Research software engineering for HPC

Radovan Bast (fosstodon.org/@radovan)

UiT The Arctic University of Norway

About me

- Theoretical chemist turned research software engineer.
- I write research software and teach programming to researchers and lead the CodeRefinery project.
- I lead the high-performance computing group and the research software engineering group at UiT.

What is "research software"?

- Script to convert data from one format to another
- Script to read data and visualize it
- Program that generates data
- Analysis script
- Set of scripts that form an analysis pipeline
- Code that is compiled
- Code that is dynamically interpreted and not compiled
- Web app
- ...

You don't need to be a "proper software engineer" to produce research software

We consider **any code, script, notebook, or file, regardless of size**, as "research software" if it is needed to generate, visualize, or reproduce data/results as part of a publication.

CodeRefinery

Typical format: 6 half-days, twice per year, online, free, live-streamed, recorded, archived asynchronous Q&A in collaborative document

- Version control
- Collaboration using Git
- Testing
- Documentation
- Notebooks
- Modular code development
- Reproducible research
- Software licensing
- How to share and publish code
- How to organize a code project
- ...

Next workshop September 19-21 and 26-28, 2023, register here: https://coderefinery.github.io/2023-09-19-workshop/

Lessons and recordings: https://coderefinery.org/lessons/

6 most important RSE topics?

- Version control
- Documentation
- Reproducibility and containers
- Building code with CMake (HPC-specific part)
- Automated testing
- Sharing and reusing

[Midjourney, CC-BY-NC 4.0]

Exercises

We will revisit these during the exercise session:

- Version control and documentation
- Reproducibility and containers
- Building code with CMake
- Sharing and reusing

Version control

📜

Inspiration and where to find more:

- Introduction to version control with Git
- Collaborative distributed version control
- Collaborating and sharing using GitHub without command line

Motivation: Version control is an answer to these questions:

"It broke ... hopefully I have a working version somewhere?"

"Can you please send me the latest version?"

"Where is the latest version?"

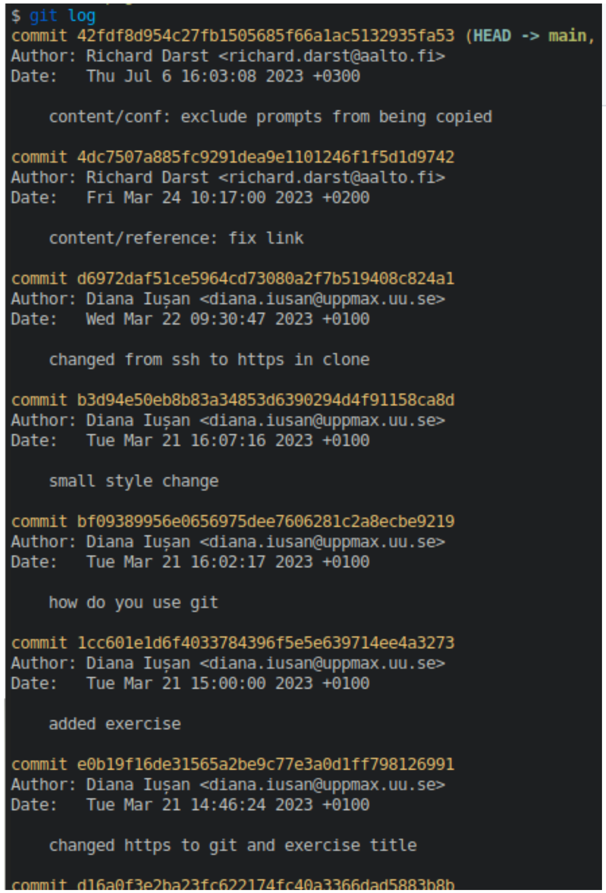
"Which version are you using?"

"Which version have the authors used in the paper I am trying to reproduce?"

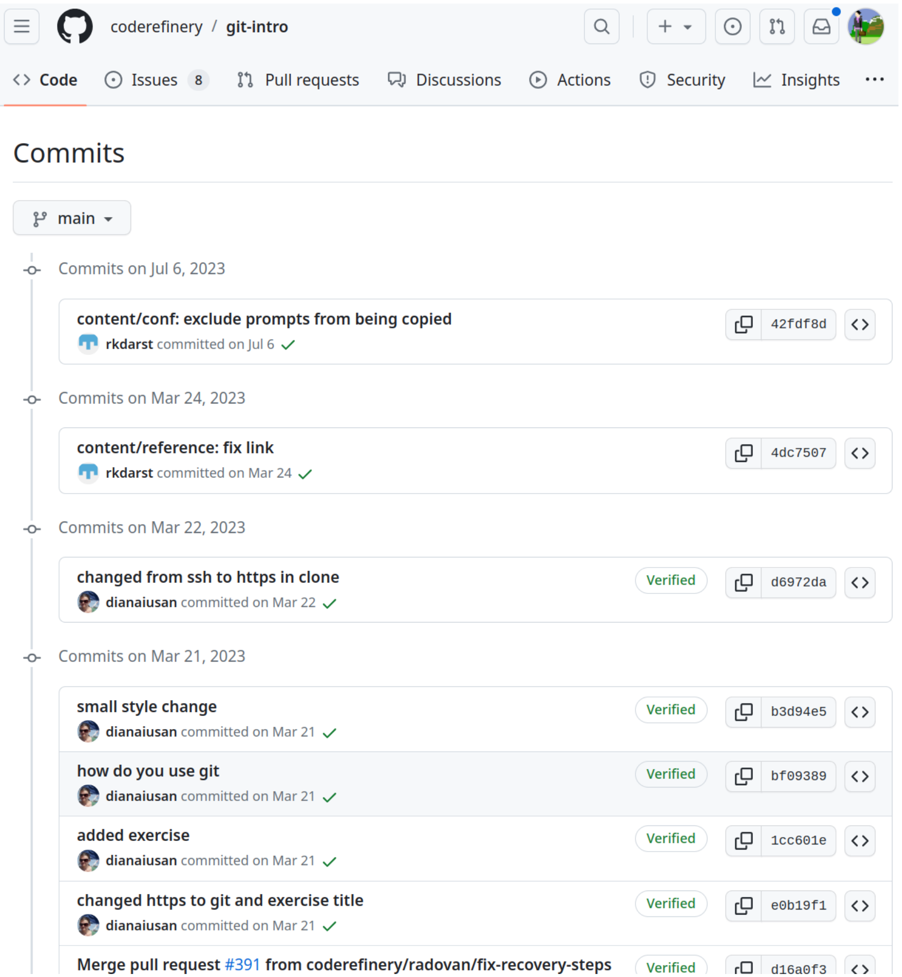
"Found a bug! Since when was it there?"

"I am sure it used to work. When did it change?"

