Reproducibility in Bioinformatics

Deb Triant & Marcus Bobar

Research Computing, University of Virginia

dtriant@virginia.edu, mb5wt@virginia.edu

Workshop outline

- Difficulties in achieving reproducibility
- Potential problems with bioinformatics pipelines
- Some helpful tools
- Snakemake & Nextflow examples

Reproducibility in science

- Reproducibility - redo a scientific experiment & generate similar results
  - Same sample, software, data, code - same result?

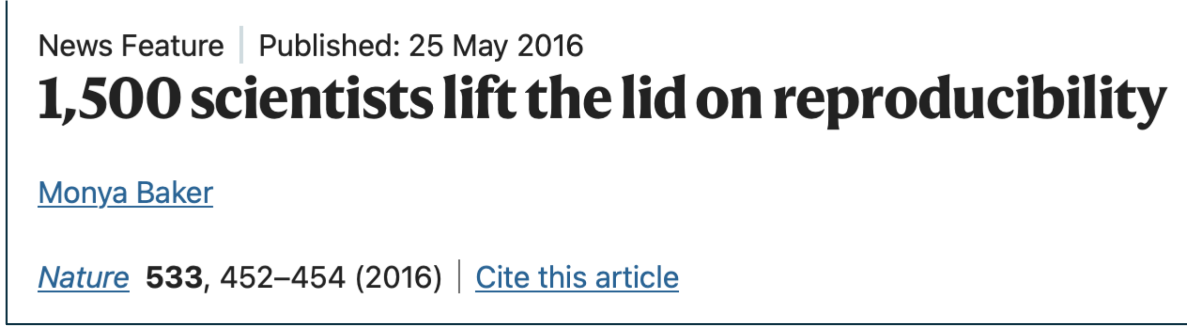
- Replication - different data, same methods - conclusions consistent?

- Reusability - Will someone be able to use your pipeline in the future?
  - Will you be able to use it?

Reproducibility Problem

- Where did you do the analysis - laptop, server, lab computer, environment
- Are you using the most recent version (scripts, datasets, analyses)
- We just used the default settings!

Studies in reproducibility


News Feature | Published: 25 May 2016

1,500 scientists lift the lid on reproducibility

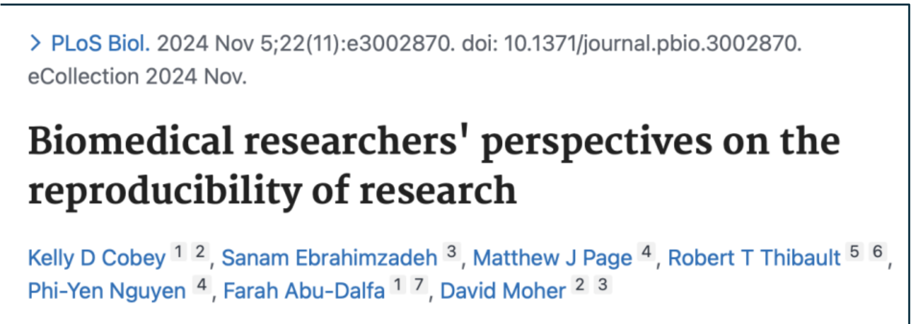
Monya Baker

Nature **533**, 452–454 (2016) | Cite this article

- Nature (2016) - Found that 70% of researchers have failed in reproducing another researcher's results & >50% failed to reproduce their own

