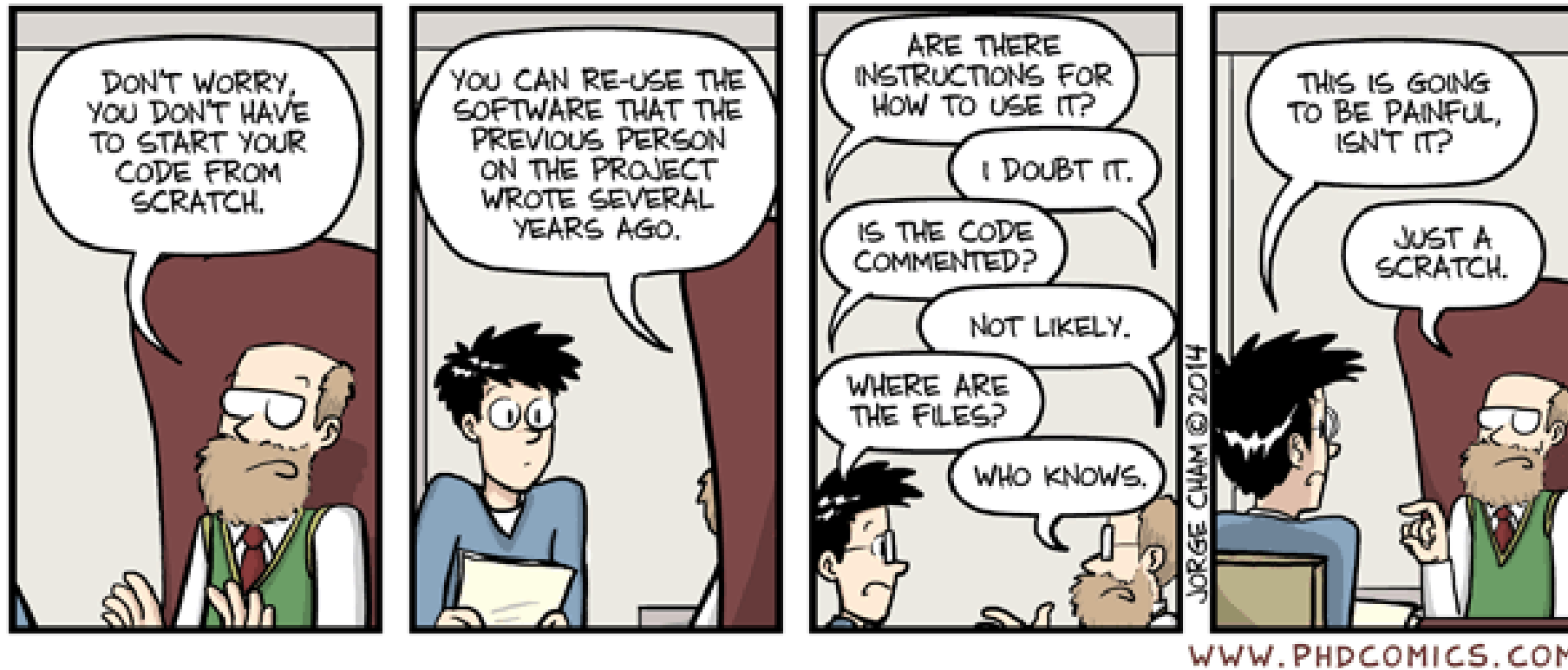


Writing reproducible code



Remember, your past self can also be the previous developer...

