



Practical Tools for Improving Reproducibility (of Bioinformatics)

Johann Dreo

► To cite this version:

Johann Dreo. Practical Tools for Improving Reproducibility (of Bioinformatics). Doctoral. Precision Medicine at the Interface of Translational Research and Systems Medicine, Paris, France. 2021, pp.37. hal-04127255

HAL Id: hal-04127255

<https://hal.science/hal-04127255>


Submitted on 13 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - ShareAlike 4.0 International License



Practical Tools for Improving Reproducibility (of Bioinformatics)

• Johann Dreo • 25/11/2021

Abstract

Experimental studies are prevalent in (Bio)informatics, as in many fields involving computer science.

While reproducibility is crucial for science, it is difficult and often overlooked. This presentation introduces a curated list of tools that can help improving reproducibility for experimental computer science.

It may incidentally insists on testing...

Abstract

Experimental studies are prevalent in (Bio)informatics, as in many fields involving computer science.

While reproducibility is crucial for science, it is difficult and often overlooked.

This presentation introduces a curated list of tools that can help improving reproducibility for experimental computer science.

It may incidentally insists on testing...

Summary

01

—

Why Reproducibility?

02

—

Tools that helps taking
practical actions

03

—

Conclusion

04

—

References

Part 1

Why Reproducibility?



A little bit of Epistemology

The *Reproducibility Crisis*

The scientific method:

- Falsifiability by *experimental* evidence.
- Experimental findings hold under *similar conditions*.
- Making for *predictable* outcomes.
- That can be used to build up *cumulative science*.

A little bit of Epistemology

The problem

Anecdotal evidences:

- More than 70% of researchers have failed in an attempt to reproduce an experiment [Baker 2016].
- Over 50% have even failed to reproduce *THEIR OWN* previous work.

A little bit of Epistemology

Some Terminology

Repeatability: same team, same experiment.

Reproducibility: different team, same experiment.

Replicability: different team, different experiment.

Generalisability: [López-Ibáñez *et al.* 2021]

Possible Actions

Cultural Actions

Researchers in permanent positions, please, think of:

- Incentivize negative results publishing (e.g. reproducibility failure).
- Do not oversell positive results.
- Fight bibliometry optimization.
- Better statistics (avoid p-hacking).
- Refrain from hypothesising after results (HARKing).
- Advertise TESTING.

Possible Actions

Technical Actions YOU can take

In a nutshell: *automate everything.*

- Keep track of Artifacts:
 - Version Control.
 - Lab Notebooks.
 - Archive Code and Data.
 - Save tests.
- Good Engineering:
 - Modular Code.
 - Test, tests, TESTS.
 - Automate Everything.
- Help your future self:
 - Containers.
 - Pipeline Automation.
 - Design of Experiment Automation.
 - Allow Reuse.
 - Publish Code and Data.
 - Automate tests.

Possible Actions

Technical Actions YOU can take

In a nutshell: *automate everything.*

- Keep track of Artifacts:
 - Version Control.
 - Lab Notebooks.
 - Archive Code and Data.
 - Save tests.
- Good Engineering:
 - Modular Code.
 - Test, tests, TESTS.
 - Automate Everything.
- Help your future self:
 - Containers.
 - Pipeline Automation.
 - Design of Experiment Automation.
 - Allow Reuse.
 - Publish Code and Data.
 - Automate tests.

Part 2

Tools that helps taking practical actions



About This Presentation

Assumptions:

- You know how to read and search documentation.
- Difficulty is to know that a tool exists.
- Everybody likes Python.

This presentation:

- Curated list of tools.
- Insists on TESTING.
- Alternatives and more info in the "see also" section.

Version Control

Git: <https://git-scm.com>

Allows to:

- Keep track of changes you make to your code.
- Go back in time, which helps debugging.
- Work with others on the same codebase.
- Works offline.

Basic idea:

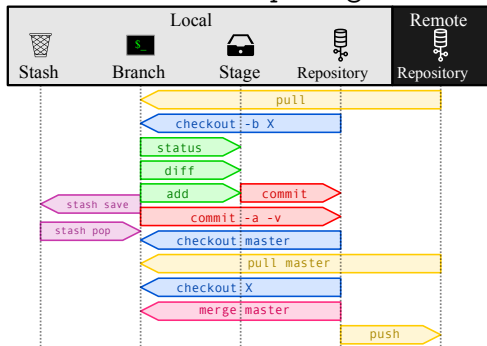
- All the source code is versioned (even tests).
- Note: you only version SOURCES, not generated files or data.
- You *commit* modifications to your code.
- You *merge* commits in the last version.

