week 12: doing computational work

Compared with working at the bench as an experimental biologist, you've probably found that computational work moves with a different rhythm, and requires a different style of organizing your work and keeping your notes.

With wet experiments, doing the work takes longer than planning it. When I was doing experiments, I often had a comfortable rhythm evenings at my desk writing down a protocol for the next day; working the next day at the bench, making occasional notes on my protocol where I improvised or did something went wrong; then back at my desk to write up the day's data and observations in my permanent notebook.

Computational work is more improvisational, with a rapid trial-and-error cycle. I have a general plan, but I make up the detailed execution as I go. The moment I think of something, I can type the command or write a line of code and see what happens. Every command I type is a little experiment. Code and analysis protocols develop by rapid evolution. It would take me longer to plan and document my work than to just keep doing it. But working like this, after a while, I can't remember what I did, I can't find when and where I did it, and I can't reproduce it.

The volume of the data and the code also becomes an issue. Even if I could print it out and paste it into my notebook, it's dead there. I need my data and code live in the computer, searchable and executable.

It was easier to learn how to do experimental work by watching people. Doing experiments, I was physically engaged with other people I could learn from, in the same lab, sharing the same equipment. I saw the style and habits of the people around me. Computational work is more insular and highly personal: me and my laptop. I rarely get to see the style and habits of other computational scientists, even in my own lab.

Here I'll tell you about how I've learned to do my own computational work. I'm still evolving my system, and always have been. Don't take this as gospel; only as ideas. I'm exposing my style and habits to you. I know that geeks love to obsess about their tools, so I'll try to restrain myself to only one or two important bits on that front.

The main spine of what I'm going to tell you about:

- a paper notebook of general plans and results, with a table of contents; roughly a page summarizing each day's work.

- an electronic "notebook" that's just a simple scheme of organizing directories on the unix filesystem, organized chronologically, holding the "supplementary data" behind each paper notebook page in sufficient detail to make the work exactly reproducible (mainly by me; and for work that gets published, by others too).

- git repositories for long-lived software and writing projects.

a paper notebook

Your mileage may vary, but one of the most important things to me is getting *away* from the computer and having a peaceful hour with pen and paper to think about what I'm doing. I still keep a paper notebook, comparable to what I did when I was an experimentalist, but I use it in a different way now. My notebook keeps the high level summary of what I was doing and why, and summarizes the results. All the reproducible details are in the computer.

A typical page from a research work day has:

- a summary of what I'm trying to do (what's the question)
- cross-references to previous work I've done (previous notebook pages: "xref H4/141" means see Harvard notebook 4, page 141)
- cross-reference to the corresponding electronic "notebook" directory where all the details are; this is a date-title directory name in my electronic notes, something like "1122-hmmer4-finished"
- a summary of what I learned (if anything), including cut-and-pasted figures and tables.

The purpose of my paper notebook is to keep track of things at a high level. All the data and code is the computer. I can look at the notebook page to get the gist of what I've done, and I can refer into the computer if I need details.

daily rhythm

At the beginning of a good work day, I sit with a cup of coffee, look back over previous work, and write down what I want to get done that day. At the end of a work day, I sit with a favorite beverage, think about whether I actually got anything done, and summarize it and write it down.

I often find that when I'm thinking about what I did, I didn't quite finish it properly, because I was working in a hurry and playing around with different ways to do it. Summarizing it in my paper notebook at the end of the day, trying to pull things together into a coherent summary almost like a draft of a mini-paper, forces me to go back and make things coherent. I realize that I forgot to record an important version number, or I didn't finish making an important plot that I wanted. So unlike my rhythm for experimental work, I iterate back to the actual work, cleaning it up in the computer as I'm summarizing it on paper.

I put myself in the mindset of "I am going to be looking at this a year from now, and I will not remember what I did". I write consciously for my future self, knowing that my future self will be busy and impatient. Similarly my software code is littered with comments like "you're going to want to rewrite this because it looks wrong but don't, because it's right, and here's why..." When I was getting started in science I thought I was going to remember everything I did (and for the most part, I did!). I thought the purpose of my notebook was for my advisor to look at what I did. But I have never once had any advisor look at my notebooks. On the other hand, I've gone back innumerable times looking for something I know I knew once (and often failed to find it). Now I realize my notebook system isn't for anyone else. It's an extension of my own memory.

one notebook to rule them all

I only keep one notebook at a time, for everything: research, notes, ideas, to-do lists, planning trips I have to take, angry interventional diatribes against myself, and so on. I used to keep multiple notebooks for different things, but this didn't work. I never had the right notebook with me when I want it. Now my notebook is a chronological diary. I remember things better that way, in context of everything I was doing at the time.

