Workflow languages in bioinformatics

C. Titus Brown

Lab for Data Intensive Biology

UC Davis School of Veterinary Medicine

ctbrown@ucdavis.edu; @ctitusbrown on Bluesky.

Slide will be available: google 'osf.io my talks titus brown'

Some intro warnings!

- This talk will (mostly) be about snakemake. Apologies – this was not an intentional bait and switch, I didn't remember the title when I was preparing my outline & slides 😭

- I have lots of strong opinions, but they are loosely held. I am hoping for discussion! Please feel free to post questions in chat, or raise your hand! I can handle distractions!

- I have about 20-30 minutes of content :)

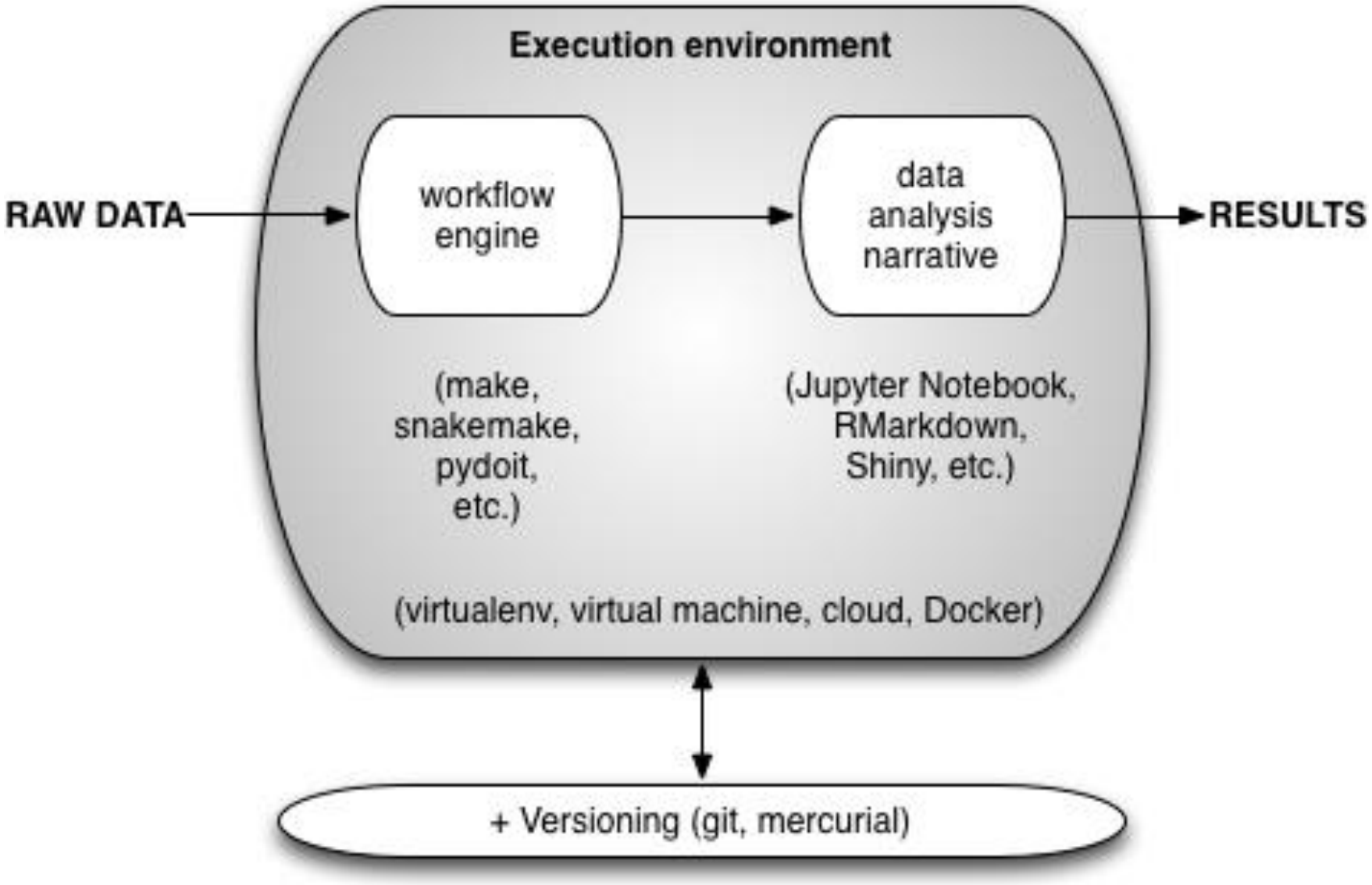
Introducing myself

- I came to biology through physics, sort of.
- Modeling, simulation, data analysis, genomics…
- Large scale data reuse…
- *Really* interested in helping people extend their reach and building capacity on a field-wide scale.
- Very open-source/open-science/reproducibility focused
- I do a fair bit of software development and engineering (github.com/ctb, github.com/dib-lab)

Why workflows??

Workflow systems may need no introduction with this crowd, but: over the years I've realized workflows address a very personal set of paranoid considerations on my part.

- Workflow systems let you know when a job fails.
- Workflow systems let you pick up execution immediately after a previous failure.
- Workflow systems manage concurrency for you.


Execution environment
RAW DATA
workflow engine
data analysis narrative
RESULTS
(make, snakemake, pydoit, etc.)
(Jupyter Notebook, RMarkdown, Shiny, etc.)
(virtualenv, virtual machine, cloud, Docker)
+ Versioning (git, mercurial)

Snakemake is my workflow language of choice. Why?

- There are 100s of workflow systems![1] How should we pick one??

IMO,

- Choosing a workflow system is a long term decision; you and your collaborators will probably be stuck with your decision for a while!
- Pick one that has a community and whose community overlaps with your domain.
