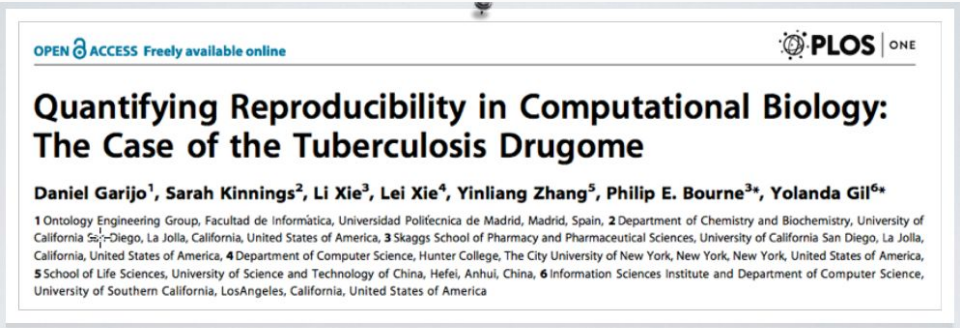
Computational Pipeline Strategies

Characteristics of Bioinformatics Analyses

- Many different files involved
- Files are often large
- Each file analyzed in sequence of steps
- Each step can be time consuming/ computationally intensive
- Use many different tools, each with analysis-specific parameters



OPEN ACCESS Freely available online

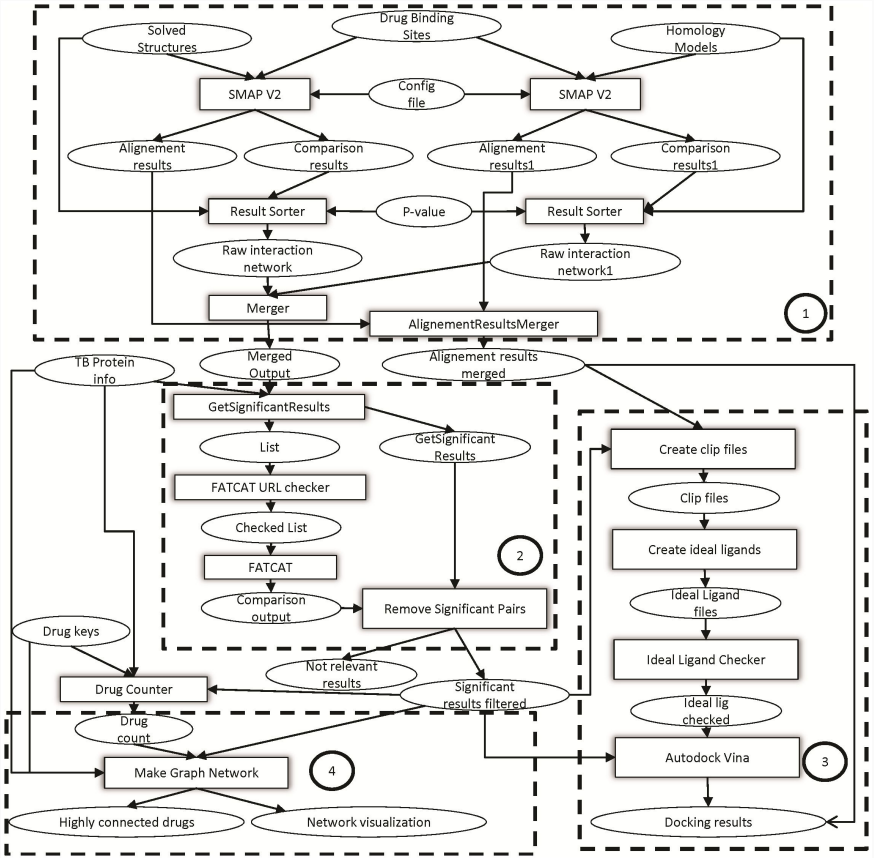
PLOS ONE

Quantifying Reproducibility in Computational Biology: The Case of the Tuberculosis Drugome