I keep a table of contents, with one descriptive line per page. I can scan quickly through years of work through these tables of contents. Because it's all chronological and in context of everything else I'm doing, and because I do this kind of scan so often, it ends up being a kind of memory palace trick. I end up burning the structure of my notebooks into my memory.

For the last twenty-some years, I've gotten my notebooks from Eureka Notebooks. They're bound black notebooks, 152 numbered pages, with front pages for a table of contents, and they fluff up enough to accommodate a bunch of cut-and-pasted additions. I go through about one or two notebooks a year. I number them (my St. Louis notebooks are STL1-11; Janelia are J1-15; I'm on H9 now here at Harvard) and I keep the old ones on my shelf. When a notebook is done, I have it scanned to a PDF (many copy/print stores can do this for you), and I keep all the PDFs on my laptop, so I always have access to all my notes even when I'm travelling. The PDFs are also my long-term archive; I'm careful to keep them backed up in multiple places. I also keep and archive the scanned PDFs of other people's notebooks from my laboratory, when they leave, and they keep the originals.

I'm compulsive about everything, down to brands of pens and mechanical pencils and chalk, but I'll spare you. The important thing is to use permanent ink, not pencil for your main notes. Your notes need to last decades, and they need to survive disasters like coffee spills.

The notebook is mostly for me to remember my own work, but a laboratory notebook is also a legal record. If there were ever a dispute over intellectual property -- when you had an idea, when you invented something -- your notebook could become evidence in court. This is *extremely rare*, but I've seen it happen. My own notebooks have never had to appear in court (I had a close call a few years ago - long story), but I have seen other people's notebooks get the legal treatment. I've served as an expert witness, and it was brutal to watch lawyers go into some poor scientist's notebook looking for ways to impugn it. So even though the main purpose of my notebook is for my own purposes, I'm also conscious of doing things by the book so long as they're not too inconvenient research-wise. For example, if I ever make a change or additional notes on an old page, I usually do it in a different color pen, and I date and cross-reference the change. This is more for my own good (forward- and backward-tracking errors) than anything else, but it also helps with the legal record if it ever came to it.

People will tell you you're not supposed to carry the laboratory notebook outside the laboratory, to minimize the risk of losing or damaging it. This is a rule I routinely violate, mostly because I work everywhere with my laptop. I carry my notebook and laptop everywhere -- airplanes, coffee shops, conferences -- which means I'm assuming those risks.

A downside of using paper is that it's not as searchable as an electronic record. I rely very heavily on my tables of contents, and my memory. One thing I haven't done yet that I probably will get to someday -- OCR'ing my scanned notebook PDFs, to make them full text searchable. Technology appears to be almost up to this now.

an electronic notebook (or rather: filing system)

It's important to distinguish the needs of creative, day-to-day research from the needs of a production pipeline. If you're building a robust production pipeline you'll get pulled in the direction of lab management software and electronic notebooks: with the digital advantages of high capacity, searchability, electronic archiving, but with disadvantages of additional overhead and inflexibility (working within a specific GUI or electronic notebook system).

For creative exploratory work, the premium is on flexibility, and therefore on simplicity. For my work I've never found it convenient to use an "electronic lab notebook" (for one thing, I like to draw freehand - another reason I like my paper notebook), and every system I've explored, I find too constraining. Instead I organize my "notebook" on the computer very simply, using the unix filesystem and some simple naming conventions. I want two things out of it:

- an organized way of keeping track of files associated with each day's work: whether they're code, data, figures, or text;
- a reproducible record of whatever commands I need to execute to turn input files into my output files.

I have a directory in my home directory called `notebook`. In there, each computer-y work thing that I do (that is, anything associated with computer files) gets an "electronic notebook page": a directory, named by date-title, something like `1122-hmmer4-finished`. In each directory, I have a file called `00README` (or lately, `00README.md`, formatted in Markdown) that contains my top-level notes and crossreferences. Anything other files or directories I want to keep from the work, I put them in the directory.

