


# Want to get started on setup?

Open up the GitHub repository for the tutorial, then follow the instructions under the section titled “Live setup”. This will save us some precious time later!

[github.com/ccbskillssem/snakemake\\_fa23](https://github.com/ccbskillssem/snakemake_fa23)

## CCB Skills Seminar (Fall 2023)

A gentle introduction to **snakemake**: A tool for automating and streamlining your analyses 🐍 

Created by [Stacy Li](#) for the [Center for Computational Biology](#) at UC Berkeley.

### What is this?

This is a repository containing a sample configuration + workflow for learning **snakemake**. I've tried to make this a minimally fussy example of how **snakemake** works, and all the good it does for me in my work 😊

### Table of contents

- [Requirements](#)
- [Live tutorial setup](#)
- [Independent setup](#)
- [Slides](#)
- [Commands](#)
- [Credits](#)

[ snakemake ]

# A gentle introduction to

a tool for automating and streamlining your analyses

Stacy Li  
Sudmant Lab  
Thursday, October 12 2023

# Agenda

- Introduction: why use snakemake?
- Learning goals
- A (hopefully) short snakemake tutorial
  - Installation
  - Example QC workflow
  - Setting up and running the example workflow
  - Tips on how to make your (snakemake) life easier
- Final questions

# Why use snakemake?

S	N	A	K	E
Forgot to change a path, accidentally overwrote a file	Returned from a break, now you can't remember where these files came from	Paralyzing fear of rerunning your full pipeline	Unsure which script you <i>actually</i> used for analysis	Copy-pasting the same line of code manually for each sample
"Your code doesn't work on my machine." (no further info)	Eyes glazing over while checking list of 20+ full output file paths for completeness	"Why is this even installed?"	Painstakingly creating analysis flowchart for presentation	Something didn't work, and you forgot to save the log file
Typos while editing output paths for every sample	Upgraded one (1) package, now nothing works	 <i>free space</i>	Unsure if you re-analyzed <i>all</i> samples with updated parameters	Nuking your base conda environment (again)
Manually saving 10+ plots with slightly different names and subsets	Checking to see if you can run your next job yet	Fighting mentally and near physically with gnu-parallel	Copy/pasting your code into different project folders	Accidentally changing only <i>some</i> file names
Manually tuning memory allocation (job killed anyway)	Analysis script is 1000 lines, but 400 of them are commented out	preprocessing.sh and preprocessing_taylors_version.sh	Mysteriously missing output files	Manually creating output folders, one by one

# Learning goals

- Become familiar with basic snakemake principles
  - How does it know what to do?
  - What do I have to do?
  - When will it do (or not do) things?
- Know when to use (and not use) snakemake
- Be able to read, modify, and run a skeleton snakemake workflow

# Setup

For today's workshop, I've prepared a pre-defined conda environment that has everything you'll need to get started.

Not familiar with conda, or don't have it installed?

Option 1 (preferred: savio & local machine): Search **miniconda** and use the appropriate installer for your system. It's very fast :)

If installing on savio, I recommend installing in your scratch directory.

Option 2 (slow: requires CalNet): Log in to the DataHub.


You'll be able to use JupyterHub's compute & terminal (conda pre-installed).

# A little more setup...

Open up the GitHub repository for the tutorial, then follow the instructions under the section titled “Live setup”. This will save us some precious time later!

[github.com/ccbskillssem/snakemake\\_fa23](https://github.com/ccbskillssem/snakemake_fa23)

## CCB Skills Seminar (Fall 2023)

A gentle introduction to **snakemake**: A tool for automating and streamlining your analyses 🐍 

Created by [Stacy Li](#) for the [Center for Computational Biology](#) at UC Berkeley.

### What is this?

This is a repository containing a sample configuration + workflow for learning **snakemake**. I've tried to make this a minimally fussy example of how **snakemake** works, and all the good it does for me in my work 😊

### Table of contents

- [Requirements](#)
- [Live tutorial setup](#)
- [Independent setup](#)
- [Slides](#)
- [Commands](#)
- [Credits](#)



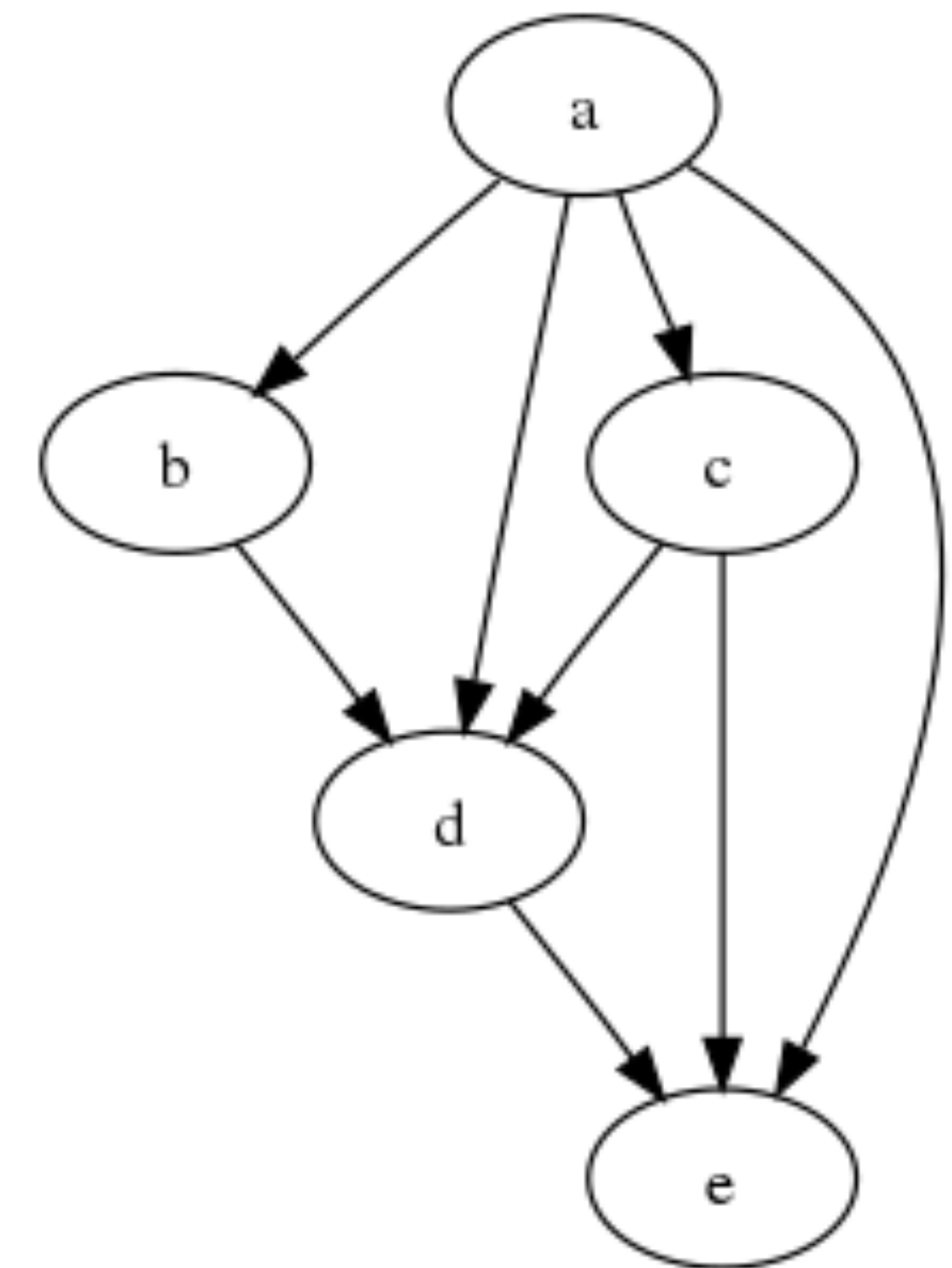
# How does snakemake work? (simplified)

Snakemake models workflows with a *directed acyclic graph*.

Each node of the graph is a task.

Tasks are related to each other by the files required as input, or generated as output.

