Introduction to Snakemake

Einar Birkeland, Researcher, Department of Pharmacy, UiO

30/10-24

Overview

- Snakemake – what is it, and why use it?
- How it works
- Demonstration
- A couple of use cases
  - Pipeline for sequence data analysis
  - Handling of data analysis
- Additional features
- Nextflow and Snakemake

Intro – What is Snakemake

- Workflow management system designed for reproducibility
- Simplifies defining complex pipelines in bioinformatics

- Modular and scalable approach
- Integration of existing scripts and tools

- DSL is based on Python
  - Python code seamlessly integrated

Core concepts of Snakemake

- Snakemake workflows are file-based and defined by rules in a “Snakefile”
    - *Input*
    - *Output*
    - *Shell/script/run*

- Definition of software environments
    - Conda
    - Containers
    - Env-modules

- Parsing of dependencies to determine which jobs to do
    - Parse
    - Prioritize
    - Execute

Why use Snakemake

Reproducible analysis

Why not use Snakemake (or equivalent tools)?

It takes time and effort to set up = **overhead**

However, reproducible code is

- Starting to be required by journals
- Good scientific practice

Raw data

Processing steps

- Tools (versions)
- Environments
- Settings

Data analysis

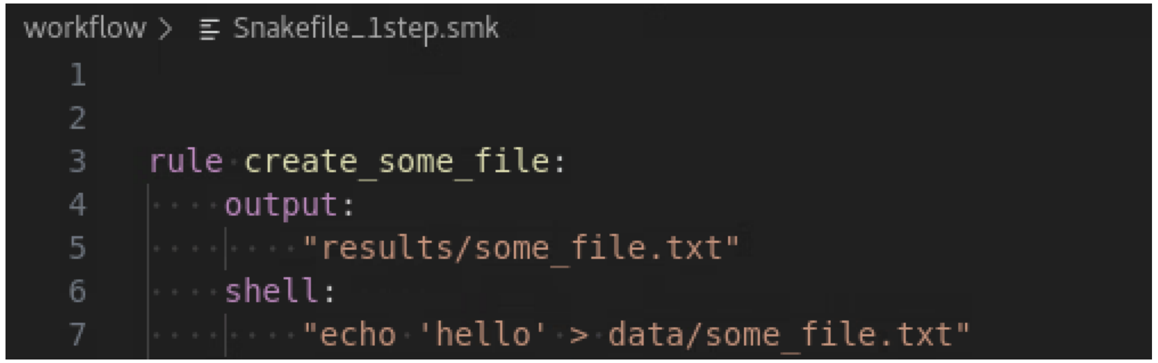
- Tools (versions)
- Environments
- Settings

Results

- Figures
- Tables
- Models, etc..

Demo: building a simple workflow – 1 step

1. Define the file you want to create: An output file – "some_file.txt"
2. Define which file(s) is necessary to create this file (optional)
3. Define what needs to be done to create the output file – in the "shell" directive