Daniel Garijo[1], Sarah Kinnings[2], Li Xie[3], Lei Xie[4], Yinliang Zhang[5], Philip E. Bourne[3]*, Yolanda Gil[6]*

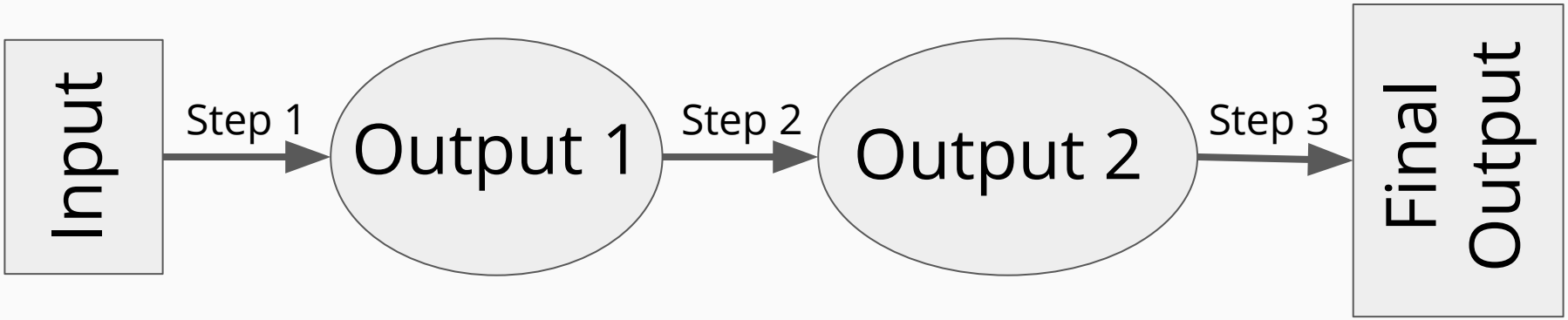
1 Ontology Engineering Group, Facultad de Informàtica, Universidad Politécnica de Madrid, Madrid, Spain, **2** Department of Chemistry and Biochemistry, University of California San Diego, La Jolla, California, United States of America, **3** Skaggs School of Pharmacy and Pharmaceutical Sciences, University of California San Diego, La Jolla, California, United States of America, **4** Department of Computer Science, Hunter College, The City University of New York, New York, New York, United States of America, **5** School of Life Sciences, University of Science and Technology of China, Hefei, Anhui, China, **6** Information Sciences Institute and Department of Computer Science, University of Southern California, LosAngeles, California, United States of America

To reproduce the result of a typical computational biology paper requires ~300 hours!



