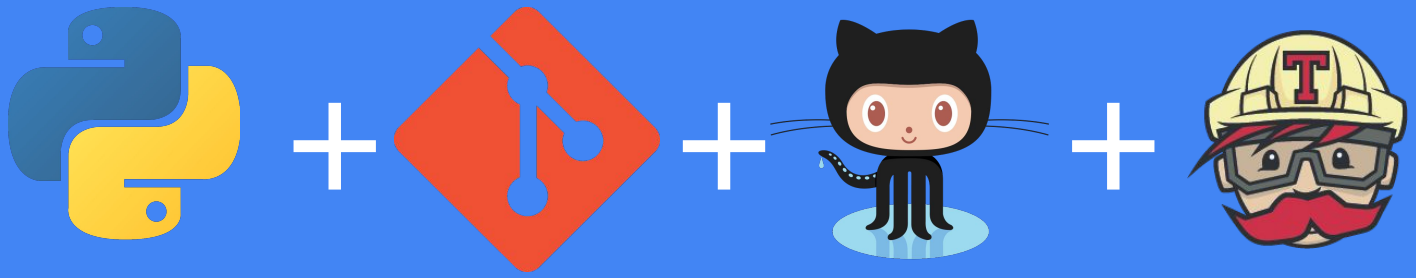


How to bioinformatics better*

Tools for open, reproducible, and reusable computational research

*In Tamas Nagy's opinion (@tlngy, tamas@tamasnagy.com)

Disclaimer:

Bioinformatics is a big field. Use the tools that work best for you and your domain, but these are pretty good set of best practices.

Disclaimer:

Bioinformatics is a big field. Use the tools that work best for you and your domain, but these are pretty good set of best practices.

Why care about open and reproducible computational research?

Why not just copy around one-off Matlab scripts?

- ➔ Proprietary languages limit uptake due to licensing/cost
- ➔ Changes aren’t tracked, research code changes constantly
- ➔ Collaborative coding is nigh impossible without version control
- ➔ Code isn’t in an easily accessible place (Github) or tested (Travis)

Python 3

- High-level, dynamic, interpreted programming language
- Easy to read and write, scales-well to larger codebases
- Good general purpose language
- Insane number of packages
  - `wget -q -O - https://pypi.python.org/simple/ | wc -l` gives almost 130,000 packages!
- Recommended installation via the Anaconda distribution

Python + science = <3

Numpy for linear algebra and fast(er) computation (https://docs.scipy.org/doc/numpy/reference/)

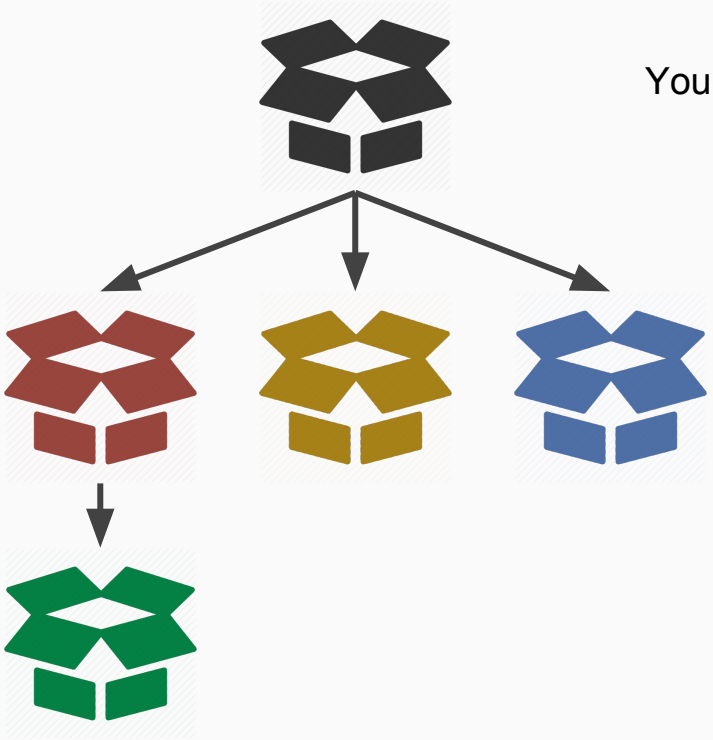
Matplotlib to make plots (+seaborn to make them look nice)

Pandas (panel data) for relational data, SQL-like queries

IPython + Jupyter notebook for interactive programming

Scikit-learn for machine learning

Why use conda?