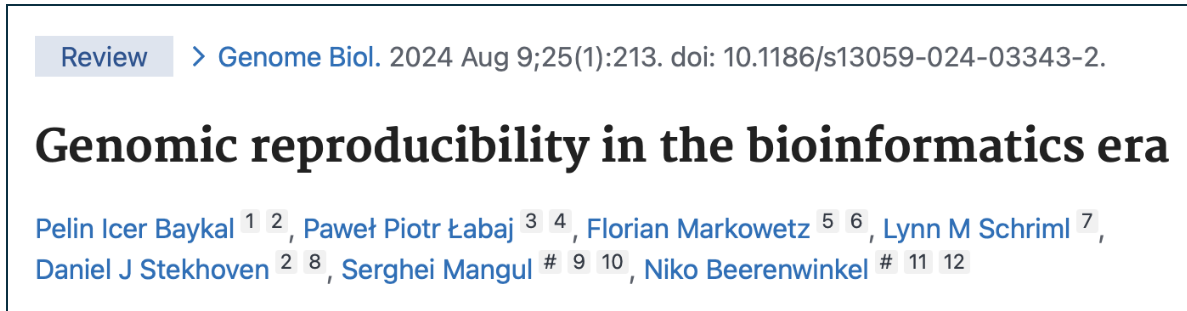
- PLoS Biology (2024) - Biomedical researchers - 72% reported "reproducibility crisis"

- Genome Biol (2024) - Reproducibility in bioinformatics era


> PLoS Biol. 2024 Nov 5;22(11):e3002870. doi: 10.1371/journal.pbio.3002870. eCollection 2024 Nov.

Biomedical researchers' perspectives on the reproducibility of research

Kelly D Cobey[1 2], Sanam Ebrahimzadeh[3], Matthew J Page[4], Robert T Thibault[5 6], Phi-Yen Nguyen[4], Farah Abu-Dalfa[1 7], David Moher[2 3]


Review > Genome Biol. 2024 Aug 9;25(1):213. doi: 10.1186/s13059-024-03343-2.

Genomic reproducibility in the bioinformatics era

Pelin Icer Baykal[1 2], Paweł Piotr Łabaj[3 4], Florian Markowetz[5 6], Lynn M Schriml[7], Daniel J Stekhoven[2 8], Serghei Mangul[# 9 10], Niko Beerenwinkel[# 11 12]