A scary anecdote

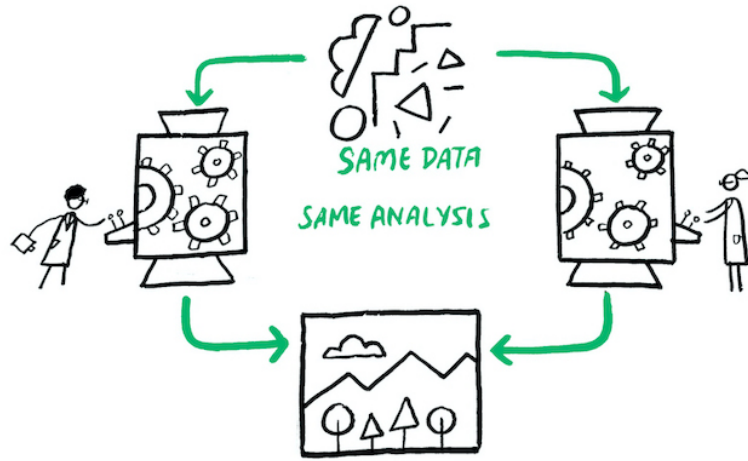
- A group of researchers obtain great results and submit their work to a high-profile journal.
- Reviewers ask for new figures and additional analysis.
- The researchers start working on revisions and generate modified figures, but find inconsistencies with old figures.
- The researchers can't find some of the data they used to generate the original results, and can't figure out which parameters they used when running their analyses.
- The manuscript is still languishing in the drawer ...

What is reproducible research?

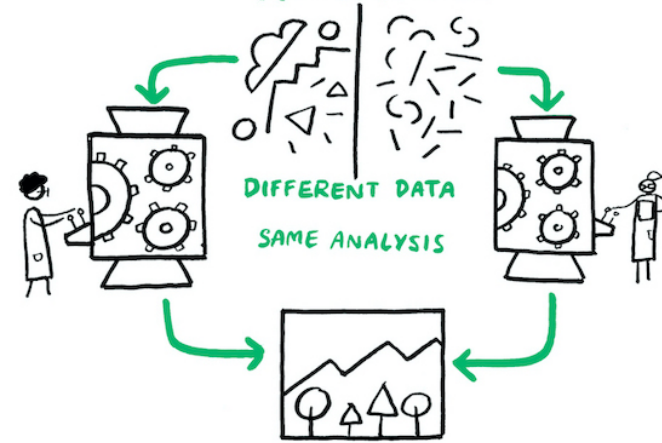
reproducibility refers to the ability of a researcher to **duplicate the results** of a prior study using the same materials as were used by the original investigator. That is, a second researcher might use the same raw data to build the same analysis files and implement the same statistical analysis in an attempt to yield the same results. Reproducibility is a **minimum necessary condition** for a finding to be believable and informative.

