CodeRefinery mini-workshop

We will together create a simple image processing pipeline that is reproducible, reusable, extensible, documented, and tested. Celebrating Eileen Collins, the first woman to pilot the Space Shuttle and the first to command a Space Shuttle mission.

- Example project: https://github.com/coderefinery/imgfilters
- Inspiration for this example: scikit-image
- This is a mini-version of the CodeRefinery workshop (6 half-days) which we teach twice a year.

Goals of this workshop

- We will together create a simple image processing pipeline that is reproducible, reusable, extensible, documented, and tested.
- We will start with something relatively simple and gradually add the missing pieces: all the way towards making it publishable.
- We will use a Python example but we will not need Python experience to follow and not focus on Python programming but rather on software development practices.

We will explore these in 3 sessions:

- Session 1: **Reproducible code changes**. Keep track of code changes and learn how we can collaborate on code using Git and GitHub.
- Session 2: Add **code documentation** and **testing**. Improve **structure**.
- Session 3: Document **dependencies** and prepare the code to be **shared**, reused, and published.

💬 Learner personas

- You write code for their own research: alone or in a small collaboration.
- You want to be able to contribute to open-source projects.
- You do data analysis and use (import) code written by others in a Jupyter Notebook (or similar) or a script.

- You neither write not use code but you want to better understand how reproducible research software is created and what matters to make it reusable.

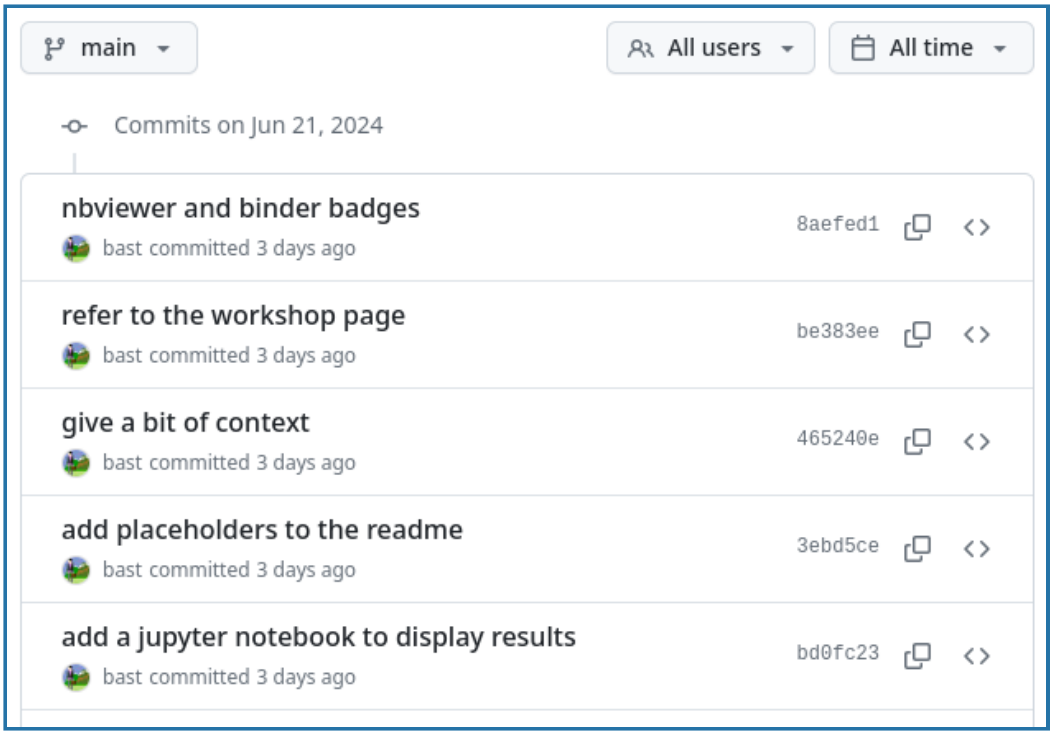
⚙ Preparation

- GitHub account if you want to actively participate.
- Enable Rich Jupyter Notebook Diffs on GitHub for a better code review experience when changing Jupyter Notebooks (click on your avatar top-right on GitHub, then "Feature preview").

Motivation for version control

Git is all about keeping track of changes

Below are screenshots of **tracked changes with Git** (from our example repository):



Web browser, GitHub view

```
commit 8aefed153328230092df31775dcd2388721bb862 (HEAD -> main,
 origin/main, origin/HEAD)
Author: Radovan Bast <bast@users.noreply.github.com>
Date:   Fri Jun 21 23:11:01 2024 +0200

    nbviewer and binder badges

commit be383eedcb4d8c2bfee00955760ce32baee1e6c3
Author: Radovan Bast <bast@users.noreply.github.com>
Date:   Fri Jun 21 23:06:43 2024 +0200

    refer to the workshop page

commit 465240e5022d94bcf870ff2199aa179dc64f801f
Author: Radovan Bast <bast@users.noreply.github.com>
Date:   Fri Jun 21 22:27:37 2024 +0200

    give a bit of context

commit 3ebd5cec15c43dfa3c36158951618eaf6757627c
Author: Radovan Bast <bast@users.noreply.github.com>
Date:   Fri Jun 21 19:16:51 2024 +0200

    add placeholders to the readme

commit bd0fc233fec3eb97be7d69a8951b965412393e21
Author: Radovan Bast <bast@users.noreply.github.com>
Date:   Fri Jun 21 18:54:00 2024 +0200

    add a jupyter notebook to display results
```

The same as above, but the terminal view

💬 Discussion

- Commits are like snapshots of the repository at a certain point in time.
- Commits carry metadata about changes: author, date, commit message, and a checksum.

