

Reproducible bioinformatics

from a user's perspective

Tom Harrop

The University of Otago

tom.harrop@otago.ac.nz

[@tharrop_](#)

2020-02-12

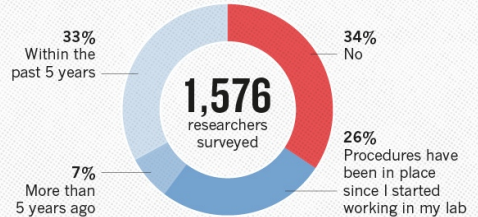
IS THERE A REPRODUCIBILITY CRISIS?



©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 Ziemann¹, Yotam Eren^{1,2} and Assam El-Osta^{1,3*}

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

2. Point-and-click software is less likely to be reproducible

NIH U.S. National Library of Medicine NCB National Center for Biotechnology Information Sign in to NCBI

BLAST » blastn suite Home Recent Results Saved Strategies Help

Standard Nucleotide BLAST

blastn blastp blastx tblastn tblastx

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 No file selected. [?](#)

Job Title


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 tube](#) [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 [?](#)

2. Running on-the-fly probably won't be reproducible

Examples:

- install software locally
- use software installed by the admin
- type 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 code you recorded the same as the code you ran?
- did you correctly record the order of steps?

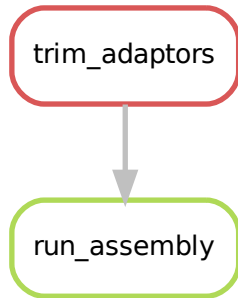
2. Workflow managers force you to record every step

Define:

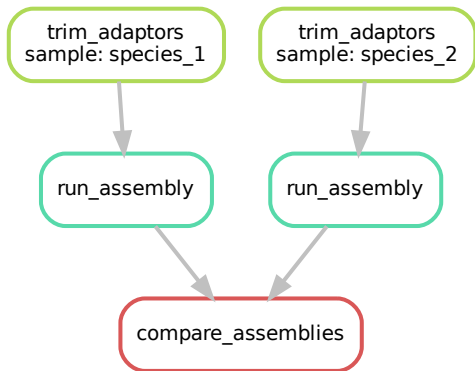
```
rule trim_adaptors:  
    input:  'data/raw_reads/{sample}.fastq',  
    output: 'output/trimmed/{sample}.fastq'  
    shell:  'trim_adaptors --raw_reads={input} > {output}'  
  
rule run_assembly:  
    input:  'output/trimmed/{sample}.fastq'  
    output: 'output/assemblies/{sample}.fasta'  
 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

Run:

```
workflow_manager 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. Reproducible computing environment

On our department's hardware:

```
salmon --version
```

```
salmon 0.9.1
```

e.g. Ubuntu 19.10:

```
apt policy salmon
```

```
salmon:
```

```
  Installed: (none)
```

```
  Candidate: 0.12.0+ds1-1
```

```
  Version table:
```

```
    0.12.0+ds1-1 500
```

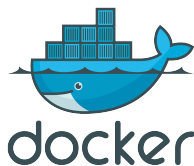
```
    500 http://nz.archive.ubuntu.com/ubuntu eoan/universe amd64 Packages
```

3. Software containers

- Isolated, complete environment (a mini OS)
- Contain specific version of software with dependencies

Singularity:

- Mobility of compute
- Reproducibility
- Support on existing traditional HPC



3. Singularity containers

Running directly:

```
salmon --help
```

Error in running command bash

Running with Singularity:

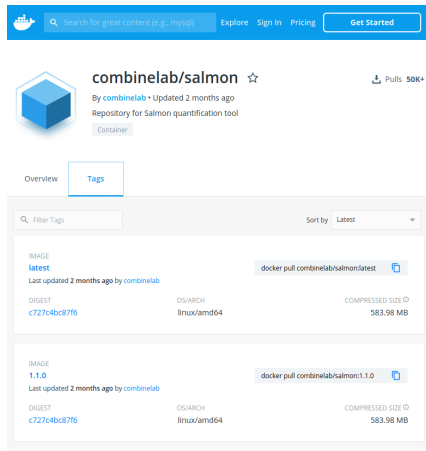
```
singularity exec \  
    salmon_1.1.0.sif \  
    salmon --help
```

Usage: salmon -h|--help or
salmon -v|--version or
salmon -c|--cite or
salmon [--no-version-check] <COMMAND> [-h | options]

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` repository. The repository is a container image created by `combinelab`, updated 2 months ago, and serves as a repository for the Salmon quantification tool. It has over 50K pulls. The 'Tags' tab is selected, showing a list of image tags. The 'latest' tag is the most recent, last updated 2 months ago. Below it, the '1.1.0' tag is also shown, also last updated 2 months ago. Both tags have a digest of `c727c4bc87f6` and are for the `linux/amd64` architecture, with a compressed size of 583.98 MB. The interface includes a search bar, navigation links (Overview, Tags), and a 'docker pull' command for each tag.

IMAGE	OS/ARCH	COMPRESSED SIZE
latest Last updated 2 months ago by combinelab	linux/amd64	583.98 MB
1.1.0 Last updated 2 months ago by combinelab	linux/amd64	583.98 MB

3. Getting software into containers

- Usually have to build it yourself

Singularity.bwa_0.7.17

Bootstrap: docker

From: ubuntu:18.10

%labels

VERSION "BWA 0.7.17"

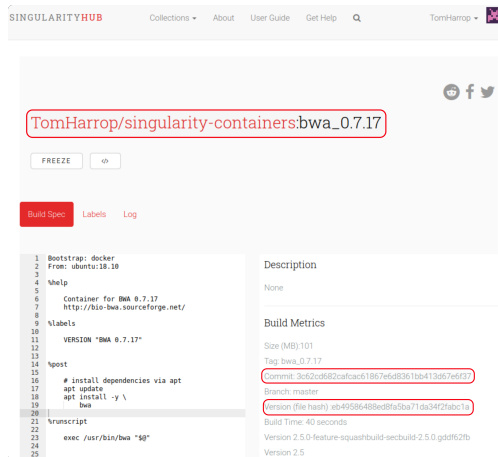
%post

apt-get update

apt-get install -y bwa

%runscript

exec /usr/bin/bwa "\$@"



SINGULARITYHUB Collections About User Guide Get Help TomHarrop

TomHarrop/singularity-containers:bwa_0.7.17

FREEZE

Build Spec Labels Log

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

Description
None

Build Metrics
Size (MB): 101
Tag: bwa_0.7.17
Commit: 3c62cd582cafcac61867ef6d9361bb413d67ef37
Branch: master
Version (file hash): eb49586488ed8fa5ba71da34f2fab01a
Build Time: 40 seconds
Version 2.5.0-feature-squashbuild-secbuild-2.5.0.gddfd62fb
Version 2.5

3. Some barriers to container usage

- **Building containers can be painful** if the dependencies are disorganised
- **Duplication of effort**
- **Some software can't 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**

2 & 3. Workflow managers support containers and clusters

```
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}'
```

Cluster execution, e.g.:

```
snakemake --drmaa " -q username" -j 32
```


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

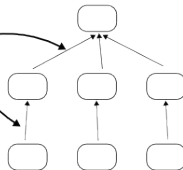
```
rule:  
  input: "A.txt", "B.txt", "C.txt"
```

```
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:

- online lectures e.g. [Adam Labadorf](#) of Boston Uni

Workflow managers:

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

Software containers:

- Singularity [Quick Start](#)

Version control:

- memorise a handful of `git` commands