



CORE | Skills

Prerequisite Day 1: Introduction to the tools and concepts

Let's see if our Anaconda installation worked

- Open the **Anaconda Navigator**:
 - <https://docs.anaconda.com/anaconda/navigator/getting-started/>
- Open **JupyterLab**:
 - Use Anaconda navigator to open a jupyter lab
 - If you have the tools you can use the command line: `jupyter notebook`
- Try opening a new notebook using a Python 3 kernel
- Try changing your browser if you have problems

If all this worked for you we are all set, close down the notebook and log out of Jupyter.

Advanced step:

- This can be done via the Anaconda launcher or the command line
- Create an environment for the workshop
 - Navigator:
<https://docs.anaconda.com/anaconda/navigator/getting-started/#managing-environments>
 - Command line:
<https://conda.io/docs/user-guide/tasks/manage-environments.html>
- Start the environment

HELLO

my name is

REBECCA

KATHRYN

CARA

Clone an existing Github Repo

CORE | Skills

- Go to: <https://github.com/core-skills/00-Prerequisite>

The screenshot shows the GitHub repository page for 'core-skills / 00-Prerequisite'. At the top, there are buttons for 'Watch' (2), 'Star' (0), and 'Fork' (3). Below the header, there are tabs for 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Security', and 'Insights'. A prominent 'Join GitHub today' modal is displayed, featuring a 'Sign up' button. The repository summary below the modal includes metrics: 15 commits, 1 branch, 0 packages, 0 releases, 2 contributors, and MIT license. The commit history lists several commits by user 'r-lange', including 'added solutions notebook', 'tidy repo', 'added solutions notebook', 'Initial commit', 'added binder badge', and 'Adding environment file', all dated 17 months ago. At the bottom right, there are 'Find file' and 'Clone or download' buttons.

core-skills / 00-Prerequisite

Watch 2 Star 0 Fork 3

Code Issues 0 Pull requests 0 Projects 0 Security Insights

Join GitHub today Dismiss

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

Sign up

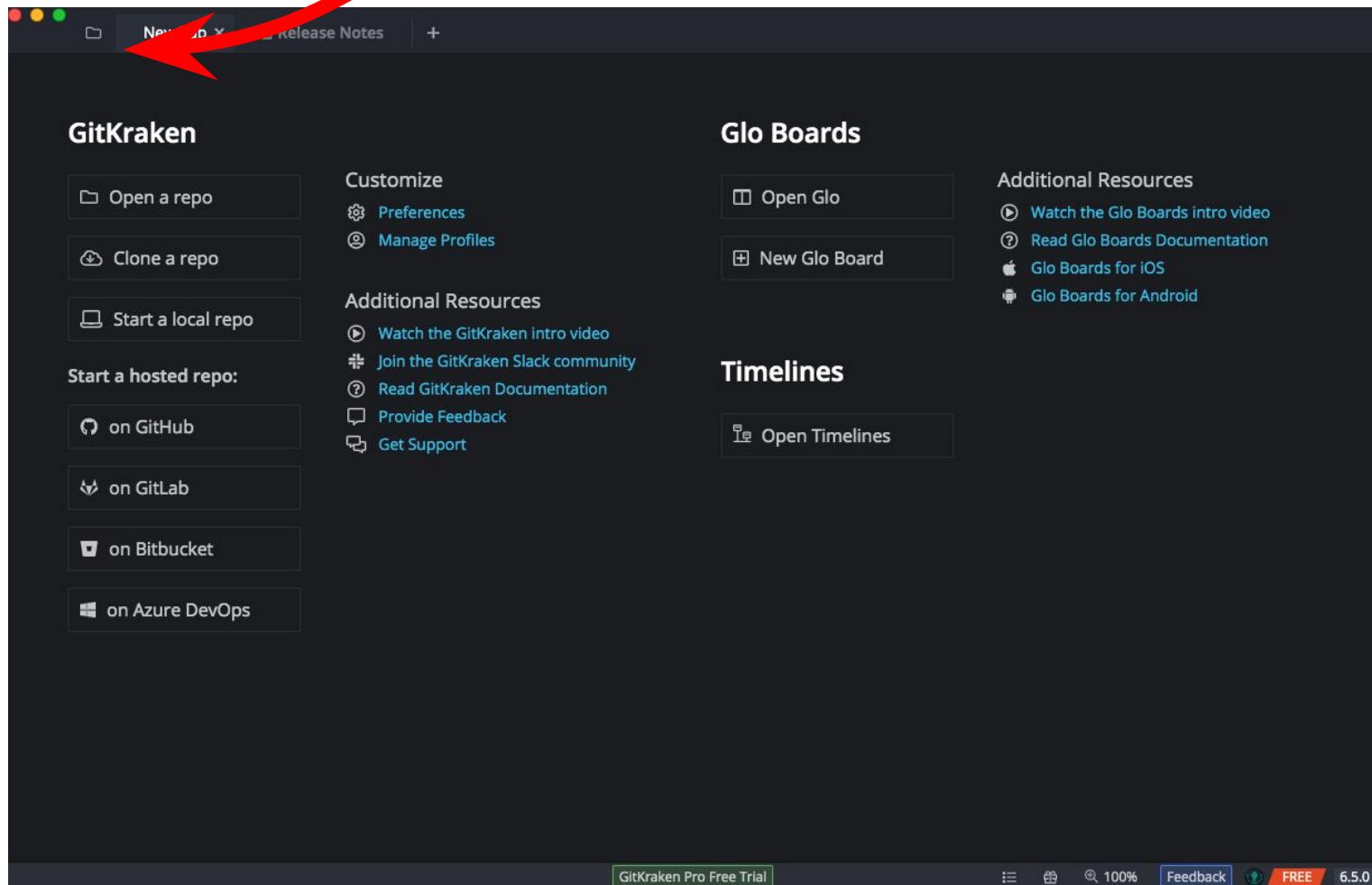
Core Skills week 0 - Prerequisite

15 commits 1 branch 0 packages 0 releases 2 contributors MIT

Branch: master ▾ New pull request Find file Clone or download ▾

File	Description	Date
r-lange added solutions notebook	Latest commit 5aa1f67 on 12 Sep 2018	
data	tidy repo	17 months ago
handouts	tidy repo	17 months ago
notebooks	added solutions notebook	17 months ago
LICENSE	Initial commit	17 months ago
README.md	added binder badge	17 months ago
environment.yml	Adding environment file	17 months ago

In GitKraken, click on the 'Folder' Icon

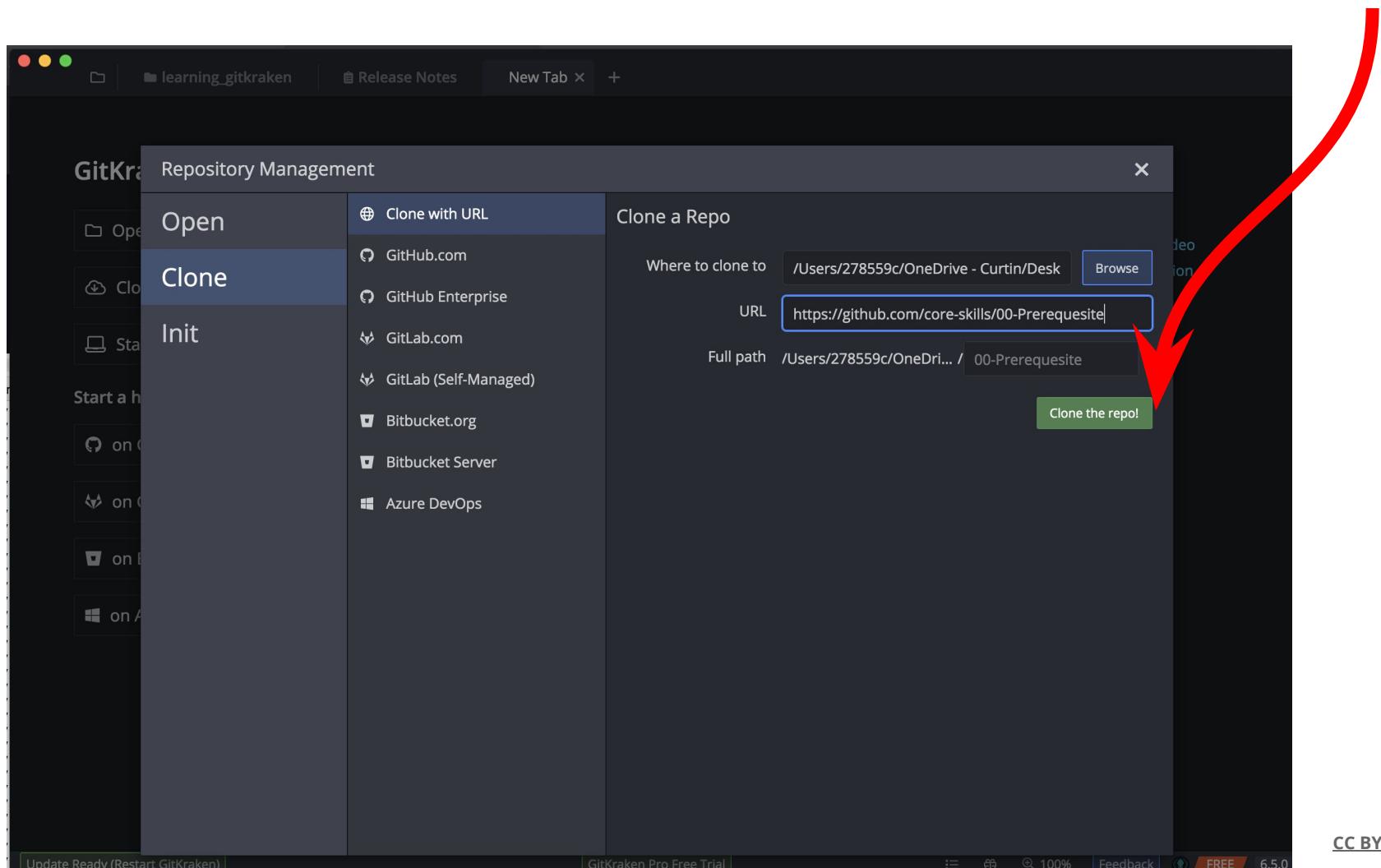


Clone an existing Github Repo

CORE | Skills

Click 'Clone', and 'Clone with URL', choose where to clone it to (in your workshop folder), enter the URL

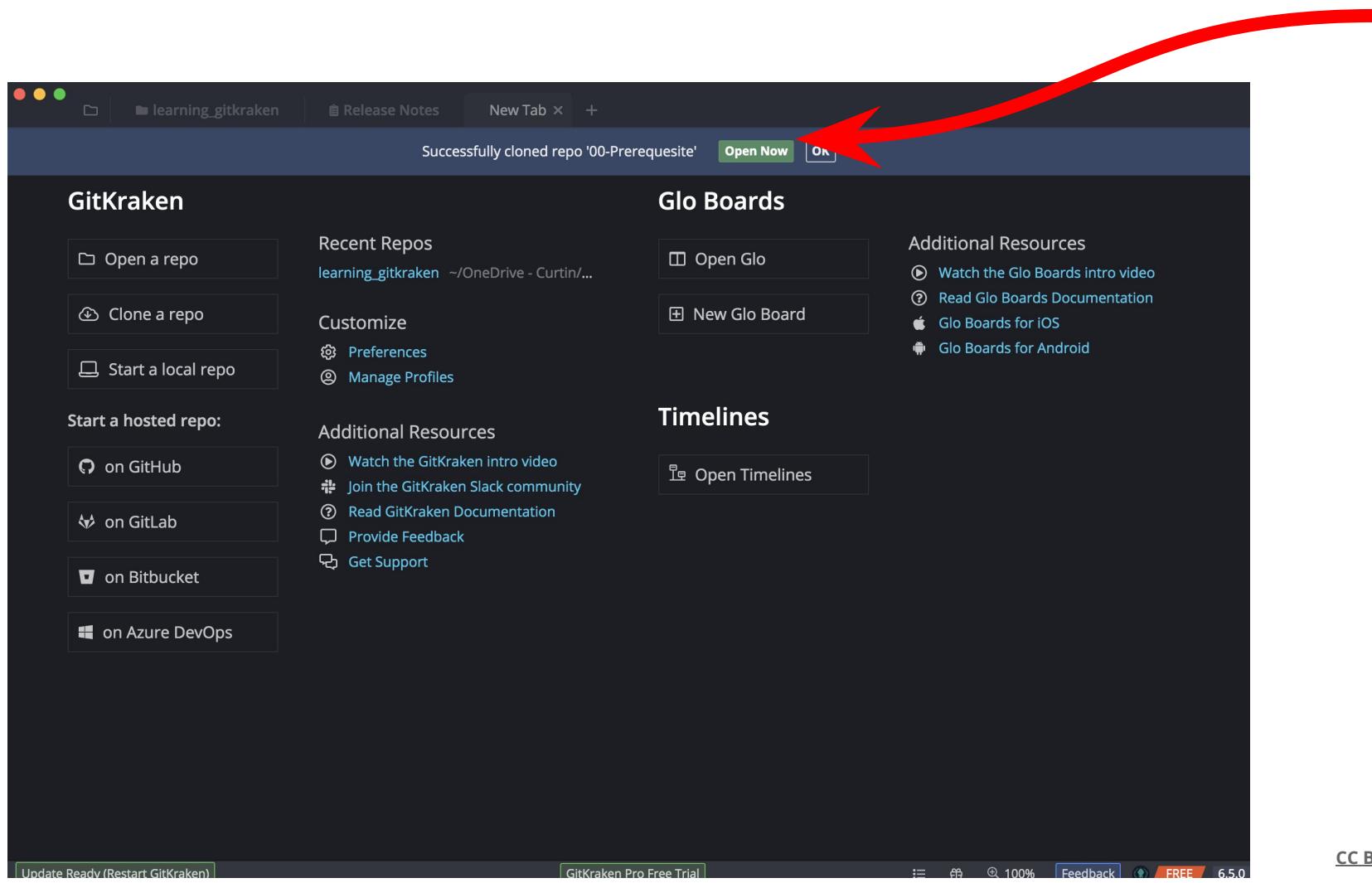
<https://github.com/core-skills/00-Prerequisite> and click 'Clone the repo!'



Clone an existing Github Repo

CORE | Skills

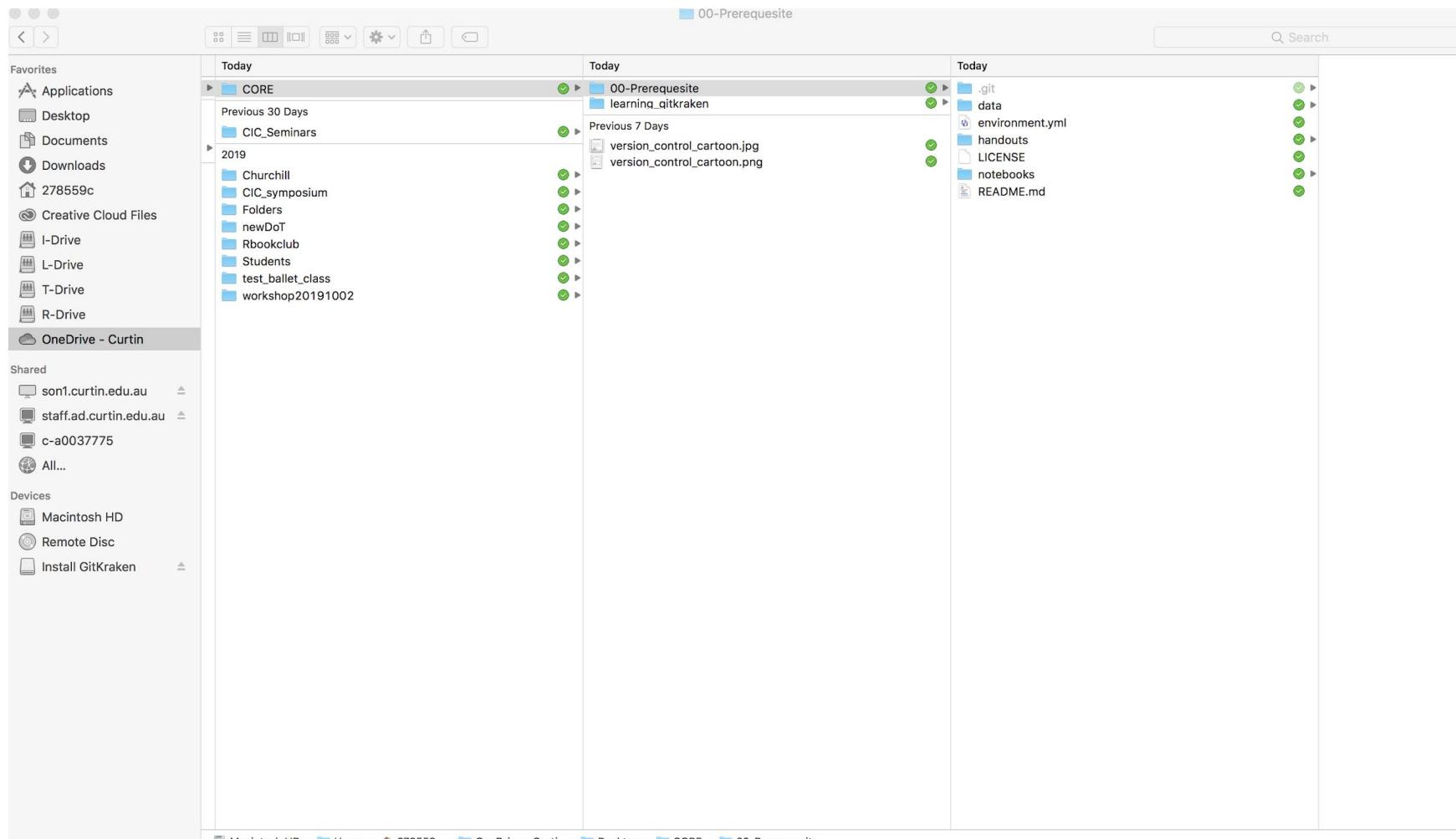
Click 'Open Now' to view the repository



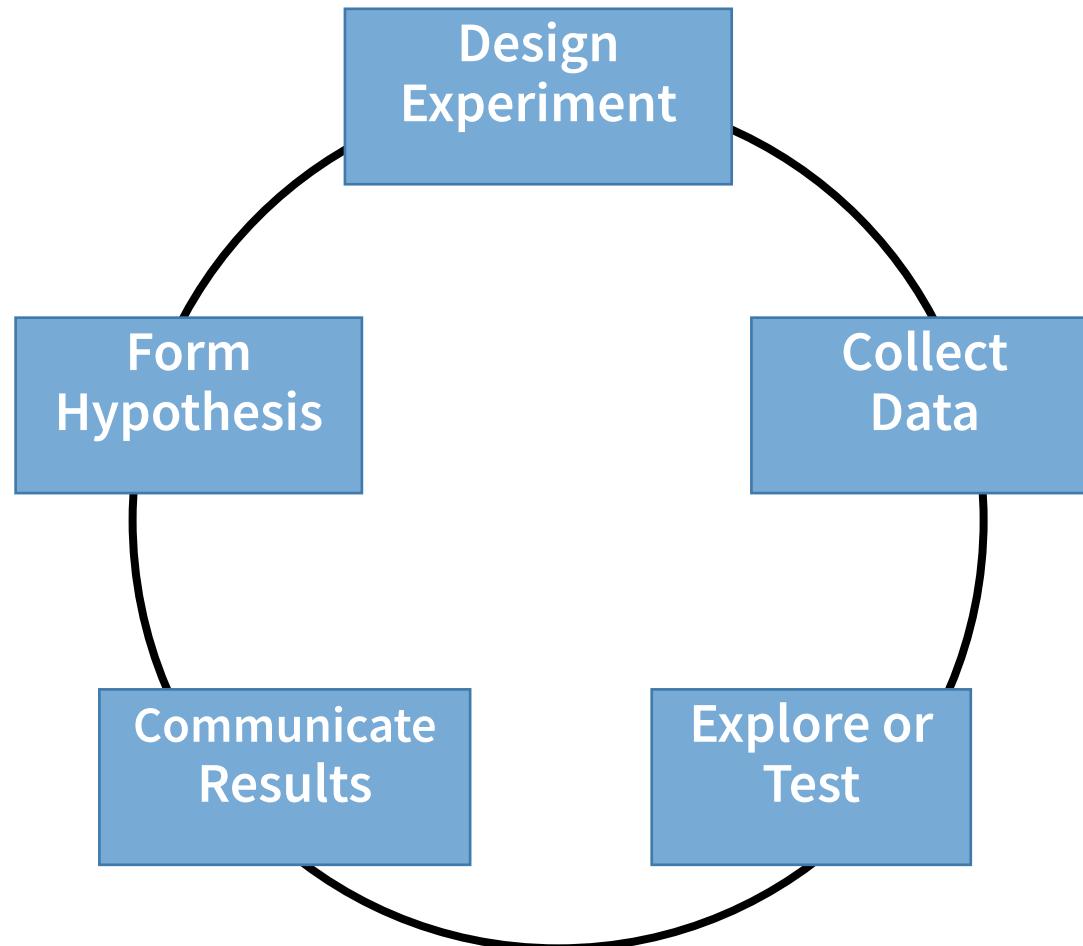
Clone an existing Github Repo

CORE | Skills

View the repository and contents

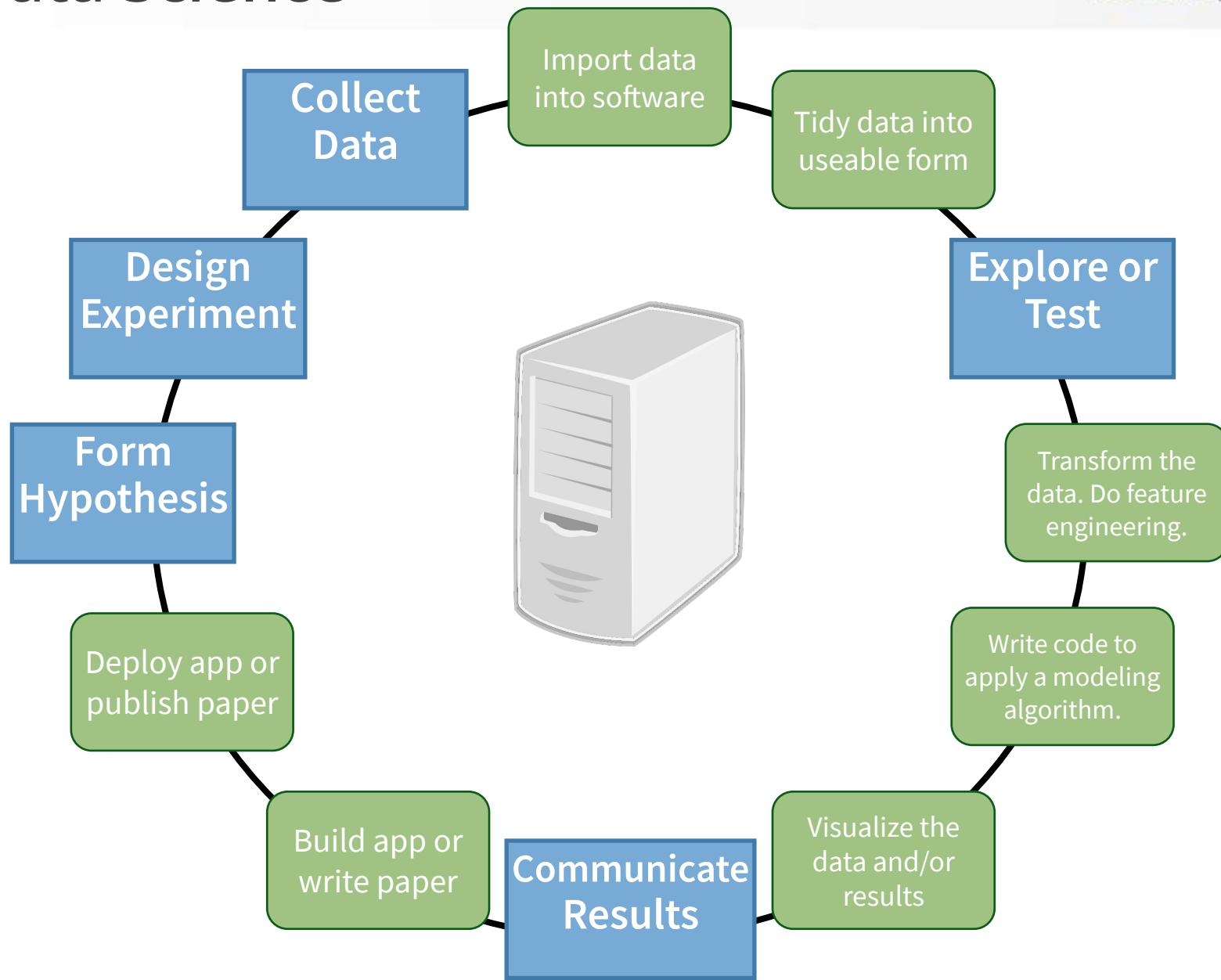


1. Overview of the tools used (Anaconda, Python, Git, Github, Gitkraken)
2. Project set up
 - a. Package management and Environments
3. Navigating your computer
 - a. File structure
 - b. Using bash or command prompt to move around
4. Intro to Jupyter notebooks + IDEs
5. Reproducibility
6. Version control
7. Good code etiquette
8. Where to find help



