BASICS OF SOFTWARE PUBLICATION

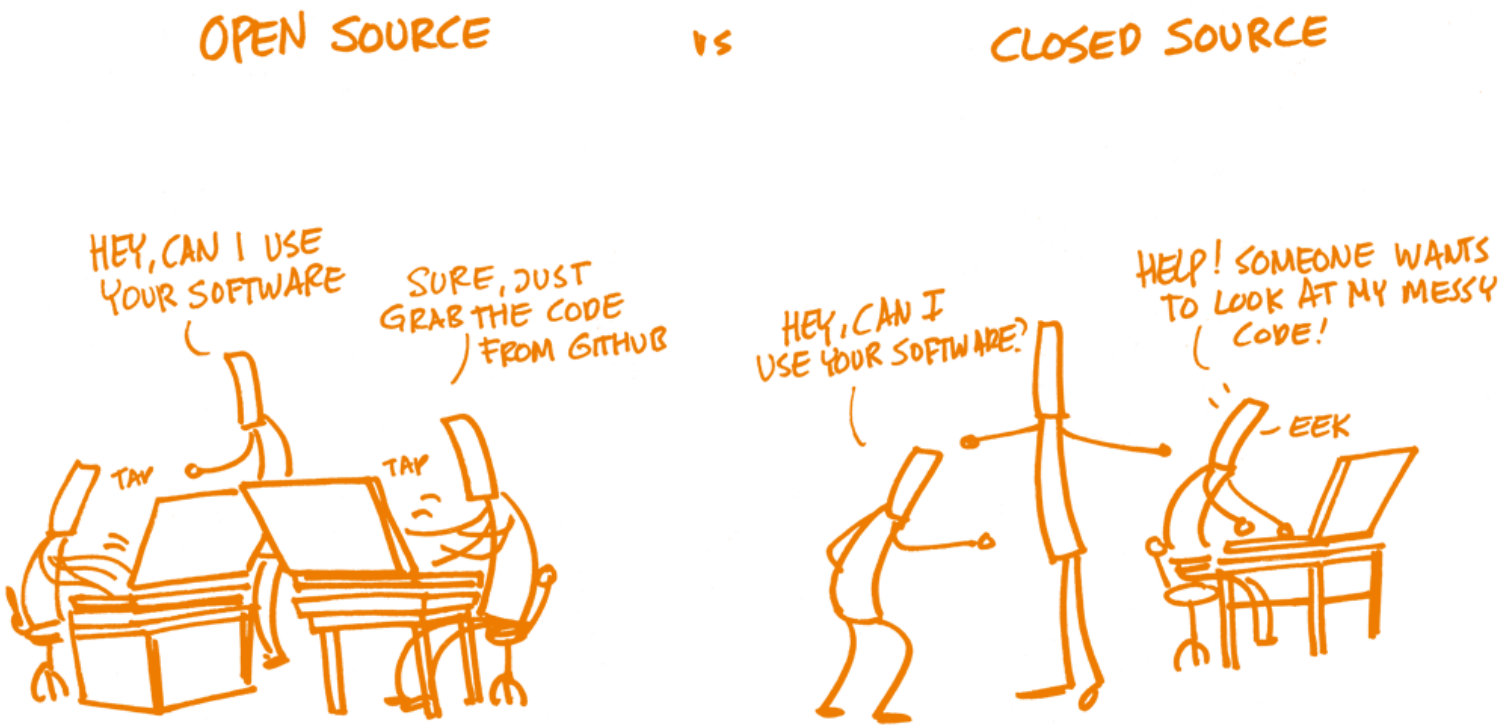
Carina Haupt
carina.haupt@dlr.de
@caha42(@scholar.social)

DLR

Reproducibility and Reuse of Software Development
Why Should I Care?

