

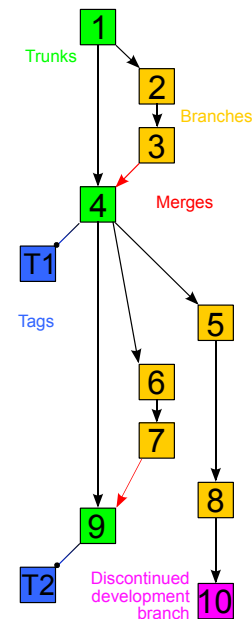
Version Control and Reproducible Research

JSC 370: Data Science II

Part I: Introduction

What is version control

[I]s the **management of changes** to documents [...] **Changes are usually identified** by a number or letter code, termed the "revision number", "revision level", or simply "revision". For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on. **Each revision is associated with a timestamp and the person making the change.** Revisions can be **compared, restored**, and with some types of files, **merged**. -- [Wiki](#)



Why do we care

Have you ever:

- Made a **change to code**, realised it was a **mistake** and wanted to **revert** back?
- **Lost code** or had a backup that was too old?
- Had to **maintain multiple versions** of a product?
- Wanted to see the **difference between** two (or more) **versions** of your code?
- Wanted to prove that a particular **change broke or fixed** a piece of code?
- Wanted to **review the history** of some code?

Why do we care (cont'd)

- Wanted to submit a **change** to **someone else's code**?
- Wanted to **share your code**, or let other people work on your code?
- Wanted to see **how much work** is being done, and where, when and by whom?
- Wanted to **experiment** with a new feature **without interfering** with working code?

In these cases, and no doubt others, a version control system should make your life easier.

-- [Stackoverflow](#) (by [si618](#))

Git: The stupid content tracker



Git logo and Linus Torvalds, creator of git

- During this class (and perhaps, the entire program) we will be using [Git](#).
- Git is used by [most developers in the world](#).
- A great reference about the tool can be found [here](#)
- More on what's stupid about git [here](#).

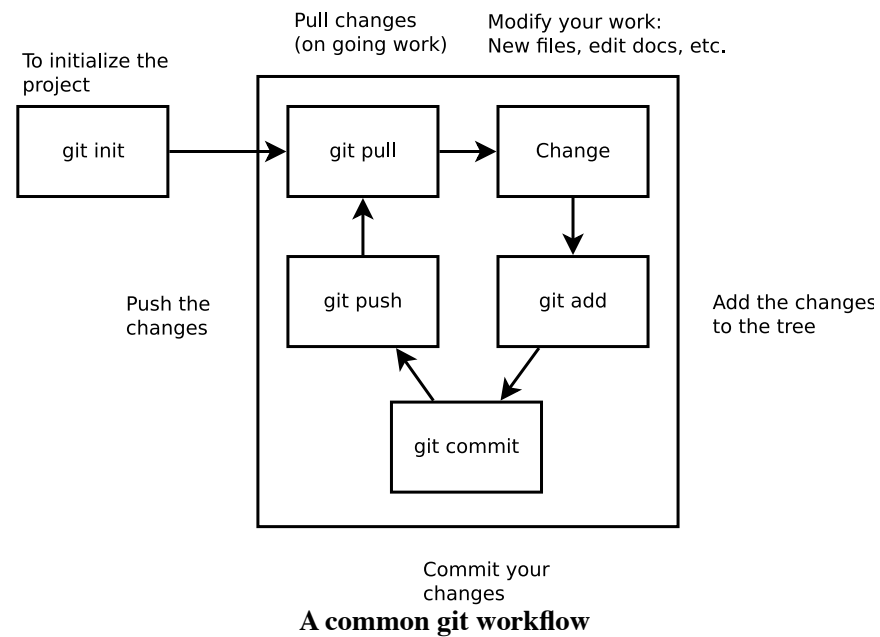
How can I use Git

There are several ways to include Git in your work-pipeline. A few are:

- Through command line
- Through one of the available Git GUIs:
 - RStudio ([link](#))
 - Git-Cola ([link](#))
 - Github Desktop ([Link](#))

More alternatives [here](#).

