Snakemake

Johannes Köster
Genome Informatics, Institute of Human Genetics,
Faculty of Medicine, University Duisburg-Essen

April 10, 2014
Structure

1. Motivation

2. Basic Idea

3. Advanced Features
Outline

1 Motivation

2 Basic Idea

3 Advanced Features
Motivation

What we liked about GNU Make:

- text based
- rule paradigm
- lightweight

And what not:

- cryptic syntax
- limited scripting
- multiple output files
- scalability
Snakemake

- hook into python interpreter
- pythonic syntax for rule definition
- full python scripting
- scalability
- workflow specific functionality beyond Make basics
- stable community:
Outline

1. Motivation
2. Basic Idea
3. Advanced Features
SAMPLES = "500 501 502 503".split()

# require a bam for each sample
def all:
    input:
        expand("{sample}.bam", sample=SAMPLES)

# map reads
def map:
    input:
        "reference.bwt",
        "{sample}.fastq"
    output:
        "{sample}.bam"
    threads: 8
    shell:
        "bwa mem -t {threads} {input} | " # refer to threads and input files
        "samtools view -Sbh - > {output}" # refer to output files

# create an index
def index:
    input:
        "reference.fasta"
    output:
        "reference.bwt"
    shell:
        "bwa index {input}"
Basic Usage

# perform a dry-run  
$ snakemake -n

# execute the workflow using 8 cores  
$ snakemake -j 8

# execute the workflow on a cluster (with up to 20 jobs)  
$ snakemake -j 20 --cluster "qsub -pe threaded {threads}"
Visualization

# visualize the DAG of jobs
$ snakemake --dag | dot | display
Outline

1. Motivation

2. Basic Idea

3. Advanced Features
Advanced Syntax

SAMPLES = "500 501 502 503".split()

rule all:
    input:
        expand("{sample}.bam", sample=SAMPLES)

    # map reads with peanut
    rule map:
        input:
            "reference.hdf5",
            "{sample}.fastq"
        output:
            "{sample}.bam"
        threads: 8
        resources: gpu=1  # define an additional resource
        version: shell("peanut --version")
        shell:
            "peanut map -t {threads} {input} | "
            "samtools view -Sbh - > {output}"

    # create an index with peanut
    rule index:
        input:
            "reference.fasta"
        output:
            "reference.hdf5"
        shell:
            "peanut index {input} {output}"
Scheduling

Maximize the number of running jobs with respect to

• priority
• number of descendants
• input size
while not exceeding

• provided cores
• provided resources

A multi-dimensional knapsack problem.
Sub-Workflows

SAMPLES = "500 501 502 503".split()

# define subworkflow
subworkflow:
    workdir: "../mapping"

rule all:
    input:
        expand("{sample}/results.xprs", sample=SAMPLES)

# estimate transcript expressions
rule express:
    input:
        REF,
        mapping("{sample}.bam")  # refer to output of subworkflow
    output:
        "{sample}/results.xprs"
    shell:
        "express {input} -o {wildcards.sample}"
HTML5 Reports

```python
from snakemake.utils import report

rule report:
    input:
        T1="results.csv", F1="plot.pdf"
    output:
        html="report.html"
    run:
        report(""
        Some Title
        
        See table T1, display some math
        
        .. math::
           |cq_0 - cq_1| > \{MDIFF\}
        
        ", output.html, **input)
```
Data Provenance

Summarize output file status

$ snakemake --summary

<table>
<thead>
<tr>
<th>file</th>
<th>date</th>
<th>rule</th>
<th>version</th>
<th>status</th>
<th>plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>500.bam</td>
<td>Thu Apr 10 10:55:17 2014</td>
<td>map 1.0</td>
<td>ok</td>
<td>no update</td>
<td></td>
</tr>
<tr>
<td>501.bam</td>
<td>Thu Apr 10 10:55:17 2014</td>
<td>map 1.0</td>
<td>ok</td>
<td>no update</td>
<td></td>
</tr>
<tr>
<td>502.bam</td>
<td>Thu Apr 10 10:55:17 2014</td>
<td>map 1.0</td>
<td>updated input files</td>
<td>update pending</td>
<td></td>
</tr>
<tr>
<td>503.bam</td>
<td>Thu Apr 10 10:55:17 2014</td>
<td>map 0.9</td>
<td>version changed to 1.0</td>
<td>no update</td>
<td></td>
</tr>
</tbody>
</table>

Trigger updates:

# update files with changed versions
$ snakemake -R `snakemake --list-version-changes`

# update files with changed code
$ snakemake -R `snakemake --list-code-changes`
Conclusion

Snakemake is a Make-like workflow system providing

- a readable syntax
- sophisticated scripting with python
- scalability from single-core to cluster
- support for hybrid computing
- data provenance
- modularization capabilities

Roadmap:

- DRMAA support
- a workflow or rule library

http://bitbucket.org/johanneskoester/snakemake