Data Science

CORE | Skills

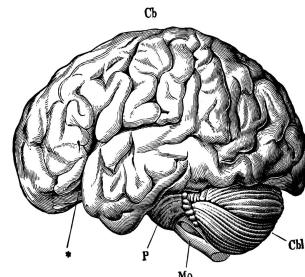


How to communicate with your PC

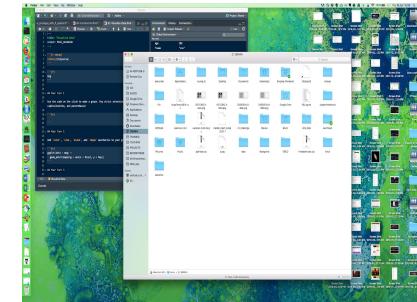
CORE | Skills



1. run programs
2. store data
3. communicate with each other, and
4. interact with us

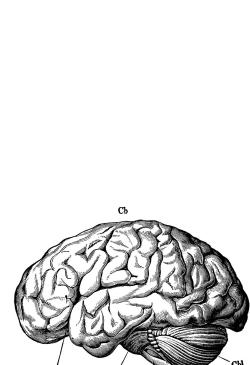


Human thought



- Simple tasks
- One at a time
- Hard to automate or reproduce

Machine language



Human thought



C++



Machine language

Tools used throughout the course

CORE | Skills

- > Distribution of Python and R with commonly used packages and tools
- > Includes package and environment management system *conda*



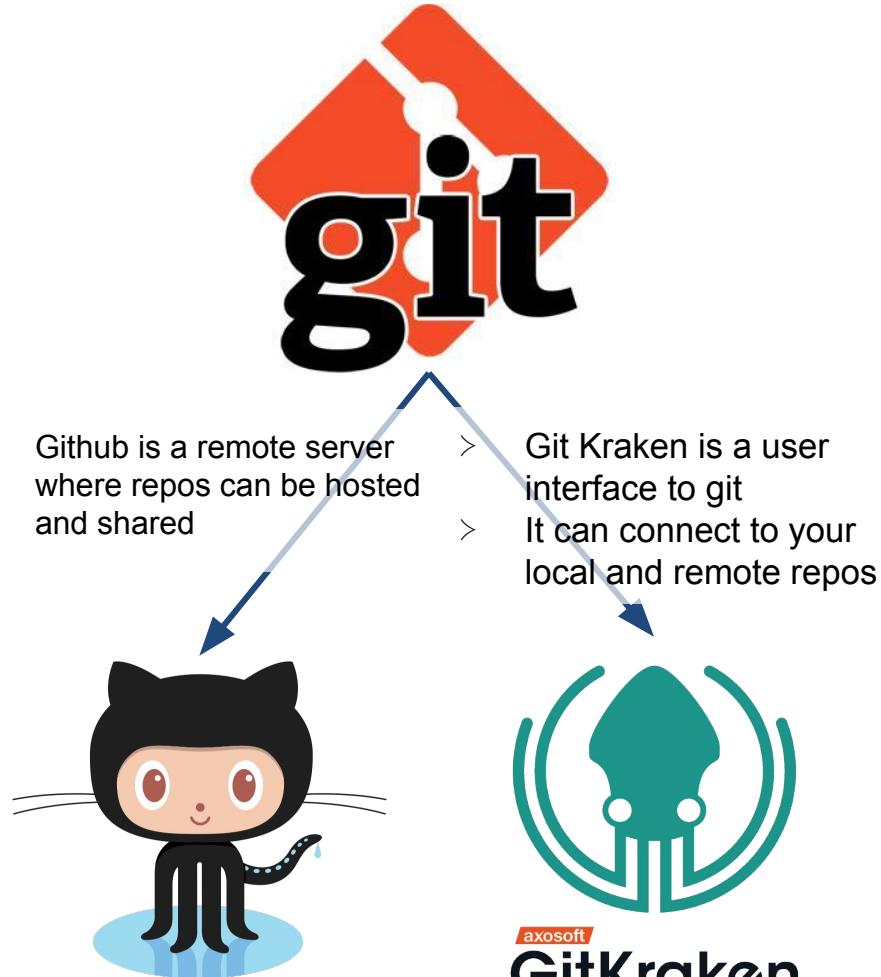
- > web-based, interactive computational environment
- > Write code and markdown to share results and work
- > High level, interpreted programming language



- > Create environments for projects which use incompatible/different versions of packages
- > Use it to download and update packages from a central repo

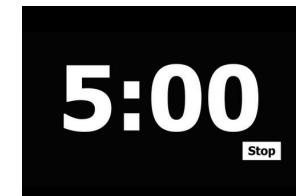
Version Control System

- > Keep track of your evolving code



- > Github is a remote server where repos can be hosted and shared
- > Git Kraken is a user interface to git
- > It can connect to your local and remote repos

Folder structures / workflow



At your table discuss how you currently organise your files, feel free to talk about a specific dataset you are working on.

Consider the following:

- How do you handle data? Where is it stored?
- How do you keep track of your workflow?
- Do you have a naming convention for directories and files?
- Can you draw your proposed folder structure?

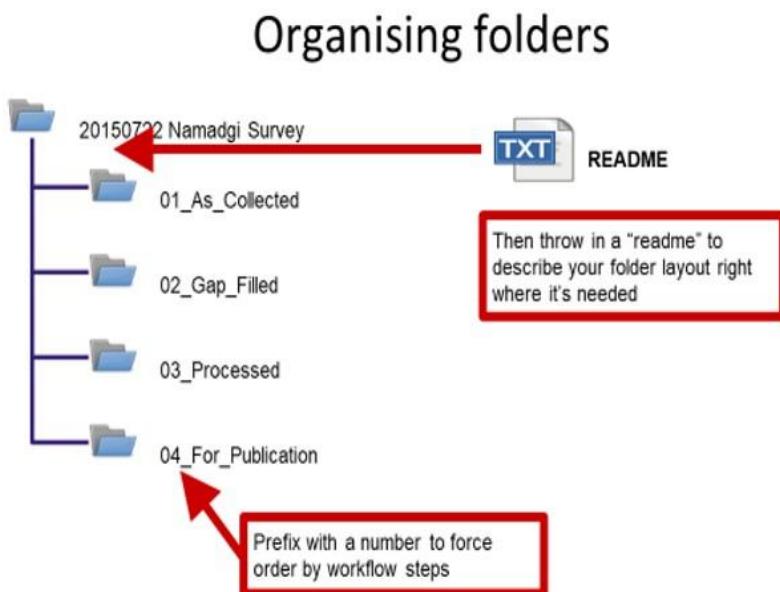
Additional:

See if you can devise a better naming convention or note one or two improvements you could make to how you name your files

Project set up

1. Folder structures

- a. Structure folders **hierarchically** - start with a limited number of folders for the broader topics, and then create more specific folders within these
- b. **Separate ongoing and completed work** - as you start to create lots of folders and files, it is a good idea to start thinking about separating your older documents from those you are currently working on
- c. Probably the simplest way to **document your structure** - for your future reference - is to add a “README” file - a text file outlining the contents of the folder.



Micah Allen
@micahgallen

My universal directory structure, forged from years of bad organization:

```
/Projects
.../inProgress
...../ProjectName
......./docs
......./code
......./data
......./figures
.../published
.../submitted
```

You're welcome.

3:20 PM · May 29, 2018 · Twitter for Android

405 Retweets 1.7K Likes

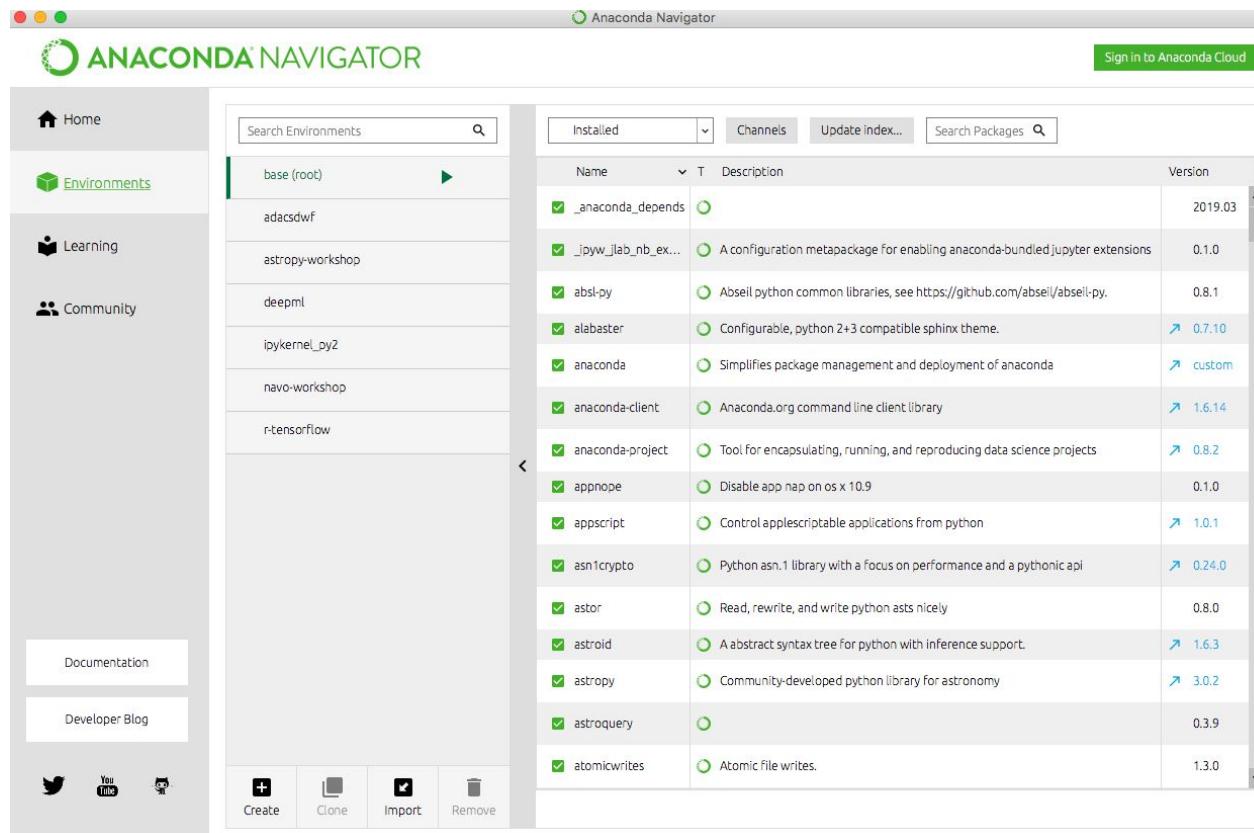
Project set up

Package and Environment Management

1. Software and packages have **different versions**
2. Different software and packages can have **different conflicting dependencies**

It is good practice to manage these dependencies by setting up an **environment**.

conda is an environment manager which can be used via the command line or the anaconda navigator.

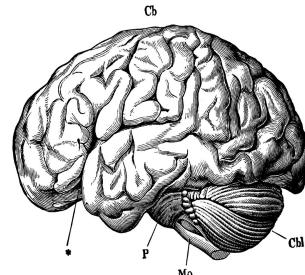


Short intro to the command line

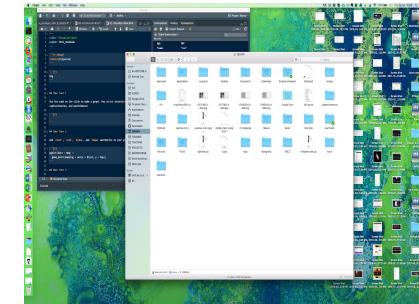
CORE | Skills



1. run programs
2. store data
3. communicate with each other, and
4. interact with us



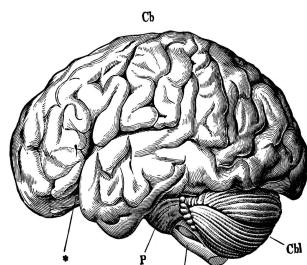
Human thought



- Simple tasks
- One at a time
- Hard to automate or reproduce



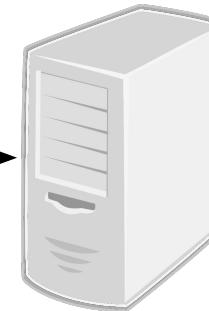
Machine language



Human thought



- Simple language + command line interface
- Read-evaluate-print loop (**REPL**)
- The shell is a program which runs other programs instead of doing its own calculation
- Great for automating tasks
- Scripting allows for easy reproducibility



Machine language

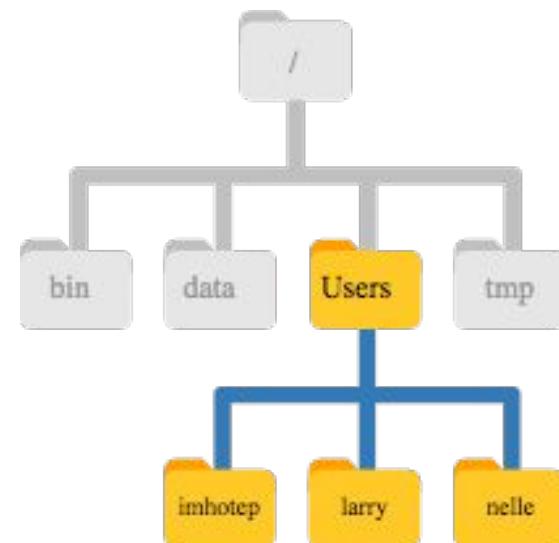
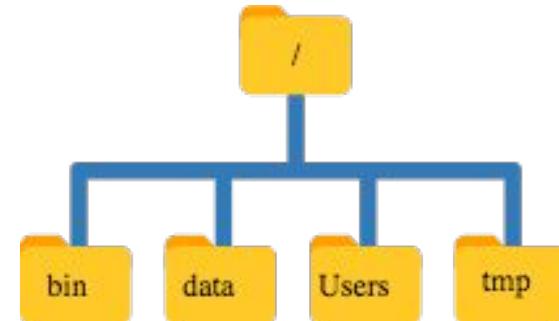
Navigating your computer

The part of the operating system responsible for **managing files and directories** is called the **file system**.

- The file system is responsible for managing information on the disk.
- Information is stored in files, which are stored in directories (folders).
- Directories can also store other directories, which forms a directory tree.

Every user on a computer will have a **home directory**.

The home directory path will look different on different operating systems. On Linux it may look like `/home/nelle`, and on Windows it will be similar to `C:\Documents and Settings\nelle` or `C:\Users\nelle`



Navigating your computer

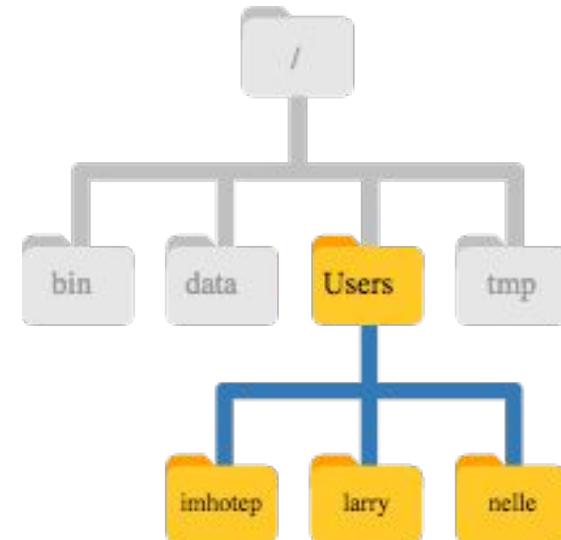
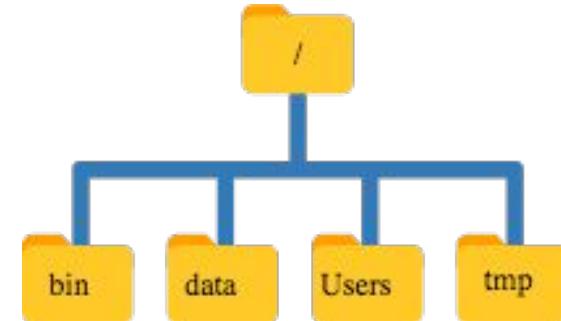
Directory Tree:

At the top is the **root directory** that holds everything else. We refer to it using a slash character, `/`, **on its own**.

Inside that directory are several other directories, in which are other directories, and so on.

Directories can also contain executables, files and links to directories/executables/files.

Most files' names are `something.extension`. The extension isn't required, and doesn't guarantee anything, but is normally used to indicate the type of data in the file, e.g., `.txt`.



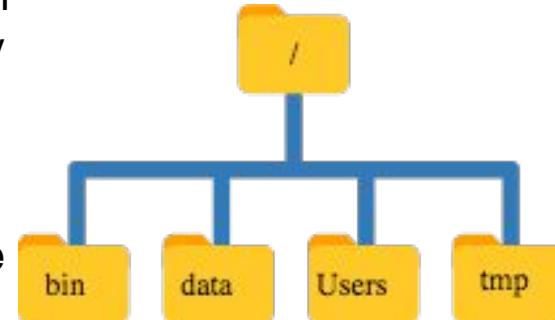
Navigating your computer

When creating a **path** (i.e. address) to a file we use its location within the directory tree to locate it. To separate directory names in the path name we use `/` or `\`:

`C:\Users\nelle\report.txt`

There are two types of paths: **relative path** and **absolute path**:

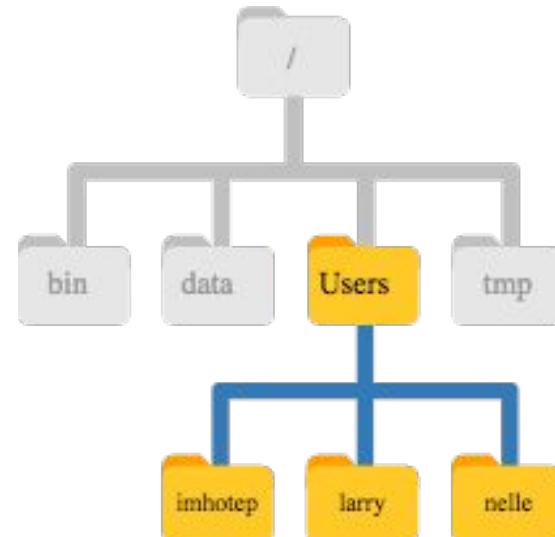
- A relative path specifies a location starting from the current location.
- An absolute path specifies a location from the root of the file system.



There are also special characters to describe locations in the directory tree:

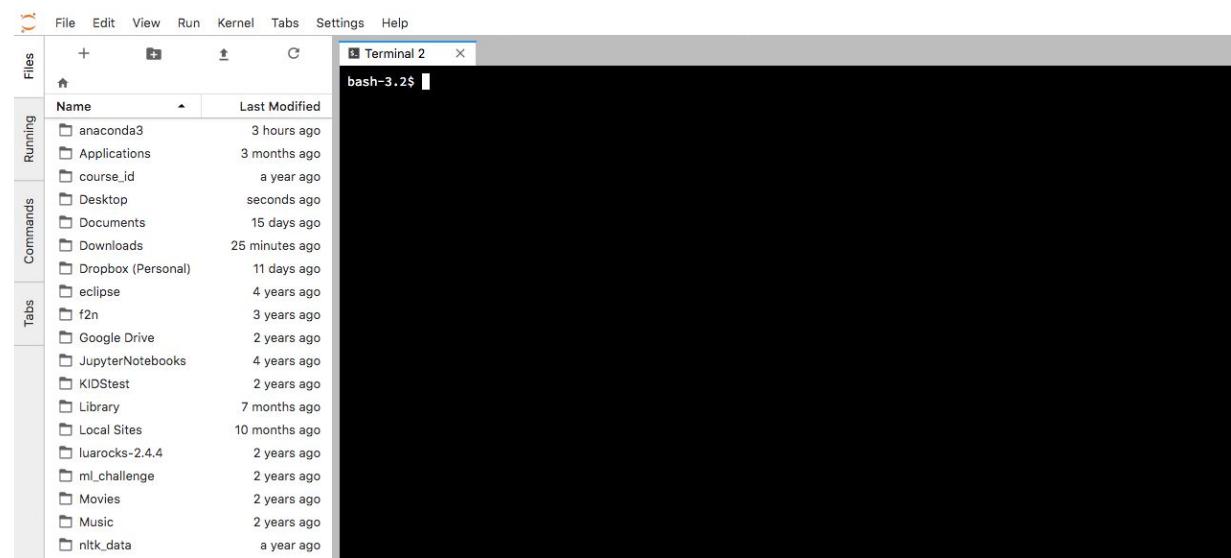
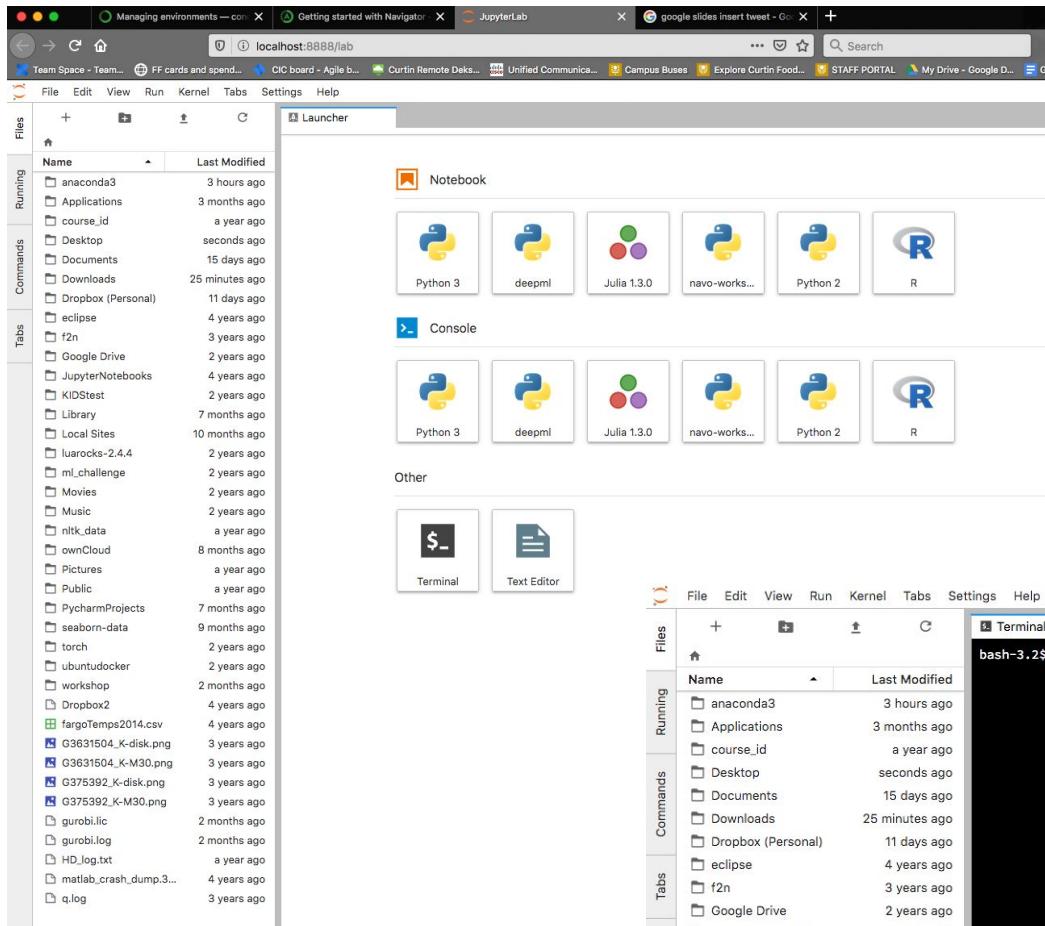
- `..` means 'the directory above the current one';
- `.` on its own means 'the current directory'.
- `~` is the current user's home directory, has to be at the start of specified path

`C:\Users\nelle\report.txt` is equivalent to `~\report.txt`

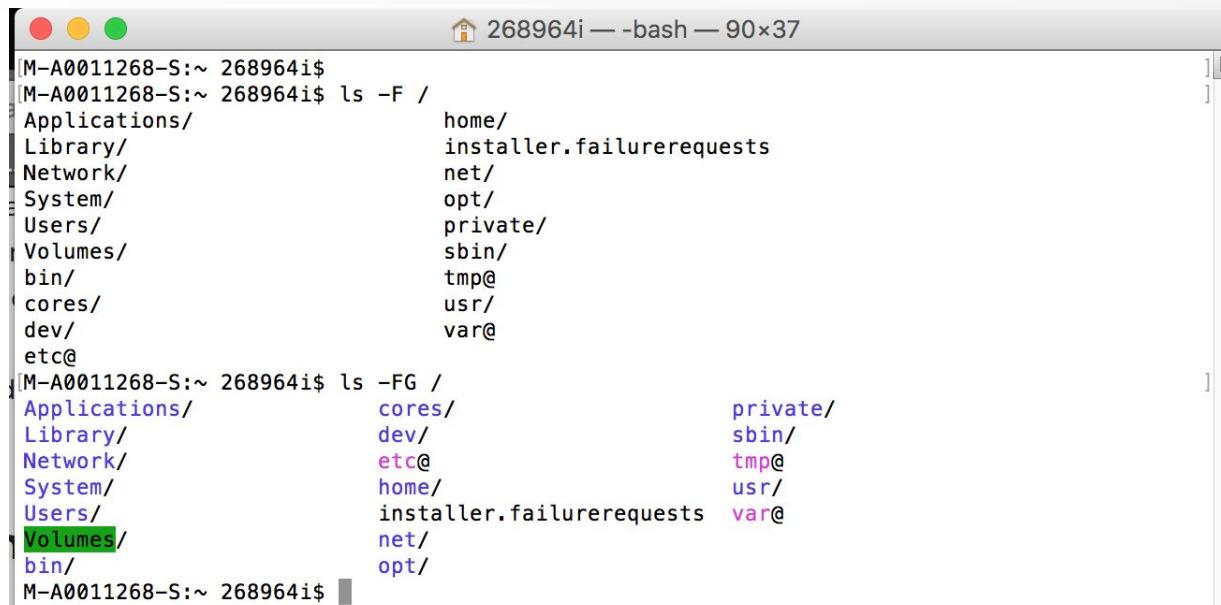


Short intro to the command line

Open JupyterLab and start a Terminal



Short intro to the command line



```
[M-A0011268-S:~ 268964i$ ls -F /
Applications/          home/
Library/                installer.failurerequests
Network/               net/
System/                opt/
Users/                 private/
Volumes/               sbin/
bin/                   tmp@
cores/                 usr/
dev/                   var@

[M-A0011268-S:~ 268964i$ ls -FG /
Applications/          cores/           private/
Library/                dev/             sbin/
Network/               etc@            tmp@
System/                home/           usr/
Users/                 installer.failurerequests  var@
Volumes/               net/            usr/
bin/                   opt/            var@

M-A0011268-S:~ 268964i$
```

The first line shows only a **prompt**, indicating that the shell is waiting for input

The part that you type, `ls -F /` in the second line of the example, typically has the following structure: a **command**, some **flags** (also called **options** or **switches**) and an **argument**.