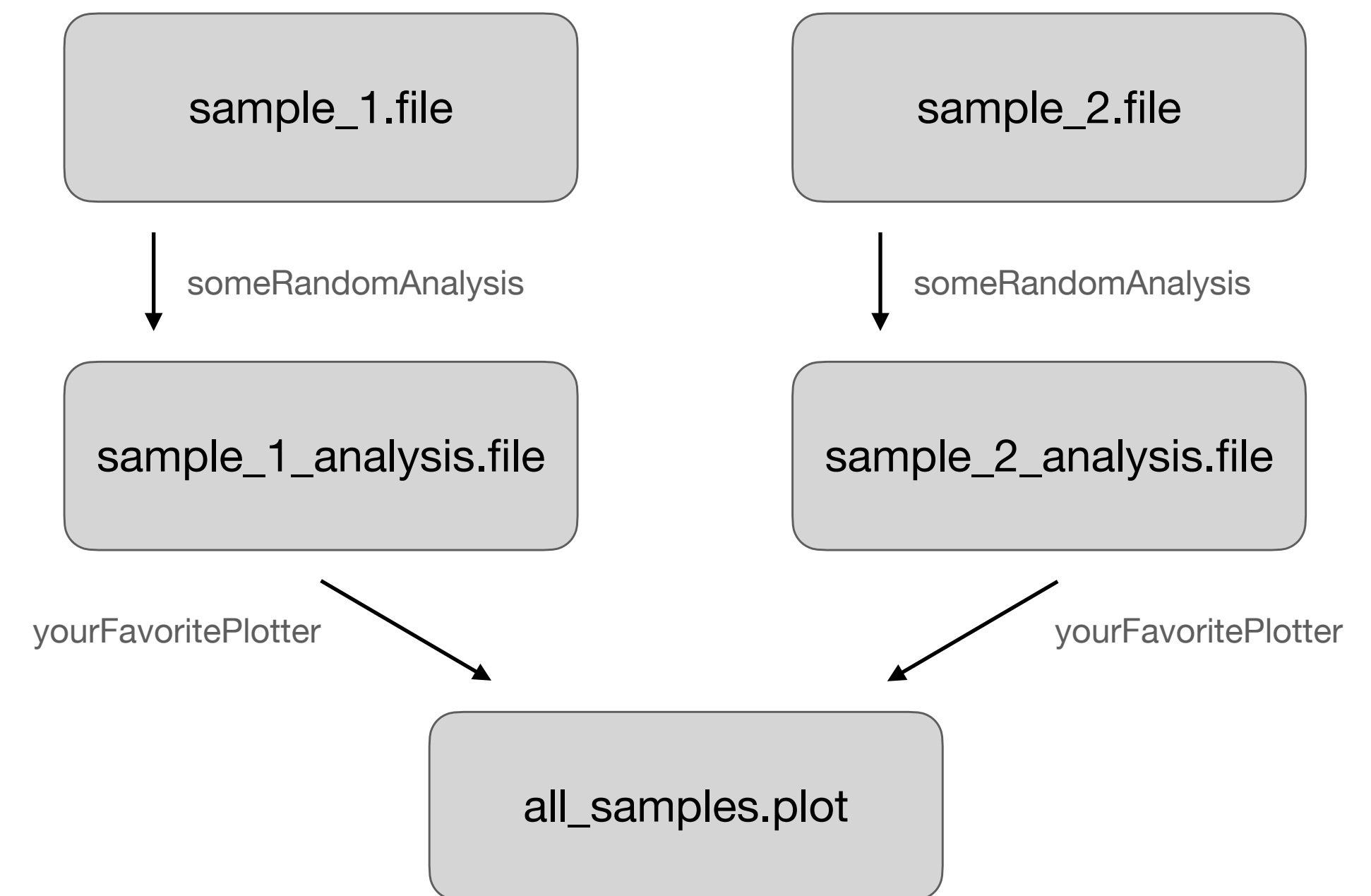
Snakemake parses this graph to understand what tasks need to be run to generate a target file.



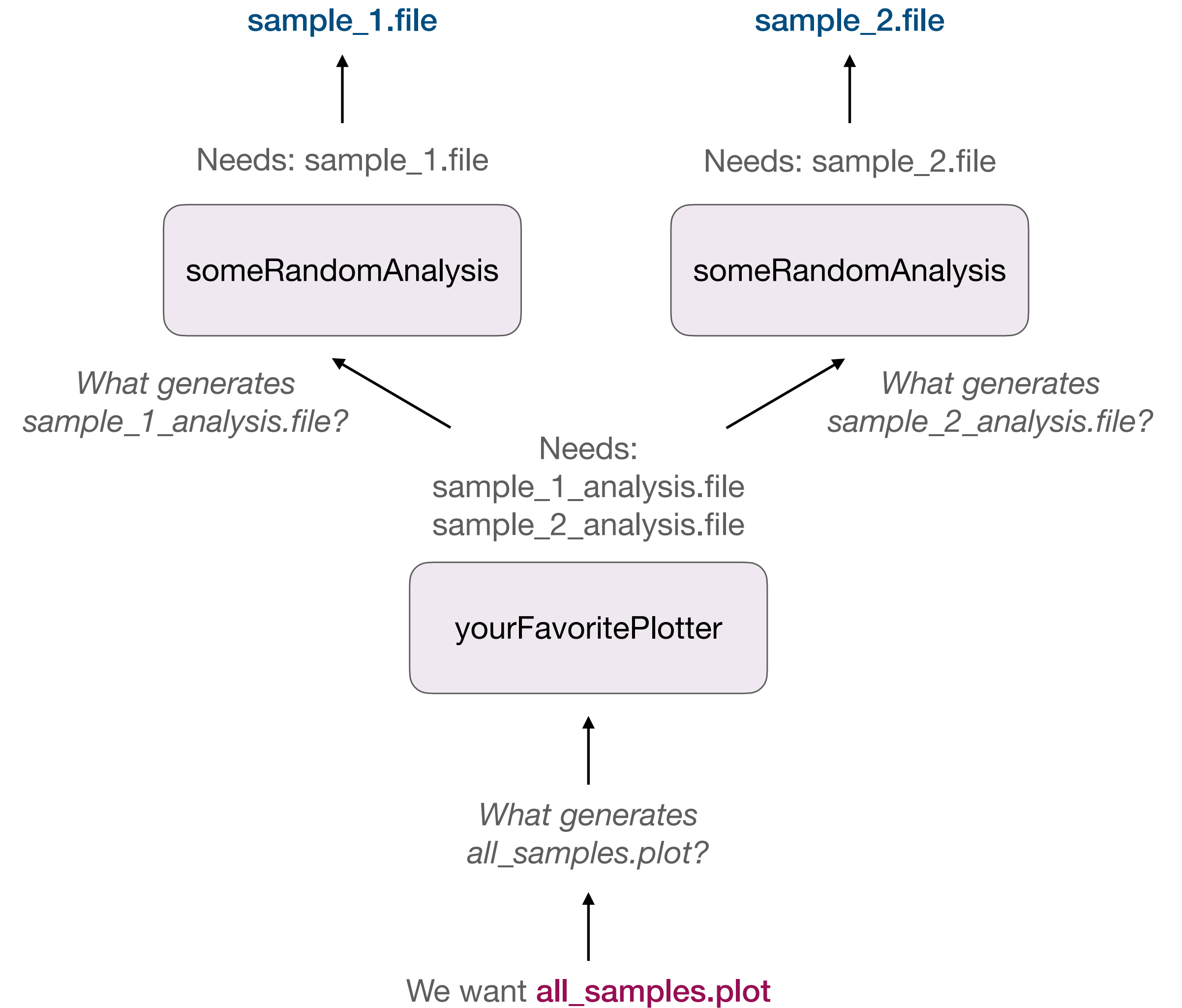


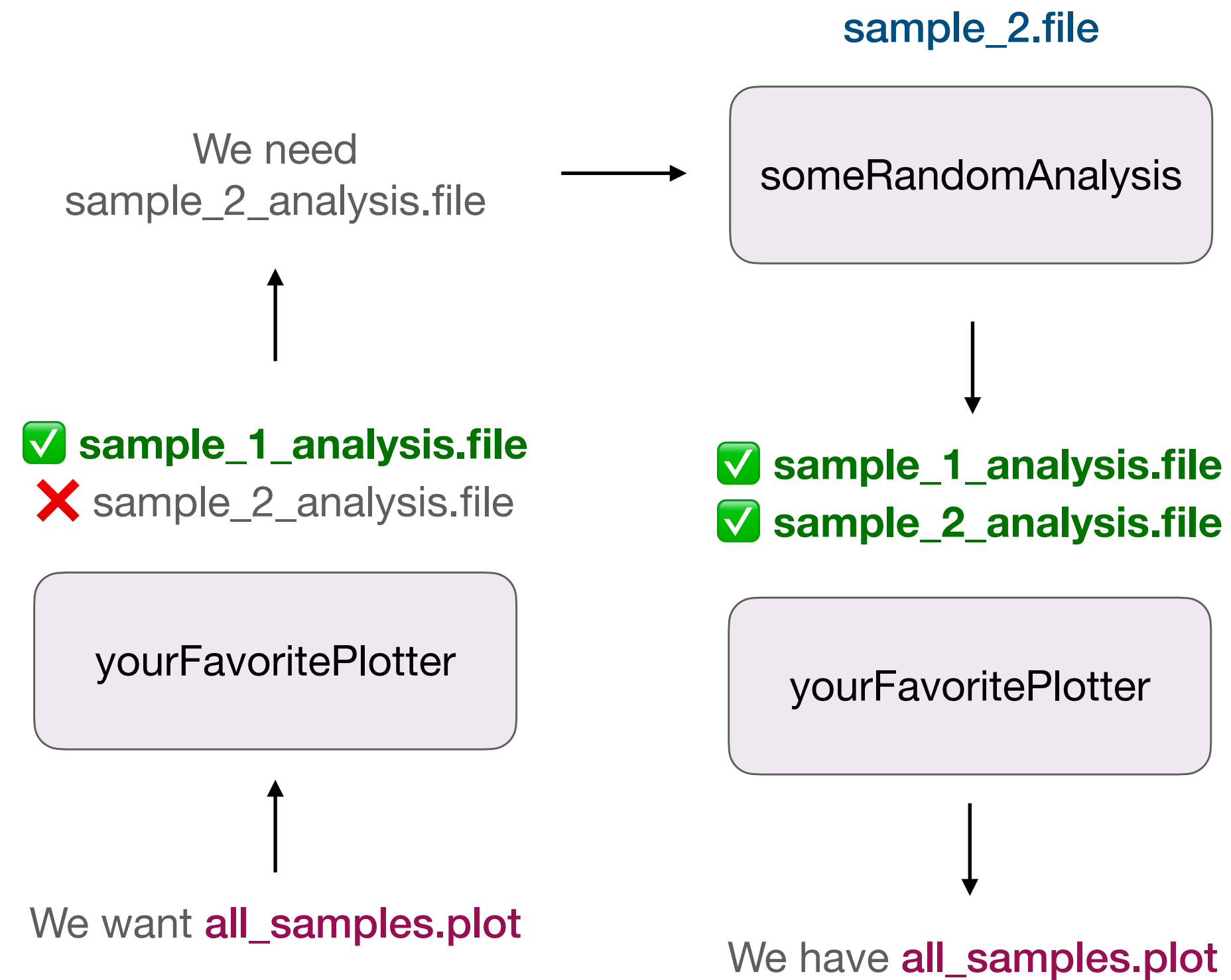
Let's start with a simple toy example.

