Data Science for Public Policy

Aaron R. Williams - Georgetown University

Reproducible Research with Git and GitHub

Reading

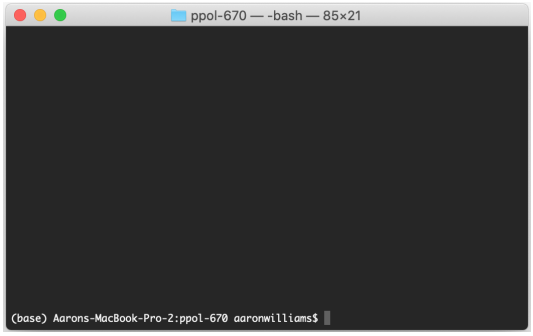
- Git Handbook
- Mastering Markdown
- Understanding the GitHub Flow
- Documenting Your Projects on GitHub
- After class: Git Tutorial

Convention: <> are used throughout this guide to indicate blanks that need to be filled in. Don't actually submit <>. Instead, replace them with the desired text.

Command Line

The command line (also known as shell or console) is a way of controlling computers without using a graphical user interface (i.e. pointing-and-clicking). The command line is useful because pointing-and-clicking is tough to reproduce or scale and because lots of useful software is only available through the command line. Furthermore, cloud computing often requires use of the command line.

We will run Bash, a command line program, using Terminal on Mac and Git Bash on Windows. Open Terminal like any other program on Mac. Right-click in a desired directory and select "Git Bash Here" to access Git Bash on Windows.



Fortunately, we only need to know a little Bash for version control with Git and cloud computing.

`pwd` - print working directory - prints the file path to the current location in the

`ls` - list - lists files and folders in the current working directory.

`cd` - change directory - move the current working directory.

`mkdir` - make directory - creates a directory (folder) in the current working directory.

`touch` - creates a text file with the provided name.

`mv` - move - moves a file from one location to the other.

`cat` - concatenate - concatenate and print a file.

Exercise 01

1. Create a new directory called `cli-exercise`.
2. Navigate to this directory using `cd` in the Terminal or Git Bash.
3. Submit `pwd` to confirm you are in the right direction.

Exercise 02

The following code will create a new text document called `haiku.txt` in the working directory.

```
pwd
ls
touch haiku.txt
ls
```

Exercise 03

The following will add the haiku “The Old Pond” by Matsuo Basho to `haiku.txt`.

```
cat haiku.txt
echo "An old silent pond" >> haiku.txt
cat haiku.txt
echo "A frog jumps into the pond-" >> haiku.txt
echo "Splash! Silence again." >> haiku.txt
echo "~Matsuo Basho" >> haiku.txt
cat haiku.txt
```

Exercise 04

```
ls
mkdir poems
ls
mv haiku.txt poems/haiku.txt
ls
cat poems/haiku.txt
```

Useful tips

- Tab completion can save a ton of typing. Hitting tab twice shows all of the available options that can complete from the currently typed text.
- Hit the up arrow to cycle through previously submitted commands.
- Use `man <command name>` to pull up help documentation. Hit `q` to exit.
- Typing `..` refers to the directory above the working directory. Writing `cd ..` changes to the directory above the working directory.
- Typing just `cd` changes to the home directory.

Programs

We can access programs from the command line like `git`. Commands from programs always start with the name of the program. For example:

```
git init
git status
```

Why version control?

Version control is a system for managing and recording changes to files over time. Version control is essential to managing code and analyses. Good version control can:

- limit the chance of making a mistake
- maximize the chance of catching a mistake when it happens
- create a permanent record of changes to code
- easily undo mistakes by switching between iterations of code
- allow multiple paths of development while protecting working versions of code
- encourage communication between collaborators
- be used for external communication

Why distributed version control?

Centralized version control stores all files and the log of those files in one centralized location. *Distributed version control* stores files and logs in one or many locations and has tools for combining the different versions of files and logs.

Centralized version control like Box is good for sharing a Microsoft Word document but it is terrible for collaborating on code. Distributed version control allows for the simultaneous editing and running of code. It also allows for code development without sacrificing a working version of the code.

Git vs. GitHub

Git is a distributed version-control system for tracking changes in code. Git is free, open-source software and can be used locally without an internet connection. It's like a turbo-charged version of track changes for code.

GitHub, which is owned by Microsoft, is an online hosting service for version control using Git. It also contains useful tools for collaboration and project management. It's like a turbo-charged version of Box for sharing repositories created using Git.

At first, it's easy to mix up Git and GitHub. Just try to remember that they are separate tools that complement each other well.