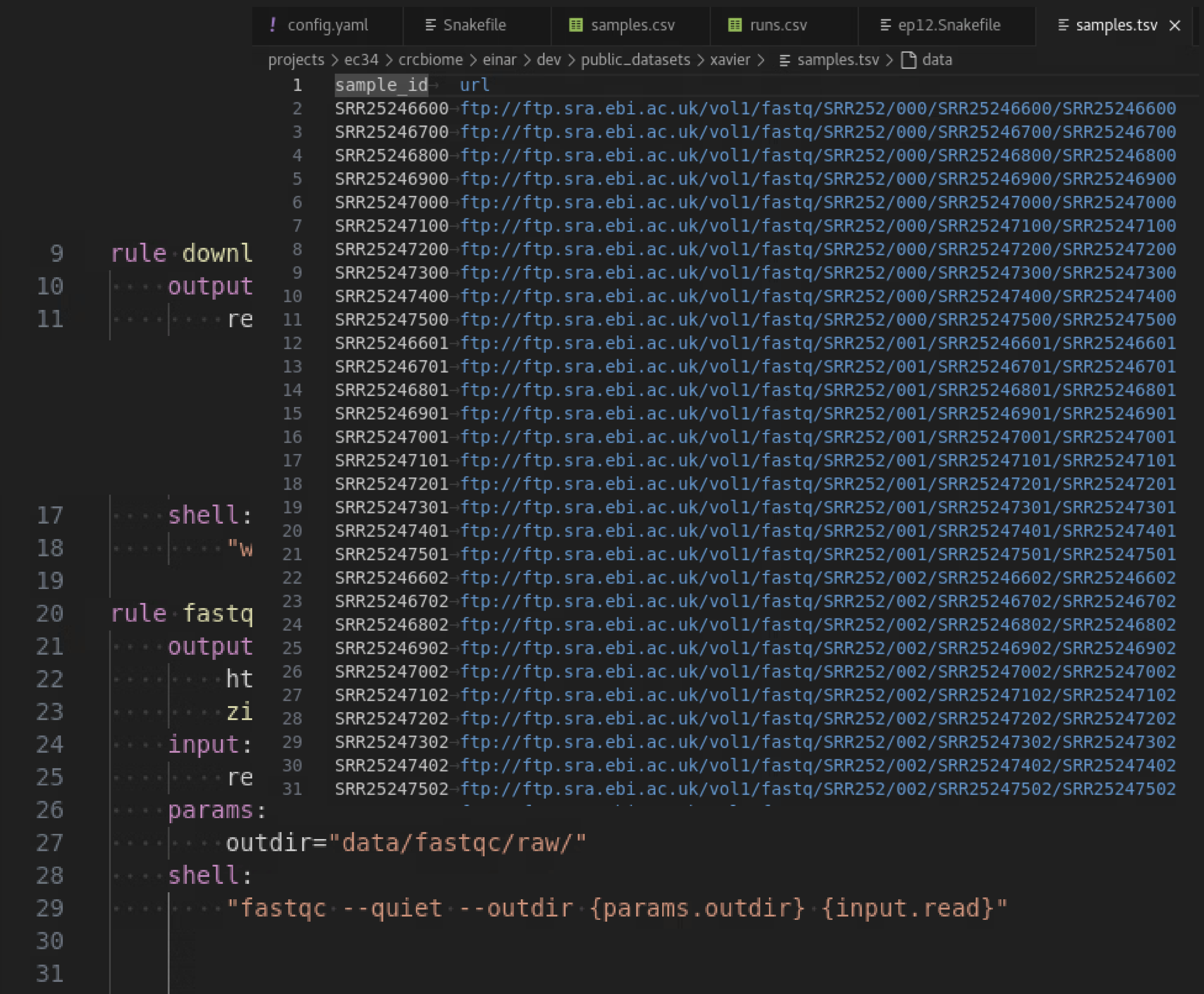


```


rule create_some_file:
    output:
        "results/some_file.txt"
    shell:
        "echo 'hello' > data/some_file.txt"
```

Demo: building a simple workflow – 2 steps

Case: We want to check out the quality of some publicly available sequencing data

1. Define the files we want to create
    1. Fastqc html files
2. Define what needs to be done to create these files
    1. Download the data
    2. Run fastqc