- In bioinformatics, the choices are:
  - Snakemake
  - Nextflow
  - CWL
  - WDL
- My lab converged on snakemake through a nonlinear process.

[1]https://github.com/common-workflow-language/common-workflow-language/wiki/Existing-Workflow-systems

Digression: How different is bioinformatics from (e.g.) physics?

Bioinformatics has:

- *More* many large files
- *More* big RAM jobs
- *Much less* numerical processing and ”pleasantly parallel” multithreaded work.
- Lots of format interconversions and sorting etc
- In bioinformatics, it’s kind of like everyone has their own personal particle accelerator for doing experiments and producing data. (Maybe?)
- tl;dr Lots of medium and small shell script jobs, with intermediate output files.

Snakemake in 5 slides or less

```
rule example:
    input:
        "file1.txt",
        "file2.txt",
    output:
        "output file1.txt",
        "output file2.txt",
    shell: """
        echo {input:q}
        echo {output:q}
        touch {output:q}
    """
```

Robust templating

```
ACCESSIONS = ["GCF_000017325.1",
              "GCF_000020225.1",
              "GCF_000021665.1",
              "GCF_008423265.1"]

#...

rule compare_genomes:
    input:
        expand("{acc}.fna.gz.sig", acc=ACCESSIONS),
```

Robust wildcarding

```
with open('accessions.txt', 'rt') as fp:
    ACCESSIONS = fp.readlines()
    ACCESSIONS = [ line.strip() for line in ACCESSIONS ]

print(f'ACCESSIONS is a Python list of length {len(ACCESSIONS)}')
print(ACCESSIONS)

rule all:
    input:
        expand("{acc}.sig", acc=ACCESSIONS)

rule sketch_genome:
    input:
        "genomes/{accession}.fna.gz",
    output:
        "{accession}.sig",
    shell: """
        sourmash sketch dna -p k=31 {input} --name-from-first -o {output}
    """
```

Good Python integration

```
GLOB_RESULTS = glob_wildcards("genomes/{acc}.fna.gz")
ACCESSIONS = GLOB_RESULTS.acc

print(f'ACCESSIONS is a Python list of length {len(ACCESSIONS)}')
print(ACCESSIONS)
```