By using the date-title naming scheme, the unix filesystem automatically sorts listings chronologically when I do an `ls`. Similarly, by naming my main notes file `00README`, `ls` will sort that file to the top of listings of contents. The brief title (the `hmmer4-finished` part) is useful if I'm trying to find something quickly by scanning the `notebook` listing.

Because I keep both my paper and electronic notes chronologically, I generally don't keep directories organized by project (some people do). Instead, different projects I'm working on are interleaved chronologically. Like my paper notebook, almost everything I do electronically goes into the notebook, in one chronological sequence: research, grant and manuscript reviews, MCB112 pset drafts, and so on. My notebook directories are accessible by default to the other people in my lab (so if I send someone a crossreference like `1122-hmmer4-finished`, they know where to look), and I `chmod 700` any directories that are confidential for some reason.

By the end of a year, I have something like 100-200 directories in my `notebook`. Now I create an `Archive2021` directory and move them all into this archive. (In a couple of years or so, when I'm no longer accessing them much, these archive directories get moved out of my current `notebook` and into a directory structure that I use for longer term archiving, along with other stuff I'm archiving.)

The `00README` crossreferences to the corresponding paper notebook page summary (`xref H2/15`); the paper notebook crossreferences to the electronic notes (`1122-hmmer4-finished`). I can search either the paper table of contents or the directory listing in the electronic `notebook` chronologically by short title/description to find previous work.

daily work pattern: what goes in the 00README

The `00README` is just a plain text file (I use emacs as my text editor), or perhaps in a lightweight markup language called Markdown. Its primary purpose is to receive a lot of back and forth cut-and-pasting as I work.

I'm mostly working at the command line in a terminal window. When a command works and it's on the main stream of what I'm trying to get done, I cut and paste it to the `00README`. Some electronic notebook systems simply log everything you type. This is *insanity*. Only a small fraction of my commands end up being on-point for the work.

At the same time, I'm organizing the `00README` into little blocks of code, massaging it into what are essentially short bash scripts, and I'm annotating it by what I'm trying to do. Then when I iterate back to redo something, or improve on something, I can cut-and-paste an existing "script", either straight into the terminal, or elsewhere in the `00README` where I make modifications to it.

As I'm working, the `00README` starts to become an organized document that sort of looks like a Jupyter Notebook page, but in plaintext, with a mix of text documentation and bash commands. Any scripts I need (perl, python, R, bash) are in .pl, .py, .r, .sh files in the directory, and I make note of their reason and existence in the `00README` at the appropriate place.

I don't like working in something like Jupyter Notebook itself, just because it comes with added inflexibility and overhead (I stick close to plaintext and the command line to keep things simple and flexible). Also, after all these years, emacs is hardwired into my neurons. I can't stand working with text in a system that doesn't have emacs power at its disposal.

"provenance" section

One of the easiest things to lose track of in computational research is the exact versions of software and data that you used. I often add a "provenance" section at the top of the `00README` that lists where I got various data and code from.

I will also often use symlinks in the notebook directory to link to exact versions of databases that I use, for example:

```
ln -s /n/eddy_lab/data/uniprot/uniprot-62.3/unipro
```

creates `swissprot.fa` in my working directory as a link to a specific permanent database on our lab disks. (We have a system in the lab for maintaining archival copies of old data that keeps such symlinks alive for a long time, and such that the directory names include version numbering; thus even when we clean up and delete old data, the now-dead symlink contains the relevant versioning info just in its name alone.) The symlink-creating `ln -s` command goes into my provenance section. We keep our installed software in a similar way, so symlinks to installed software will automatically convey version info.

When I'm using my own development code, I'll typically record the git commit version in the provenance section.

Because I generally work chronologically rather than by projects, a particular project evolves over time across multiple directories. When I do a new version of an older experiment, I make copies of scripts or other things I need, and note those `cp` commands in my provenance section. Each notebook directory is a standalone archive of what I did on a particular day (or perhaps over a few days). Like my paper notebook, it's unusual that I go back and edit old notebook directories. This means I have a lot of copies of copies, which in general in software development and computational science is supposed to be a Bad Thing that you're supposed to use version control for. Arguably, I could achieve the same ends in a version control system that snapshots an entire directory (such as `git`), but I've found that I prefer to be able to immediately look back over the chaotic chronology of a project's directories, rather than organizing it as a single git project and pretending that it's less chaotic than it actually is.

working toward a manuscript

As time goes on, various experiments are working their way toward being figures and tables of a manuscript, as I redo them (over and over and over again). Notebook directories are starting to get titles like `1122-projectx-figure4-v2`. The amount of copying increases (copying `v1` of an experiment to `v2` to improve and redo it), and the provenance tracking gets more careful. The `00README` gets tighter, evolving toward something that's starting to look more like supplementary material than disorganized chaos.

