Anatomy of a Snakefile

Basic structure of a rule

```
rule:
    output: "results/sample1.stats.txt"
    shell:
        """
        echo -e "sample1\t50%" > {output}
        """
```

```
$ snakemake -c 1 results/sample1.stats.txt
Building DAG of jobs...
Using shell: /bin/bash
Provided cores: 1 (use --cores to define parallelism)
Rules claiming more threads will be scaled down.
Job counts:
        count   jobs
        1       1
        1
Select jobs to execute...

[Tue Sep 28 11:50:45 2021]
rule 1:
    output: results/sample1.stats.txt
    jobid: 0

[Tue Sep 28 11:50:45 2021]
Finished job 0.
1 of 1 steps (100%) done

$ cat results/sample1.stats.txt
sample1 50%
```

Basic structure of a rule

More commonly, rules are named and have both input and output:

```
rule generate_stats:
    output: "results/sample1.stats.txt"
    input: "results/sample1.bam"
    shell:
        """
        samtools flagstat {input} > {output}
        """
```

Wildcards

Wildcards generalize a workflow. Imagine you have not just sample1 but samples 1..100.

Instead of writing 100 rules...

```
rule generate_stats_sample1:
    output: "results/sample1.stats.txt"
    input: "results/sample1.bam"
...
rule generate_stats_sample100:
    output: "results/sample100.stats.txt"
    input: "results/sample100.bam"
```

...we can introduce one or more wildcards which Snakemake can match to several text strings using regular expressions.

In our example, we replace the actual sample ids with the wildcard `sample`:

```
rule generate_stats:
    output: "results/{sample}.stats.txt"
    input: "results/{sample}.bam"
    shell:
        """
        samtools flagstat {input} > {output}
        """
```

Wildcards

Rules can have multiple wildcards...

```
rule generate_stats:
    output: "results/{sample}_{lane}.stats.txt"
    input: "results/{sample}_{lane}.bam"
    shell:
      """
      samtools flagstats {input} > {output}
      """
```

...but wildcards must be present in the output section.

Will work:

```
rule generate_stats:
    output: "results/{sample}_{lane}.stats.txt"
    input: "results/{sample}.bam"
```

Won't work:

```
rule generate_stats:
    output: "results/{sample}.stats.txt"
    input: "results/{sample}_{lane}.bam"
```

```
Wildcards in input files cannot be determined from output files: 'lane'
```

Rule ambiguities

Ambiguities can arise when two rules produce the same output:

```
rule generate_stats:
    output: "results/{sample}.stats.txt"
    input: "results/{sample}.bam"
    shell:
        """
        samtools flagstat {input} > {output}
        """

rule print_stats:
    output: "results/{sample}.stats.txt"
    input: "results/{sample}.log"
    shell:
        """
        grep "% alignment" {input} > {output}
        """

rule make_report:
    output: "results/{sample}.report.pdf"
    input: "results/{sample}.stats.txt"
```

```
$ snakemake -c 1 -n results/sample1.report.pdf
Building DAG of jobs...
AmbiguousRuleException:
Rules generate_stats and print_stats are ambiguous for the file results/sample1.stats.txt.
```

Rule ambiguities

Ambiguities can arise when two rules produce the same output:

This can be handled in a number of ways:

- by changing the output file name of one of the rules
- or via the `ruleorder` directive:

```
ruleorder: generate_stats > print_stats
```

- or by specifically referring to the output of a certain rule:

```
rule make_report:
output: "results/{sample}.report.pdf"
input: rules.generate_stats.output
```

Logging

Logfiles and messages add descriptions and help with debugging:

```
rule generate_stats:
    output: "results/{sample}.stats.txt"
    input: "results/{sample}.bam"
    log: "results/{sample}.flagstat.log"
    message: "Generating stats for sample {wildcards.sample}"
    shell:
        """
        samtools flagstat {input} > {output} 2>{log}
        """
```

Log files are not deleted by snakemake if there's an error.

Resources

Compute resources can be set with threads and resources:

```
rule generate_stats:
    output: "results/{sample}.bam"
    input: "results/{sample}.stats.txt"
    log: "results/{sample}.flagstat.log"
    message: "Generating stats for sample {wildcards.sample}"
    threads: 4
    resources:
        mem_mb=100
    shell:
        """
        samtools flagstat --threads {threads} {input} > {output} 2>{log}
        """
```

Resources

Compute resources can be set with threads and resources:

```
rule generate_stats:
    output: "results/{sample}.bam"
    input: "results/{sample}.stats.txt"
    log: "results/{sample}.flagstat.log"
    message: "Generating stats for sample {wildcards.sample}"
    threads: workflow.cores * 0.5 # <---- threads as a function of workflow cores
    resources:
        mem_mb=100
    shell:
        """
        samtools flagstat --threads {threads} {input} > {output} 2>{log}
        """
```

It's also possible to set threads based on the cores given to snakemake (e.g. `--cores 8` or `-c 8`).

More on resources in a lecture tomorrow.

