Managing dependencies

Managing and executing analysis workflow

Versioning and collaborating on code (and some other files)

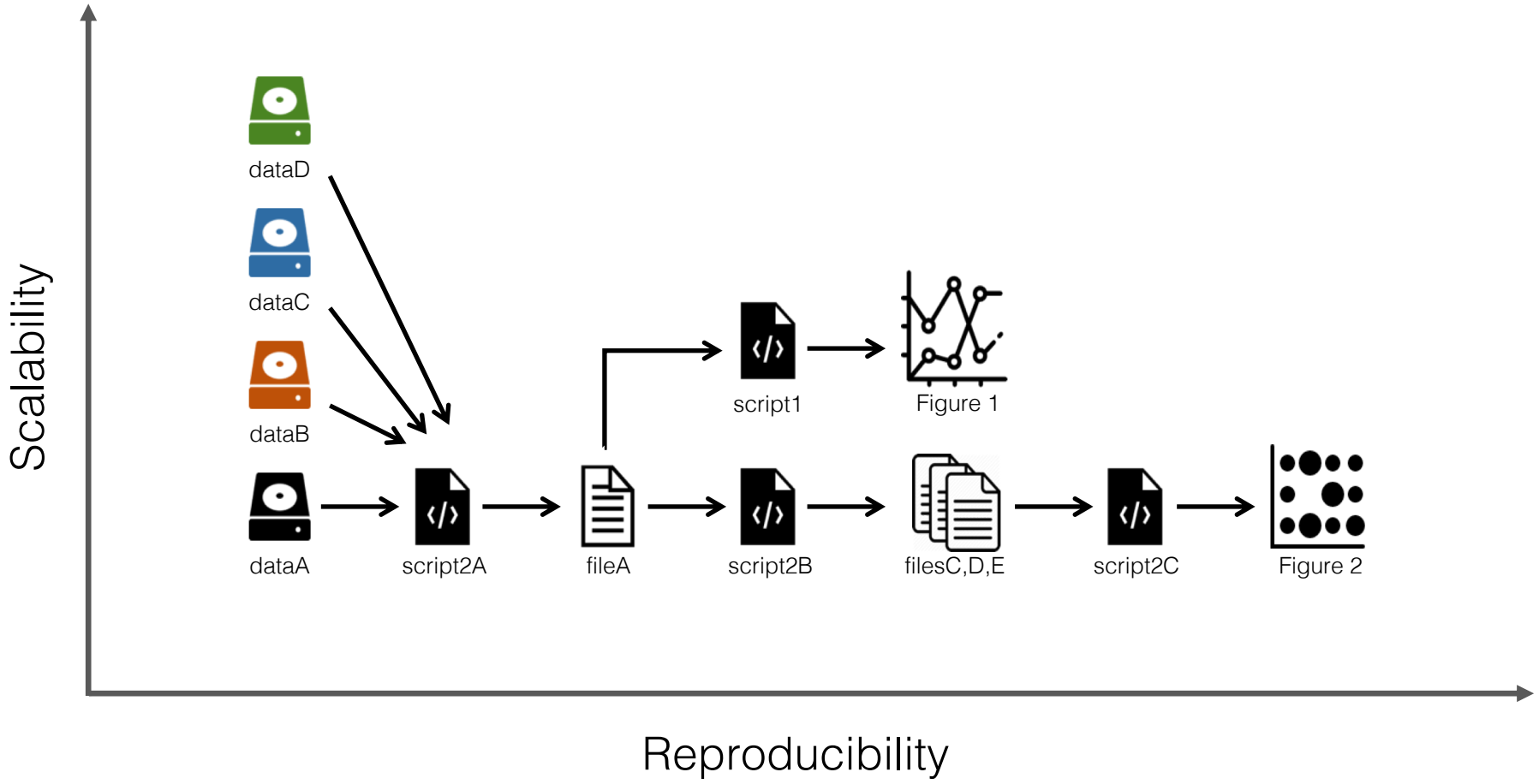
Connecting code and reporting

and…

Isolating and exporting environment

As projects grow or age, it becomes
increasingly difficult to keep track of all the parts
and how they fit together.

“Snakemake is a workflow management system that
aims to reduce the complexity of creating workflows by
providing a fast and comfortable execution environment,
together with a clean and modern specification
language in python style.”