config.yaml | Snakefile | samples.csv | runs.csv | ep12.Snakefile | samples.tsv

projects > ec34 > crcbiome > einar > dev > public_datasets > xavier > samples.tsv > data

```
sample_id	url
SRR25246600	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/000/SRR25246600/SRR25246600
SRR25246700	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/000/SRR25246700/SRR25246700
SRR25246800	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/000/SRR25246800/SRR25246800
SRR25246900	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/000/SRR25246900/SRR25246900
SRR25247000	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/000/SRR25247000/SRR25247000
SRR25247100	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/000/SRR25247100/SRR25247100
SRR25247200	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/000/SRR25247200/SRR25247200
SRR25247300	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/000/SRR25247300/SRR25247300
SRR25247400	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/000/SRR25247400/SRR25247400
SRR25247500	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/000/SRR25247500/SRR25247500
SRR25246601	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/001/SRR25246601/SRR25246601
SRR25246701	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/001/SRR25246701/SRR25246701
SRR25246801	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/001/SRR25246801/SRR25246801
SRR25246901	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/001/SRR25246901/SRR25246901
SRR25247001	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/001/SRR25247001/SRR25247001
SRR25247101	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/001/SRR25247101/SRR25247101
SRR25247201	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/001/SRR25247201/SRR25247201
SRR25247301	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/001/SRR25247301/SRR25247301
SRR25247401	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/001/SRR25247401/SRR25247401
SRR25247501	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/001/SRR25247501/SRR25247501
SRR25246602	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/002/SRR25246602/SRR25246602
SRR25246702	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/002/SRR25246702/SRR25246702
SRR25246802	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/002/SRR25246802/SRR25246802
SRR25246902	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/002/SRR25246902/SRR25246902
SRR25247002	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/002/SRR25247002/SRR25247002
SRR25247102	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/002/SRR25247102/SRR25247102
SRR25247202	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/002/SRR25247202/SRR25247202
SRR25247302	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/002/SRR25247302/SRR25247302
SRR25247402	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/002/SRR25247402/SRR25247402
SRR25247502	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR252/002/SRR25247502/SRR25247502
```

```
rule downl
    output
        re

    shell:
        "w

rule fastq
    output
        ht
        zi
    input:
        re
    params:
        outdir="data/fastqc/raw/"
    shell:
        "fastqc --quiet --outdir {params.outdir} {input.read}"
```

```
rule download_reads:
    output:
        reads=temp("data/raw_reads/{sample}_{R}.fastq.gz")
```

```
    shell:
        "wget -nc {params.url} -P {params.out_dir}"

rule fastqc:
    output:
        html="data/fastqc/raw/{sample}_{R}_fastqc.html",
        zip="data/fastqc/raw/{sample}_{R}_fastqc.zip",
    input:
        read="data/raw_reads/{sample}_{R}.fastq.gz"
    params:
        outdir="data/fastqc/raw/"
    shell:
        "fastqc --quiet --outdir {params.outdir} {input.read}"


```

Demo: building a simple workflow – 2 steps

Case: We want to check out the quality of some publicly available sequencing data

1. Define the files we want to create
    1. Fastqc html files
2. Define what needs to be done to create these files
    1. Download the data
    2. Run fastqc

Additional considerations

Python integration

1. Use pandas to read a file with information about samples
2. Used a lambda function to retrieve information from the pandas dataframe
3. Imported fastqc as a module

