Bioinformatics Skills for Microbial Genomics Workshop
2nd February 2022

Getting things done with Conda & Snakemake

Dr. Anna Price

Research Software Engineer, Cardiff University

Supporting material for this presentation

A copy of this presentation and supporting material can be found on the GitHub organization for this workshop:

https://github.com/Bioinfo-skills-2022-CLIMB-VM/Getting-things-done-with-Conda-and-Snakemake

Overview of Conda

- Cross-platform package manager that installs and manages software packages from a repository called Anaconda
- Over 1,500 packages are available in the Anaconda repository
- Can create isolated environments that can contain different versions of software packages and different versions of Python
- The installed software packages can be written in any language

Conda, Miniconda, Anaconda and Bioconda

- **Conda** is the **package manager**
- **Miniconda** and **Anaconda** are **distributions**
- **Bioconda** is a **channel** for bioinformatics software

The Anaconda distribution contains pre-installed packages.

The Miniconda distribution just includes conda with its minimal dependencies

Miniconda is installed on the CLIMB-BD VM

Why should I use Conda?

- Much easier to install software packages using Conda than trying to install from source

- The isolated environments that conda can build can be very useful!

  - Using YAML files we can create reproducible environments
  - Software with conflicting underlying build dependencies can be isolated in different environments

Getting started with Conda on CLIMB-BD VM

Initialise conda

```
conda init bash
```

Reload bashrc

```
source ~/.bashrc
```

You should see the base environment activated

Working with Conda Environments

The default conda environment is called **base**. Don’t install packages in your base environment! Instead create new separate environments to keep your software packages isolated from each other

For example, to install the bacterial genome assembler Shovill (https://github.com/tseemann/shovill) in its own environment

```
conda create -c conda-forge -c bioconda -c defaults shovill --name shovillenv
```

Using YAML files to build Conda Environments

environment.yml files can be used to share your Conda environment with others

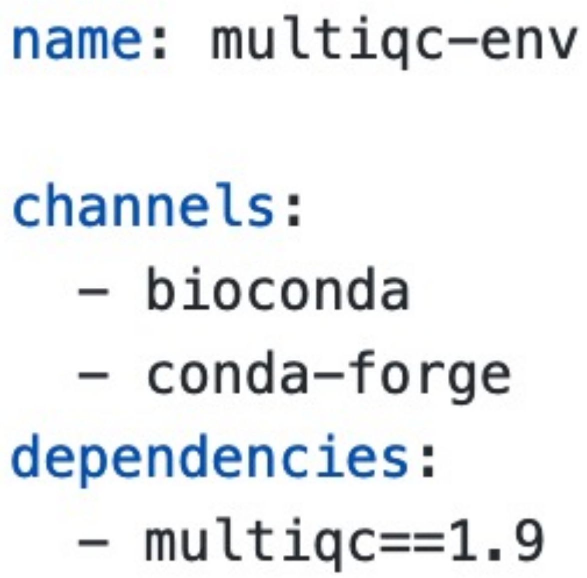
To create an environment from an yml

```
conda env create -f environment.yml
```

You can also export an existing conda env into an yml

```
conda env export > environment.yml
```

For reproducibility pin the software versions in the conda env



```yaml
name: multiqc-env

channels:
  - bioconda
  - conda-forge
dependencies:
  - multiqc==1.9
```

What is Mamba and why should I use it?

- Mamba is a reimplementation of conda in C++
- It uses multithreading and libsolv for faster dependency solving

  => Faster way of building conda environments

- Pre-installed on the CLIMB-BD VM
- Uses same command line parser as conda

Mamba can be very useful when building conda environments with lots of underlying dependencies

Working with Mamba

For example, to install the bacterial genome assembler Shovill (https://github.com/tseemann/shovill) in its own environment using mamba

```
mamba create -c conda-forge -c bioconda -c defaults shovill --name shovillenv
```

Limitations of Conda

There are advantages and disadvantages to Conda:

- Easy to build conda envs, either on command line or using yaml files
- Widely used
- Even using mamba, environments are sometimes slow to resolve
- Non-deterministic dependency resolution; can be differences between platforms

Conda environments are not inherently cross-platform!

Overview of Snakemake

- Workflow management system that can be used to create **reproducible** and **scalable** workflows
- Can build workflows from existing software and tools
- Defines a software analysis in terms of a rule. Rules are used to build workflows
- Can be used in conjunction with Conda and Singularity
- Python-based syntax

Why should I use Snakemake?