Can apply operations to many files at once

```
# pull in all files with .fastq on the end in the #data
FILES = glob_wildcards('data/{name}.fastq')

# extract the {name} values into a list
NAMES = FILES.name

rule all:
    input:
        # use the extracted name values to build new filenames
        expand("subset3/{name}.subset.fastq", name=NAMES)

rule subset:
    input:
        "data/{n}.fastq"
    output:
        "subset3/{n}.subset.fastq"
    shell: """
        head -400 {input} > {output}
    """
```

Things about snakemake that I don’t love

Snakemake is amazing! But even crushes have limits ;)

- The checkpointing system for dynamically building new DAGs on the basis of previous outputs is well implemented but confusing to use.
- The use of the Python parser results in error messages that are confusing to newbies.
- More generally, snakemake is not super welcoming to newbies. (But, see later slides)
- In the past, snakemake has stopped working well past ~50,000 jobs.

How does snakemake compare/contrast to other workflow languages in bioinformatics?

- I've heard great things about nextflow! I just haven't used it.
  - Great toolchain ecosystem
  - Wonderful pre-built / reusable workflows
  - Fantastic community
- Common Workflow Language (CWL) and Workflow Definition Language (WDL) –
  - Different approach: define standard language, support multiple runners
  - Used by production platforms
  - *IMO*, less about "let's build a research workflow that we will tweak a few times, run a few times, and then need to tweak again"
  - More about "I need to run 100s of thousands of jobs in as efficient a manner as possible"
  - Still, this is an increasingly mature ecosystem!
- Note, you can wrap snakemake workflows in CWL or WDL! (Not yet sure if this is a good idea)

Back to community considerations…

- These days, you *really* want to be able to find answers on the Internet
  - Stackoverflow, tool documentation, and ChatGPT,
  - There's nothing worse than searching your error message and finding your own unanswered post from 2 years ago…
- This is reasonably synonymous with *community*… at least in biology.
- Snakemake and nextflow have really robust online communities.
- It is *probably* worth considering fixing or extending snakemake to meet your needs, vs writing your own 😱
- Note: snakemake v8 will have a robust plugin architecture!

Why not write applications around workflow software?

PLOS COMPUTATIONAL BIOLOGY

OPEN ACCESS

EDUCATION

Ten simple rules and a template for creating workflows-as-applications

Michael J. Roach, N. Tessa Pierce-Ward, Radoslaw Suchecki, Vijini Mallawaarachchi, Bhavya Papudeshi, Scott A. Handley, C. Titus Brown, Nathan S. Watson-Haigh, Robert A. Edwards

Published: December 15, 2022 • https://doi.org/10.1371/journal.pcbi.1010705

Workflows as applications

- It is relatively easy to embed snakemake in a Python command-line application.
- You can provide good default behavior and hide much of snakemake complexity!
- Upsides:
  - Build command line applications that resume from failure well!
  - Natively support full resource allocation, scheduling, cluster utilization, etc!
- A few downsides –
  - Snakemake error messages come through regardless…
  - Testing is rather challenging

Teaching snakemake to new bioinformaticians

- Many biologists/biomedical data scientists should be using workflow systems, IMO.
- But, we don’t teach computing or data science to undergrad biologists…
- …so graduate students arrive with little to no knowledge of *computing.*
- There is also maybe something about biology undergrads avoiding math and computing, although I think this is changing?
- We *also* do a terrible job, in general, of teaching computing basics…

Thought: it's *easy* to learn to run one sample

…but no one ever teaches you to analyze *100* samples.

In bioinformatics, at least, you can always find the commands you need to run for a particular analysis by googling.

But many labs today have dozens to 100s of samples.

And that is a COMPLETELY different kettle of fish!