Version Control

Git: <https://git-scm.com>

See also:

- Shortest tutorial: <https://github.com/nojhan/gitcrux>



- Gitk: <https://www.atlassian.com/git/tutorials/gitk>
- Liquidprompt: <https://github.com/nojhan/liquidprompt>
- Mercurial: <https://www.mercurial-scm.org>

Version Control

Gitlab: <https://about.gitlab.com>

Online service:

- Online backups of Git repositories (see push & pull commands).
- Issue tracker (e.g. test failure report).
- Merge requests from others (e.g. more tests).
- Basic public page hosting.

See also:

- Tutorial:
https://docs.gitlab.com/ee/topics/plan_and_track.html
- Github: <https://github.com>
- DevOps (= automate tests): <https://en.wikipedia.org/wiki/DevOps>

Version Control

Gitlab: <https://about.gitlab.com>

Online service:

- Online backups of Git repositories (see push & pull commands).
- Issue tracker (e.g. test failure report).
- Merge requests from others (e.g. more tests).
- Basic public page hosting.

See also:

- Tutorial:
https://docs.gitlab.com/ee/topics/plan_and_track.html
- Github: <https://github.com>
- DevOps (= automate tests): <https://en.wikipedia.org/wiki/DevOps>

Notebooks

Jupyter: <https://jupyter.org/>

Interactive page:

- Sequence of paragraphs, executable code, figures.
- Keep track of your ongoing pipeline/experiment.
- Easy to share with others (as a single archive or an online page).
- Perfect for tutorials or demo (tests are good candidates).

See also:

- Online demo: <https://jupyter.org/try>
- Tutorial · s: <https://jupyter.readthedocs.io/en/latest/content-quickstart.html>
- Serve notebooks: <https://jupyter.org/hub>

Notebooks

Jupyter: <https://jupyter.org/>

Interactive page:

- Sequence of paragraphs, executable code, figures.
- Keep track of your ongoing pipeline/experiment.
- Easy to share with others (as a single archive or an online page).
- Perfect for tutorials or demo (tests are good candidates).

See also:

- Online demo: <https://jupyter.org/try>
- Tutorial · s: <https://jupyter.readthedocs.io/en/latest/content-quickstart.html>
- Serve notebooks: <https://jupyter.org/hub>

Modern Engineering

Tests

Golden rule of software engineering:

1. Modular code (= functions).
2. Tests = first lines of defense:
 - 2.1 Asserts on pre and post-conditions.
 - 2.2 Asserts everywhere in the code.
 - 2.3 Unit tests (= stand-alone test of functions, on simple data).
 - 2.4 Integration tests (= with real data).
3. When you finish a feature: RUN TESTS.

Modern Engineering

Test!

```
def f(arg):
    assert(0 < len(arg) <= 100) # Pre-condition.
    assert(all(0 < i <= 10 for i in arg))
    inter = [float(i) for i in arg]
    assert(len(inter) == len(arg)) # Intermediate.
    result = sum(inter)
    assert(0 < result <= len(arg)*10) # Post-condition.
    return result

if __name__ == '__main__':
    assert( f(range(10)) == 45 ) # Basic unit test.
    if __debug__: # Those are declutchable when running python -O
        failed = False
        try:
            res = f([0]*10)
        except AssertionError:
            failed = True
    assert(failed)
```

Modern Engineering

TEST!

Tests helps:

- Avoiding bugs.
- Finding bugs.
- Fixing bugs.
- On the long term.
- Actually coding faster.
- Double-checking intermediate data of your pipeline.

See also:

- R's "stopifnot()" or packages named "assert".
- Python's unittest:
<https://docs.python.org/3/library/unittest.html>
- PyTest Framework: <https://pypi.org/project/pytest>

Modern Engineering

TEST!

Tests helps:

- Avoiding bugs.
- Finding bugs.
- Fixing bugs.
- On the long term.
- Actually coding faster.
- Double-checking intermediate data of your pipeline.

See also:

- R's "stopifnot()" or packages named "assert".
- Python's unittest:
<https://docs.python.org/3/library/unittest.html>
- PyTest Framework: <https://pypi.org/project/pytest>

Modern Engineering

Continuous Integration

Automate rule #3:

- Every time you push on Gitlab, tests are ran automatically.
- And someone (you) get blamed for pushing bugs.

