Git/GitHub + Reproducible Environments

January 27, 2025

Lab 1 Clarifications

- Problem formulation:
  - You are not expected to come up with a groundbreaking scientific question
  - Descriptive questions are ok, no “modeling” is necessary
    - End goal: exploratory data analysis/visualizations
  - The connection between your analysis and domain problem should be reasonable, not bulletproof
  - Ex: How much light reaches the bottom of a redwood tree?
    - Understanding the amount of light available gives us insight into the types of plants that are able to grow in that environment
  - 1-3 sentences per question in the problem formulation part should suffice
  - It’s ok to ask questions. This is part of collaboration.
- Quickstart on data cleaning and EDA: coming up Wednesday
  - If you have not read Tolle et al., please do so before next class
- Do not wait until the last minute to work on this. Chip away at it; don’t binge.

Plan for Today

1 Git and GitHub

+ Creating your final project GitHub repository
+ GitHub workflow: git pull, add, commit, push

2 Towards a reproducible workflow

+ Setting up your project file structure
+ Creating a reproducible environment (renv, conda, ...)

A necessary tool for today: quarto

Download quarto: https://quarto.org/docs/get-started/

What is **quarto**?

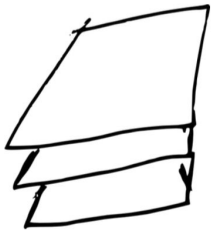
- For reproducible documentation with code
- Essentially like R Markdown, but is not specific to R
- Can be used with R, Python, Julia, ...
- Can render reports/articles, presentations, dashboards, websites, blogs, and books in HTML, PDF, MS Word, ...

Introduction to Git/GitHub

Typical Git/GitHub Pipeline

(2) make local changes

(e.g., create file called filename.txt)



LOCAL REPOSITORY

(1) git pull

(to retrieve the most recent version from the server)

REMOTE REPOSITORY

(3) git add filename.txt

(changes are staged/waiting to be committed)

(4) git commit -m "[description of changes]"

(commit when you have made some changes and want to be able to save your current checkpoint as a snapshot)

(5) git push

(make changes available to everyone with access to the repo)

Warning: remember to "git pull" before "git push" to mitigate potential merge conflicts

Setting up a GitHub repository

1. Three (of many) possible starting points:
   a. Start with an *existing* remote repository from GitHub
   b. Create a *new* remote repository on GitHub
      i. I usually initialize a **private** repo with a **README** file and with a **license** (e.g., MIT)
   c. Convert local folder into a git/GitHub repository

Setting up a GitHub repository

1. Three (of many) possible starting points:
   a. Start with an *existing* remote repository from GitHub
   b. Create a *new* remote repository on GitHub
      i. I usually initialize a **private** repo with a **README** file and with a **license** (e.g., MIT)
   c. Convert local folder into a git/GitHub repository

2. Clone remote repository to your local computer via one of the following:
   a. **Command Line:**
      `git clone https://github.com/[username]/[repositoryname].git`
      e.g,. `git clone https://github.com/tiffanymtang/dsip-s25.git`
   b. **GitHub Desktop:** click green “Code” button > “Open with GitHub Desktop”

Setting up a GitHub repository

1. Three (of many) possible starting points:
   a. Start with an *existing* remote repository from GitHub
   b. Create a *new* remote repository on GitHub
      i. I usually initialize a **private** repo with a **README** file and with a **license** (e.g., MIT)
   c. Convert local folder into a git/GitHub repository

2. Clone remote repository to your local computer via one of the following:
   a. **Command Line:**
      `git clone https://github.com/[username]/[repositoryname].git`
      e.g,. `git clone https://github.com/tiffanymtang/dsip-s25.git`
   b. **GitHub Desktop:** click green “Code” button > “Open with GitHub Desktop”

You should now see a local folder named `repositoryname/`

Making changes to your repository

3. Make changes to your repository, e.g.,
   a. Create a new file called info.txt
   b. In your info.txt file, add the following two lines:
      name = “Jane Smith”
      github_name = “janesmith”