Demo: building a simple workflow – 4 steps

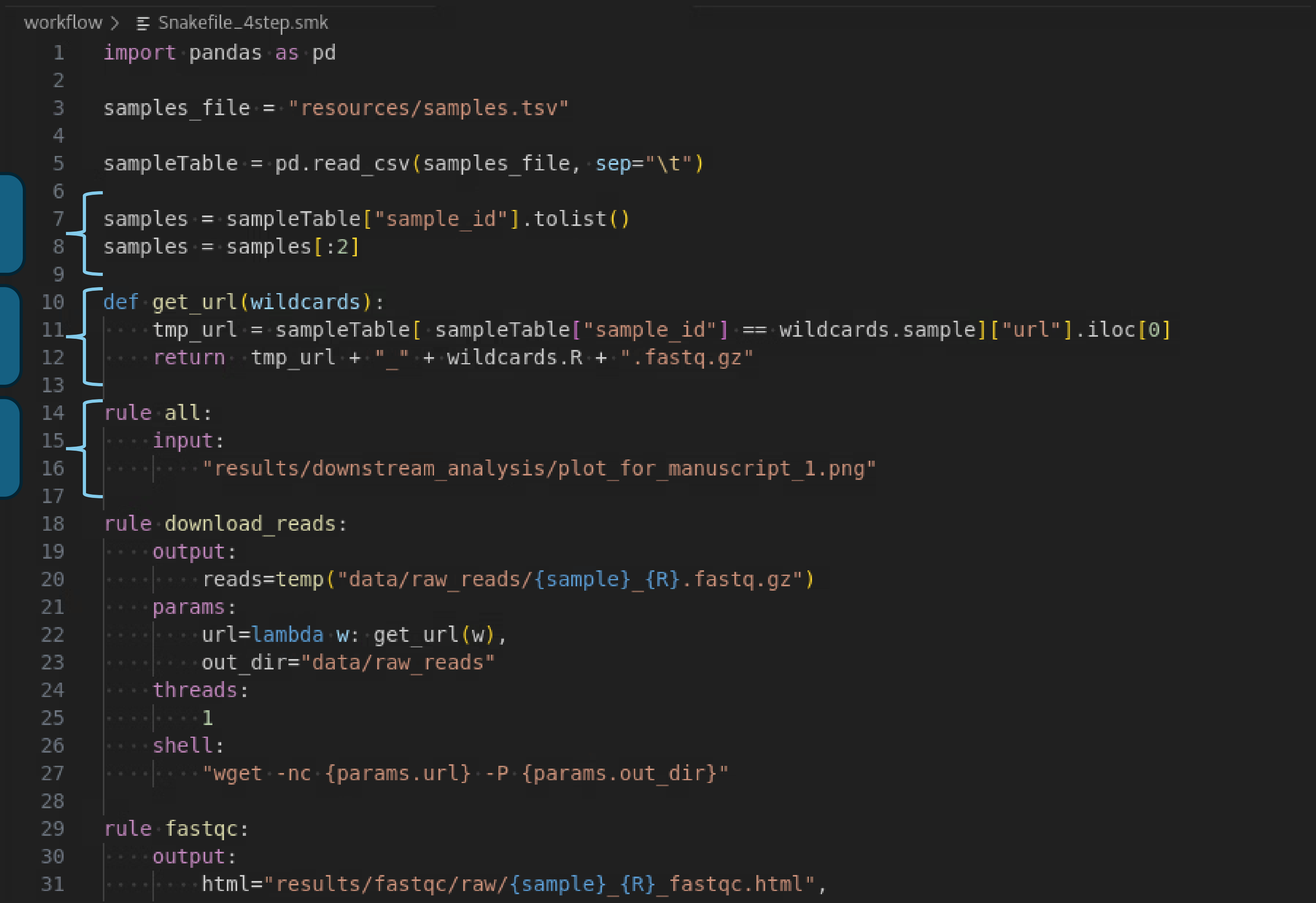
Case: We want to check out the quality of some publicly available sequencing data, and run an R script on the multiqc output

1. Define the files we want to create
    1. **Multiqc** html file
    2. A plot based on multiQC output
2. Define what needs to be done to create these files
    1. Download the data
    2. Run fastqc
    3. Run multiqc on all fastqc output
    4. Call an R script that produces the output we want