Flags start with a single dash (`-`) or two dashes (`--`), and change the behaviour of a command.

Arguments tell the command what to operate on (e.g. files and directories).

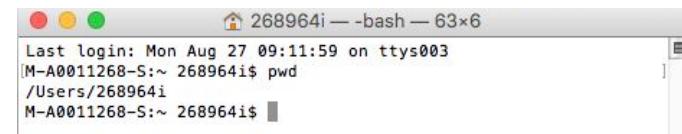
A command can be called with more than one flag and more than one argument: but a command doesn't always require an argument or a flag!

Intro to Bash - Navigating

When we open the Bash terminal we start out in our **home directory**.

Let's find out where this is exactly by running a command called `pwd` (which stands for "print working directory").

```
$ pwd
```

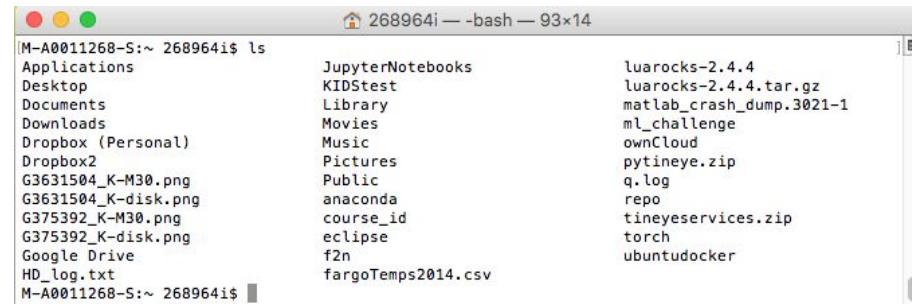


A screenshot of a Mac OS X terminal window titled "268964i — -bash — 63x6". The window shows the command "pwd" being run, which returns the path "/Users/268964i".

We can see what's in our home directory by running `ls`, which stands for "listing"

```
$ ls
```

```
$ dir
```



A screenshot of a Mac OS X terminal window titled "268964i — -bash — 93x14". The window shows the command "ls" being run, listing various files and directories in the user's home directory.

File/Folder	File/Folder	File/Folder
Applications	JupyterNotebooks	luarocks-2.4.4
Desktop	KIDTest	luarocks-2.4.4.tar.gz
Documents	Library	matlab_crash_dump.3021-1
Downloads	Movies	ml_challenge
Dropbox (Personal)	Music	ownCloud
Dropbox2	Pictures	pytineye.zip
G3631504_K-M30.png	Public	q.log
G3631504_K-disk.png	anaconda	repo
G375392_K-M30.png	course_id	tineyeservices.zip
G375392_K-disk.png	eclipse	torch
Google Drive	f2n	ubuntudocker
HD_log.txt	fargoTemps2014.csv	

Remember a **command** can often be followed by **flags** and/or **argument**, e.g.:

```
$ ls -FG
```

`ls` has lots of other **flags**. There are two common ways to find out how to use a command and what flags it accepts:

```
$ ls --help      or      $ man ls
```

There is also a handy *tldr* online, explaining the most commonly used command options:

<https://tldr.ostera.io/>

Intro to Bash - Navigating

CORE | Skills

Next let's **change our location** to a different directory, so we are no longer located in our home directory.

The command to change locations is `cd` ("change directory") followed by a directory name to change our working directory:

```
$ cd workshop
```

To check this worked:

```
$ pwd
```

And to see the content of the folder, including hidden files and directories:

```
$ ls -aFG
```

Special names:

-  → this location
-  → the directory above
-  → the current user's home directory, has to be at the start of specified path
-  → the previous directory I was in

Useful command

\$ ls -Flag [location]	list content of specified location, using specified flags
\$ dir [location] specified location	windows/anaconda command prompt: list content of specified location
\$ pwd	print working directory → current location in filesystem
\$ cd [location] . and .. ~ and -	change directory to specified location, relative paths work special characters denoting <i>here</i> and <i>directory above</i> special characters denoting <i>HOME</i> and <i>previous directory</i>
\$ mkdir [name]	make directory with specified name (can include paths)
\$ nano [filename] CTRL-O <i>then</i> <Enter> CTRL-X	open specified file using the <i>nano</i> text editor <i>nano</i> command to save content of file <i>nano</i> command to close file (asks confirmation if file changed)
\$ touch [filename]	creates empty file with specified name if file does not exist

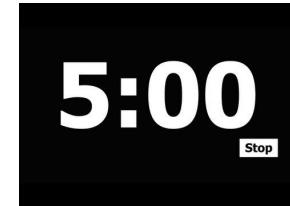
If using windows command prompt check out these useful commands:

<https://www.digitalcitizen.life/command-prompt-how-use-basic-commands>

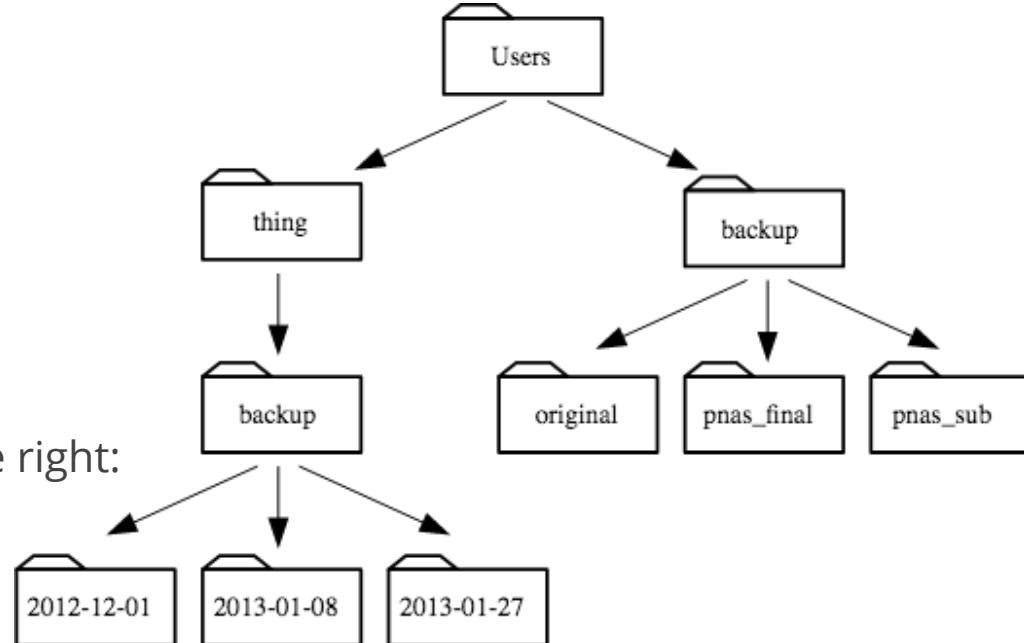
Navigating your computer

Starting from `/Users/amanda/data/`, which of the following commands could be used to navigate to the home directory, which is `/Users/amanda`?

1. `cd .`
2. `cd /`
3. `cd /home/amanda`
4. `cd ../../`
5. `cd ~`
6. `cd home`
7. `cd ~/data/..`
8. `cd`
9. `cd ..`



Using the file system diagram on the right:
if `pwd` displays `/Users/thing`,
what will `ls -F ../backup` display?



1. `../backup: No such file or directory`
2. `2012-12-01 2013-01-08 2013-01-27`
3. `2012-12-01/ 2013-01-08/ 2013-01-27/`
4. `original/ pnas_final/ pnas_sub/`

Jupyter Notebooks

The image shows the Jupyter Notebook interface. On the left is a file browser window titled 'Launcher' with a sidebar containing icons for File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The sidebar also shows the path: / ... / 00-Prerequisite / notebooks /. Below the path is a table with columns 'Name' and 'Last Modified'. It lists several files: 'Intro to Jupyter.ipynb' (modified 2 minutes ago), 'Intro_to_python_pandas_answers....' (modified 2 hours ago), 'Intro_to_python_pandas.ipynb' (modified a year ago), and 'my_script.py' (modified a year ago). A blue arrow points from the text 'Launch existing notebook Intro to Jupyter.ipynb' to the 'Intro to Jupyter.ipynb' entry in the file list.

Your Working Directory

Desktop/workshop/00-Prerequisite/notebooks

Launch a new notebook using specified kernel

Notebook

Python 3

deepml

Julia 1.3.0

navo-workshop

Python 2

R

Console

Python 3

deepml

Julia 1.3.0

navo-workshop

Python 2

R

Other

Terminal

Text File

Markdown File

Show Contextual Help

Intro to Jupyter.ipynb

Interactive Development Environment

CORE | Skills

And IDE is like a text editor but with lots of extra fancy-ness added on.

In fact you can take your favorite text editor (emacs or vim) and give it an upgrade with plugins that will turn it into more of an IDE.

- Syntax Highlighting and Checking
- Auto Indentation
- Spell Checking (language aware)

Get a 'real' IDE

Includes: debugging tools, integration with version control, refactoring tools, templates for new modules/files and docstrings.

- PyCharm (not just for python)
- RStudio

Others:

- Spyder (part of anaconda install)
- Emacs and Vim have lots of plug-ins and extensions for this
- Many others -> see what people around you are using