Meanwhile I'm writing the paper in a separate directory organization, under something like `writing/manuscripts/projectx`. In there I have something like a `Figures` subdirectory, and in there I am either symlinking directly to the current versions of figures and tables that are in my notebook.

By the time the manuscript is ready, the methods and supplementary data are essentially already written in my notebook: the relevant `00README`'s for each figure or table contain the exact commands needed to reproduce the work. I aim to be able to cut-and-paste whole blocks of commands from the `00README` and nigh-automagically reproduce the figures and tables exactly. We bundle all this stuff up, turn symlinks into copies, organize it and clean it a little more, make sure the commands all still work (it's surprisingly easy to mess up a trivial detail in the cleaning-up phase, breaking everything) and I release it as a tarball accompanying the paper.

code, version control, and git

For software and code that I'm using often and across projects, I eventually adopt them into a project-oriented directories in a top-level directory called `src` (for "source code") for software, and `scripts` for scripts. New scripts and programs are born in notebook directories, and get promoted to `src` and `scripts`. Once there, they come under version control, and currently I'm using `git` as my version control system.

It's easy to think of `git` and `GitHub` (as in github.com] as the same thing. GitHub is an online open source code sharing system, and it's great. A bunch of stuff from our lab is hosted at GitHub, for our largest and most widely distributed software projects. MCB112 itself is hosted in a private GitHub repository.

`git`, though, is a command line program that you can run easily on your laptop without ever dealing with github. I have innumerable git repositories on my laptop, for different personal projects. I use `git` as part of my normal daily workflow, whereas github is part of our lab publication and dissemination strategy. I use git not just for code and software, but for pretty much anything where I generally only want to see one current up-to-date version, where I also want to keep old versions and log info about my changes.

lightning git intro

If you haven't used git or version control before, it's important to know how easy it is. If you don't have it installed, go here and follow the instructions. Suppose you have a new project with a couple of files, like so:

```
mkdir myproj
cd myproj
echo "This is file1" > file1
echo "This is file2" > file2
```

then, sitting in that directory, you initialize a new git repository like so:

```
git init
```

and boom, you have a git repository: it's stored in `.git`, a hidden directory in your project.

To tell git which files you want it to take care of ("track"), you use `git add`:

```
git add file1 file2
```

When you have your project in a state that you want to save ("commit"), you use `git commit` to take a versioned snapshot of it:

```
git commit
```

It'll ask you to write a log message, where you make notes about what you've done so far.

When you make changes, bringing the project to a new state, you `git add` the new and changed files, and `git commit` a new snapshot of the project:

```
echo "This is a new line on file2" >> file2
git add file2
git commit
```

To look back at your version history, you type `git log`.

Specific git commits are identified by a horrendous long string like `6f85a71cb764a82e33d5402ac052a5c52260cd75` (the "SHA-1 hash", in the jargon) but are traditionally referred to by just a brief prefix of the hash; any (unique) prefix will actually do, but git pros seem to default to 7, as in `6f85a71`.

To revert the project to an earlier state, you use `git checkout`. For example:

```
git checkout 6f85a71
```

if your git log shows that you have a commit prefixed `6f85a71`. The entire state of the project (the files that git is tracking, anyway) is immediately timewarped back to the way it was in commit `6f85a71`. The main line of your commits is called `master` by default, so to get back to your current state:

```
git checkout master
```

That's all you really need to know to get started using git productively. Good git tutorials online include one from Atlassian, and the Scott Chacon and Ben Straub book *Pro Git*.

Further reading on computational work

Two good articles about organizing computational work:

- Santiago Schnell, *Ten Simple Rules for a Computational Biologist's Laboratory Notebook*
- William Stafford Noble, *A Quick Guide to Organizing Computational Biology Projects*

Further reading on doing science

Two great essays on doing science:

- Don Coffey, *The Final Exam*
- Richard Hamming, *You and Your Research*