Computational Pipelines

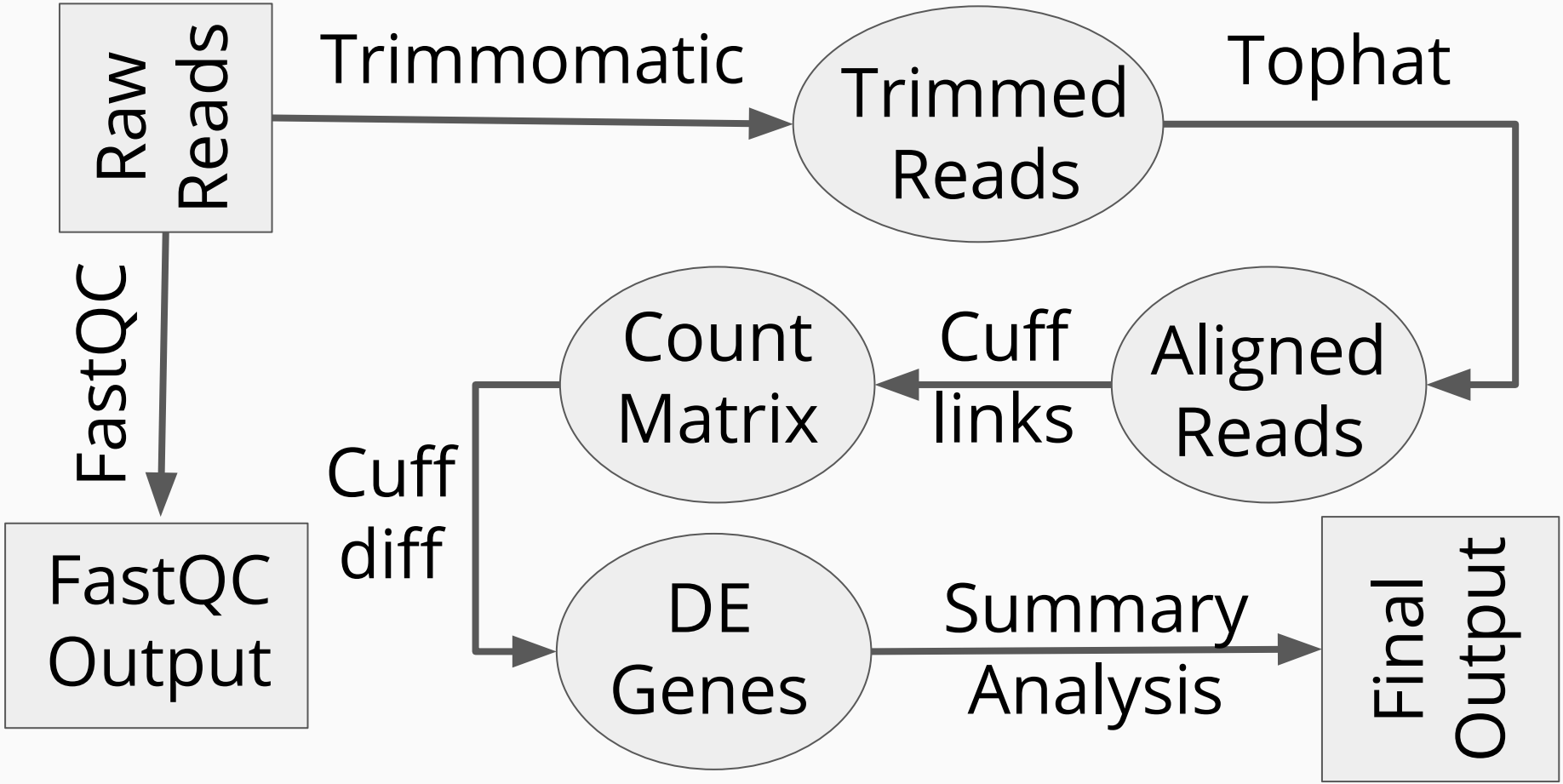
- **Pipeline** or **workflow**: A serialized set of data processing steps where output from one step is input into the following step



Example Analysis

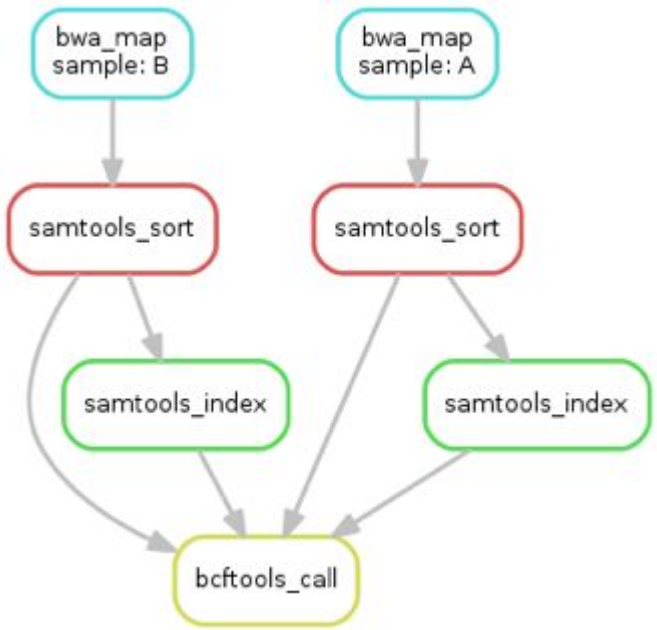
- Goal: Profile transcriptional landscape of mammalian cardiac regeneration
- Steps:
  - Perform quality control on FASTQ files
  - Adapter- and quality- trim reads
  - Align reads to reference genome
  - Quantify gene expression
  - Identify differentially expressed genes
  - Interpret findings

