Reproducible bioinformatics

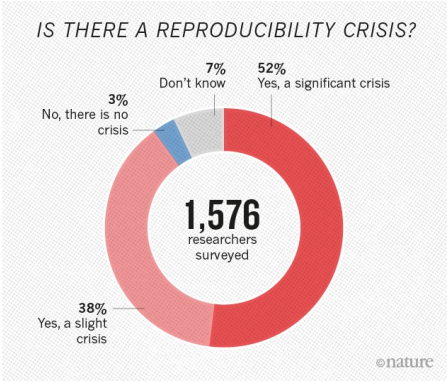
from a user's perspective

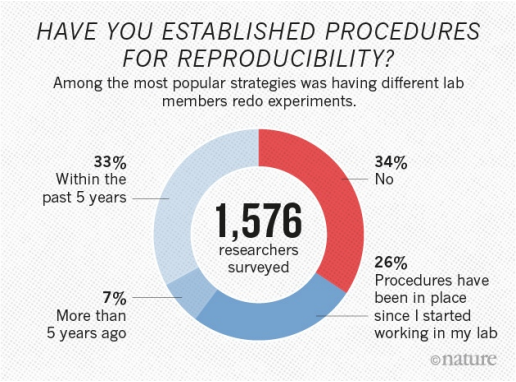
The University of Otago

tom.harrop@otago.ac.nz

@tharrop_

2020-02-12





Jenny Bryan https://youtu.be/7oyiPBjLAWY & Nature News, 10.1038/533452a

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

Adapted from Adam Labadorf's slides

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[1], Yotam Eren[1,2] and Assam El-Osta[1,3*]

Abstract

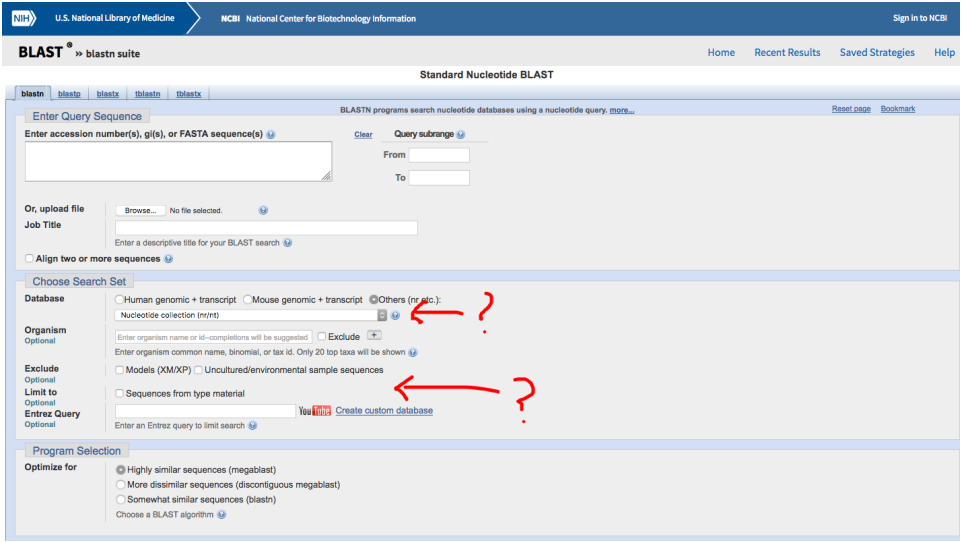
The spreadsheet software Microsoft Excel when used with default settings, is known to convert gene names to dates and floating-point numbers. A programmatic scan of leading genomics journals reveals that approximately one-fifth of papers with supplementary Excel gene lists contain erroneous gene name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

Keywords: Microsoft Excel, Gene symbol, Supplementary data

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

Ziemann *et al.*, 2016. 10.1186/s13059-016-1044-7

Point-and-click analysis may be hard to reproduce



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

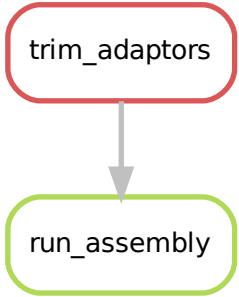
Possible issues:

- 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`:

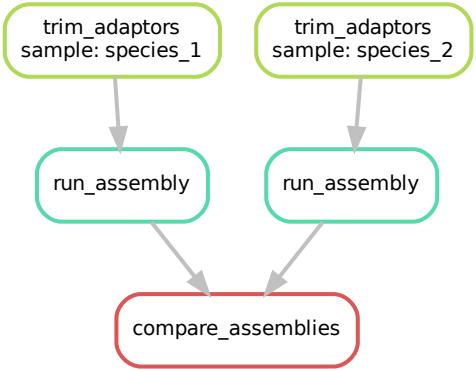
```
step trim_adaptors:
    input:  'data/raw_reads/{sample}.fastq',
    output: 'output/trimmed/{sample}.fastq'
    shell:  'trim_adaptors --raw_reads={input} > {out

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.*

Adapted from Adam Labadorf's slides

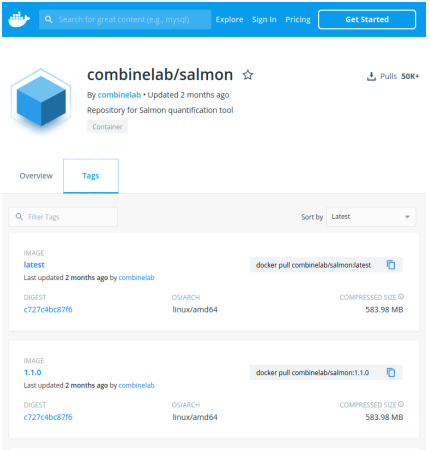
3. Software containers

- 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
```



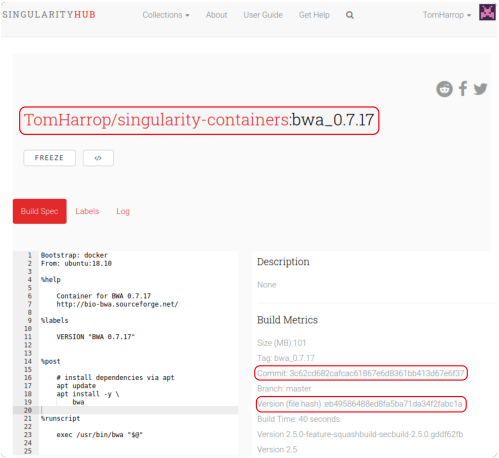
hub.docker.com

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 "$@"
```



singularity-hub.org

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**

See the DTU's license here

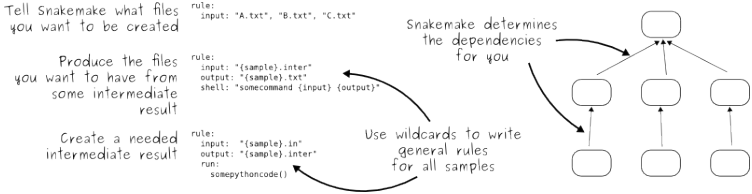
Reproducible analysis stack

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`)



snakemake.readthedocs.io

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

These slides: https://github.com/TomHarrop/eresearch2020