Scalability
dataD
dataC
dataB
dataA
script2A
fileA
script1
Figure 1
script2B
filesC,D,E
script2C
Figure 2
Reproducibility

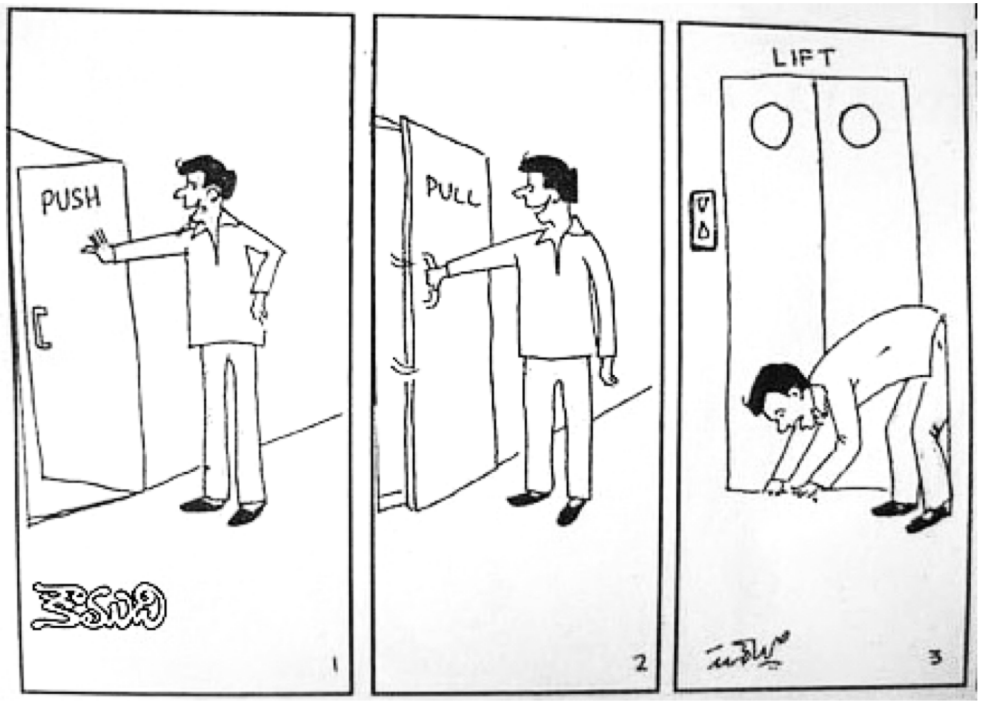
Workflow management systems come in different flavors.

Explicit syntax ("Push")

"Here are my inputs, please perform these operations in this order on them."

Implicit syntax ("Pull")

"I need this output, could you please figure out which operations to perform and in which order?"



Explicit approach using Bash

trim_and_zip.sh

```
for sample in *.fastq
do
    id=$(echo ${sample} | sed 's/.fastq//')

    # Trim fastq file
    echo “Trimming ${id}“
    seqtk trimfq -b 5 -e 10 $sample > \
    ${id}.trimmed.fastq

    # Compress fastq file
    echo “Compressing ${id}“
    gzip -c ${id}.trimmed.fastq > \
    ${id}.trimmed.fastq.gz

    # Remove intermediate files
    rm ${id}.trimmed.fastq
done
```