- Greatly simplifies the writing of bioinformatics workflows
- For Python users the syntax will be very familiar
- Integration with conda can be very useful
- Comes with useful features such as linting (checks quality of code) and can automatically generate unit tests

Getting started with Snakemake on CLIMB-BD VM

Use mamba to create a Snakemake environment

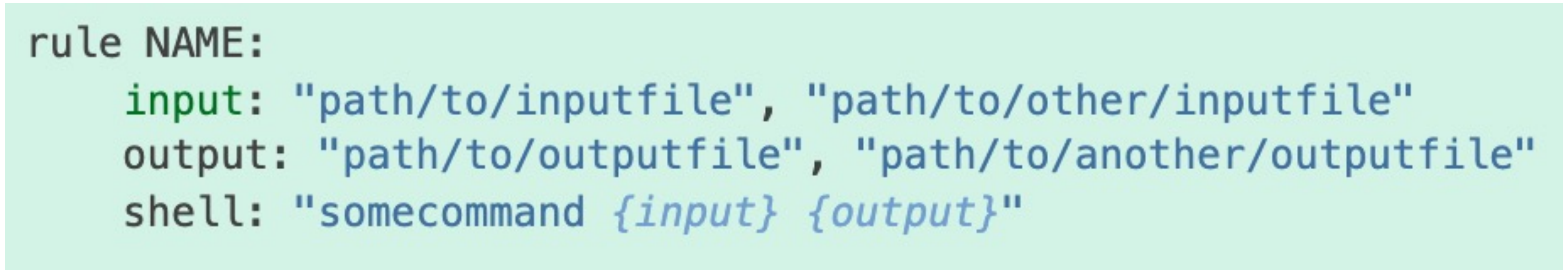
```
mamba create -c conda-forge -c bioconda -n snakemake snakemake –y
```

Activate Snakemake environment

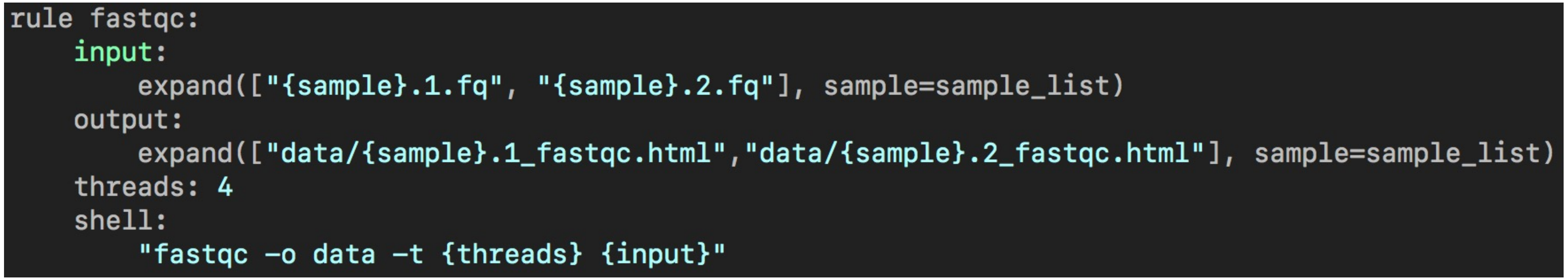
```
conda activate snakemake
```

Snakemake Rules

- A Snakemake workflow defines a software analysis in a rule
- Rules are specified in a Snakefile



```
rule NAME:
    input: "path/to/inputfile", "path/to/other/inputfile"
    output: "path/to/outputfile", "path/to/another/outputfile"
    shell: "somecommand {input} {output}"
```



```
rule fastqc:
    input:
        expand(["{sample}.1.fq", "{sample}.2.fq"], sample=sample_list)
    output:
        expand(["data/{sample}.1_fastqc.html","data/{sample}.2_fastqc.html"], sample=sample_list)
    threads: 4
    shell:
        "fastqc -o data -t {threads} {input}"
```

Snakemake and Reproducibility

Need to consider **distribution** and **reproducibility** when creating a workflow:

- Snakemake workflows are often reliant on many software packages
- Unreasonable to expect user to install all these packages themselves
- Pipeline should run uniformly regardless of infrastructure

Solution:

Use Conda environments or Singularity containers!

Using Conda with Snakemake

- Conda environments can be defined for the entire workflow or on a per rule basis. Upon execution of a workflow, Conda can obtain and deploy the defined software
- To deploy Conda when running your Snakemake workflow, use the --use-conda flag

```
rule fastqc
    input:
        expand(["{sample}_1.fq", "{sample}_2.fq"], sample=sample_list)
    output:
        expand(["data/{sample}_1_fastqc.html" , "data/{sample}_2_fastqc.html"], sample=sample_list)
    threads: 4
    conda: "envs/fastqc.yml"
    container: "docker://quay.io/biocontainers/fastqc"
    shell:
        "fastqc -o data -t {threads} {input}"
```

Using Singularity with Snakemake

Singularity is a container engine (more on this in the next session!)

To deploy Singularity when running your Snakemake workflow, use the `--use-singularity` flag

There are different ways of managing Singularity containerisation with Snakemake:

- One container for entire workflow **or** define containers at a rule level
- Write your own Singularity recipes **or** use existing container images from a resource such as biocontainers **or** use ad-hoc combination of Conda package management with containers

Snakemake Tutorials

Tutorials for Snakemake can be found on their website:

Basic tutorial:

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

Advanced tutorial:

https://snakemake.readthedocs.io/en/stable/tutorial/advanced.html