IDEs can help

The screenshot shows the PyCharm preferences interface. On the left, the sidebar has sections for Appearance & Behavior, Keymap, Editor (General, Font, Color Scheme), Code Style (Python selected), EditorConfig, Erlang, HTML, and JSON. The main area is titled 'Editor > Code Style > Python'. It shows a scheme dropdown set to 'Default IDE' and tabs for Tabs and Indents, Spaces, Wrapping and Braces, Blank Lines, Imports, and Other. Under Tabs and Indents, there are checkboxes for 'Use tab character' (unchecked) and 'Smart tabs' (unchecked). The 'Tab size:' field is set to 4. The 'Indent:' field is also set to 4. The 'Continuation indent:' field is set to 8. A code editor window displays Python code:

```
def foo():
    print
    'bar'

def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

On the right, a large 'Options' dialog is open. It has tabs for Editing, Display, Saving, Completion, and Diagnostics. The 'Editing' tab is active. It contains sections for General, Code, Appearance, Pane Layout, Packages, R Markdown, Sweave, Spelling, Git/SVN, Publishing, and Terminal. Under General, the following checkboxes are checked:

- Insert spaces for tab (checked)
- Insert matching parens/quotes (checked)
- Auto-indent code after paste (checked)
- Vertically align arguments in auto-indent (checked)

Under Surround selection on text insertion, the dropdown is set to 'Quotes & Brackets'. Keybindings dropdown is set to 'Default'.

The 'Execution' section has two checked checkboxes:

- Always save R scripts before sourcing (checked)
- Focus console after executing from source (unchecked)

Ctrl+Enter executes dropdown is set to 'Multi-line R statement'.

The 'Snippets' section has one checked checkbox:

- Enable code snippets (checked)

At the bottom are buttons for OK, Cancel, and Apply.

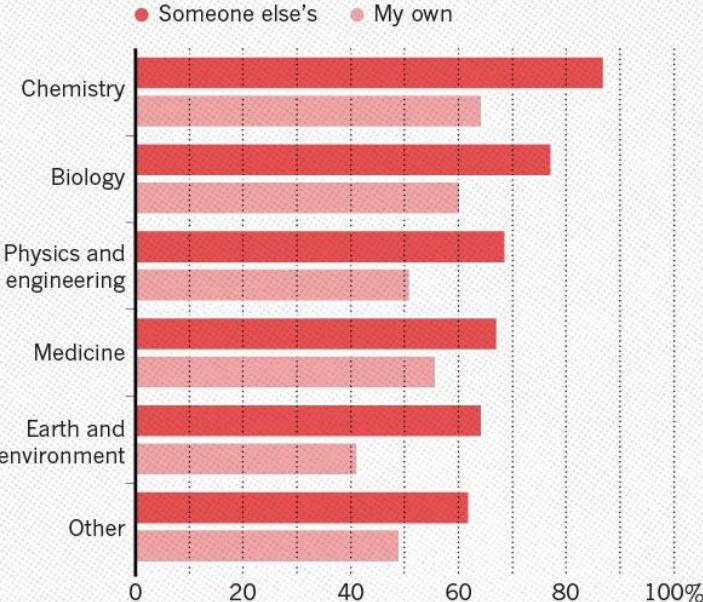
1. Tools used
2. Navigating your computer
3. Reproducibility
 - a. Amanda Miotto's workshop:
<https://guereslib.github.io/Reproducible-Research-Things/>
 - b. Documentation
 - c. Naming conventions:
https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/osmanagement/filename_conv.html
4. Version control
 - a. Introducing git: <https://opensource.com/resources/what-is-git>
 - b. Clone an existing Github repo
 - c. Introducing GitKraken
 - d. GitKraken tutorial
 - e. Collaboration? Github and remote repositories
5. Some good code etiquette hints
6. Where to find help

Reproducibility

HAVE YOU FAILED TO REPRODUCE AN EXPERIMENT?

Skills

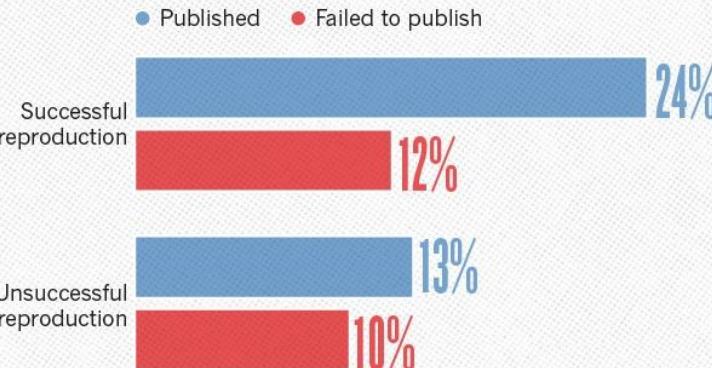
Most scientists have experienced failure to reproduce results.



Why is it important?

HAVE YOU EVER TRIED TO PUBLISH A REPRODUCTION ATTEMPT?

Although only a small proportion of respondents tried to publish replication attempts, many had their papers accepted.



5:00

Stop

Scenario - personnel loss

- i. What would happen? Could you continue their work?
 - 1. Where is the data, code, info on what they are doing?
- ii. How can you improve this scenario?

Documentation:

- Documentation is the idea of documenting your procedures for your experiment so that an outsider could understand the workings of your team.
- Bus factor
- Documentation is a love letter to your future self - Damian Conway

Naming Conventions:

- Easier to process - All team members won't have to over think the file naming process
- Easier to facilitate access, retrieval and storage of files
- Easier to browse through files saving time and effort
- Harder to lose!

Former PhD student and subsequent founder of the Figshare platform, Mark Hahnel, typified a common challenge:

'During my PhD I was never good at managing my research data. I had so many different file names for my data that I always struggled to find the correct file quickly and easily when it was requested. My former PI was so horrified upon seeing the state of my data organisation that she held an emergency lab book meeting with the rest of my group when I was leaving'.