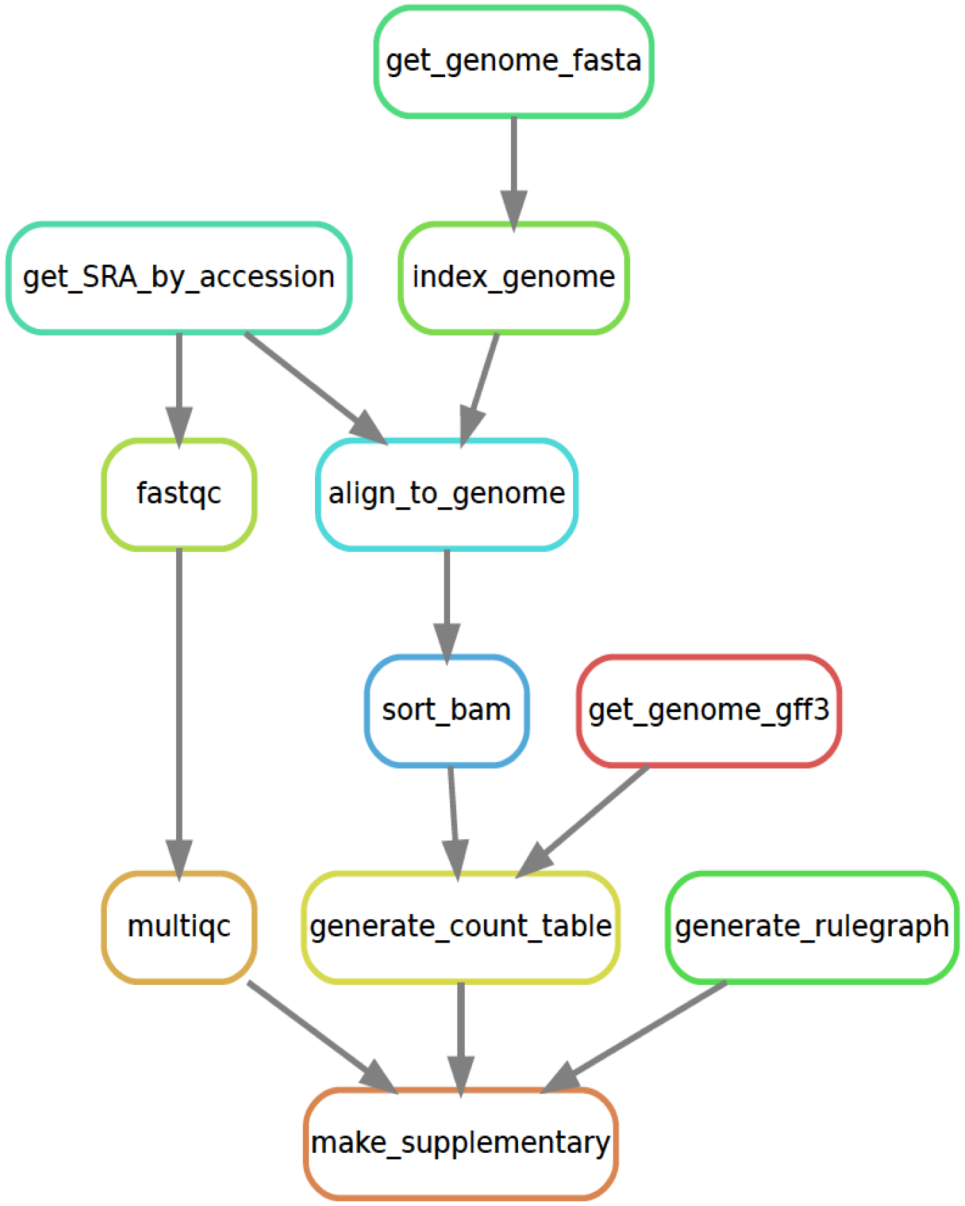
```
$bash trim_and_zip.sh
Trimming sample: a
Compressing sample: a
Trimming sample: b
Compressing sample: b
```

Implicit approach using Snakemake

Snakefile

```
rule trim_fastq:
    input: "{prefix}.fastq"
    output: temp("{prefix}.trimmed.fastq")
    shell:
        "seqtk trimfq -b 5 -e 10 {input} > {output}"

rule gzip:
    input: "{prefix}"
    output: "{prefix}.gz"
    shell:
        "gzip -c {input} > {output}"
```