U.S. National Science Foundation (NSF) subcommittee on replicability in science

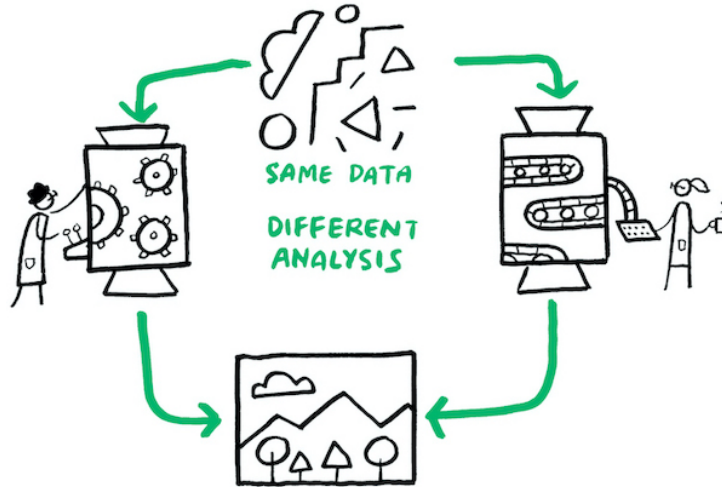
REPRODUCIBLE



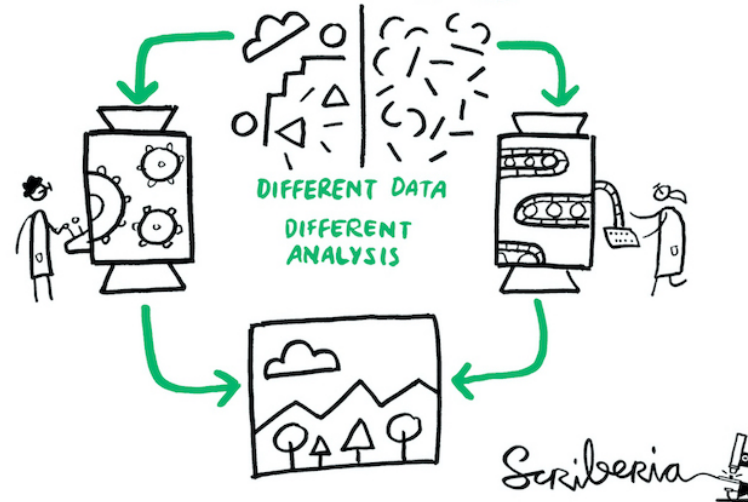
REPLICABLE



ROBUST



GENERALISABLE



Scriberia 

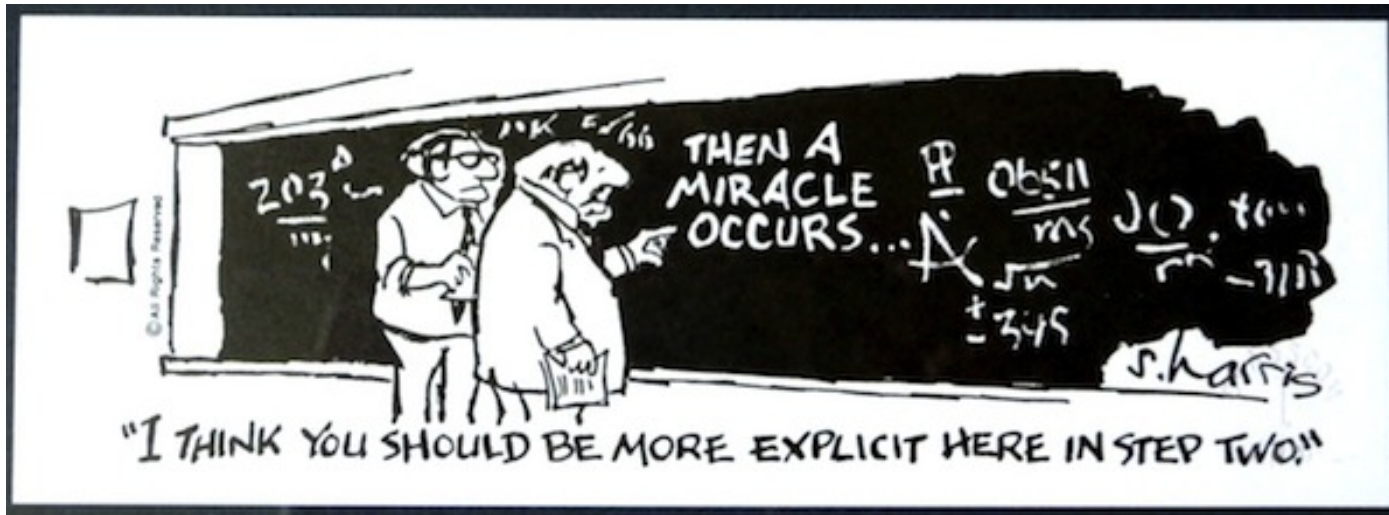
Advantages of reproducible code

1. Track a complete history of your research (version control really is a *must*)
2. Facilitate collaboration and review process ([blog](#) Anton Akhmerov)
3. Publish validated research and avoid misinformation
4. Write your papers, thesis and reports efficiently
5. Get credits for your work fairly
6. Ensure continuity of your work

Would this be enough?

- Access to the code
- Access to the data
- (And let's assume we can replicate the environment)

How confident do you feel?



