

3. Git & GitHub

Self-Paced Workshop #3

Prof. Jack Reilly

Fall 2026

GitHub Workshop: Your First Repository

In lecture we drew the workflow diagram: a repository lives in the cloud, a copy lives on your computer, and four actions move work between them — **add**, **commit**, **push**, and **pull**. This workshop is where we actually do it. By the end you will have a real repository on GitHub that mirrors a real folder on your machine, and you will have moved a change in both directions.

Work through this at your own pace, in front of your own computer. If something doesn't look like what's described here, note where you got stuck and bring it to class — this is an area where being stuck for ten minutes is normal and being stuck for an hour is not.

Before you start

You need two things from the [Week 3](#) page:

1. A GitHub account.
2. GitHub Desktop installed **and signed in** to that account.

If either is missing, do that first — it takes about five minutes. GitHub Desktop signs you in through your browser; you only need a Personal Access Token if you are using a third-party client or the shell (command line).

The five magic words for understanding git

Git has its own language, but the words mean pretty logical things:

Word	What it means
repository ("repo")	A folder that git is watching. It exists in two places: on GitHub, and on your computer.
commit	A labelled snapshot of your folder at one moment. This is the unit of version history.
push	Send your commits up to GitHub.
pull	Bring GitHub's commits down to your computer.

Word	What it means
clone	Make the first local copy of a repo that already exists on GitHub.

Remember the point from lecture: **truth lives on GitHub**, not on your laptop. Your computer holds a working copy. That is why the day — or work session — ends with a push.

You may also hear about **add** (or “stage”) — choosing *which* changed files go into the next commit. GitHub Desktop does this with the checkboxes next to each filename, so you mostly won’t need to think about it. If you want, however, note that it is possible to commit only *some* files while leaving others uncommitted through the **add** functionality.

Part 1. Create a repository

We are going to build the repo you will use for Assignment 4, so this is not a throwaway exercise.

1. In GitHub Desktop, choose **File > New repository...**
2. **Name:** `dvw-assignment-4`
3. **Local path:** put it wherever you keep coursework. Click *Choose...* and navigate there.
4. Check **Initialize this repository with a README**.
5. **Git ignore:** choose **R** from the dropdown.
6. Leave **License** as None.
7. Click **Create repository**.

Why the repo name has dashes and no spaces

Repo names become URLs and folder names. Spaces, apostrophes, and # all cause trouble somewhere down the line. `dvw-assignment-4` is safe forever; Jack's DWV Assignment #4 is a problem waiting to happen. Many like using underscores (`_`) as well as dashes for this purpose.

What the `.gitignore` is for

A `.gitignore` file is a list of things git should pretend not to see. The **R** template you just picked ignores `.Rhistory`, `.RData`, and `.Rproj.user` — session junk that changes every time you open R and that nobody needs in your version history. Without it, every commit is cluttered with noise. You can read the template [here](#).

macOS users: Macs *unhelpfully* have a habit of littering your git repos with `.DS_Store` files. You can exclude these from GitHub and repo tracking by adding them to the `.gitignore` file. (`.gitignore` is just a plain text file! You can add whatever you like!)

Now look at what you made. Choose **Repository > Show in Finder** (macOS) or **Repository > Show in Explorer** (Windows). You should see `README.md` and, if your system shows hidden files, a `.gitignore` and a `.git` folder.¹ That `.git` folder is the entire version history. Don't touch it, ever — but know that it's what makes this folder a repo rather than an ordinary folder.

At this point the repo exists *only* on your computer. Fix that:

8. Back in GitHub Desktop, click **Publish repository** in the top bar.
9. Check **Keep this code private**.
10. Click **Publish repository**.

Checkpoint 1

Go to <https://github.com> in your browser and look at your profile. `dww-assignment-4` should be sitting there, with a `README` in it. If it isn't, stop here — don't work past a failed checkpoint, since every later step builds on this one.

Part 2. Make a change and commit it

1. Open the repo folder (**Repository > Show in Finder / Show in Explorer**).
2. Create a Quarto document in that folder and save it as `DWV Assignment 4.qmd`. Put anything in it for now — a title and one sentence is plenty.
3. Return to GitHub Desktop and look at the **Changes** tab on the left.

Your new file is listed, with a checkbox beside it. That checkbox is “add” — it decides what goes into this snapshot. Leave it checked.

4. In the **Summary** box at the bottom left, type a real message: `Add starter Quarto file for assignment 4`.
5. Click **Commit to main**.
6. Click the **History** tab.

You now have two commits: the initial one, and yours. Click your commit and you can see exactly which lines changed. This is the thing that makes version control worth the trouble — not the backup, the *record*.

Write commit messages for future you

`Add starter Quarto file for assignment 4` is useful. `stuff`, `asdf`, and `update` are not. Six weeks from now, when you're trying to find the version that still worked, the message is all you have.

Checkpoint 2

¹On macOS, you can hit `cmd-shift-.` to reveal hidden files on your desktop. On Windows, select **View → Show → Hidden items**.

Refresh your repo page on github.com. **Your new file is not there.** That's correct and it is the single most important thing to understand in this workshop: a commit is *local*. You have snapshotted the change on your own machine and told GitHub nothing about it.

Part 3. Push

1. In GitHub Desktop, click **Push origin** in the top bar. (It shows a small number — the count of commits waiting to go up.)
2. Refresh your repo page on github.com.

