

Version Control

IMPROVING REPRODUCIBILITY IN DATA SCIENCE

The Meaning of Reproducibility

Project structure

- Clear organization of analytic components

Version control

- **Controlled tracking of changes over time**
- **Ability to update & iterate while maintaining functioning code**

Coding standards

- Commenting
- Simplification
- Modularity

Documentation

- Providing context to code, both within code (e.g., commenting) and via external documents (e.g., README.md)

Virtual environments

- Enables reproducibility across multiple machines post-distribution

Code testing

- Small, modular testing of segments of code combined w/ large testing upon modification

The Importance of Version Control

Prevents programmer from losing track of what's what

- Multiple versions of code
- Multiple “final” versions of code

Enables updates without breaking functioning code

- Enables reversion to older iterations, just in case


Documents the history of changes made for tracking purposes


Significantly improves collaboration amongst teams writing programs


- Though the workflow can be difficult to pick up


Enables simpler & more effective dissemination


- Point people to your GitHub to share code


 code.py

 code_v1.py

 code_v2.py

 code_vfinal.py

 code_vfinal_final.py

 code_vfinal_final_srsly.py

Introduction to Git



Most widely-used version control system

- Completely open-source system developed by [Linus Torvalds](#) (who also created Linux) in 2005

What does Git do?

- Manage coding projects using repositories
- Controls & tracks changing w/ staging & committing
- **Enables code revision & updates w/o the risk of breaking code by branching & merging**
- Provides a local working copy via cloning (not like working on a Google doc, for example)



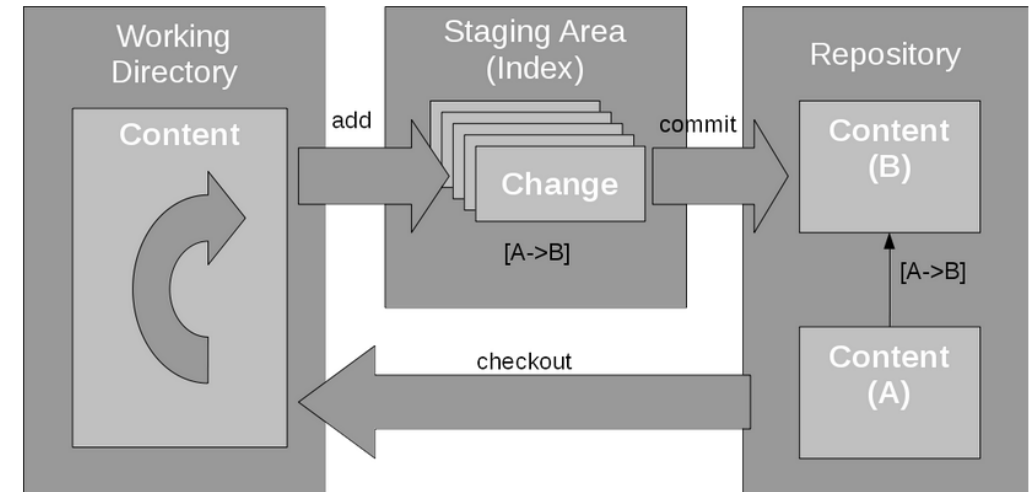
...but how does Git work?

Three primary components:

- **Working directory** – Where changes are made
- **Staging area** – Where changes are indexed (accounted for) & prepared for application
- **Repository** – Where changes are applied

It's all hidden in the .git directory...

