

# Reproducible bioinformatics

from a user's perspective

The University of Otago

[tom.harrop@otago.ac.nz](mailto:tom.harrop@otago.ac.nz)

[@tharrop\\_](#)

2020-02-12

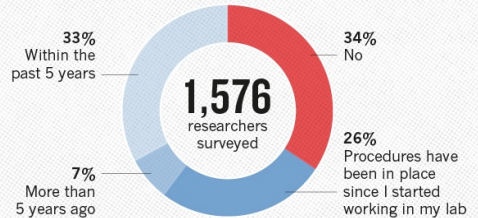
### IS THERE A REPRODUCIBILITY CRISIS?



©nature

### HAVE YOU ESTABLISHED PROCEDURES FOR REPRODUCIBILITY?

Among the most popular strategies was having different lab members redo experiments.



©nature

# What is reproducibility?

**Reproduce:** under identical conditions to the previous result, repeat the analysis and get the **exact** same result

In bioinformatics:

- **same data**
- **same methodology** (code)
- **same result**

Guidelines for reproducible analysis:

1. Don't modify raw data
2. Record the code
3. Capture the computing environment

# 1. Take care peeking at the data

Ziemann *et al. Genome Biology* (2016) 17:177  
DOI 10.1186/s13059-016-1044-7

Genome Biology

COMMENT

Open Access



## Gene name errors are widespread in the scientific literature

Mark Ziemann<sup>1</sup>, Yotam Eren<sup>1,2</sup> and Assam El-Osta<sup>1,3\*</sup>


### Abstract

The spreadsheet software **Microsoft Excel** when used with default settings, is known to convert gene names to dates and floating-point numbers. A programmatic scan of leading genomics journals reveals that approximately one-fifth of papers with supplementary Excel gene lists contain erroneous gene name conversions.

**Keywords:** Microsoft Excel, Gene symbol, Supplementary data

**Abbreviations:** GEO, Gene Expression Omnibus; JIF, journal impact factor

# Point-and-click analysis may be hard to reproduce

 U.S. National Library of Medicine

NCBI National Center for Biotechnology Information

Sign in to NCBI

BLAST® » blastn suite


HomeRecent ResultsSaved StrategiesHelp

Standard Nucleotide BLAST


blastnblastblastxtblastntblastx

BLASTN programs search nucleotide databases using a nucleotide query. [more...](#)[Reset page](#)[Bookmark](#)

Enter Query Sequence

Enter accession number(s), gi(s), or FASTA sequence(s) 


Clear


Query subrange 


From To

Or, upload file

Job Title


Browse... No file selected. 

Enter a descriptive title for your BLAST search 

☐ Align two or more sequences 


Choose Search Set


Database

☐ Human genomic + transcript ☐ Mouse genomic + transcript ☒ Others (nr etc.):  
Nucleotide collection (nr/nt) 

Organism

Optional

Enter organism name or id—completions will be suggested ☐ Exclude 

Enter organism common name, binomial, or tax id. Only 20 top taxa will be shown 

Exclude

Optional

☐ Models (XM/XP) ☐ Uncultured/environmental sample sequences


Limit to


Optional

☐ Sequences from type material

Entrez Query

Optional


You  [Create custom database](#)

Enter an Entrez query to limit search 

Program Selection

Optimize for

☒ Highly similar sequences (megablast)  
☐ More dissimilar sequences (discontiguous megablast)  
☐ Somewhat similar sequences (blastn)

Choose a BLAST algorithm 

# *Ad hoc* analysis may be hard to reproduce

## Examples:

- install software locally
- use software installed by the admin
- paste your commands directly into the console and hit enter
- save a set of scripts to run in order

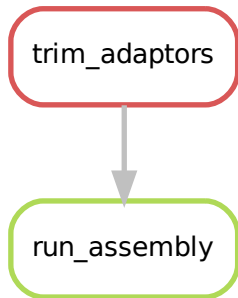
## Possible issues:

- will it run again?
- are all the steps documented?
- is the recorded code exactly what was run?
- are the steps in the right order?