Why do we need to keep track of versions?

Version control is an answer to the following questions (do you recognize some of them?):

- “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?”
- “My laptop is gone. Is my thesis now gone?”

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
  - Different people can work on the same code/project without interfering
  - You can experiment with an idea and discard it if it turns out to be a bad idea

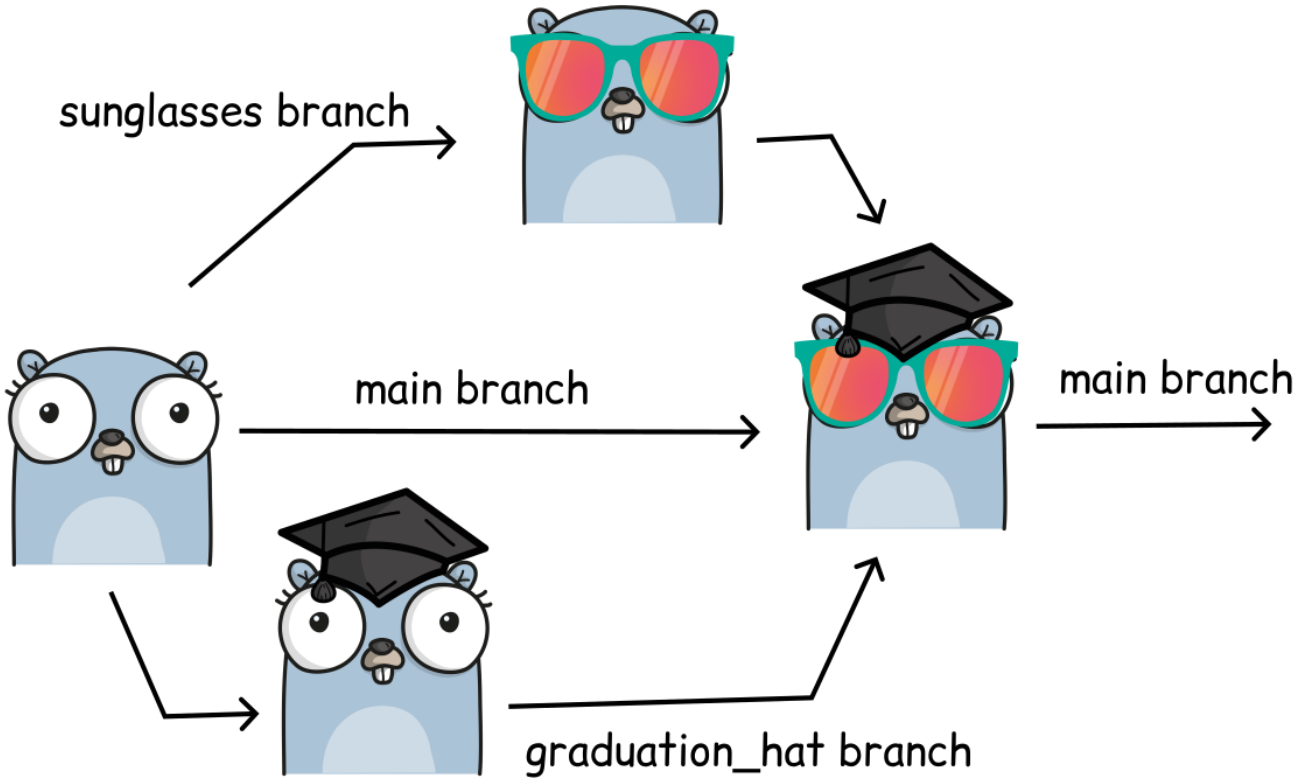


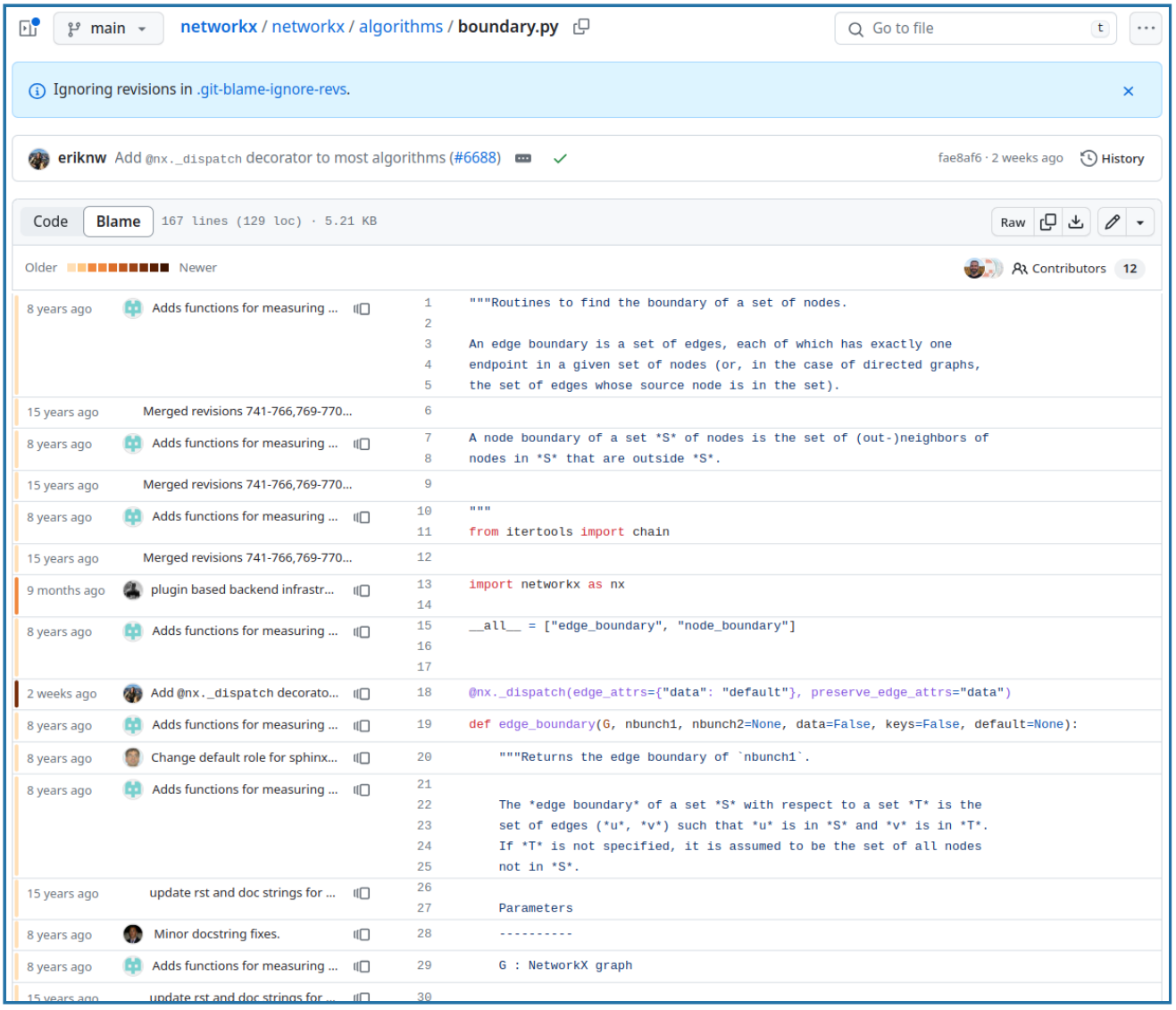
Image created using https://gopherize.me/ (inspiration).

- **Collaboration**: review, compare, share, discuss
- Example network graph

Reproducibility

- **Someone asks you about your results from 5 years ago. Can you get the same results now?**
- How do you indicate which version of your code you have used in your paper?
- When you find a bug, how do you know **when precisely** this bug was introduced (Are published results affected? Do you need to inform collaborators or users of your code?).

With version control we can "annotate" code (browse this example online):

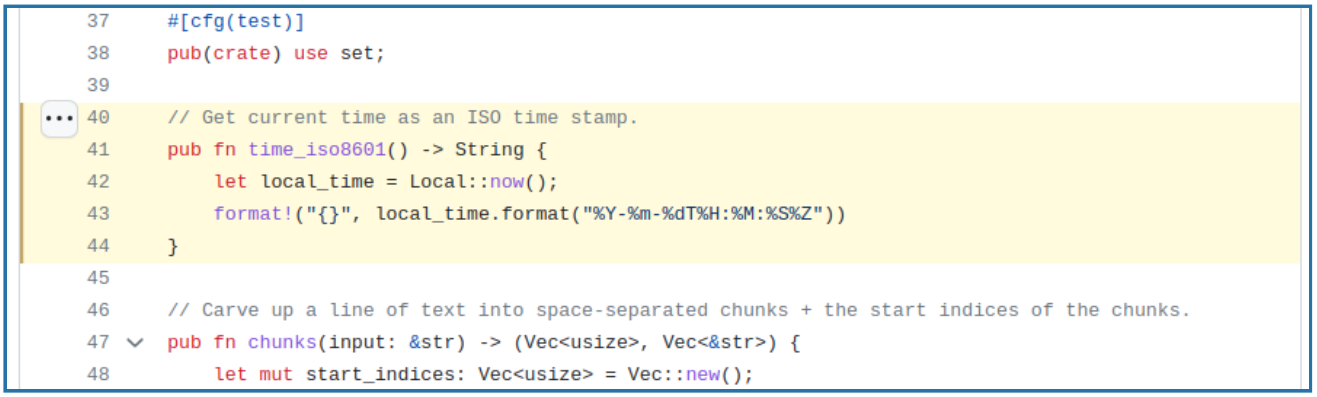


Example of a git-annotated code with code and history side-by-side.

Talking about code

You want to show someone a few lines from one of your projects. Which of these two is more practical?

- "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:



Permalink that points to a code portion.

What we typically like to snapshot

- Software (this is how it started but Git/GitHub can track a lot more)
- Scripts
- Notebooks
- Documents (plain text files much better suitable than Word documents)
- Manuscripts (Git is great for collaborating/sharing LaTeX or Quarto manuscripts)
- Configuration files
- Website sources
- Data

Copy and browse an existing project

❗ Objectives

- We will fork the example project.
- We will find a point in history when a code default changed (using `git annotate`).
- We will create a branch and a commit with a new file added to `exercise` directory.

Repositories and branches

What is a **Git repository**?

- Git repository contains a bunch of files and tracks their changes over time.
- Git repositories typically reside on GitHub or GitLab or on your computer or on some server.
- Git repositories track changes using commits.

What are **branches**?

