The idea of Dev/Ops

Modern DevOps for Systems Biologists

Jun Allard

Why systems biologists need Dev/Ops now

Motivation 1 of 4: The elevation of computational and mathematical biology

A realization that dawned on me in March 2020.

Recent worldwide events have challenged the mathematical biology community.

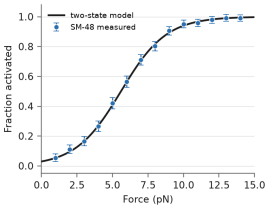
- A global pandemic put mathematical biology, and mathematical biologists, on the world stage — predictive epidemiology, in front of policy decision-makers.
- To keep that impact and respect, our standards of rigor in **coding** have to be elevated to match our standards of rigor in **mathematics**.
- The gap is troubling, because mathematical rigor has been a point of pride in the field. It is also an opportunity.



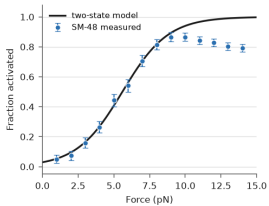
Motivation 2 of 4: A case where version control saved me

An early grad student convinced me to use version control: github.com/allardjun/a-typical-paper.

The plot we had



One day, while prepping the manuscript...





John M. Mola
@_JohnMola

Finishing a PhD is like finishing a group project where your partner made a ton of mistakes at the beginning of the assignment. Except your partner is just you 4 years ago.

8:20 PM · 2019-05-05 · Twitter Web Client

Motivation 3 of 4: The DevOps approach to science

Software development

Develop, operate, develop, operate, develop, operate.



Ship something that runs, then make it better.



Building a car

Motor, chassis, wheels, frame, ...



Or: first build a bicycle.



Henrik Kniberg

Graduate school

Bench phase, then coffee-shop phase.







GRADUATE JOURNAL

Lab makeover

As a high-school student I had a vision of graduate school, probably fuelled by watching too many French-art-house films. This typified erudite academics sitting in coffee shops drinking espressos. Now, at the end of my graduate career, I'm finally living my dream. I marvel that, in reality, multiple cups of coffee make my hands shake and the questions I'm working on are so abstruse that most of my friends have no idea what I'm talking about.

Before my coffee-shop phase, I was in what I call my wacko-lab period — spending most of my time collecting data in the lab. This reflects a far-removed view of graduate school that I had as an undergraduate, in which I would be free to doggedly repeat experiments, with robotic precision, and so therefore you need to think creatively and freely about your science.

Now, as I approach the end of my graduate career, I look at objective spending up the development of future graduate students overtake the two phases of graduate school. To do so, I suggest changing the ambience of our dumbbells. Add some light jazz, a few paintings by local artists, and bento. This, as my dream demonstrates, will inspire graduate students to generate intellectual property that is more valuable than the sum of its parts. And, for the safety of all, I would let me install an espresso machine next to my lab bench.

Miles de Vinci is a molecular-biology graduate student at the Massachusetts Institute of Technology.

NATURE Vol 442|27 July 2006

The DevOps approach is to make **small-batch, continuous, end-to-end** improvements.

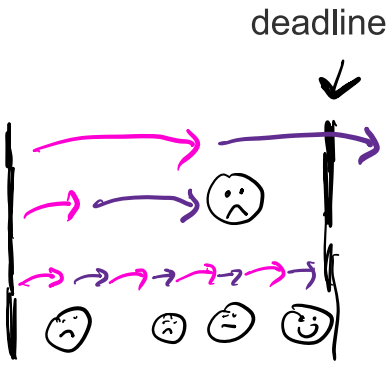
Pros and cons of working this way

Cons

- **Wasteful.** You don't use a lot of what you do. Think of the industrial revolution and assembly lines.
- **Misses the advantages of scale** that come from doing things in parallel.
- **Might invalidate statistical design** — consider a randomized control trial.
- **A lot of time spent keeping different versions organized.**

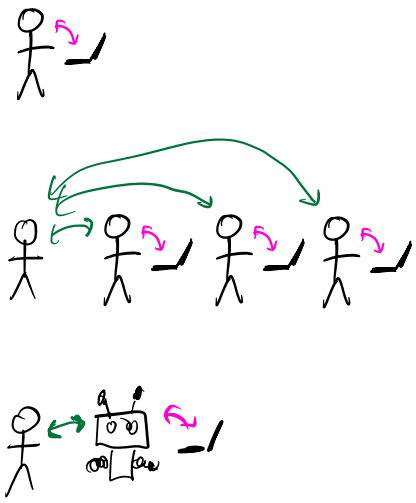
Pros

- For any complex, uncertain project (i.e., science!), **all the parts of the project gradually adapt** to achieve the goal.
- **Reduces time-to-deadline uncertainty.** Too slow or too unambitious? No problem.