Research Information, April/May 2014

As previously suggested, consistent and meaningful naming of files and folders can make everyone's life easier. See this example below:

YYYYMMDD_SiteA_SensorB.CSV Date Location Sensor

Which when applied, would look like this below:

20150621_Yaouk_Humidity.CSV

Some characters may have special meaning to the operating system so avoid using these characters when you are naming files. These characters include the following: / \ " ' * ; - ? [] () ~ ! \$ { } < # @ & | space tab newline

5:00

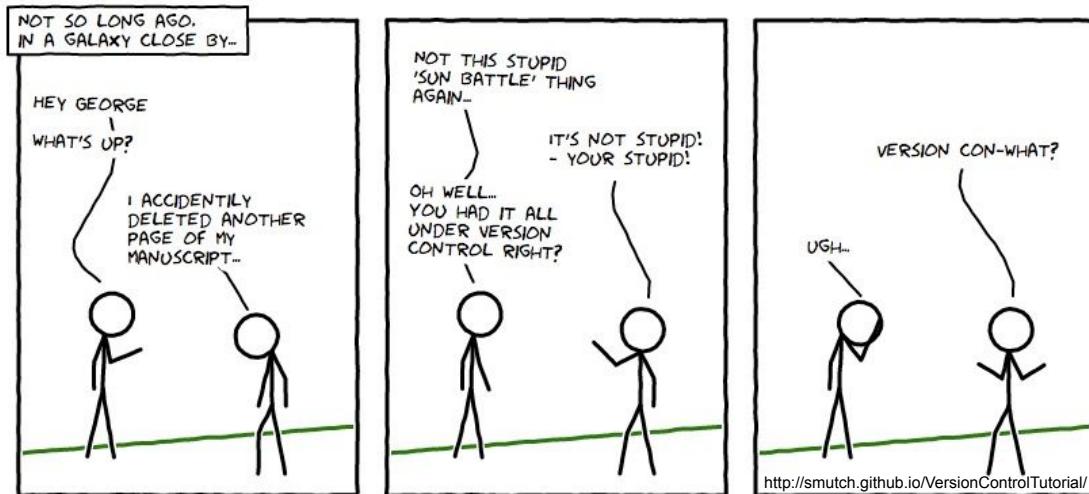
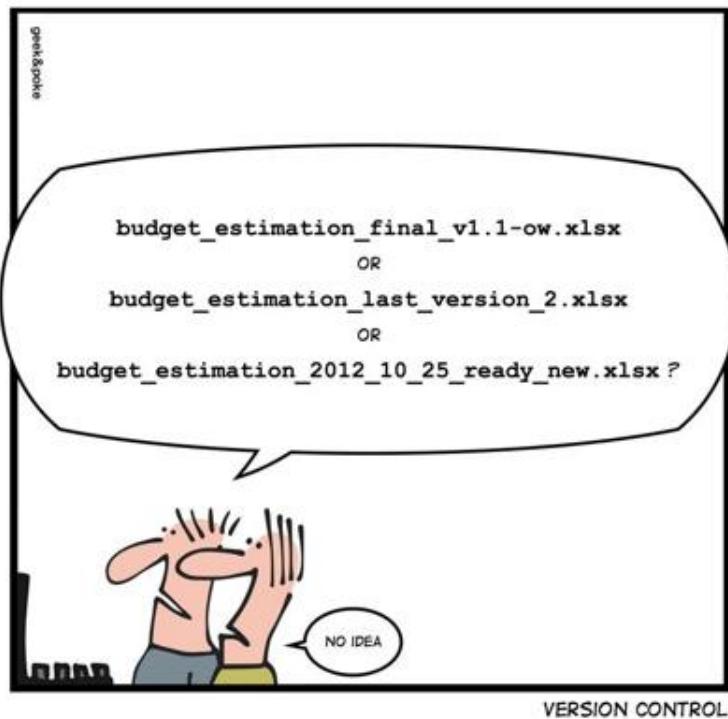
Stop

Discuss - Good or Bad File Names?

- i. Myfile.docx
- ii. 20200130_core_instructions_draft.docx
- iii. Fig 2.png
- iv. 20200130_core_instructions_v1.docx
- v. My figure *v2.png
- vi. fig02_core_instructions_presentation.png

The Problem

SIMPLY EXPLAINED



Version control, a.k.a. revision control / source code management, is basically a system for **recording and managing changes made to files and folders**.

You can track:

- source code (e.g. Python, R, Bash scripts),
- other files containing mostly text (e.g. LaTeX, csv, plain text),
- work by a lone developer, or
- collaboration on projects (track who's done what, branch to develop different streams, etc).

Why Version Control?

As data scientist, we spend much of our time writing code, whether it be for data cleaning, machine learning, or visualisation. As such, our codes are often constantly evolving. By putting all of our code under version control we can:

- **tag code** versions for later reference (*via tags*).
- record a **unique identifier** for the exact code version used to produce a particular plot or result (*via commit identifiers*).
- **roll back** our code to previous states (*via checkout*).
- **identify** when/how **bugs** were introduced (*via diff/blame*).
- **keep multiple versions** of the same code in sync with each other (*via branches/merging*).
- efficiently **share and collaborate** on our codes with others (*via remotes/online hosting*).

Why Version Control?

It's important to also realise that many of the advantages of version control are not limited to just managing code. For example, it can also be useful when writing papers/reports. Here we can use version control to:

- **bring back** that paragraph we accidentally deleted last week.
- **try out a different structure** and simply disregard it if we don't like it.
- **concurrently work on a paper** with a collaborator and then **automatically merge** all of our **changes** together.

The upshot is ***you should use version control for almost everything***. The benefits are well worth it...

In this tutorial we will be using [Git](#) for version control.

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

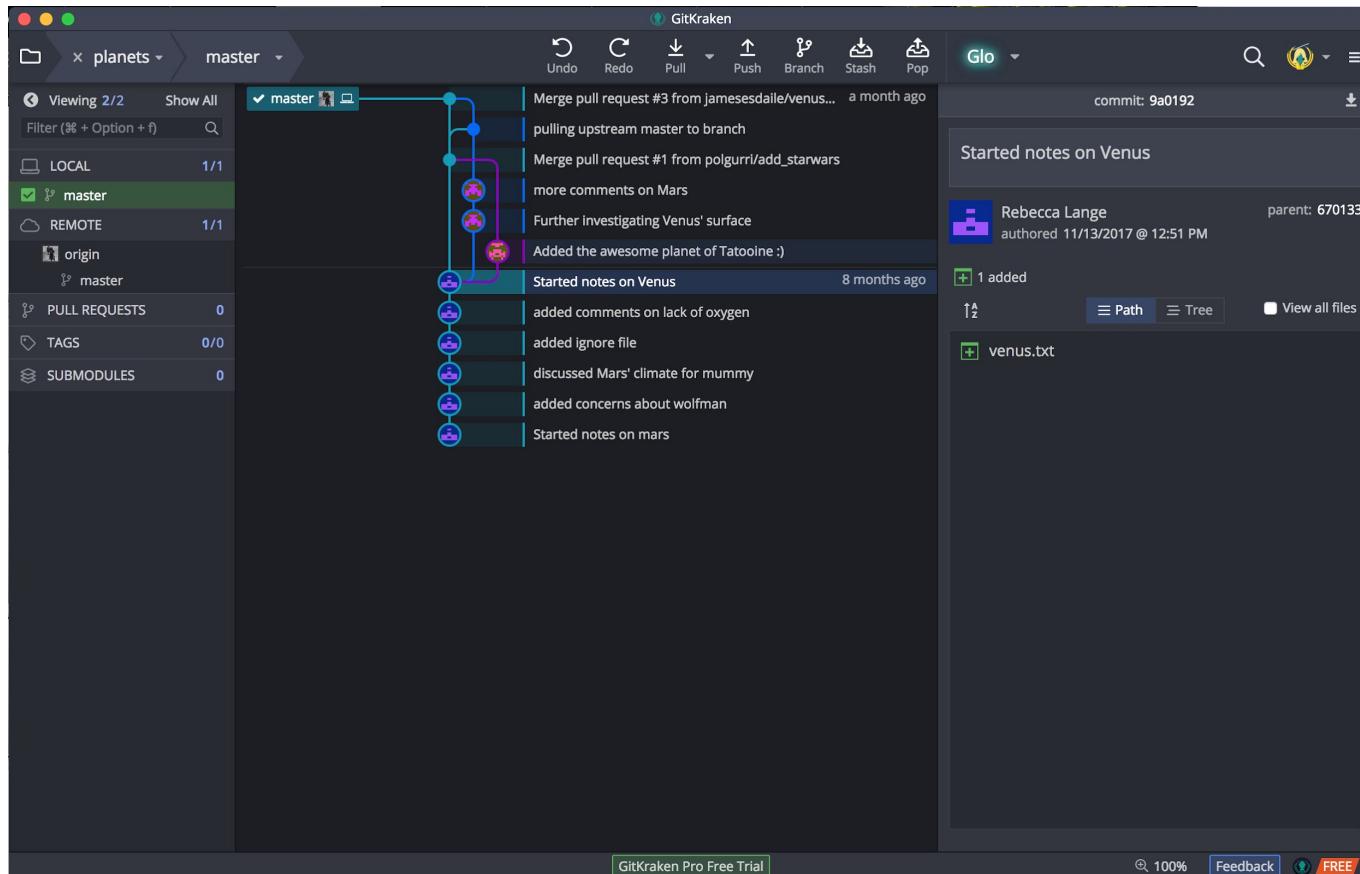
Git tracks changes made to files. You can initialise a Git Repository (the .git/ folder inside a project folder) for a project to build a history of your project over time:

- Added files/folders
- Changed files/folders
- Renamed files/folders
- Removed files/folders

Introducing GitKraken

CORE | Skills

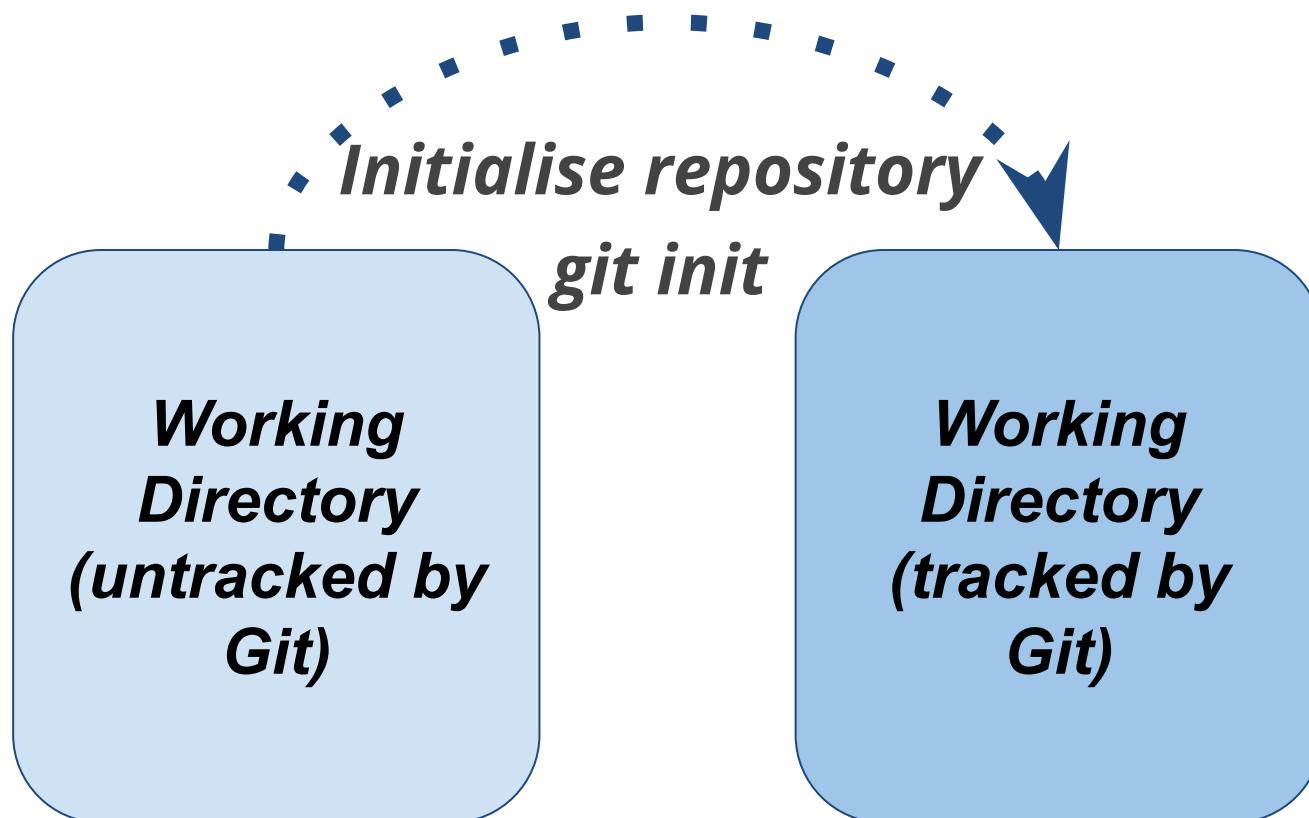
GitKraken is a Git GUI client for Windows, Mac and Linux that is free for non-commercial use. It provides a way to track your project changes by using a graphical interface as an alternative to the command line.



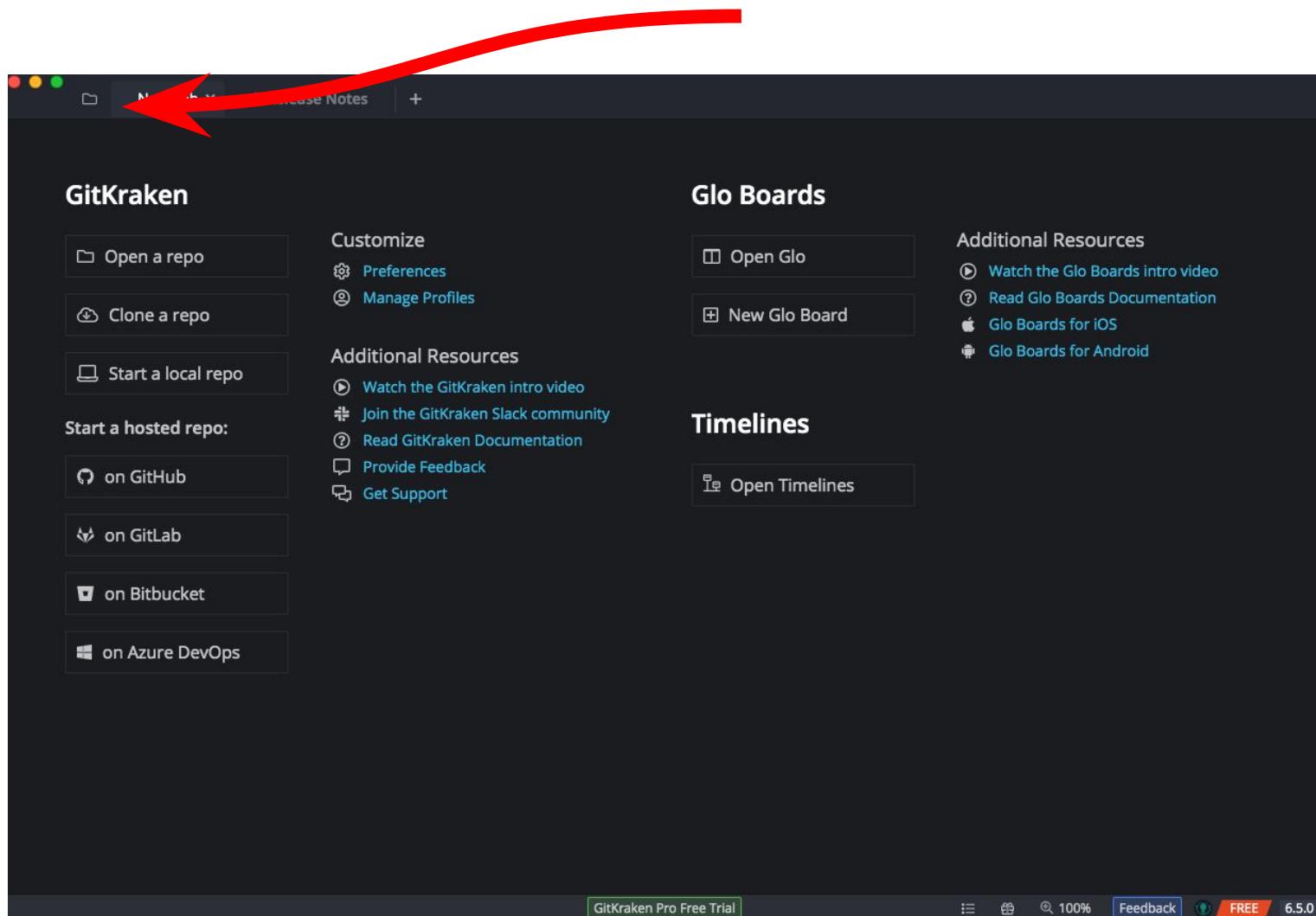
Create a new folder for in the workshop folder called “learning_gitkraken”



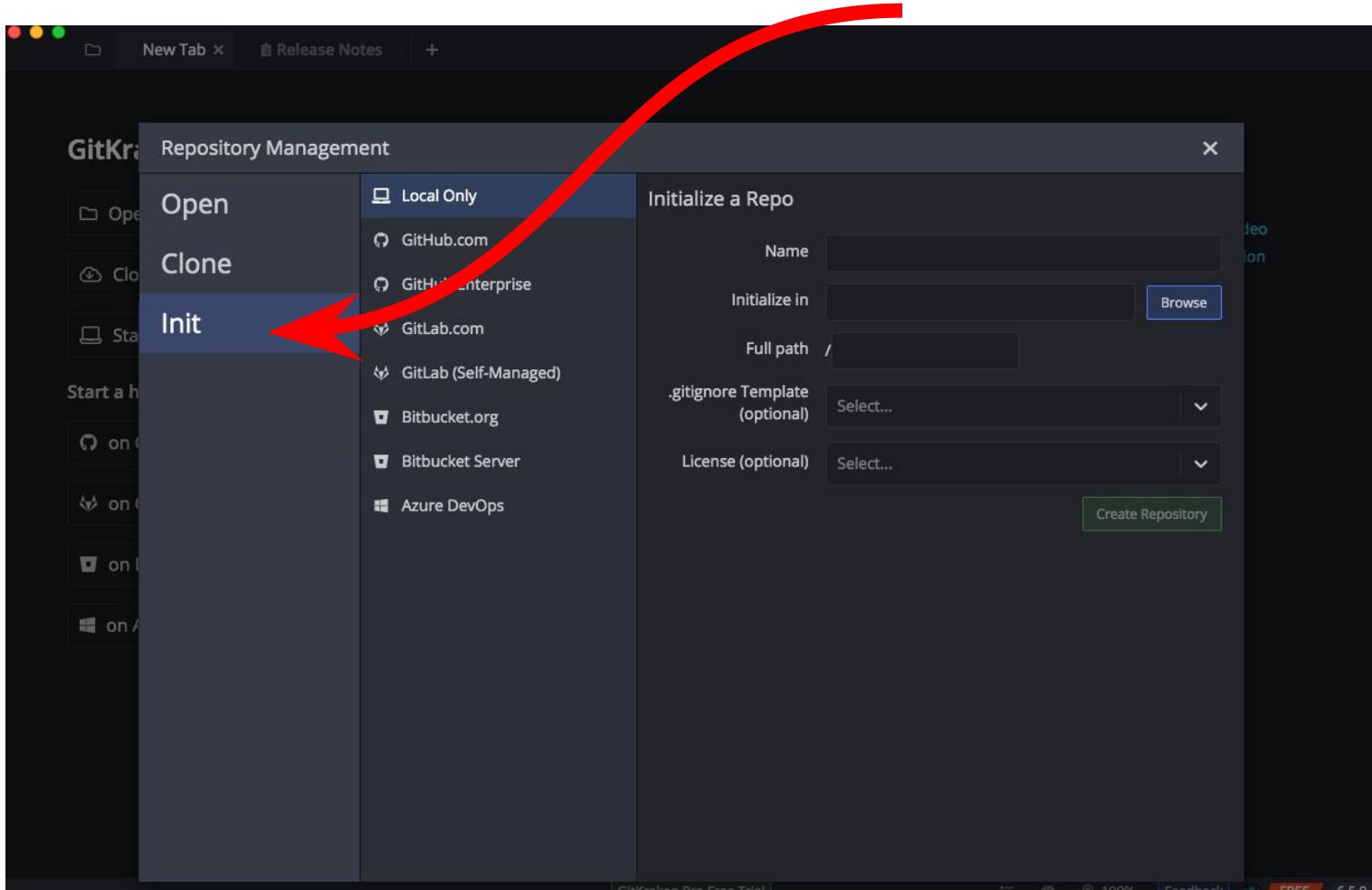
This folder is our 'Working Directory'



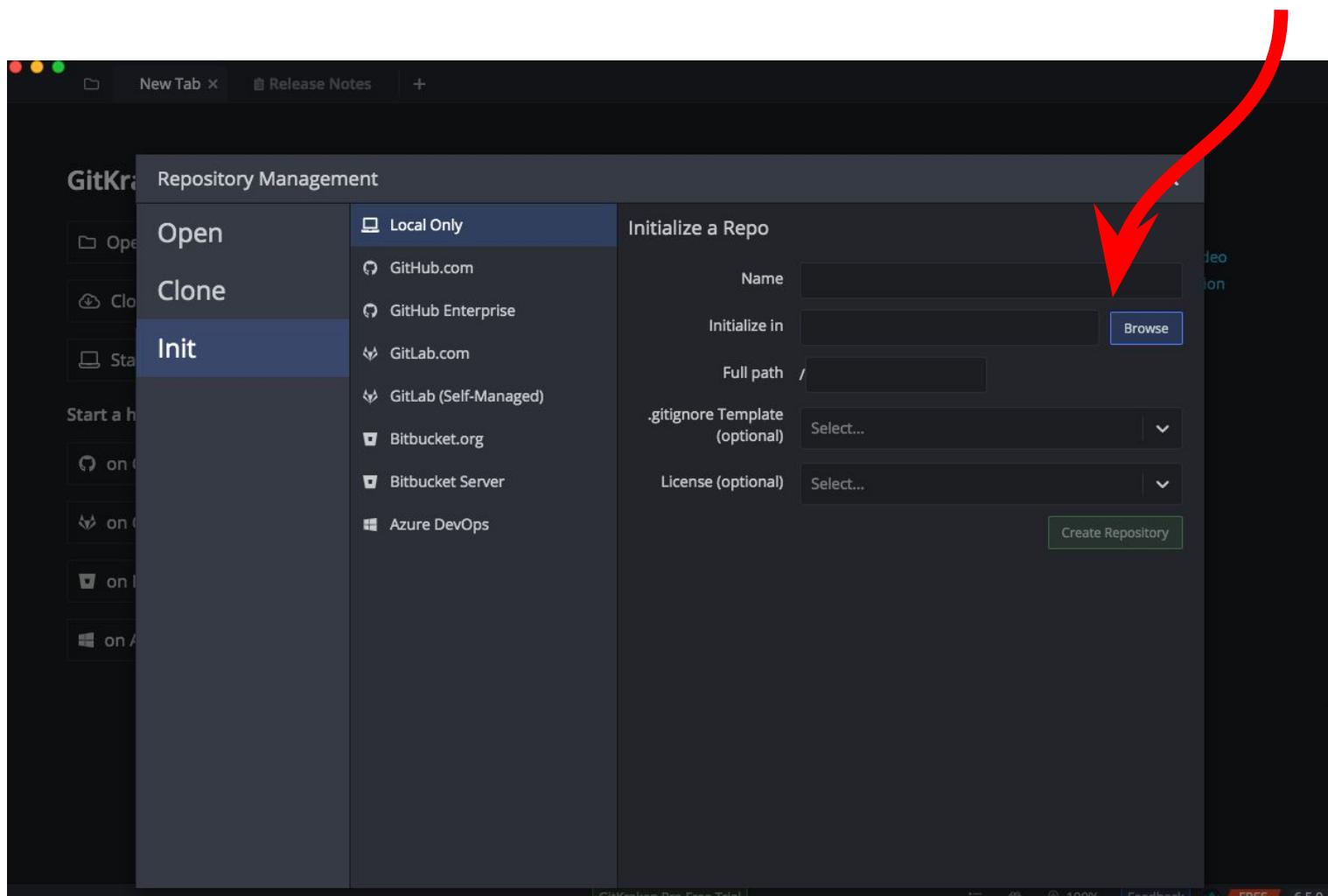
Open GitKraken, click on the 'Folder' Icon



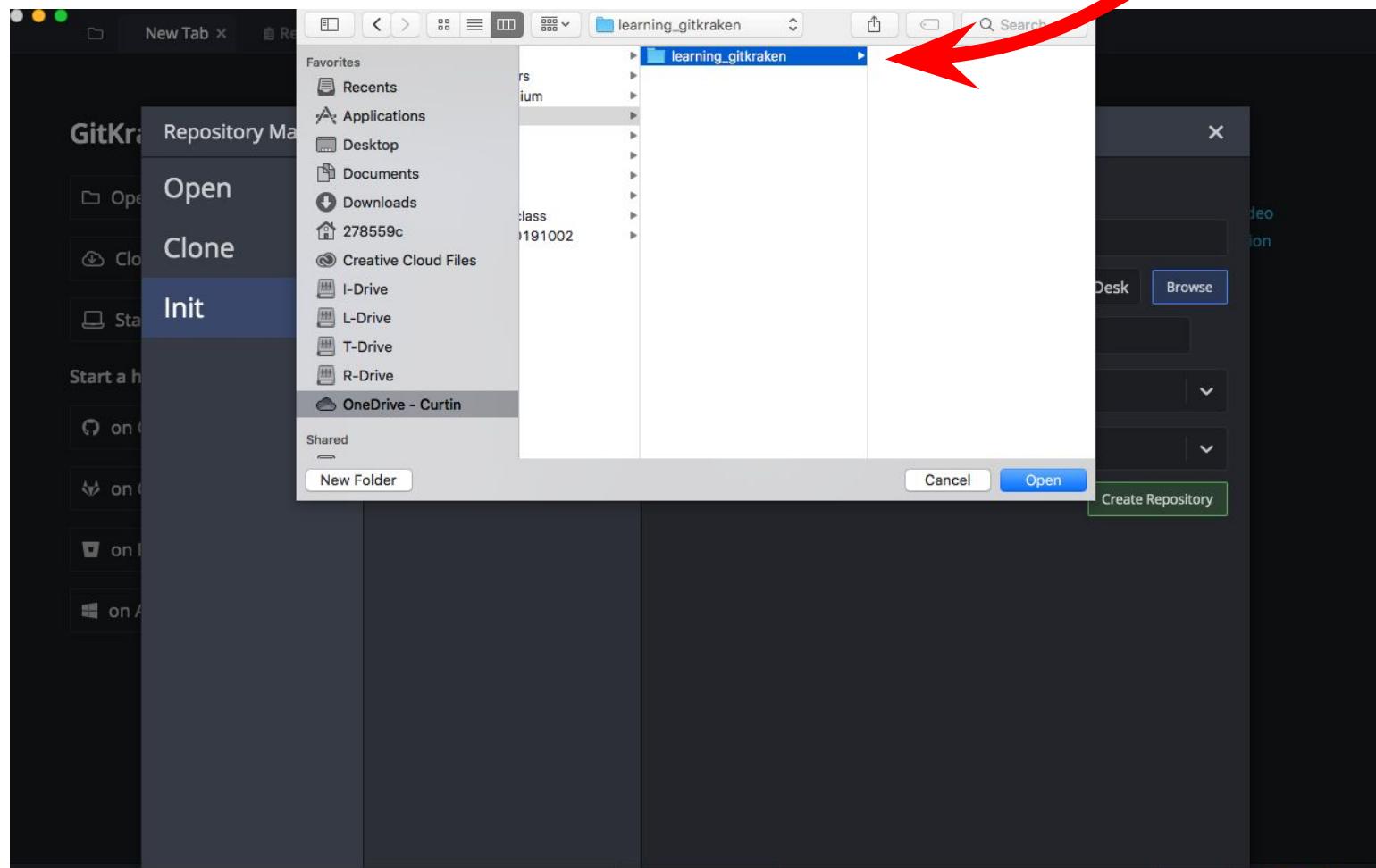
Click 'Init' to initialise the Git Repository



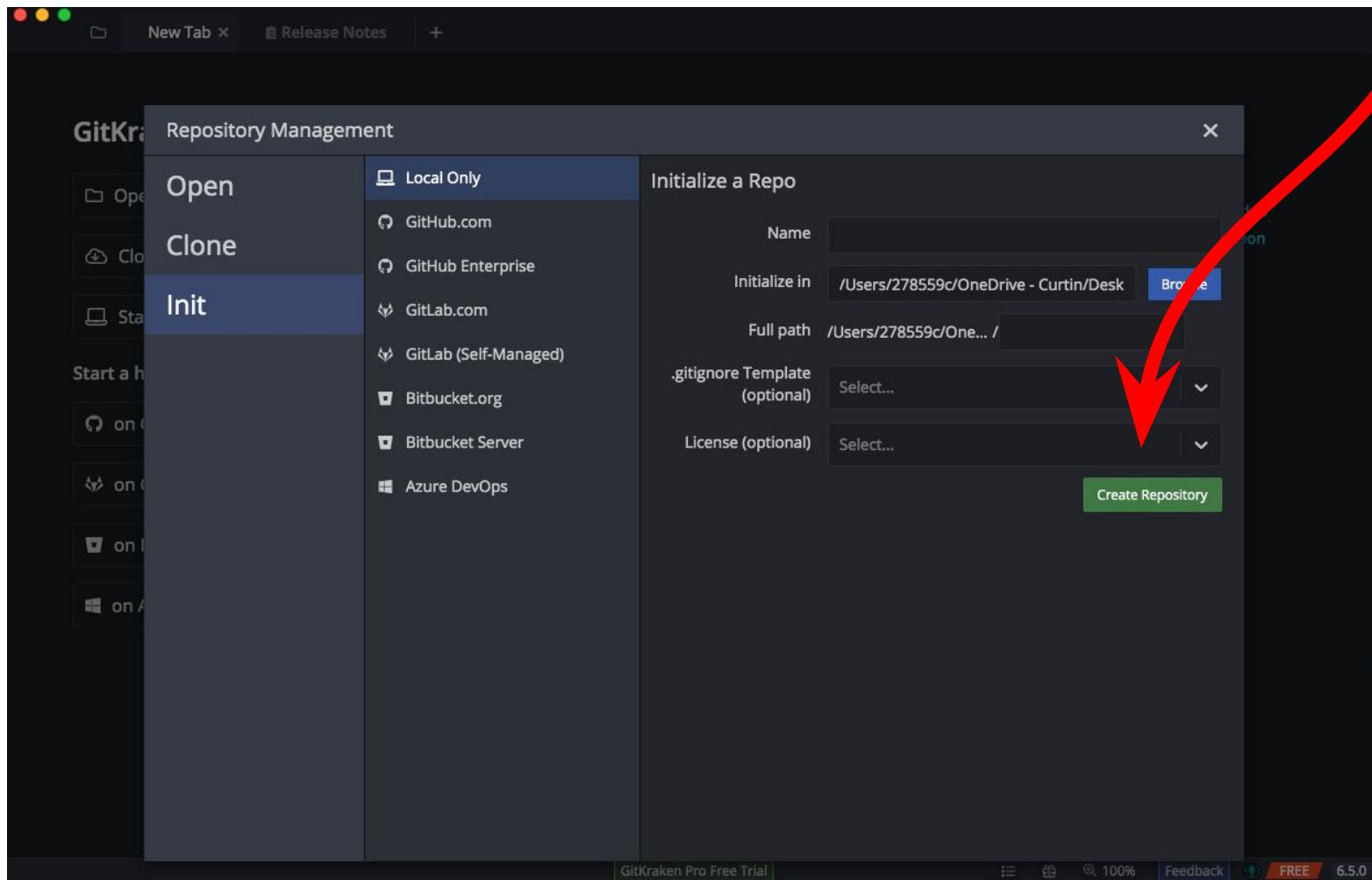
Click 'Browse' to select the folder you created



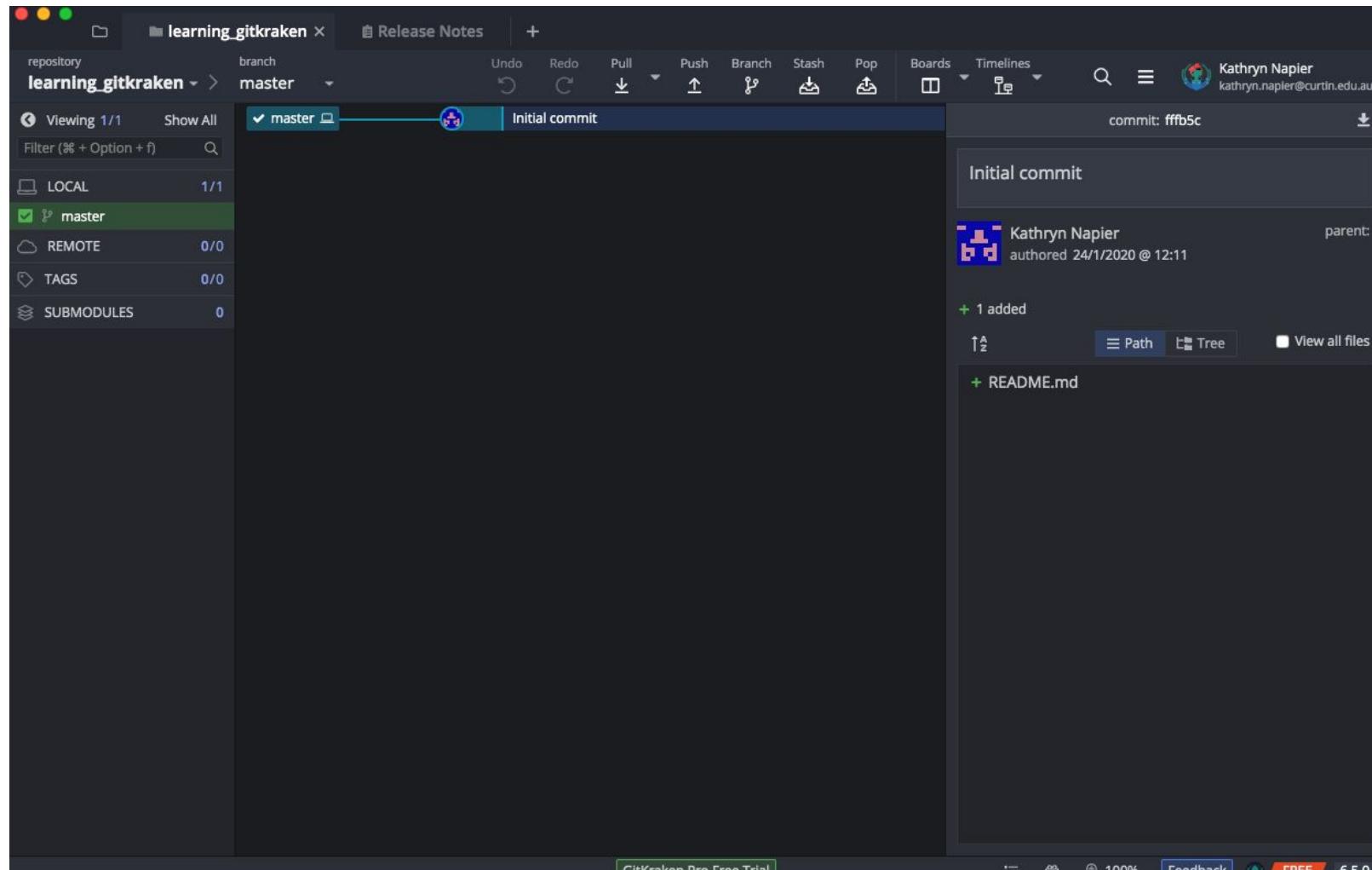
Select the 'learning_gitkraken' folder