- Commits form a history of changes and the history can be visualized as a tree which contains branches.
- Each repository has a default branch. It is often called `main` or `master`.
- Branches are like parallel universes where you can experiment with changes without affecting the default branch.
- Browse the network of branches in our example repository ("Insights" -> "Network"): https://github.com/coderefinery/imgfilters/network

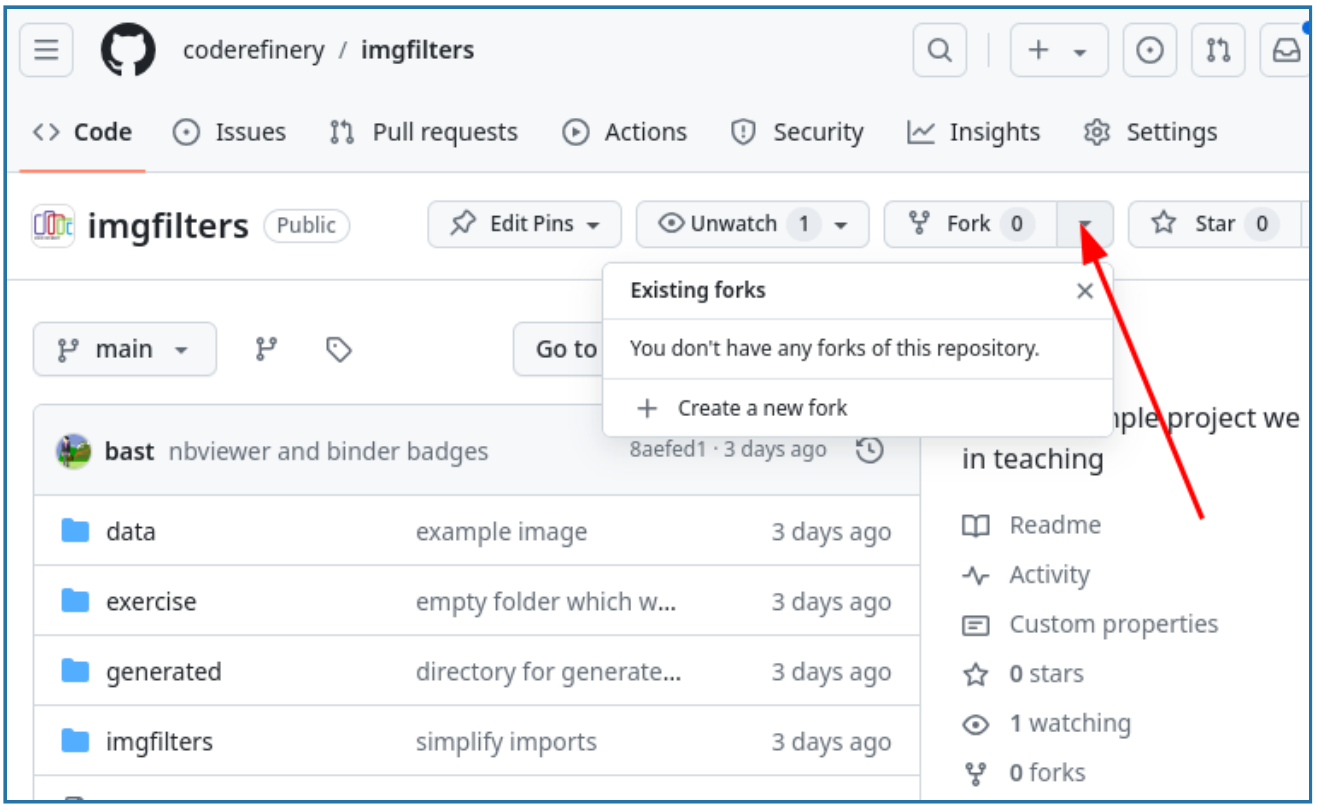
Annotated history

✍ Let us try the annotation feature ("git blame")