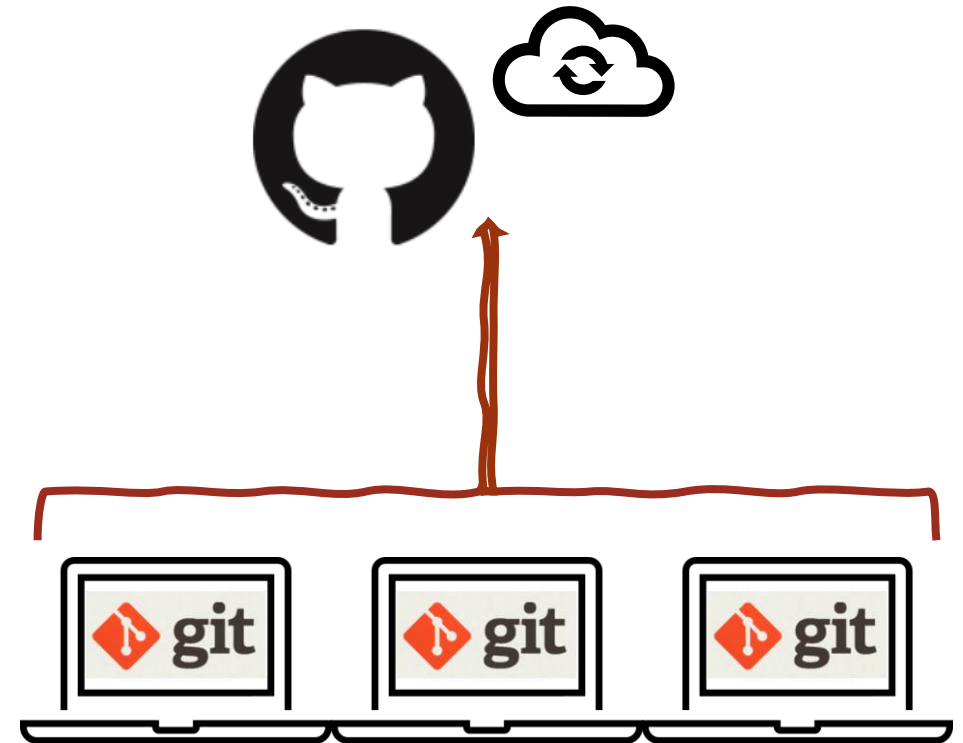
- objects/ stores snapshot ("commit") metadata
- refs/ points to commits
- /index stores pointers to commits (e.g., branches)
- /HEAD points to current branch



Git or GitHub?

GitHub is a centralized, online hub for hosting & managing projects using Git

- In a sense, provides a visual representation of what Git does, which enables simpler hosting, collaborating, & sharing





Search or jump to...



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)



[curtismartin](#) / **Courses** **Private** Repo access type

[Unwatch](#) 1

[Fork](#) 0

[Star](#) 0

Current repo – one per project

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#) [Settings](#)

Current branch – master is default

master 1 branch 0 tags

[Go to file](#)

[Add file](#)

[Code](#)

About



Course materials which have been developed over time...

[Readme](#)

0 stars

1 watching

0 forks

	Update directory tree	386a394 11 hours ago 6 commits
	Working	Update directory tree 11 hours ago
	.gitignore	Update directory tree 11 hours ago
	README.md	Start building version control lesson 11 hours ago

Code & related information in current branch

[README.md](#)

[Courses](#)

Packages together materials I've put together for various data science curriculums over the years.

Releases

No releases published
[Create a new release](#)