So that it‘s like this…

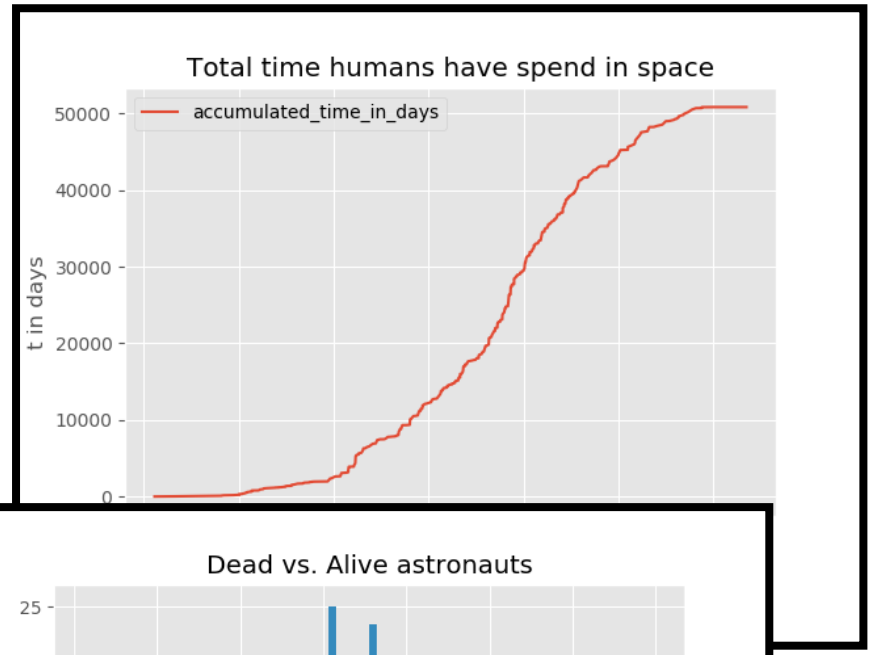
…instead of this.

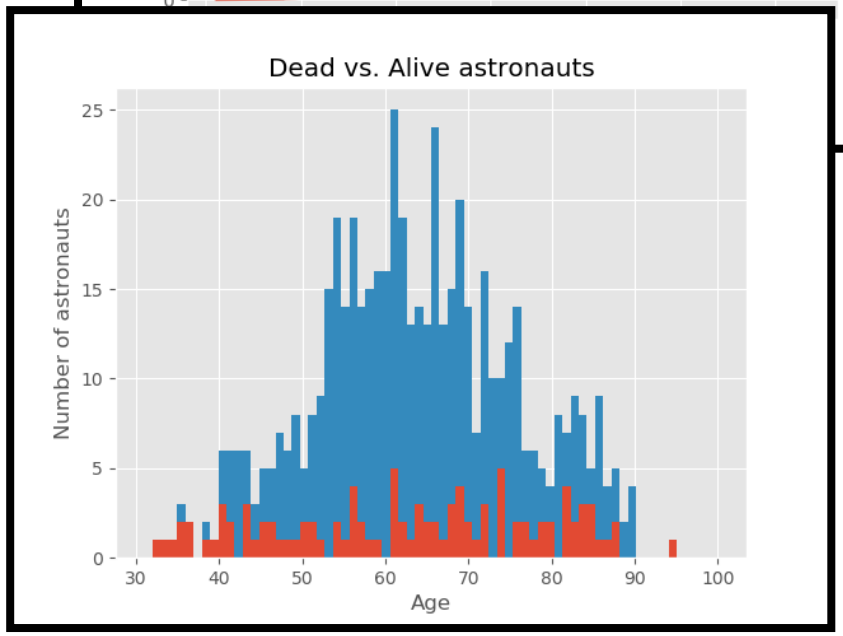


OVERVIEW ABOUT TYPICAL GOOD PRACTICES

Example: Astronaut Analysis

- **Astronauts Analysis is a data publication consisting of:**
  - Data set
  - Analysis script written in Python using pandas and matplotlib
  - Result plots
- **Scenario:**
  - I created it on my own as part of my job.
  - I want to make its reuse as easy as possible and make it available under an open source license.