Commits: keeping track of changes (example repository)



```
$ git log
commit 42fdf8d954c27fb1505685f66a1ac5132935fa53 (HEAD -> main,
Author: Richard Darst <richard.darst@aalto.fi>
Date:   Thu Jul 6 16:03:08 2023 +0300

    content/conf: exclude prompts from being copied

commit 4dc7507a885fc9291dea9e1101246f1f5d1d9742
Author: Richard Darst <richard.darst@aalto.fi>
Date:   Fri Mar 24 10:17:00 2023 +0200

    content/reference: fix link

commit d6972daf51ce5964cd73080a2f7b519408c824a1
Author: Diana Iușan <diana.iusan@uppmax.uu.se>
Date:   Wed Mar 22 09:30:47 2023 +0100

    changed from ssh to https in clone

commit b3d94e50eb8b83a34853d6390294d4f91158ca8d
Author: Diana Iușan <diana.iusan@uppmax.uu.se>
Date:   Tue Mar 21 16:07:16 2023 +0100

    small style change

commit bf09389956e0656975dee7606281c2a8ecbe9219
Author: Diana Iușan <diana.iusan@uppmax.uu.se>
Date:   Tue Mar 21 16:02:17 2023 +0100

    how do you use git

commit 1cc601e1d6f4033784396f5e5e639714ee4a3273
Author: Diana Iușan <diana.iusan@uppmax.uu.se>
Date:   Tue Mar 21 15:00:00 2023 +0100

    added exercise

commit e0b19f16de31565a2be9c77e3a0d1ff798126991
Author: Diana Iușan <diana.iusan@uppmax.uu.se>
Date:   Tue Mar 21 14:46:24 2023 +0100

    changed https to git and exercise title

commit d16a0f3e2ba23fc622174fc40a3366dad5883b8b
```



Commits

Commits on Jul 6, 2023

content/conf: exclude prompts from being copied
rkdarst committed on Jul 6 ✓ 42fdf8d

Commits on Mar 24, 2023

content/reference: fix link
rkdarst committed on Mar 24 ✓ 4dc7507

Commits on Mar 22, 2023

changed from ssh to https in clone
dianaiusan committed on Mar 22 ✓ Verified d6972da

Commits on Mar 21, 2023

small style change
dianaiusan committed on Mar 21 ✓ Verified b3d94e5

how do you use git
dianaiusan committed on Mar 21 ✓ Verified bf09389

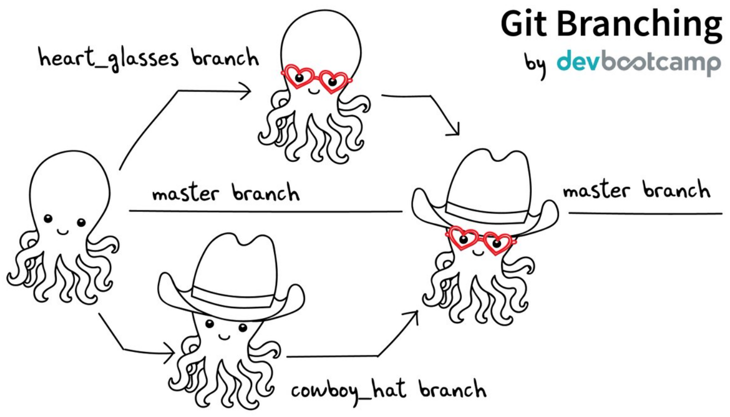
added exercise
dianaiusan committed on Mar 21 ✓ Verified 1cc601e

changed https to git and exercise title
dianaiusan committed on Mar 21 ✓ Verified e0b19f1

Merge pull request #391 from coderefinery/radovan/fix-recovery-steps
Verified d16a0f3

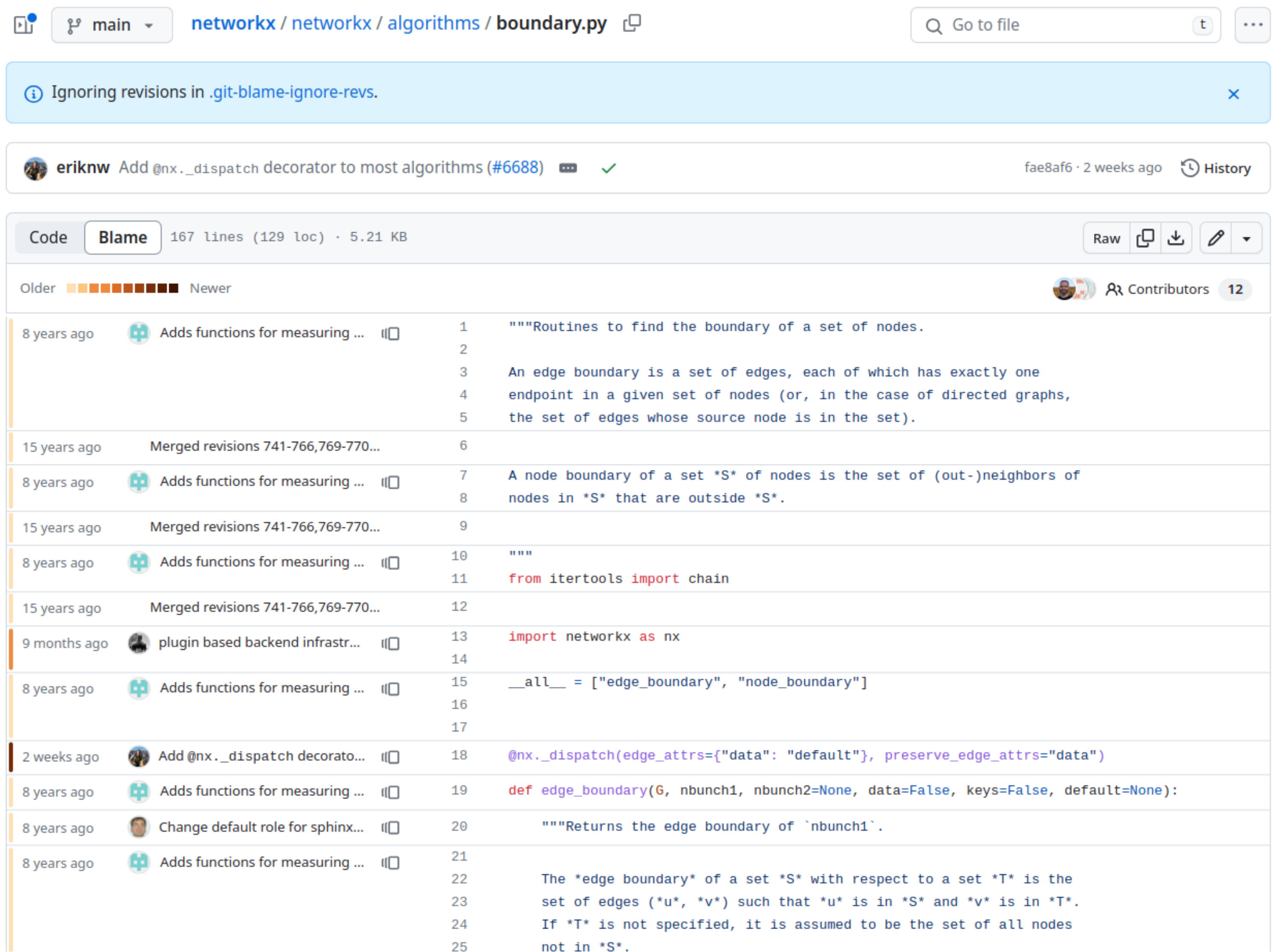
Features: roll-back, branching, merging, collaboration