We have two samples: we need to run a certain analysis on each sample, then create a summary plot.

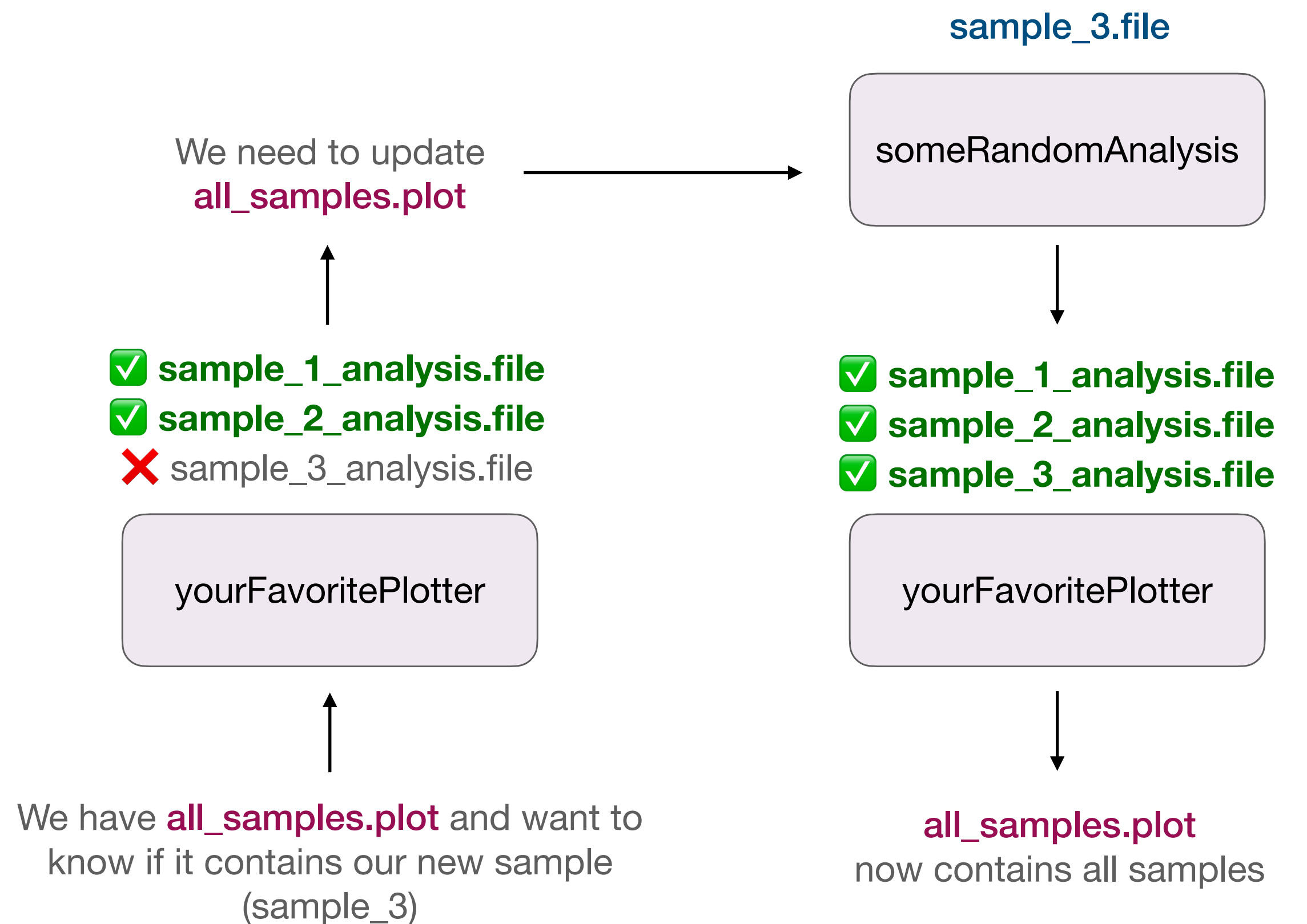


Now, let's look at how snakemake would handle this workflow.





Checks for presence of existing files,  
only queues missing files



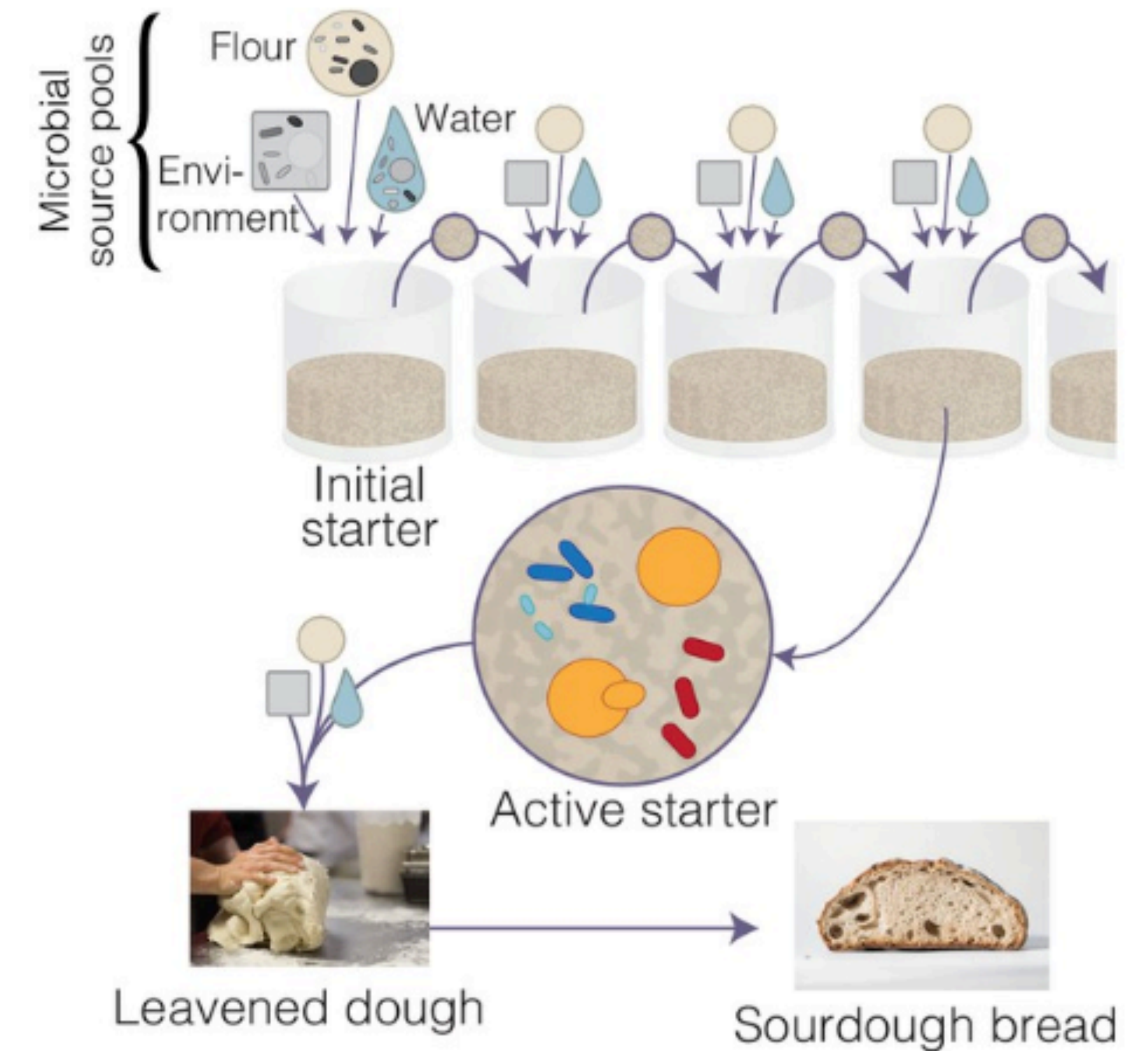
Check that all outputs are current,  
re-run if outdated