Make your code reusable

- Step 1: Put your code under version control
- Step 2: Make sure that your code is in a sharable state
- Step 3: Add essential documentation
- Step 4: Add a license
- Step 5: Release your code

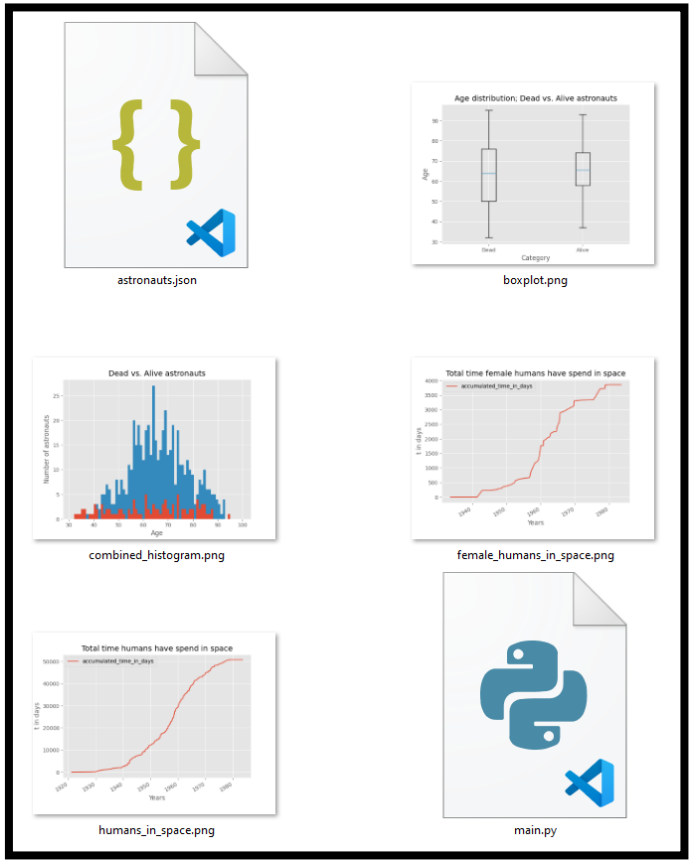
Essential aspects which you should try to already address for “internal” software!