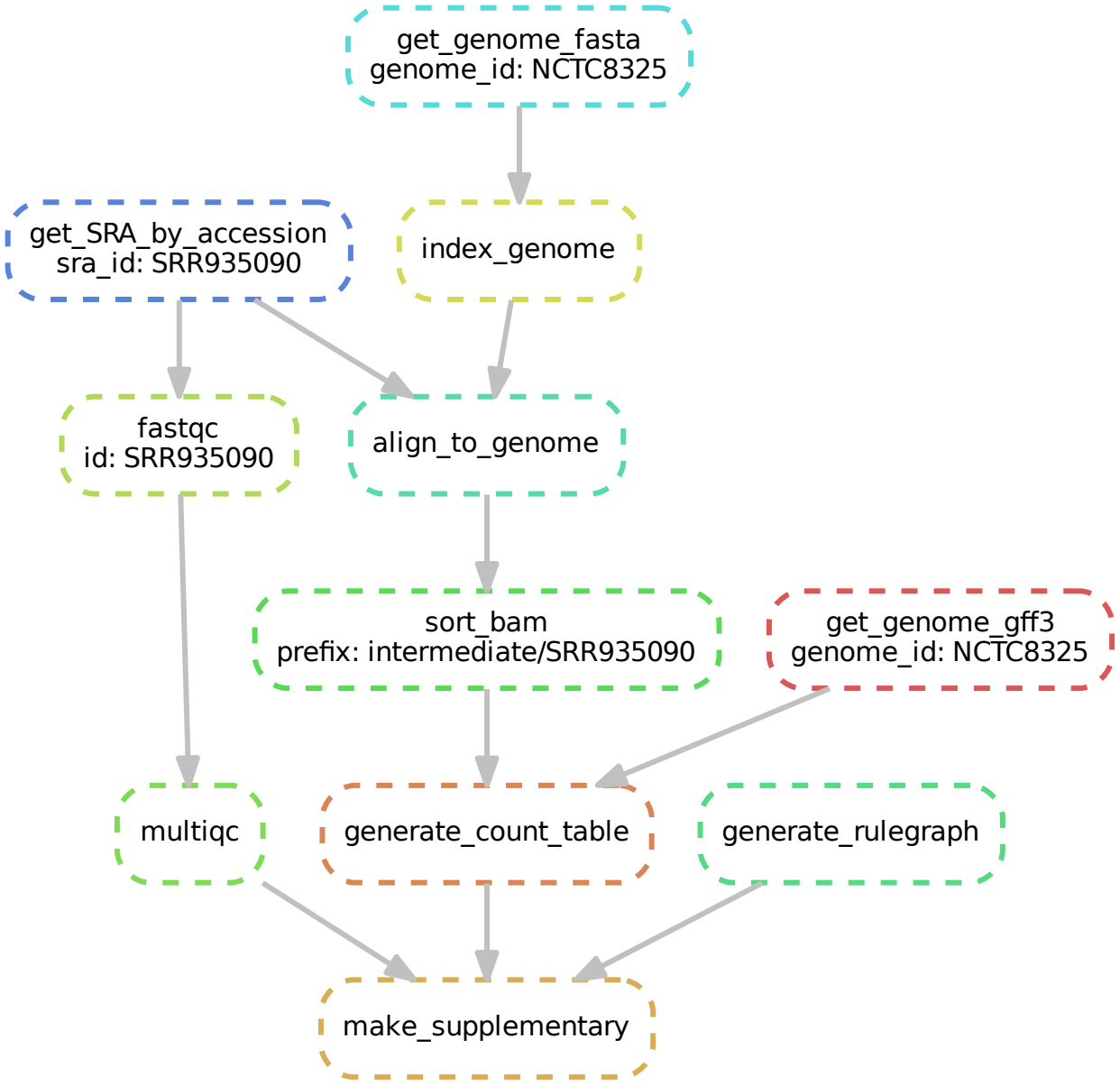
```
$snakemake {a,b}.trimmed.fastq.gz
Provided cores: 1
Rules claiming more threads will be
scaled down.
Job counts:
count   jobs
2       gzip
2       trim_fastq
4
rule trim_fastq:
    input: a.fastq
    output: a.trimmed.fastq
    wildcards: prefix=a
1 of 4 steps (25%) done
rule gzip:
    input: a.trimmed.fastq
    output: a.trimmed.fastq.gz
    wildcards: prefix=a.trimmed.fastq
Removing temporary output file a.trimmed.fastq.
2 of 4 steps (50%) done
rule trim_fastq:
    input: b.fastq
    output: b.trimmed.fastq
    wildcards: prefix=b
3 of 4 steps (75%) done
rule gzip:
    input: b.trimmed.fastq
    output: b.trimmed.fastq.gz
    wildcards: prefix=b.trimmed.fastq
Removing temporary output file b.trimmed.fastq.
4 of 4 steps (100%) done
```



Snakemake figures out how rules can be pieced together to generate some requested output.

Here we ask for `supplementary.pdf`, which is an R Markdown report generated by the rule `make_supplementary`.