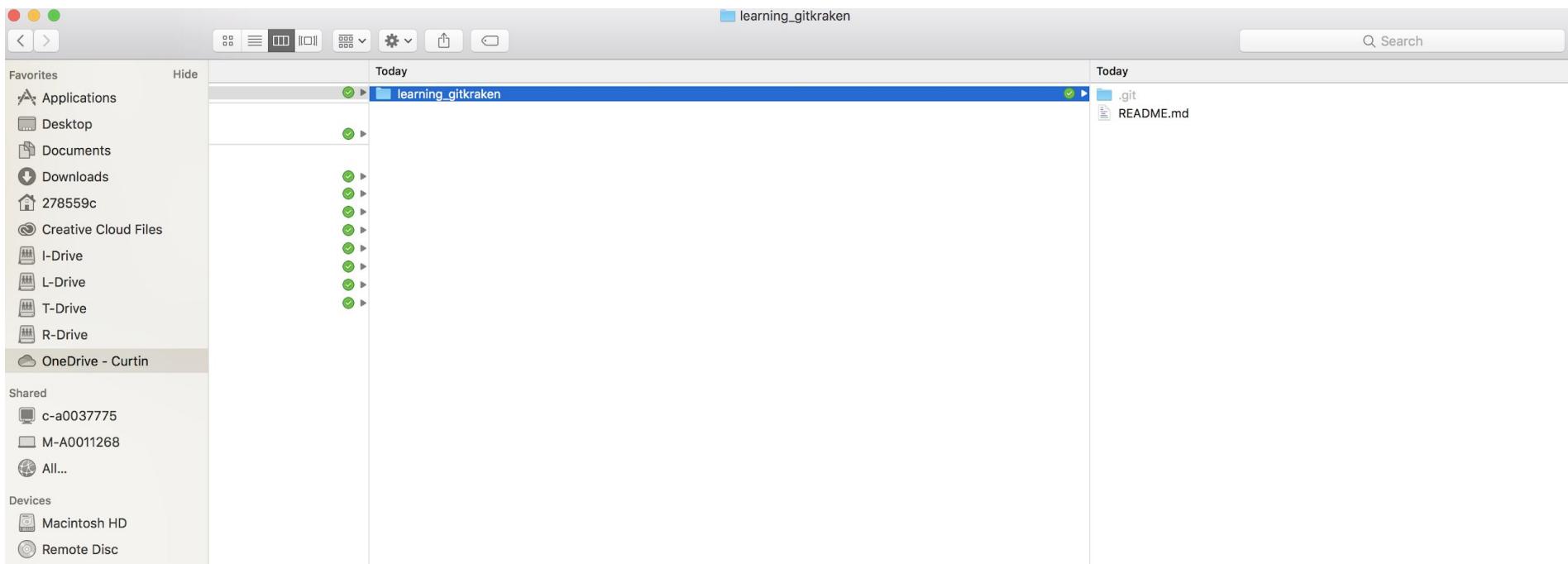
Click 'Create Repository'



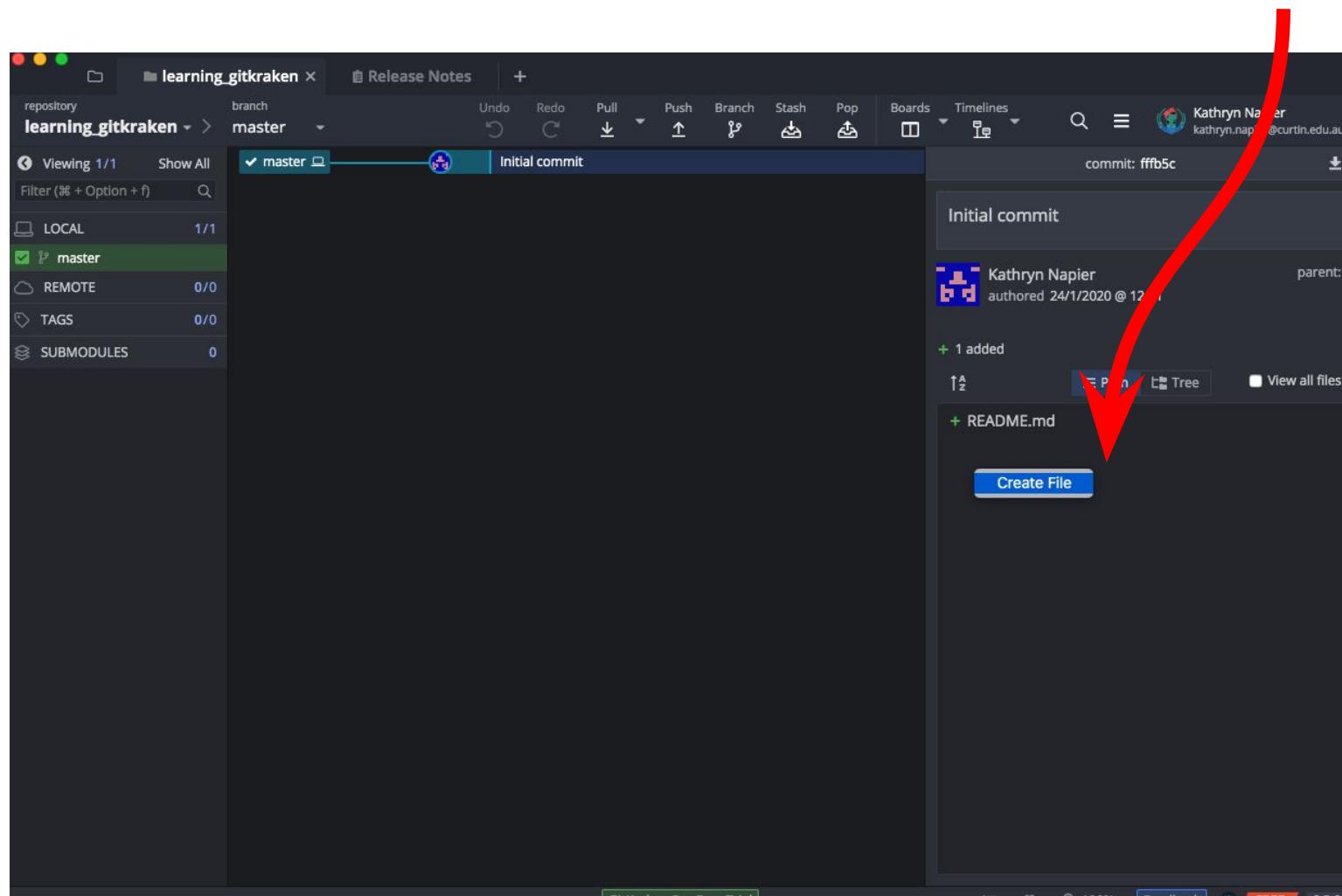
You can now start tracking changes!



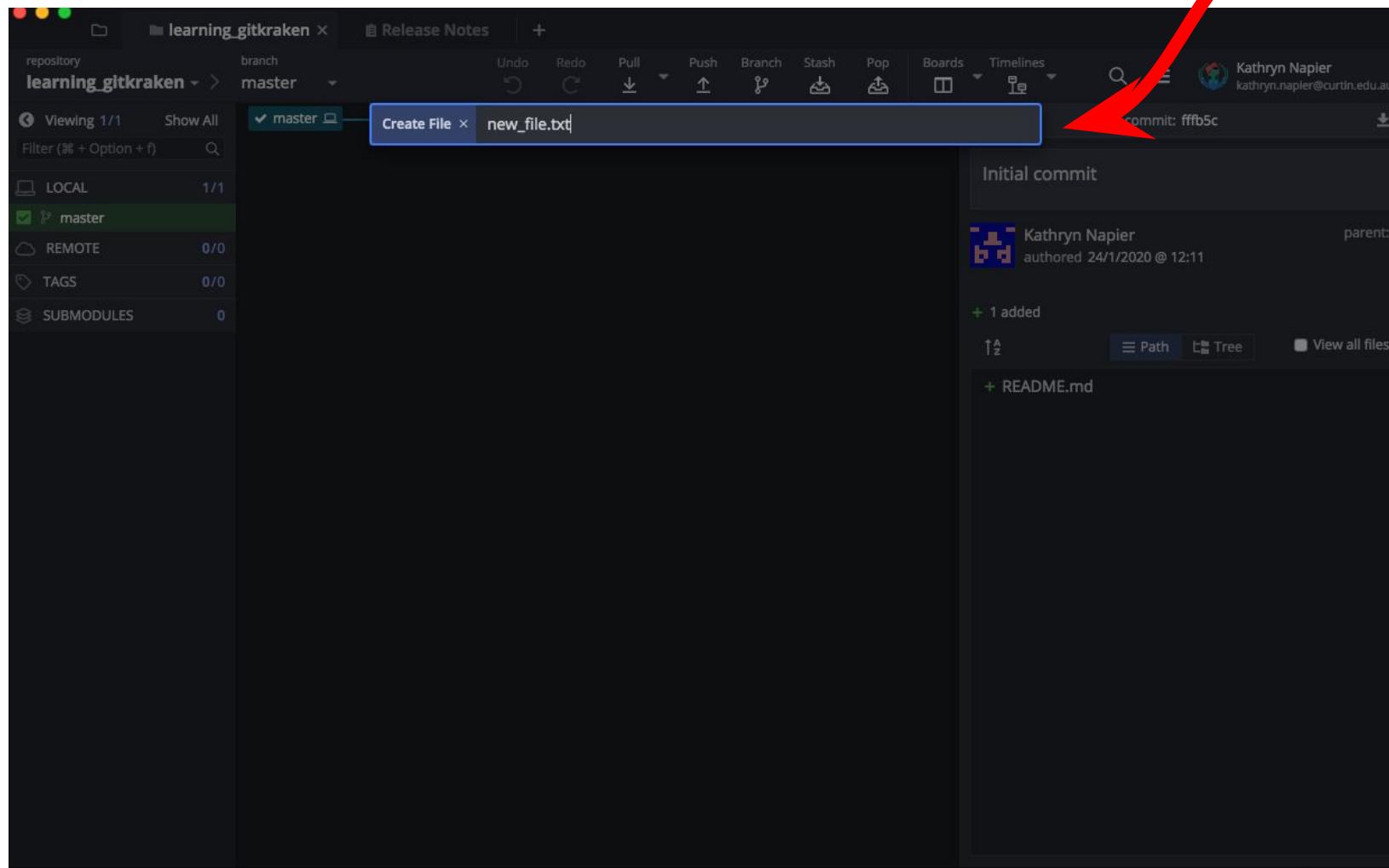
You can now start tracking changes!



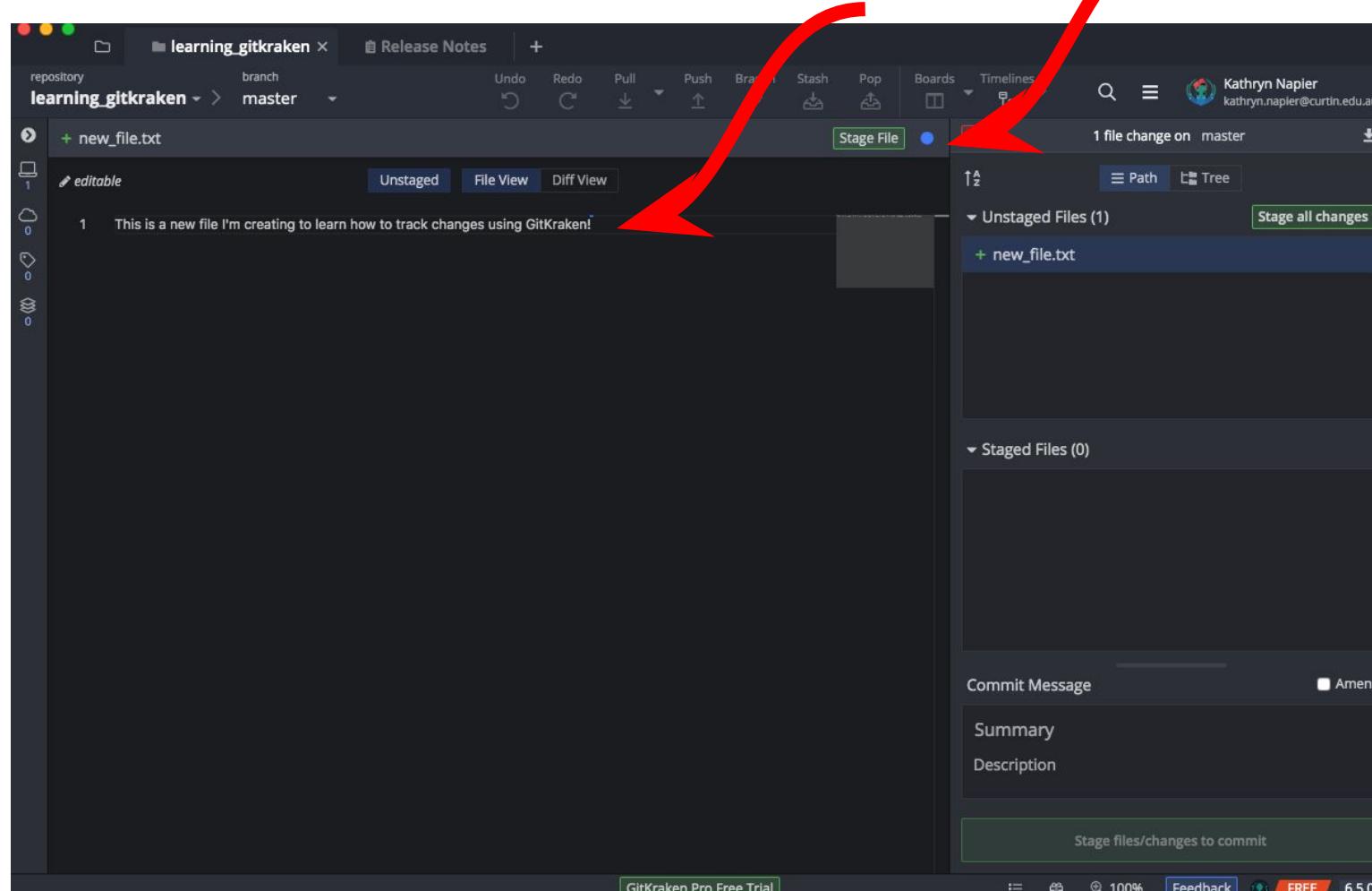
Add a new file in your project folder: right click in the blank 'commit' screen



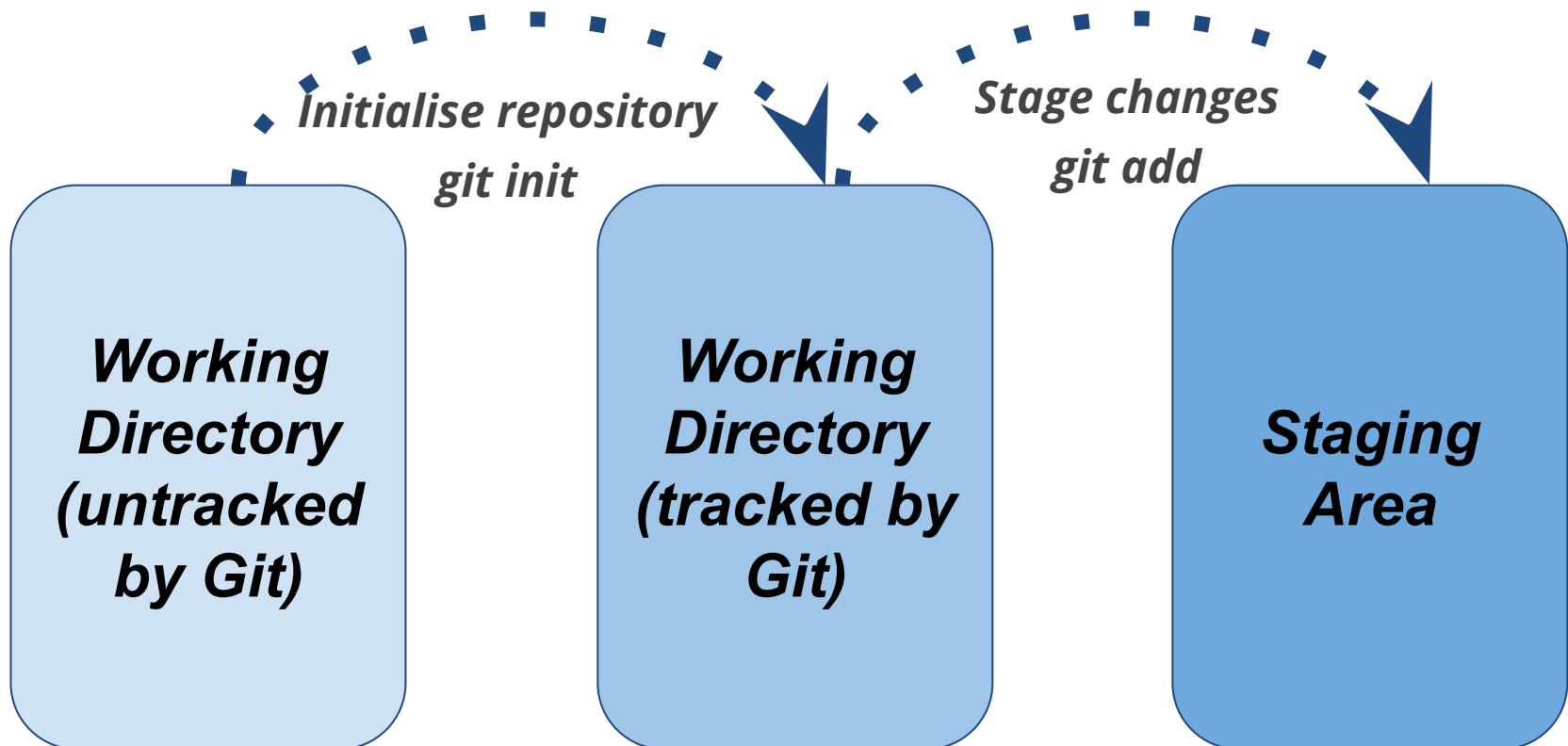
Name your file 'new_file.txt'



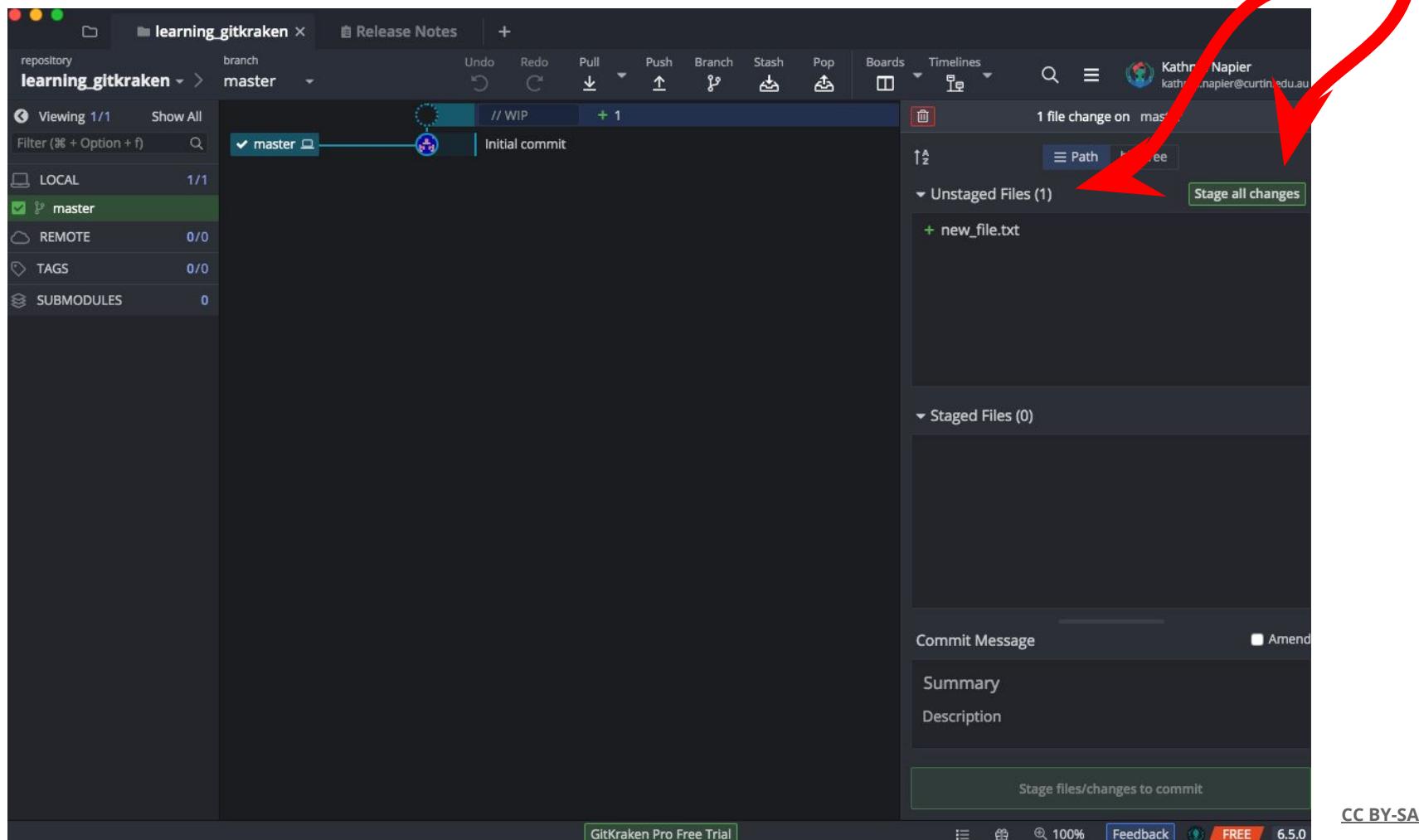
Type in some text, then click the blue circle to save the changes



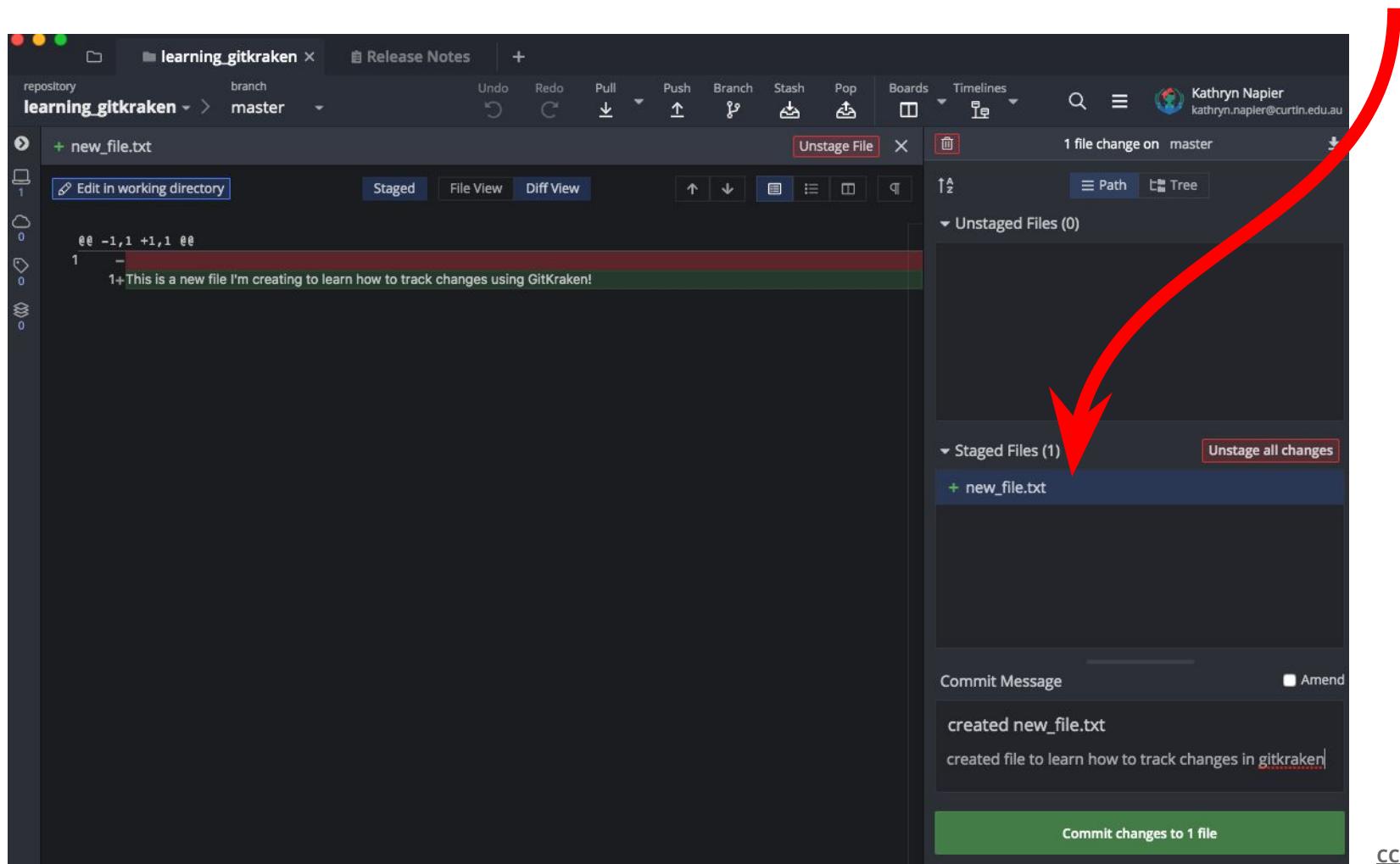
The Staging Area is where you add the files to be committed to Git for version tracking



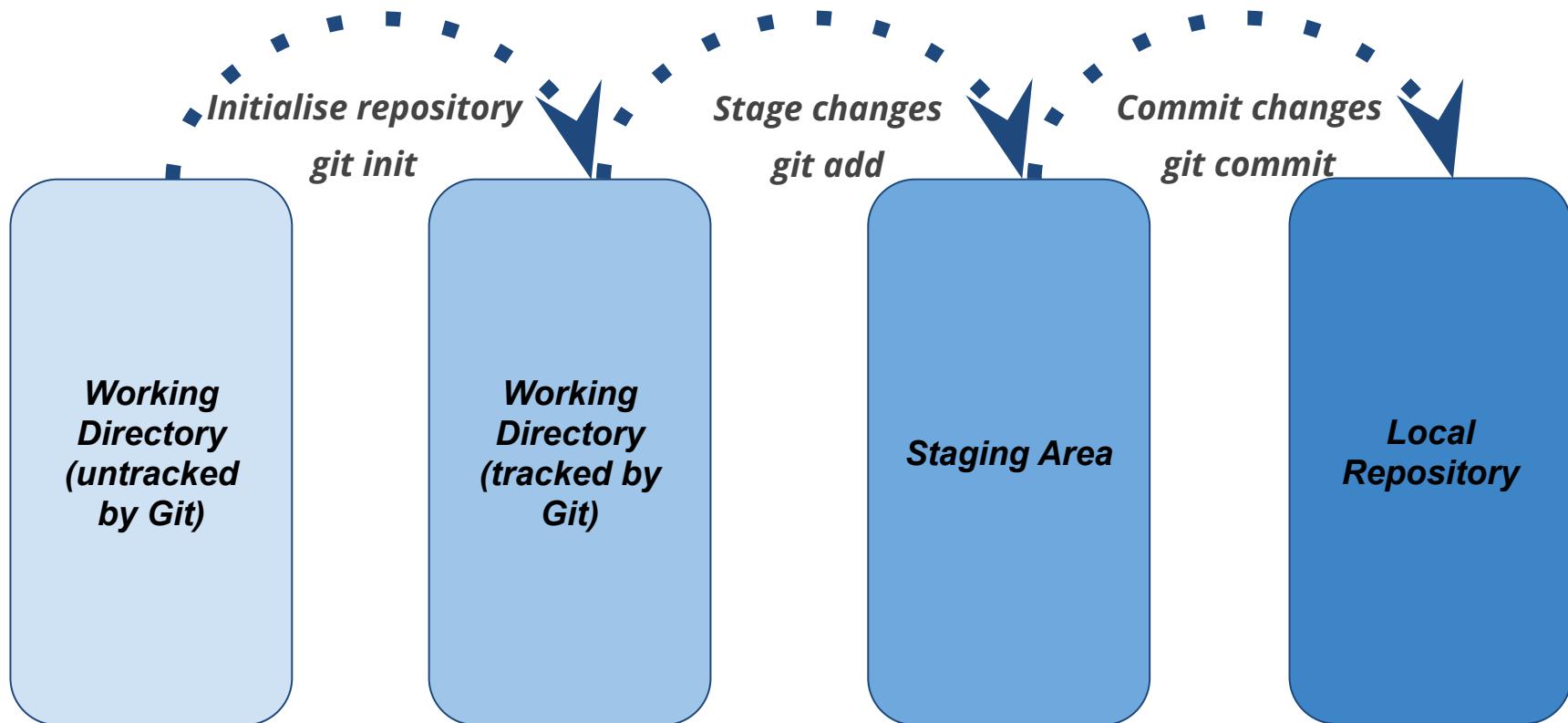
The file has been saved, but is 'Unstaged'. Click 'Stage all changes'



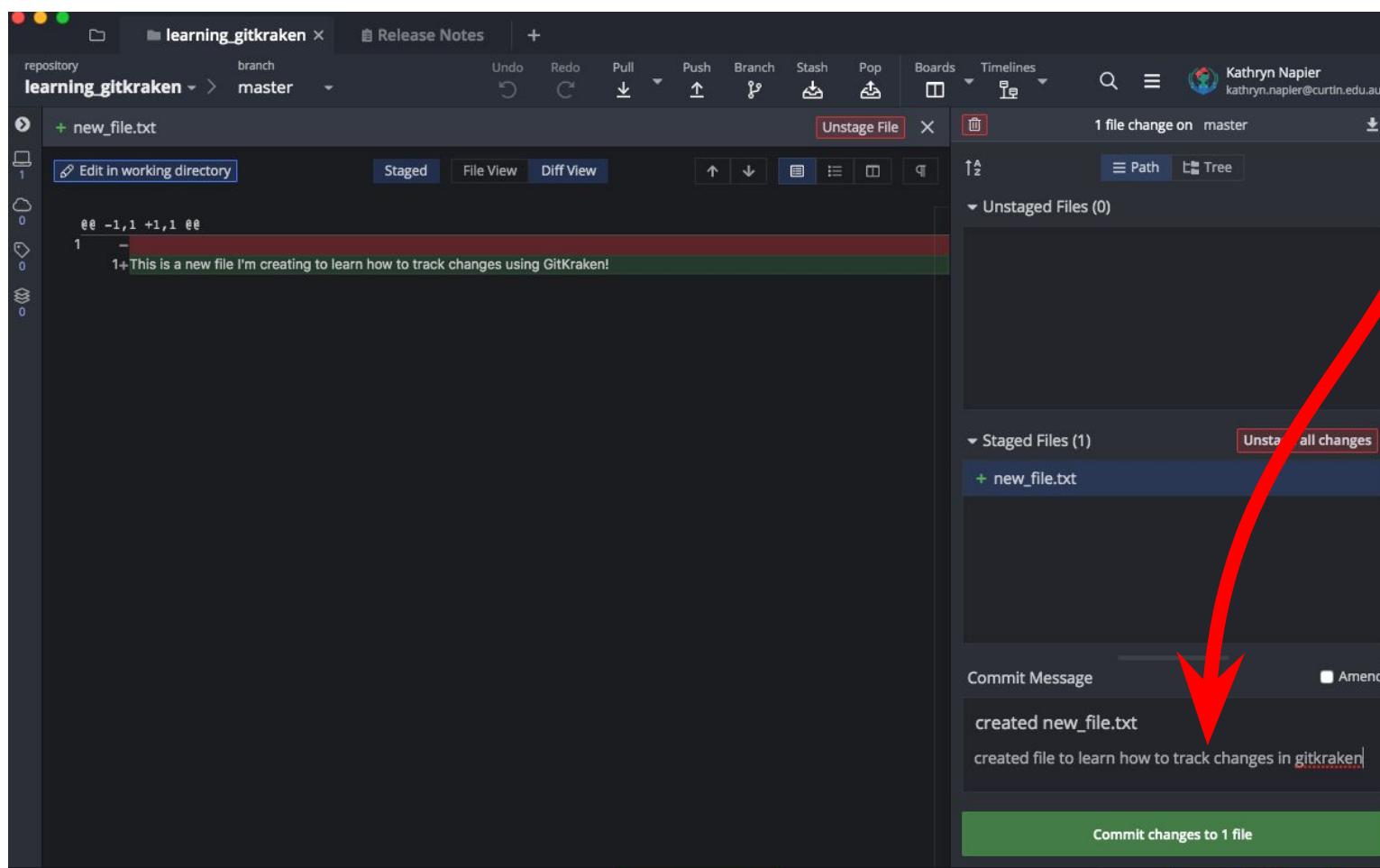
You can click on the file to see the 'Diff View'



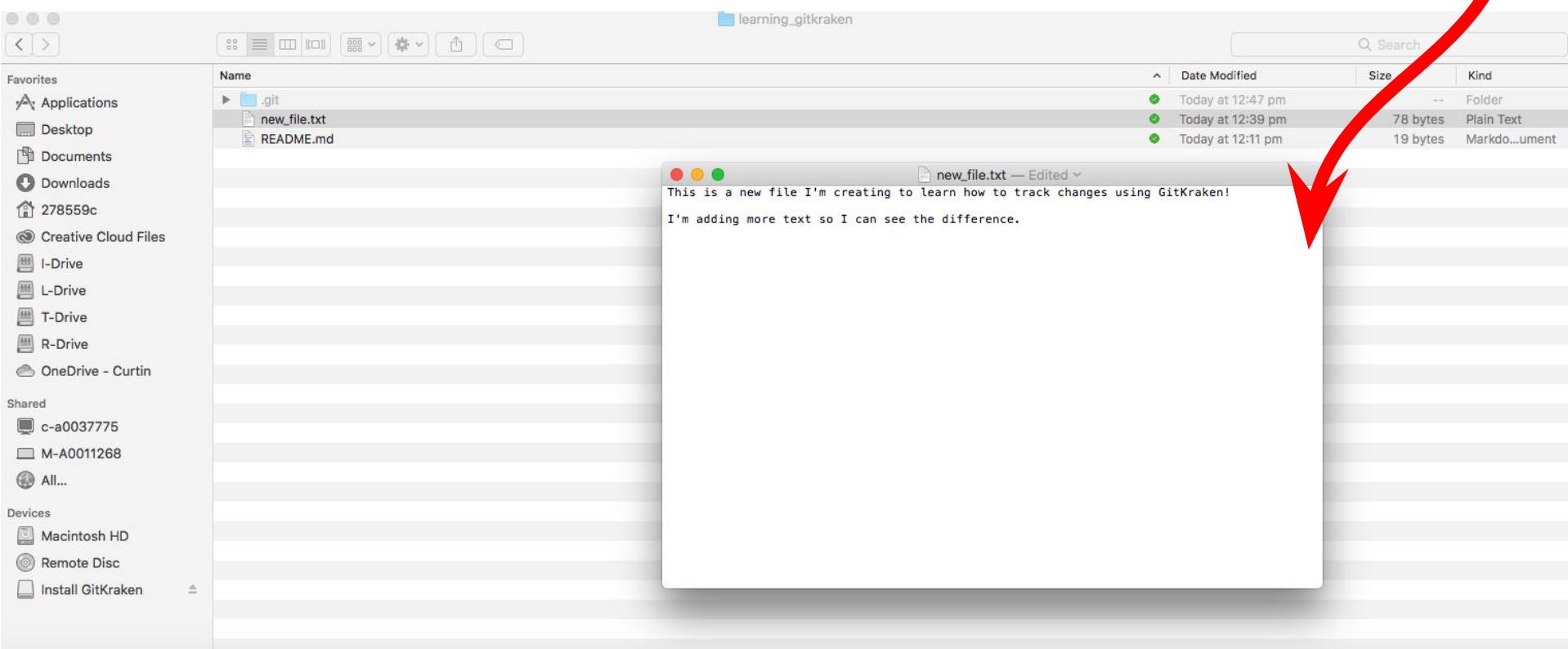
Git records all commits, so you can track versions and changes over time



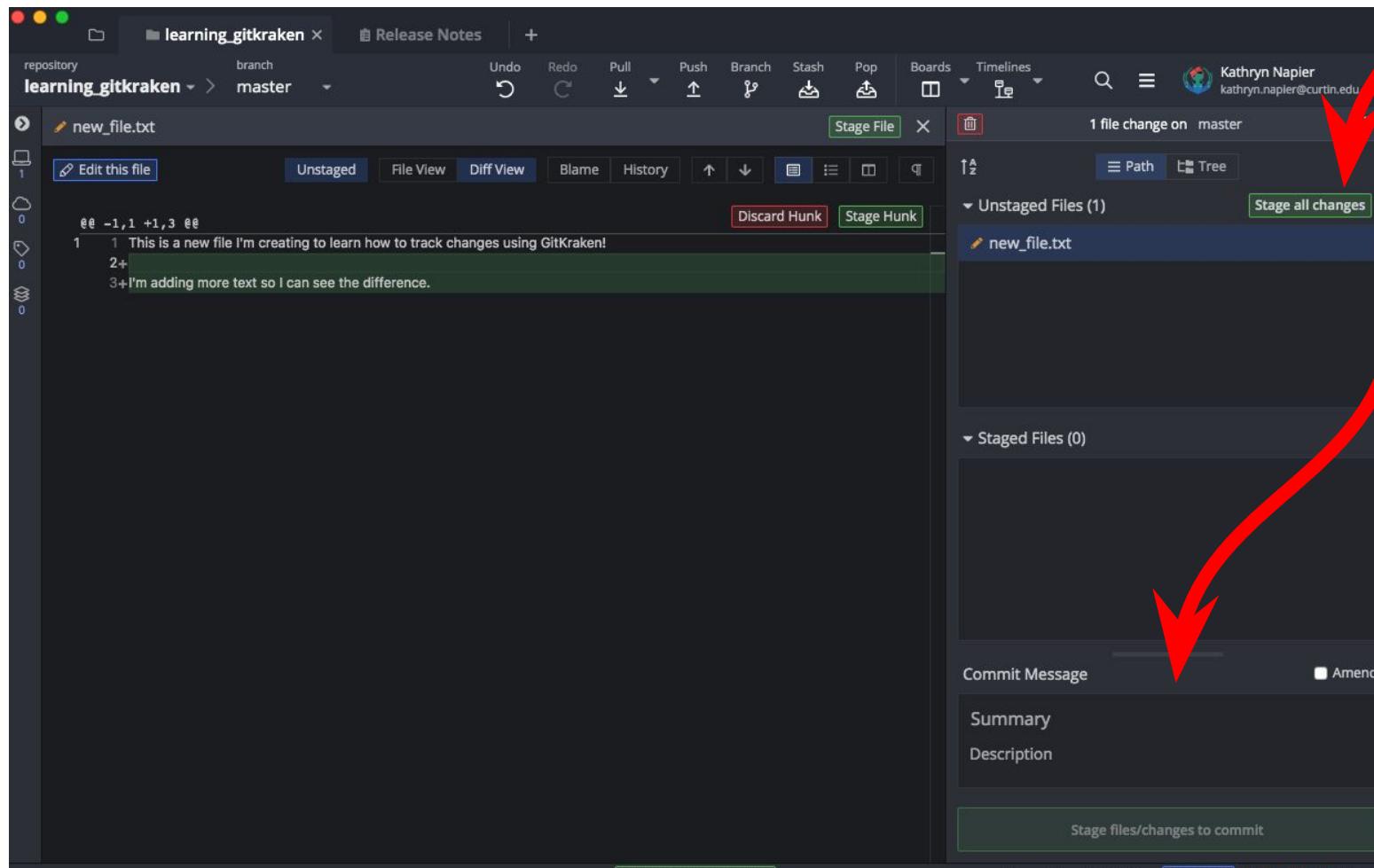
Next step is to 'Commit' the changes. Include a 'Summary' and a 'Description' then click 'Commit changes to 1 file'



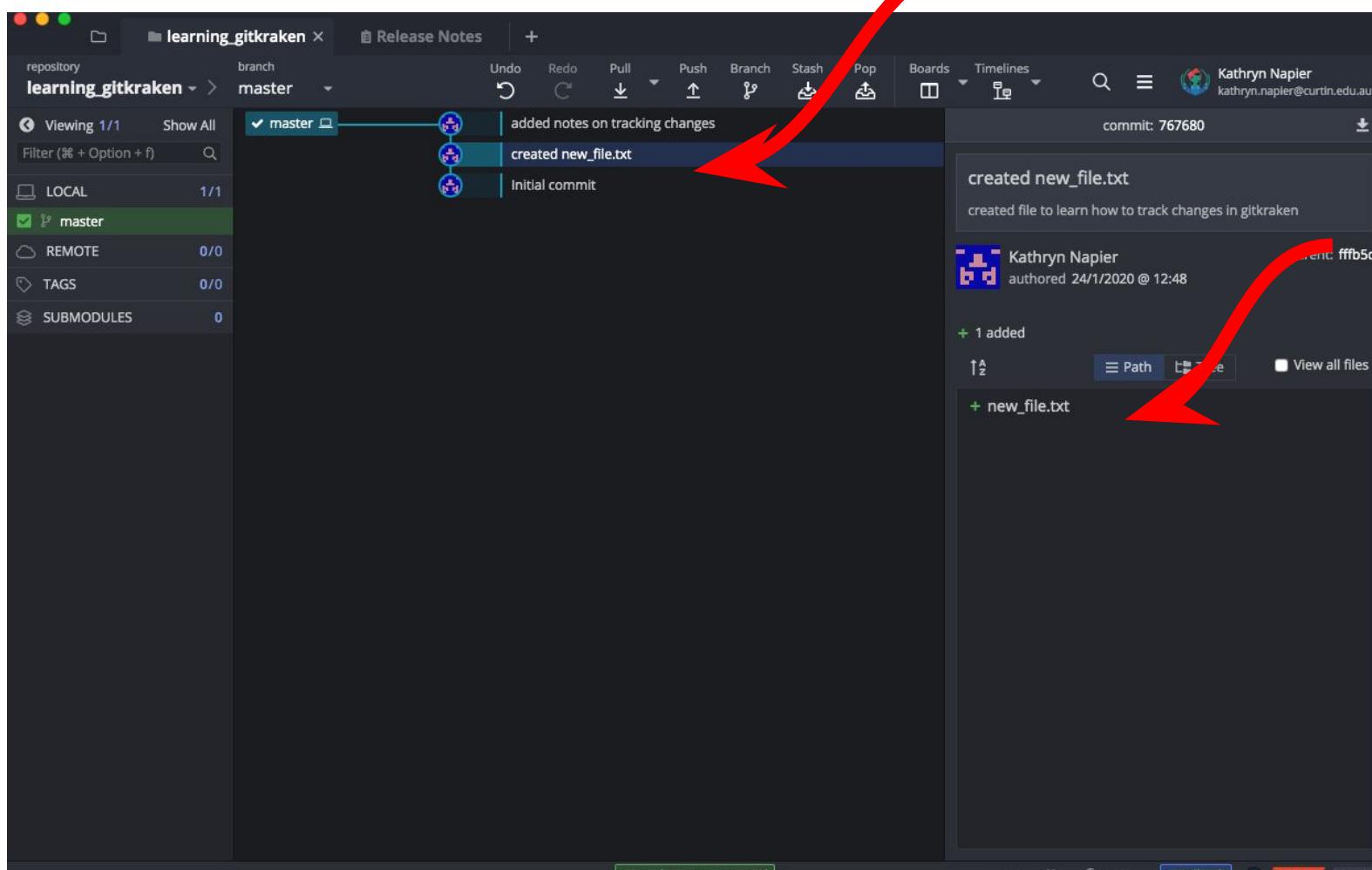
Navigate to your project folder, open the file and add some more text and save the file



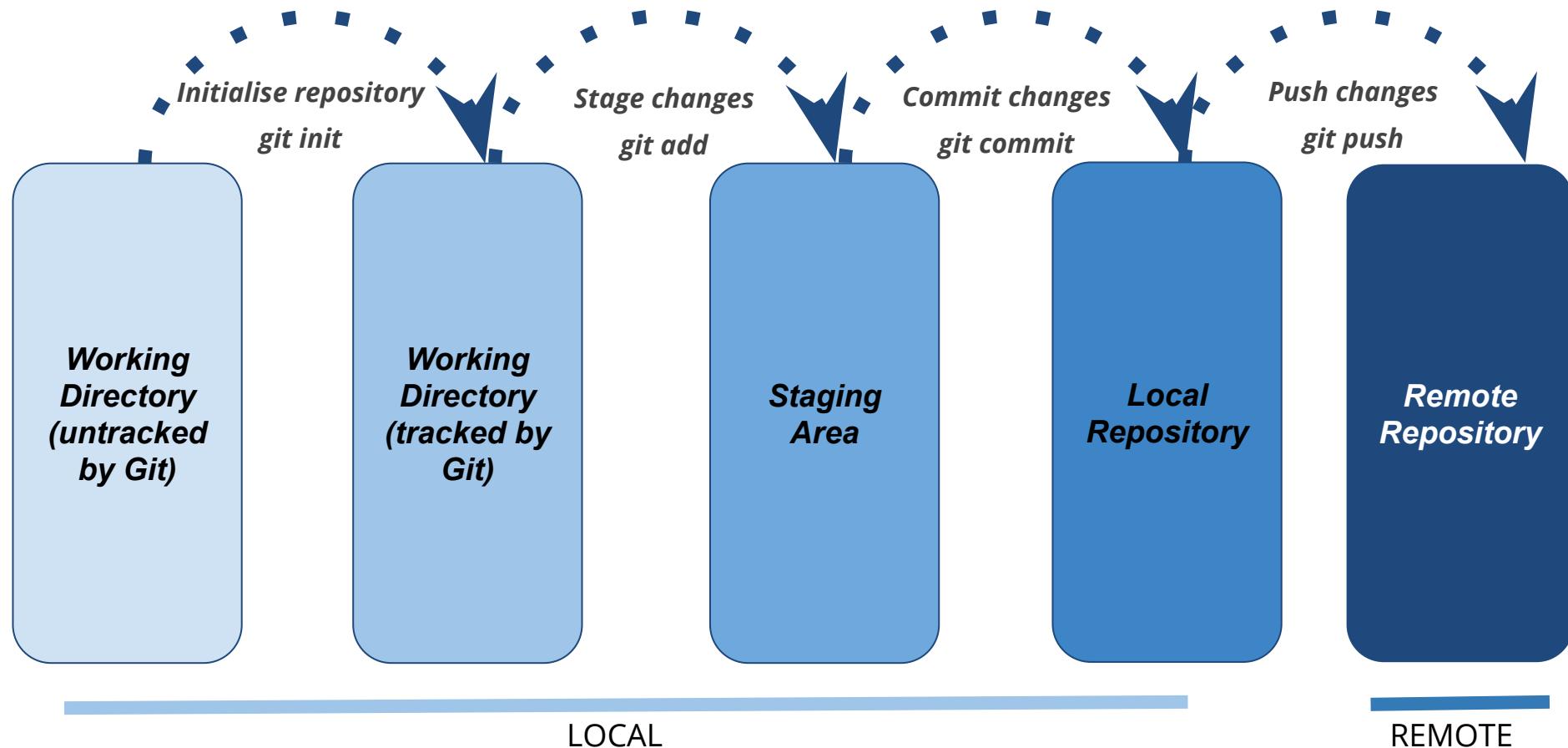
Go back to GitKraken and follow the previous steps to stage and commit the latest changes



You can look at previous commits to see the changes made



What if you need to collaborate?



LOCAL

REMOTE

Why Github

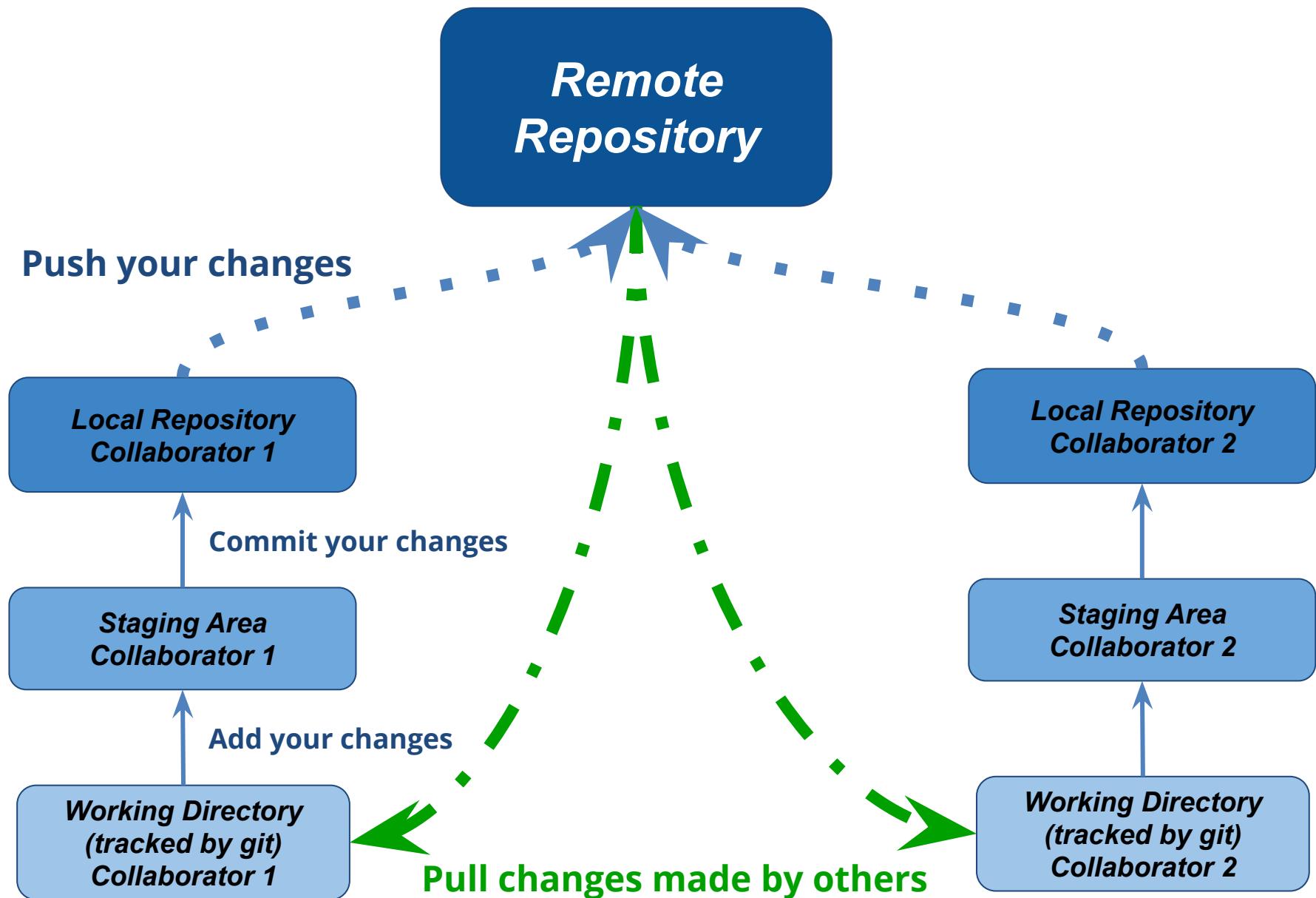


- Remote repository
- Version Control
- Visible code and reproducibility
- Open code and reuse
- Collaborative code development
- Open code development

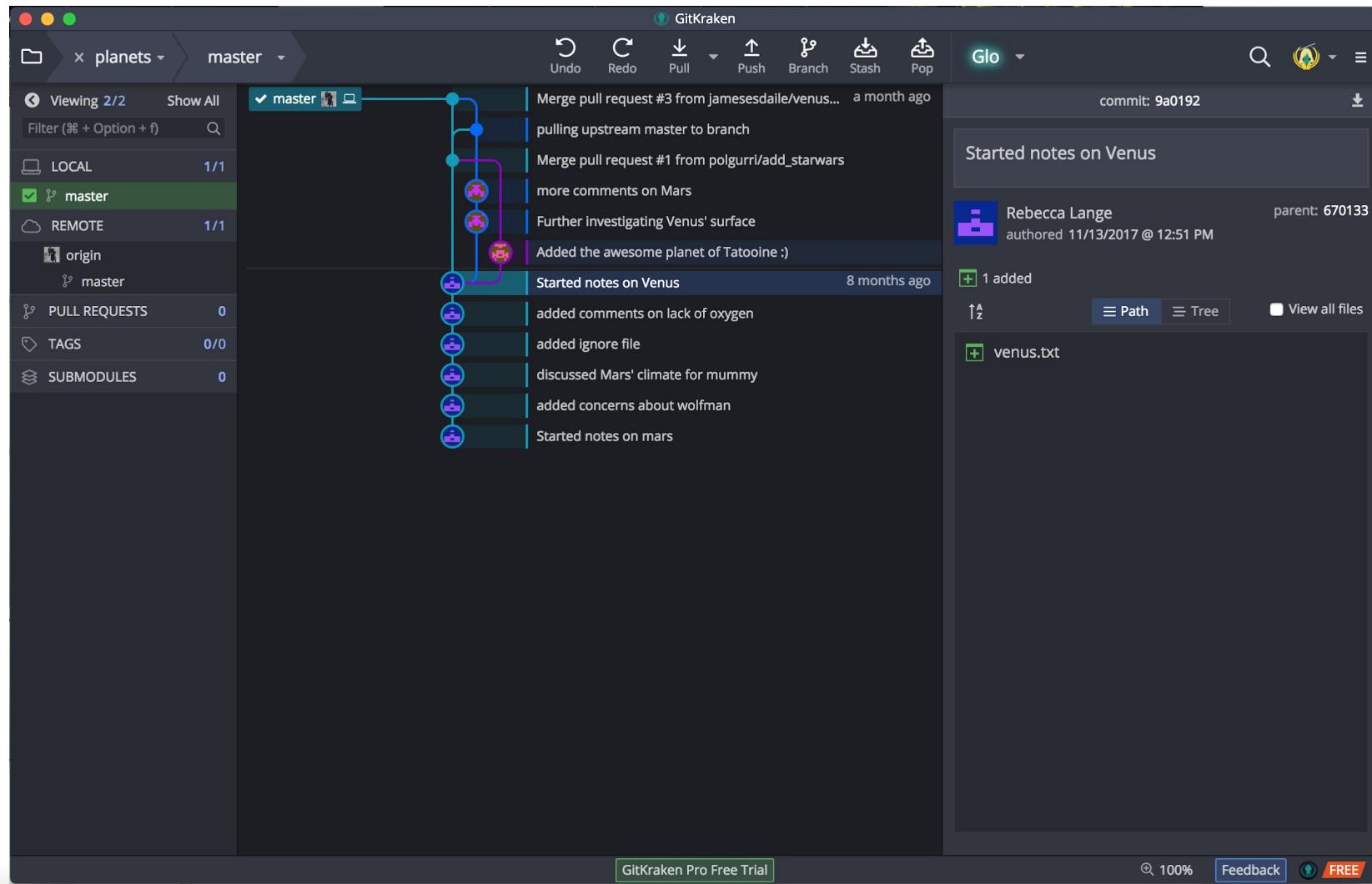


Github and Remote Repositories

CORE | Skills



Multiple collaborators



GitKraken Tutorials

<https://www.gitkraken.com/learn-git>

Github Git Cheatsheet

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

1. Tools used
2. Navigating your computer
3. Version control
4. Reproducibility
5. Some good code etiquette hints
 - a. Programming “Rules of Thumb”
 - b. Make code readable and understandable
 - c. Good practice
6. Where to find help
 - a. Error messages
 - b. Google
 - c. Stackoverflow
 - d. Documentation
 - e. Code review groups

Programming "Rules of Thumb"

Programming Rules of Thumb

1. K.I.S.S. (Keep It Simple, Stupid)
 - a. Subprograms should do precisely ONE conceptual task and no more.
 - b. If a problem can be decomposed into two or more independently solvable problems, do so.
2. Rule of Three
 - a. When you copy/paste a piece of code 3 or more times turn it into a function.
3. 90-90 rule (failure to anticipate the hard parts)
 - a. "The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time."
 - b. —Tom Cargill, Bell Labs
4. Efficiency vs clarity (chasing false efficiency)
 - a. Never sacrifice clarity for some perceived efficiency.
5. Naming of things
 - a. Naming conventions are there to make code easier to read

"Programs must be written for people to read, and only incidentally for machines to execute." – Harold Abelson, Structure and Interpretation of Computer Program

A style guide is about **consistency**. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is the most important.

[PEP8 style guide]

Why care?

- provides consistency
- makes code easier to read
- makes code easier to write
- makes it easier to collaborate

[[Beautify your R code](#) by Saskia Freytag]

Python

- ❖ Python Enhancement Proposals
<https://www.python.org/dev/peps/#numerical-index>
- ❖ PEP 8 -- Style Guide for Python Code

R [[Beautify your R code](#) by Saskia Freytag]

- ❖ tidyverse style guide
 - most comprehensive, underscore for naming conventions
- ❖ Advanced R style guide
 - fairly comprehensive, underscore for naming conventions
- ❖ Google style guide
 - first of its kind, CamelCase for naming conventions

Create readable code

