

# Snakemake

Phil Ashton, PGI Advanced Training - KEMRI Wellcome, April 2022

Why are we going to talk to you  
about workflow managers?

# Advantages of workflow managers

- Each job/task can run within its own environment/container
- Failures handled elegantly - and re-starting.
- Portable - can send it to collaborator and it should work (?)
- Scalable - same code can run on your laptop or an HPC with minor changes
- Efficient - parallelisable jobs run in parallel.

# Workflow managers

- **Snakemake**
- Nextflow
- Cromwell
- Galaxy
- Ruffus
- BPIPE
- ...

Choosing which one to use is a trade-off between learning curve, feature richness, and ease of use.

# One way to think about Snakemake scripts

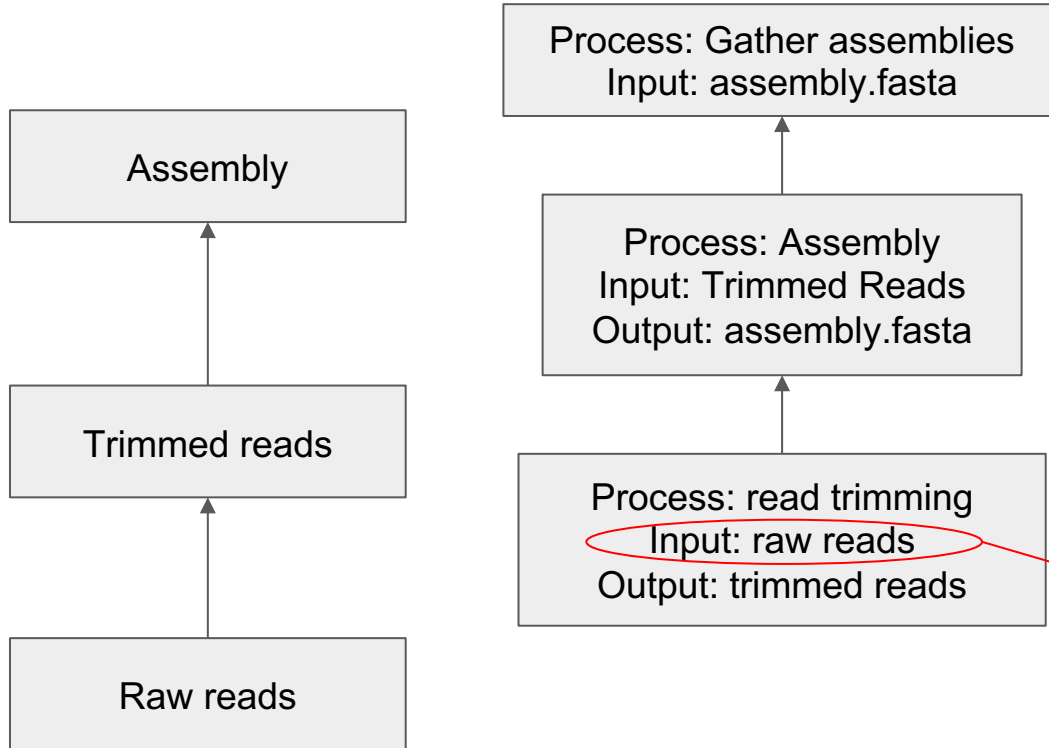
1. What is the final thing you want out of this pipeline?
2. What is the process that will give you that thing?
3. What are the inputs for that process?
4. Where do those inputs come from?

Input files



```
graph LR; IF[Input files] --> Q4[4. Where do those inputs come from?]; Q4 --> Q3[3. What are the inputs for that process?]; Q4 --> Q2[2. What is the process that will give you that thing?]; Q4 --> Q1[1. What is the final thing you want out of this pipeline?];
```

# One way to think about Snakemake scripts



=



Given as files

# Snakemake



- Based on Python & Make
- Split your workflow into separate “rules”
  - One rule per process normally
- Rules have:
  - A rule name
  - Input files
  - Output files
  - A command to turn the input into the output

# An example Snakemake script

```
rule bwa_map:
```

```
    input: "data/genome.fa", "data/samples/A.fastq"
```

```
    output: "mapped_reads/A.bam"
```

```
    shell: "bwa mem {input} | samtools view -Sb - > {output}"
```



# Running Snakemake scripts

If the contents of the previous slide is saved as “my\_mapping\_pipeline.smk” then to run it we would do:

```
`snakemake -s my_mapping_pipeline.smk`
```

Alternatively you can save the contents in a file called `Snakefile` and then just run `snakemake` in the directory containing `Snakefile` and it will run the script.

