu>
Date:   Sun Sep 4 20:25:18 2022 -0500
```

Modified paper template

Notice the long commit code. This is the unique identifier for each version of the repository. To go back to any commit, we first need to copy this code. Then, get out of the log by pressing q on the keyboard. Finally, use `git revert + commit code` to revert the changes. Here is where the Mac gets a bit tricky. When you run Revert, it will prompt you to provide a commit message. But Terminal can freeze. Get out of terminal, reenter, use `cd` and point the terminal to your git repository and use `git status` to commit the changes.

You can go to the commit log on the GitHub website or the file itself to see your changes undone.

The screenshot shows a GitHub repository page for 'lin-jennifer / hello_git'. The 'Code' tab is selected. Below it, the 'main' branch is shown. The commit history for 'main' on Sep 4, 2022, includes three commits:

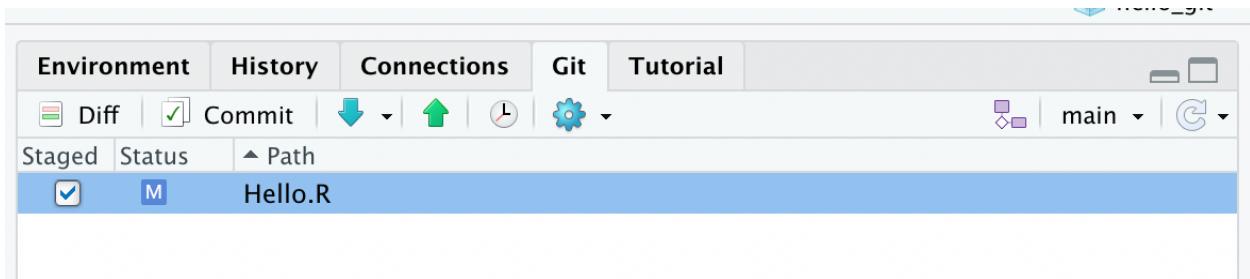
- Revert title change (commit cbfaa08)
- Change Title (commit 1254ebc)
- Modified paper template (commit 29b0fd8)

Git Point and Click

If the whole Terminal and Web App is not your taste, you have other options to interact with `git`. Most text editors have their connections to `git` that operate quite similarly as the Terminal but are more point and click friendly.

GitHub and R Studio

One of the ways to point and click is with R Studio. In our example, we created an R project in the repository. We can use R Studio to demonstrate its point and click capacities. Let's revisit the R script and make some changes.



Using the Git tab on the upper right hand corner, we can make commits to Git. NOTE that this only works if you open an R project in a folder that is a GitHub repository.

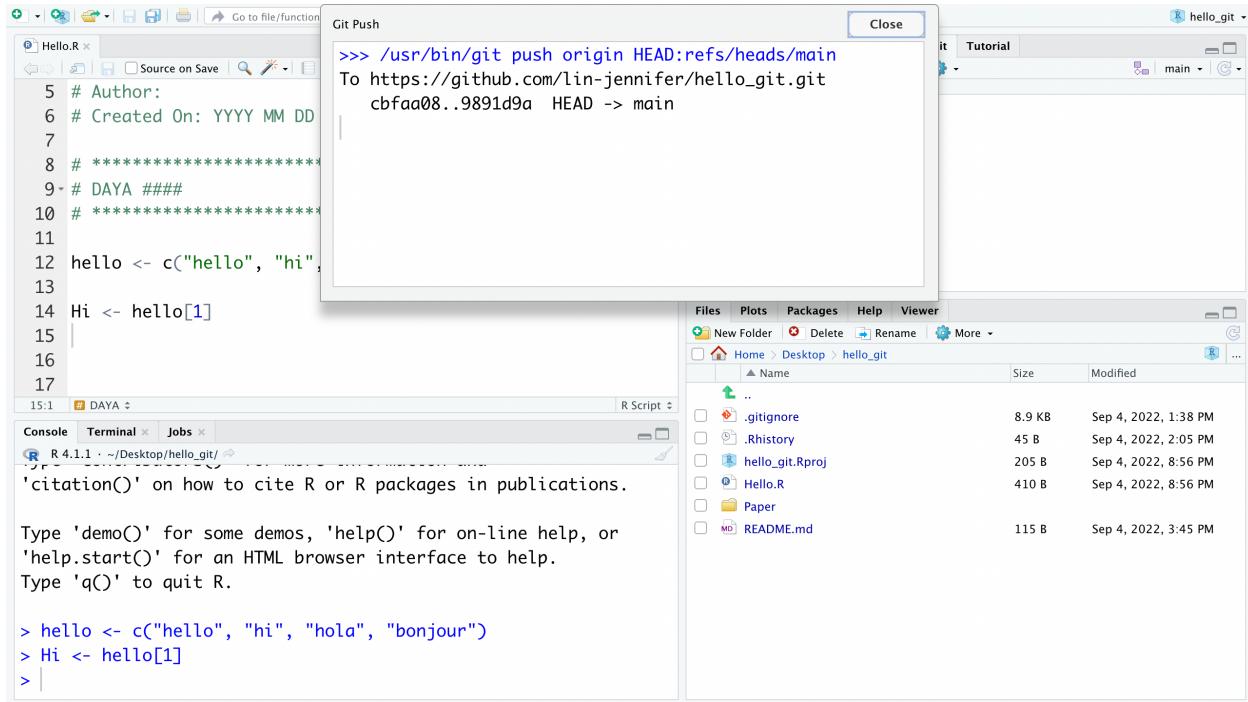
We can commit by hitting the “Commit” button and a window will pop up and will ask for a commit message, while showing the changes that were made.