4. Follow “Typical GitHub pipeline”: in command line,

```
a. git pull                          # Retrieve latest version of repository from remote
b. git add info.txt                  # Stage changes (can add/remove multiple files)
c. git commit -m “added info”        # Commit staged changes to save a snapshot
d. git push                          # Push from local to remote repository
```

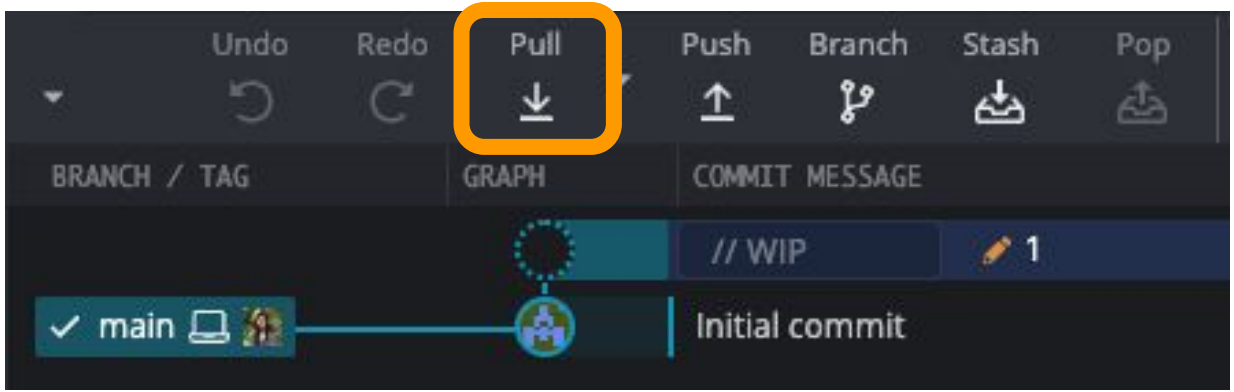
* use `git status` to see what has changed

Alternatively, can accomplish all of these steps using GUIs like GitHub Desktop or GitKraken.

Managing your repository using GitKraken

3. Open README.md and edit file
4. Follow “Typical GitHub pipeline”: in GitKraken,

```
(a) git pull    # Retrieve latest version of repository from remote
```

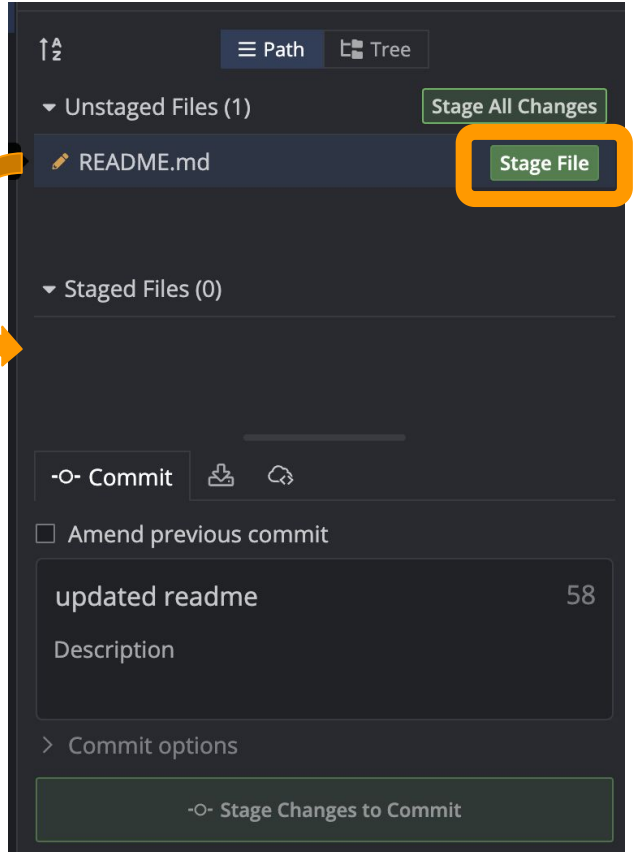


Managing your repository using GitKraken

3. Open README.md and edit file
4. Follow “Typical GitHub pipeline”: in GitKraken,

```
(b) git add README.md

# Stage changes (can add/remove multiple files)
```