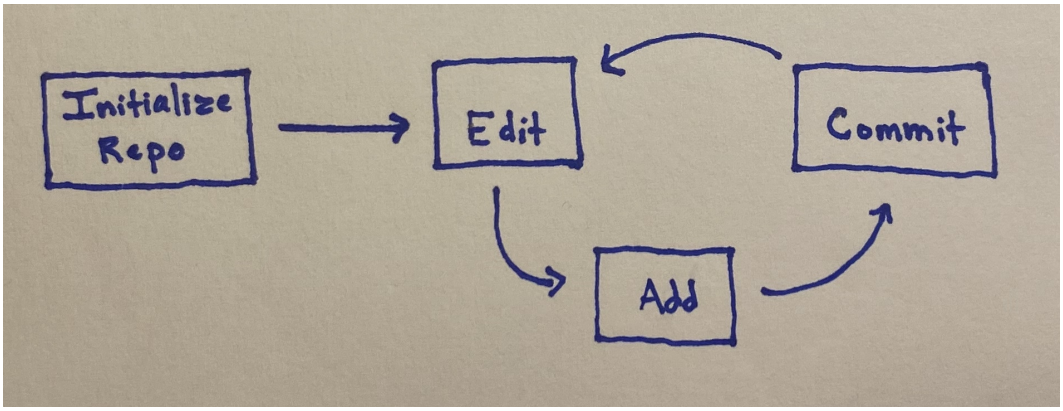
Workflow 1: Local Git

Repository: A collection of files, often a directory, where files are organized and logged by Git.

Git and GitHub organize projects into repositories. Typically, a "repo" will correspond with the place where you started a .Rproj. In our first Git workflow, we will initialize a repo on our computer. Later, we will learn how to copy (clone) someone else's repo for collaboration.

Basic Approach

1. Initialize a repository for a project.
2. Tell Git which files to track. Track scripts. Avoid tracking data or binary files like `.xlsx`.
3. Take a snapshot of tracked files and add a commit message.
4. Repeat, repeat, repeat



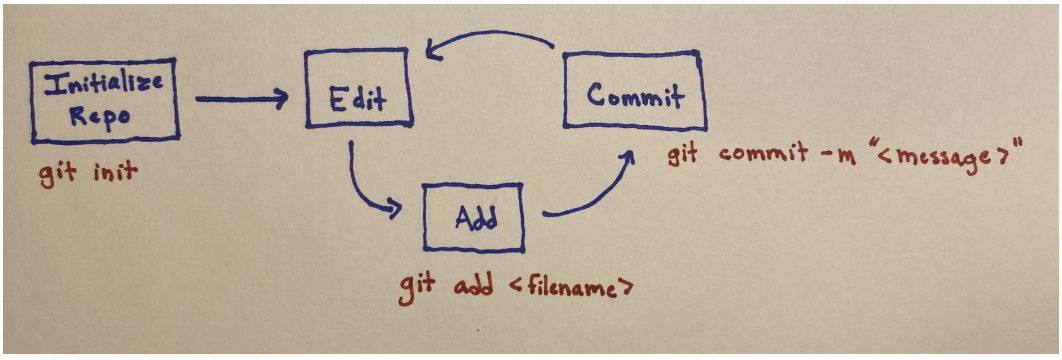
Commands

`git init` initializes a local Git repository. This only needs to be run once per project.

`git status` prints out all of the important information about your repo. Use it before and after most commands to understand how code changes your repo.

`git add <file-name>` adds a file to staging. It says, "hey look at this".

`git commit -m "<message>"` commits changes made to added files to the repository. It says, "hey, take a snapshot of the things you looked at in the previous command." **Don't forget the `-m`.**



`git log --oneline` shows the commit history in the repository.

`git diff` shows changes since the last commit.

Exercise 05

Navigate (or stay) in your `cli-exercise` directory.

```
# confirm that there is no git repo
git status

# start a git repo
git init

# confirm that there is a git repo
git status

# add
git add poems/haiku.txt
git status

# commit
git commit -m "initialize the repo and a poem"
git status
```

Let's make a few changes.

```
# edit the poem
echo " " >> poems/haiku.txt

touch README.md
echo "The repo contains poems " >> README.md
```

Now use `git add` and `git commit` to repeat the process above with `README.md` and `poems/haiku.txt`. You don't need to repeat `git init`.

Workflow 2: Git + GitHub (no branching)

Workflow 1 is only for solo work and doesn't use GitHub. Let's add GitHub to our workflow so we can better collaborate with others.

Option A: Create a new repository

This only needs to happen once per repository

1. Initialize a local repository as outlined above.
2. On GitHub, click the plus sign in the top, right corner and select `New Repository`.
3. Create a repository with the same name as your directory.
4. Copy the code under **...or push an existing repository from the command line** and submit it in the command line that is already open.