```

@@ -9,8 +9,8 @@
 9 | # DAYA #####
10 | # ****
11 |
12 | hello <- c("hello", "hi", "hola", "bonjour")
13 |
14 | Hi <- hello[1]
15 |
16 |

```

We can hit “Commit” and close the window. To push, press the green arrow that points up on the same menu bar where you found “Commit”. Here is that result:



To pull, the blue down arrow is what you would need. Notice that there is also a dropdown for your branches. Now, we are on `main` but if you click that, you can also see your other branches.

GitHub Desktop

Besides that, there is an app called [GitHub Desktop](#) that is even more point and click than the R version. It is very user friendly.

So let's open up GitHub desktop. Once you log in, you will see options to get started with creating your own repository, among others. Since I already have a repository created and working from before, I am just going to clone it in.³ Notice that you need to clone it on the Desktop App in order to use GitHub Desktop. Most of the repositories on the list in the opening page live on my computer already but I cannot integrate them with the Desktop unless I clone them on the Desktop app.

To clone, select the repository you want to clone. Here, I will use the `hello_git` one that we worked on throughout this demonstration.

³My GitHub is rather public with many private repositories but at least this demonstration let's you see how much I have used GitHub over the years. All my projects and course notes are all included in here.

Let's get started!

Add a repository to GitHub Desktop to start collaborating



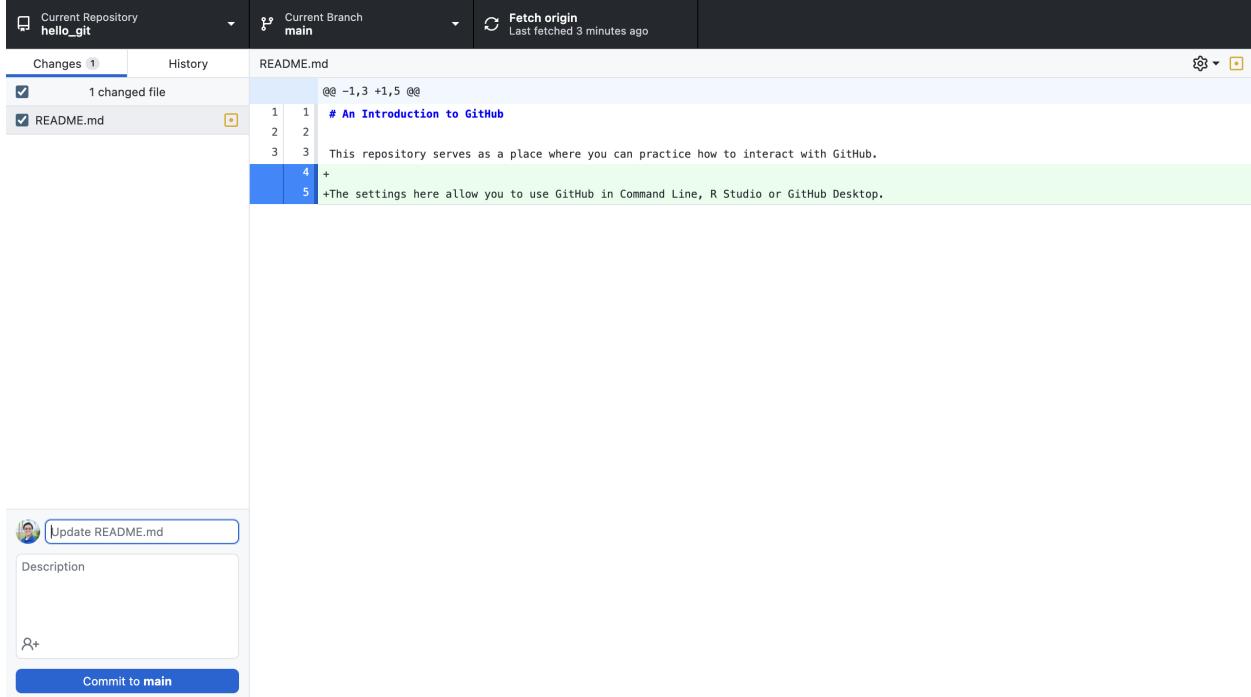
The screenshot shows the GitHub Desktop application interface. On the left, there's a sidebar with options: 'Create a Tutorial Repository...', 'Clone a Repository from the Internet...', 'Create a New Repository on your Hard Drive...', and 'Add an Existing Repository from your Hard Drive...'. A 'ProTip!' box says: 'You can drag & drop an existing repository folder here to add it to Desktop'. The main area lists repositories under 'lin-jennifer': COVIDStates, DemDebateAnalysis, Dissertation, DistToPolls, EliteAppeals, FLSpatialPolitics, ggplot-workshop, GOTV, GradApps, GradNotes, hello_git (which is highlighted in blue), hot-and-spicy, LaTeXTemplates, and lin. On the right, there are icons for creating a new repository, cloning, pushing, pulling, and more. At the bottom, a blue button says 'Clone lin-jennifer/hello_git'.

To clone, you see the same URL that we copied in the `git clone` demonstration above. As we did above, we need to pick a file path. The default is your Documents folder, but I don't want that. I want my Desktop and so I will indicate as such.