See also:

- Documentation: <https://docs.gitlab.com/ee/ci/>

Modern Engineering

Continuous Integration

Automate rule #3:

- Every time you push on Gitlab, tests are ran automatically.
- And someone (you) get blamed for pushing bugs.

See also:

- Documentation: <https://docs.gitlab.com/ee/ci/>

Virtualization

Objectives:

- Ease the not-built-here affliction.
- Get rid of the dependencies nightmare.
- Avoid the backward incompatibility plague.

Historically:

- Virtual Machines,
- Package Managers & Virtual Environments,
- Containers.

Virtualization

Objectives:

- Ease the not-built-here affliction.
- Get rid of the dependencies nightmare.
- Avoid the backward incompatibility plague.

Historically:

- Virtual Machines,
- Package Managers & Virtual Environments,
- Containers.

Virtualization

VirtualBox: <https://www.virtualbox.org>

Virtual Machine:

- Replace the computer with software.
- Run any OS on any other OS.
- (A bit) slower.
- Takes a lot of space (all the OS is installed).

See also:

- VMware: <https://www.vmware.com/>

Virtualization

VirtualBox: <https://www.virtualbox.org>

Virtual Machine:

- Replace the computer with software.
- Run any OS on any other OS.
- (A bit) slower.
- Takes a lot of space (all the OS is installed).

See also:

- VMware: <https://www.vmware.com/>

Virtualization

Conda: <https://conda.io>

Package manager + virtual environments:

- Install (several) softwares with (several) specific versions.
- Environment = dependencies that just works.
- Run several environments on your machine.
- Distribute environments configurations.
- Historically for Python, but works for R (and other languages).

See also:

- Bioconda: <https://bioconda.github.io>
- Miniconda: <https://docs.conda.io/en/latest/miniconda.html>

Virtualization

Conda: <https://conda.io>

Package manager + virtual environments:

- Install (several) softwares with (several) specific versions.
- Environment = dependencies that just works.
- Run several environments on your machine.
- Distribute environments configurations.
- Historically for Python, but works for R (and other languages).

See also:

- Bioconda: <https://bioconda.github.io>
- Miniconda: <https://docs.conda.io/en/latest/miniconda.html>

Virtualization

Singularity: <https://sylabs.io/singularity/>

Containers:

- Virtual OS.
- Run any software on any OS.
- Faster than VM, take less space.
- More stable than virtual environments.
- Install a container on your computer, run it on a cluster.
- (Do not forget to run tests)

See also:

- Docker: <https://www.docker.com>
- Docker hub: <https://hub.docker.com>
- docker2singularity:
<https://github.com/singularityhub/docker2singularity>

Virtualization

Singularity: <https://sylabs.io/singularity/>

Containers:

- Virtual OS.
- Run any software on any OS.
- Faster than VM, take less space.
- More stable than virtual environments.
- Install a container on your computer, run it on a cluster.
- (Do not forget to run tests)

See also:

- Docker: <https://www.docker.com>
- Docker hub: <https://hub.docker.com>
- docker2singularity:
<https://github.com/singularityhub/docker2singularity>

Pipeline Management

SnakeMake <https://snakemake.readthedocs.io>

Pipeline Automation:

- Like a (Python) script calling every steps of your pipeline.
- States which step's input depends on which step's output.
- Automates:
 - parallelization,
 - job submission (e.g. on a cluster),
 - Intermediate step tests.
- Works well with Conda and Singularity.

See also:

- Nextflow: <https://www.nextflow.io>

Pipeline Management

SnakeMake <https://snakemake.readthedocs.io>

Pipeline Automation:

- Like a (Python) script calling every steps of your pipeline.
- States which step's input depends on which step's output.
- Automates:
 - parallelization,
 - job submission (e.g. on a cluster),
 - Intermediate step tests.
- Works well with Conda and Singularity.

See also:

- Nextflow: <https://www.nextflow.io>

Design of Experiment Management

openMOLE: <https://openmole.org>

DoE:

- Explore the parameters space of your model.
- Automates:
 - sampling (understand the response function),
 - calibration (optimize parameters to match reality),
 - sensitivity analysis,
 - job submission.

See also:

- modeFrontier: <https://engineering.esteco.com/modefrontier>
- ModelCenter:
<https://www.phoenix-int.com/product/modelcenter-integrate/>

Design of Experiment Management

openMOLE: <https://openmole.org>

DoE:

- Explore the parameters space of your model.
- Automates:
 - sampling (understand the response function),
 - calibration (optimize parameters to match reality),
 - sensitivity analysis,
 - job submission.