# Example workflow: short-read analysis

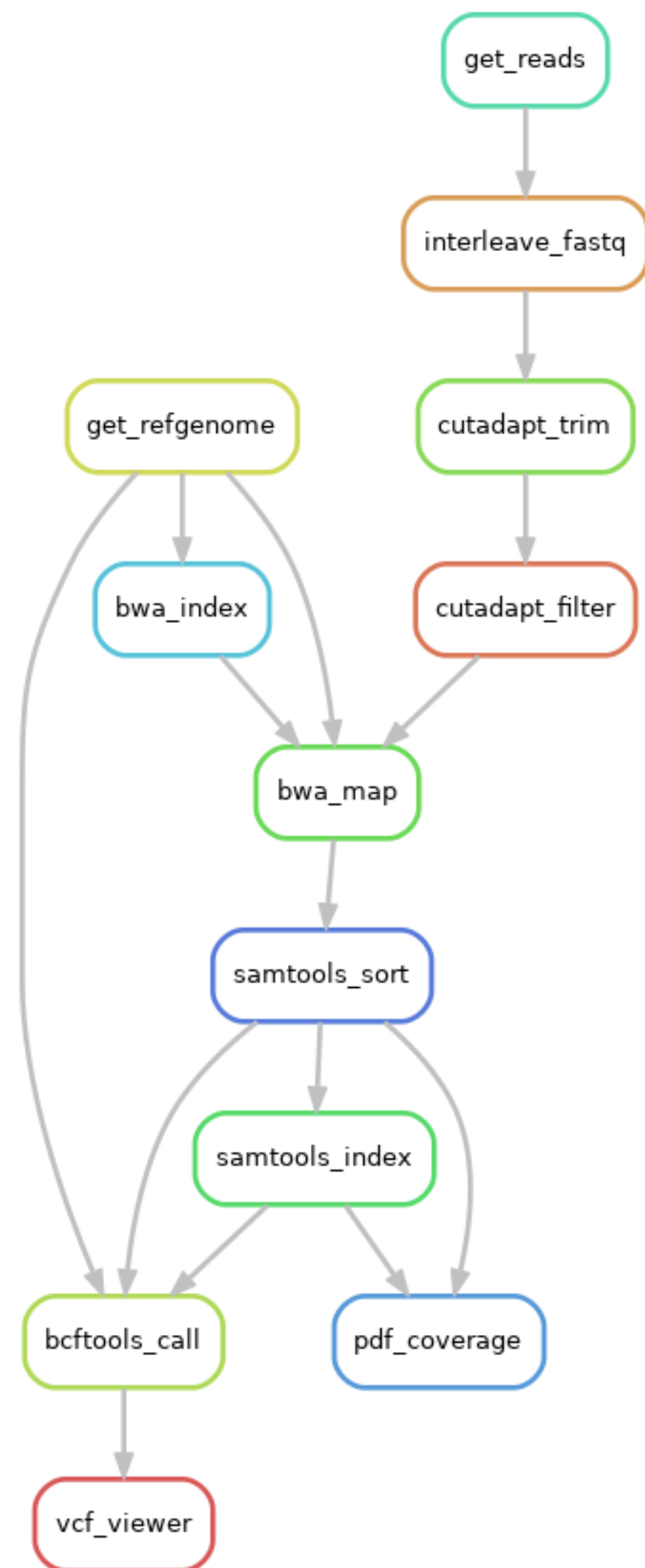
Imagine that you are a fabulously wealthy researcher, and your two greatest passions are genomics and baking sourdough bread.

You decide to conduct a study of sourdough starters across the world, aiming to characterize genetic diversity of *Fructilactobacillus sanfranciscensis* (the dominant species in most sourdough starters).

Once the data comes back, you realize that it's been weeks (months?) since you've actually baked any sourdough: how can you expedite the analysis process and get back to your *real* passion?



# Example workflow: short-read analysis



1. Download short-read data and reference genome
2. Preprocess reads (trim, filter, etc)
3. Map reads to reference genome
4. Sort .bam file
5. Index .bam file
6. Call variants
7. Create summary plots comparing:
  - Coverage per sample
  - Variants per sample

# Setting up your workflow



# Disclaimer

Everything I'm about to show you is based on my *personal* research needs and preferences.

You *do not* have to do everything I do, especially if it doesn't make sense for what you want to do!

The goal is for you to **learn how and why these pieces fit together**, and once you've gotten some practice, you'll be able to tailor your configuration to your own needs.



# Getting started

```
├── config
│   ├── config.yml
│   ├── subset.tsv
│   └── all_samples.tsv
├── envs
│   ├── snakemake_fa23.yml
│   ├── get_data.yml
│   ├── preprocessing.yml
│   └── ...
├── rules
│   ├── get_data.smk
│   ├── preprocessing.smk
│   └── ...
├── * data
├── * output
├── .gitignore
├── README.md
└── Snakefile
```

This should be what your working directory looks like.

As we go along, I'll highlight the location of each component we're discussing, along with a peek at its contents.

*\* created during workflow*

# Setting up your sample table

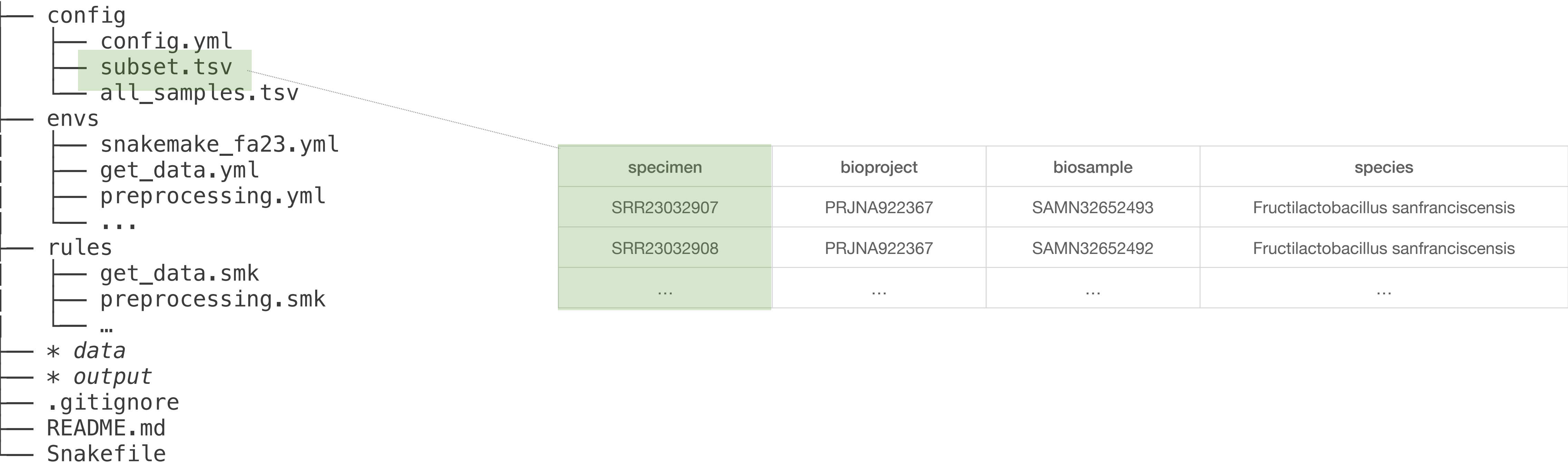
Let’s return to method acting: we’re fabulously wealthy, and we have many samples to analyze.

A sample table for snakemake is a basic tab-delimited file: you can use whatever method you like to create it.

At minimum, this table needs to have a single column with your **sample IDs**. You can also fill in more columns with other metadata that you may want to use in your workflow.

specimen	bioproject	biosample	species
SRR23032907	PRJNA922367	SAMN32652493	Fructilactobacillus sanfranciscensis
SRR23032908	PRJNA922367	SAMN32652492	Fructilactobacillus sanfranciscensis
...	...	...	...

# Where is the sample table?



\* created during workflow

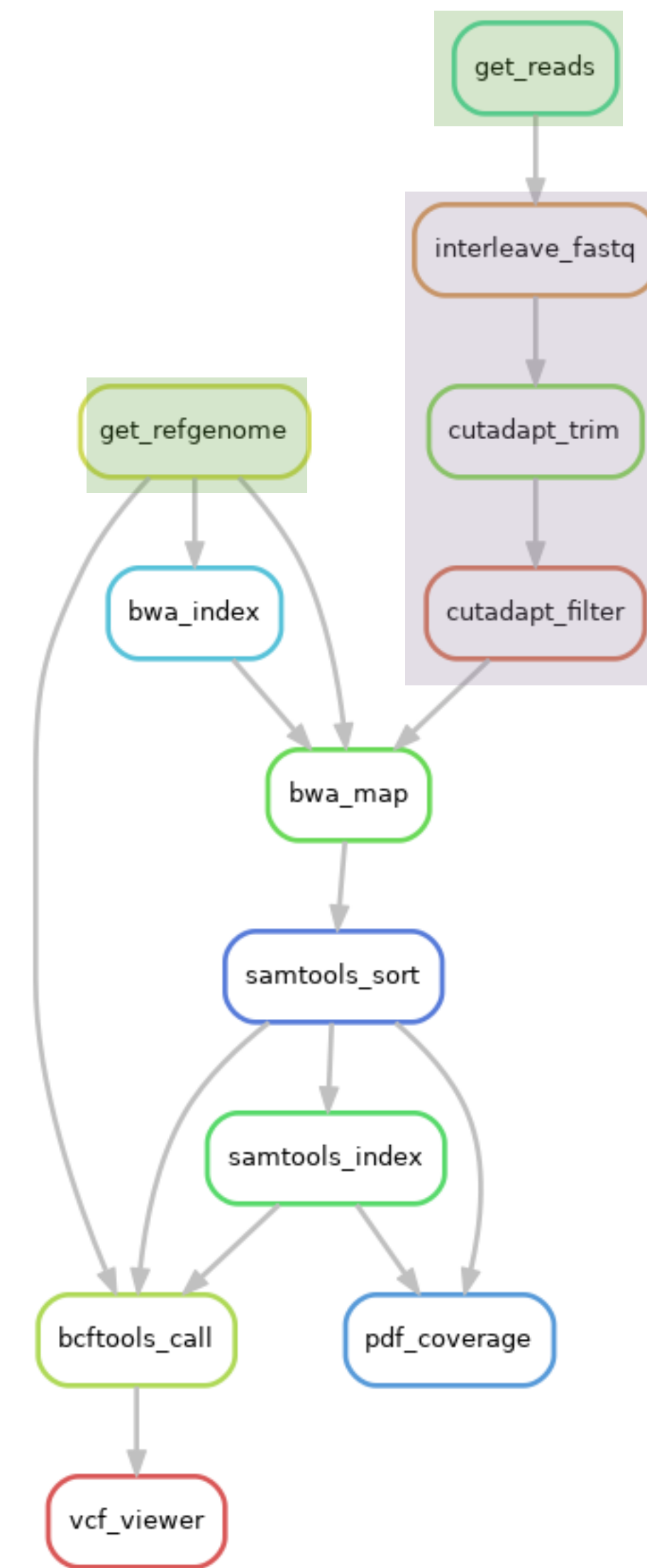
# Setting up “rules”

Each node of our workflow graph corresponds to a snakemake “rule”.