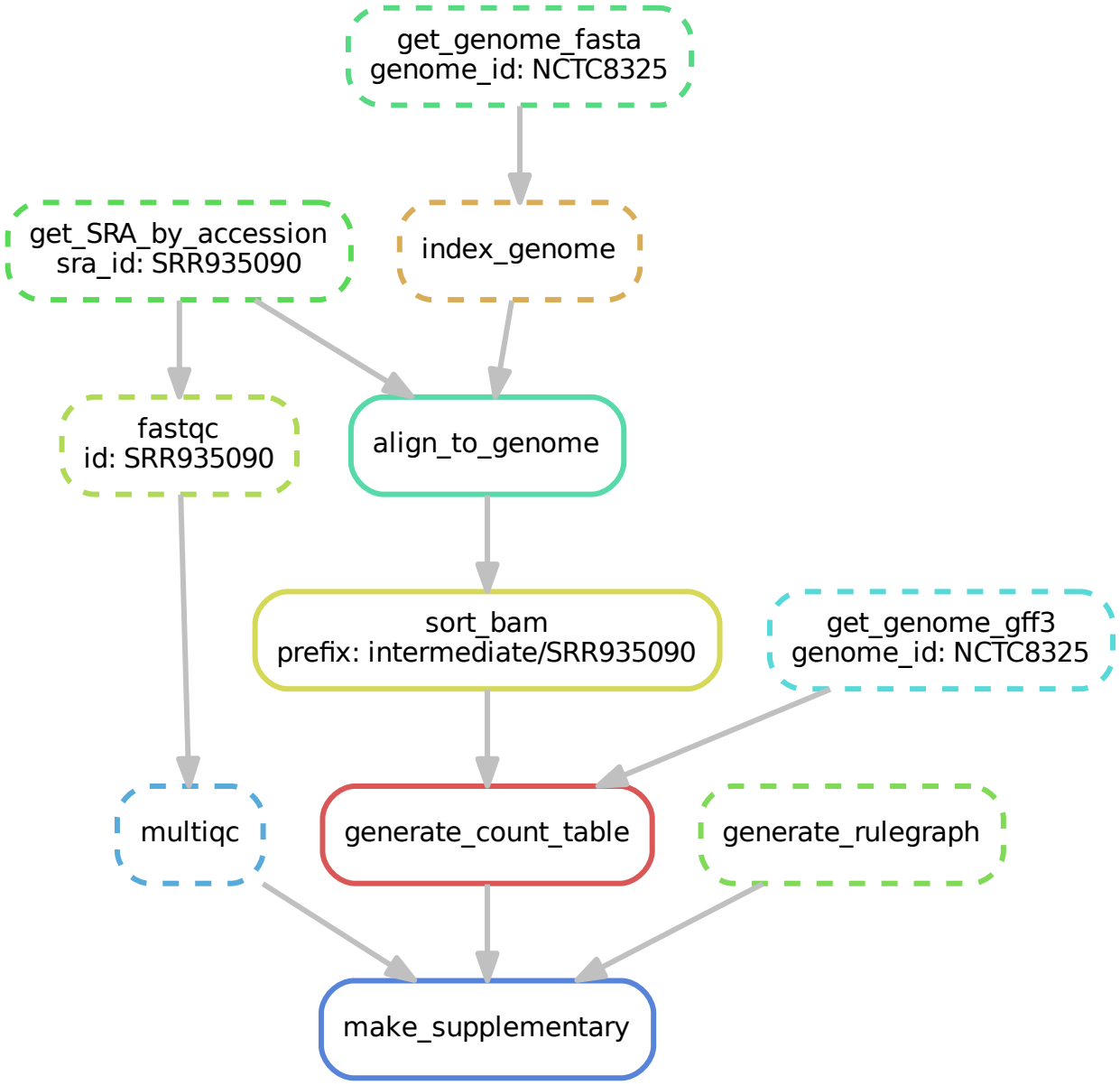
```
$snakemake supplementary.pdf --rulegraph | dot -Tpdf > rulegraph.pdf
```



Snakemake keeps track of when files were generated and by which rules.

