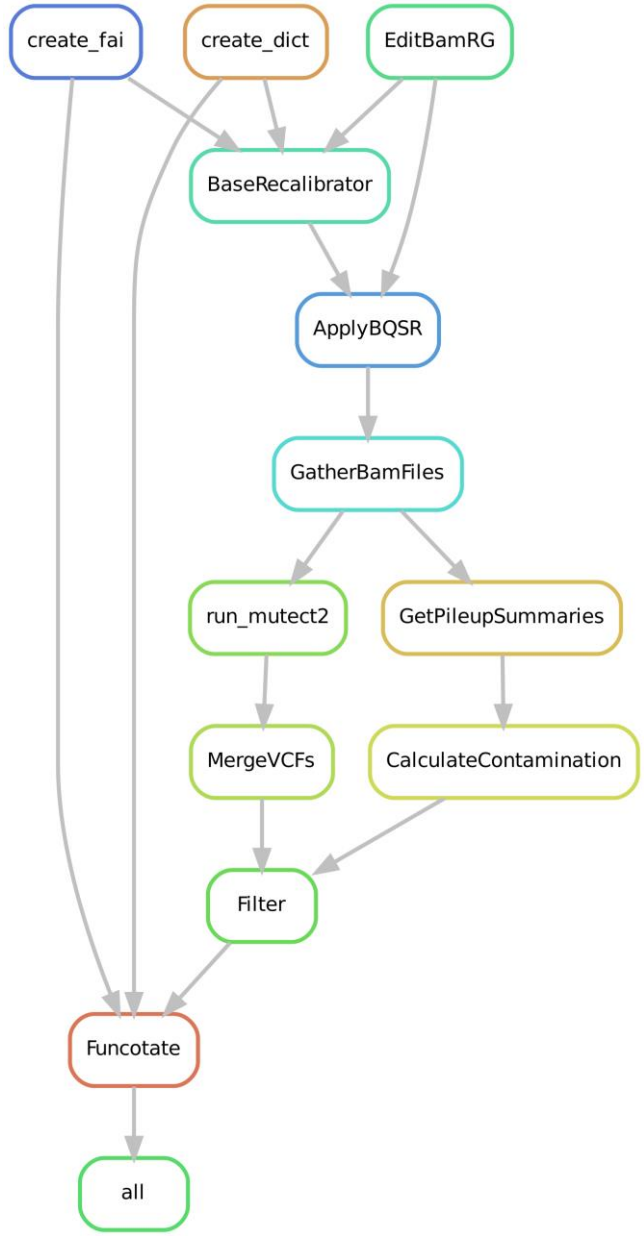
Workflow management with Snakemake

S. Rao

Workflow management

- Reproducibility
- Efficient use of resources (parallelisation)
- Readability
- Speed
- Ease of execution (automation)
- Minimising errors



Bioinformatics workflow software

- Snakemake
- Nextflow
- Cromwell
- Toil
- Bpipe
- Ruffus

BIOINFORMATICS APPLICATION NOTE

Vol. 28 no. 19 2012, pages 2520–2522
doi:10.1093/bioinformatics/bts480

Genome analysis

Advance Access publication August 20, 2012

Snakemake—a scalable bioinformatics workflow engine

Johannes Köster[1,2,*] and Sven Rahmann[1]

[1]Genome Informatics, Institute of Human Genetics, University of Duisburg-Essen and [2]Paediatric Oncology, University Childrens Hospital, 45147 Essen, Germany

Associate Editor: Alfonso Valencia

Outline

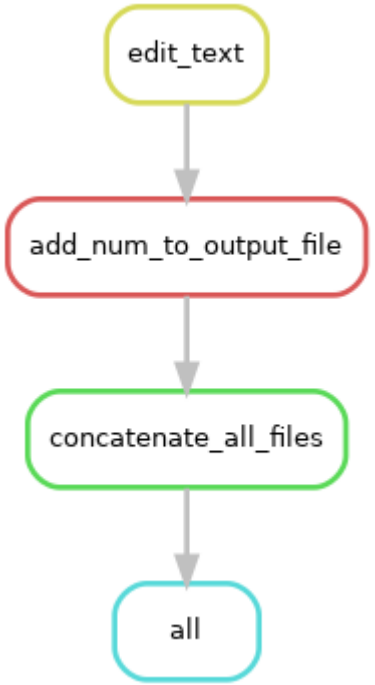
- Writing and running a basic rule
  demo
- Running on a cluster
- Constructing a workflow with many rules
- Wildcards
  demo + try-it-yourself
- Threads and resources
- Configuration file

```
Python/snakemake code
```

```
Terminal commands/arguments
```

A simple snakemake rule

```
rule rule_name:
    input: "data.txt"
    output: "results.txt"
    shell: "cat {input} > {output}"
```



4 ways to run a command within a rule

```
shell       # any shell command

run         # python code, can include shell()

script      # Python, R, Julia

notebook    # Jupyter notebooks
```