# A generalised Snakemake script

```
rule bwa_map:
```

```
    input: "data/genome.fa", "data/samples/{sample}.fastq"
```

```
    output: "mapped_reads/{sample}.bam"
```

```
    shell: "bwa mem {input} | samtools view -Sb - > {output}"
```


- Snakemake uses “named wildcards”, in this case `{sample}`.
- In this case snakemake would find all the files matching the pattern `data/samples/{sample}.fastq` and run the rule `bwa\_map` on them
- The `{sample}` wildcard will “propagate” through to the output, so the output bam name will match the input fastq name.

# A more useful generalised Snakemake script

```
samples = ['sample1', 'sample2']
```

```
rule bwa_map:
```

```
    input: ref_genome = "data/genome.fa", fastqs =  
    expand("data/samples/{sample}.fastq", sample = samples)  
    output: "mapped_reads/{sample}.bam"  
    shell: "bwa mem {input} | samtools view -Sb - > {output}"
```



The `expand` function lets you  
give a “todo list” to a rule.

# A two-step snakemake workflow

rule bbduk:

```
input: r1 = '{root_dir}/{sample}_1.fastq.gz', r2 = '{root_dir}/{sample}_2.fastq.gz'
output: r1 = '{root_dir}/{sample}_bbduk_1.fq.gz', r2 = '{root_dir}/{sample}_bbduk_2.fq.gz'
shell: 'bbduk.sh in={input.r1} in2={input.r2} out={output.r1} out2={output.r2}'
```

rule shovill:

```
input: r1 = rules.bbduk.output.r1, r2 = rules.bbduk.output.r2
output: final = '{root_dir}/{sample}/shovill_bbduk/contigs.fa',
shell: 'shovill --outdir {root_dir}/{wildcards.sample}/shovill -R1 {input.r1} -R2 {input.r2}'
```

# A three-step snakemake workflow



```
rule all:
```

```
    input: expand('{root_dir}/{sample}/shovill_bbduk/contigs.fa', sample = todo_list, root_dir =  
root_dir)
```

```
rule bbduk:
```

```
    input: r1 = '{root_dir}/{sample}_1.fastq.gz', r2 = '{root_dir}/{sample}_2.fastq.gz'
```

```
    output: r1 = '{root_dir}/{sample}_bbduk_1.fq.gz', r2 = '{root_dir}/{sample}_bbduk_2.fq.gz'
```

```
    shell: 'bbduk.sh in={input.r1} in2={input.r2} out={output.r1} out2={output.r2}'
```

```
rule shovill:
```

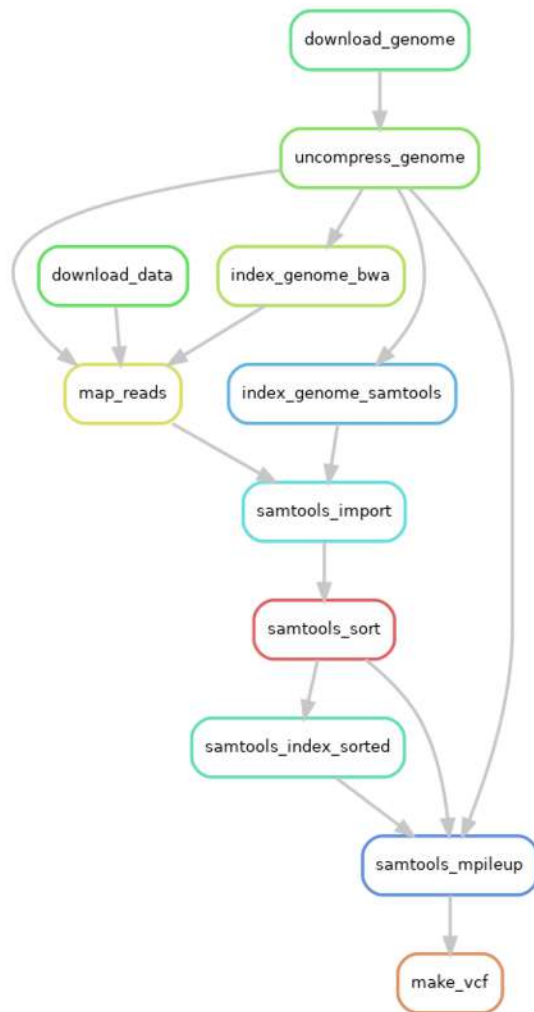
```
    input: r1 = rules.bbduk.output.r1, r2 = rules.bbduk.output.r2
```

```
    output: final = '{root_dir}/{sample}/shovill_bbduk/contigs.fa',
```

```
    shell: 'shovill --outdir {root_dir}/{wildcards.sample}/shovill -R1 {input.r1} -R2 {input.r2}'
```