There's your file. You have now completed the full outbound trip: edit -> commit -> push.

Checkpoint 3

DWV Assignment 4.qmd is visible on github.com, and clicking it shows your text.

Part 4. Pull

Push sends work up. Pull brings work down. You need pull whenever the GitHub copy has changed and your local copy hasn't — because a collaborator committed something, because you worked on a different computer, or, most commonly in this course, because you edited a file in your browser.

Let's create that situation deliberately.

1. On github.com, in your repo, click README.md.
2. Click the **pencil** (edit) icon.
3. Add a line: This repo holds my work for Assignment 4 in PAI 741.
4. Scroll down and click **Commit changes**.

You have just made a commit that exists on GitHub and *not* on your computer. Confirm that: open README.md in the folder on your machine. Your new line isn't there.

5. In GitHub Desktop, click **Fetch origin**. The button changes to **Pull origin** with a count.
6. Click **Pull origin**.
7. Open README.md on your computer again.

Checkpoint 4

The line you typed in the browser is now in the file on your laptop. You have moved a change in both directions, which means you know how to use git.

Part 5. The rhythm

That's the whole loop, and it has an order:

pull -> work -> commit -> push

The one habit that prevents most problems

Pull before you start working, push when you stop.

Almost every git horror story you will hear comes from someone editing a stale copy — they started work without pulling, so their computer and GitHub both changed independently, and now git has two versions of the same line and no way to know which one you meant. That's a *merge conflict* — quite possibly the most terrifying phrase anyone using git will ever hear. It is fixable, but the easy move is not creating one.²

Practice the loop twice more before you start Assignment 4: add a real section to your `.qmd`, commit it with a sensible message, push it. Then edit the README in the browser again and pull it down. Three repetitions and this stops feeling like a ritual.

What belongs in a repo

Git was built for plain text — code, `.qmd` files, `.R` scripts, README files. It handles those beautifully, showing you line-by-line what changed. It is much worse with everything else.

Put it in the repo	Keep it out
<code>.R</code> and <code>.qmd</code> files	Large datasets
README.md	Anything confidential or restricted-use
<code>.gitignore</code>	Session junk (<code>.RData</code> , <code>.Rhistory</code>) — the R template handles this
Rendered <code>.html</code> / <code>.docx</code> you're submitting	Anything over 100 MB (GitHub will refuse the push)

For this course, keep your data in a separate folder *outside* the repo, as the assignment describes. If you do that, say in your README where the data came from and where it should go, so that someone cloning your repo knows what they're missing.

Two things git cannot save you from

²Seriously. There are plenty of ways to fix it — and many of the intimidating words you'll hear around git refer to fixing merge conflicts, like *mergetool* — but just don't let yourself get into this situation in the first place. Git is, by design, *intentional file sync* — we should have a human in the loop to always be assured of file provenance — but that does leave this particular pain point always possibly open.

A repo is version control, not a backup — if your only copy of the repo is the one on your laptop, you have one copy. And a private repo is a *courtesy*, not a security guarantee: never commit anything you would be upset to see public.

When it goes wrong

Symptom	What's usually happening
Nothing appears in the Changes tab	You didn't save the file, or you saved it outside the repo folder. Check the path.
"Authentication failed"	Sign out and back in: GitHub Desktop > Settings/Options > Accounts .
Push origin is greyed out	You haven't committed yet. Commit first, then push.
Desktop wants you to pull before pushing	GitHub has commits you don't. Pull, then push.
The push is rejected for file size	Something over 100 MB is in the repo — usually a dataset. Add it to <code>.gitignore</code> .
A merge conflict	Don't panic and don't start deleting files. Bring it to class or email me.

The nuclear option is legitimate

If a repo gets into a state you can't reason about, it is entirely acceptable to copy your files somewhere safe, delete the repo, and start a fresh one. Professionals do this all the time (even if they'll sometimes only admit to it in a dark room where they don't think they can be overheard). Happy Git even has a chapter for it, cheerfully titled "[Burn it all down](#)".

If you aren't using GitHub Desktop

The steps are the same and the vocabulary is identical; only the buttons move.

- **RStudio:** the **Git** pane (top right) has Commit, Pull, and Push. See [Happy Git, ch. 12, "Connect RStudio to Git and GitHub"](#). Note that pushing over HTTPS from RStudio *does* require a Personal Access Token — [Happy Git ch. 9, "Personal access token for HTTPS"](#) covers it.
- **Positron:** the **Source Control** panel in the sidebar.
- **Command line:** `git add .`, `git commit -m "message"`, `git push`, `git pull`.
- Other software: poke around. It'll be somewhere!

Where this is going

Assignment 4 is the version control assignment, and you have just done its entire setup section. Before you submit, the repo needs:

- A README
- A .gitignore
- Your DWV Assignment 4.qmd
- Your rendered .html and .docx, **pushed** (not just committed)
- Either public, or private and shared with me — I'm @jacklreilly on GitHub
- A link to the repo posted on Blackboard

Going further

- [Happy Git with R](#), by Jenny Bryan and colleagues — the standard reference for this material. Ch. 11, “Connect to GitHub”, is the push/pull walkthrough we just did by hand.
- [The Plain Person's Guide to Plain Text Social Science](#), ch. 2 — why any of this matters.
- GitHub's own [first-repository walkthrough](#).
- Bryan, “Excuse me, do you have a moment to talk about version control?”