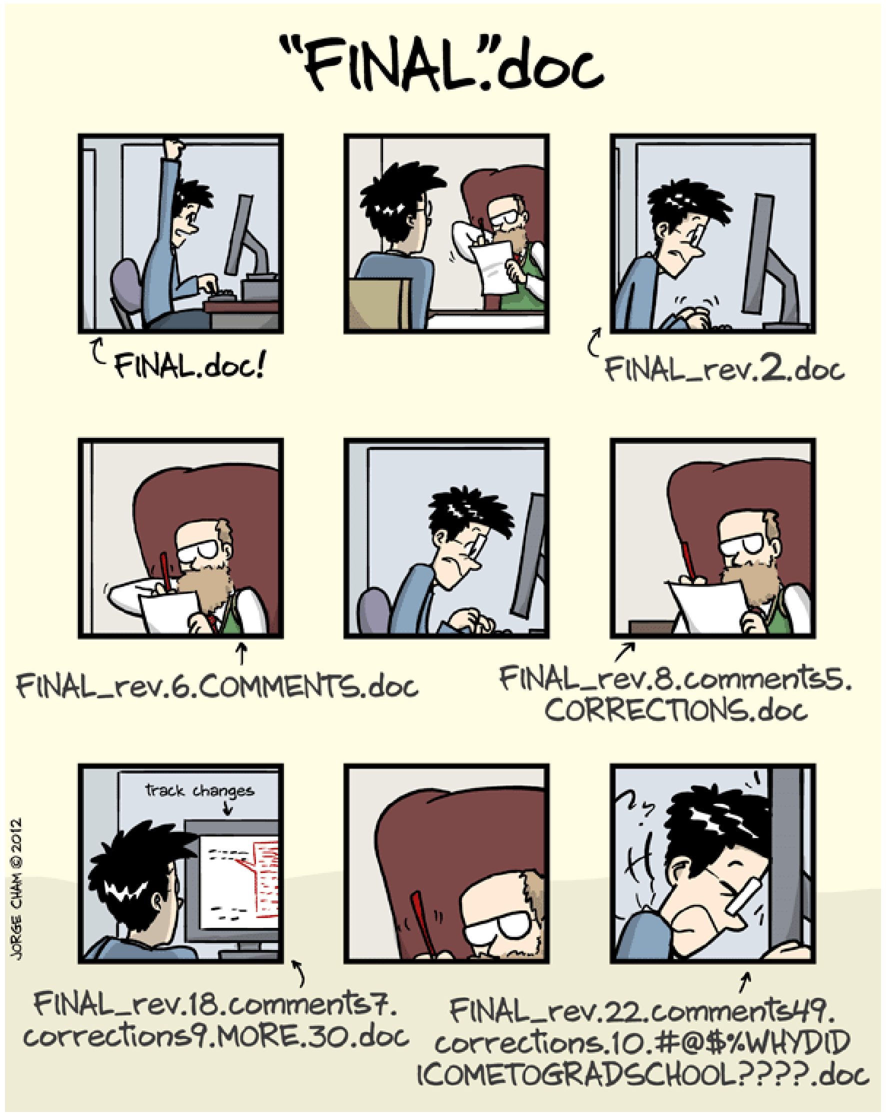
Challenges of Bioinformatics

- So many tools, often with:
    - Multiple versions & releases
    - Complex dependencies & hidden parameters, starting seeds
    - Running tools locally vs on HPC
    - Formatting conversions between software
    - Scalability - how tools handle datasets increasing in size
    - Keeping codes organized!

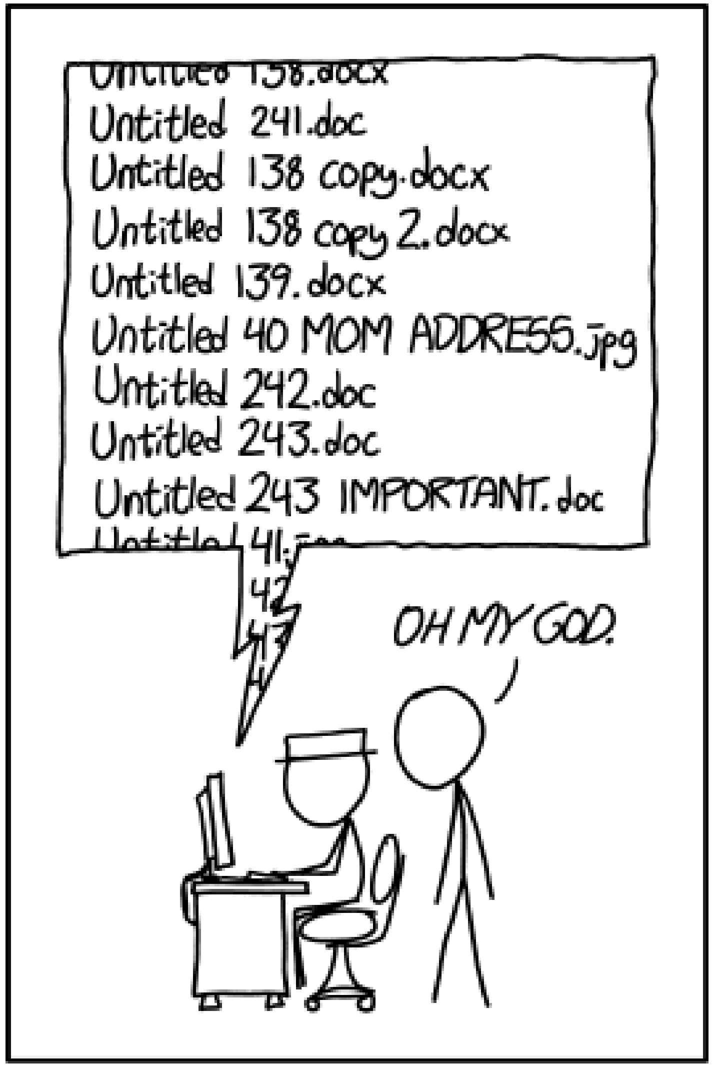
Aspects of reproducibility

- Version control
- Environment management
- Data storage
- Containers
- Tool/software maintenance

Saving document versions


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDIDICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012

phdcomics.com



PROTIP: NEVER LOOK IN SOMEONE ELSE'S DOCUMENTS FOLDER.