- Roll-back: you can always go back to a previous version and compare
- Branching and merging: work on different ideas at the same time
- Collaboration: review, compare, share, discuss
- [Example network graph]



[Source: https://twitter.com/jay_gee/status/703360688618536960]

Reproducibility (browse this example online)


main networkx / networkx / algorithms / boundary.py

Go to file t

Ignoring revisions in .git-blame-ignore-revs.

eriknw Add @nx._dispatch decorator to most algorithms (#6688) ✓ fae8af6 · 2 weeks ago History

Code Blame 167 lines (129 loc) · 5.21 KB Raw

Older 🟧🟧🟧🟧🟧🟧🟧🟧🟧🟧 Newer Contributors 12

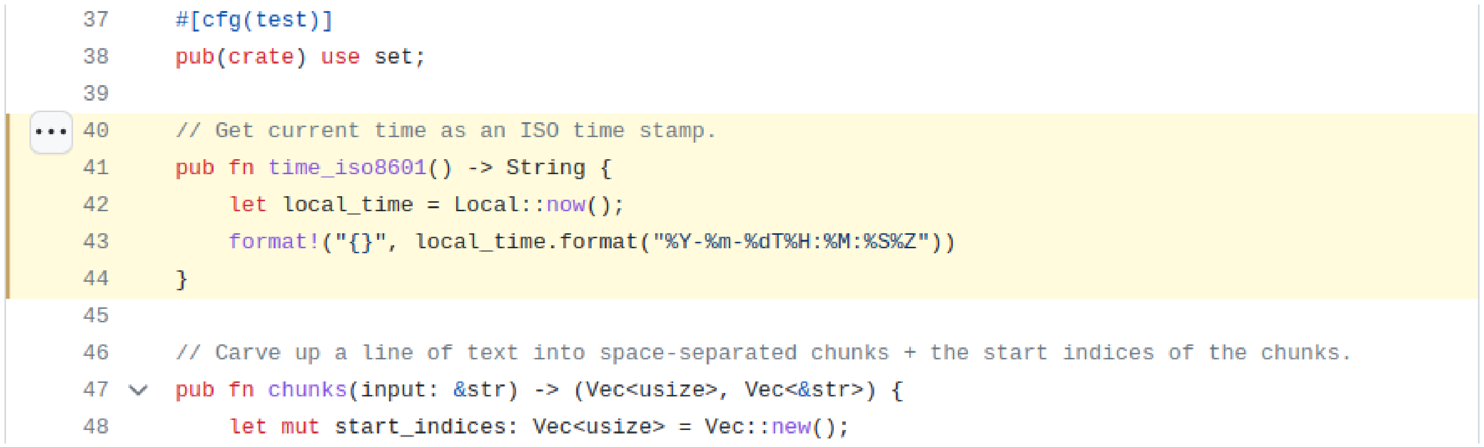
8 years ago	Adds functions for measuring ...
15 years ago	Merged revisions 741-766,769-770...
8 years ago	Adds functions for measuring ...
15 years ago	Merged revisions 741-766,769-770...
8 years ago	Adds functions for measuring ...
15 years ago	Merged revisions 741-766,769-770...
9 months ago	plugin based backend infrastr...
8 years ago	Adds functions for measuring ...
2 weeks ago	Add @nx._dispatch decorato...
8 years ago	Adds functions for measuring ...
8 years ago	Change default role for sphinx...
8 years ago	Adds functions for measuring ...

```python
"""Routines to find the boundary of a set of nodes.

An edge boundary is a set of edges, each of which has exactly one
endpoint in a given set of nodes (or, in the case of directed graphs,
the set of edges whose source node is in the set).

A node boundary of a set *S* of nodes is the set of (out-)neighbors of
nodes in *S* that are outside *S*.

"""
from itertools import chain

import networkx as nx

__all__ = ["edge_boundary", "node_boundary"]


@nx._dispatch(edge_attrs={"data": "default"}, preserve_edge_attrs="data")
def edge_boundary(G, nbunch1, nbunch2=None, data=False, keys=False, default=None):
    """Returns the edge boundary of `nbunch1`.

    The *edge boundary* of a set *S* with respect to a set *T* is the
    set of edges (*u*, *v*) such that *u* is in *S* and *v* is in *T*.
    If *T* is not specified, it is assumed to be the set of all nodes
    not in *S*.
```

Talking about code

Clone the code, go to the file "src/util.rs", and search for "time_iso8601". Oh! But make sure you use the version from August 2023.