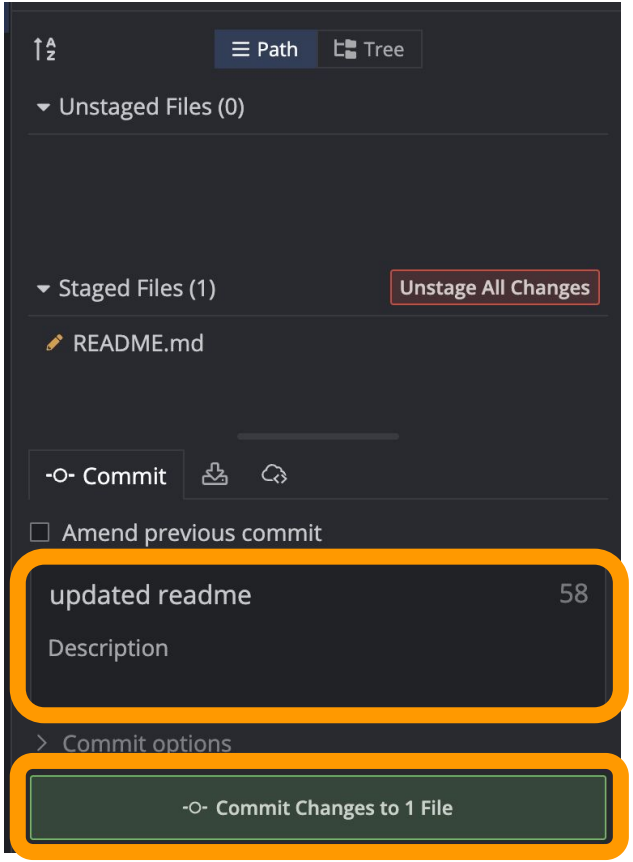


Managing your repository using GitKraken

3. Open README.md and edit file
4. Follow “Typical GitHub pipeline”: in GitKraken,

```
(c) git commit -m “updated readme”

# Commit staged changes to save a snapshot
```



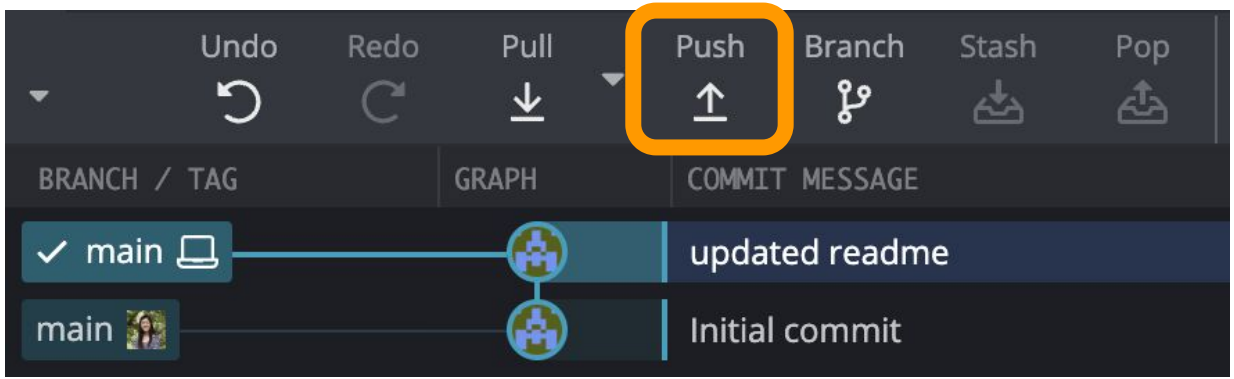
(i) add description

(ii) click commit

Managing your repository using GitKraken

3. Open README.md and edit file
4. Follow “Typical GitHub pipeline”: in GitKraken,

```
(d) git push    # Push from local to remote repository
```