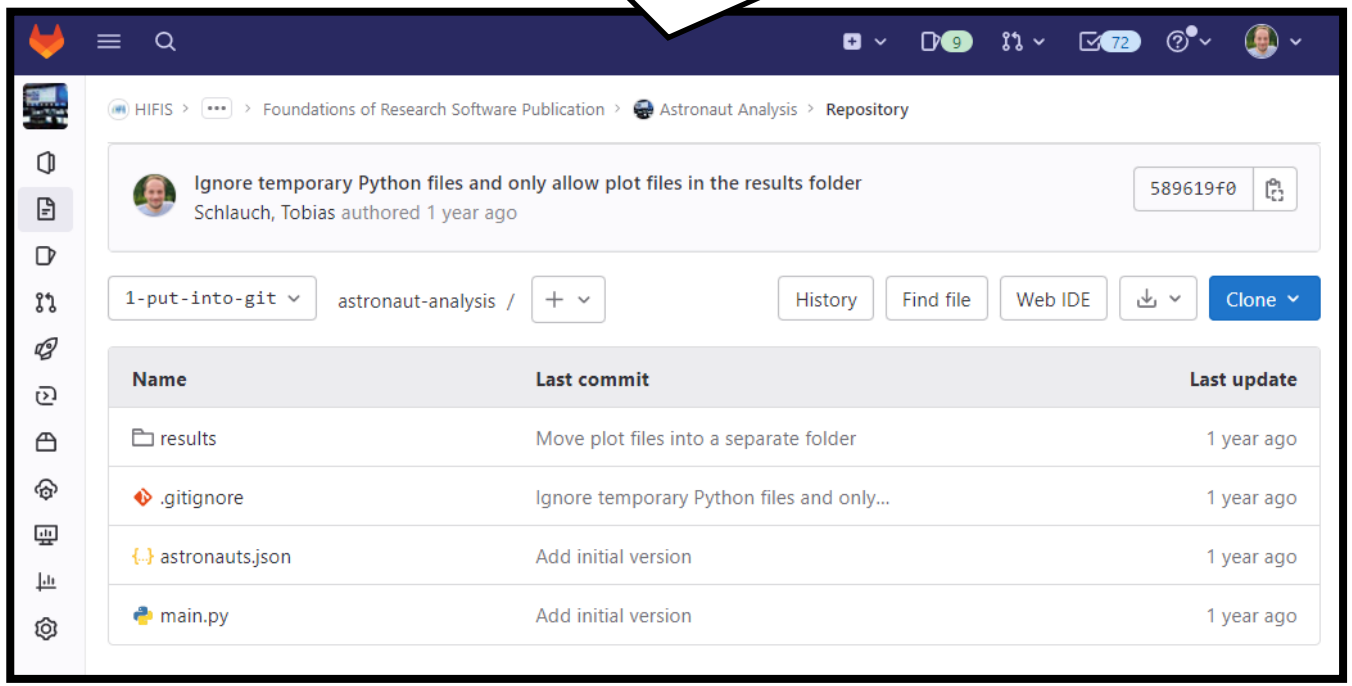
VERSION CONTROL

Step 1: Put Your Code Under Version Control
Where Should I Store My Code?

Minimum: Use a local Git repository + backup

Recommended: Use a code collaboration platform





Step 1: Put Your Code Under Version Control
What Belongs in the Repository?

- **Everything to make a usable version of your code** such as:
  - Source code, documentation, build scripts, test cases, configuration files, input data, …
- **Avoid adding generated files** such as:
  - Third-party libraries, generated binaries, ...
- **How to handle large (data) files?**
  - Available could be git-lfs, git-annex, Datalad or your research data management publication repository
- **Please note:**
  - Details depend on the “product” that you manage in the Git repository
  - **.gitignore files** helps you to control what goes into your repository. See also https://gitignore.io/ for templates.

SHARABLE STATE

DLR

Step 2: Make Sure That Your Code Is in a Sharable State
General Hints

- Make sure others can run your code:
  - No dependencies on internal resources (servers, storage, licensed software, ...)
  - No absolute paths
  - Clearly state dependencies + provide required build / installation scripts (e.g.: pip-tools, poetry) => crucial aspect of reproducibility
- Organize files in a suitable directory structure (e.g.: Python Application Layouts, Good Data Practices)
- Do not share sensitive data such as passwords, user accounts, SSH keys, internal IP addresses, etc. (e.g.: gitleaks)
- Orientate on standards of your domain / community

Step 2: Make Sure That Your Code Is in a Sharable State
Improve Your Code Style and Structure

- Strive for understandable code:
  - Apply a code style – consistency is more important than convenience (e.g.: PEP8)
  - Use a consistent and light code layout
  - Structure your code in suitable "building blocks" such as functions
  - Use specific and appropriate names for all artifacts
  - Provide sufficient level of code comments
- Read code of others for inspiration
- Try to do pair programming and reviews (even if it is with your rubber duck)

Step 2: Make Sure That Your Code Is in a Sharable State
Think About Testing and Automation

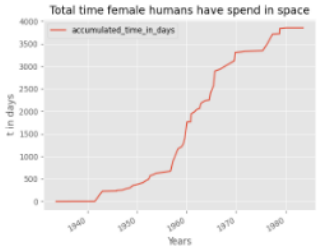
- Small tests are done easily but already show effect:
  - Code linters and checkers help to find poor code snippets and help to enforce coding styles (e.g.: flake8, black)
  - Automated tests work as an executable documentation (e.g.: pytest)
- Tests offer a good starting point for your automation efforts!

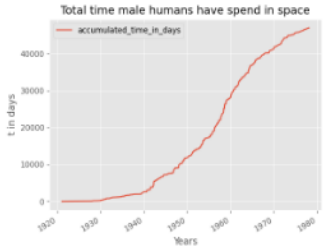
DOCUMENTATION

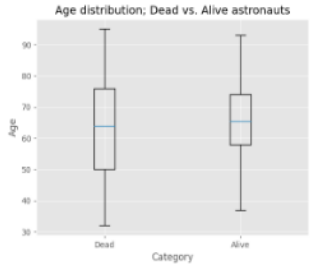
Step 3: Add Documentation

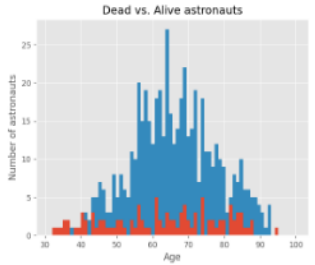
Astronaut Analysis

This analysis is based on publicly available astronauts data from Wikidata. In this context, we investigated aspects such as time humans spent in space as well as the age distribution of the astronauts.