0

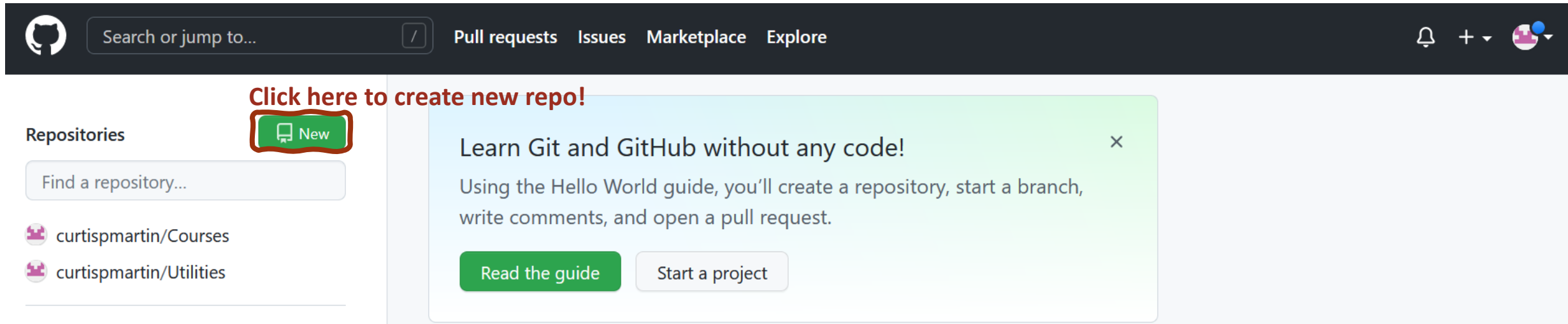
Setting up Git/Hub

First things first: Download necessary software & create accounts

- Git: [Download Git for Windows here](#); download Git for Mac here?
- GitHub: [Create an account here](#)

Create your first repo on GitHub

- Call it “XXX XXX”



The screenshot shows the GitHub homepage. At the top is a dark navigation bar with the GitHub logo, a search bar, and links for Pull requests, Issues, Marketplace, and Explore. On the left, under the 'Repositories' section, there is a search bar and a list of repositories including 'curtispmartin/Courses' and 'curtispmartin/Utilities'. A red box highlights the green 'New' button in the 'Repositories' section. A red text annotation 'Click here to create new repo!' points to this button. A light green notification box on the right contains the text 'Learn Git and GitHub without any code!' and two buttons: 'Read the guide' and 'Start a project'.

Click here to create new repo!

Repositories

Find a repository...

curtispmartin/Courses

curtispmartin/Utilities

Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

Read the guide Start a project

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Name your repo

Owner *



curtismartin ▾

Repository name *

add_repo_name_here



Great repository names are short and memorable. Need inspiration? How about [reimagined-octo-robot?](#)

Description (optional)

This repo is for this course I'm taking...



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Make repo private for this course

Initialize this repository with:

Skip this step if you're importing an existing repository.



Add a README file

This is where you can write a long description for your project. [Learn more.](#)

This file will help you describe what the repo is for, as well as how it works



Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

This file tells Git what types of files should not be added to staging/repo... Set "template" to Python

⚙ General

Access

👤 **Collaborators**

Code and automation

🔗 Branches

▶ Actions

🔗 Webhooks

📄 Pages

Security

🛡 Security and analysis

Who has access

PRIVATE REPOSITORY 🔒

Only those with access to this repository can view it.

[Manage](#)

DIRECT ACCESS 👤

0 collaborators have access to this repository. Only you can contribute to this repository.

Manage access



You haven't invited any collaborators yet

Add people

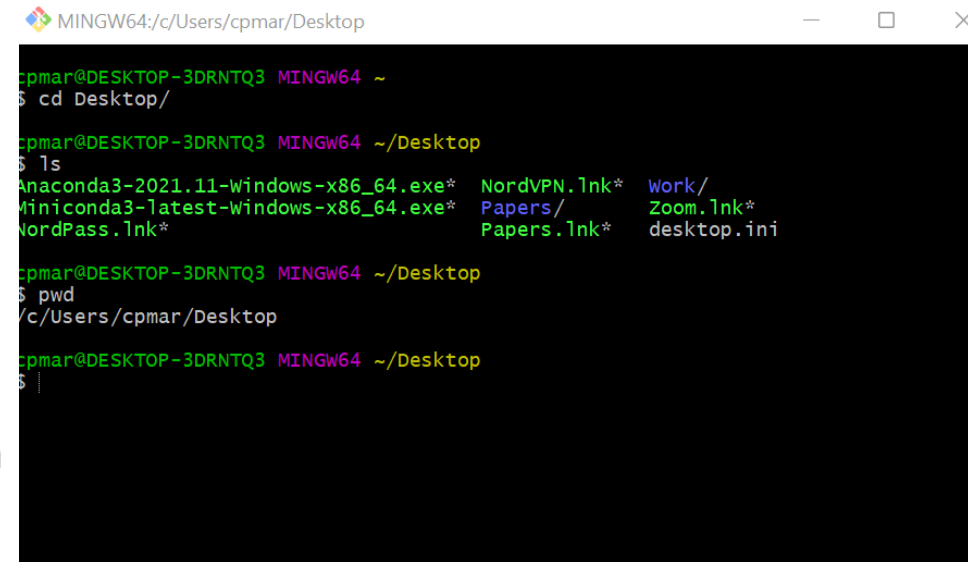
**Go to Settings -->
Collaborators & add
me as a Collaborator**

Setting up Git/Hub

Navigate to your Git bash... A few important commands are needed!