Would this be enough?

We need to do more to inspire trust

- The code is correct (and I have made it easy for you/someone to check)
- My workflow is robust
- My workflow *itself* is accessible, and I will be guiding you through it.
- Sufficient documentation on how the experiment is conducted and data is generated
- Clear steps on how to recreate the software environment (OS, dependencies, version)

Method and data

When results are produced by complex computational processes using large volumes of data, the methods section of a traditional scientific paper is insufficient to convey the necessary information for others to reproduce the results.

Recommendations

- Provide input, intermediate, and output data
- Provide detailed methods, ideally in executable format
- Provide information about the computational environment

What is a computational environment?

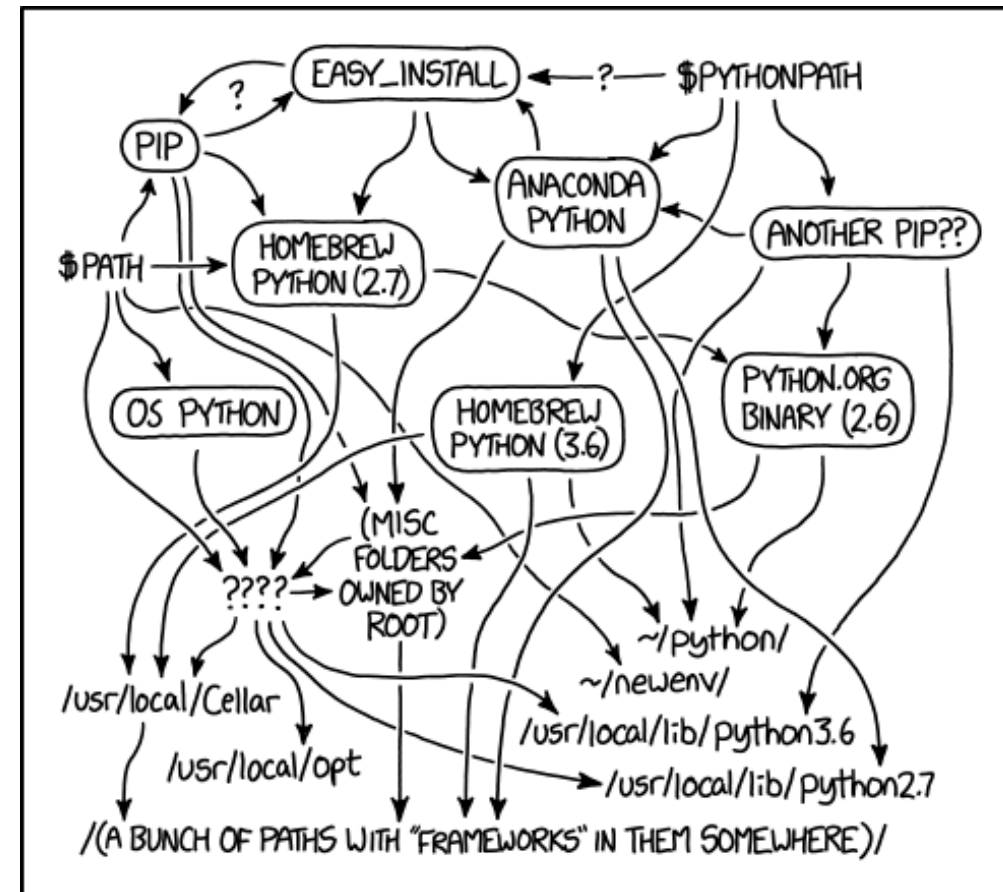
Information about the computational environment where the study was originally executed, such as operating system, hardware architecture, and library dependencies.

Do you think this is sufficient?

What other information would you need?

You should also consider...