The repository is organized as follows:

- data: Contains the astronauts data set retrieved from Wikidata
- code: Contains the astronaut analysis script
- results: Contains the resulting analysis plots

Astronaut Data

The data set has been generated using the following SPARQL query [1] (retrieval date: 2018-10-25).

You can also analyze a recent version of the astronaut data by replacing the data set and re-running the analysis script:

- Run the SPARQL query
- Download the resulting data formatted as JSON
- Replace the file `data/astronauts.json`
- Run the analysis script

Astronaut Analysis Script

The script requires Python >= 3.8 and uses the libraries pandas as well as matplotlib.

> The script has been successfully tested on Windows 10 and Linux with Python 3.8.

Typical Structure:

- **Software name**
- **Purpose**
- **Install**
- **Usage**
- **Contributing**
- Citation Hint
- **License**

Step 3: Add Documentation
General Hints

- **Mind your target groups:**
  - **Typical perspectives:** Users, contributors
  - **Users:** Installation / usage instructions, tutorials, support channels, …
  - **Contributors:** Contribution guidelines, technical overview, …
- **Think about adding typical documentation files** such as:
  - `README` (project front page), `CONTRIBUTING` (contributions guidelines), `CODE_OF_CONDUCT` (communication rules), `LICENSE` (license information), `CHANGELOG` (major changes), `CITATION` (citation metadata)
- **Please note:**
  - Markdown or another markup language is quite often used to write documentation
  - Usually, you will need additional documentation, for example, in a `docs` directory (e.g.: Sphinx, MkDocs)