xkcd.com

Version Control

GitHub: https://github.com

- Track and manage changes to your code & files
- Store and label changes at every step
- Small or large projects
- Collaborate on projects and minimize conflicting edits
- Works on multiple platforms (MacOS, Windows, Linux)

Environment Management

- Conda/Mamba environments
  - Isolated spaces for each project with specific tool versions
  - Manage Python versions and dependencies
  - Install packages and software directly into environment
  - Stable and reproducible place to run code and applications
  - Not limited to Python, can run bash, Rscript
  - YAML configuration file to create or export and transfer an environment

Storing results

- Public repositories for sequence data - required for most journals
  NCBI: https://www.ncbi.nlm.nih.gov

- Ensembl: https://www.ensembl.org/index.html

- Always document and archive changes, especially if unpublished:
  - genome assembly versions
  - sequence data: SNPs, isoforms

Containers

- Portable environments that run across different computing environments
- Contain packages, software and dependencies that remain isolated from host infrastructure
- Standalone unit of software and can produce same results on different machine or server

Bioinformatic Pipelines

- Typical bioinformatics workflows involve many steps:
- FASTQ → QC → Alignment → Sorting → Variant Calling → Annotation
  - FASTQ files need quality check and trimming
  - Cutadapt
  - BWA
  - Samtools
  - Freebayes
  - VCFtools
- Create pipeline to string software together for “final” output

Bioinformatic Pipeline challenges

- Complex dependencies between steps
- Formatting inconsistencies
- Hard to reproduce results - scalability, parameters, version changes
- Difficult to parallelize efficiently
- Manual scripts often fail on HPC

Bioinformatic Pipelines on HPC

- Which modules were loaded?
- Where are scripts being run
- Tracking paths - hard-coded in scripts?
- Out/error files - software vs slurm conflicts

Goal: Automate and track these workflows

Snakemake

https://snakemake.github.io/

- **Snakemake** is a workflow management system designed for scientific pipelines
- Created by Johannes Köster, first released in 2012
- Based on UNIX make - originally created in 1976 but still standard use
- Python based - "*snake-make*"
- Free and open source, available on Mac, Windows, Unix

- https://snakemake.readthedocs.io/en/stable/
- https://github.com/snakemake

Snakemake format

- Similar to writing shell scripts but snake files contains sets of rules
- Format is based on Python structure
- Snakemake reads from snakefile that defines the rules
- Snakefile rules have a target output
- Snakemake uses pattern matching to follow the inputs, outputs and commands contained in rules to reach final target output

Snakemake Core Idea

Instead of defining *steps*, you define **rules that produce files**.

```
rule align:
    input:
        "reads.fastq"
    output:
        "aligned.bam"
    shell:
        "bwa mem ref.fa {input} > {output}"
```

Snakemake builds a **directed acyclic graph (DAG)** automatically.

Fastq → Cutadapt → BWA → Sorted BAM → Freebayes → VCF

Recommended Pipeline Directory Structure

Example:

```
bioinformatics_pipeline/

├── Snakefile
├── config/
│   └── config.yml
├── envs/
│   └── bwa.yml
├── rules/
│   ├── alignment.smk
│   ├── qc.smk
│   └── variant_calling.smk
├── scripts/
│   └── custom_processing.py
├── data/
│   └── raw/
├── results/
│   ├── bam/
│   ├── qc/
│   └── variants/
└── logs/
```

Benefits:

- separates **workflow logic from data**
- easier debugging
- easier collaboration

Common practice:

- `config/` → parameters and sample tables
- `envs/` → reproducible environments
- `rules/` → modular workflow steps
- `results/` → generated outputs

A clean directory structure makes pipelines easier to maintain and reproduce.