```
a="""Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
"""
```

Python was designed to be readable

Code-blocks are defined by indentation

Line continuations are not required

Syntax is human readable

```
lines = a.split('\n') # \n is the newline character  
num_lines = len(lines)  
  
nwords = 0  
for line in lines:  
    words = line.split()  
    nwords += len(words)
```

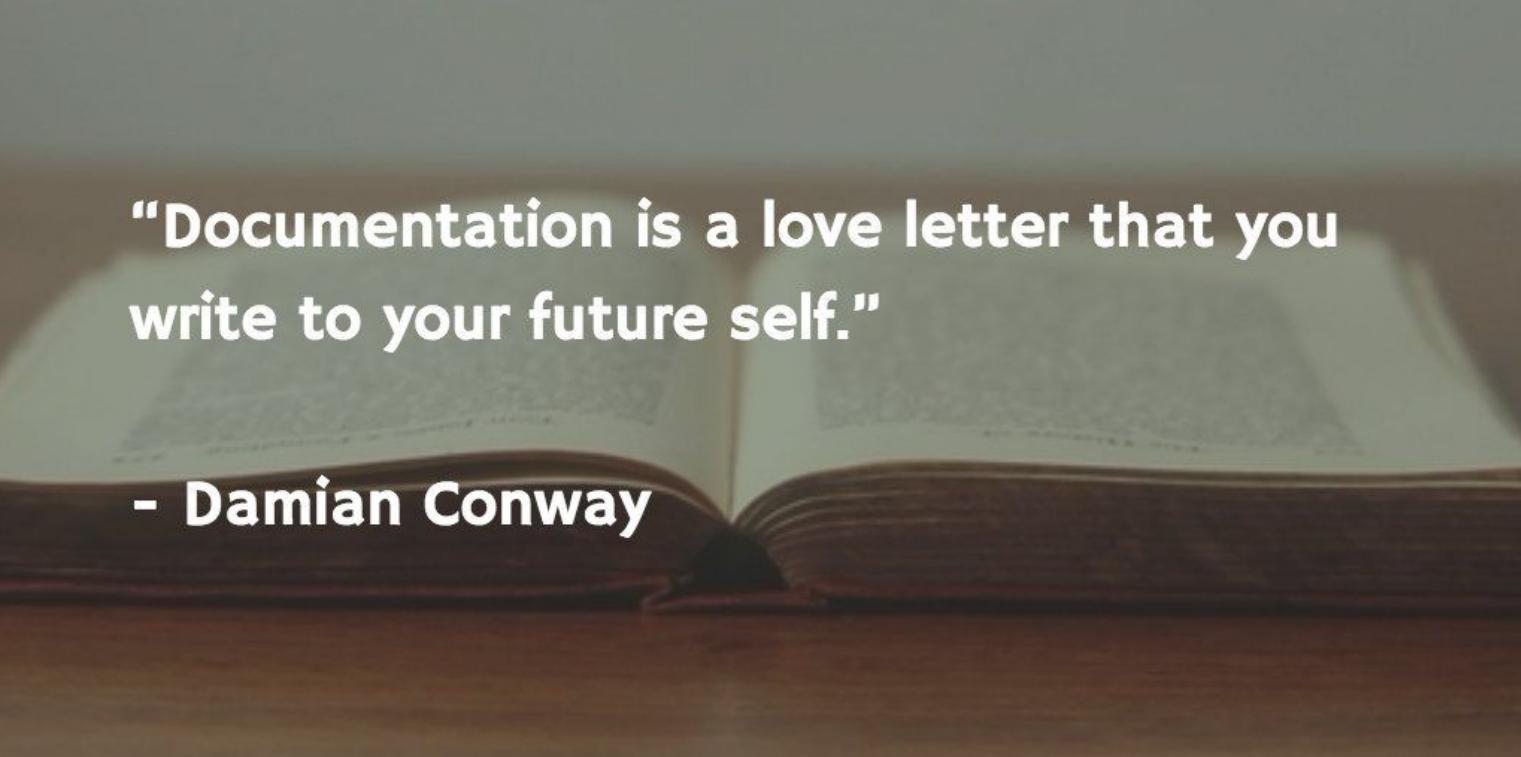
```
def do_lots_of_things(option1, # required  
                      option2=0, # not required, has default value  
                      option3=1,  
                      labels=(),  
                      sqlconnection=None,  
                      retries=3,  
                      verbose=False,  
                      do_print=True):  
    pass
```

Use words! Be verbose but not needlessly so.

- nouns for classes and variables,
- verbs for functions,
- (adjectives for decorators?)
- Underscores_for_functions
- CamelCaseForClasses
- ALL_CAPS_FOR_STATIC_VARIABLES

Document your work

CORE | Skills



“Documentation is a love letter that you write to your future self.”

- Damian Conway

Comments are not documentation

CORE | Skills

Writing code is not a story that unfolds and entertains people with twists and character developments. It's a **recipe**.

1. Ingredients for the shopping list ⇒ modules to import
2. Description of techniques ⇒ functions
3. Directions ⇒ code in main scope

Documentation is for people **using** the code (regular folks)

Documentation describes the ingredients and what kind of cakes are made.

Comments are for people **reading** the code (ie developers and future you)

Comments are about the cake making process.

DRY or DIE!

Don't Repeat Yourself (Duplication Is Evil)

Duplicated code means duplicated errors and bugs

Write a function, call it many times

Better still,

- write a **module** in Python and **import** this, or
- save your collection of **functions** in a separate .R script and **source** it

The DRY principle - II (or DRO maybe?)

Don't Repeat Others

- (re-) implementing code often means going through the same growth/development curve of bugs and corner cases
- Common problems have common solutions, use them!
- ‘**import**’ / ‘**library**’ your way to success

DRY_examples.ipynb

Having a script that needs to be edited every time it runs is just asking for trouble.

KeepThemSeparated.ipynb

“Finding your bug is a process of confirming the many things that you believe are true – until you find one which is not true.”

–Norm Matloff

The only thing that people write less than documentation is test code.

Pro-tip: Both documentation and test code is easier to write if you do it as part of the development process.

1. Write function definition and basic docstring
2. Write function contents
3. Write test to ensure that function does what the docstring claims.
4. Update code and/or docstring until (3) is true.

Whatever you currently do to convince yourself that your code works is a test!

Everytime you find a bug or some corner case, write a test that will check it.

Making mistakes doesn't make you a bad person, making the **same mistake** over and over does.

Testing in R:

<http://r-pkgs.had.co.nz/tests.html>

Testing in Python:

<https://docs.python-guide.org/writing/tests/>

Testing.ipynb



Hadley Wickham

@hadleywickham

Follow



The only way to write good code is to write tons of shitty code first. Feeling shame about bad code stops you from getting to good code

6:11 AM - 17 Apr 2015

- Writing good code takes practice.
- Reuse things that work for you.
- Develop a support group you can call on for help.
 - We have weekly meet-up groups like hacky-hour
- Share your code on GitHub or similar, with documentation, so others can benefit from your work.
 - People can help you debug by reporting issues and submitting bug fixes via pull requests
 - Remember sharing your code on github does not mean you can be held accountable for its maintenance.

*Publish your code and cite that of others.

Where to get help

CORE | Skills

- Read error messages, they are there to assist you
- Google is your friend
- Stackoverflow is your bible
 - Basically Yahoo answers/Quora for code
 - Most questions already exist

← → ⌂ ⌄ https://stackoverflow.com/questions/tagged/python

stackoverflow Products [python] Search

Questions tagged [python]

Ask Question

Home PUBLIC Stack Overflow Tags Users Jobs TEAMS What's this? Free 30 Day Trial

Python is a multi-paradigm, dynamically typed, multipurpose programming language, designed to be quick (to learn, to use, and to understand), and to enforce a clean and uniform syntax. Two similar but incompatible versions of Python are commonly in use, Python 2.7 and 3.x. For version-specific Python questions, add the [python-2.7] or [python-3.x] tag. When using a Python variant or library (e.g. Jython, PyPy, Pandas, Numpy), please include it in the tags.

Unwatch Tag Ignore Tag Learn more... Improve tag info Top users Synonyms (4) python jobs

131,090 questions Newest Active Bountied 48 Unanswered More Filter

612 How to test multiple variables against a value?
votes I'm trying to make a function that will compare multiple variables to an integer and output a string of three letters. I was wondering if there was a way to translate this into Python. So say: `x = 0 ...`
24 answers python if-statement comparison match boolean-logic asked Feb 27 '13 at 12:26 user1877442 6,311 ● 3 ● 10 ● 5 207k views

3095 Understanding slice notation
votes I need a good explanation (references are a plus) on Python's slice notation. To me, this notation needs a bit of picking up. It looks extremely powerful, but I haven't quite got my head around it...
32 answers python list slice iterable asked Feb 3 '09 at 22:31 Simon 62k ● 24 ● 80 ● 116 1.6m views

533 Asking the user for input until they give a valid response
votes I am writing a program that accepts an input from the user. #note: Python 2.7 users should use 'raw_input', the equivalent of 3.X's 'input' `age = int(input("Please enter your age: "))` if `age >= 18`:...
19 answers python validation loops python-3.x user-input community wiki 11 revs, 8 users 67%

Where to get help

CORE | Skills

- Read error messages, they are there to assist you
- Google is your friend
- Stackoverflow is your bible
 - Basically Yahoo answers/Quora for code
 - Most questions already exist
- Documentation
 - Inbuilt doc strings
 - Online manuals
 - Tutorials
 - Awesome lists on github

Inbuilt doc strings

CORE | Skills

```
In [1]: help(len)
Help on built-in function len in module builtins:

len(...)
    len(object) -> integer
```

Return the number of items of a sequence **or** mapping.

```
In [2]: len?
Type:      builtin_function_or_method
String form: <built-in function len>
Namespace: Python builtin
Docstring:
len(object) -> integer
```

Return the number of items of a sequence **or** mapping.

Jupyter Python Playground (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [1]: `import tensorflow as tf`

In []: `X = tf.placeholder()`

Signature: `tf.placeholder(dtype, shape=None, name=None)`
Docstring:
Inserts a placeholder for a tensor that will be always fed.

Important: This tensor will produce an error if evaluated. Its value must be fed using the `feed_dict` optional argument to `Session.run()`, `Tensor.eval()`, or `Operation.run()`.

For example:

```python

The screenshot shows the Python 3.8.1 documentation page. At the top, there's a browser header with navigation icons, a URL bar showing <https://docs.python.org/3/>, and a search bar. Below the header, the Python logo and "Python » English" are on the left, along with dropdown menus for version (3.8.1) and documentation type.

**Download**  
Download these documents

**Docs by version**

- Python 3.9 (in development)
- Python 3.8 (stable)
- Python 3.7 (stable)
- Python 3.6 (security-fixes)
- Python 3.5 (security-fixes)
- Python 2.7 (EOL)
- All versions

**Other resources**

- PEP Index
- Beginner's Guide
- Book List
- Audio/Visual Talks
- Python Developer's Guide

**Python 3.8.1 documentation**

Welcome! This is the documentation for Python 3.8.1.

**Parts of the documentation:**

- What's new in Python 3.8?**  
or all "What's new" documents since 2.0
- Tutorial**  
start here
- Library Reference**  
keep this under your pillow
- Language Reference**  
describes syntax and language elements
- Python Setup and Usage**  
how to use Python on different platforms
- Python HOWTOs**  
in-depth documents on specific topics
- Indices and tables:**
- Global Module Index**  
quick access to all modules
- General Index**
- Installing Python Modules**  
installing from the Python Package Index & other sources
- Distributing Python Modules**  
publishing modules for installation by others
- Extending and Embedding**  
tutorial for C/C++ programmers
- Python/C API**  
reference for C/C++ programmers
- FAQs**  
frequently asked questions (with answers!)
- Search page**  
search this documentation

# Github: Awesome lists

CORE | Skills

Curated list of Python resources for data science.

awesome awesome-list data-science datascience python deep-learning data-analysis data-visualization data-mining machine-learning  
artificial-intelligence deeplearning statistics bayes

325 commits

1 branch

0 packages

0 releases

4 contributors

View license

Branch: master ▾

New pull request

Create new file

Upload files

Find file

Clone or download ▾

|                                                                                        |                  |                                  |
|----------------------------------------------------------------------------------------|------------------|----------------------------------|
|  r0f1 | Update README.md | Latest commit 12d5588 2 days ago |
|       | CONTRIBUTING.md  | Update CONTRIBUTING.md           |
|       | INTERESTING.md   | Update INTERESTING.md            |
|       | LICENSE          | License                          |
|       | README.md        | Update README.md                 |

 README.md

## Awesome Data Science with Python

A curated list of awesome resources for practicing data science using Python, including not only libraries, but also links to tutorials, code snippets, blog posts and talks.

### Core

pandas - Data structures built on top of numpy.

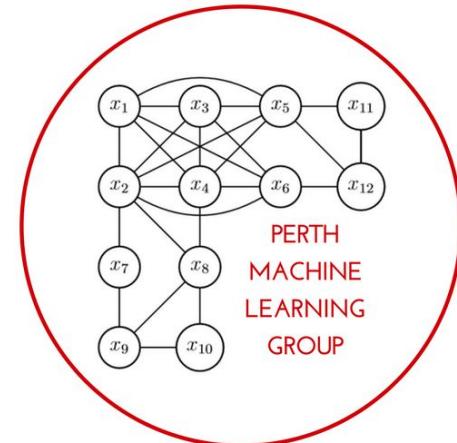
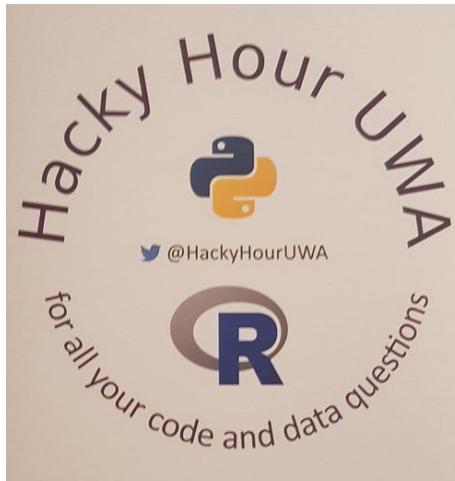
scikit-learn - Core ML library.

matplotlib - Plotting library.