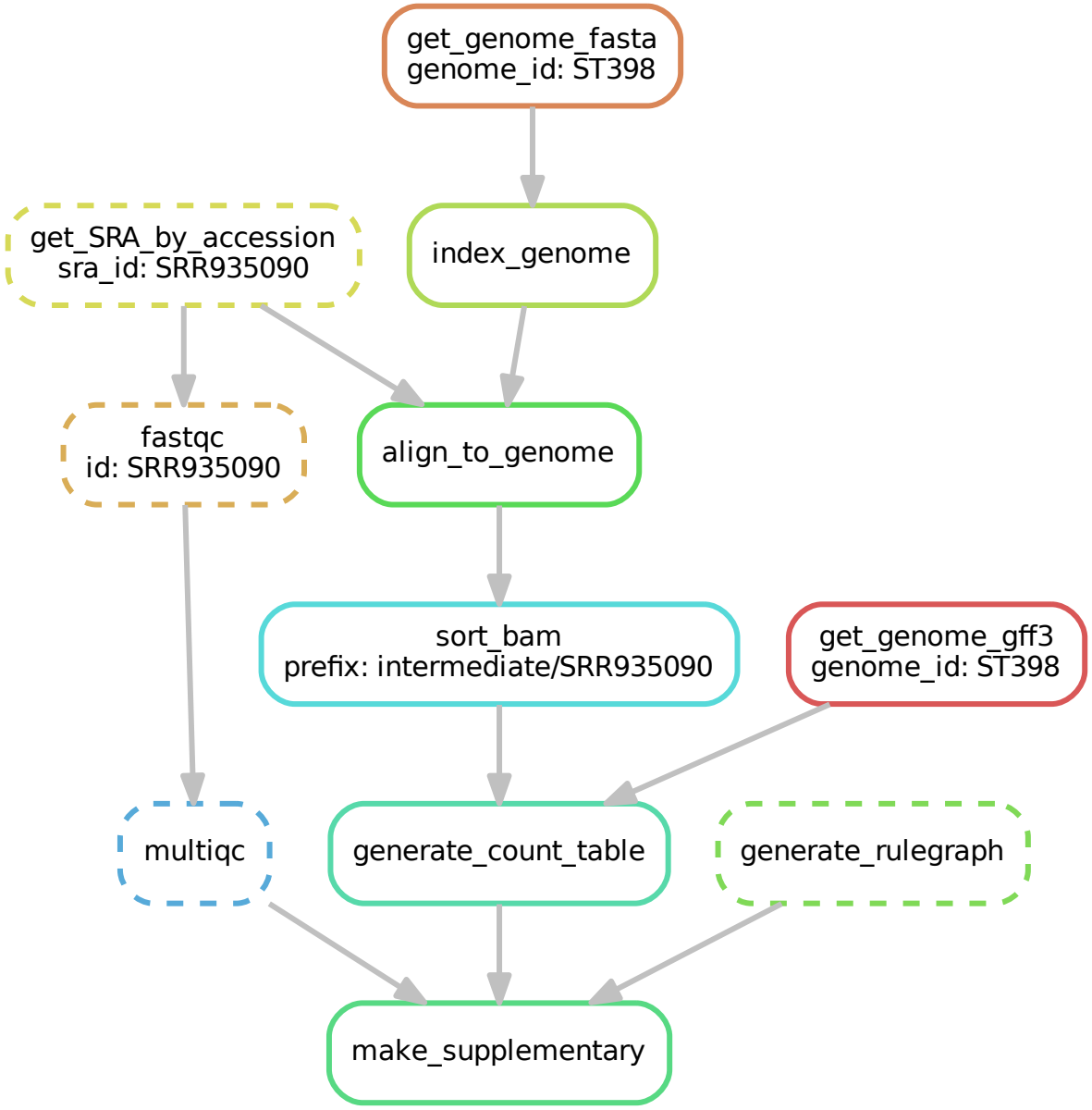
Dotted rule boxes show that `supplementary.pdf` already exists and that it's newer than its dependencies (recursively).

```
$snakemake supplementary.pdf --dag | dot -Tpdf > dag.pdf
```



Here Snakemake detects that a file used in `align_to_genome` is newer than downstream files, so it reruns the necessary rules.

```
$touch intermediate/NCTC8325.1.bt2
$snakemake supplementary.pdf --dag | dot -Tpdf > dag.pdf
```



Forcing a rule (`get_genome_fasta` here) to be rerun also leads to rerunning all rules that depend on it.