Example Pipeline in Bioinformatics



Analysis as a DAG

- Analysis is a **D**irected **A**cyclic **G**raph (DAG)
  - Deterministic sequence of steps
  - Output of step → Input of next
  - Computation “flows” down graph, no cycles
- Pipelines are an implementation of a DAG



Naive Model

- Run all commands on all files manually
- User must track:
  - All parameters and commands for all steps
  - Which steps completed for each file
  - Whether a step failed, is file complete?
- Onerous for even simple workflows
- Impossibly impractical for large numbers of samples!

Slightly-Less-Naive Model

- Solution:
  - Bash scripts
  - Parallelize w/ qsub
- Single-machine or cluster command-line workflows
- No easy restart from Failure
- Sometimes expedient for small workflows

```
# run_pipeline.sh - implements NGS analysis
module load sratoolkit bwa

# -sync tells qsub to wait until job is done
QARGS=-P bf528 -b y -sync -cwd

# downloads reference sequence
qsub $QARGS ./download_GRCh38.sh

# index reference w/ bwa
qsub $QARGS bwa index GRCh38.genome.fa.gz

qsub $QARGS fastq-dump --split-files SRR4243395
qsub $QARGS "bwa mem GRCh38.genome.fa \
    SRR4243395_1.fq.gz SRR4243395_2.fq.gz \
    > SRR4243395.sam"

qsub $QARGS fastq-dump --split-files SRR4243396
qsub $QARGS "bwa mem GRCh38.genome.fa \
    SRR4243396_1.fq.gz SRR4243396_2.fq.gz \
    > SRR4243396.sam"

# and so on...
```

Properties of the ideal pipeline system

- **General purpose:** familiar language, can apply to any task
- **Modular:** any language, well-tested components
- **Scalable:** parallelize for free, independent of components
- **Integrated:** metadata, viz, versioning, reporting
- **Versioned:** reproduce from snapshots in time
- **Idempotent**: resume from failure, guarantee outputs

Common Pipeline/Workflow Tools

make	Tool for building software on UNIX-based systems
snakemake	**Python-based workflow language & software**
Nextflow	**Groovy-based, implements *dataflow* programming model**
Airflow	Airbnb/Apache pipeline creator, python-based
SciPipe	GO-based pipeline language

snakemake

Snakemake

- Python-based workflow language/software
- Based on GNU make
- File-based production *rules*:
  - Rule is: input file(s) → transform → output file(s)
- Excellent documentation
- Implicit parallelism and cluster integration
- Includes reproducibility tools
  - Shared rule wrappers
  - Conda and singularity integration

Snakemake Example: Variant Calling

- Starting Data
  - Short read sequences in fastq format
  - Publicly available genome fasta file
- Required Steps:
  - Map reads to genome
  - Sort and index mapped reads
  - Call Variants
- Target Output:
  - VCF file of found variants

Snakemake Example - bwa index

3. ...on the file **data/genome.fa**,
(assume we have already created it)

1. If we want to create the file
data/genome_index...

```
rule index:
    input: "data/genome.fa"
    output: "data/genome_index"
    shell:
        "bwa index -p genome_index {input}"
```

2. ...we need to run this command...

Snakemake Example - Mapping

From previous rule, snakemake knows how to make **data/genome_index** using **data/genome.fa**

```
rule bwa_map:
    input:
        index="data/genome_index",
        fq="data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem {input.index} {input.fq} | \
           samtools view -Sb - > {output}"
```

Snakemake Example - Generalize

```
rule bwa_map:
    input:
        index="data/genome_index",
        fq="data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    shell:
        "bwa mem {input.index} {input.fq} | \
           samtools view -Sb - > {output}"
```