Snakefile breakdown

- Fastq files that need trimming - input: sample.fastq

↓

- Cutadapt - output: sample-trimmed.fastq

↓

- BWA - align trimmed fastq to assembly output: sample-aligned.sam

↓

- Samtools sorting, indexing - output: sample-sorted.bam

↓

- Freebayes variant calling - output: sample-variants.vcf

Example snakefile

```
rule all:
     input:
         "variants/sample1.vcf"
  rule trim:

        input:

          "reads/sample1.fastq"

        output:

          "trimmed_reads/sample1-trimmed.fastq"

     shell:

        cutadapt -A TCCGGGTS -o {output} {input}


  rule align:
     input:
         "trimmed_reads/sample1-trimmed.fastq"
     output:
         "bam/sample1.bam"
     threads: 1
     shell:
         "bwa mem -t {threads} ref.fa {input} | samtools view -Sb - > {output}"


  rule call_variants:
     input:
         "bam/sample1.bam"
     output:
         "variants/sample1.vcf"
     shell:
         "freebayes -f ref.fa {input} > {output}"
```

Snakemake takes first rule as the target then constructs graph of dependencies

{wildcards} serve as placeholders within rules to operate on multiple files via pattern matching

Snakemake exercises on HPC

- Class data:
/project/hpc_training/reproducibility/snakemake
- `$ cp /project/hpc_training/reproducibility/snakemake .`

- GCF_000005845.2_ASM584v2_genomic.fna - genome assembly
- SRR2584863_1.fastq - fastq sequence file, paired-1
- SRR2584863_2.fastq - fastq sequence file, paired-2
- *.smk - snakemake files
- config_variant.yml - configuration file
- submit_snakemake.sh - sample slurm file

Running jobs on interactive node

- Run interactively - good for testing

```
$ ijob -c 1 -A hpc_training -p interactive –v -t 2:00:00

$ cp /project/hpc_training/reproducibility/snakemake .
```

Modules

```
$ module spider <package>
    - specifics and version of package available

$ module spider snakemake

$ module load snakemake/9.8.1

$ module list

$ snakemake -help
```

Other modules needed for today

```
$ module load bwa/0.7.17
$ module load  cutadapt/4.9
$ module load snakemake/9.8.1
$ module load freebayes/1.3.10
$ module load  samtools/1.21
```

Running snakemake - genome alignment

- Snakefile - file.smk, contains rules for snakemake

- `$ snakemake -c 1 -s align.smk`
- --dry-run -np good to test first without producing output

  -n only show steps, don't run, -p print shell commands

  -c number of cores

  -s needed if using a named snakefile (if just called "snakefile", don't need the –s flag)

```
$ snakemake --dag| dot -Tpng > dag_align.png
```

Running snakemake - variant detection

- Snakefile - file.smk, contains rules for snakemake

```
$ snakemake -c 1 -s variant-call.smk
```

--dry-run

-c number of cores

-s needed if using a named snakefile (if just called "snakefile", don't need)

```
$ snakemake --dag -s variant-call.smk | dot -Tpng \
> dag_variant.png
```

Snakemake Examples on HPC

- Not recommended to hard-code files within snake file
- Can organize sample names, file paths, and software parameters in a YAML configuration file
- YAML - serialization language that transforms data into a format that can be shared between systems
- With snakemake, configuration file is a reference for the workflow

Running snakemake with config file

- Snakefile - file.smk, contains rules for snakemake

```
$ snakemake -c 1 -s variant-yml.smk --
configfile config_variant.yml
```

--configfile – directing snakemake to a config file

-c number of cores

-s needed if using a named snakefile

Reproducible environments

Snakemake supports reproducible environments.

Example with Conda:

```
rule fastqc:
     input: "reads.fastq"
     output: "qc.html"
     conda:
         ”~/.conda/envs/fastqc_env” #path to conda
 environment
     shell:
         "fastqc {input}"
```

Benefits: Easy dependency management, portable workflows

Using Environments

```
├── Snakefile
├── config/
│   └── config.yml
├── envs/
│   └── bwa.yml
├── rules/
│   ├── alignment.smk
│   ├── qc.smk
│   └── variant_calling.smk
├── scripts/
│   └── custom_processing.py
├── data/
│   └── raw/
├── results/
│   ├── bam/
│   ├── qc/
│   └── variants/
└── logs/
```