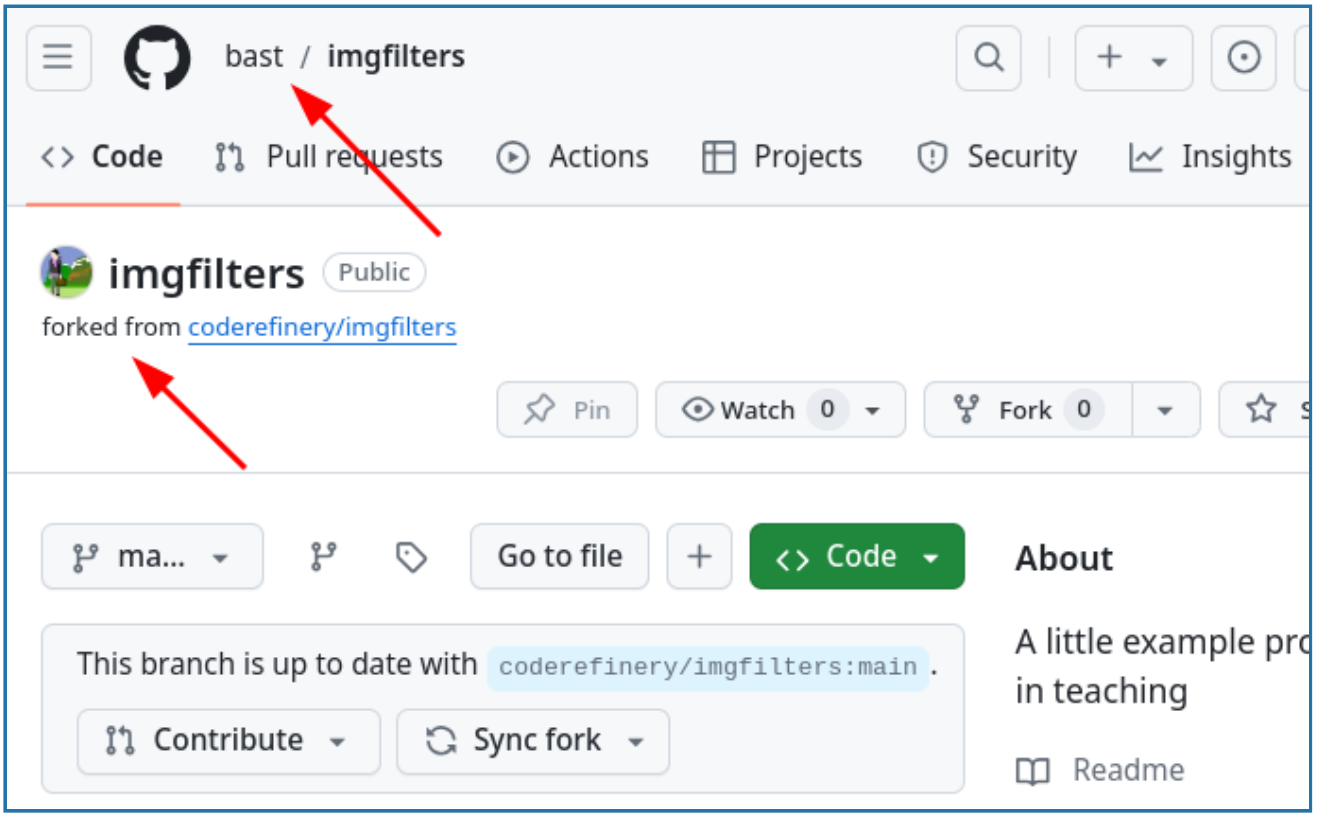
- The function gaussian_smoothing has a default value for sigma. But this default used to be different.
- Find out **when** this default value was changed (which commit).
- Would you be able to do this archaeology work with your project?
- Is the commit message informative about **why** this was changed?

What is a fork? Let's create and browse it!

- A fork is a full copy of a repository.
- You have then write permissions to your copy.
- Once a fork is created, you decide when to sync changes from the original repository - it does not happen automatically.



Creating a fork on GitHub



Verify that you are on your fork

Exercise/demo: creating a new branch and a new commit

You can try this or you watch the instructor doing it.

✍ Creating a new branch and a new commit

- Create a new branch in your fork and give it a descriptive name.
- Add a new file in the `exercise` directory to the new branch.
- In this file we will try to answer: "What do you know now about programming that you wish you knew when you started?".
- The new branch and new file now only exist on the fork, not yet in the original repository.

How does this relate to your own work?

Summary

- Forking is useful if you want to modify a repository that you do not have write access to or if you want to allow others to experiment with your repository.
- Within your research project you probably want to give all project members write access and then you can work within the same repository.
- If you are more than 1 person working on a project, it can be useful to make the `main` branch protected and to always create a new branch for each new feature or bug-fix or idea.
- In the next episode (Collaboration and code review using issues and pull requests) we will discuss how to suggest changes from a fork or within the same repository.

More resources

In this short workshop we focus on **what is possible** and **what are good practices** but we cannot focus on **how to do these in detail** across the many user interfaces (GitHub web, GitLab web, command line, VS Code, other editors).

- Here we go through those in detail: Introduction to version control with Git

Collaboration and code review using issues and pull requests

Objectives

- Using issues as a way to discuss and plan work.
- Using pull requests as a way to review and discuss code changes.

What is a pull request (GitHub) or merge request (GitLab)?

- Think of merge requests as **change proposals**.

- Conceptually, they are similar to “suggesting changes” in Google Docs: They can be commented on and accepted (merged).

🌻 Five nice things about pull requests which are often misunderstood

1. **Issues don’t have to be only about bugs**: Open one to share an idea before you start coding and collect feedback. Later cross-reference the issue in the pull request.
2. **Reviewing code is mainly for ~~quality assurance~~ collaborative learning**: It is mainly for knowledge transfer. Two persons know about a change instead of one. Both have a chance to learn something new.
3. **Modifying an open pull request** does not require closing and opening a new one. Pull requests are from a source branch to a target branch. You can add new commits to the source branch and the pull request will be updated automatically. This also means: Always create a new branch for each new pull request.
4. **Draft pull requests** are a nice way to inform about unfinished work and to collect feedback early.
5. **Reviewing changes in Jupyter Notebooks** can be pleasant. Enable Rich Jupyter Notebook Diffs.

Exercise/demo

The proof of the pudding is in the eating. Let’s try the above in two pull requests:

✍️ Exercise

1. Open an issue, describe an idea, then open a pull request that cross-references the issue, then improve the pull request based on feedback (as an example, we can try to change one of the colors in the warhol_effect filter).
2. Compare the following “diff” between two branches that modifies our Notebook. View it with and without the Rich Jupyter Notebook Diffs feature enabled (click on your avatar top-right on GitHub, then “Feature preview”): https://github.com/coderefinery/imgfilters/compare/8aefed1..4dbae92

How to organize your own project

How large should a commit be?

- Better too small than too large (easier to combine than to split).
- Smaller sized commits may be easier to review for others than huge commits.
- Imperfect commits are better than no commits.

Writing useful commit messages: Useful commit messages summarize the change and provide context.

What level of branching complexity is necessary?