Run a workflow in your terminal

```
$ snakemake --dry-run # -n
```

```
$ snakemake --cores 1 # -j
```

```
$ snakemake --cores 1 --snakefile Snakefile
```

```
$ snakemake --cores 1 target_rule
```

Running on the cluster: with --cluster-config

Important: check the BMRC warning (link) before using it on Rescomp.

```
$ snakemake --max-status-checks-per-second 0.01
[…other args…]
```

```
$ snakemake --max-status-checks-per-second 0.01 \
        --cluster-config cluster.json \
        --cluster "qsub -N {cluster.job-name} -o {cluster.output} -q {cluster.time} -P
{cluster.project} -pe shmem {cluster.slots} -j y"
```

```
$ cat cluster.json
```

```
{
        "__default__":
                {
                "time": "short.qc",
                "slots": 1,
                "project": "project.prjc",
                "output": "path/to/output_folder/"
                "job-name": "default_job"
        }
}
```

Multiple rules

- By default, snakemake runs the first rule in a Snakefile, if a target rule is not specified at the command-line

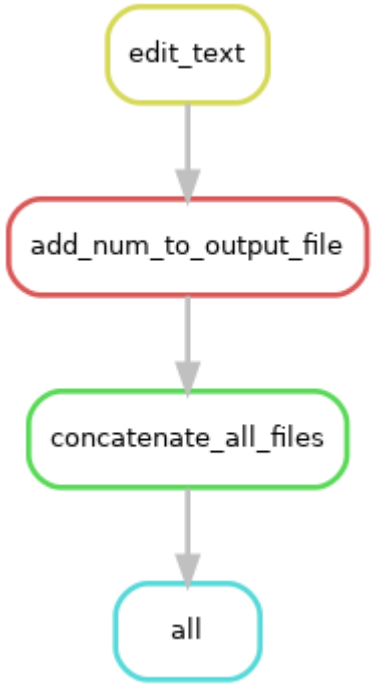
- By default, snakemake does not re-run rules whose output already exists. To re-run:
  - Update input files, e.g. `$ touch input_files/*.*`
  - Use `--force` argument (See below)
  - Use `--delete-all-output` argument (beware!)

- A specific rule can be specified at the command-line
  - `$ snakemake -j 1 --snakefile Snakefile third_rule`

How to build a multi-step workflow

- When constructing a workflow, it may help to think ‘backwards’
  - The `input` of the first rule (= target rule) is the final results you desire; target rule does not need `output`
  - Which rule’s `output` creates the `input` for the target rule? (let’s call this rule penultimate_rule)
  - Which rule creates the `input` for penultimate_rule?
  - and so on… until you write the rule that takes your existing input files (e.g. your raw fastq files) as `input`

```
rule all:
    input:
        r1 = “final_result1.txt”,
        r2 = “final_result2.txt”
```



expand()

- Say you have files
  - sample_A.fastq.gz
  - sample_B.fastq.gz
  - sample_C.fastq.gz
- Try the following in a python console or a Jupyter notebook

```
>>> import snakemake.io as sio
>>> filenames = ['A', 'B', 'C']
>>> fastq_files = sio.expand("somefile_{name}.txt", name = filenames)
>>> fastq_files
    ['somefile_A.txt', 'somefile_B.txt', 'somefile_C.txt']
```

Wildcards – snakemake uses regex

sample_{name}.fastq.gz	sample_{name}.trimmed.fastq.gz	sample_{name}.bam
sample_A.fastq.gz sample_B.fastq.gz sample_C.fastq.gz	Sample_A.trimmed.fastq.gz sample_B.trimmed.fastq.gz sample_C.trimmed.fastq.gz	sample_A.bam sample_B.bam sample_C.bam

- Can be accessed in input, output, params using the `{wildcard_name}` notation
- Can be accessed in run or shell directives using the `wildcards.wildcard_name` notation
- Notation looks similar in `expand()` but that is not wildcards
- Different notation to refer to wildcards within `expand()` – `{{wildcard_name}}`

Threads and resources

- Local execution
  - Total cores available for snakemake will not exceed `--cores / -j`
  - Threads for each rule allocated using `threads` keyword
  - If total threads == 10, and
    - A rule's `threads` keyword value is set to 5, 2 jobs based on that rule can run parellelly
    - A rule's `threads` keyword value is set to 6, 1 job based on that rule can run at a time using 6 threads
    - A rule's `threads` keyword value is set to 15, 1 job based on that rule can run at a time using 10 threads
- Cluster execution
  - Total jobs submitted to the cluster at a time will not exceed `--cores / -j`
  - slots for each rule allocated using the cluster json script, with a definition for each rule

- Arbitrary resources can be allocated at workflow level using --resources
  - At the rule level, this is set using the `resources` keyword

```
$ snakemake --resources mem=100
```

```
resources:
    mem = 50
```