Each “rule” describes how a task should be run.  
Rules are stored in files with .smk (“snakemake”) extensions.

Consider each .smk as a module: ideally, you want to write them so they can be mixed and matched as necessary!

Do not make a giant master.smk file, I beg you.



- config
  - config.yml
  - subset.tsv
  - all\_samples.tsv
- envs
  - snakemake\_fa23.yml
  - get\_data.yml
  - preprocessing.yml
  - ...
- rules
  - get\_data.smk
  - preprocessing.smk
  - ...
- \* data
- \* output
- .gitignore
- README.md
- Snakefile

\* created during workflow

# .smk files should contain multiple related rules

```
├── config
│   ├── config.yml
│   ├── subset.tsv
│   └── all_samples.tsv
├── envs
│   ├── snakemake_fa23.yml
│   ├── get_data.yml
│   ├── preprocessing.yml
│   └── ...
├── rules
│   ├── get_data.smk
│   └── preprocessing.smk
│   └── ...
├── * data
├── * output
├── .gitignore
├── README.md
└── Snakefile
```

*\* created during workflow*

```
rule interleave_fastq:
    # Reformats the independent read 1 and read 2 fastqs into a single interleaved fastq using a script from bmap.
    # After the interleaved file is created, the *_R1.fastq and *_R2.fastq files will be cleaned up.
    input:
        r1 = "data/{id}_1.fastq",
        r2 = "data/{id}_2.fastq",
    output:
        interleaved = "data/{id}.fastq.gz"
    conda:
        "../envs/preprocessing.yml"
    threads: 10
    shell:
        """
        reformat.sh in1={input.r1} in2={input.r2} out={output.interleaved}
        """

rule raw_fastqc:
    # Creates a report describing the length and quality of the raw (not trimmed or filtered) reads.
    input:
        fastq = "data/{id}.fastq.gz"
    output:
        "output/fastq/raw/{id}.html",
        "output/fastq/raw/{id}.zip"
    conda:
        "../envs/preprocessing.yml"
    threads: 5
    params:
        outdir = "output/fastqc/raw"
    shell:
        """
        fastqc {input.fastq} -d {params.outdir}
        """

...
```

# Anatomy of a rule

```
rule cutadapt_trim:
    # Trims pesky sequencing adapters from reads.
    input:
        fastq = "data/{id}.fastq.gz"
    output:
        trimmed = temp("output/cutadapt/{id}.trimmed.fastq.gz")
    conda:
        "../envs/preprocessing.yml"
    threads: 5
    params:
        overlap = config['cutadapt']['overlap']
    log:
        "logs/cutadapt/{id}.trimmed.json"
    shell:
        """
        cutadapt --interleaved \
        -A CTGTCTCTTATACACATCT \
        -G AGATGTGTATAAGAGACAG \
        -A CAAGCAGAAGACGGCATACGAGAT \
        -G AATGATACGGCGACCACCGA \
        -B TCTACACATATTCTCTGTC \
        --cores={threads} \
        -O {params.overlap} \
        -o {output.trimmed} \
        --json={log} \
        {input.fastq}
        """
```

Let's start with the basics.

**input:** Declares paths to input files necessary for the task.

We use **wildcards** to indicate a variable part of the file path.

Example: data/SRR2303207.fastq.gz and data/SRR2303208.fastq.gz vary by the indicated {id} component in data/{id}.fastq.gz.

If the target output file is output/cutadapt/SRR2303207.trimmed.fastq.gz, snakemake understands to form-fill every instance of {id} with SRR2303207.

**output:** Declares paths to output files that you expect to be generated by the task.

**shell:** The shell command to run to execute this task.

**Note that you are not limited to only shell commands!**

I just don't have time to cover every type of task/command that snakemake can do. *\*sad snake noises\**



# Anatomy of a rule

```
rule cutadapt_trim:
    # Trims pesky sequencing adapters from reads.
    input:
        fastq = "data/{id}.fastq.gz"
    output:
        trimmed = temp("output/cutadapt/{id}.trimmed.fastq.gz")
    conda:
        "../envs/preprocessing.yml"
    threads: 5
    params:
        overlap = config['cutadapt']['overlap']
    log:
        "logs/cutadapt/{id}.trimmed.json"
    shell:
        """
        cutadapt --interleaved \
        -A CTGTCTCTTATACACATCT \
        -G AGATGTGTATAAGAGACAG \
        -A CAAGCAGAAGACGGCATACGAGAT \
        -G AATGATACGGCGACCACCGA \
        -B TCTACACATATTCTCTGTC \
        --cores={threads} \
        -O {params.overlap} \
        -o {output.trimmed} \
        --json={log} \
        {input.fastq}
        """
```

Next, the optional:

**threads:** Declares the *maximum number* of threads to be occupied in one task execution.

**log:** Declares a path to a logfile: useful if your task writes to a log, or if you want to pipe stderr to a logfile.

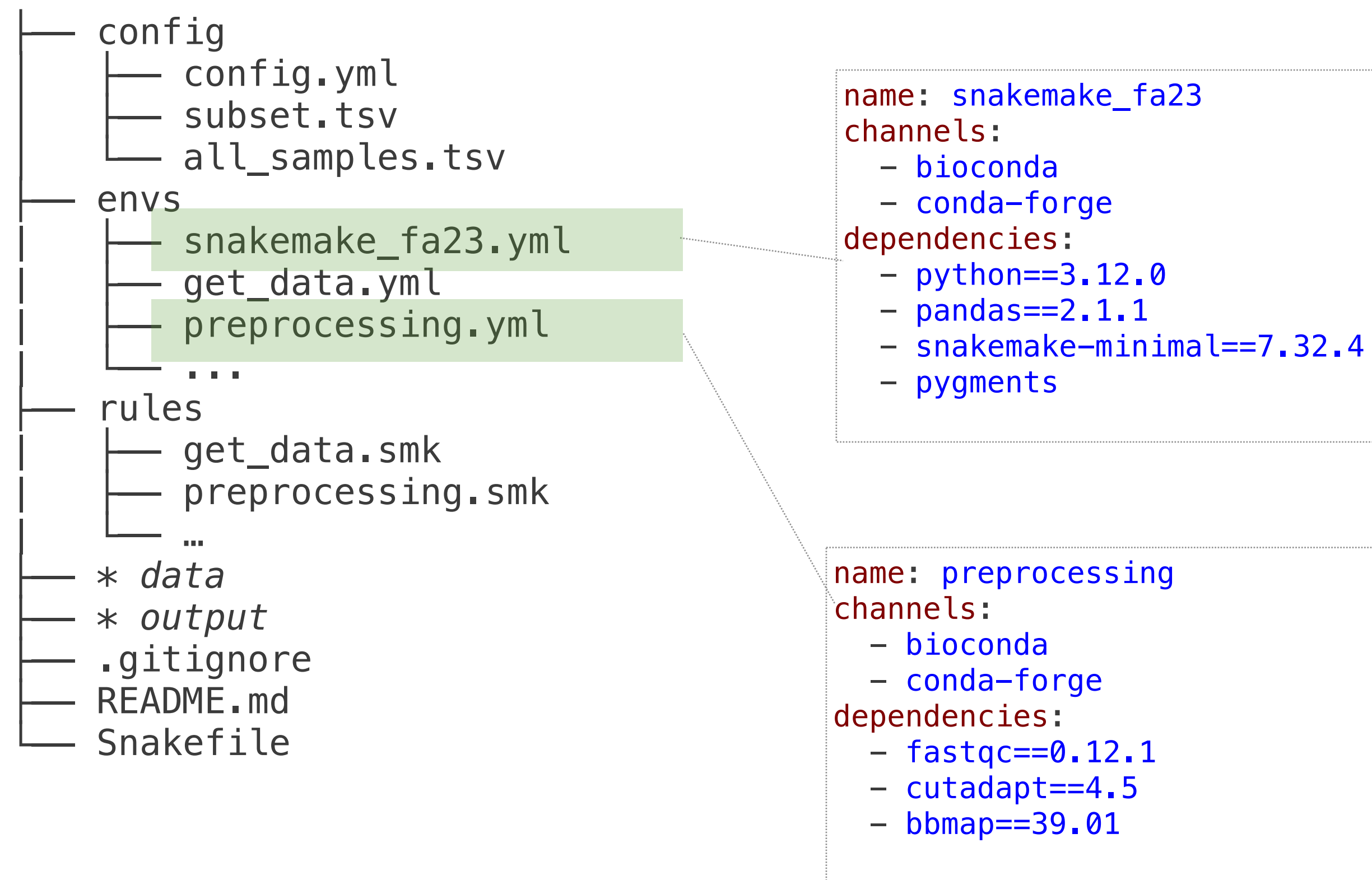
**conda:** A path to a .yml file describing the conda environment you want to use when executing this task.