- **ls** – List what's in your current working directory ("CWD")
- **cd** <filepath> – Change CWD to that specified by filepath
- **mkdir** <newdirectory> – Make new directory (folder) in CWD
- **pwd** – Print path to CWD
- **mv** <filename> <newfilepath> – Move file in CWD to new location
- **cp** <filename> <newfilepath> – Copy file in CWD to new location
- **cat** <filename> – Print out file contents in bash
- **touch** <filename> – Create empty file w/ file name in CWD

You can [find more commands here](#)



```
MINGW64:/c/Users/cpmar/Desktop

cpmar@DESKTOP-3DRNTQ3 MINGW64 ~
$ cd Desktop/

cpmar@DESKTOP-3DRNTQ3 MINGW64 ~/Desktop
$ ls
Anaconda3-2021.11-Windows-x86_64.exe*  NordVPN.lnk*  work/
Miniconda3-latest-Windows-x86_64.exe*  Papers/       Zoom.lnk*
NordPass.lnk*                        Papers.lnk*   desktop.ini

cpmar@DESKTOP-3DRNTQ3 MINGW64 ~/Desktop
$ pwd
/c/Users/cpmar/Desktop

cpmar@DESKTOP-3DRNTQ3 MINGW64 ~/Desktop
$
```

Note: In this course, angle brackets (<>) indicate user input required

Connect Git to GitHub

Must authenticate Git client w/ your GitHub

- SSH (recommended) – [More on SSH here](#)
- HTTPS

Step 1: Create SSH key using Git bash

- ``ssh-keygen -t ed25519 -C <your_email_address>``
- Accept default location for key
- Enter & re-enter key passphrase for enhanced security

Step 2: Add SSH key to ssh-agent

- ``eval $(ssh-agent -s)``
- ``ssh-add ~/.ssh/id_ed25519``

Step 3: Add SSH key to Github account

- ``clip < ~/.ssh/id_ed25519.pub``
- Log into Github
- Go to settings > SSH and GPG keys
- Click New SSH key
- Paste into “Key” text box
- Add “Title” (name of computer you’re using)
- Click “Add SSH key”

Note: In general, backticks (`) indicate commands or lines of code

Cloning your Git Repo

Last thing before we start using Git:
You need to clone your repo onto
your computer