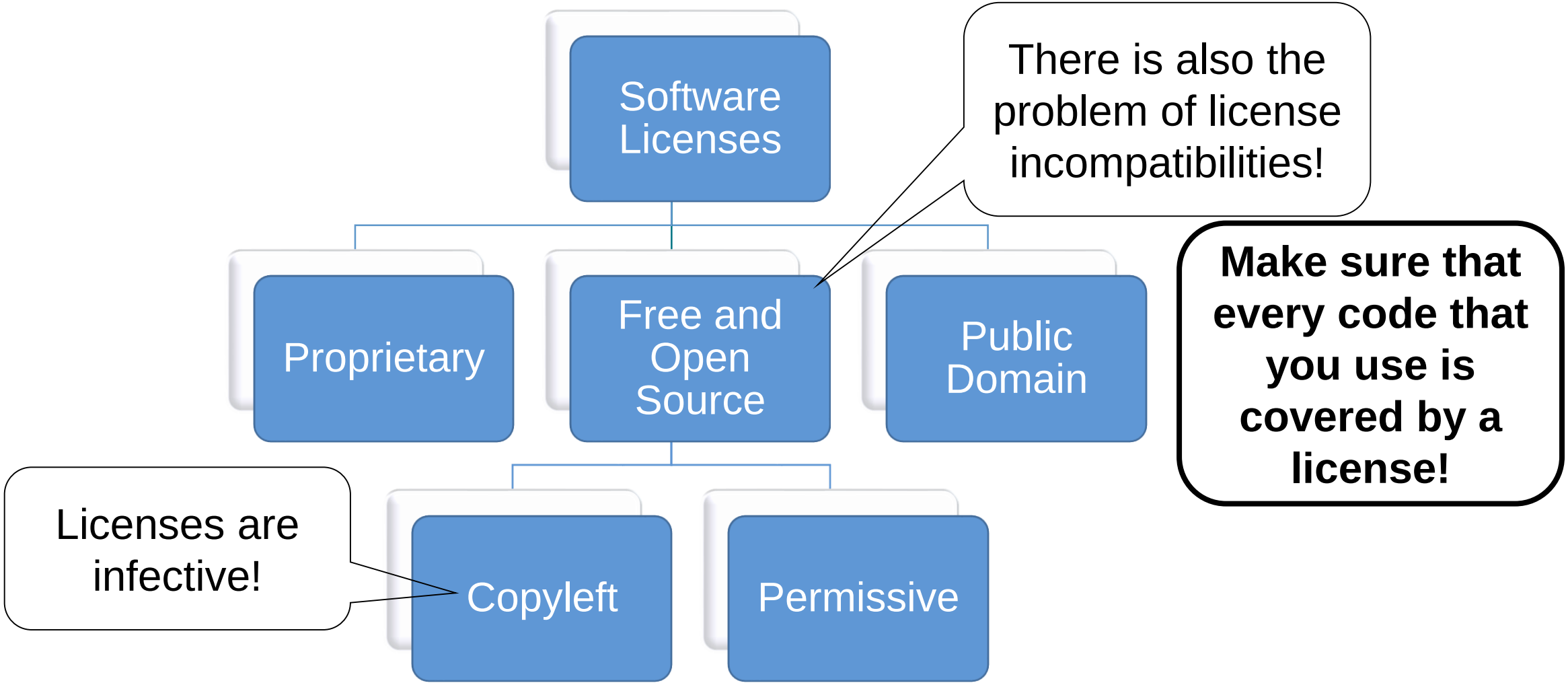
SOFTWARE LICENSING

Copyright Basics

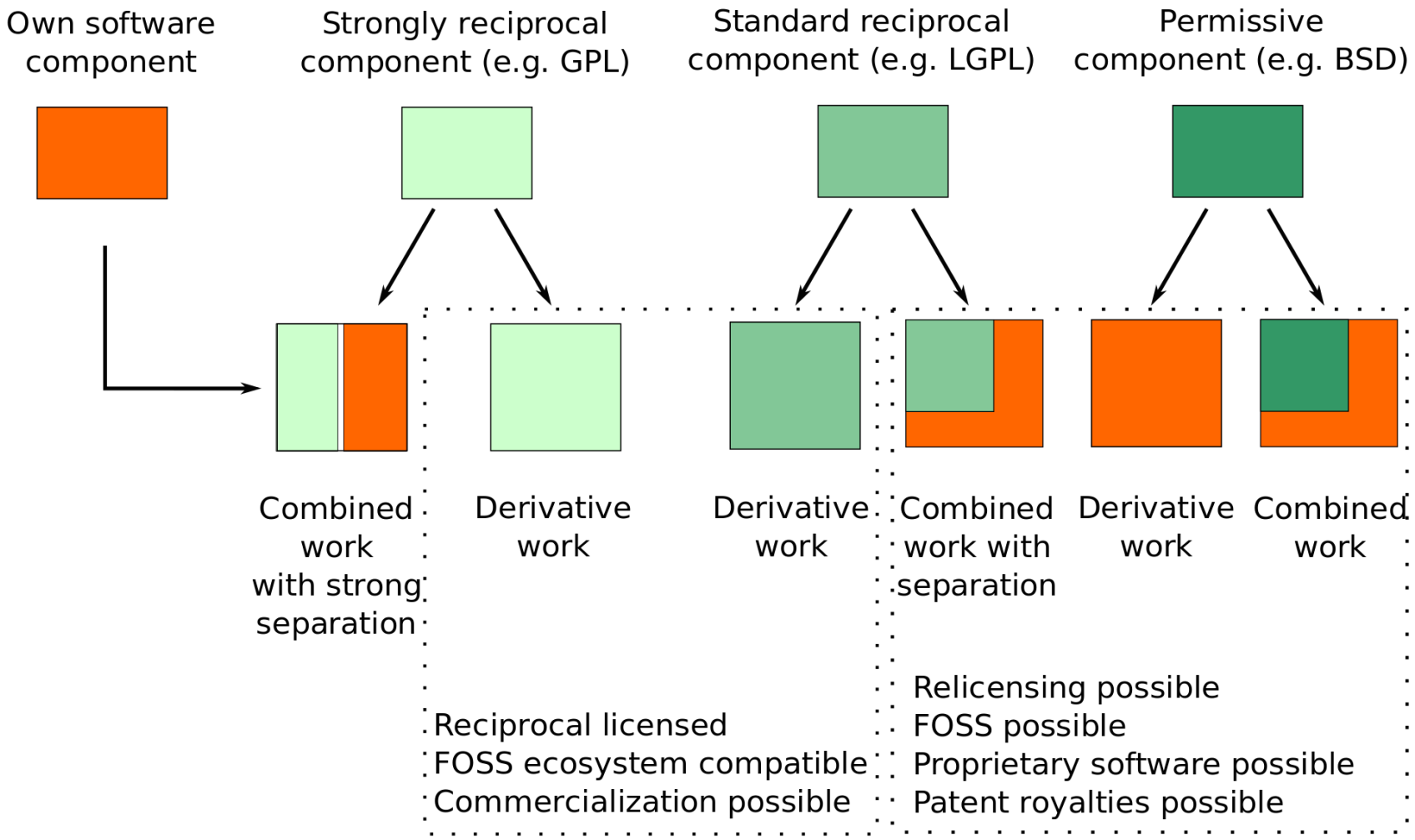
- **Copyright**
  - Software is protected by copyright.
  - Copyright protects the expression of an idea.
  - Copyright grants exclusive rights to the copyright holder.