- Simple **personal projects**: Typically start with just the `main` branch. Use branches when you are not sure about a change.
- Projects with **few persons**: Create a new branch for each new feature or bug fix. Changes are reviewed by others. Consider to write-protect the `main` branch so that it can only be changed with pull requests or merge requests.

More resources

- CodeRefinery lesson on collaborative Git

How to turn your project to a Git repo and share it

Objectives

- Turn our own coding project (small or large, finished or unfinished) into a Git repository.
- Be able to share a repository on the web to have a backup or so that others can reuse and collaborate or even just find it.

Exercise

✍ Optional exercise/homework: Turn your project to a Git repo and share it

1. Create a new directory called **myproject** with one or few files in it. This represents our own project. It is not yet a Git repository. You can try that with your own project or use a simple placeholder example.
2. Turn this new directory into a Git repository.
3. Share this repository on GitHub (or GitLab, since it really works the same).

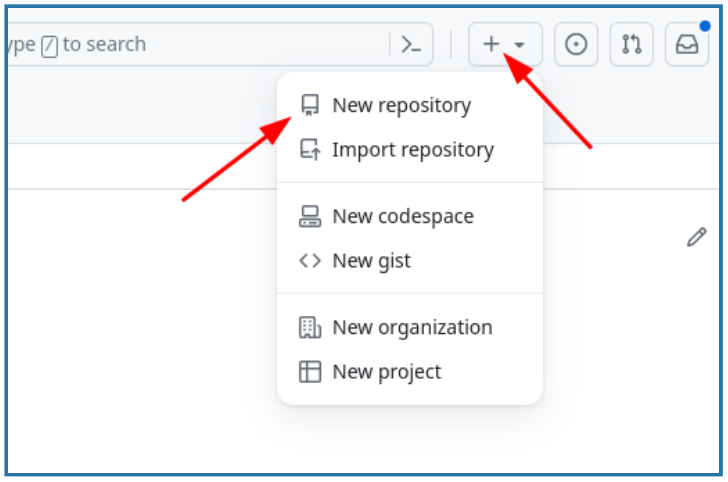
We offer **three different paths** of how to do this exercise.

- Via **GitHub web interface**: easy and can be a good starting point if you are completely new to Git.
- **VS Code** is quite easy, since VS Code can offer to create the GitHub repositories for you.
- **Command line**: you need to create the repository on GitHub and link it yourself.

Only using GitHub | VS Code | Command line | RStudio

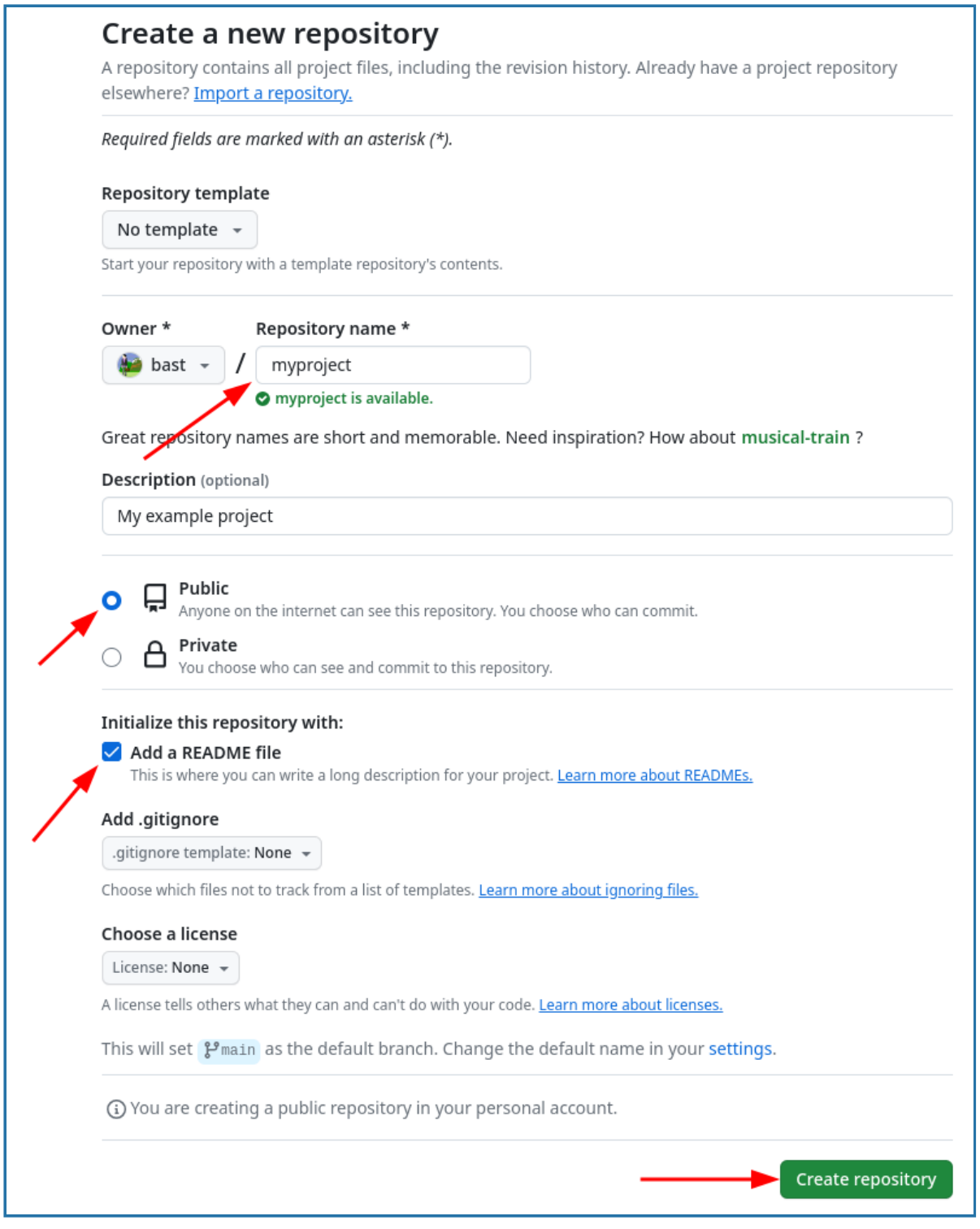
Create an repository on GitHub

First log into GitHub, then follow the screenshots and descriptions below.