# Directed acyclic graphs (DAGs)

When snakemake runs, it converts the snakefile into a directed acyclic graph.

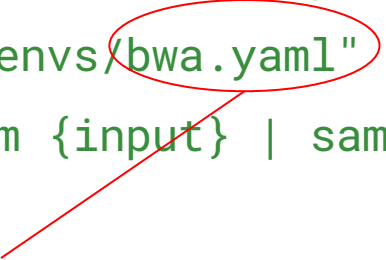


# Practical exercise 1 - basic and advanced snakemake

1. [Basics: An example workflow — Snakemake 7.3.2 documentation](#)
2. [Advanced: Decorating the example workflow — Snakemake 7.3.3 documentation](#)

# Using conda with snakemake

```
rule bwa_map:
    input: ref_genome = "data/genome.fa", fastqs =
expand("data/samples/{sample}.fastq", sample = samples)
    output: "mapped_reads/{sample}.bam"
    conda: "../..envs/bwa.yaml"
    shell: "bwa mem {input} | samtools view -Sb - > {output}"
```



A yaml file defining the conda packages required to run this rule.



# Practical exercise 2 - conda integration

- Modify your script from the basic conda exercise to use conda environments for each rule
  - a. [Distribution and Reproducibility — Snakemake 7.3.5 documentation](#)

# Using snakemake with HPC

- Submitting/managing jobs on an HPC can be a hassle
- Especially when jobs depend on each other
- Snakemake makes it much easier to use an HPC
- If you don't have an HPC, snakemake can also be configured to use e.g. Amazon Web Services.

# Using snakemake with HPC

```
snakemake -s ~/scripts/snakemake/salmonella_workflow.smk \
--cluster-config ~/.config/snakemake/salmonella_slurm/config.yaml \
--cluster "sbatch --cpus-per-task={cluster.cpus-per-task} \
--mem-per-cpu={cluster.mem-per-cpu}" \
--jobs 1000
```

```
$ cat ~/.config/snakemake/salmonella/config.yaml
```

```
__default__:
  cluster: sbatch
  cpus-per-task: 1
  mem-per-cpu: 4000
  jobs: 100
```

```
bbduk:
  cpus-per-task: 8
```

```
shovill:
  cpus-per-task: 8
  mem-per-cpu: 6000
```

# Practical exercise 3 - using snakemake on HPC

- Modify your snakemake script from the basic workflow to execute on the HPC
  - More information - [Cluster Execution — Snakemake 7.3.2 documentation](#)
  - You will need to write a config file like on the previous slide, for this example you can just set a sensible `__default__` for all rules.
  - Then execute using the snakemake command from previous slide as a template.

# Acknowledgements & further reading

Anna Price, Cardiff/CLIMB - [https://www.youtube.com/watch?v=qORviM\\_ELdk](https://www.youtube.com/watch?v=qORviM_ELdk)

[Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers | Nature Methods](#)

[Snakemake - Reproducible and Scalable Bioinformatic Workflows](#)

<https://snakemake.readthedocs.io/en/stable/tutorial/basics.html>