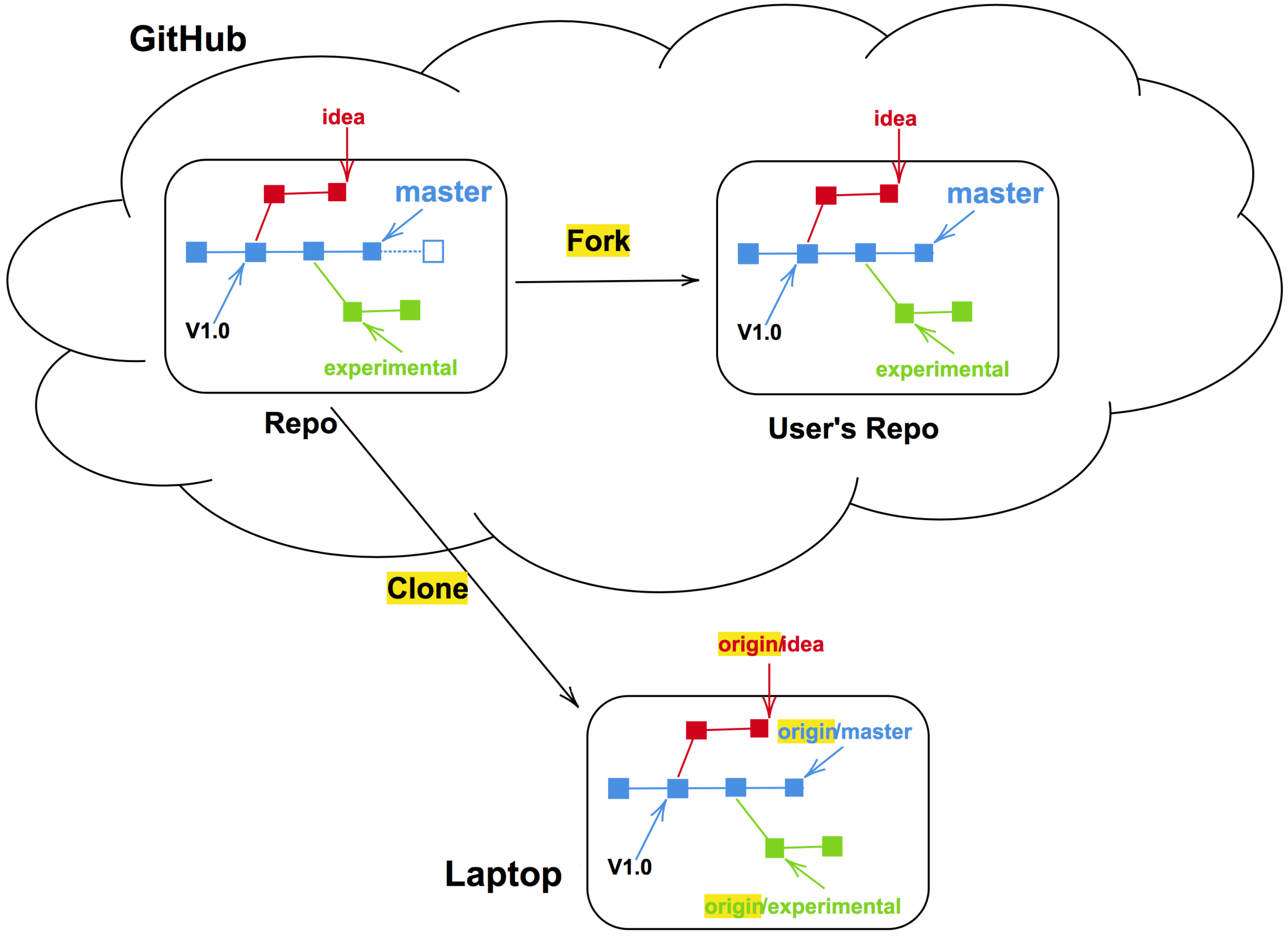
Or I can send you a [permalink]



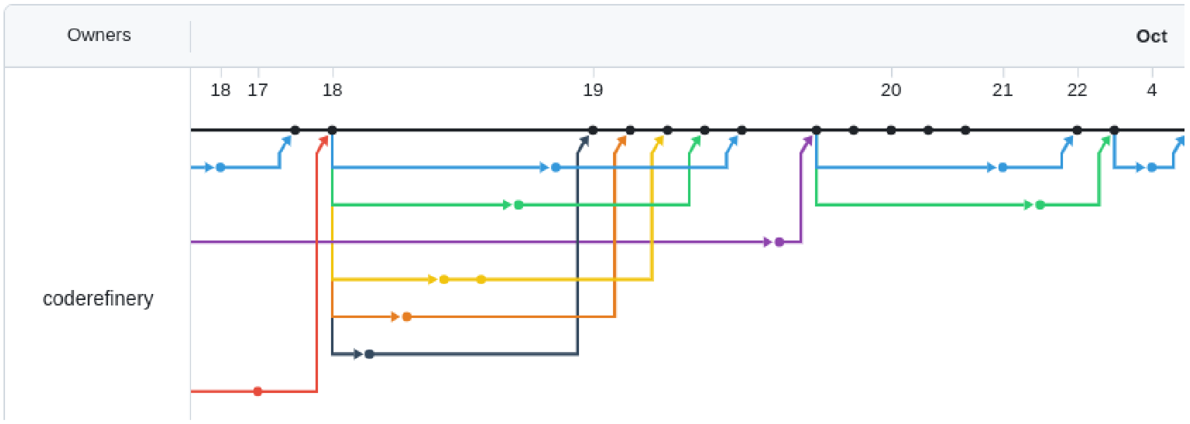
```
#[cfg(test)]
pub(crate) use set;

// Get current time as an ISO time stamp.
pub fn time_iso8601() -> String {
    let local_time = Local::now();
    format!("{}", local_time.format("%Y-%m-%dT%H:%M:%S%Z"))
}

// Carve up a line of text into space-separated chunks + the start indices of the chunks.
pub fn chunks(input: &str) -> (Vec<usize>, Vec<&str>) {
    let mut start_indices: Vec<usize> = Vec::new();
```

[https://github.com/NordicHPC/sonar/blob/75daafc86582feb06299d6a47c82112f39888152/src/util.rs#L40-L44]

Collaboration through branches or forks



Code review



- Changes are reviewed before they are merged
- Main motivation for code review is the collaborative learning
- Also: better code quality

Where to start? Simple personal projects

- Start with just the **`main`** branch
- Later use branches for unfinished/untested ideas
- Use tags to mark important milestones (**`phd-thesis-submitted`**, **`published-manuscript`**)
- Better too many commits than too few
- Better imperfect commits than no commits

Projects with few persons

- Write-protect the **`main`** branch
- New idea/feature: new branch
- Use code review: changes are reviewed and discussed before they are merged

- Install and configure Git
- In 3 commands from nothing to first commit:

```
$ git init
$ git add myscript.py
$ git commit
```

- Go through CodeRefinery lessons (Git intro and Collaborative Git)

Documentation

💗✉️ to your future self

Inspiration and where to find more:

- Documentation lesson material by CodeRefinery
- Talk material "Documenting code" by S. Wittke

Why? 💗✉️ to your future self

- You will probably use your code in the future and may forget details.
- You may want others to use your code (almost impossible without documentation).
- You may want others to contribute to the code.
- Time is limited - let the documentation answer FAQs.

Checklist

- Purpose
- Installation instructions
- Dependencies and their versions or version ranges
- Copy-paste-able example to get started
- Tutorials covering key functionality
- Reference documentation (e.g. API) covering all functionality
- How do you want to be asked questions (mailing list or forum or chat or issue tracker)
- Possibly a FAQ section
- Authors
- Recommended citation
- License
- Contribution guide

See also:

- JOSS review checklist

Not very useful (more commentary than comment):

```
# now we check if temperature is larger than -50
if temperature > -50:
    print("ERROR: temperature is too low")
```

More useful (explaining why):

```
# we regard temperatures below -50 degrees as measurement errors
if temperature > -50:
    print("ERROR: temperature is too low")
```

Keeping zombie code "just in case" (rather use version control):

```
# do not run this code!
# if temperature > 0:
#     print("It is warm")
```

Emulating version control:

```
# somebody: threshold changed from 0 to 15 on August 5, 2013
if temperature > 15:
    print("It is warm")
```