Click on the "plus" symbol on top right, then on "New repository".

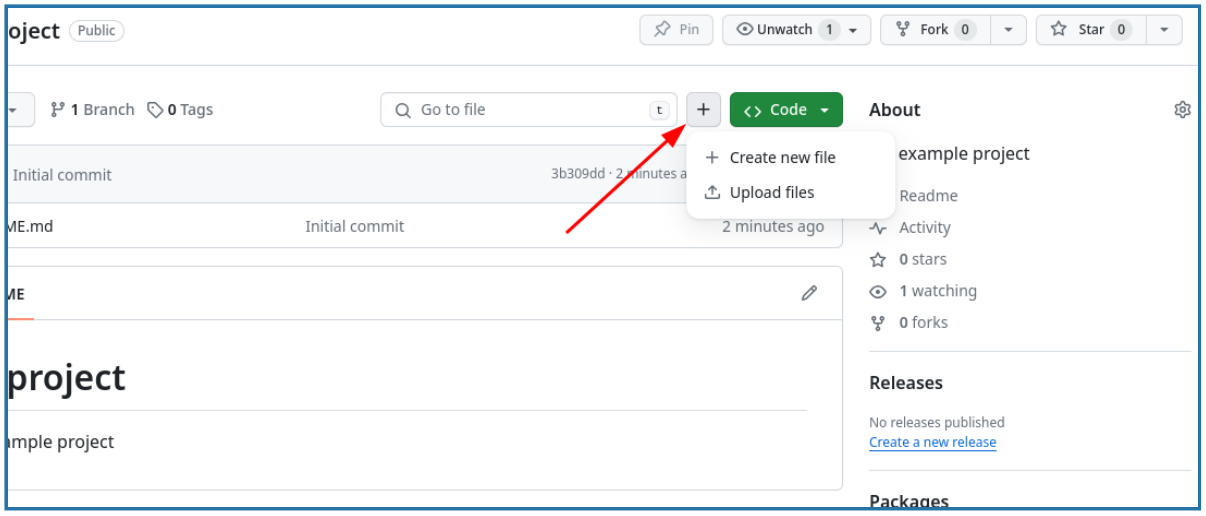
Then:



*Choose a repository name, add a short description, and in this case **make sure to check** "Add a README file". Finally "Create repository".*

Upload your files

Now that the repository is created, you can upload your files:



Click on the "+" symbol and then on "Upload files".

Remote repositories

In this exercise we have pushed our local repository to a remote repository. You can learn how to work with remote repositories in detail in the collaborative distributed version control lesson.

To store your Git data on another server, you use **remotes**. A remote is a repository on its own, with its own branches. We can **push** changes to the remote and **pull** from the remote.

You might use remotes to:

- Back up your own work or make your work findable.
- To collaborate with other people.

There are services that can be a remote:

- If you have a server you can SSH to, you can use that as a remote.
- GitHub is a popular, closed-source commercial site.
- GitLab is a popular, open-core commercial site. Many universities have their own private GitLab servers set up.
- Bitbucket is yet another popular commercial site.
- Another option is NotABug.
- There are more …

Is putting software on GitHub/GitLab/… publishing?

It is a good first step but to make your code truly **findable and accessible**, consider making your code **citable and persistent**: Get a persistent identifier (PID) such as DOI in addition to sharing the code publicly, by using services like Zenodo or similar services.

We will practice this together in the episode How to publish your code.

Code documentation

Objectives

- Improve the README of our example project.
- Add Sphinx documentation to the project.
- Deploy the result to GitHub Pages.

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.

In-code documentation

Not very useful (more commentary than comment):

```
# now we check if temperature is below -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:

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

Many languages allow "docstrings"

Example (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
```

Often a README is enough - checklist

- **Purpose**
- Installation instructions
- Requirements
- **Copy-paste-able example to get started**
- Tutorials covering key functionality
- Reference documentation (e.g. API) covering all functionality
- Authors and **recommended citation**
- License
- Contribution guide

See also the JOSS review checklist.

What if you need more than 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

Exercise/demo

✍ Exercise

- Open an issue and send a pull request (referencing the issue) which improves the README of the example project.
- We will together open a pull request from a branch that adds Sphinx documentation and review it.
- We will verify that from there on documentation is built automatically by sending another pull request with a small change.

More resources

- Documentation lesson material
- Talk material "Documenting code" by S. Wittke
- Inside Sphinx we can add tables, images, equations, code snippets, ... (more information.