Define the samples we want to use

Define a function to get url

“rule all” specifies the target file

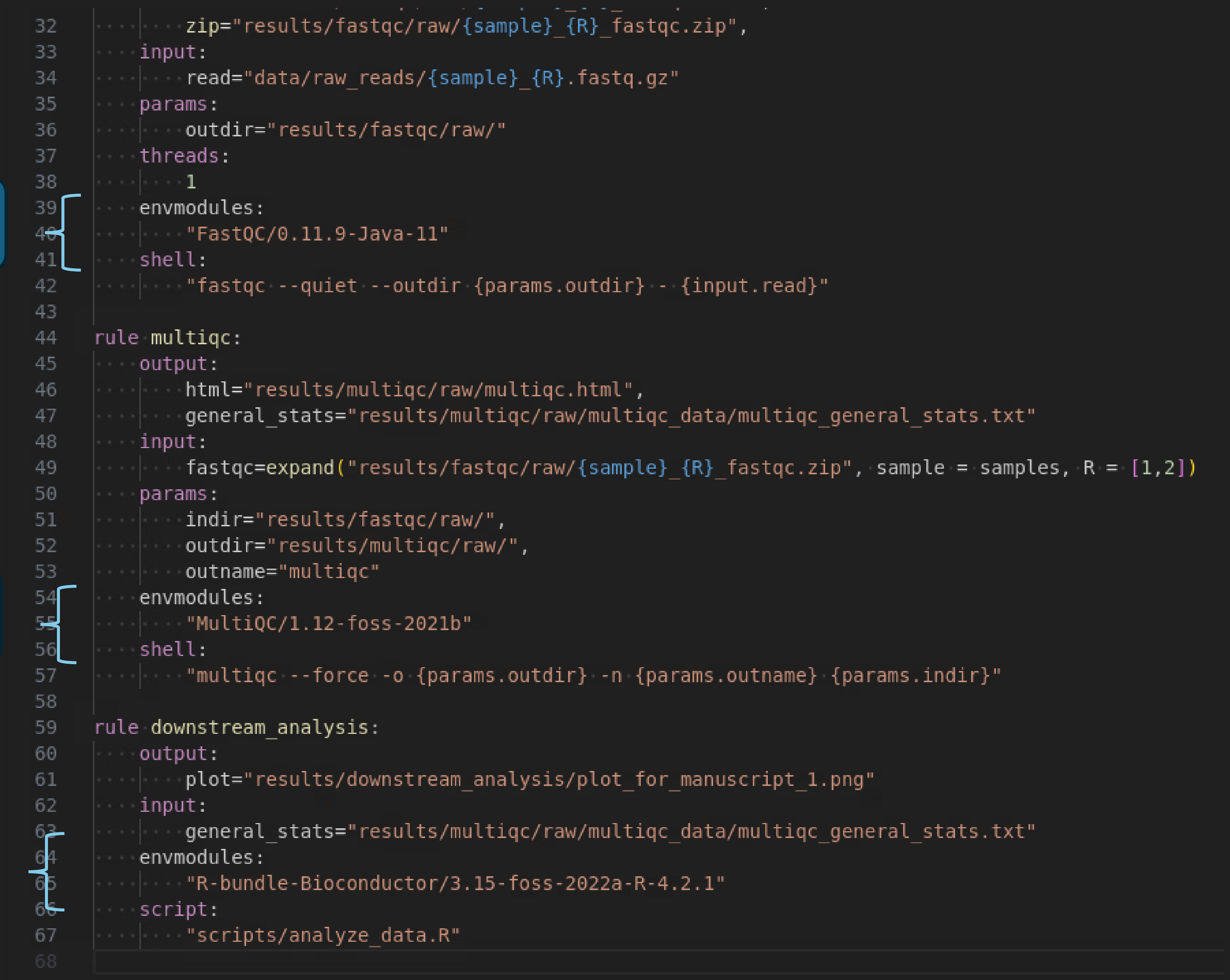


```
import pandas as pd

samples_file = "resources/samples.tsv"

sampleTable = pd.read_csv(samples_file, sep="\t")

samples = sampleTable["sample_id"].tolist()
samples = samples[:2]

def get_url(wildcards):
    tmp_url = sampleTable[ sampleTable["sample_id"] == wildcards.sample]["url"].iloc[0]
    return  tmp_url + "_" + wildcards.R + ".fastq.gz"

rule all:
    input:
        "results/downstream_analysis/plot_for_manuscript_1.png"

rule download_reads:
    output:
        reads=temp("data/raw_reads/{sample}_{R}.fastq.gz")
    params:
        url=lambda w: get_url(w),
        out_dir="data/raw_reads"
    threads:
        1
    shell:
        "wget -nc {params.url} -P {params.out_dir}"

rule fastqc:
    output:
        html="results/fastqc/raw/{sample}_{R}_fastqc.html",
```