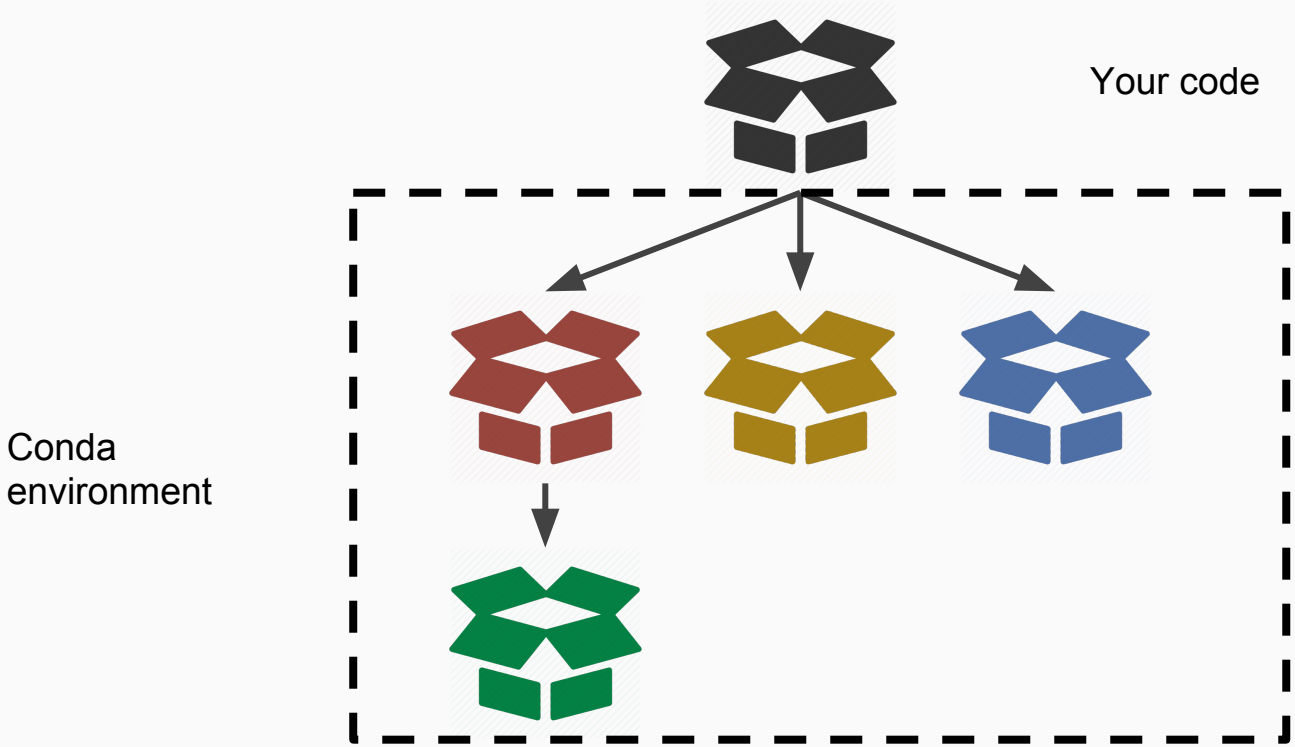


Your code

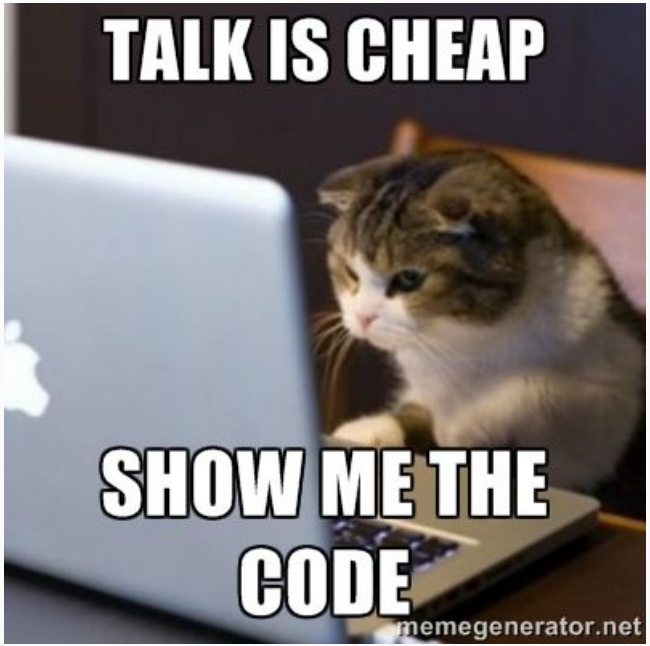
Depends on a whole lotta code

Which depends on more code

Why use conda?



Hacking with conda

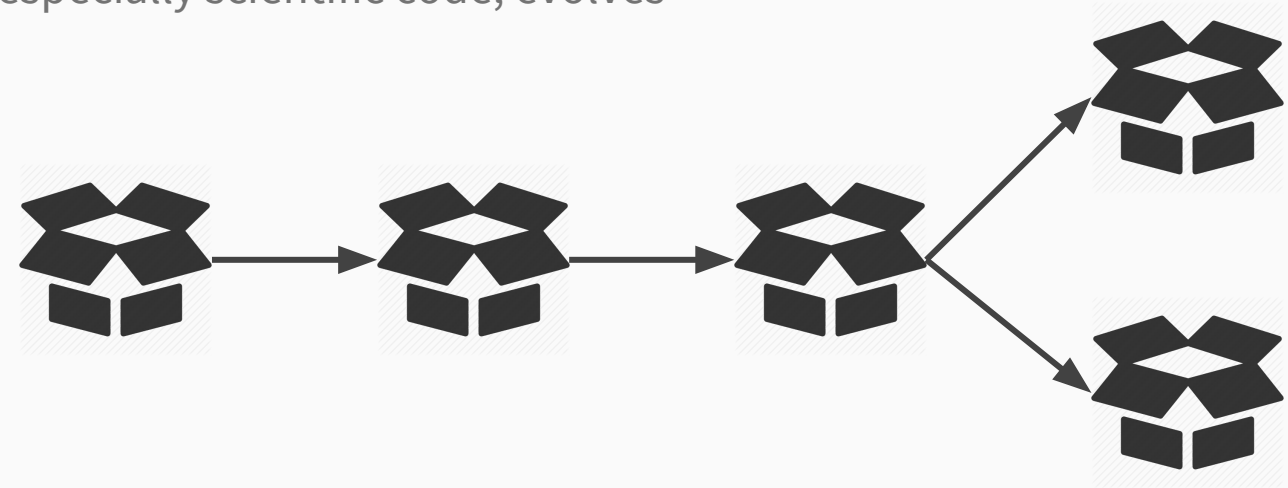


Hacking with conda

```
conda create -n test_env python=3.5 # new environment called test_env using python 3.5

source activate test_env # switch to the test_env environment

conda info -e # list environments

conda list -e # list packages in current environment

conda update -n test_env --all # update packages in test_env environment
```

Version control

Code, especially scientific code, evolves



Version control

Git to the rescue!

Version control...why use (g)it?

1. History of all your files
    a. Know exactly what you ran, when, and why you changed it
2. Branching and merging
    a. Try things out without regrets!
3. Collaboration
    a. No sending `Awful_Paper_v5_Tamas_edit_5.txt` via email

Intro to git

Go to https://try.github.io/levels/1/challenges/1 and run through it real quick. It should only take 15 minutes.

How to write git commit messages

https://chris.beams.io/posts/git-commit/

https://xkcd.com/1296/

Github* for social coding

- Great way to collaborate with others
- Nice graphical, web-based interface
- Good integration with other services (like continuous integration!)

