Snakemake Live Demo

This document shows all steps performed in the official Snakemake live demo,
such that it becomes possible to follow them at your own pace.
Solutions to each step can be found at the bottom of this document.

Prerequisites

First, install Snakemake via Conda, as outlined in the docs.
The minimal version of Snakemake is sufficient for this demo.

Step 1

First, create an empty workflow in the current directory with:

```
touch Snakefile
```

Once a Snakefile is present, you can perform a dry run of Snakemake with:

```
snakemake -n
```

Since the Snakefile is empty, it will report that nothing has to be done.
In the next steps, we will gradually fill the Snakefile with an example analysis workflow.

Step 2

The data folder in your working directory looks as follows:

```
data
├── genome.fa
├── genome.fa.amb
├── genome.fa.ann
├── genome.fa.bwt
├── genome.fa.fai
├── genome.fa.pac
├── genome.fa.sa
└── samples
    ├── A.fastq
    ├── B.fastq
    └── C.fastq
```

You will create a workflow that maps the sequencing samples in the `data/samples` folder to the reference genome `data/genome.fa`.
Then, you will call genomic variants over the mapped samples, and create an example plot.

First, create a rule called `bwa`, with input files

- `data/genome.fa`
- `data/samples/A.fastq`

and output file

- `mapped/A.bam`

To generate output from input, use the shell command

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

Providing a shell command is not enough to run your workflow on an unprepared system.
For reproducibility, you also have to provide the required software stack and define the desired version.
This can be done with the Conda package manager, which is directly integrated with Snakemake: add a directive `conda: "envs/mapping.yaml"` that points to a Conda environment definition, with the following content

```
    channels:
      - bioconda
      - conda-forge
    dependencies:
      - bwa =0.7.17
      - samtools =1.7
```

Upon execution, Snakemake will automatically create that environment, and execute the shell command within.

Now, test your workflow by simulating the creation of the file `mapped/A.bam` via

```
snakemake --use-conda -n mapped/A.bam
```

to perform a dry-run and

```
snakemake --use-conda mapped/A.bam
```

to perform the actual execution.

Step 3

Now, generalize the rule `bwa` by replacing the concrete sample name `A` with a wildcard `{sample}` in input and output file the rule `bwa`.
This way, Snakemake can apply the rule to map any of the three available samples to the reference genome.

Test this by creating the file `mapped/B.bam`.

Step 4

Next, create a rule `sort` that sorts the obtained `.bam` file by genomic coordinate.
The rule should have the input file

- `mapped/{sample}.bam`

and the output file

- `mapped/{sample}.sorted.bam`

and uses the shell command

```
samtools sort -o {output} {input}
```

to perform the sorting. Moreover, use the same `conda:` directive as for the previous rule.

Test your workflow with

```
snakemake --use-conda -n mapped/A.sorted.bam
```

and

```
snakemake --use-conda mapped/A.sorted.bam
```

Step 5

Now, we aggregate over all samples to perform a joint calling of genomic variants.
First, we define a variable

```
    samples = ["A", "B", "C"]
```

at the top of the `Snakefile`.
This serves as a definition of the samples over which we would want to aggregate.
In real life, you would want to use an external sample sheet or a config file for things like this.

For aggregation over many files, Snakemake provides the helper function `expand` (see the docs).
Create a rule `call` with input files

- `fa="data/genome.fa"`
- `bam=expand("mapped/{sample}.sorted.bam", sample=samples)`
- `bai=expand("mapped/{sample}.sorted.bam.bai", sample=samples)`,

output file

- `"calls/all.vcf"`

and shell command

```
samtools mpileup -g -f {input.fa} {input.bam} | bcftools call -mv - > {output}
```

Further, define a new conda environment file with the following content:

```
    channels:
      - bioconda
      - conda-forge
    dependencies:
      - bcftools =1.7
      - samtools =1.7
```

Step 6

Finally, we strive to calculate some exemplary statistics.
This time, we don't use a shell command, but rather employ Snakemake's ability to integrate with scripting languages like R and Python.

First, we create a rule `stats` with input file

- `"calls/all.vcf"`

and output file

- `"plots/quals.svg"`.

Instead of a shell command, we write

```
    script:
        "scripts/plot-quals.py"
```

and create the corresponding script and its containing folder in our working directory with

```
mkdir scripts
touch scripts/plot-quals.py
```

We open the script in the editor and add the following content

```
    import matplotlib
    matplotlib.use("Agg")
    import matplotlib.pyplot as plt
    from pysam import VariantFile

    quals = [record.qual for record in VariantFile(snakemake.input[0])]
    plt.hist(quals)

    plt.savefig(snakemake.output[0])
```

As you can see, instead of writing a command line parser for passing parameters like input and output files, you have direct access to the properties of the rule via a magic `snakemake` object, that Snakemake automatically inserts into the script before executing the rule.