- **Who is the copyright holder of a software?**
  - All contributors are considered as copyright holders and jointly exercise the rights granted by copyright.
  - A company paying an employed developer obtains most of the exclusive rights.

Software Licenses



Combining Modules under Different Licenses



Take care when combining code under different licenses!

Minimal License Checklist

1. Choose a license

- Consider strategical implications
- Comply with licenses of third-party dependencies

2. Ask your boss for permission

3. Add copyright holder and license information

- **Please note:**
  - DLR Open Source Brochure (German only) provides further detailed information.

Find out about your organizational processes!

Ask for legal advice if you are unsure!

Example: Astronaut Analysis
Choose a License

- After checking the recommendation from https://choosealicense.com/, I want to use the MIT License.
  - But do the licenses of my dependencies fit?
  - But what about the non-code artifacts?

```
$ liccheck -s liccheck.ini -r requirements.txt --no-deps
gathering licenses...
3 packages.
check unknown packages...
3 packages.
    flake8 (3.9.2): ['MIT']
    matplotlib (3.4.2): ['Python Software Foundation']
    pandas (1.2.4): ['BSD']
```

- **Final copyright and license decisions:**
  - Copyright holder: German Aerospace Center
  - Source code: MIT
  - Data set: CC0-1.0
  - Docs and plots: CC-BY-4.0
  - Insignificant files: CC0-1.0

My boss is fine with it ☺ But how do I annotate this information “correctly”?

Example: Astronaut Analysis
Add Copyright Holder and License Information

- **Goal:** Add license file(s) and note copyright holder(s)
- **REUSE:** Make it easy to determine what license a file is licensed under and who owns the copyright!
  - Heavily builds on SPDX and provides the reuse helper tool
  - For more information: Tutorial, FAQ, Specification

```
$ reuse annotate --copyright="German Aerospace Center" --license="MIT" code/*
Successfully changed header of code\requirements.txt
Successfully changed header of code\test.sh
Successfully changed header of code\astronaut-analysis.py
```

```
# SPDX-FileCopyrightText: 2023 German Aerospace Center
#
# SPDX-License-Identifier: MIT

""" This script analysis the astronaut data set and creates different plots as result. """
```