Motivation 4 of 4: Supervising powerful-but-unreliable team members

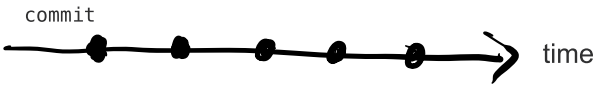
- At early career stages, **you write the code**.
- For many career paths — including outside science, and outside coding — you eventually **mentor, coach and supervise others**.
- Your job becomes helping them, critically testing their approaches, and catching bugs in their thinking.
- You need a tool that lets them **say what they are thinking**, lets you **revert** if necessary, and gives you an opportunity to **look over what they are contributing**.



In the age of agentic coders, the skill of **supervising** needs to be learned at earlier career stages.

Working with your past and future self

Version control with git



Advantage: **bisection search** for trouble. When something broke between here and there, git will help you halve the interval until you find it.

Vocabulary

- repo
- commit
- stage / add
- commit message
- commit hash
- `HEAD`
- checkout - going to a different commit

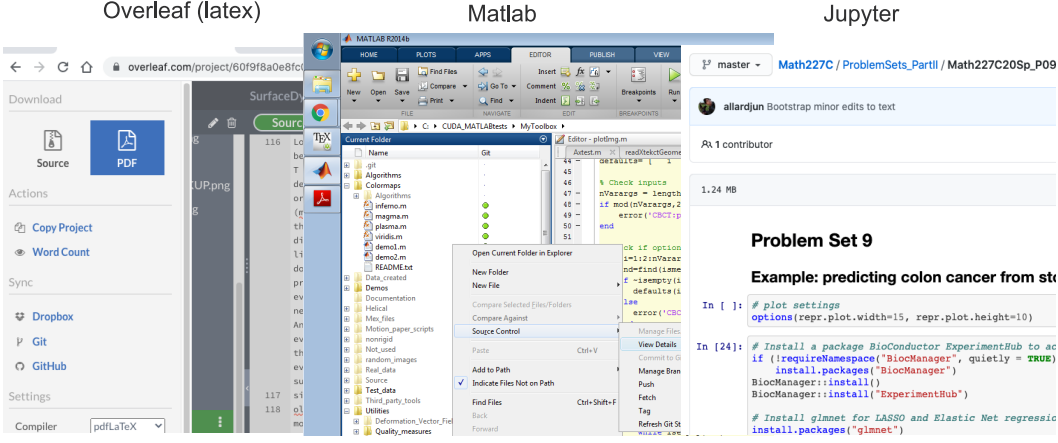
Two different things

- **git** — an open-source technology
- **GitHub** — a company

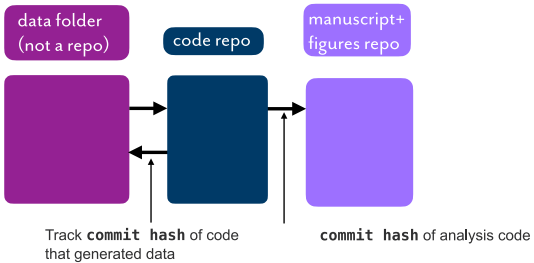
git is already there

Git is already built in to almost all the software you interact with.

- **VSCode**, **Jupyter**, **MATLAB** — full support, already installed.
- **Microsoft Office**, **Mathematica** — it works, but misses some features.



Directory architecture and large data



How we (the Allard group) organize a project — **three places**:

1. **Large files**, outside of git, backed up (e.g., Google Drive, Dropbox, linux server rack).
2. **Code**, in a git repo.
3. **Manuscript and figures**, in another git repo (not shared).

The three locations share each other's commit hashes, so every figure knows which commit it came from.

You *can* put large files in the git repo and reach for `.gitignore` instead — but the three-place split is what has held up.

Version control with git can be used smoothly for entire projects — analysis and write-ups, not just code. That is what lets you take a continuous-improvement "DevOps" approach, with the pros and cons above.

How often should you commit?

Ask what the action you just took actually was.

- Running something to get a result — `experiment`
- Adding a feature, mode or setting — `feat`
- Fixing a bug or troubleshooting — `bugfix`
- Refactoring, cleaning up or improving without adding new capabilities — `refactor`

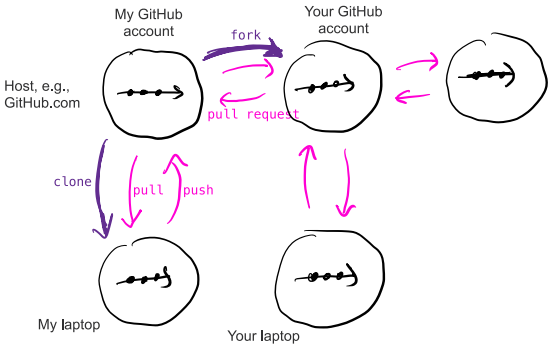
A commit should not include more than one of those. "Make your commits more *atomic*".