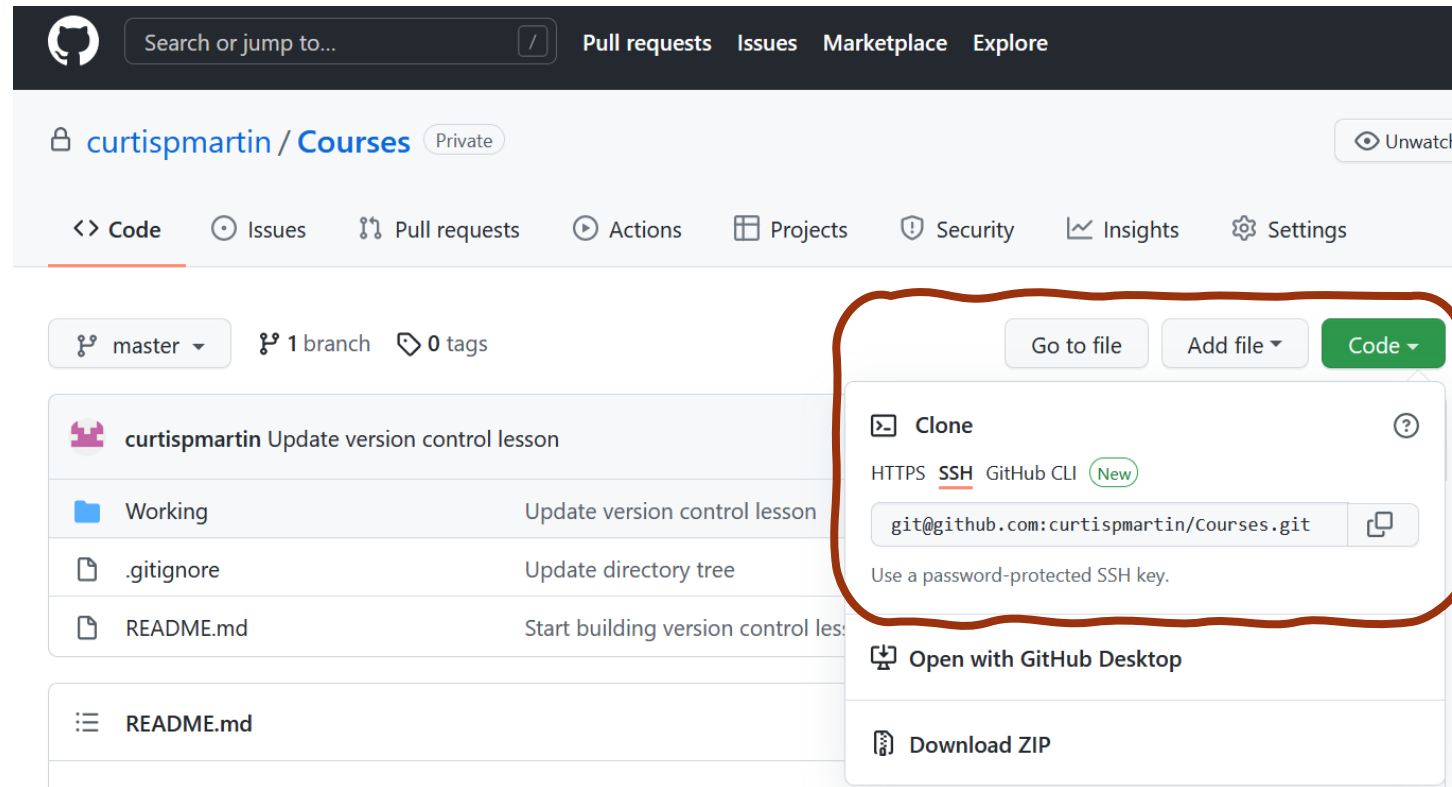
- Makes a local copy of your repo so you can write/update/manage code

Step 1: Copy SSH link

- In repo, go to Code > SSH

Step 2: Clone to location of choice

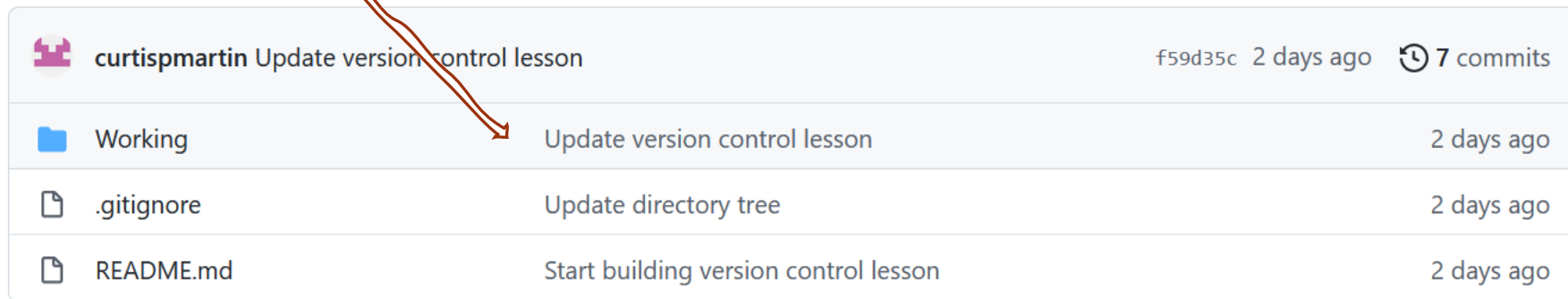
- ``cd <path_to_location>``
- ``git clone <SSH link>``