Snakemake Example - Sort Alignments

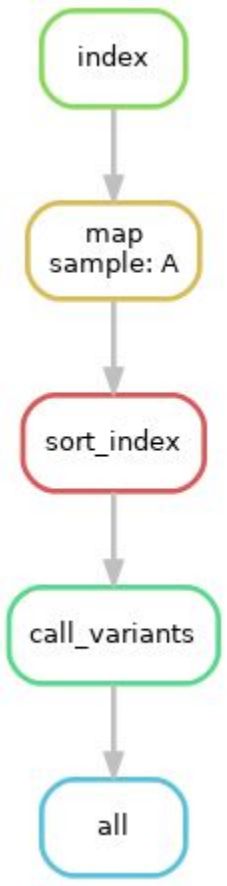
From previous rule, snakemake knows how to make **mapped_reads/{sample}.bam** using **data/{sample}.fastq**

```
rule samtools_sort:
    input:
        "mapped_reads/{sample}.bam"
    output:
        bam="sorted_reads/{sample}.bam",
        bai="sorted_reads/{sample}.bam.bai"
    shell: """
        samtools sort -T \
           sorted_reads/{wildcards.sample} \
           -O bam {input.bam} > {output}
        samtools index {output}
    """
```

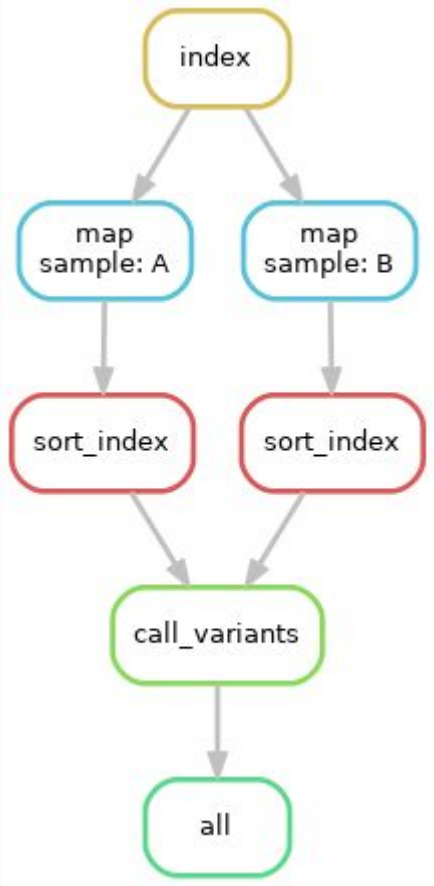
Snakemake Example - Call Variants

```
SAMPLES = [“A”, “B”]
rule bcftools_call:
    input:
        fa=”data/genome.fa”,
        bam=expand(“sorted_reads/{sample}.bam”,
                   sample=SAMPLES),
        bai=expand(“sorted_reads/{sample}.bam.bai”,
                   sample=SAMPLES)
    output:
        "calls/all.vcf"
    shell:
        "samtools mpileup -g -f {input.fa} {input.bam} |
         bcftools call -mv - > {output}"
```