Utilizing .gitignore

- Often, we do not want to track changes and push all files to GitHub
- Some types of files or folders that we usually do not want to track:
  - __pycache__/
  - .DS_Store
  - data/
- We can add these files to the .gitignore file:

```
.Rproj.user
.Rhistory
.RData
.Ruserdata
.DS_Store

__pycache__/*
data/*
```

Git Cheat Sheet

- To check the status of working directory and staging area: **git status**
- To see what is different/changed in file(s) but not staged: **git diff**
- To create a new branch at the current commit: **git branch [branch-name]**
- To switch to another branch: **git checkout**
- To save modified and staged changes (e.g., in order to change branches or pull from remote): **git stash**
- To retrieve previous stash: **git stash pop**

Full cheat sheet: https://education.github.com/git-cheat-sheet-education.pdf

Setting up your Lab 1 file structure

A Suggested Project File Structure

```
dsip/
|_ .git/
|_ lab1/     # contains all files/scripts to reproduce your lab 1 project
LICENSE
README.md
```

A Suggested Project File Structure

```
dsip/
|_ lab1/
  |_ data/                  # store all raw and processed data
  |_ notebooks/             # store all notebooks (.Rmd/.qmd/.ipynb)
     |_ lab1.qmd            # main report file
     |_ lab1.html           # rendered report file
  |_ other/                 # miscellaneous
  |_ R/ (and/or python/)    # store R/python functions
  |_ renv/                  # do not edit; created automatically by renv (R only)
  |_ results/               # store all results
  |_ scripts/               # store R/python scripts
 lab1.Rproj                 # R project (R only)
 renv.lock                  # do not edit; created automatically by renv (R only)
 environment.yml            # yml file to reproduce conda environment (python only)
 conda-lock.yml             # conda lock file to reproduce conda environment (python only)
```

Code reproducibility

Your entire analysis (including the generation of all results, figures, etc.) should be **fully and easily reproducible**.

What do I mean by **reproducible**?

- Not just that "code and seeds to reproduce analysis are available" (but it's all a mess)
- At a minimum, code, seeds, and *steps* to reproduce analysis/results are clear and documented (e.g., in your readme)
- Best case scenario and our ultimate goal:
    a. Install all packages/dependencies with one command (e.g., using renv, conda/mamba)
    b. Click a button to reproduce your final report (including the generation of all results/figures) using reproducible documentation tools (e.g., R Markdown, quarto, Jupyter notebooks)

Code reproducibility

Your entire analysis (including the generation of all results, figures, etc.) should be **fully and easily reproducible**.

What do I mean by **reproducible**?

- Not just that "code and seeds to reproduce analysis are available" (but it's all a mess)
- At a minimum, code, seeds, and *steps* to reproduce analysis/results are clear and documented (e.g., in your readme)
- Best case scenario and our ultimate goal:
  a. **Install all packages/dependencies with one command (e.g., using renv, conda/mamba)**
  b. Click a button to reproduce your final report (including the generation of all results/figures) using reproducible documentation tools (e.g., R Markdown, quarto, Jupyter notebooks)

Reproducible environments

Managing a reproducible environment

Your lab project will inevitably depend on numerous packages/dependencies.

How do we manage these dependencies in a reproducible environment?

- In R: using **renv**
- In python: using **conda/mamba** (or venv)
  a. Conda: most popular package management system
  b. Mamba: newer replacement for conda that is generally faster and better at resolving dependencies
     - Not as widely used (yet?)
  c. Most commands: simply replace conda with mamba or vice versa

Reproducible environments in R using `renv`

1. Create R project for lab1/
2. Initialize renv: `renv::init()`
    a. By default, this will initialize a environment with all discovered dependencies (typically, this includes all necessary dependencies, but some may be missed)
3. Add dependencies:
    a. Ex. `renv::install("skimr")`
4. Check status of dependencies: `renv::status()`
5. Snapshot dependencies (i.e., update lock file): `renv::snapshot()`