Git: Basic Workflow

Five basic commands to start:

- ``git status`` – Gives you a status report for each file
- ``git add <filename>`` – Adds file to staging... use ``*`` to add all files
- ``git commit -m <message>`` – Commits changes in staging area; message should briefly summarize the point of the update



 curtismartin Update version control lesson	f59d35c 2 days ago	 7 commits
 Working	Update version control lesson	2 days ago
 .gitignore	Update directory tree	2 days ago
 README.md	Start building version control lesson	2 days ago

- ``git push`` – Send commit to repo
- ``git pull`` – Pull most recent commit from repo (only necessary for collaborative work)

Git: Branching

What is a branch?

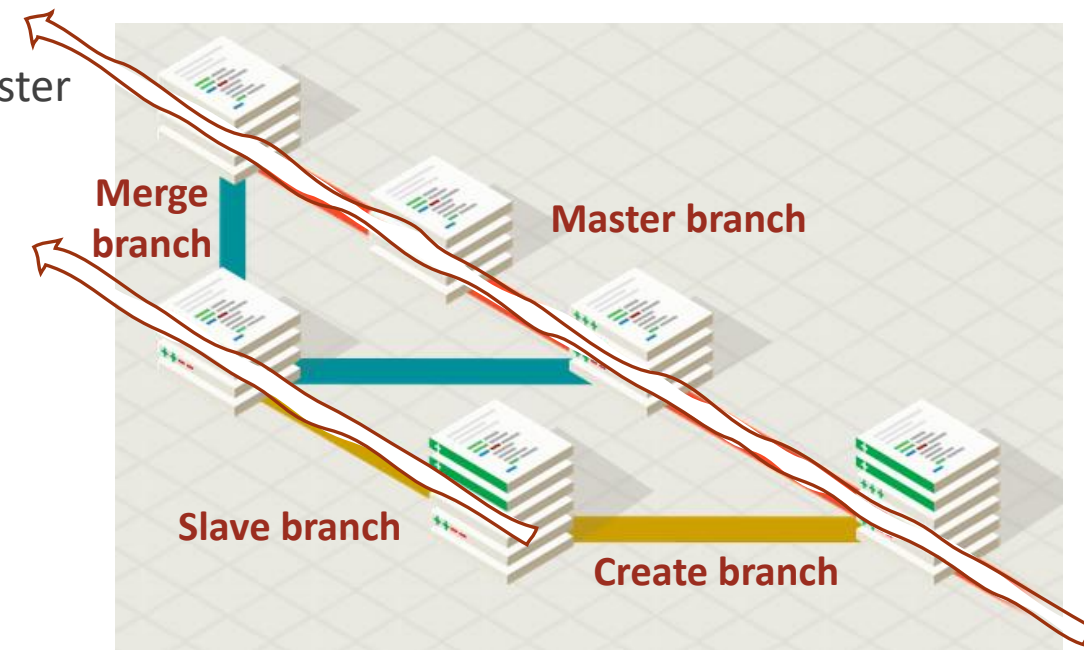
- Branches are essentially tracked copies of the master
- Modifications in a branch aren't incorporated into the master until explicitly merged

Why branching?

- Allows you to modify code w/o the risk of breaking it
- Simplifies (kinda) collaboration on projects

When to create a branch?