Can also create a environment.yml file, list conda envs and what to install

name: bwa.yml

channels:

- conda-forge

- bioconda

dependencies:

-bwa=0.7.17

Snakemake with conda environment

$ module load miniforge

$ conda create

$ conda activate

$ snakemake command

$ screen/tmux

- keeps session running when disconnected
  - make sure to connect to same login node,
  - confirm login node with: `hostname`

Can create different conda environment for different rules

Using environments via surm

submit_snakemake.sh

```
#!/bin/bash
#SBATCH -A hpc_training                     # account name
#SBATCH -p standard                         # partition/queue
#SBATCH --nodes=1                           # number of nodes
#SBATCH --ntasks=1                          # 1 task
#SBATCH --cpus-per-task=1                   # total cores per task
#SBATCH -t 01:00:00                         # time limit: 1 hour
#SBATCH -J snakemake-test                   # job name
#SBATCH -o snakemake-test-%A.out            # output file
#SBATCH -e snakemake-test-%A.err            # error file

module purge  # good practice to purge all modules
module load miniforge # to run conda
module load bwa/0.7.17 # these modules should already be in your environmant
module load cutadapt/4.9
module load snakemake/9.8.1
module load freebayes/1.3.10
module load samtools/1.21

cd /scratch/user-name
conda activate snakemake_env #if you have created a conda environment
snakemake -c 1 -s variant.smk --config config_variant.yml
```

Smakemake and containers

Snakemake also supports containers:

```
rule align:
     container:
         "docker://biocontainers/bwa"
```

Advantages:

- identical software environments
- portable across HPC systems
- easier collaboration

Best Practices for HPC

Recommendations:

Use threads and resources properly
Avoid huge single jobs
Break workflows into modular rules
Use conda or containers
Use `--dry-run` before submitting large workflows
Store configuration in YAML files

Common HPC Pitfalls with workflow managers

Examples:

- requesting too many cores per rule
- forgetting to specify memory
- submitting thousands of tiny jobs
- running Snakemake or Nextflow themselves on a login node

Key Takeaways with workflow managers

Snakemake & Nextflow provide:

- reproducible pipelines
- automatic dependency tracking
- scalable HPC execution
- environment management
- workflow portability

Nextflow

Snakemake & Nextflow provide:

- reproducible pipelines
- automatic dependency tracking
- scalable HPC execution
- environment management
- workflow portability

What is Nextflow?

Nextflow is a workflow management system that helps automate and organize multi-step computational pipelines.

At a high level, it connects software steps together, manages how data moves between them, and handles execution across local machines, HPC schedulers like SLURM, or cloud platforms.

Nextflow Pipelines

Key concepts:

- Processes, workflows, and parameters

In general, we are going to:

- Create processes to execute desired commands
- Specify parameters to represent workflow settings
- Define a workflow to execute processes in a specific order

Key files:

- main.nf and nextflow.config

Toy example: print the text "Hello World!"

First, create a process called HELLO with our shell command:

```
process HELLO {
   script:
   """
   echo "Hello World!"
   """
}
```

Then we execute this process in our workflow:

```
workflow {
   HELLO()
}
```

Create a new file called main.nf

```
process HELLO {
    script:
    """
    echo "Hello World!"
    """
}

workflow {
    HELLO()
}
```

Let's make some changes

```
process hello {
 output:
 path 'hello.txt'

 script:
 """
 echo 'Hello world!' > hello.txt
 """
}
```

Add a publishDir for output file destination

```
process hello {
 publishDir "results/" , mode: "copy"
 output:
 path 'hello.txt'

 script:
 """
 echo 'Hello world!' > hello.txt
 """
}
```

Let's look at our snakemake "trim" rule from earlier

```
rule trim:
    input:
    ”reads/sample1.fastq”

    output:
    ”trimmed_reads/sample1-trimmed.fastq”

    shell:
    cutadapt -A TCCGGGTS -o {output} {input}
```

What do we need to update in Nextflow?

```
process HELLO {
    publishDir "results/" , mode: "copy"

    output:
    path 'hello.txt'

    script:
    """
    echo 'Hello world!' > hello.txt
    """
}
```