Note that we also change the parameter "genome_id" to use another genome to align to. This causes `get_genome_gff3` to be rerun as well.

```
$snakemake supplementary.pdf --config genome_id=ST398
        -f get_genome_fasta --dag | dot -Tpdf > dag.pdf
```

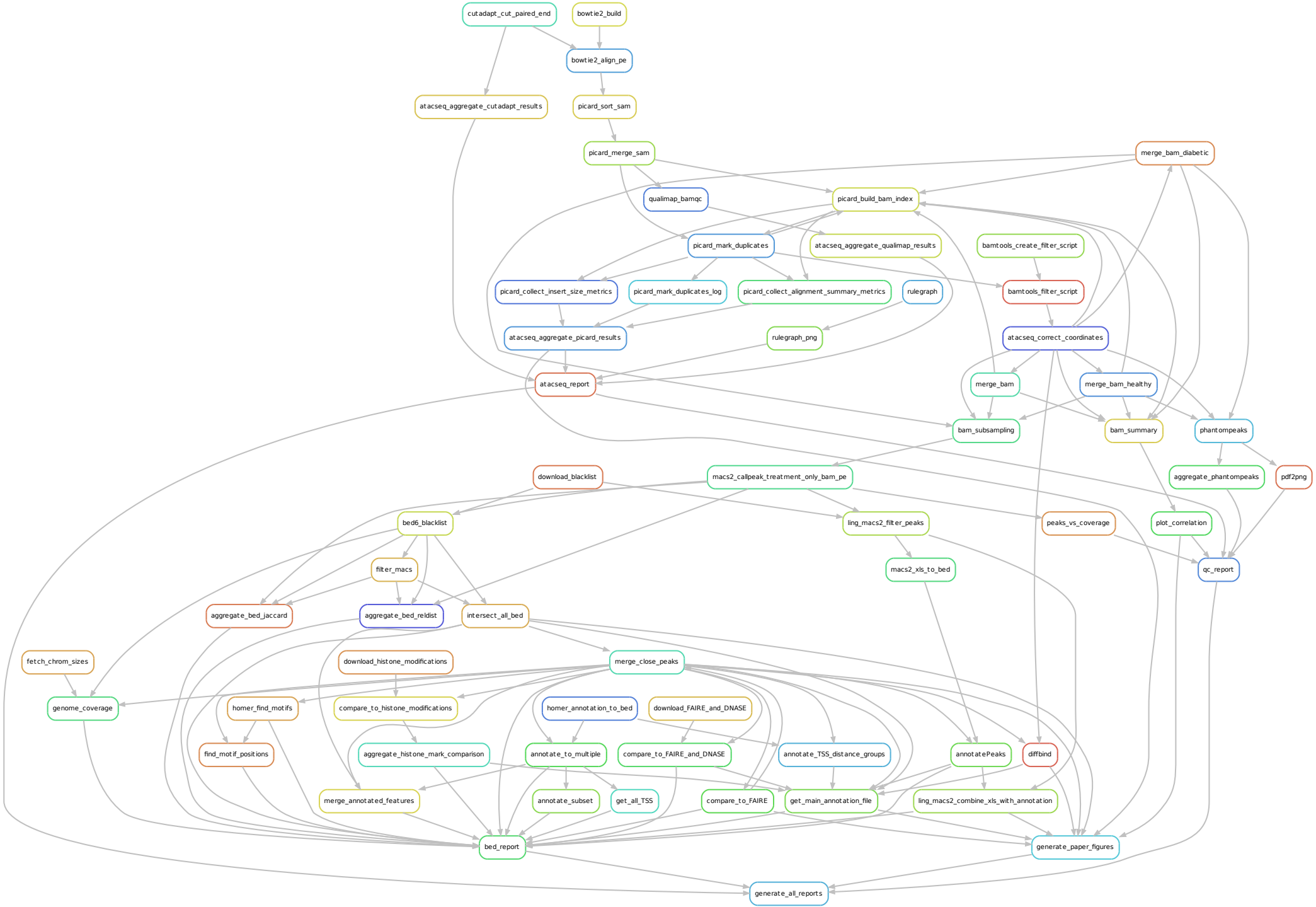
Anatomy of a Snakemake rule

```
import os

rule trim_fastq:
    input: "{prefix}.fastq"
    output: temp("{prefix}.trimmed.fastq")
    params:
        leftTrim=5,
        rightTrim=10
    log: "logs/trim_fastq.log"
    version: "0.1"
    message: "Trimming {input[0]}."
    shadow: True
    threads: 8
    priority: 90
    resources: mem=64
    conda: "envs/seqtk.yaml”
    singularity: "docker://quay.io/biocontainers/seqtk"
    run:
        if (os.stat(input[0]).st_size > 0):
            shell("seqtk trimfq -t {threads} –b {params.leftTrim}
                   -e {params.rightTrim} {input} > {output} 2> {log}")
        else:
            raise IOError(input[0]+" is empty.")
```

