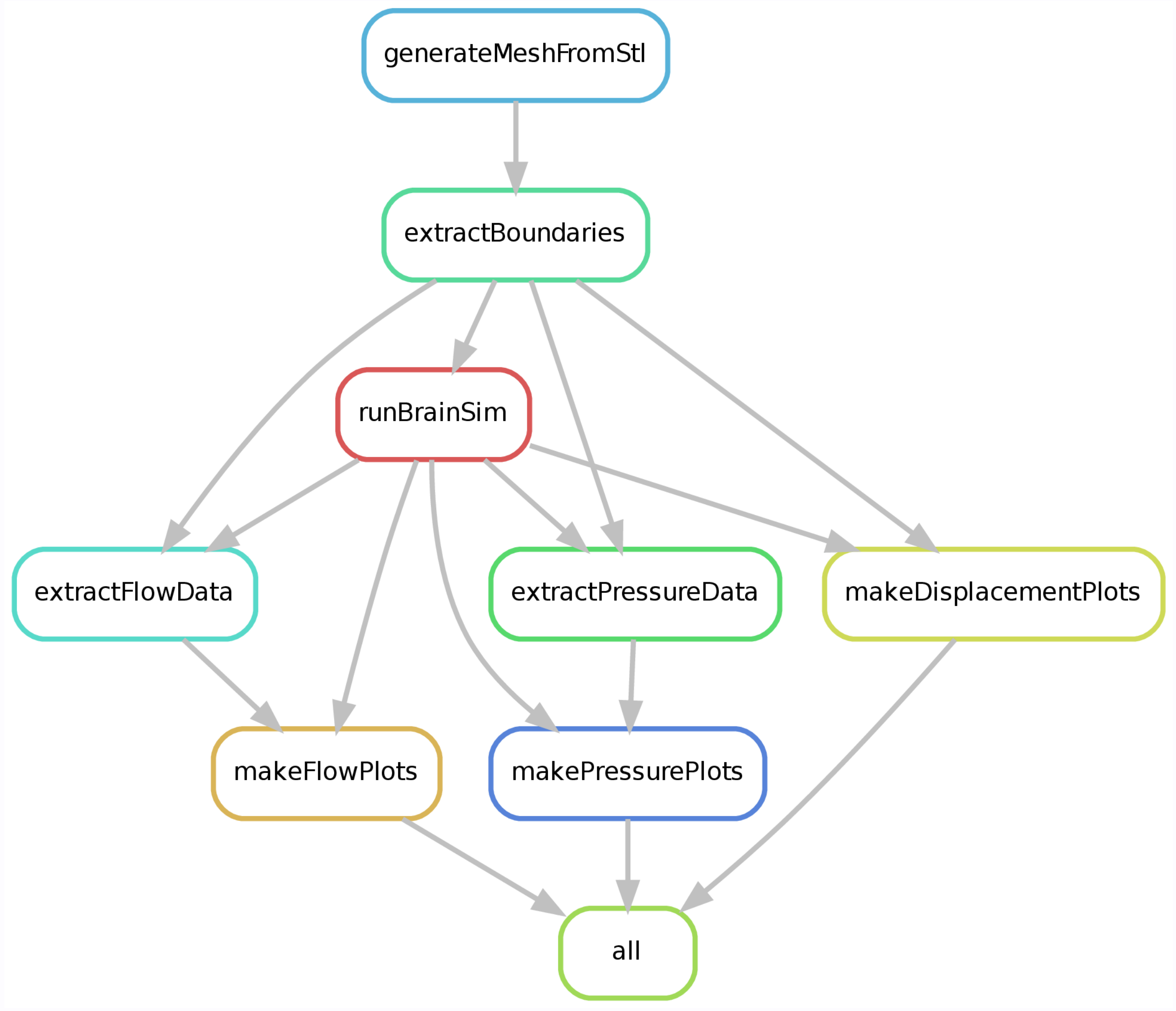
Manage your workflow with snakemake

A typical workflow



The workflow should be:

- easily executable
- portable
- well-documented
- scalable
- ...

Use a simple `pipeline.sh`?

- small changes require rerunning all steps
- does not scale easily
- ...

Snakemake principles

- decompose pipeline into rules
- rules define how to obtain output files from input files
- snakemake infers dependencies and execution order
- python based
  - any python code runs in your snakefile
  - easy to learn
- snakemake is file based
  - think of a consistent naming scheme for your files

Snakemake basics

Installation using conda:

```
conda create -c conda-forge -c bioconda -n snakemake snakemake
```

Create a `Snakefile` and specify each step as **rule**:

```
rule example_rule:
    input:
        "data/input.csv"
    output:
        "results/output.csv"
    shell:
        "python run.py data/input.csv results/output.csv"
```

Integration with conda

`env.yml` file:

```yaml
channels:
  - conda-forge
dependencies:
  - numpy
  - pandas
```

`Snakefile` file:

```
rule example:
    input:
        "data/input.csv"
    output:
        "results/output.csv"
    conda:
        "env.yml"
    shell:
        "python run.py ..."
```

run snakemake: `snakemake --use-conda -j1`

or use container...

Snakemake on HPC

- integration with SLURM
  - `cookiecutter gh:Snakemake-Profiles/slurm`
- specify resources
- dynamic resources, e.g.
  - `mem_mb=lambda wildcards, input, attempt: (input.size//1000000) * attempt * 10`
- repeat failed jobs (e.g. with more resources)

Other features

- automatic generation of unit tests
- integration with cloud computing (send your jobs to the cloud)
- generate reports
  - runtime statistics
  - visualization of the workflow topology
  - include any of the output files, e.g. plots
- visualize your workflow
  - `snakemake --rulegraph | dot -Tpdf > dag.pdf` or
  - `snakemake --dag | dot -Tpdf > dag.pdf`
- integrate jupyter notebooks
- access remote file systems