The screenshot shows the 'Clone a Repository' dialog box. It has tabs for 'GitHub.com' and 'GitHub Enterprise', with 'GitHub.com' selected. The 'URL' tab is active, showing the URL 'https://github.com/lin-jennifer/hello_git.git'. Below it, the 'Local Path' field contains '/Users/jenniferlin/Desktop/hello_git'. There are 'Choose...' and 'Cancel' buttons. At the bottom, there's a link to 'lin-jennifer/GradNotes'.

Now, when you go out to your destination, you will see the file cloned there if it is not there

already. I am going to open up the `hello_git` folder and make a change. I will add another note to the `README.md` file. It does not matter what you add or change. When you make those changes and save them to your computer, inside the folder for the repository, GitHub Desktop will immediately recognize it and show that there is a change to be committed.



The screenshot shows the GitHub Desktop application interface. At the top, there are three dropdown menus: "Current Repository" set to "hello_git", "Current Branch" set to "main", and "Fetch origin" with a note "Last fetched 3 minutes ago". Below these are two tabs: "Changes" (which is selected, showing 1 changed file) and "History". Under "Changes", a list shows "1 changed file" named "README.md". The diff view shows the following content:

```
@@ -1,3 +1,5 @@
# An Introduction to GitHub
This repository serves as a place where you can practice how to interact with GitHub.
+The settings here allow you to use GitHub in Command Line, R Studio or GitHub Desktop.
```

At the bottom left, there is a "Commit to main" button. To its left is a "Description" field which is currently empty. Above the "Commit to main" button is a "Commit message" field containing the text "R+".

Immediately, you will see a few sections that might look familiar from earlier sections. There is a list of the files that changed, which is similar to `git status`, information on what changed, similar to the diff tool on the GitHub website, and a window on the bottom left hand corner for you to leave a commit message. I am going to hold off on the commit for now and I am going to open up the R script in this folder and make another change. This is to demonstrate separate adds with separate commit messages, as I did in previous sections.

Current Repository: hello_git | Current Branch: main | Fetch origin: Last fetched 10 minutes ago

Changes 2 | History

2 changed files

Hello.R

README.md

README.md