In-code documentation

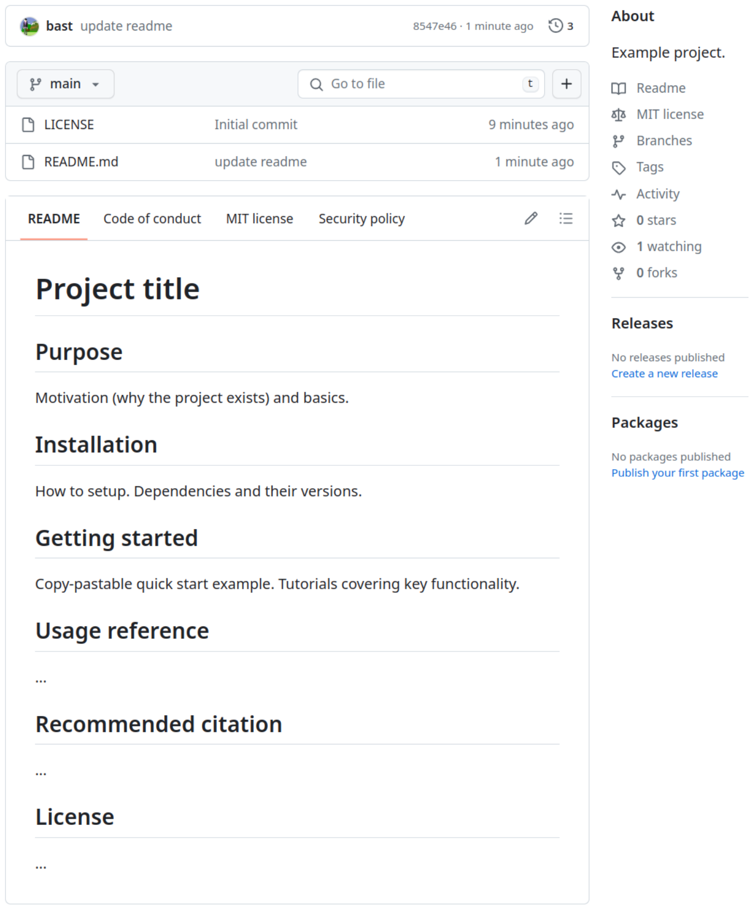
- Useful for those who want/need to understand and modify the code

- Docstrings can be useful both for developers and users of a function

```python
def kelvin_to_celsius(temp_k: float) -> float:
    """
    Converts temperature in Kelvin to Celsius.

    Parameters
    ----------
    temp_k : float
        temperature in Kelvin

    Returns
    -------
    temp_c : float
        temperature in Celsius
    """
    assert temp_k >= 0.0, "ERROR: negative T_K"

    temp_c = temp_k - 273.15

    return temp_c


print(kelvin_to_celsius.__doc__)
```

Often a README is enough (first impression!)

```
# Project title

## Purpose

Motivation (why the project exists)
and basics.

## Installation

How to setup. Dependencies and their
versions.

## Getting started

Copy-pastable quick start example.
Tutorials covering key functionality.

## Usage reference

...

## Recommended citation

...

## License

...
```



Example project.

LICENSE	Initial commit	9 minutes ago
README.md	update readme	1 minute ago

Project title

Purpose

Motivation (why the project exists) and basics.

Installation

How to setup. Dependencies and their versions.

Getting started

Copy-pastable quick start example. Tutorials covering key functionality.

Usage reference

...

Recommended citation

...

License

...

When projects grow out of a README

- Write documentation in Markdown (.md) or reStructuredText (.rst) or R Markdown (.Rmd)
- In the same repository as the code -> version control and **reproducibility**
- Use one of many tools to build HTML out of md/rst/Rmd: Sphinx, Zola, Jekyll, Hugo, RStudio, knitr, bookdown, blogdown, ...
- Deploy the generated HTML to GitHub Pages or GitLab Pages

Examples

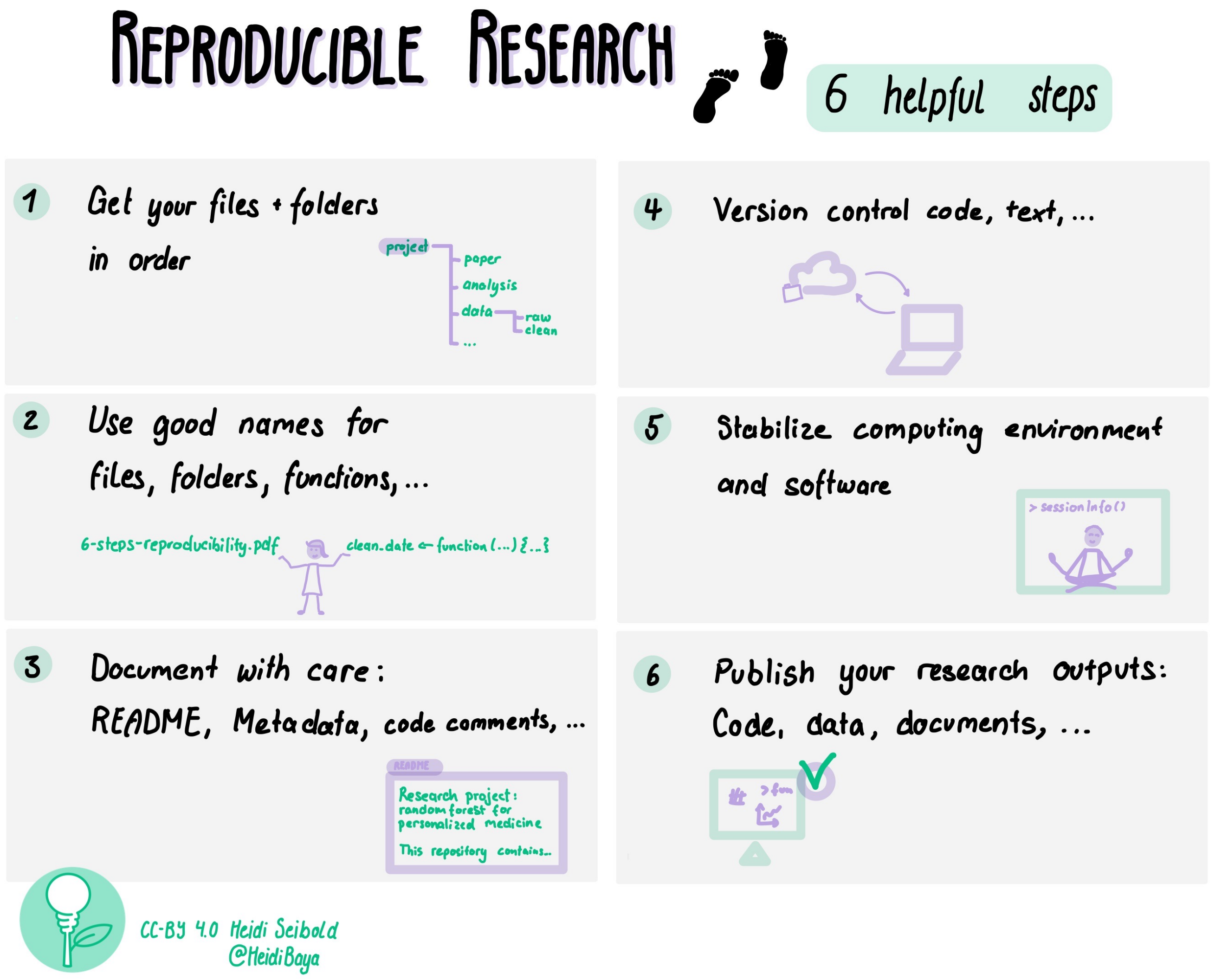
- All CodeRefinery lessons
- https://github.com/networkx/networkx