**I STRONGLY recommend that you use this option.**

By creating an isolated environment for each .smk module, you can minimize the risk of breaking your analyses when you need to add, remove, or upgrade packages in your environment.



# Anatomy of a conda .yml



\* created during workflow

My general rules for filling out these files:

- **Only list the absolute core packages.**  
This file should be easy to read, and conda will install relevant sub-dependencies
- **Pin versions if functionality updates occur frequently and may break your analysis.**  
*Example:* pandas changed data frame syntax a year or so ago, and it broke several packages I use.

# Setting up a configuration file

```
├── config
│   ├── config.yml
│   ├── subset.tsv
│   └── all_samples.tsv
├── envs
│   ├── snakemake_fa23.yml
│   ├── get_data.yml
│   ├── preprocessing.yml
│   └── ...
├── rules
│   ├── get_data.smk
│   ├── preprocessing.smk
│   └── ...
├── * data
├── * output
├── .gitignore
├── README.md
└── Snakefile
```

```
#####
### Project configuration ###
#####

sample_table: "config/subset.tsv"

#####
### Analysis config options ###
#####

reference:
  ftp: 'https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/...'
  fasta: "data/GCF_009496975.1_ASM949697v1_genomic.fna"

fastq-dump:
  n_reads: 100000

cutadapt:
  overlap: 5
  min_length: 30
```

Your config file is like a reference for your workflow settings.

**At minimum, you should include the sample table path.**

The entries here are parsed by snakemake into a list of lists, stored under the config variable.

*\* created during workflow*

```
#####
### Project configuration ###
#####

sample_table: "config/subset.tsv"

#####
### Analysis config options ###
#####

reference:
  ftp: 'https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/...'
  fasta: "data/GCF_009496975.1_ASM949697v1_genomic.fna"

fastq-dump:
  n_reads: 100000

cutadapt:
  overlap: 5
  min_length: 30
```

```
rule cutadapt_trim:
  # Trims pesky sequencing adapters from reads.
  input:
    fastq = "data/{id}.fastq.gz"
  output:
    trimmed = temp("output/cutadapt/{id}.trimmed.fastq.gz")
  conda:
    "../envs/preprocessing.yml"
  threads: 5
  params:
    overlap = config['cutadapt']['overlap']
  log:
    "logs/cutadapt/{id}.trimmed.json"
  shell:
    """
    cutadapt --interleaved \
    -A CTGTCTCTTATACACATCT \
    -G AGATGTGTATAAGAGACAG \
    -A CAAGCAGAAGACGGCATACGAGAT \
    -G AATGATACGGCGACCACCGA \
    -B TCTACACATATTCTCTGTC \
    --cores={threads} \
    -O {params.overlap} \
    -o {output.trimmed} \
    --json={log} \
    {input.fastq}
    """
```

# Setting up your “Snakefile”

```
— config
  |— config.yml
  |— samples.tsv
— envs
  |— snakemake_fa23.yml
  |— get_data.yml
  |— preprocessing.yml
  |— ...
— rules
  |— get_data.smk
  |— preprocessing.smk
  |— ...
— * data
— * output
— .gitignore
— README.md
— Snakefile
```

*\* created during workflow*

We’ve reached the final part of our setup!

A Snakefile declares the relevant rules and config file for the workflow. In this manner, you can quick-swap .smks and config files to alter existing workflows or build new ones.

```
configfile: "config/config.yml"

include: "rules/common_utils.smk"
include: "rules/get_data.smk"
include: "rules/preprocessing.smk"
include: "rules/mapping.smk"
include: "rules/call_variants.smk"
```

This is why I said to not make one giant master.smk.

# Making snakes make

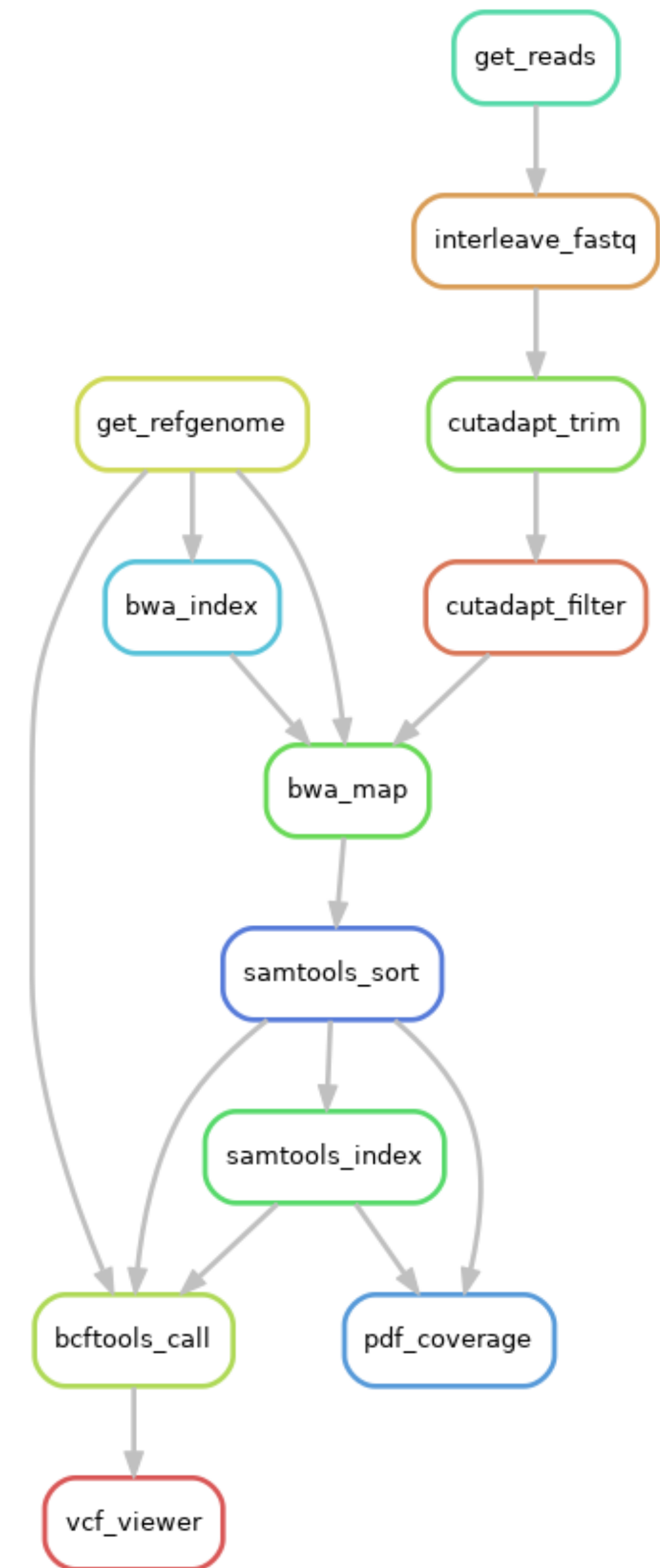
You can direct snakemake to generate a target file using the following command:

```
snakemake -pj{n} --use-conda {file path here}
```

n is the max number of jobs you want snakemake to run in parallel.

My general rule of thumb is to just set n to the number of cores I have available.

```
# example: set n = 10  
snakemake -pj10 --use-conda {file path here}
```





# Testing the example workflow

In our example workflow, we generate several summary plots.

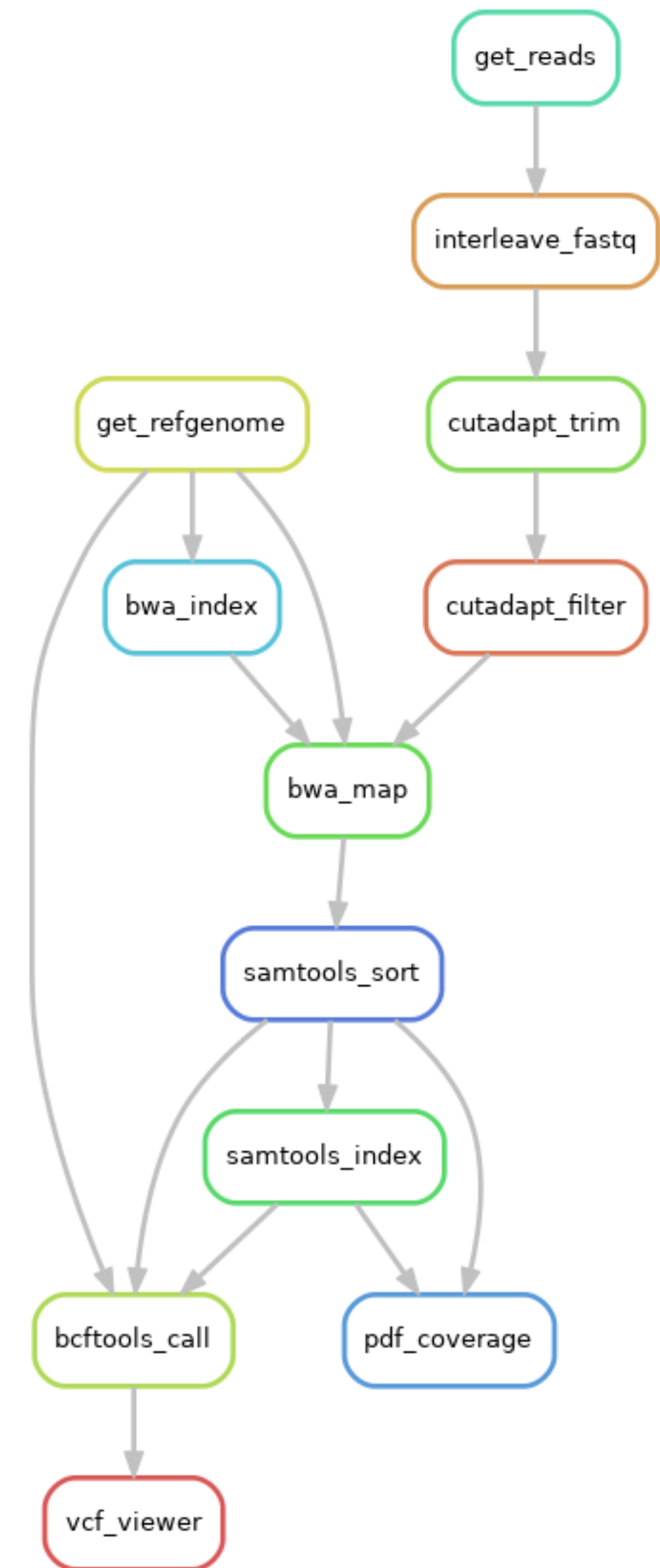
Let's say that we want to generate the vcf heatmap.