RELEASE

Release basics

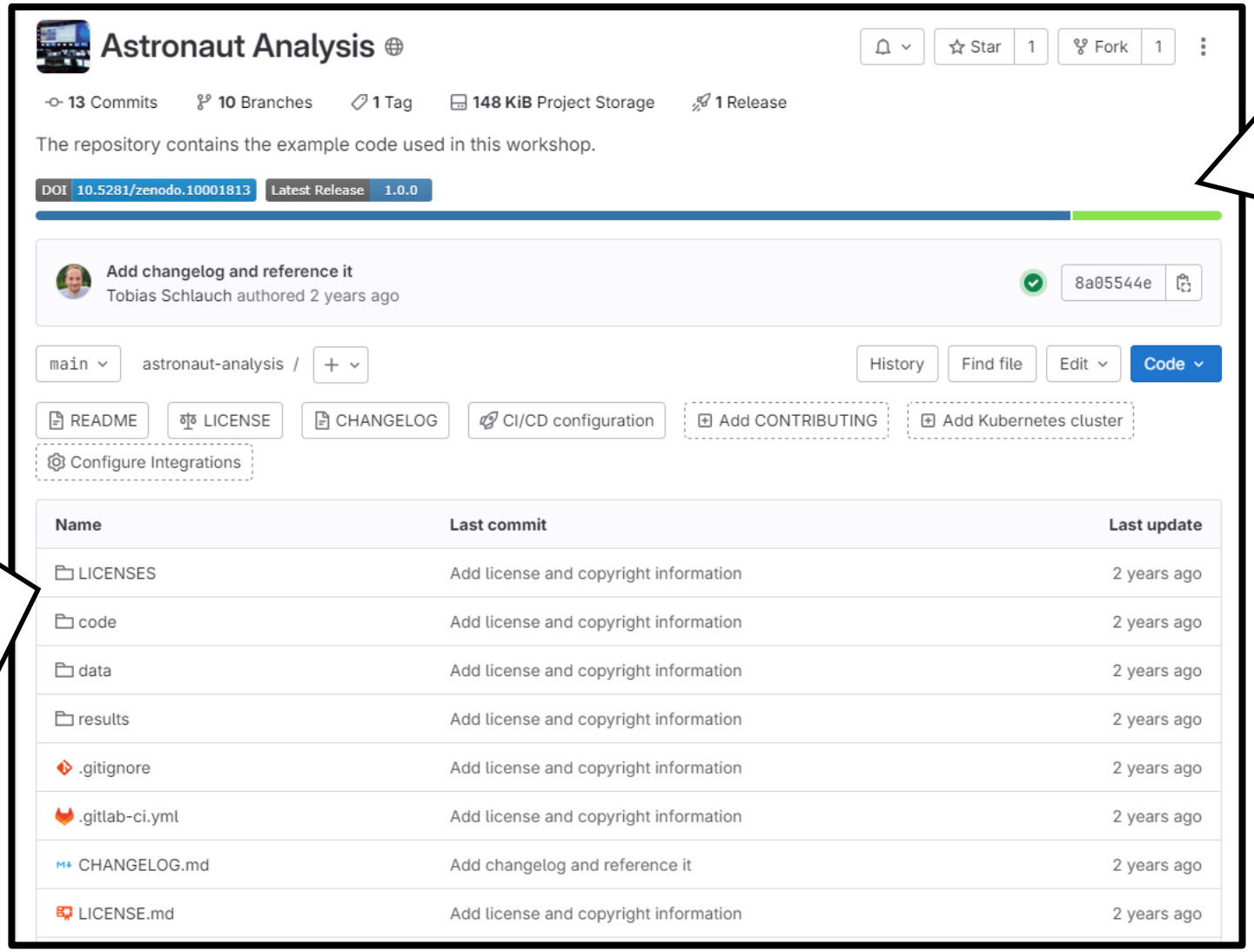
- A **release** is a specific working software version
- The **release number** uniquely identifies the release (e.g., 1.0.1 or 2022-03-17)
- A user uses the **release package** to install and use the released software:
  - Contains code + documentation
  - Simplest form: snapshot of your source code repository packaged as Zip file
- Important **changes between releases** are documented in a changelog

What do I have to do?

1. **Prepare your code for release**
   a) Define the release number
   b) Update the documentation and citation metadata
2. **Check your code**
3. **Publish and archive the release**
   a) Mark the release in the source code repository using a tag
   b) Create the release package
   c) Archive the release package in the publication repository

Astronaut Analysis Release 1.0.0

License information for code, data, results properly annotated via REUSE



Release 1.0.0 marked as **Git tag** in the repository

Summary

- Step 1: Put your code under version control
- Step 2: Make sure that your code is in a sharable state
- Step 3: Add essential documentation
- Step 4: Add a license
- Step 5: Release your code

Thank you!

What are your Questions?

Email: carina.haupt@dlr.de
Mastodon: https://scholar.social/@caha42

Impressum

- Content created based on DLR/HIFIS training “Foundations of Research Software Publication” and example project “Astronaut Analysis”
  - https://codebase.helmholtz.cloud/hifis/software/education/hifis-workshops/foundations-of-research-software-publication/workshop-materials
  - https://codebase.helmholtz.cloud/hifis/software/education/hifis-workshops/foundations-of-research-software-publication/astronaut-analysis

Copyright and License Information