- Developing a new feature
- Making significant updates to current code



Git: Branching

Step 1: Create the branch

- Create new branch: ``git checkout -b <branchname>``
- Push branch to repo: ``git push origin -u <branchname>``

Step 2: Do work in branch

- Switch to branch: ``git checkout <branchname>``
- Make your updates; add, commit, push to repo as if nothing has changed

Step 3: Merge branch

- Once feature/update is complete, checkout master: ``git checkout master``
- Then merge branch w/ master: ``git merge <branchname>``
- Delete branch using: ``git branch -d <branchname>``
- Push deletion to repo: ``git push origin -d <branchname>``

Preparing for Success

Directory structure

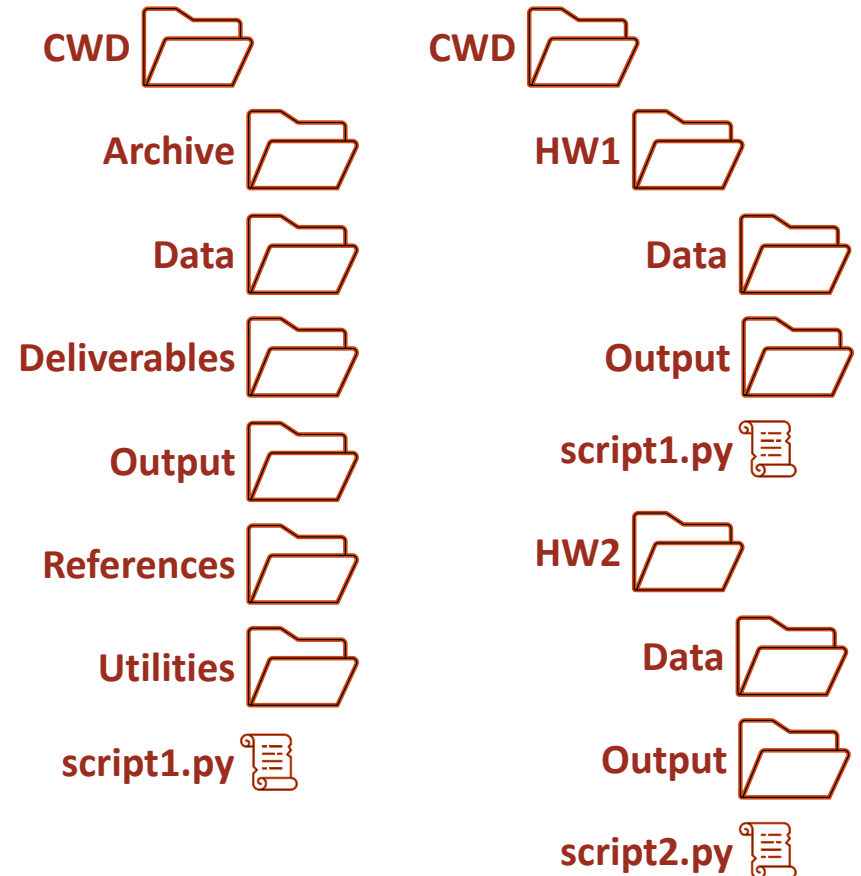
- Create a system for organizing code & related files
- Stick to it (but don't be afraid to adapt)!
- Highly dependent on the type of work you do

Internal documentation

- Header detailing what the code does, inputs/outputs (I/O), etc.
- Comments describing what blocks of code do

External documentation

- **“README.md” in Git repo – What is intent? How do files relate?**
- User manual – Not relevant here, but could be in future



“README.md”

README files are found at top level of repo and inside each analysis folder

- [Markdown](#) is used to create documentation

🔗 Lesson 1: Introduction to Data Science

Here, we provide a broad overview of the content covered in data science, as well as what will be tackled in this course. We also talk about setting up your data science environment. In this course, we'll be using:

- [Miniconda](#)

In preparation for next lecture, go ahead & setup your GitHub account & install Git Bash if needed:

- [GitHub](#)
- [Git Bash](#) for those on Windows
- ...for those on Mac?

🔗 Lesson 2: Version Control

Here, we introduce version control as a best practice when writing & developing your code. We talk about standard problems in data science when it comes to reproducibility & where version control fits in. We illustrate how Git & GitHub help solve these problems. We discuss best practices in using Git & GitHub. We end by covering coding in a collaborative setting.