## 2. Workflow managers force you to record every step

Define my\_workflow:

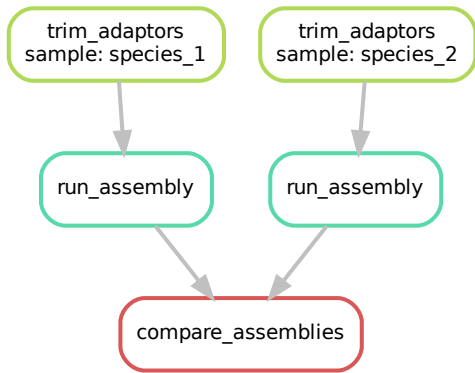
```
step trim_adaptors:  
  input:  'data/raw_reads/{sample}.fastq',  
  output: 'output/trimmed/{sample}.fastq'  
  shell:  'trim_adaptors --raw_reads={input} > {outp  
  
step run_assembly:  
  input:  'output/trimmed/{sample}.fastq'  
  output: 'output/assemblies/{sample}.fasta'  
  shell:  'choice_assembler --reads={input} > {outp
```



Run:

```
workflow_manager my_workflow run_assembly
```

## 2. Reproducibility and convenience



- The code *is* the documentation
- Scale the same code to different data
- Version control → versioned results

**Lots of good options:**

**snakemake** ← python3

**nextflow** ← java

**CWL** ← 'vendor-neutral  
specification'

**drake** ← R

**make** ← DIY



### 3. Reproducible computing environment

Software has

- a **version**,
- other software **dependencies** (with versions)
- all with **system dependencies**

*e.g.* DESeq2

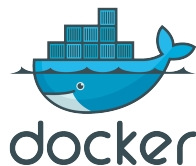
DESeq2\_1.26.0

Bioconductor 3.10.1

libblas3 3.8.0, libc6 2.30, etc.

### 3. Software containers

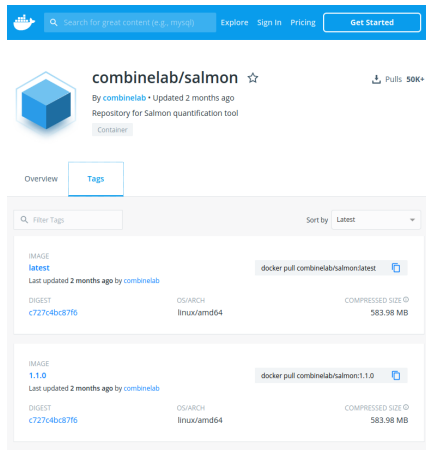
- Isolated, complete environment (a mini OS)
- Contain specific version of software with dependencies
- Mobility of compute
- Reproducibility
- Singularity can run on traditional HPC



### 3. Getting software in containers

- Some developers provide docker containers

```
singularity pull \  
  --name salmon_1.1.0.sif \  
  docker://combinelab/salmon:1.1.0
```



The screenshot shows the Docker Hub interface for the `combinelab/salmon` image. The header includes a search bar, navigation links (Explore, Sign In, Pricing), and a 'Get Started' button. The main section displays the image name `combinelab/salmon` with a star icon, the maintainer `combinelab`, and the description 'Repository for Salmon quantification tool'. Below this, there are tabs for 'Overview' and 'Tags'. The 'Tags' tab is active, showing a list of image tags. The first tag is `latest`, which was last updated 2 months ago. It has a digest of `c727c4bc87f6` and a compressed size of 583.98 MB. The second tag is `1.1.0`, also last updated 2 months ago, with the same digest and size. Both tags are for the `linux/amd64` architecture. The interface includes a 'Filter Tags' search bar and a 'Sort by' dropdown set to 'Latest'. A 'docker pull' command is provided for each tag.

IMAGE	OS/ARCH	COMPRESSED SIZE
<code>latest</code> Last updated 2 months ago by <code>combinelab</code> DIGEST: <code>c727c4bc87f6</code>	<code>linux/amd64</code>	583.98 MB
<code>1.1.0</code> Last updated 2 months ago by <code>combinelab</code> DIGEST: <code>c727c4bc87f6</code>	<code>linux/amd64</code>	583.98 MB

### 3. Getting software into containers

- Often have to build our own containers

**Singularity.bwa\_0.7.17**

```
Bootstrap: docker
From: ubuntu:18.10
```

```
%labels
    VERSION "BWA 0.7.17"

%post
    apt-get update
    apt-get install -y bwa

%runscript
    exec /usr/bin/bwa "$@"
```