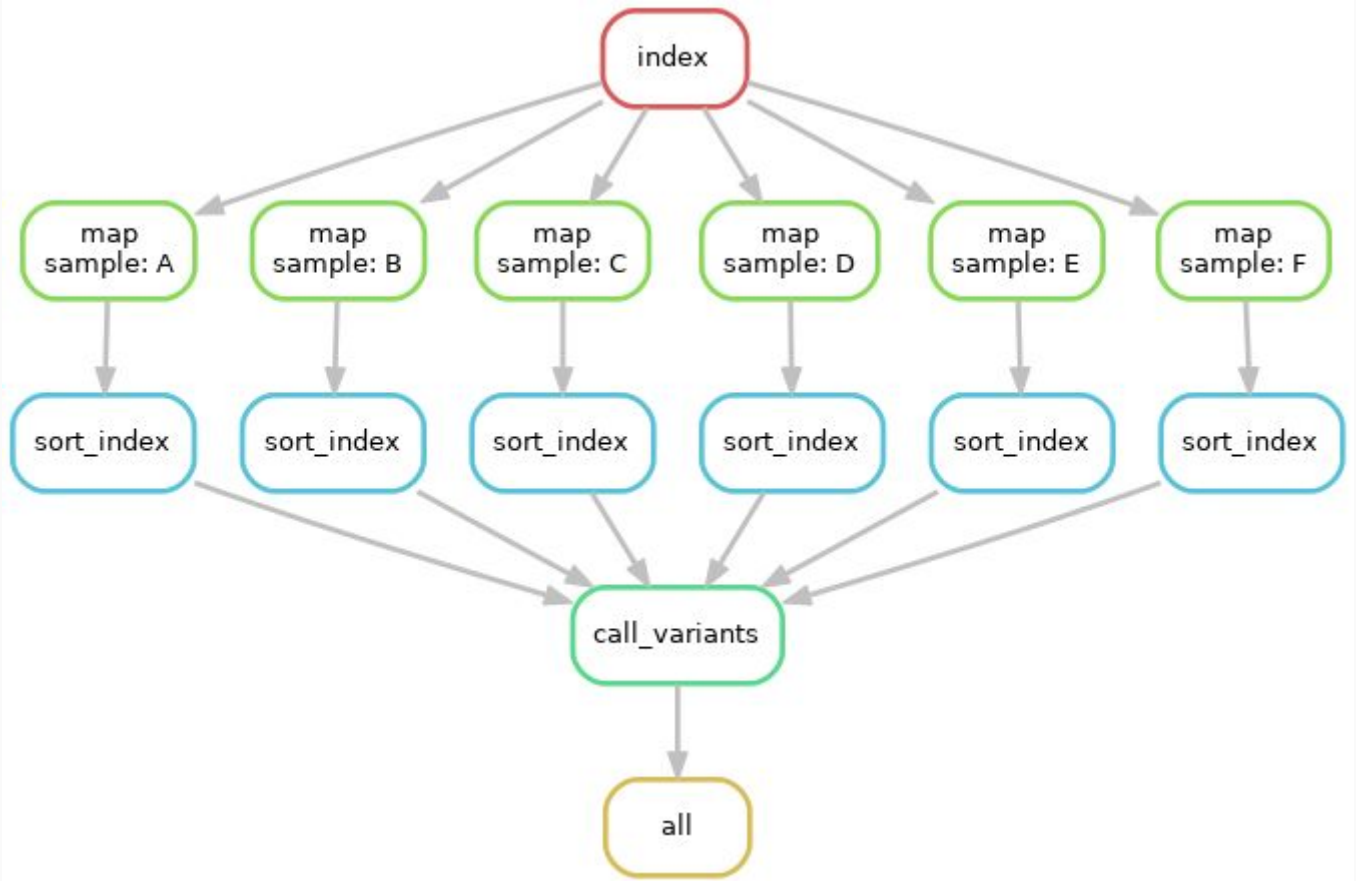
Snakemake Example : One Sample



Snakemake Example : Two Samples



Snakemake Example : Many Samples



Snakemake Alternative: Nextflow

- Implements *dataflow* programming model
- *Process*: Input → transform → output
- Processes are connected via “pipes”
- Different strategies:
  - snakemake: “pull” data by specifying outputs
  - Nextflow: “push” data through pipes from input
- Each tool has its trade-offs

Nextflow Example

```
params.in = “$baseDir/data/sample.fa”

/*
 * split a fasta file in multiple files
 */
process splitSequences {

    input:
    path 'input.fa' from params.in

    output:
    path 'seq_*' into records

    """
    awk '/^>/{f="seq_"++d} {print > f}' < input.fa
    """
}
```

```
/*
 * Simple reverse the sequences
 */
process reverse {

    input:
    path x from records

    output:
    stdout into result

    """
    cat $x | rev
    """
}
```

Summary

- Workflow management software is indispensable:
  - Simplifies analyzing many files in the same way
  - Seamlessly runs code locally or on a cluster
  - Documents your analysis - reproducible!
  - ***Learn how to use it!***
- Many different workflow/pipeline tools
  - No one is best, each has strengths/weaknesses
  - Pick which one you like