# Where to get help

CORE | Skills

- Read error messages, they are there to assist you
- Google is your friend
- Stackoverflow is your bible
  - Basically Yahoo answers/Quora for code
  - Most questions already exist
- Documentation
  - Inbuilt doc strings
  - Online manuals
  - Tutorials
  - Awesome lists on github
- Join or start a code review group
  - Meetup
  - Hacky hour



## Perth Django and Python Developers Western Australian R Group



Thank you.

Questions?

What if you need to **collaborate**?

# Create a Github account

CORE | Skills

- Go to: <https://github.com/> and click 'Sign up'



Join GitHub

## Create your account

**Username \***

**Email address \***

**Password \***

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more](#).

**Email preferences**

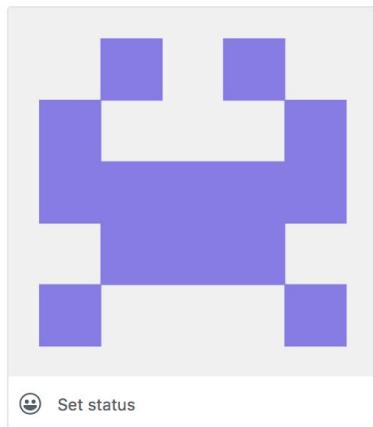
Send me occasional product updates, announcements, and offers.

**Verify your account**

# Create a Github repo

CORE | Skills

- Click on 'New' to create a new github repository



kathrynnapier



Senior Data Scientist

Curtin Institute for Computation

Perth, Western Australia

ProTip! Updating your profile with your name, location, and a profile picture helps other GitHub users get to know you. [Edit profile](#) [X](#)

Overview    **Repositories 5**    Projects 0    Packages 0    Stars 0    Followers 0    Following 0

Find a repository...    Type: All ▾    Language: All ▾    [New](#)

 Private    [Star](#)

Jupyter Notebook    Updated yesterday

**SWC-20191204**    [Star](#)

Updated on 6 Dec 2019

**SWC-2019-12-04-curtin**     [Star](#)

Updated on 6 Dec 2019

A screenshot of a GitHub repository list. At the top, there is a 'ProTip!' message about updating your profile. Below it, there are tabs for Overview, Repositories (which is selected), Projects, Packages, Stars, Followers, and Following. There is a search bar for finding repositories, filters for Type and Language, and a prominent green 'New' button. The repository list shows three entries: one private repository (redacted name) and two public repositories ('SWC-20191204' and 'SWC-2019-12-04-curtin'). Each repository entry includes its name, a 'Star' button, and an update timestamp. A red arrow points to the 'New' button.

# Create a Github repo

CORE | Skills

- **Name the repository, set to public (you can work with private repo's in the GitKraken Pro version), DO NOT initialise with a README. Click 'Create repository'**

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

---

Owner                          Repository name \*

 kathrynnapier / learn\_gitkraken ✓

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

Description (optional)

 **Public**  
Anyone can see this repository. You choose who can commit.

 **Private**  
You choose who can see and commit to this repository.

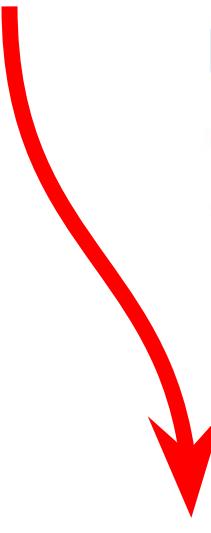
---

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

**Initialize this repository with a README**  
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾      Add a license: **None** ▾ 

**Create repository**



# Create a Github repo

CORE | Skills

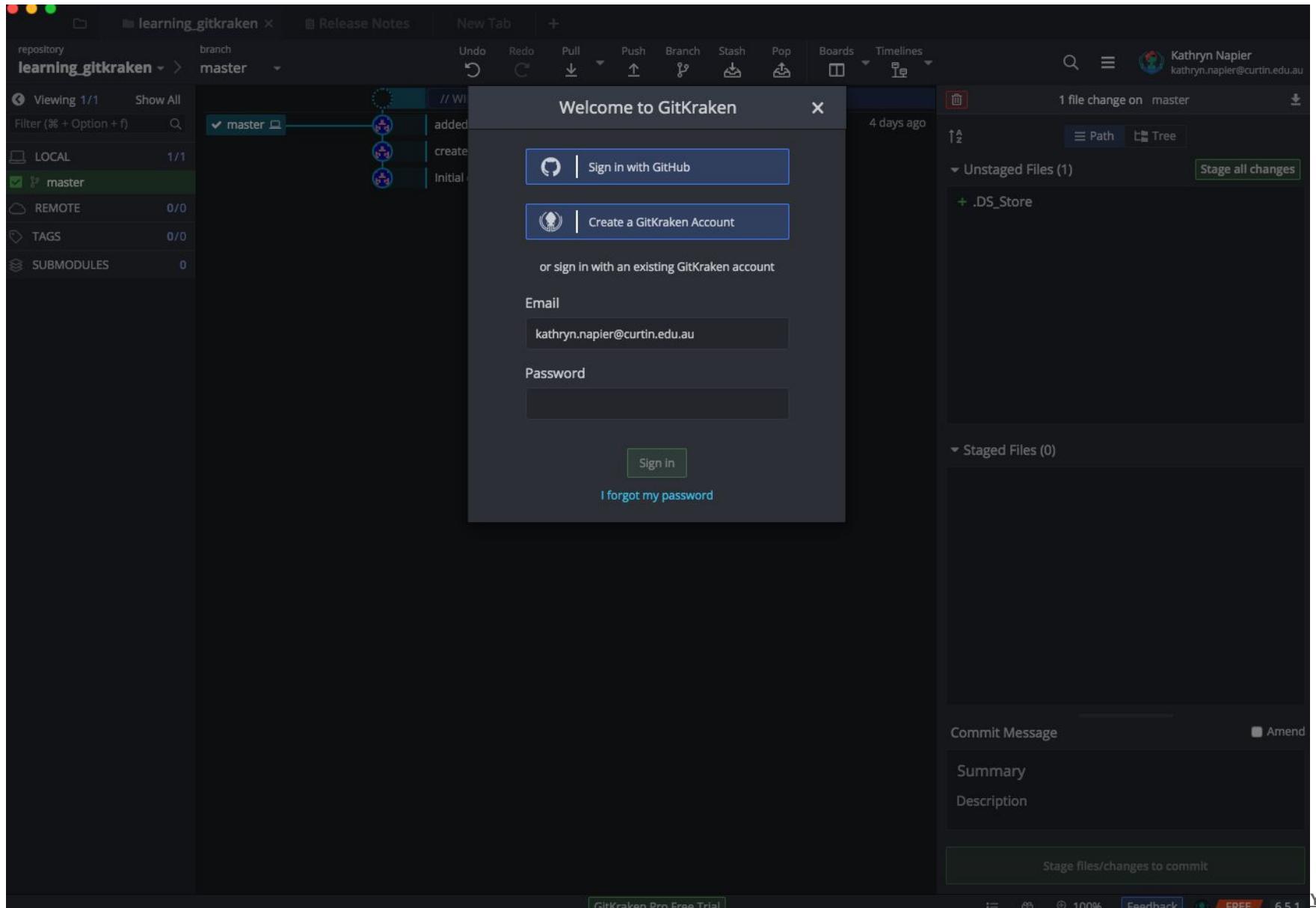
- Copy the HTTPS link

A screenshot of a GitHub repository page for 'kathrynnapier / learn\_gitkraken'. The page shows basic repository statistics: 1 unwatched, 0 stars, 0 forks. Below the header, there are navigation links for Code, Issues (0), Pull requests (0), Actions, Projects (0), Security, Insights, and Settings. A red arrow points from the top right towards the 'HTTPS' clone URL field. The 'HTTPS' tab is selected, displaying the URL 'https://github.com/kathrynnapier/learn\_gitkraken.git'. Below this, instructions advise creating a new file or uploading an existing one, mentioning README, LICENSE, and .gitignore. At the bottom, command-line setup instructions are provided:

```
echo "# learn_gitkraken" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/kathrynnapier/learn_gitkraken.git
git push -u origin master
```

# Sign into GitKraken with Github

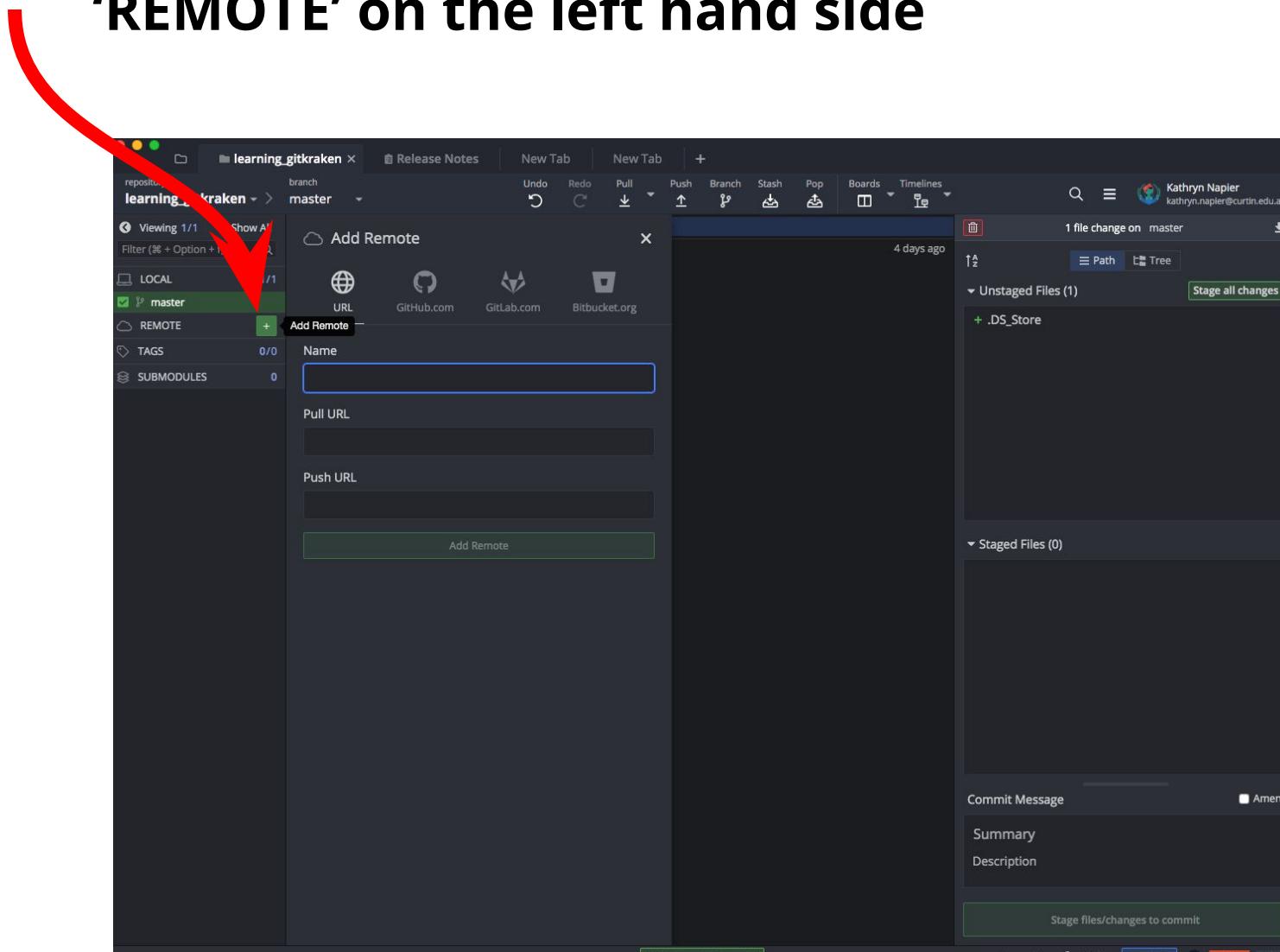
CORE | Skills



# Link your local and remote

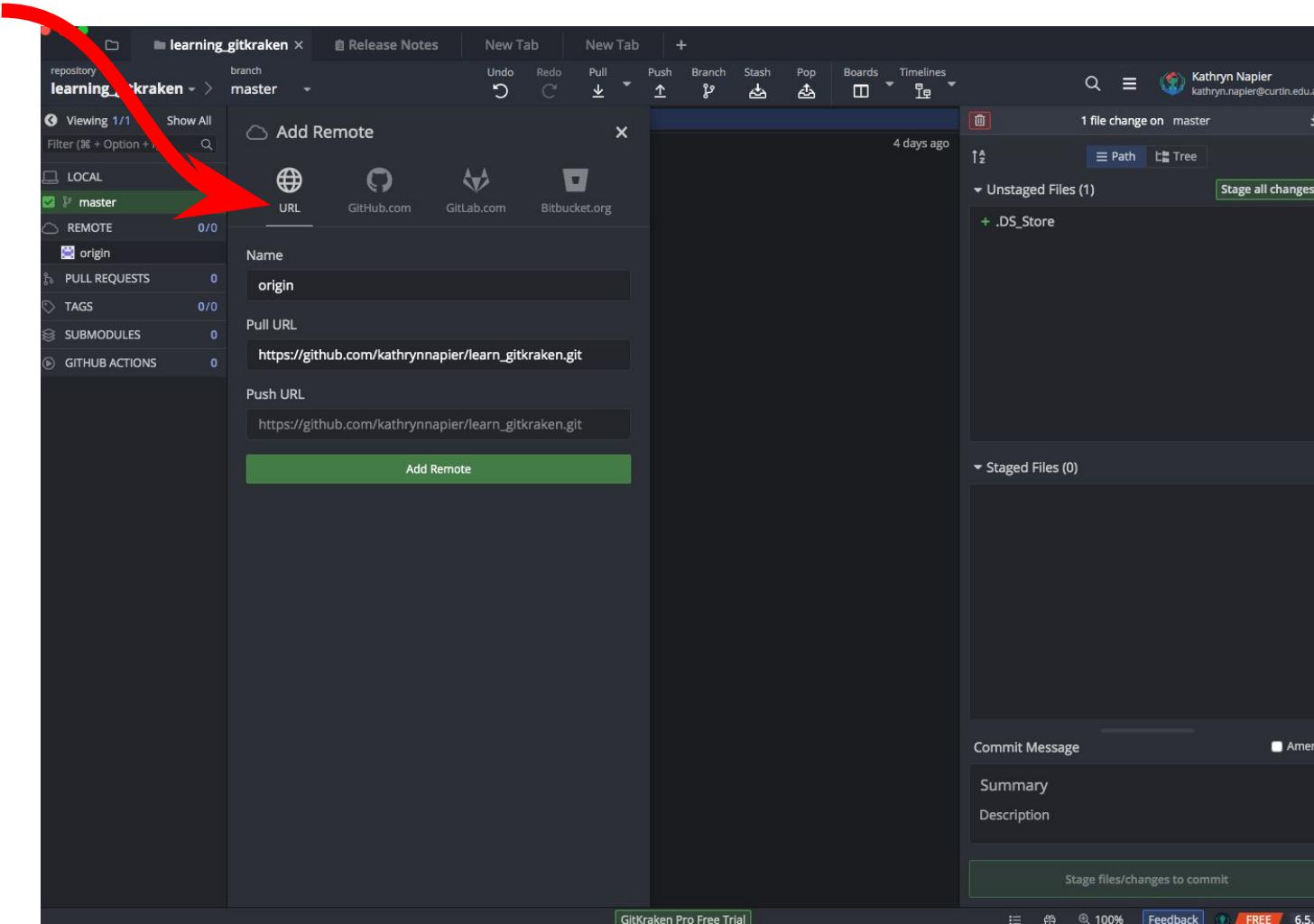
CORE | Skills

- Click the '+' that appears when you hover on 'REMOTE' on the left hand side



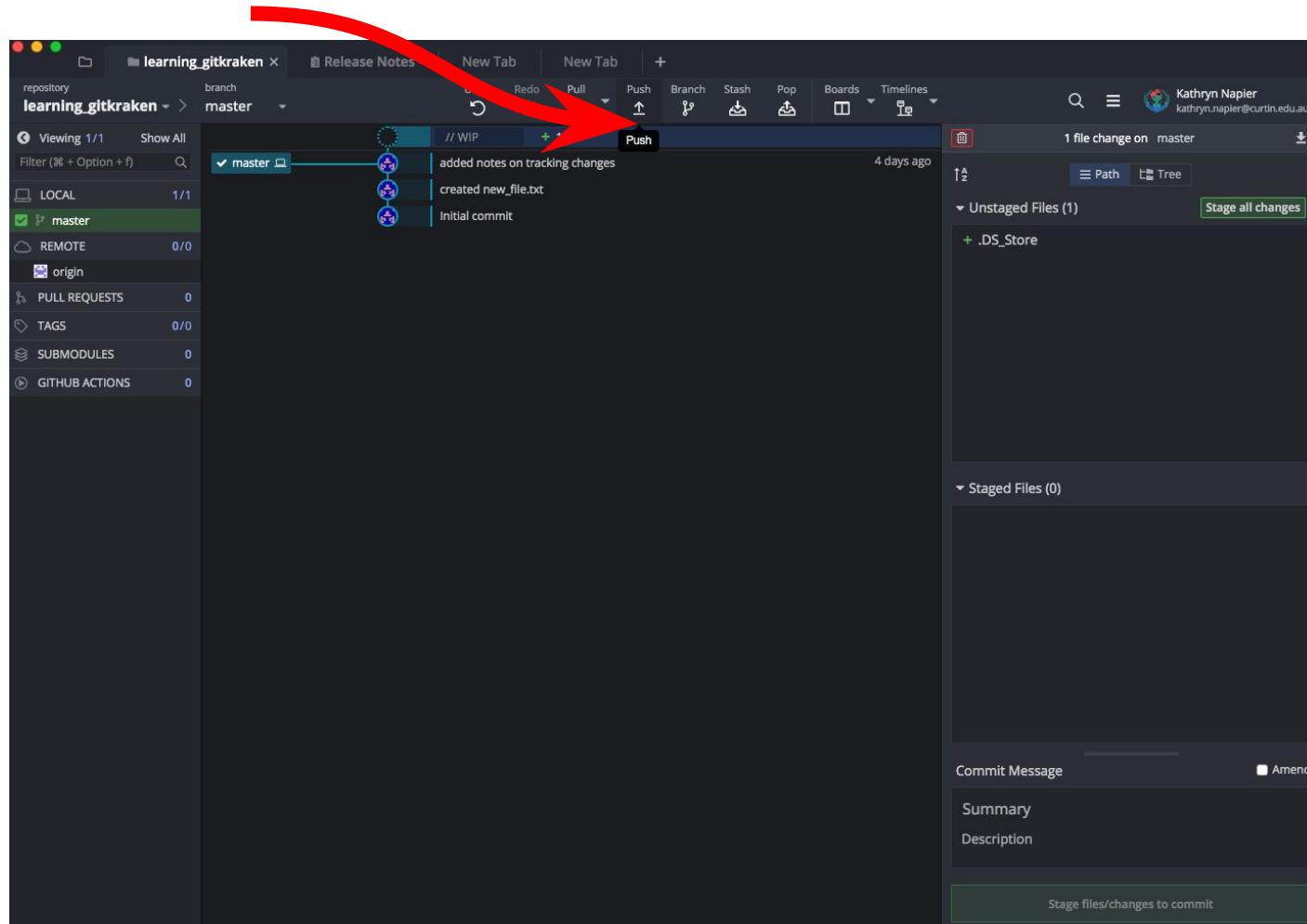
# Link your local and remote

- Click 'URL', set the name to 'origin', and copy the Github url into pull and push URL, then 'Add Remote'



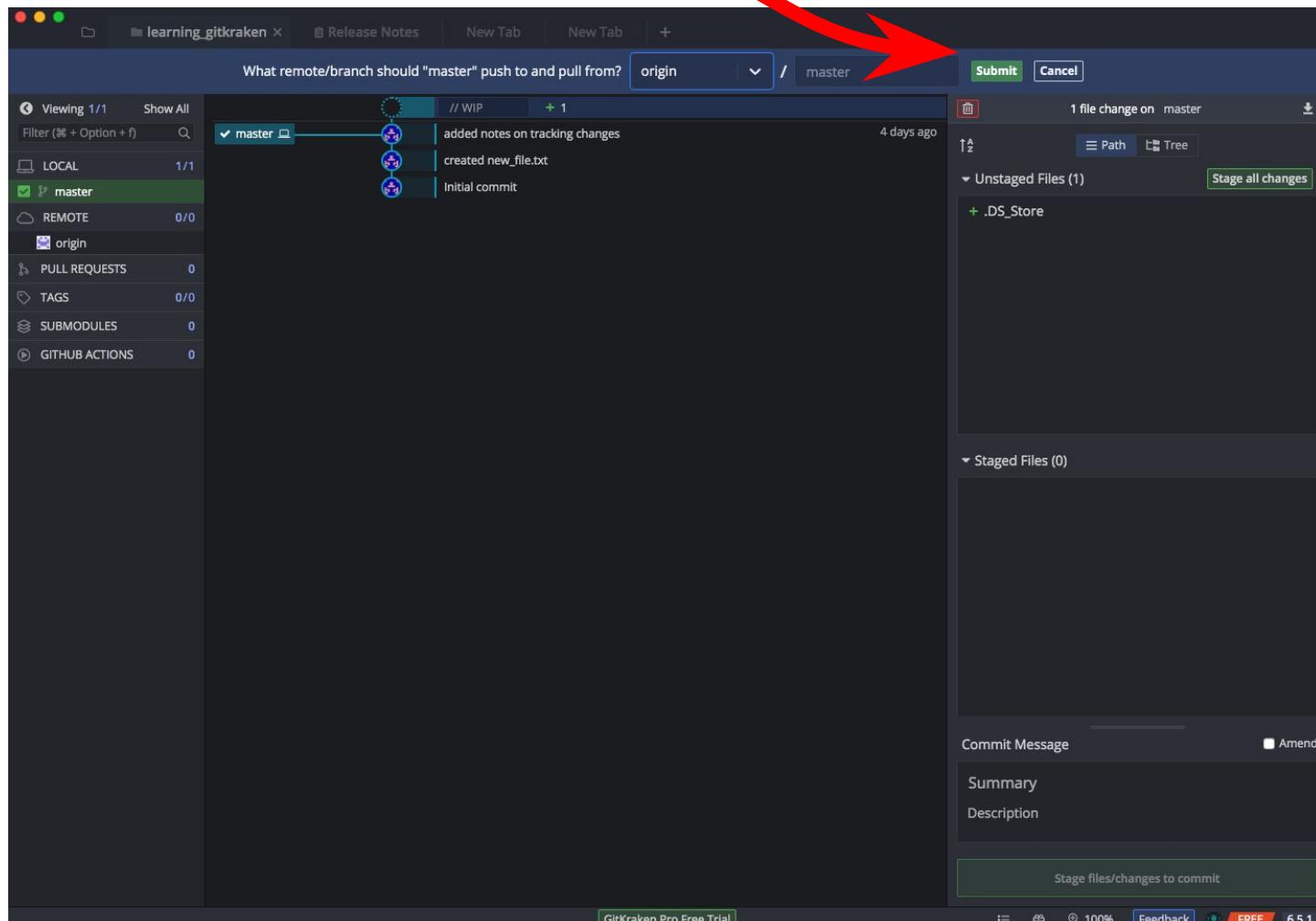
# Push to Github

- Click 'Push'



# Push to Github

- Click 'Submit'



- You can now see your files on the remote repository!

The screenshot shows a GitHub repository page for the user 'kathrynnapier' named 'learn\_gitkraken'. The repository is private, has 1 watch, 0 stars, and 0 forks. The 'Code' tab is selected. A note says 'No description, website, or topics provided.' Below it, there are buttons for 'Edit' and 'Manage topics'. Key statistics show 3 commits, 1 branch, 0 packages, and 0 releases. The commit history lists three entries:

- kathrynnapier added notes on tracking changes (Latest commit d7b6c02 5 days ago)
- Initial commit (5 days ago)
- new\_file.txt added notes on tracking changes (5 days ago)

A file editor for 'README.md' is shown at the bottom, containing the text 'learning\_gitkraken'.