Finally, we have to define a conda environment for the rule, say `envs/stats.yaml`, that provides the required Python packages to execute the script:

```
    channels:
      - bioconda
      - conda-forge
    dependencies:
      - pysam =0.14
      - matplotlib =2.2
      - python =3.6
```

Make sure to test your workflow with

```
snakemake --use-conda plots/quals.svg
```

Step 7

So far, we have always specified a target file at the command line when invoking Snakemake.
When no target file is specified, Snakemake tries to execute the first rule in the `Snakefile`.
We can use this property to define default target files.

At the top of your `Snakefile` define a rule `all`, with input files

- `"calls/all.vcf"`
- `"plots/quals.svg"`

and neither a shell command nor output files. This rule simply serves as an indicator of what shall be collected as results.

Step 8

As a last step, we strive to annotate our workflow with some additional information.

Automatic reports

Snakemake can automatically create HTML reports with

```
snakemake --report report.html
```

Such a report contains runtime statistics, a visualization of the workflow topology, used software and data provenance information.

In addition, you can mark any output file generated in your workflow for inclusion into the report. It will be encoded directly into the report, such that it can be, e.g., emailed as a self-contained document.
The reader (e.g., a collaborator of yours) can at any time download the enclosed results from the report for further use, e.g., in a manuscript you write together.
In this example, please mark the output file `"plots/quals.svg"` for inclusion by replacing it with `report("plots/quals.svg", caption="report/calling.rst")` and adding a file `report/calling.rst`, containing some description of the output file.
This description will be presented as caption in the resulting report.

Threads

The first rule `bwa` can in theory use multiple threads. You can make Snakemake aware of this, such that the information can be used for scheduling.
Add a directive `threads: 8` to the rule and alter the shell command to

```
bwa mem -t {threads} {input} | samtools view -Sb - > {output}
```

This passes the threads defined in the rule as a command line argument to the `bwa` process.

Temporary files

The output of the `bwa` rule becomes superfluous once the sorted version of the `.bam` file is generated by the rule `sort`. Snakemake can automatically delete the superfluous output once it is not needed anymore.
For this, mark the output as temporary by replacing `"mapped/{sample}.bam"` in the rule `bwa` with `temp("mapped/{sample}.bam")`.

Solutions

Only read this if you have a problem with one of the steps.

Step 2

The rule should look like this:

```
rule bwa:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped/A.bam"
    conda:
        "envs/mapping.yaml"
    shell:
        "bwa mem {input} | samtools view -Sb - > {output}"
```

Step 3

The rule should look like this:

```
rule bwa:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped/{sample}.bam"
    conda:
        "envs/mapping.yaml"
    shell:
        "bwa mem {input} | samtools view -Sb - > {output}"
```

Step 4

The rule should look like this:

```
rule sort:
    input:
        "mapped/{sample}.bam"
    output:
        "mapped/{sample}.sorted.bam"
    conda:
        "envs/mapping.yaml"
    shell:
        "samtools sort -o {output} {input}"
```

Step 5

The rule should look like this:

```
    samples = ["A", "B", "C"]

    rule call:
      input:
          fa="data/genome.fa",
          bam=expand("mapped/{sample}.sorted.bam", sample=samples)
      output:
          "calls/all.vcf"
      conda:
          "envs/calling.yaml"
      shell:
          "samtools mpileup -g -f {input.fa} {input.bam} | "
          "bcftools call -mv - > {output}"
```

Step 6

The rule should look like this:

```
    rule stats:
        input:
            "calls/all.vcf"
        output:
            "plots/quals.svg"
        conda:
            "envs/stats.yaml"
        script:
            "scripts/plot-quals.py"
```

Step 7

The rule should look like this:

```
    rule all:
        input:
            "calls/all.vcf",
            "plots/quals.svg"
```

It has to appear as first rule in the `Snakefile`.

Step 8

The complete workflow should look like this:

```
samples = ["A", "B"]


rule all:
    input:
        "calls/all.vcf",
        "plots/quals.svg"


rule bwa:
    input:
        "data/genome.fa",
```

```
        "data/samples/{sample}.fastq"
    output:
        temp("mapped/{sample}.bam")
    conda:
        "envs/mapping.yaml"
    threads: 8
    shell:
        "bwa mem -t {threads} {input} | samtools view -Sb - > {output}"


rule sort:
    input:
        "mapped/{sample}.bam"
    output:
        "mapped/{sample}.sorted.bam"
    conda:
        "envs/mapping.yaml"
    shell:
        "samtools sort -o {output} {input}"



rule call:
    input:
        fa="data/genome.fa",
        bam=expand("mapped/{sample}.sorted.bam", sample=samples)
    output:
        "calls/all.vcf"
    conda:
        "envs/calling.yaml"
    shell:
        "samtools mpileup -g -f {input.fa} {input.bam} | "
        "bcftools call -mv - > {output}"

rule stats:
    input:
        "calls/all.vcf"
    output:
        report("plots/quals.svg", caption="report/calling.rst")
    conda:
        "envs/stats.yaml"
    script:
        "scripts/plot-quals.py"
```