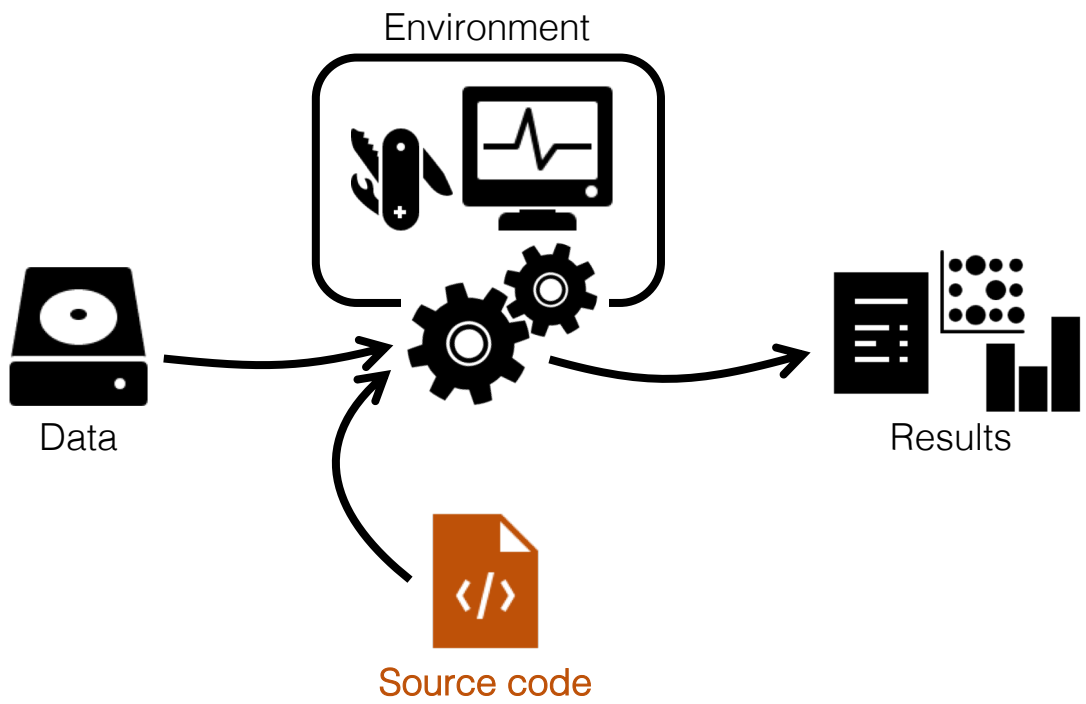
Command line interface

```
# execute the workflow with target a.trimmed.fastq.gz
snakemake a.trimmed.fastq.gz

# execute the workflow with the first rule as target
snakemake

# dry-run, print shell commands and reason for execution
snakemake -n –p -r

# visualize the DAG of jobs using the Graphviz dot command
snakemake --dag | dot -Tsvg > dag.svg

# execute the workflow with 8 cores
snakemake --cores 8

# run the workflow on a SLURM cluster
snakemake --cluster-config cluster.yml --cluster \
      "sbatch -A {cluster.account} -t {cluster.time}"
```

Snakemake



Things can get rather complex...




```
project
|- doc/
|
|- data/
|   |- raw_external/
|   |- raw_internal/
|   |- meta/
|
|- code/
|- notebooks/
|
|- intermediate/
|- scratch/
|- logs/
|
|- results/
|   |- figures/
|   |- tables/
|   |- reports/
|
|- Snakefile
|- config.yml
|- environment.yml
|- Dockerfile
```