Parameters

Rule parameters can be set with the params directive:

```
rule generate_stats:
    output: "results/{sample}.bam"
    input: "results/{sample}.stats.txt"
    log: "results/{sample}.flagstat.log"
    message: "Generating stats for sample {wildcards.sample}"
    threads: workflow.cores * 0.5
    resources:
        mem_mb=100
    params:
        verbosity = 2
    shell:
        """
        samtools flagstat --verbosity {params.verbosity} \
          --threads {threads} {input} > {output} 2>{log}
        """
```

Software environments

Software environments can be set for each rule using `conda:`:

```
rule generate_stats:
    output: "results/{sample}.bam"
    input: "results/{sample}.stats.txt"
    log: "results/{sample}.flagstat.log"
    message: "Generating stats for sample {wildcards.sample}"
    threads: workflow.cores * 0.5
    resources:
        mem_mb=100
    params:
        verbosity = 2
    conda: "envs/samtools.yml"
    shell:
        """
        samtools flagstat --verbosity {params.verbosity} \
          --threads {threads} {input} > {output} 2>{log}
        """
```

```
### Contents of envs/samtools.yml ###
name: samtools
channels:
  - bioconda
dependencies:
  - samtools=1.15.1
```

To make Snakemake use the conda environment, specify `--use-conda` on the command line.

Software environments

Or by using `envmodules`, e.g. in compute clusters:

```
rule generate_stats:
    output: "results/{sample}.bam"
    input: "results/{sample}.stats.txt"
    log: "results/{sample}.flagstat.log"
    message: "Generating stats for sample {wildcards.sample}"
    threads: workflow.cores * 0.5
    resources:
        mem_mb=100
    params:
        verbosity = 2
    conda: "envs/samtools.yml"
    envmodules:
        "bioinfo-tools",
        "samtools"
    shell:
        """
        samtools flagstat --verbosity {params.verbosity} \
          --threads {threads} {input} > {output} 2>{log}
        """
```

To make Snakemake use envmodules, specify `--use-envmodules` on the command line.

More on conda and envmodules tomorrow.

Config files

Config files allow you to configure workflows without having to change the underlying code.

Config files should be in `YAML` or `JSON` format:

```
### Contents of config.yml ###
samples: ["sample1", "sample2", "sample3"]
verbosity: 2
```

```
### Contents of config.json ###
{
  "samples": [
    "sample1",
    "sample2",
    "sample3"
  ],
  "verbosity": 2
}
```

Specify one or more config files on the command line with:

```
snakemake --configfile config.yml -j 1
```

Or directly in a snakefile, e.g.:

```
configfile: "config.yml""
```

Config files

The config parameters are available as a dictionary inside your snakefiles:

```
### Contents of config.yml ###
samples: ["sample1", "sample2", "sample3"]
verbosity: 2
```

```
configfile: "config.yml"
print(config)
{'samples': ['sample1', 'sample2', 'sample3'], 'verbosity': 2}
```

This allows you to manipulate the config variable inside your snakemake files and python code.

```
config["samples"].append("sample4")
print(config)
{'samples': ['sample1', 'sample2', 'sample3', 'sample4'], 'verbosity': 2}
```

Config files

In our example rule, verbosity level can be controlled via a config file like this:

```
configfile: "config.yml"

rule all:
    input:
      expand("results/{sample}.stats.txt", sample = config["samples"])

rule generate_stats:
    output: "results/{sample}.bam"
    input: "results/{sample}.stats.txt"
    log: "results/{sample}.flagstat.log"
    message: "Generating stats for sample {wildcards.sample}"
    threads: workflow.cores * 0.5
    resources:
        mem_mb=100
    params:
        verbosity = config["verbosity"] # <-----
    conda: "envs/samtools.yml"
    envmodules:
        "bioinfo-tools",
        "samtools"
    shell:
        """
        samtools flagstat --verbosity {params.verbosity} \
          --threads {threads} {input} > {output} 2>{log}
        """
```

Config files

Config files are also convenient for defining what the workflow will do.

If no targets are given on the command line, Snakemake will run the first rule specified.

By convention this rule is named `all` and is used as a 'pseudo-rule' to define what the workflow will generate.

```
rule all:
    input:
        "results/sample1.stats.txt",
        "results/sample2.stats.txt",
        "results/sample3.stats.txt"

rule generate_stats:
    output: "results/{sample}.bam"
    input: "results/{sample}.stats.txt"
    log: "results/{sample}.flagstat.log"
...
```

Config files

Config files are also convenient for defining what the workflow will do.

If no targets are given on the command line, Snakemake will run the first rule specified.

By convention this rule is named `all` and is used as a 'pseudo-rule' to define what the workflow will generate.

```
samples = ["sample1", "sample2", "sample3"]
rule all:
    input:
        expand("results/{sample}.stats.txt", sample = config["samples"])

rule generate_stats:
    output: "results/{sample}.bam"
    input: "results/{sample}.stats.txt"
    log: "results/{sample}.flagstat.log"
...
```

If we define a list of samples we can condense the input section of the `all` rule using the expand function.

Config files

By defining samples in the config file (either directly or via a sample list that is read and stored in the config dictionary), your workflow becomes way more flexible.

```
### Contents of config.yml ###
samples: ["sample1", "sample2", "sample3"]
verbosity: 2
```

```
configfile: "config.yml"
rule all:
    input:
        expand("results/{sample}.stats.txt", sample = config["samples"])

rule generate_stats:
    output: "results/{sample}.bam"
    input: "results/{sample}.stats.txt"
    log: "results/{sample}.flagstat.log"
...
```

What else?

Snakemake is constantly being updated with new features. Check out the documentation (https://snakemake.readthedocs.io/), and specifically the section about writing rules.

Questions?