The simple things

- Title headers are created at three levels using # (largest), ## (second largest), & ### (smallest)
- Text not preceded by # will become body text
- Asterisks or dashes to create bulleted lists
- Inline code: `this will show up as code`

```
### `Lesson 1: Introduction to Data Science`  
Here, we provide a broad overview of the content covered in data science, as well as what will be tackled in this course.  
We also talk about setting up your data science environment. In this course, we'll be using:  
- [Miniconda](https://docs.conda.io/en/latest/miniconda.html)
```

```
In preparation for next lecture, go ahead & setup your GitHub account & install Git Bash if needed:  
- [GitHub](https://github.com/) for those on Windows  
- [Git Bash](https://gitforwindows.org/) for those on Windows  
- ...for those on Mac?
```

```
### [Lesson 2: Version Control](https://github.com/curtismartin/Courses/tree/master/Working/2_VersionControl)  
Here, we introduce version control as a best practice when writing & developing your code.  
We talk about standard problems in data science when it comes to reproducibility & where version control fits in.  
We illustrate how Git & GitHub help solve these problems.  
We discuss best practices in using Git & GitHub.  
We end by covering coding in a collaborative setting.
```

“.gitignore”

“.gitignore” file is how you tell repo what not to track

- By default, repos track all files

Tend to want to ignore either specific file types... (**/*.<filetype>)

- Data – .csv, .xlsx, .pkl, .Rdata, etc.
- Output – .png, .jpg, .gif, .tif
- Other – .docx, .pdf, .pptx (unless you’re teaching a class...)

...or we want to ignore specific directories (directory/* or **/directory/)

- Want to put administrative stuff on GitHub?
- Want to put sandboxes up?
- What about old stuff which is no longer relevant?

```
**/*.docx  
**/*.gif  
**/*.html  
**/*.jpg  
**/*.png  
**/*.pptx  
**/*.xls  
**/*.xlsb  
**/*.xlsx
```

```
__pycache__/*  
Admin/*  
Archive/*  
Output/*  
Resources/*
```

```
**/Figures/  
**/Internal/  
**/.ipynb_checkpoints/
```

Git: Collaborative Coding

The process for using Git/Hub is largely the same, w/ (at least) one big exception: [Pull requests](#)

The workflow:

- You create a new branch to start building out some new functionality
- You write & finish the new functionality
- You stage & commit the new code to the repo
- **You then submit a pull request which tells your collaborators the code is ready for merging**
- **This prompts your teammates (or at least the owner of the repo) to review new code before merging**
- **Once agreeable, you or a teammate can then accept the request, merge the branches, then `git pull`**

The screenshot shows the GitHub interface for the repository 'curtispmartin / Education', which is marked as 'Private'. The top navigation bar includes links for 'Code', 'Issues', 'Pull requests' (which is the active tab), 'Actions', 'Projects', 'Security', 'Insights', and 'Settings'. On the right side of the repository header, there are buttons for 'Unwatch' (1), 'Fork' (0), and 'Star' (0). Below the navigation bar, there is a search bar with the text 'is:pr is:open' and a 'Filters' dropdown. To the right of the search bar are buttons for 'Labels' (9) and 'Milestones' (0). A green button labeled 'New pull request' is located on the far right.

In-Class Activity...

Set up Git/Hub on your environment

- Download & install Git
- Set up GitHub account
- Create SSH key on local machine
- Link Git to GitHub using SSH key
- Make your repo private

Create your own repo for this course called
“XXX XXX”

- Add me & TAs as collaborators
- Confirm local copy & connection to GitHub

Create a subdirectory in repo called
“2_VersionControl”

- In it, create a Jupyter notebook (“inclass.ipynb”) & copy over code from file w/ same name

Stage & commit your repo

Create new branch called “newname”

- Open “inclass.ipynb” using Jupyter
- Change ‘myname’ to your name
- Push new branch to GitHub

Once functional, merge “newname” w/
“master” branch, then stage & commit

Til next time...

HOMEWORK?