See also:

- modeFrontier: <https://engineering.esteco.com/modefrontier>
- ModelCenter:
<https://www.phoenix-int.com/product/modelcenter-integrate/>

Metadata

CookieCutter: <https://cookiecutter.readthedocs.io>

Projects Templates:

- Create project directories and base files.
- Allows to follow best practices.
- Avoids forgetting important features.
- Example: cookiecutter
`gh:snakemake-workflows/cookiecutter-snakemake-workflow`

See also:

- Search for templates:
`https://github.com/search?q=cookiecutter&type=Repositories`

Metadata

CookieCutter: <https://cookiecutter.readthedocs.io>

Projects Templates:

- Create project directories and base files.
- Allows to follow best practices.
- Avoids forgetting important features.
- Example: cookiecutter
`gh:snakemake-workflows/cookiecutter-snakemake-workflow`

See also:

- Search for templates:
`https://github.com/search?q=cookiecutter&type=Repositories`

Metadata

License · s

Authors Rights Belongs to:

- Writings = you.
- Diagrams = you.
- Code = your employer(s).
- (But information disclosure authorizations may be limited)

License = what Authors Rights are given to Users, covers:

- Distribution of the work,
- Modification of the work,
- Sublicensing of derived work,
- Linking of the work,
- Patent & Trademark grants.

Metadata

License · s

Authors Rights Belongs to:

- Writings = you.
- Diagrams = you.
- Code = your employer(s).
- (But information disclosure authorizations may be limited)

License = what Authors Rights are given to Users, covers:

- Distribution of the work,
- Modification of the work,
- Sublicensing of derived work,
- Linking of the work,
- Patent & Trademark grants.

Metadata

Free (Software) Licenses

Guarantee Freedom of:

- Use (run) the work,
- Study the source code,
- Redistribute the work,
- Improve the work.

These freedoms are necessary to reproducibility, even more for replicability.

- Distribution = essential,
- Modification = necessary,
- Sublicensing = necessary,
- Linking = important,
- Patent grant = important,
- Trademark grant = optionnal.

Metadata

Free (Software) Licenses

Guarantee Freedom of:

- Use (run) the work,
- Study the source code,
- Redistribute the work,
- Improve the work.

These freedoms are necessary to reproducibility, even more for replicability.

- Distribution = essential,
- Modification = necessary,
- Sublicensing = necessary,
- Linking = important,
- Patent grant = important,
- Trademark grant = optionnal.

Metadata

Suggested Licenses

For source code:

- Affero GPL v3: <https://www.gnu.org/licenses/agpl-3.0.en.html>
- LGPL v3: <https://www.gnu.org/licenses/lgpl-3.0.html>
- Apache license: <https://www.apache.org/licenses/LICENSE-2.0>

For generic work:

- Creative Commons Attribution Share-Alike (CC-BY-SA):
<https://creativecommons.org/licenses/by-sa/4.0/>

See also:

- Choose a Creative Commons license:
<https://creativecommons.org/choose/>
- https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licences
- YOUR EMPLOYER.

Metadata

Suggested Licenses

For source code:

- Affero GPL v3: <https://www.gnu.org/licenses/agpl-3.0.en.html>
- LGPL v3: <https://www.gnu.org/licenses/lgpl-3.0.html>
- Apache license: <https://www.apache.org/licenses/LICENSE-2.0>

For generic work:

- Creative Commons Attribution Share-Alike (CC-BY-SA):
<https://creativecommons.org/licenses/by-sa/4.0/>

See also:

- Choose a Creative Commons license:
<https://creativecommons.org/choose/>
- https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licences
- YOUR EMPLOYER.

Metadata

Suggested Licenses

For source code:

- Affero GPL v3: <https://www.gnu.org/licenses/agpl-3.0.en.html>
- LGPL v3: <https://www.gnu.org/licenses/lgpl-3.0.html>
- Apache license: <https://www.apache.org/licenses/LICENSE-2.0>

For generic work:

- Creative Commons Attribution Share-Alike (CC-BY-SA):
<https://creativecommons.org/licenses/by-sa/4.0/>

See also:

- Choose a Creative Commons license:
<https://creativecommons.org/choose/>
- https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licences
- YOUR EMPLOYER.

Publication

Open-Access

Reproducibility needs open-access.