Reproducibility and containers

📦

Inspiration and where to find more:

- Reproducible research
- The Turing Way: Guide for Reproducible Research
- Ten simple rules for writing Dockerfiles for reproducible data science
- Computing environment reproducibility



Reproducible Research

6 helpful steps

1 Get your files + folders in order

2 Use good names for files, folders, functions, ...

3 Document with care: README, Metadata, code comments, ...

4 Version control code, text, ...

5 Stabilize computing environment and software

6 Publish your research outputs: Code, data, documents, ...

project
paper
analysis
data
raw
clean
...
6-steps-reproducibility.pdf
clean.date <- function(...){...}
README
Research project: random forest for personalized medicine
This repository contains..
> sessionInfo()
CC-BY 4.0 Heidi Seibold
@HeidiBaya

[Heidi Seibold, CC-BY 4.0, https://twitter.com/HeidiBaya/status/1579385587865649153]

It all starts with a good directory structure ...

```
project_name/
├──── README.md              # overview of the project
├──── data/                  # data files used in the project
│     ├──── README.md         # describes where data came from
│     └──── sub-folder/       # may contain subdirectories
├──── processed_data/        # intermediate files from the analysis
├──── manuscript/            # manuscript describing the results
├──── results/               # results of the analysis (data, tables, figures)
├──── src/                   # contains all code in the project
│     ├──── LICENSE           # license for your code
│     ├──── requirements.txt  # software requirements and dependencies
│     └──── ...
└──── doc/                   # documentation for your project
     ├──── index.rst
     └──── ...
```

Lottery factor: If you win the lottery and leave research today, will others be able to continue your work?

"it works on my machine 🤷"

Recording dependencies

Conda, Anaconda, pip, virtualenv, Pipenv, pyenv, Poetry, rye, requirements.txt, environment.yml, renv, …

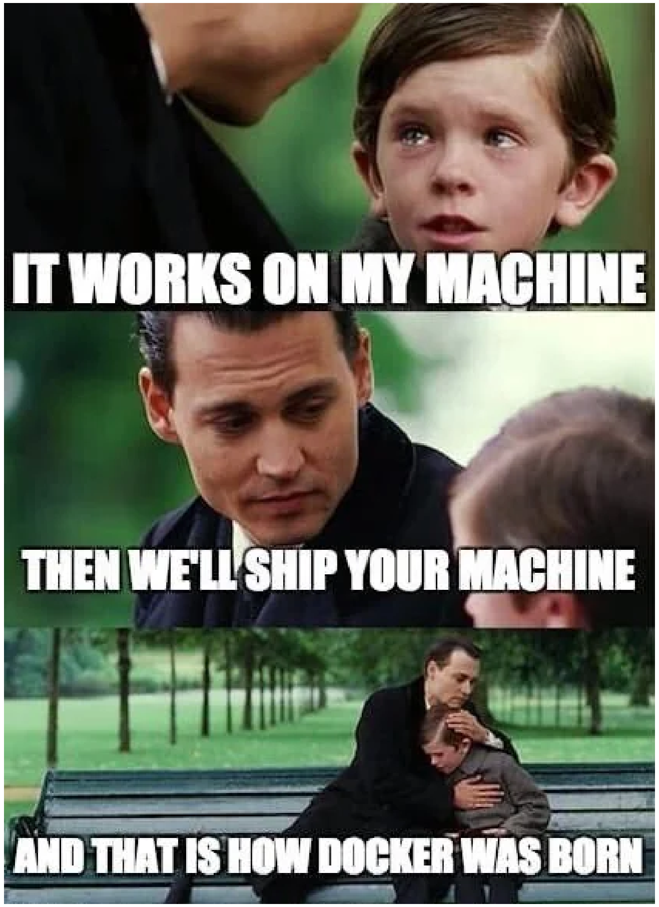
- Define dependencies
- Communicate dependencies
- Install these dependencies
- Record the versions
- Isolate environments
- Provide tools and services to share packages

Isolated environments help you make sure that you know your dependencies!

[Midjourney, CC-BY-NC 4.0]

Kitchen analogy

- Software <-> recipe
- Data <-> ingredients
- Libraries <-> cooking books/blogs



[From reddit]

Kitchen analogy

- Our codes/scripts <-> cooking recipes
- Container definition files <-> like a blueprint to build a kitchen with all utensils in which the recipe can be prepared.
- Container images <-> example kitchens
- Containers <-> identical factory-built mobile food truck kitchens

Container: "operating system inside a file"

Example SingularityCE/Apptainer definition file ("recipe"):

```
Bootstrap: docker
From: ubuntu:20.04

%post
    export DEBIAN_FRONTEND=noninteractive
    apt-get update -y

    apt install -y git build-essential pkg-config
    apt install -y libz-dev libbz2-dev liblzma-dev
    apt install -y libcurl4-openssl-dev libssl-dev libgsl-dev

    git clone https://github.com/someuser/sometool.git
    cd sometool

    make

%runscript
    export PATH=/sometool/bin:$PATH

    $@
```

Popular implementations: Docker, SingularityCE (popular on HPC) Apptainer (popular on HPC, fork of Singularity), podman

Container use cases

- Create a time capsule and share it on Zenodo (or similar)
- Document and communicate dependencies
- Have a common platform to test the code
- Easier to move it to other Linux computers/clusters
- Forward "travel in time": if cluster has too old software
- Backwards "travel in time": if software is no longer maintained and does not build on laptop/cluster

Typical critique points

- "not the proper way to build"
- performance
- composability

Recording computational steps

We need a way to record and communicate computational steps

- **README** (steps written out "in words")
- **Scripts** (typically shell scripts)
- **Notebooks** (Jupyter or R Markdown)
- **Workflows** (Snakemake, doit, ...)

[Midjourney, CC-BY-NC 4.0]

Building code with CMake

🧱

Inspiration and where to find more:

- [CMake introduction and hands-on workshop]

Why is Make not enough?