The screenshot shows the SingularityHub interface for a container named 'TomHarrop/singularity-containers:bwa\_0.7.17'. The container's build specification is displayed in a table with line numbers 1 through 25. The build process includes installing dependencies via apt and running the bwa command. The right sidebar shows the container's description (None), build metrics (Size: 101 MB, Tag: bwa\_0.7.17), and commit information (Commit: 3c62cd582cafcac61867e6d8361bb413d67e6f37, Branch: master, Version (file hash): eb49586488ed8fa5ba71da34f2fabc1a, Build Time: 40 seconds, Version 2.5.0-feature-squashbuild-secbuild-2.5.0.gdd162fb, Version 2.5).

Line	Build Spec
1	Bootstrap: docker
2	From: ubuntu:18.10
3	
4	%help
5	Container for BWA 0.7.17
6	http://bio-bwa.sourceforge.net/
7	
8	%labels
9	
10	VERSION "BWA 0.7.17"
11	
12	%post
13	
14	# install dependencies via apt
15	apt update
16	apt install -y \
17	bwa
18	
19	%runscript
20	
21	exec /usr/bin/bwa "\$@"
22	
23	
24	
25	

**Description**  
None

**Build Metrics**  
Size (MB): 101  
Tag: bwa\_0.7.17  
Commit: 3c62cd582cafcac61867e6d8361bb413d67e6f37  
Branch: master  
Version (file hash): eb49586488ed8fa5ba71da34f2fabc1a  
Build Time: 40 seconds  
Version 2.5.0-feature-squashbuild-secbuild-2.5.0.gdd162fb  
Version 2.5

## 2 & 3. Workflow managers support containers

```
rule trim_adaptors:
    input:          'data/raw_reads/{sample}.fastq',
    output:         'output/trimmed/{sample}.fastq'
    singularity:    'docker://my_repos/trim_adaptors:2.9'
    shell:          'trim_adaptors --raw_reads={input} > {output}'

rule run_assembly:
    input:          'output/trimmed/{sample}.fastq'
    output:         'output/assemblies/{sample}.fasta'
    singularity:    'shub://my_repos/choice_assembler:1.5'
    shell:          'choice_assembler --reads={input} > {output}'
```

### 3. Some barriers to container usage

- **Building containers can be painful** if the dependencies are disorganised
- **Duplication of effort**
- **Some software shouldn't go in a container** because of “unfortunate licensing issues”
  - DTU software *e.g.* rnammer, tmhmm
  - GATech: GeneMark
  - GIRInst's RepBase
- **Getting Singularity installed**

# Reproducible analysis stack

## Guidelines:

1. Don't modify raw data
2. Record the code  
(with version control)
3. Capture the computing environment

## Stack:

md5sum raw\_reads.fastq? chmod 444?  
+ Workflow manager (snakemake, nextflow)  
+ VCS (git)  
+ Software containers (Singularity)

Tell Snakemake what files  
you want to be created

Produce the files  
you want to have from  
some intermediate  
result

Create a needed  
intermediate result

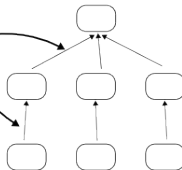
```
rule:  
  input: "A.txt", "B.txt", "C.txt"
```

```
rule:  
  input: "{sample}.inter"  
  output: "{sample}.txt"  
  shell: "somecommand {input} {output}"
```

```
rule:  
  input: "{sample}.in"  
  output: "{sample}.inter"  
  run:  
    somepythoncode()
```

Snakemake determines  
the dependencies  
for you

Use wildcards to write  
general rules  
for all samples



# Getting started

## Reproducibility for bioinformatics:

- Joep de Ligt: *Scalable workflows and reproducible data analysis for genomics*
- plenty of online talks e.g. [Adam Labadorf](#) of Boston Uni

## Workflow managers:

- [Snakemake Tutorial](#)
- Nextflow: [Get started](#)

## Software containers:

- Blair Bethwaite: *Containers in HPC* Tutorial
- Singularity [Quick Start](#)

## Version control:

- memorise a handful of `git` commands