```

@@ -1,3 +1,5 @@
 1 1 # An Introduction to GitHub
 2 2
 3 3 This repository serves as a place where you can practice how to interact with GitHub.
 4 +
 5 +The settings here allow you to use GitHub in Command Line, R Studio or GitHub Desktop.

```



When I had just updated the README, I get a default commit message that said “Update README.md”. Now, I no longer get a default message. Also, notice that both files are checked. This is the same as `git add .` in the earlier sections. I want to change that. Let’s commit the changes on the R file first and the README second.

To do that, I am going to un-check the README and just look at the R script. I will write a commit message that is not the default “Update Hello.R” and press “Commit to Main”.

Current Repository: hello_git | Current Branch: main | Fetch origin: Last fetched 13 minutes ago

Changes 2 | History

2 changed files

Hello.R

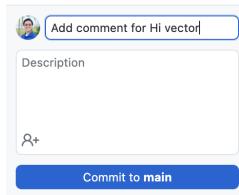
README.md

Hello.R

```

... @@ -11,6 +11,7 @@
 11 11
 12 12 hello <- c("hello", "hi", "hola", "bonjour")
 13 13
 14 +# Get the first "hello" from the vector above
 15 15 Hi <- hello[1]
 16 16
 17 17

```



Now I will commit the second using another custom message (“Add Note on GitHub Possibilities”).

To see the results, let’s take a detour to the GitHub Web App.

The screenshot shows a GitHub repository page for 'lin-jennifer/hello_git'. The repository is public. The navigation bar includes 'Code' (selected), 'Issues', 'Pull requests', 'Actions', and 'Projects'. Below the navigation, it shows 'main' branch selected, 2 branches total, and 0 tags. The commit history is listed:

Commit	Message
	lin-jennifer Add a vector for hi
	Paper Revert title change
	.gitignore Create .gitignore
	Hello.R Add a vector for hi
	README.md Update README.md
	hello_git.Rproj Initiate R Project for Repo

Notice the commit messages. The most recent one is always the one next to your account name. I should have “Add Note on GitHub Possibilities” next to my name, but I do not. This means that something did not push. To Push, go to GitHub Desktop and finish adding all of your files. Once you do, you will see the following:

No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.



Push commits to the origin remote

You have 2 local commits waiting to be pushed to GitHub.

[Push origin](#)

Always available in the toolbar when there are local commits waiting to be pushed or ⌘ P

Click the blue button to push. Now, let's see the result on the Web App.

[lin-jennifer / hello_git](#) Public

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Raw](#)

main 2 branches 0 tags

lin-jennifer Add Note on GitHub possibilities

Paper	Revert title change
.gitignore	Create .gitignore
Hello.R	Add comment for Hi vector
README.md	Add Note on GitHub possibilities
hello_git.Rproj	Initiate R Project for Repo

Much better! Notice that both of the commit messages that we wrote using GitHub Desktop added with one push. This is possible to do with command line and with R Studio as well.

Concluding Thoughts

`git` is a powerful tool. It can potentially save you some time at some point, and it can also help you save some computer storage space with all the versions that you are creating. However, like all powerful tools, `git` and GitHub are not useful if you don't use it. Committing and pushing, and writing good commit messages takes time and practice.

The major commands of `git pull`, `git status`, `git add`, `git commit -m "Message"` and `git push` will almost become second nature if you use it enough. In this tutorial, I showed you the three main methods that people use to interact with GitHub and `git`. This is to say, too, that if you wanted the main highlight from this tutorial, these five commands will serve you well in the long run if you become familiar with it. And, this is also to say that you can have your own preferences for what works for you. You are managing your own workflow after all, so you should not work to the tastes of others.

If, after reading this tutorial, you are not finding the version tracking and comparison tools that `git` has to offer to be appealing, then perhaps, I can point you to some references that might convince you otherwise. You might still be asking “Why Bother?” My answer is “why not?” But, for a more elaborate answer, I would recommend checking out Kieran Healy’s book [*A Plain Person’s Guide to Plain Text Social Science*](#). In here, he covers version control, plain text editors, and overall, why you should bother with plain text tools for your research workflow.