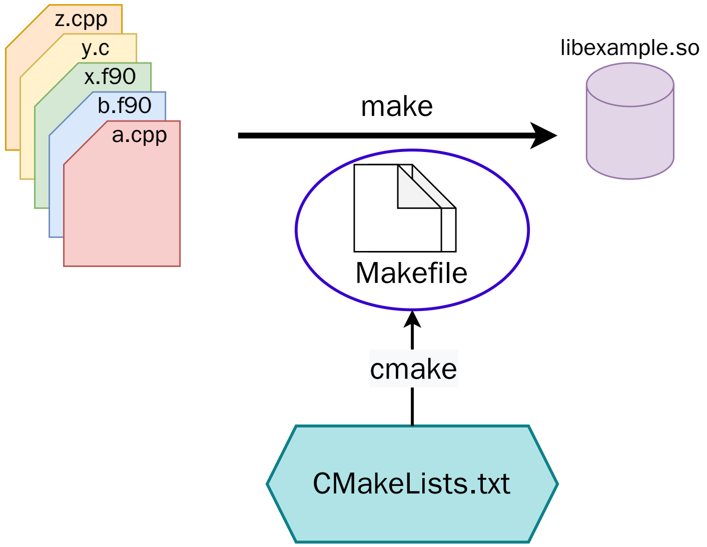
- Make only knows about targets and dependencies
- Make does not know which compiler (options) we want and which environment we are on
- We need to tell Make what depends on what (Fortran 90+ projects)
- Modular projects become clunky to maintain

What is CMake?

- Cross-platform (this is the C in CMake, not the C language)
- Open-source
- Manages the build process in a compiler-independent manner
- Provides a family of tools and a domain-specific language

CMake is not a build system

It generates files for build systems.



```
  Green Hills MULTI
* Unix Makefiles
  Ninja
  Ninja Multi-Config
  Watcom WMake
  CodeBlocks - Ninja
  CodeBlocks - Unix Makefiles
  CodeLite - Ninja
  CodeLite - Unix Makefiles
  Eclipse CDT4 - Ninja
  Eclipse CDT4 - Unix Makefiles
  Kate - Ninja
  Kate - Unix Makefiles
  Sublime Text 2 - Ninja
  Sublime Text 2 - Unix Makefiles
```

How do CMakeLists.txt files look?

```cmake
cmake_minimum_required(VERSION 3.14)

project(example LANGUAGES CXX)

add_executable(hello hello.cpp)

add_library(greeting
  SHARED
    greeting.cpp
    greeting.hpp
  )

find_package(MPI REQUIRED COMPONENTS CXX)

target_link_libraries(hello
  PRIVATE
    greeting
    MPI::MPI_CXX
  )
```

Why CMake?

- Excellent support for Fortran, C, C++, and mixed-language projects.
- Separation of source and build path: Out-of-source compilation.
- Really cross-platform (Linux, Mac, Windows, AIX, iOS, Android).
- Modular code development: Excellent support for multi-component and multi-library projects.
- Tools: Testing and packaging framework with CTest and CPack.
- Good at discovering environment, libraries, and packages.
- Non-intrusive: All you need is a **`CMakeLists.txt`**. CMake won't mind if other build tools are there as well in the project.

Automated testing

🤖 🚨 ✅

Inspiration and where to find more:

- [Software testing lesson material]

Technical possibilities

Any programming language has tools/libraries to perform:

- Unit tests: test a function or a module and compare function result to a reference
- End-to-end test: run the whole code and compare result to a reference
- Coverage analysis: Give overview of which parts of the code are tested
- The test (set) can be run automatically on GitHub Actions or GitLab CI after every Git commit

Motivation

- Less scary to change code: tests will tell you whether something broke
- Unit tests can guide towards better structured code: complicated code is more difficult to test
- Easier for new people to join
- Easier for somebody to revive an old code

Where to start

- A simple script or notebook probably does not need an automated test

If you have nothing yet

- Start with an end-to-end test
- Describe in words how *you* check whether the code still works
- Translate the words into a script
- Run the script automatically on every code change

If you want to start with unit-testing

- You want to rewrite a function? Start adding a unit test right there first.

Sharing and reusing

🌻

Inspiration and where to find more:

- UiT research software licensing guide (draft)
- Social coding lesson material by CodeRefinery

Why software licenses matter

- You find some great code or data that you want to reuse for your own publication (good for the original author: you will cite them and maybe other people who cite you will cite them).

- You need to modify the code a little bit, or you remix the data a bit.

- When it comes time to publish, you realize there is no license.

Now we have a problem:

- You manage to **publish the paper without the software/data** but others cannot build on your software and data and you don't get as many citations as you could.
- Or, you **cannot publish it at all** if the journal requires that papers should come with data and software so that they are reproducible.

Beginning of a project

[Midjourney, CC-BY-NC 4.0]

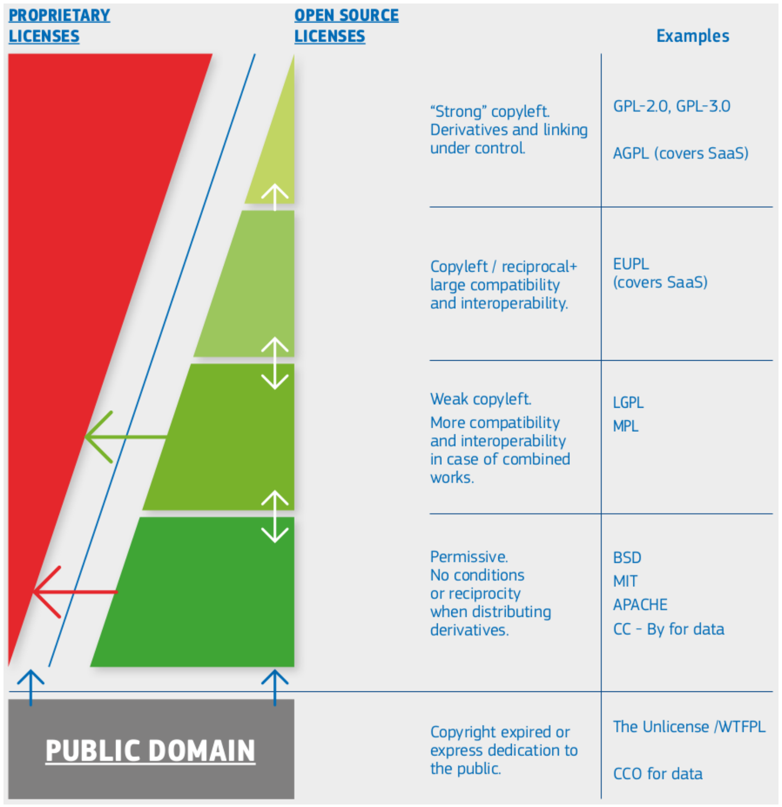
- License does not seem important
- Easy to change (*)
- Work as if the code is public even though it still may be private
- "Open core" approach: Core can be open and on a public branch, unpublished code can be on a private repository

Later in the project

[C.Stadler/Bwag, CC-BY-SA 4.0]

- Can be important
- Especially when combining codes or organizations
- Difficult to change
- Difficult to remove code that should not be published
- Authors change affiliation

Is your work derivative work or not?