A Common workflow



A Common workflow

1. Start the session by pulling (possible) updates: `git pull`
2. Make changes
 - a. (optional) Add untracked (possibly new) files: `git add [target file]`
 - b. (optional) Stage tracked files that were modified: `git add [target file]`
 - c. (optional) Revert changes on a file: `git checkout [target file]`
3. Move changes to the staging area (optional): `git add`
4. Commit:
 - a. If nothing pending: `git commit -m "Your comments go here."`
 - b. If modifications not staged: `git commit -a -m "Your comments go here."`
5. Upload the commit to the remote repo: `git push`.

Part 2: Hands-on local git repo

Hands-on 0: Introduce yourself

Set up your git install with `git config`, start by telling who you are

```
$ git config --global user.name "Meredith Franklin"  
$ git config --global user.email "mfranklin@email.com"
```

If you have already set up git previously, you can check your settings

```
$ git config --list
```

(to get out of the list in terminal, press q)

Try it yourself (5 minutes) (more on how to configure git [here](#))

Hands-on 1: Remote repository

We will start by working on our very first project. To do so, you are required to start using Git and Github so you can share your code with your team. For this exercise, you need to

- a. Create a new (empty) repository on GitHub (you can try JSC370). Make sure to include a README.md (checkbox)
- b. Go to the local directory where you want to store the files for this repo.
- c. Clone the repository (in GitHub copy the repo link) `git clone https://github.com/...`
- d. Back in terminal, edit the README.md. You can use nano in the terminal or open in another app such as RStudio or SublimeText.
- e. Add the edited README.md file to the tree using the `git add` command, and check the status.
- f. Make the first commit using the `git commit` command adding a message, e.g.

```
$ git commit -m "My first commit ever!"
```

And use `git log` to see the history.

Note: We are assuming that you already [installed git in your system](#).

Hands-on 1: Remote repository

The following code is fully executable (copy-pastable)

```
# (a) Creating the folder for the project (and getting in there)
mkdir ~/JSC370
cd ~/JSC370

# (b) Initializing git, creating a file, and adding the file
git init

# (c) Creating the Readme file
echo An empty line > README.md

# (d) Adding the file to the tree
git add README.md
git status

# (e) Committing and checkout out the history
git commit -m "My first commit ever!"
git log
```