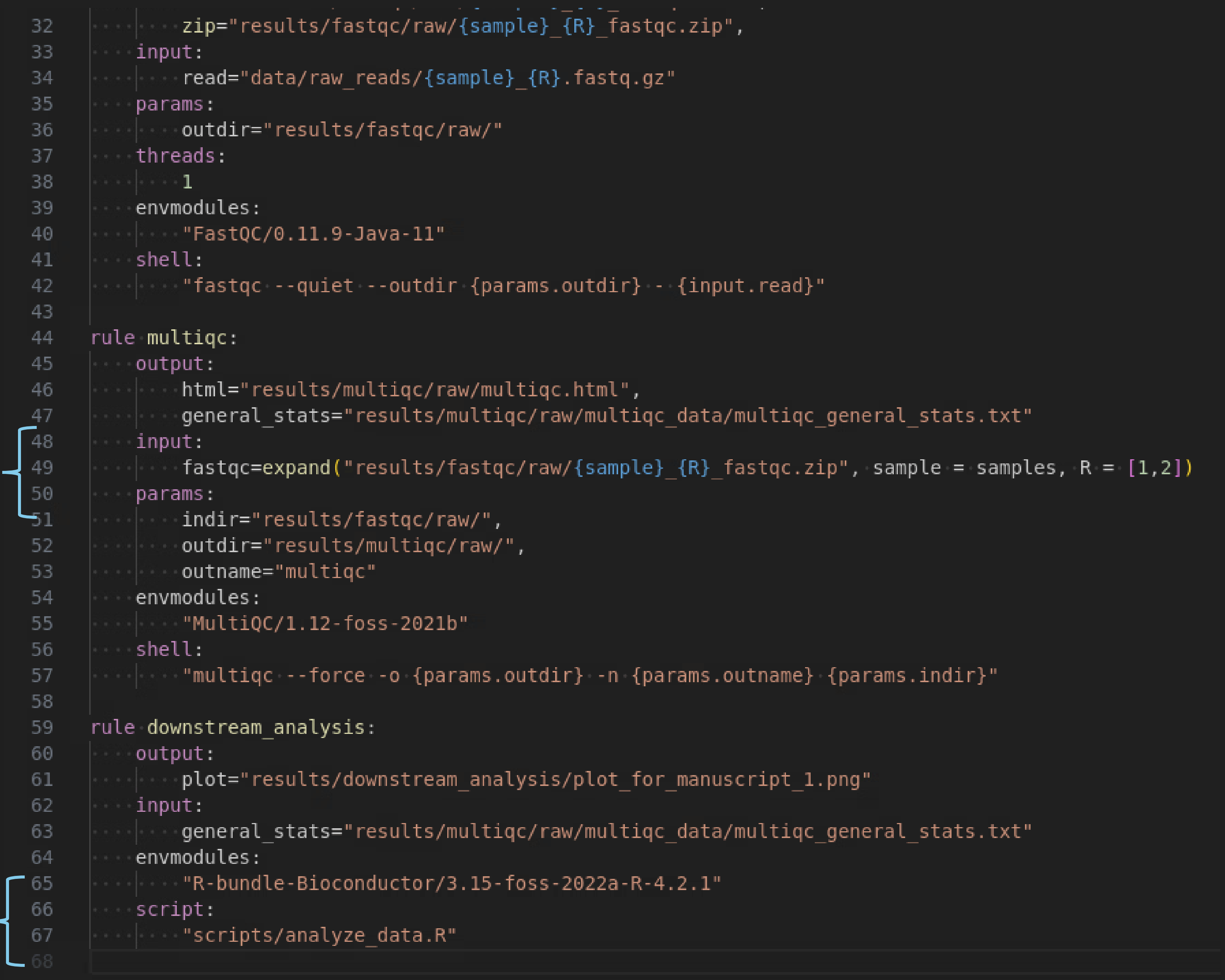
Same envmodule as before for FastQC

Envmodule specific for MultiQC

Envmodule specific for R + bioconductor



```
        zip="results/fastqc/raw/{sample}_{R}_fastqc.zip",
    input:
        read="data/raw_reads/{sample}_{R}.fastq.gz"
    params:
        outdir="results/fastqc/raw/"
    threads:
        1
    envmodules:
        "FastQC/0.11.9-Java-11"
    shell:
        "fastqc --quiet --outdir {params.outdir} - {input.read}"

rule multiqc:
    output:
        html="results/multiqc/raw/multiqc.html",
        general_stats="results/multiqc/raw/multiqc_data/multiqc_general_stats.txt"
    input:
        fastqc=expand("results/fastqc/raw/{sample}_{R}_fastqc.zip", sample = samples, R = [1,2])
    params:
        indir="results/fastqc/raw/",
        outdir="results/multiqc/raw/",
        outname="multiqc"
    envmodules:
        "MultiQC/1.12-foss-2021b"
    shell:
        "multiqc --force -o {params.outdir} -n {params.outname} {params.indir}"

rule downstream_analysis:
    output:
        plot="results/downstream_analysis/plot_for_manuscript_1.png"
    input:
        general_stats="results/multiqc/raw/multiqc_data/multiqc_general_stats.txt"
    envmodules:
        "R-bundle-Bioconductor/3.15-foss-2022a-R-4.2.1"
    script:
        "scripts/analyze_data.R"
```



```
        zip="results/fastqc/raw/{sample}_{R}_fastqc.zip",
    input:
        read="data/raw_reads/{sample}_{R}.fastq.gz"
    params:
        outdir="results/fastqc/raw/"
    threads:
        1
    envmodules:
        "FastQC/0.11.9-Java-11"
    shell:
        "fastqc --quiet --outdir {params.outdir} - {input.read}"

rule multiqc:
    output:
        html="results/multiqc/raw/multiqc.html",
        general_stats="results/multiqc/raw/multiqc_data/multiqc_general_stats.txt"
    input:
        fastqc=expand("results/fastqc/raw/{sample}_{R}_fastqc.zip", sample = samples, R = [1,2])
    params:
        indir="results/fastqc/raw/",
        outdir="results/multiqc/raw/",
        outname="multiqc"
    envmodules:
        "MultiQC/1.12-foss-2021b"
    shell:
        "multiqc --force -o {params.outdir} -n {params.outname} {params.indir}"

rule downstream_analysis:
    output:
        plot="results/downstream_analysis/plot_for_manuscript_1.png"
    input:
        general_stats="results/multiqc/raw/multiqc_data/multiqc_general_stats.txt"
    envmodules:
        "R-bundle-Bioconductor/3.15-foss-2022a-R-4.2.1"
    script:
        "scripts/analyze_data.R"
```