Our class has a repo: https://github.com/ucsf-bmi-203/

*Not a good way to distribute code long-term because they could disappear any time

Test your code, pretty please

NEWS OF THE WEEK | SCIENTIFIC PUBLISHING

A Scientist's Nightmare: Software Problem Leads to Five Retractions

Greg Miller

Science 22 Dec 2006:
Vol. 314, Issue 5807, pp. 1856-1857
DOI: 10.1126/science.314.5807.1856

Due to an error caused by a homemade data-analysis program, on page 1875, Geoffrey Chang and his colleagues retract three *Science* papers and report that two papers in other journals also contain erroneous structures. (Read more.)

Test your code, pretty please

In September, Swiss researchers published a paper in Nature that cast serious doubt on a protein structure Chang's group had described in a 2001 Science paper. When he investigated, Chang **was horrified to discover that a homemade data-analysis program had flipped two columns of data**, inverting the electron-density map from which his team had derived the final protein structure.

http://dx.doi.org/10.1126/science.314.5807.1856

Continuous integration for better code

- Aka running your code with tests in an online container
- Runs your tests every time you make a commit to `master`
- It's not a panacea, but it can help squash 99%* of reproducibility bugs and can prevent regressions
  - Makes sure your new code doesn't break your old code

*I totally made this number up, but it's a lot

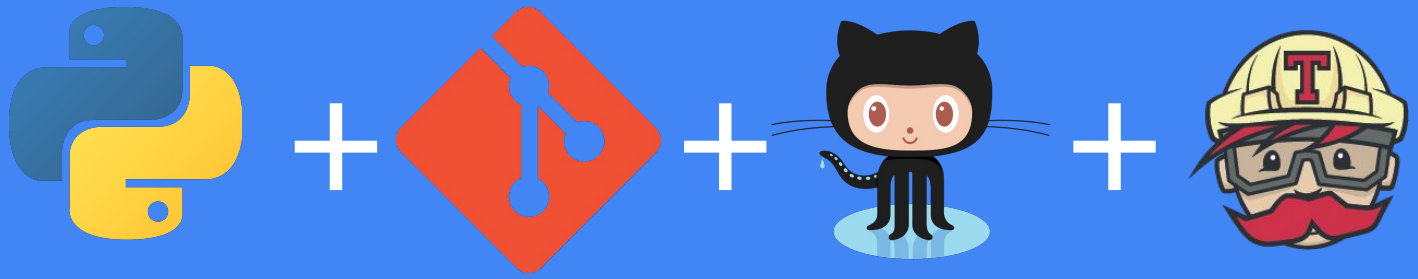
Travis

- Continuous integration service with good integration with Github
- Test your code easily in Ubuntu containers
- Makes sure your code runs on another system, i.e. check if your code is reproducible
- Runs after every commit and catches regressions
- Can easy test against multiple Python versions

.travis.yml is where you configure Travis

Editing .py/.md/.yml/.whatevs files

- Use your favorite text editor (Vim, Atom, not emacs, etc)
- ~~Or Yhat's Rodeo is nice environment with plotting, interactive console, etc~~
  - Try Jupyter Notebooks or nteract instead



One possible project layout

Layout of https://github.com/ucsf-bmi-203/example/

```
.
├── LICENSE
├── README.md
├── example
│   ├── __init__.py
│   ├── __main__.py
│   ├── algs.py
│   └── run.py
├── requirements.txt
└── test
    └── test_algs.py
```

In-class exercise

1. Fork the https://github.com/ucsf-bmi-203/example/ repo
2. Activate Travis (sign in with github to https://travis-ci.org/YOUR-GH-USERNMAE/example)
3. Clone repo to your machine
4. Figure out how to run the main file and the tests locally
5. Fix the tests by fixing the assert statements
6. Update the link to Travis in the README
    a. It should look like ([![Build Status](https://travis-ci.org/**YOUR-GH-USERNAME**/example.svg?branch=master)](https://travis-ci.org/**YOUR-GITHUB-USERNAME**/example)
7. Make your first commit!
8. Push and verify that Travis is passing
9. Send me a link to your repo (tamas@tamasnagy.com)