Update for running cutadapt

```
process CUTADAPT {

    publishDir "results/" , mode: "copy"
    output:
    path 'trimmed.fastq'

    script:
    """
    cutadapt -a AACCGGTT -o trimmed.fastq ~/sample1.fastq
    """
}
workflow {
CUTADAPT()
}
```

More common approach for input files

```
process CUTADAPT {

    publishDir "results/" , mode: "copy"
    input:
    path reads_var
    output:
    path 'trimmed.fastq'

    script:
    """
    cutadapt -a AACCGGTT -o trimmed.fastq $reads_var
    """
}
workflow {
CUTADAPT(Channel.fromPath('~/sample1.fastq', checkIfExists: true))
}
```

Dynamically scaling to many samples

```
process CUTADAPT {

 publishDir "results/", mode: "copy"

 input:
 path reads_var

 output:
 path "${reads_var.simpleName}_trimmed.fastq"

 script:
 """
 cutadapt -a AACCGGTT -o ${reads_var.simpleName}_trimmed.fastq $reads_var
 """
}

workflow {
 CUTADAPT(Channel.fromPath('*.fastq', checkIfExists: true))
}
```

Parameter options for input files

- Add a parameter for '--reads' in your 'nextflow run' command
- Add a params.reads at the top of your main.nf file
- Add a params.reads to a nextflow.config file
- Works for one file ('reads/sample1.fastq') or many ('reads/*.fastq')

Less hard-coding = more reproducibility

From:

```
  workflow {
   CUTADAPT(Channel.fromPath(~/sample1.fastq', checkIfExists: true))
  }
```

To:

```
workflow {
   CUTADAPT(Channel.fromPath(params.reads, checkIfExists: true))
  }
```

Loading software – main.nf

Use a 'beforeScript' in the CUTADAPT process in main.nf

- beforeScript runs specified shell command(s) before running the script command
- Load the cutadapt module: beforeScript 'module load cutadapt'
- Can also do other things like export variables or create directories

```
beforeScript """
   module purge
   module load cutadapt
   mkdir results
   export PATH="$PATH:/opt/tools"
"""
```

Loading software – nextflow.config

Again, we use a 'beforeScript' specific to the CUTADAPT process

```
Process {
  withName: CUTADAPT {
     beforeScript = '''
     module purge
     module load cutadapt
      '''
```

Adding SLURM options – nextflow.config

```
Process {
   withName: CUTADAPT {
      beforeScript = '''
      module purge
      module load cutadapt
      '''

   executor = 'slurm'
   queue = 'standard'
   cpus = 2
   mem = '16 GB'
   time = '1h'
   clusterOptions = '--account=hpc_build'
}
```

Now we have:

- Workflow logic in main.nf
- Software and slurm options in nextflow.config

Extend to CUTADAPT → BWA_ALIGN → FREEBAYES

- Same rules apply – largely rinse and repeat for additional processes
- Create processes for each step: inputs/outputs, commands, etc.
- Software and slurm options in nextflow.config
- Main difference is our workflow - more processes and channels
  - Send channel into process
  - Process produces output
  - Output becomes new channel for next process.

Workflow for CUTADAPT → BWA_ALIGN → FREEBAYES

```
workflow {
    reads_ch = Channel.fromPath("${params.reads_dir}/*.fastq", 
   checkIfExists:  true)
    trimmed_ch = CUTADAPT(reads_ch)
    aligned_ch = BWA_ALIGN(trimmed_ch)
    FREEBAYES(aligned_ch)
}
```

Additional links

- https://nf-co.re/rnaseq/3.23.0/
- https://training.nextflow.io
- https://github.com/nextflow-io/nextflow

Workflows for computational data analysis

- https://github.com/common-workflow-language/common-workflow-language/wiki/Existing-Workflow-systems

- https://github.com/pditommaso/awesome-pipeline

- Galaxy platform - bioinformatic software, pipeline and workflows:
  https://usegalaxy.org