When I am trying to reproduce your code, I will navigate to your lab1/ folder and run: `renv::restore()`

Reproducible environments in R using `renv`

- `renv::init()`: initialize renv in a project
- `renv::status()`: check current status of environment; report inconsistencies between lockfile, library, and dependencies
- `renv::install()`: install packages
- `renv::remove()`: remove packages
- `renv::update()`: update packages
- `renv::snapshot()`: record current state of project library in the lockfile
- `renv::restore()`: restore project library from a lockfile

Full list of available functions here

Reproducible environments in python using `conda/mamba`

1. In terminal, create conda environment: `conda create --name dsip_lab1`
2. Activate environment:
    a. If working in terminal, run: `conda activate dsip_lab1`
    b. If working in VS Code, set python interpreter to `dsip_lab1` by:
        i. Open command palette (cmd/ctrl + shift + P)
        ii. Select "Python: Select Interpreter" and choose `dsip_lab1`
3. Install necessary packages, e.g.,
    a. `conda install pandas statsmodels`
    b. If running into a "No module named 'yaml'" error when rendering .qmd: `conda install jupyterlab`
4. Export conda environment:
    a. `conda env export > environment.yml` # exports entire environment (can be problematic for different os platforms)
    b. `conda env export --from-history > environment.yml` # exports explicitly installed packages only
5. Create conda lock file:
    a. Install conda lock: `conda install lock`
    b. Create lock file: `conda lock`

When I am trying to reproduce your code, I will navigate to your lab1/ folder and run: `conda-lock install --name [env_name]`

Reproducible environments in python using conda/mamba

Conda cheat sheet

- `conda create --name [env_name]`: initialize a conda environment
- `conda env remove --name [env_name]`: remove a conda environment
- `conda activate [env_name]`: activate a conda environment
- `conda deactivate`: deactivate current conda environment
- `conda install [pkg_name]`: install package in active conda environment
- `conda update [pkg_name]`: update package in active conda environment
- `conda remove [pkg_name]`: remove package from active conda environment
- `conda list`: list all packages installed in active conda environment
- `conda env export --from-history > environment.yml`: export conda environment (explicitly installed packages only) to yml
- `conda env create -f environment.yml`: create conda environment from yml

When it comes time to replicate your labs, I will...

1. Run:
   - `renv::restore()` [for R users]
   - `conda-lock install --name [env_name]` [for python users]
2. Follow the steps outlined in your `lab1/readme.md`
   - This could include how to download the data files and where to store the data. By default, I will assume the data should be placed in the `dsip/lab1/data/` folder. Note: do not push data to GitHub due to strict file size limits (100MB).
   - I will rerun your analysis locally to generate the results myself. If your analysis is too computationally expensive to run locally in a reasonable amount of time, please store and push all results necessary to reproduce your lab report. However, you should also include the code and instructions on how to generate these results.
   - Note: Be careful with your file names and make sure this is not specific to your computer. Use relative paths or `here::here()` (in R).
3. Output of this process should be your rendered report (html, pdf, ...)

Coding Style

For R users: I would recommend following the tidyverse style guide

- See part I Analyses of https://style.tidyverse.org/syntax.html#object-names
- Includes recommendations on:
  - How to name files, functions, variables
  - Maximum line length of 80 characters
  - Indenting code
  - Spacing
  - Using <- instead of =

For python users: I would recommend following the PEP-8 style guide

More importantly, BE CONSISTENT

Recap + Next Time

Recap

- **Git** is a version control system. **GitHub** hosts git repositories remotely + does more.
- **Typical GitHub workflow:** git pull, add, commit, push
- **Reproducible environments:** renv in R and conda/mamba in python

Next time

- We will get started with data cleaning + exploratory data analysis
  - Please carefully read through Tolle et al. before next class
- Hands-on practice with data cleaning + exploratory data analysis [chapters 4 and 5 from VDS textbook]