Thought: workflow systems are a “cracked mirror” of computing

Almost every feature in a workflow system exists because of a feature that is *lacking* in standard computing…

- Notification of failed jobs
- Inability to precisely specify execution “flows” other than linear
- Inability to robustly resume where we left off
- Not-that-great foundational shell-scripting languages for pattern matching etc.
- Hard delineations between single-chassis computers and clusters

So… maybe if we teach workflow systems well, that’s a good entry point to computing?

My favorite way to teach snakemake

Start with just shell commands:

```
rule download_data:
    shell: """
        curl -JLO https://osf.io/4rdza/download
    """

rule download_genome:
    shell:
        "curl -JLO https://osf.io/8sm92/download"

rule map_reads:
    shell: """
        minimap2 -ax sr ecoli-rel606.fa.gz SRR2584857_1.fastq.gz > SRR2584857_1.x.ecoli-rel606.sam
    """
```

Evolve from there:

And then walk them through *connecting* shell commands with input/output

```
12      rule map_reads:
13          input:
14              reads="SRR2584857_1.fastq.gz",
15              ref="ecoli-rel606.fa.gz"
16          output: "SRR2584857_1.x.ecoli-rel606.sam"
17          shell: """
18              minimap2 -ax sr {input.ref} {input.reads} > {output}
19          """
```

Rationale: you can always find the right set of shell commands to run ;)

A draft snakemake book

1. Introduction

2. Acknowledgements

3. Section 1 - A Stepwise Introduction to Snakemake

- **3.1.** Chapter 1 - snakemake runs programs for you!
- **3.2.** Chapter 2 - snakemake connects rules for you!
- **3.3.** Chapter 3 - snakemake helps you avoid redundancy!

4. Section 2 - Building an even more useful Snakefile

- **4.1.** Chapter 4 - running rules in parallel
- **4.2.** Chapter 5 - visualizing workflows
- **4.3.** Chapter 6 - using wildcards to make rules more generic
- **4.4.** Chapter 7 - giving snakemake filenames instead of rule names
- **4.5.** Chapter 8 - adding new genomes
- **4.6.** Chapter 9 - using expand to make filenames
- **4.7.** Chapter 10 - using default rules
- **4.8.** Chapter 11 - our final Snakefile - review and discussion

https://ngs-docs.github.io/2023-snakemake-book-draft/

…with good intro level materials, I think.

5. Section 3 - Beyond Your First Snakefile

- **5.1.** input: and output: blocks
- **5.2.** Using wildcards to generalize your rules
- **5.3.** params: blocks and {params}
- **5.4.** Using expand to generate filenames
- **5.5.** Running rules and choosing targets from the command line
- **5.6.** Techniques for debugging snakemake workflows
- **5.7.** Basic syntax rules for Snakefiles
- **5.8.** Visualizing your workflow
- **5.9.** String formatting "minilanguage"
- **5.10.** Using configuration files

6. Section 4 - Snakemake Patterns and Recipes

- **6.1.** Subsampling FASTQ files
- **6.2.** Using split to split up files
- **6.3.** Applying one rule to to many files - replacing for loops in shell scripts
- **6.4.** Never fail me - how to make shell commands always succeed
- **6.5.** Subsetting FASTQ files to a fixed number of records

7. Section 5 - Advanced Features

- **7.1.** Beyond -j - parallelizing snakemake
- **7.2.** Resource constraints and job management

Evolving the snakemake book

- Executable examples => automated testing
- Continuous integration!
- Room for a robust community contribution model…
- Fills a unmet need in the snakemake community: it's hard to get started.

I'd *really* like to build something that can be used for upper level undergrad teaching…

Resources mentioned here -

- Snakemake blog posts: http://ivory.idyll.org/blog/tag/snakemake.html
- Snakemake book draft: https://ngs-docs.github.io/2023-snakemake-book-draft/
  - Please file issues/questions at https://github.com/ngs-docs/2023-snakemake-book-draft

- Introduction to remote computing: https://ngs-docs.github.io/2021-august-remote-computing/

Thanks!

Always happy to chat –

ctbrown@ucdavis.edu

@ctitusbrown on bluesky

Or via github issues ;)