Committing is the one opportunity to slow down and think. Avoid automatic commits – this is not the time.

- In revisions, one commit per Peer Reviewer comment (e.g., `Reviewer1MajorComment3`)

Working with a team

git fork, clone, branch, merge, pull request



In the cloud

GitHub, and other providers: companies, university servers, ...

- push, pull
- clone
- fork
- pull request

The idea of **repo ownership**: taking is free, giving requires permission.

Advantages *and* disadvantages over automatic history, like Apple Time Machine or Google Docs.

Much we did not discuss

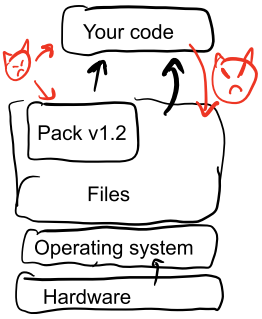
There is much we did not discuss.

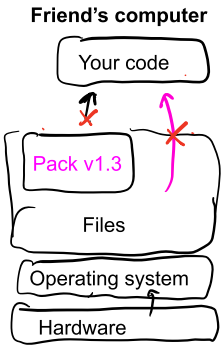
- Branching
- Cherry-picking

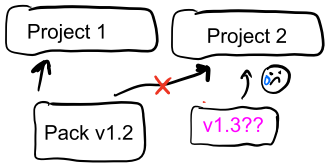
Working with the world, across teams and time

Reproducibility, dependency, security and isolation

For your code to impact the scientific community, it should work across teams, computers, and dependency upgrades.



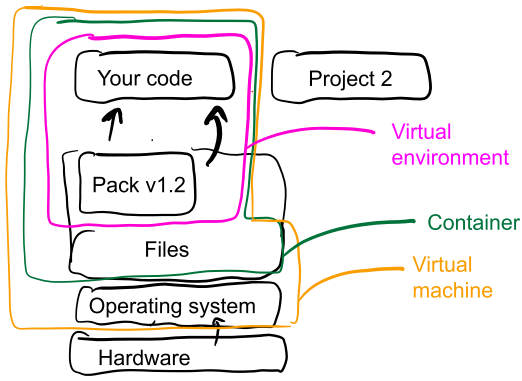




"But it worked on my machine."

Three levels of isolation

- Continually assuring that it works on **any** machine, for **any** person, without you helping.
- One project should not screw up another project, even on the same machine.
- One actor — human? agent? — should not screw up another actor, even on the same machine.



Environments in Python, Conda, Julia, R and Matlab

Same idea everywhere: make a local copy of your dependencies, activate it, and *record* it so someone else can rebuild it.

Python

- Older way: a `venv`, memorialized in `requirements.txt`; someone else recreates it with `pip install -r requirements.txt`.
- Newer way: `uv.lock`; someone else recreates it with `uv run`.

Conda

- Make one and switch to it: `conda create -n myproject`, then `conda activate myproject`.
- Memorialized in `environment.yml`, written by `conda env export`; someone else recreates it with `conda env create`.

Julia

- All dependencies needed by any project on this machine live in `$JULIA_DEPOT`.
- Each project records what it needs in `Project.toml` and `Manifest.toml`.
- Someone else recreates your environment from those.

Containerization: Docker and dev-containers

Some different things

- **containers** — an open-source technology
- **Docker** — a company
- **dev-container** — the VSCode/GitHub/Microsoft specification for containers

The definition of the machine lives in a `Dockerfile` or in `.devcontainer/devcontainer.json`.
They are shorter than you might guess, because they layer on top of other people's definitions:

```
"image": "mcr.microsoft.com/devcontainers/python:1-3.13-bookworm",
```

Both the **Our Story** activity and **Activity 2** take place in containers. You can read their `.devcontainer/devcontainer.json` to see exactly how little is in there.

The four kinds of documentation, and the primacy of QUICKSTART.md

The standard classification of the kinds of documentation that good coders use:

- references
- tutorials
- how-tos
- explanations

One kind of how-to is QUICKSTART.md: instructions for someone else, starting from recreating the environment and ending at a generated output, e.g. a plot.

````
# QUICKSTART

Install [uv](https://docs.astral.sh/uv/) and then run:

```bash
uv sync
uv run python generate_kinetics_versus_length.py
```

This will generate `fig_kinetics.png`.
````

Keep QUICKSTART.md continuously up to date - know that someone else could run it without you — and you will find yourself at peace. It eliminates the "heroics" and "heist mode" at the end of a project.

Much we did not discuss

There is much we did not discuss.

- Attribution, licensing ("Creative Commons", "MIT", "GPL"), confidentiality and harvesting and Open Science.
- What makes a good commit message?
- What makes a good end-to-end test? How do you come up with good end-to-end tests?