- versions of dependencies (and their interoperability)
- configuration files and databases
- required (commercial) licenses, e.g. MATLAB
- tool for managing OS dependency (containers)
- for interactive systems, all user input provided by the user
- locales (language conventions)



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Your turn: list your dependencies

Inspect your project and list your dependencies

- operating system
- python dependencies
- MATLAB Add-ons

Can you list the steps required to recreate your results?

Organizing your code and data

- Contain your project in a single recognizable folder
- Distinguish folder *types*, name them accordingly:
 - **Read-only:** data, metadat
 - **Human-written:** code, paper, documentation
 - **Project-generated:** clean data, figures, models ...
- Initialize a **README** file, document your project
- Choose a **license**
- Publish your project

```
.
├── .gitignore
├── CITATION.md
├── LICENSE.md
├── README.md
├── requirements.txt
├── bin
│   └── external
├── config
├── data
│   ├── processed
│   ├── raw
│   └── temp
├── docs
│   ├── manuscript
│   └── reports
├── results
│   ├── figures
│   └── output
└── src
```

<- Compiled and external code, ignored by git (PG)
<- Any external source code, ignored by git (RO)
<- Configuration files (HW)
<- All project data, ignored by git
<- The final, canonical data sets for modeling. (PG)
<- The original, immutable data dump. (RO)
<- Intermediate data that has been transformed. (PG)
<- Documentation notebook for users (HW)
<- Manuscript source, e.g., LaTeX, Markdown, etc. (HW)
<- Other project reports and notebooks (e.g. Jupyter, .Rmd) (HW)

<- Figures for the manuscript or reports (PG)
<- Other output for the manuscript or reports (PG)
<- Source code for this project (HW)

Tracking source code, data, and results

- All code is version controlled and goes in the `src/` directory
- Include appropriate LICENSE file and information on software requirements
- You can also version control data files or input files under `data/`
- If data files are too large (or sensitive) to track, untrack them using `.gitignore`
- Intermediate files from the analysis are kept in `data/processed`
- Consider using Git tags to mark specific versions of results (version submitted to a journal, dissertation version, poster version, etc.):

Example - CodeRefinery [word-count](#)

A note on paths

- Your project should be transportable between computers
- For this reason, you should use **relative paths**, compare
 - `C:/User/maurits/_projects/pvmd_toolbox/data/measurement.csv`
 - `./data/measurement.csv`
- It is fine to have the main script (e.g. `main.m`) in the home folder

To access code in subdirectories in MATLAB, you could make use of

```
% main.m
here = mfilename('fullpath');
addpath(genpath(here));
```

Choosing a license

- Copyright is implicit, others cannot use your code without your permission
- Licensing gives that permission, and its boundaries and conditions
- Choosing a license early on means being aware of your license as the project proceeds (and not creating conflicts with dependencies)

What is important to you? What does your group use?

- [Choose a license](#)
- The TU Delft offers pre-approved licenses: [TU Delft Research Software Policy](#)

Publishing your project

Uh... Isn't 'publication' the thing you do... *at the end?*

No! Publishing your project at an early stage

- forces you to consider readability throughout
- minimizes the mess you have to deal with when you (finally) decide to publish
- allows collaboration and support
- facilitates sharing and re-use.

But what if someone scoops my code! I'm a revolutionary, they will steal my ideas!

If you want to be more careful, you can always opt for a private repository. It is your work and up to you. But consider the advantages!

Where do I publish?

Living project: GitHub / GitLab

- synergistic with version control software git
- makes history public and accessible (eek!)
- allows publication of different releases
- provides a platform for interaction and collaboration

Archival

- Zenodo or 4TU.ResearchData



"FINAL".doc



FINAL.doc!



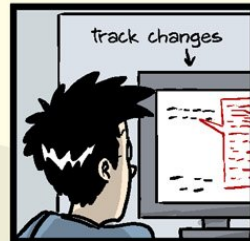
FINAL_rev.2.doc



FINAL_rev.6.COMMENTS.doc



FINAL_rev.8.comments5.
CORRECTIONS.doc



FINAL_rev.18.comments7.
corrections9.MORE.30.doc



FINAL_rev.22.comments49.
corrections.10.#@\$%WHYDID
ICOMETOGRADSCHOOL?????.doc

JORGE CHAM © 2012

Why do you need version control?