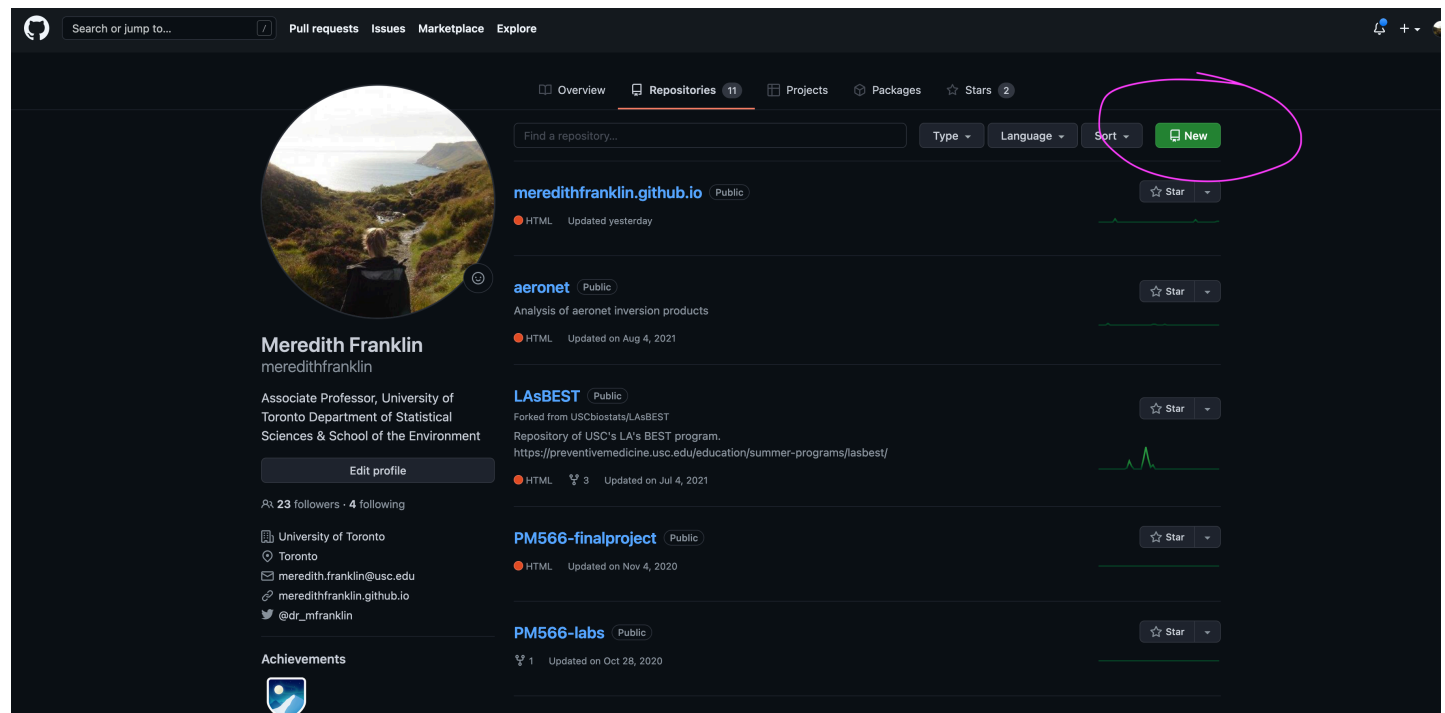
Hands-on 1: Remote repository

If you add a wrong file to the tree, you can remove files from the tree using `git rm --cached`, for example, imagine that you added the file `class-notes.docx` (which you are not supposed to track), then you can remove it using

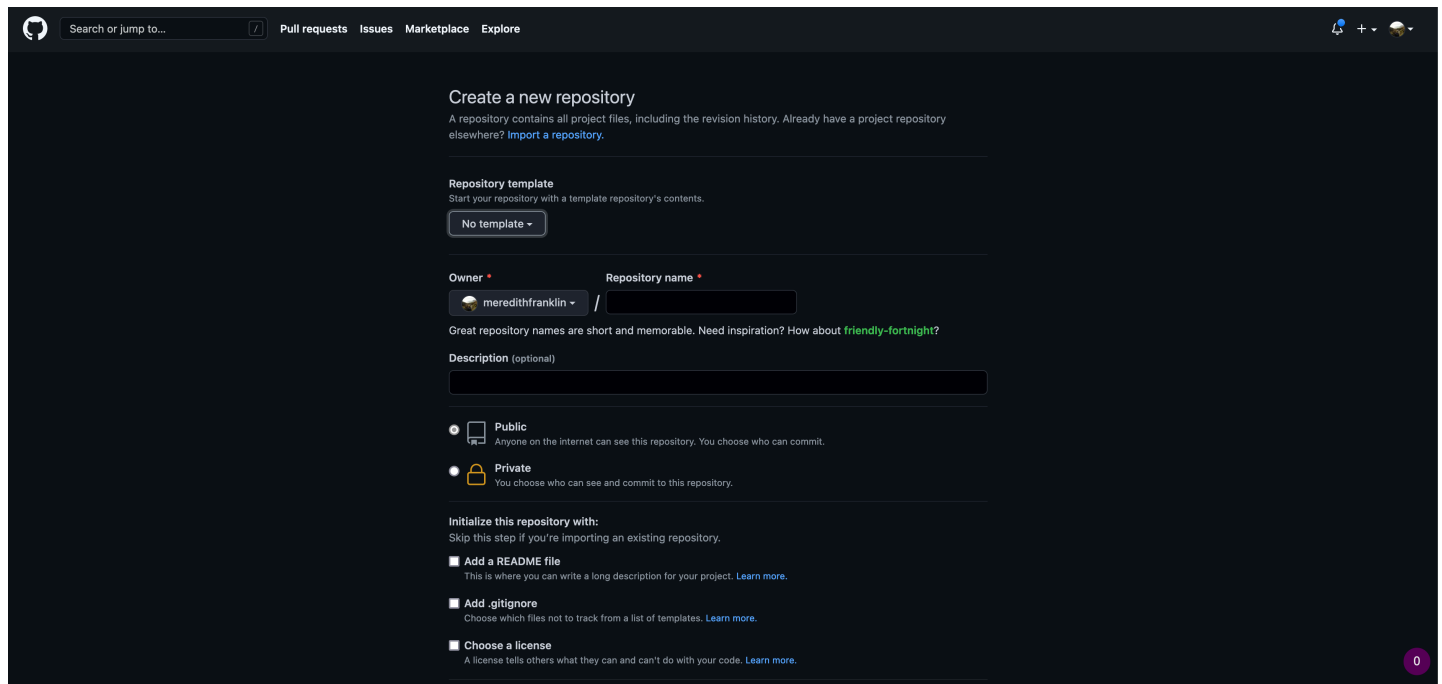
```
$ git rm --cached class-notes.docx
```