Option B: Clone an existing repository

This only needs to happen once per repository

1. Right click where you want to add the repo and select `Git Bash Here`. This will open the command prompt.

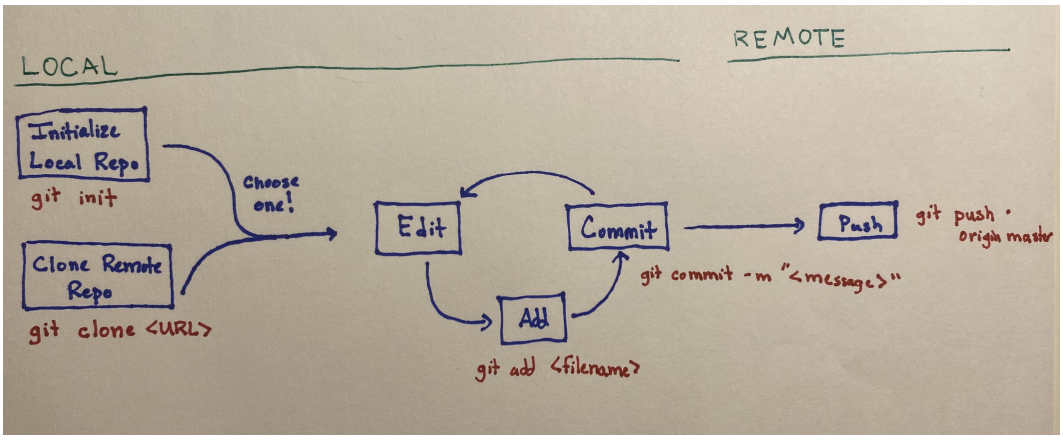
2. Navigate to the GitHub page of the desired repo. Click the big green button titled `Clone or download`.
3. Copy the link and then submit `git clone` followed by the copied link.
4. Navigate into the recently cloned directory.

Basic commands and workflow

The objective is to create local repositories that track code, and sometimes data, and then push those repositories to GitHub for back-up and collaboration.

`git clone <link>` copies a remote repository from GitHub to the location of the working directory on your computer.

`git push origin main` pushes your local commits to the remote repository on GitHub. It says, "hey, take a look at these snapshots I just made". It is possible to push to branches other than `main`. Simply replace `main` with the desired branch name.



Exercise 06

1. Go to GitHub.
2. Click the big green "New" button.
3. Call the repo `cli-exercise`. Don't select anything else.
4. Follow the instructions under "...or push an existing repository from the command line".
5. **Something may break here. We are going to fix it together.** If you have a new version of Git, it may ask you for your GitHub password. Otherwise, we will need to create a Personal Access Token (PAT).
6. Make a change to the README.
7. Repeat all Git and GitHub steps necessary to add, commit, and push this change to GitHub.

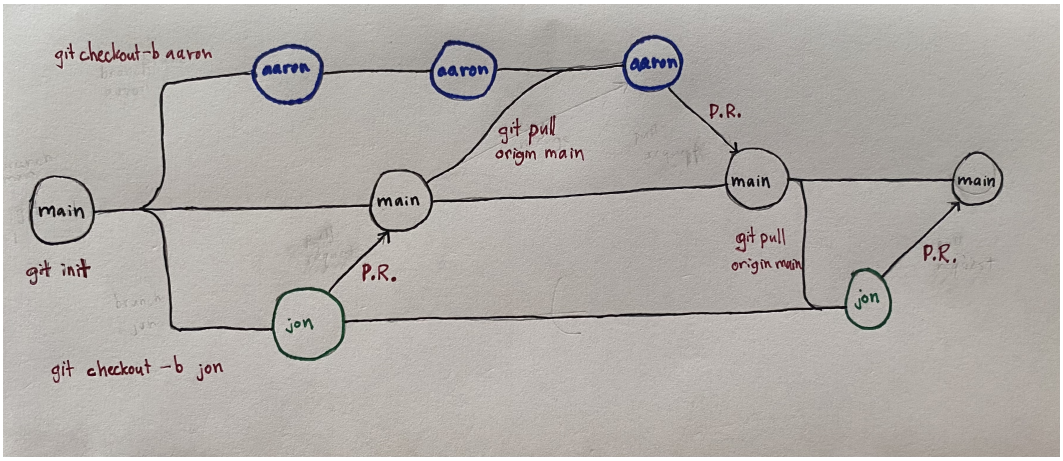
We won't practice cloning right now. Note that if you click the big green "Code" button in a repo, you can copy the URL that is used with `git clone <url>`.

Workflow 3: Git + GitHub (with branching)

There are two main models for collaborating with GitHub.

1. Shared repository with branching
2. Fork and pull

I've almost exclusively seen approach one used by collaborative teams. We won't learn branching today but I encourage everyone to use branching for their projects.



Git is Confusing

I promise that it's worth it.

Resources

- Git Cheat Sheet
- Happy Git and GitHub for the UserR
- Git Pocket Guide
- Getting Git Right
- Git Learning Lab