The *best* way to test out the workflow is to do a “dry run”.

```
snakemake -pj{n} --use-conda output/visuals/vcf_heatmap.pdf -np
```

The **-n** flag tells snakemake to “dry run” the workflow, and the **-p** flag tells snakemake to print the details.

This is handy for making sure that your workflow is actually working as intended.



# Running the example workflow

Now that we've gone through all the nitty gritty details, try it out on your own!

```
snakemake -pj{n} --use-conda output/visuals/vcf_heatmap.pdf -np
```

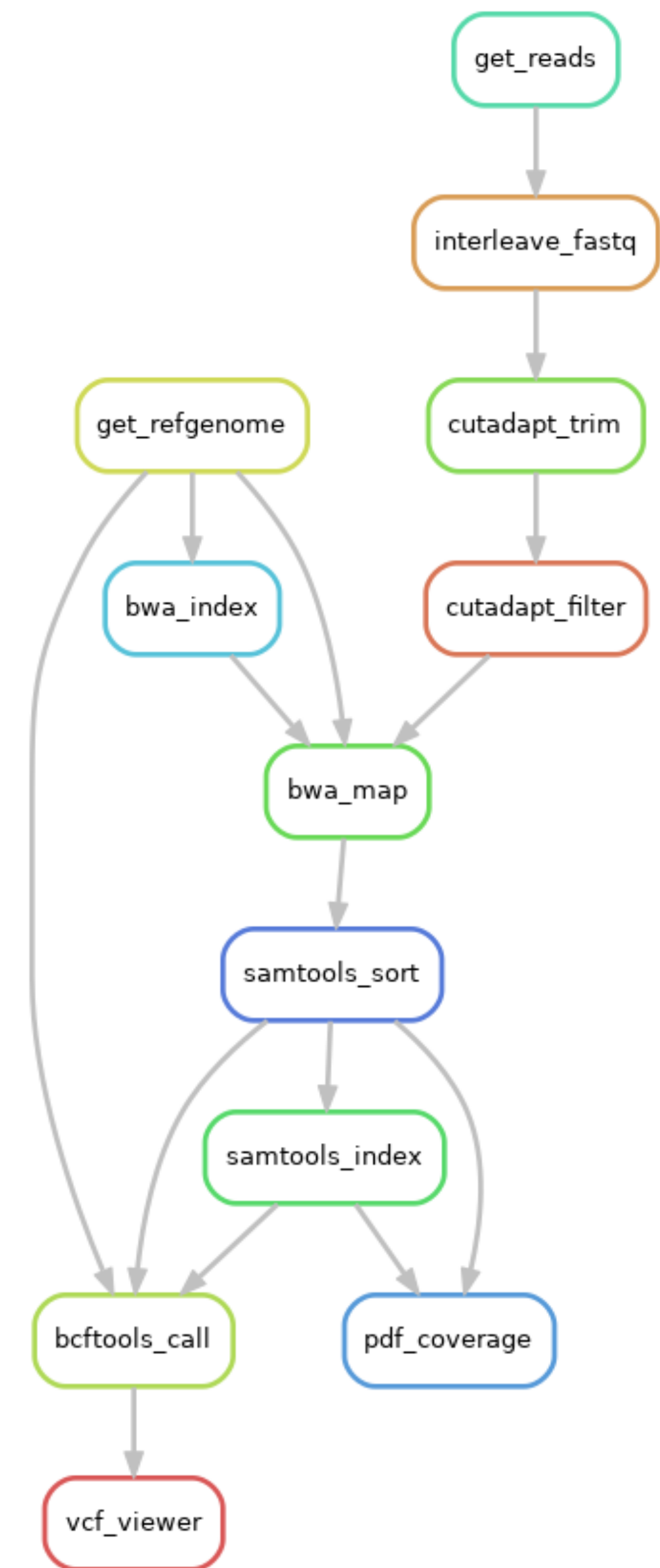
First, do a dry run with the `-np` flags set as shown above.

Once you've verified that everything works, remove the `-np` flag and run the workflow for real.

*Note:* Runtime will depend on your compute resources.

On DataHub, conda environment creation takes up to 30 minutes, so you may not be able to run the workflow yet.

Once that's done, running the workflow should take < 5 min.