This will remove the file from the tree **but not from your computer**. You can go further and ask git to avoid adding docx files using the [.gitignore file](#)

Hands-on 1: Remote repository



Hands-on 1: Remote repository



The screenshot shows the GitHub 'Create a new repository' page. At the top, there's a navigation bar with the GitHub logo, a search bar, and links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The main heading is 'Create a new repository', followed by a subtext: 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'

Below this is the 'Repository template' section, which says 'Start your repository with a template repository's contents.' and has a button 'No template -'.

The next section is for repository details. It has two fields: 'Owner' (with a dropdown menu showing 'meredithfranklin') and 'Repository name' (with a text input field). Below these fields is a note: 'Great repository names are short and memorable. Need inspiration? How about [friendly-fortnight?](#)'

There's a 'Description (optional)' text input field.

Below the description is a section for visibility. It has two radio buttons: 'Public' (selected) and 'Private'. The 'Public' option is described as 'Anyone on the internet can see this repository. You choose who can commit.' The 'Private' option is described as 'You choose who can see and commit to this repository.'

At the bottom, there's a section 'Initialize this repository with:' which says 'Skip this step if you're importing an existing repository.' It has three checkboxes: 'Add a README file' (with a note 'This is where you can write a long description for your project. [Learn more.](#)'), 'Add .gitignore' (with a note 'Choose which files not to track from a list of templates. [Learn more.](#)'), and 'Choose a license' (with a note 'A license tells others what they can and can't do with your code. [Learn more.](#)').

In the bottom right corner, there's a purple circle with the number '0'.

Example for .gitignore

Example extracted directly from Pro-Git [\(link\)](#).

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODD
/TODD

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

Resources

- Git's everyday commands, type `man giteveryday` in your terminal/command line. and the very nice [cheatsheet](#).
- My personal choice for nightstand book: The Pro-git book (free online) [\(link\)](#)
- Github's website of resources [\(link\)](#)
- The "Happy Git with R" book [\(link\)](#)
- Roger Peng's Mastering Software Development Book Section 3.9 Version control and Github [\(link\)](#)
- Git exercises by Wojciech Frącz and Jacek Dajda [\(link\)](#)
- Checkout GitHub's Training YouTube Channel [\(link\)](#)