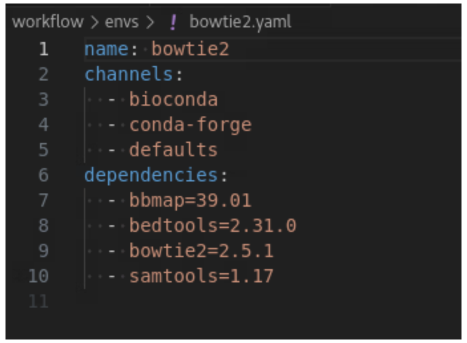
“expand” combines variables into a list

“script” directive allows integration btw snakemake and R/python/Julia... etc

Additional features of Snakemake

Environments:

- Envmodules
- Conda
  - YAML specification
- Containers



workflow > envs > ! bowtie2.yaml

```yaml
name: bowtie2
channels:
  - bioconda
  - conda-forge
  - defaults
dependencies:
  - bbmap=39.01
  - bedtools=2.31.0
  - bowtie2=2.5.1
  - samtools=1.17
```

Scalability:

- Parallelization
- Cluster compatability
  - Slurm, others
- Cloud computing
  - AWS, google cloud, etc

- Modularity
- Logging
- Benchmarking
- DAG generation: Visualization of dependencies

Collaboration/community

- Snakemake wrappers
  - Best practices for common tasks

- Snakemake workflows
  - Collection of all snakemake workflows on github

Find out more

Snakemake homepage

- Paper
- Tutorials
  - Slides
  - Online tutorial
- Wrappers
- Workflows

Nextflow or Snakemake?

Pipeline complexity	Snakemake Pros/Cons	Nextflow Pros/Cons
Simple	Pros: Easy syntax, quick setup, File-based management	Pros: Containerization, Parallelization
	Pros: Limited flexibility, Container handling	Cons: Groovy syntax complexity, setup overhead
Intermediate	Pros: Modularity, DAG visualization, cluster compatability	Pros: Dynamic workflows, resource management, Modularity
	Cons: Limited branching, Manual resource setting	Cons: Steeper learning curve, Debugging complexity
Complex	Pros: Python integration DAG visualization, cloud support	Pros: Adaptive workflows, strong container/cloud support
	Cons: Rigid execution, limited dynamic handling	Cons: Resource-intensive, Groovy complexity

Comparison to Nextflow

- **Syntax/Language**
  - Snakemake: Python-like syntax, more accessible for beginners
  - Nextflow: Groovy-based DSL may have a steeper learning curve
- **Best Use Cases**
  - Snakemake: Local/small-scale workflows, quick customizations
  - Nextflow: Large-scale, distributed workflows
- **Reproducibility with Containers**
  - Both tools support Docker and Singularity for reproducibility
- **Scalability**
  - Top-down rule dependency resolving in Snakemake vs. bottom-up in Nextflow
    - Snakemake may be slower for large and complex workflows
  - Snakemake less easily portable to large-scale computational resources