- It will help you manage ~~your code~~ most of your files (it is like track changes on steroids: it applies to all files in a folder).
- It allows you to trace back your steps: if something breaks, you can figure out what happened.
- NO MORE thesis_final_final_SERIOUSLYFINAL.md

even better

- a good version control system allows you to collaborate and share!
- a good version control system facilitates experimentation!
- access to use third-party **services** such as code quality checkers, correctness checkers.

Also useful if you do not code

- Working together on projects
- Setting up your website
- Making your work available to others (slides, newsletters)
- Keeping track of other projects (stars)
- Project management tools (Project Boards, Issues)

Examples

- Research: [Event Horizon Telescope](#)
- Tools: [Jupyter](#) & [scikit-learn](#) (Written in Python)
- [Matpower](#) (written in MATLAB)

Some questions

- Are you using version control?
- How do you currently share your code?
- How do you collaborate on code (and handle issues)

Topics we could discuss:

- [GitHub without the command line](#)
- [Setting up git](#)
- Discuss collaborative practices in PVMD toolbox

Want to learn more?

- Software Carpentry [lessons](#)
- Code Refinery [lessons](#)

Join a workshop (and receive GS credits)


- TU Delft [Training for Researchers](#) (not available for students)
- Self-paced study (available to all!)

The FAIR principles

Original paper demands that all scholarly digital research objects should be findable, accessible, interoperable, and reusable

Increasingly recognized as essential for the transition towards Open Science

www.nature.com/scientificdata

SCIENTIFIC DATA

Amended: Addendum

OPEN

SUBJECT CATEGORIES

- » Research data
- » Publication characteristics

Comment: The FAIR Guiding Principles for scientific data management and stewardship

Mark D. Wilkinson *et al.*[#]

There is an urgent need to improve the infrastructure supporting the reuse of scholarly data. A diverse set of stakeholders—representing academia, industry, funding agencies, and scholarly publishers—have come together to design and jointly endorse a concise and measurable set of principles that we refer to as the FAIR Data Principles. The intent is that these may act as a guideline for those wishing to enhance the reusability of their data holdings. Distinct from peer initiatives that focus on the human scholar, the FAIR Principles put specific emphasis on enhancing the ability of machines to automatically find and use the data, in addition to supporting its reuse by individuals. This Comment is the first formal publication of the FAIR Principles, and includes the rationale behind them, and some exemplar implementations in the community.

Received: 10 December 2015
Accepted: 12 February 2016
Published: 15 March 2016

FAIR for Research Software

Not a settled matter...

- The Netherlands eScience Center - ["Five Recommendations for FAIR Software"](#)
- TU Delft DCC Guides - [Checklist](#)

Steps towards reproducibility

1. Make sure you can find it (in space)
2. Make sure you can find it (in time)
3. Make sure you can recreate the environment where it lived at a specific time
4. Make sure you can execute the same sequence of operations
5. Make sure your environment and sequence of operations is robust and no human is needed to replicate what was done

Continuum of practices in (data) science

I do not want
to do this job
anymore

Bare minimum
practices

Good enough
practices

Best practices in
data & project
management



A single folder for all projects where everyone
can write
No backup strategy
No README files
No comments on code
No version control
No separation between raw/derivative data
No meaningful filenames
No history log
No way to replicate past analysis
No way for external person to know what's
going on

(Open) data archived with DOI
Data versioning with git-annex/DVC
Registered protocols for data collection
GIT version control + branches & tags
(Open) code on GitHub + archived w DOI
Unit tests and continuous integration
Singularity/Docker container
Automation with Make/Snakemake
Consistent folder structure across projects
Extensive documentation
Collaboration tools (slack, trello, etc)