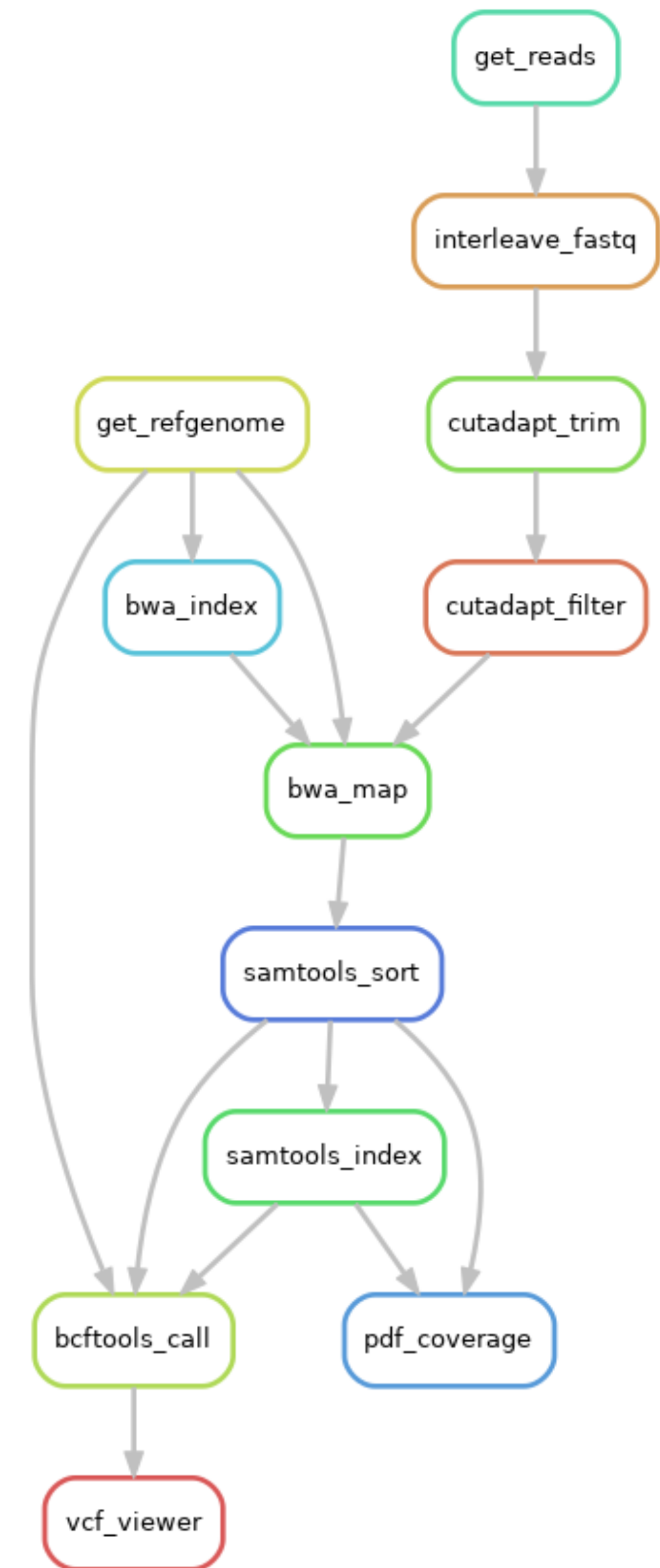
# Optional: Visualizing the example

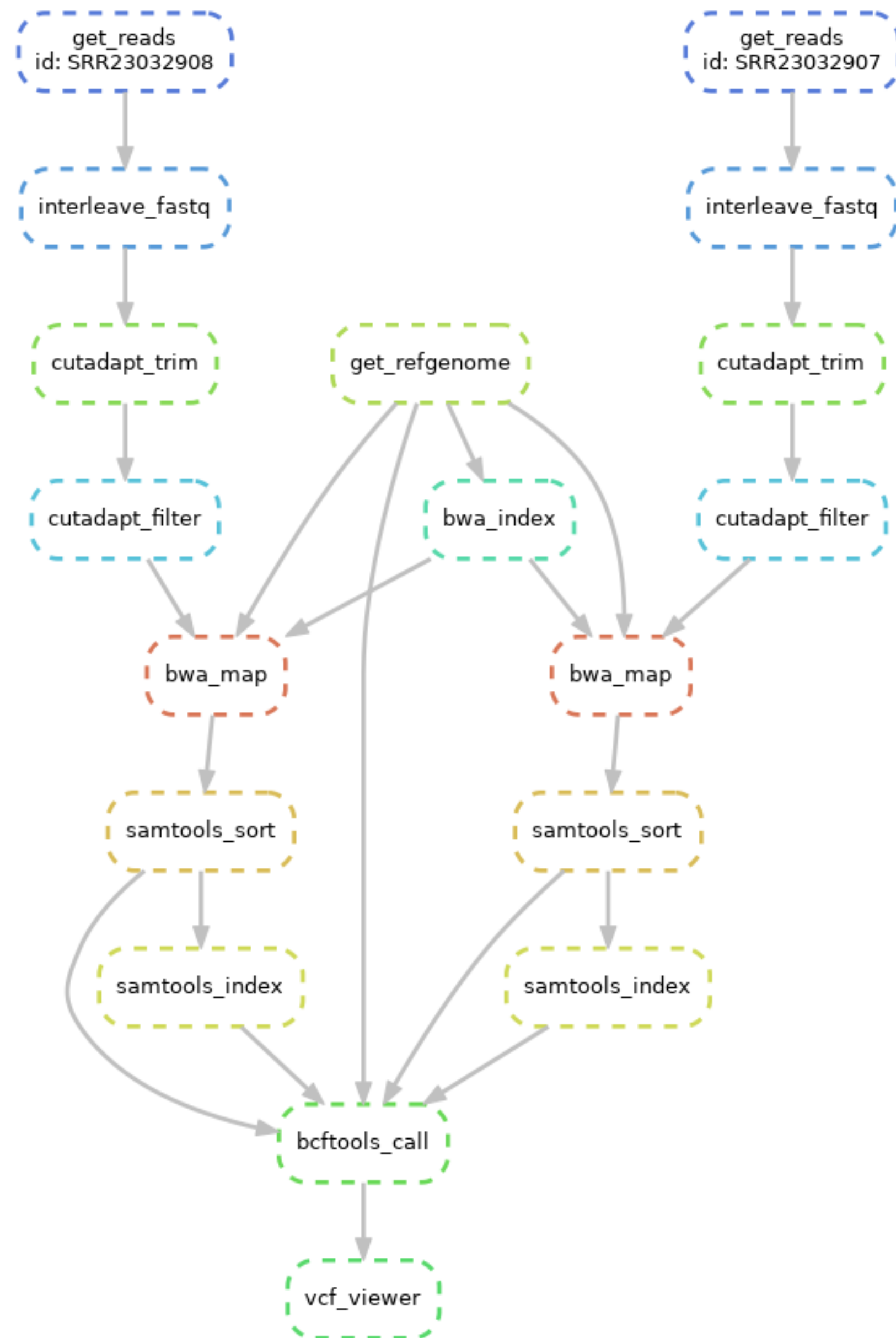
It's also a good idea to generate visualizations of your workflow as a sanity check.

This command generates the graph I've been showing throughout the tutorial.

```
snakemake --rulegraph --use-conda output/visuals/vcf_heatmap.pdf |  
dot -Tpng > output/visuals/workflow_rulegraph.png
```

It's a bit of a handful to type out, so you can copy this comment from the GitHub homepage ("Commands").





There are many ways for you to generate snakemake graphs!

# More ways that snakes can make things for *you*

Use “rule all” in your Snakefile to create a default set of target files!

```
rule all:
    input:
        coverage_html = "output/visuals/sample_coverage.html",
        coverage_pdf = "output/visuals/sample_coverage.pdf",
        vcf = "output/variants/all_samples.vcf.gz",
        vcf_heatmap = "output/visuals/vcf_heatmap.pdf"
```

You can write helper functions in Python and invoke them in your rules

```
def make_symlink_paths(wildcards):
    symlink_path_format = "data/PacBio-HiFi/{specimen}/{lane}
    {smrtcell}.ccs.bam"
    samples = pd.read_table("samples.tsv", index_col=False)
    samples = samples.to_records(index=False)
    symlink_paths = [symlink_path_format.format(specimen=s[0],
    lane=s[2], smrtcell = s[3]) for s in samples]
    return symlink_paths
```

snakemake can submit jobs to savio via SLURM!

```
snakemake --cluster "sbatch -A CLUSTER_ACCOUNT -t
CLUSTER_TIME -p CLUSTER_PARTITION -N CLUSTER_NODES"
-jobs NUM_JOBS_TO_SUBMIT
```

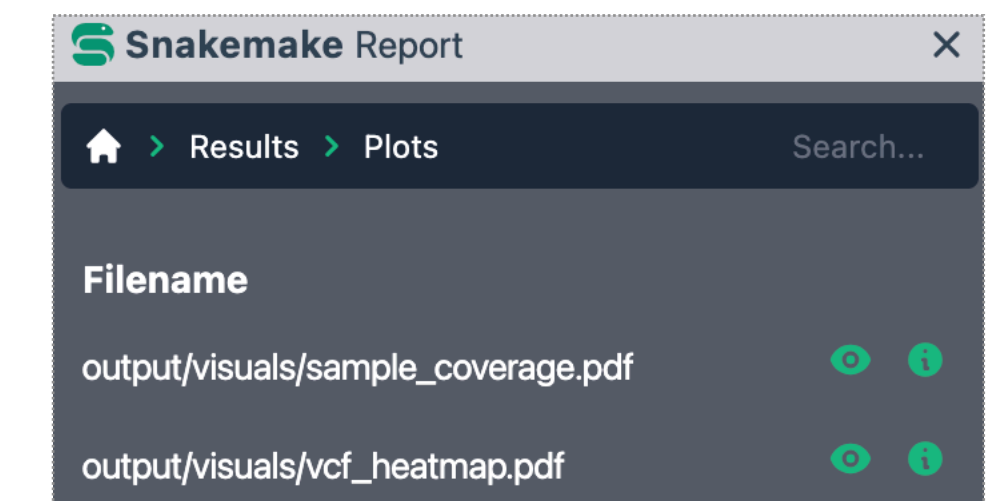
snakemake can retry failed jobs while adjusting resources (memory, etc)

```
rule bwa_map:
    input:
        ref="ref.fna",
        reads="{sample}.fq.gz",
    output:
        mapped=temp("{sample}.bam"),
    resources:
        mem_mb=lambda wildcards, attempt: 20000 + 5000 *
        (attempt - 1)
```

snakemake can run Jupyter notebooks (great for plots!)

```
rule hello:
    output:
        "test.txt"
    notebook:
        "notebooks/hello.py.ipynb"
```

snakemake can generate a report html file with key outputs and workflow metrics





snakemake can run Docker containers and Dockerize your workflow!

```
snakemake --containerize > Dockerfile
```

# Thanks for listening!

If you enjoyed today’s workshop, let’s stay in touch!

-  **Contact card:** [linktr.ee/stacy\\_li](https://linktr.ee/stacy_li)
-  **Personal website:** [stacy.li](https://stacy.li) (yes, *that’s the full URL*)
-  **Twitter:** [@evocryptid](https://twitter.com/evocryptid)

Interested in more seminars?  
Make sure to sign up for the CCB email list and bookmark our seminar website at [ccbskillssem.github.io](https://ccbskillssem.github.io).

credits & acknowledgements

The short-read *Fructilactobacillus sanfranciscensis* data used in this workshop is from Rogalski et al 2020.

The `vcfR` heatmap plotting script is a modified version of a script from Olawoye et al 2020.

Thank you to my wonderful research group, The Sudmant Lab. As always, I am especially for my advisor Dr. Peter Sudmant and mentor Dr. Juan Manuel Vazquez. Everything I do is only possible because I stand upon the shoulders of giants. 🌟

S	N	A	K	E
Forgot to change a path, accidentally overwrote a file	Returned from a break, now you can’t remember where these files came from	Paralyzing fear of rerunning your full pipeline	Unsure which script you <i>actually</i> used for analysis	Copy-pasting the same line of code manually for each sample
“Your code doesn’t work on my machine.” (no further info)	Eyes glazing over while checking list of 20+ full output file paths for completeness	“Why is this even installed?”	Painstakingly creating analysis flowchart for presentation	Something didn’t work, and you forgot to save the log file
Typos while editing output paths for every sample	Upgraded one (1) package, now nothing works	 <i>free space</i>	Unsure if you re-analyzed <i>all</i> samples with updated parameters	Nuking your base conda environment (again)
Manually saving 10+ plots with slightly different names and subsets	Checking to see if you can run your next job yet	Fighting mentally and near physically with gnu-parallel	Copy/pasting your code into different project folders	Accidentally changing only <i>some</i> file names
Manually tuning memory allocation (job killed anyway)	Analysis script is 1000 lines, but 400 of them are commented out	preprocessing.sh and preprocessing_taylors_version.sh	Mysteriously missing output files	Manually creating output folders, one by one



# the sudmant lab

the evolution and diversity of aging, genome structure, and mutation



*also please check out our lab's new review, featuring yours truly*

## Trends in Genetics

REVIEW | [ONLINE NOW](#)

### The evolution of aging and lifespan

[Stacy Li](#) • [Juan Manuel Vazquez](#) • [Peter H. Sudmant](#)  