PROPRIETARY LICENSES
OPEN SOURCE LICENSES
Examples
"Strong" copyleft. Derivatives and linking under control.
GPL-2.0, GPL-3.0
AGPL (covers SaaS)
Copyleft / reciprocal+ large compatibility and interoperability.
EUPL (covers SaaS)
Weak copyleft. More compatibility and interoperability in case of combined works.
LPGL
MPL
Permissive. No conditions or reciprocity when distributing derivatives.
BSD
MIT
APACHE
CC - By for data
PUBLIC DOMAIN
Copyright expired or express dedication to the public.
The Unlicense /WTFPL
CCO for data

- Derivative work: You have started from an existing code and made changes to it or if you incorporated an existing code into your code
- You have started from scratch: not derivative work

[European Union Public Licence (EUPL): guidelines July 2021,

https://data.europa.eu/doi/10.2799/77160]

How do I add a license to my work?

- Create a **`LICENSE`** file or **`LICENSES/`** folder in your project which will hold license texts.
- On top of each file add and adapt the following header (more examples):

```
# SPDX-FileCopyrightText: 2023 Jane Doe <jane@example.com>
#
# SPDX-License-Identifier: MIT
```

- Add a CITATION.cff file (example later)

Practical steps for making **changes to an existing project** (with a license that allows you to do so):

- Fork (copy) the project.
- Summarize your changes in file headers and bigger-picture changes in the README.
- Some licenses are more permissive (you can keep your changes private) but some licenses require you to publish the changes (share-alike).

Make it persistent and citable

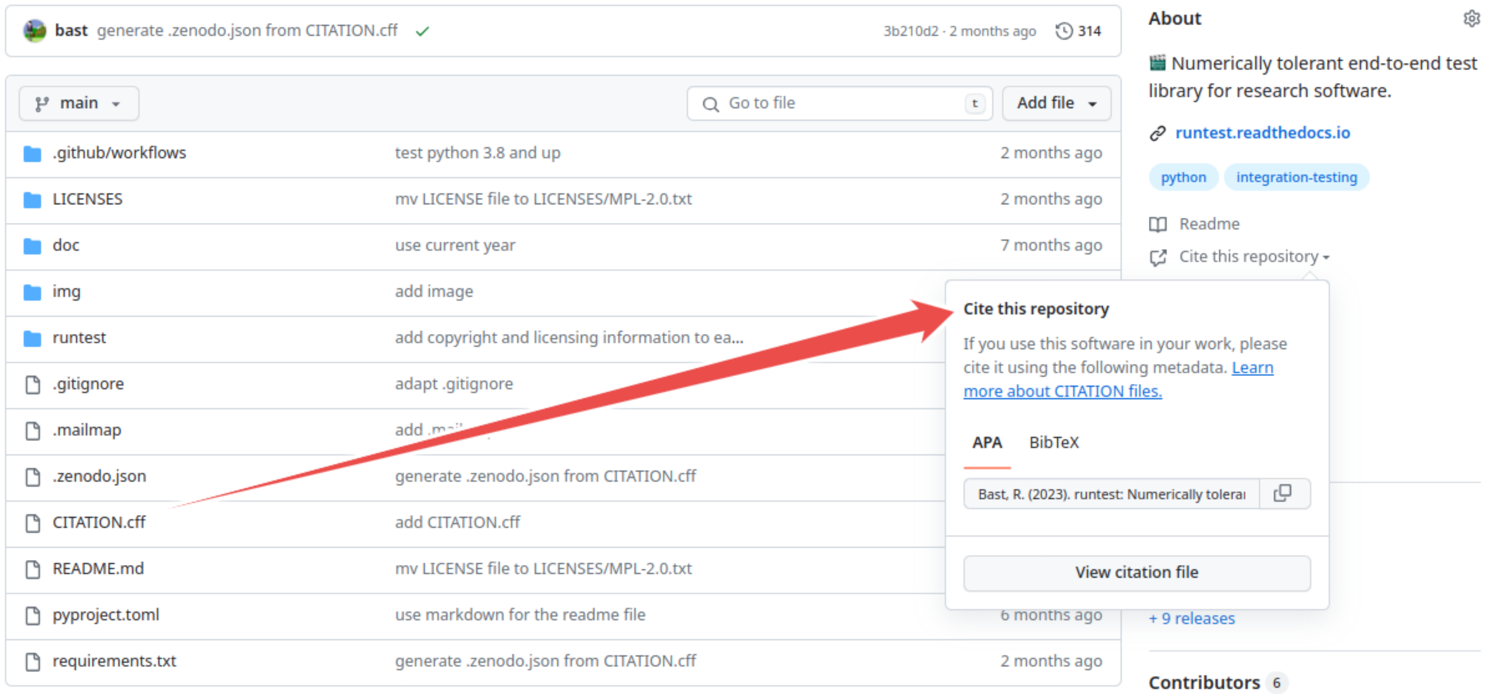
- Add a CITATION.cff file:

```
cff-version: 1.2.0
message: "If you use this software, please cite it as below."
authors:
- family-names: Doe
given-names: Jane
orcid: https://orcid.org/1234-5678-9101-1121
title: "My Research Software"
version: 2.0.4
doi: 10.5281/zenodo.1234
date-released: 2021-08-11
```

- Get a digital object identifier (DOI) for your code Zenodo or similar.

- Software Heritage and CodeMeta exist as an alternative ecosystem that is currently receiving some attention on a European level. Comparison and links to converters can be found in https://zenodo.org/record/8086413.

Many tools understand CITATION.cff



bast generate .zenodo.json from CITATION.cff ✓ 3b210d2 · 2 months ago 314

.github/workflows	test python 3.8 and up	2 months ago
LICENSES	mv LICENSE file to LICENSES/MPL-2.0.txt	2 months ago
doc	use current year	7 months ago
img	add image	
runtest	add copyright and licensing information to ea...	
.gitignore	adapt .gitignore	
.mailmap	add .m...	
.zenodo.json	generate .zenodo.json from CITATION.cff	
CITATION.cff	add CITATION.cff	
README.md	mv LICENSE file to LICENSES/MPL-2.0.txt	
pyproject.toml	use markdown for the readme file	6 months ago
requirements.txt	generate .zenodo.json from CITATION.cff	2 months ago

About

🎬 Numerically tolerant end-to-end test library for research software.

runtest.readthedocs.io

Cite this repository

If you use this software in your work, please cite it using the following metadata. Learn more about CITATION files.

Bast, R. (2023). runtest: Numerically tolerai

Sharing and reusing - Great resources

- UiT research software licensing guide (draft)
- Guide from the Aalto University in Finland: "Opening your Software at Aalto University"
- Joinup Licensing Assistant - Find and compare software licenses
- Joinup Licensing Assistant - Compatibility Checker
- Social coding lesson material by CodeRefinery
- Citation File Format (CFF)
- License Selector

Conclusions/recommendations

It's about communicating!

- Track your code with Git
- Help each other with reviewing code: great learning
- Documentation: start with a README in the same Git repo
- Document your dependencies and computational steps
- When adding tests, start with an end-to-end test
- Make your code/script/notebook citable and give it a license
- Join a CodeRefinery workshop