Preprints repositories:

- arXiv: <https://arxiv.org>
- bioRxiv: <https://www.biorxiv.org>
- HAL: <https://hal.archives-ouvertes.fr>

See also:

- https://en.wikipedia.org/wiki/Open_access

Publication

Open-Access

Reproducibility needs open-access.

Preprints repositories:

- arXiv: <https://arxiv.org>
- bioRxiv: <https://www.biorxiv.org>
- HAL: <https://hal.archives-ouvertes.fr>

See also:

- https://en.wikipedia.org/wiki/Open_access

Publication

Zenodo: <https://zenodo.org>

Long term archiving of code and data associated with a publication:

- Archive Code (with tests),
- Link to a Git tag on Github,
- Archive Data,
- Link to/from publications (via DOI or preprint ID),
- Get a DOI (easier to cite),
- Allows versions.

See also:

- Software Heritage: <https://www.softwareheritage.org>

Publication


Zenodo: <https://zenodo.org>

Long term archiving of code and data associated with a publication:

- Archive Code (with tests),
- Link to a Git tag on Github,
- Archive Data,
- Link to/from publications (via DOI or preprint ID),
- Get a DOI (easier to cite),
- Allows versions.

See also:

- Software Heritage: <https://www.softwareheritage.org>

A background image showing two scientists in a laboratory. A man in a white lab coat is leaning over a woman, also in a white lab coat and wearing glasses. They appear to be working together on a task. The setting is a modern laboratory with large windows in the background.

Part 3

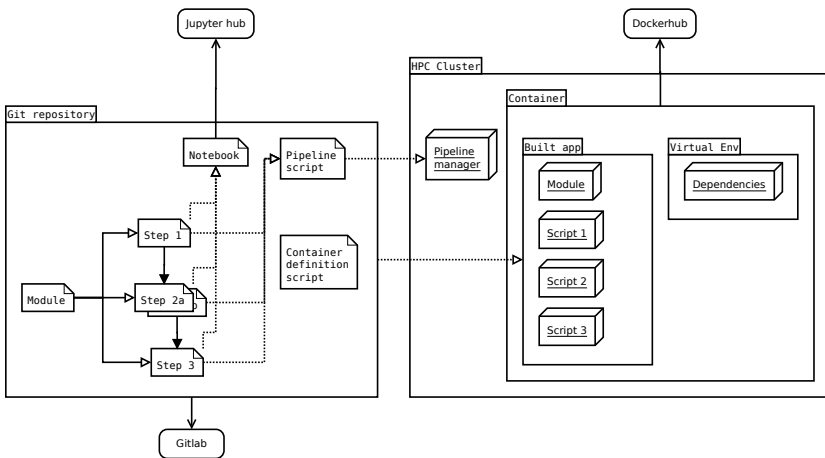
Conclusion

Minimal Workflow

- You start with a notebook, coding a simple proof of concept,
- Once it works, you modularize out some functions,
- You code a module, in a local Git repository.
- You add asserts as you type.
- When a function works, you write the related test.
- The notebook becomes a tutorial on a how to use your code.
- When a feature works, you test it in a container.
- The container installs dependencies *via* Conda.
- To run experiments, you write a pipeline script, calling the container.
- You push your repository on Gitlab.
- You archive the code & data on Zenodo.
- You write the publication, citing the archives.
- You put the publication on a preprint archive.

Summary

App management



Summary

Essential tools

- Git — <https://git-scm.com>
- Gitlab — <https://about.gitlab.com>
- Jupyter — <https://jupyter.org>
- Conda — <https://conda.io>
- Singularity — <https://syslabs.io/singularity>
- SnakeMake — <https://snakemake.readthedocs.io>
- openMOLE — <https://openmole.org>
- CookieCutter — <https://cookiecutter.readthedocs.io>
- AGPLv3 — <https://www.gnu.org/licenses/agpl-3.0.en.html>
- bioRxiv — <https://www.biorxiv.org>
- Zenodo — <https://zenodo.org>

Happy reproducing.

Read more about reproducible pipelines: [Wratten *et al.* 2021]

—




Do not hesitate to test your code & to ask questions:

`johann.dreo@pasteur.fr`

Part 4

References



-
-  Baker, M., Is there a reproducibility crisis? Nature 533, 2016, 452–454.
 -  Wratten, L., Wilm, A. & Göke, J., Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. Nat Methods 18, 2021, 1161–1168.
 -  López-Ibáñez, M., Branke, J., Paquete, L., Reproducibility in Evolutionary Computation, ACM Transactions on Evolutionary Learning and Optimization, 1(4), 2021, 1–21.