Optional: How to auto-generate documentation from docstrings in Python

Add the following highlighted lines to `conf.py`:

```
# Configuration file for the Sphinx documentation builder.
#
# For the full list of built-in configuration values, see the documentation:
# https://www.sphinx-doc.org/en/master/usage/configuration.html

import os
import sys

# this is here so that the package can be imported by sphinx
sys.path.insert(0, os.path.abspath(".."))

# -- Project information -----------------------------------------------------
# https://www.sphinx-doc.org/en/master/usage/configuration.html#project-information

project = "imgfilters"
copyright = "2024, Authors"
author = "Authors"
release = "0.1"

# -- General configuration ---------------------------------------------------
# https://www.sphinx-doc.org/en/master/usage/configuration.html#general-configuration

exclude_patterns = ["_build", "Thumbs.db", ".DS_Store"]

extensions = [
    "myst_parser",  # in order to use markdown
    "sphinx.ext.autodoc",  # to automatically document the code from docstrings
    "sphinx.ext.napoleon",  # to parse numpy-style docstrings
]

myst_enable_extensions = [
    "colon_fence",  # ::: can be used instead of ``` for better rendering
]

# -- Options for HTML output -------------------------------------------------
# https://www.sphinx-doc.org/en/master/usage/configuration.html#options-for-html-output

html_theme = "sphinx_rtd_theme"
```

Instead of one command to build the documentation we now need two:

```
$ sphinx-apidoc -o doc imgfilters
$ sphinx-build doc _build
```

Add to `index.md`:

```
:::{toctree}
:maxdepth: 2
:caption: API documentation

modules.rst
:::
```

Automated testing: from unit tests to end-to-end tests

Objectives

- Get an overview of possibilities for automated testing.
- Add a unit test to a function.
- Add an end-to-end test or discuss how it could look.
- Make the testing part of code review.
- Discuss where to start in your own project.

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

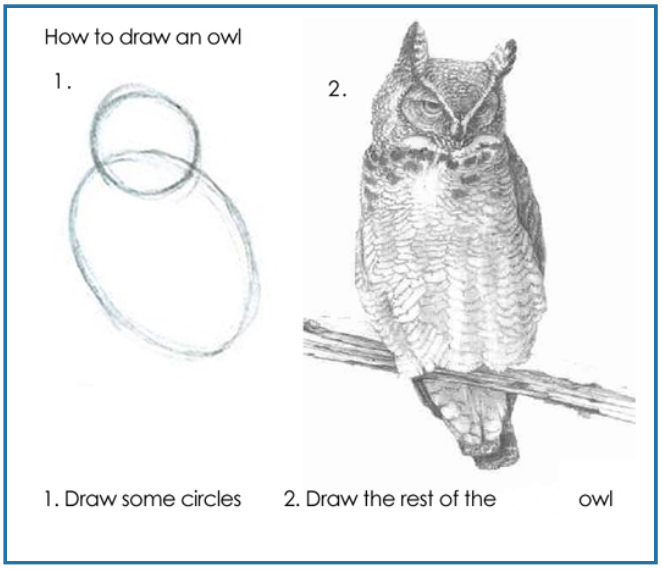
- **Less scary to change code**: tests will tell you whether something broke.
- Unit tests can **guide towards better structured code**: complicated code is more difficult to test.
- **Easier for new people** to join.
- Easier for somebody to **revive an old code**.

How testing is often taught

```
def add(a, b):
    return a + b


def test_add():
    assert add(1, 2) == 3
```

How this feels:



[Citation needed]

Instead, we will look at and **discuss a real example** where we test components from our image processing project.

Exercise/demo

✍ Exercise

- We will together open a pull request from a branch that implements testing and review it.
- We will verify that from there on tests are run automatically.
- We will try to open another pull request later which would break tests (and not merge it).

💬 Discussion

- We will discuss the potential but also limitations of the code changes.
- Do we need this for a simple Notebook?

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.

Recommendations for Notebooks

- Automated testing is often too much for notebooks.
- Instead you can test modules that you import in the notebook.
- What is often more useful is to provide an example input and show how the result should look like. Good example: scikit-image

More resources

- Software testing lesson material

How to structure the code as it grows

Objectives

- Discuss the advantages of structuring code into functions and files.
- Add a command-line interface to the code and discuss its advantages.

From no functions to functions

- Many projects start with a single file and all code is in the global scope (meaning there are no functions).
- A bit later we often introduce functions into the code.

Motivation to structure code into functions:

- Less code repetition.
- Functions can make the overall code flow more readable and easier to follow. Example: https://github.com/coderefinery/imgfilters/blob/main/example.py
- They allow us to encapsulate details and complexity.
- We can make code reusable across projects and notebooks.
- Code becomes easier to test.
- Unintended side effects are less likely to occur.
- For languages with garbage collection, functions can help to manage memory.

Prefer pure functions

These are functions without side-effects, meaning they do not modify any global variables or have any other side-effects.

Functions without side-effects are easier to understand and to copy-paste into other projects.

Examples for impure functions:

- Functions which modify global variables.

- Functions which modify input data.
- Functions which read from or write to files or databases.

From one file to multiple files

Motivation to split code into multiple files:

- When the one file grows long and becomes hard to navigate.
- When we want to reuse some functions in other projects.
- We can browse https://github.com/coderefinery/imgfilters/network and find the point where that happened.

Exercise/demo: Adding a command-line interface

Adding a command-line interface (CLI) to a script is an often undervalued "superpower" of programming.

✍ Exercise

- We will together add a command-line interface to our example code.

💬 Discussion

- We will discuss the advantages of doing this:
  - **Easier to use**: Give the user the freedom to change data and settings without having to modify the code.
  - **Easier to parallelize**: Give the user the freedom to choose their parallelization strategy.
  - **Easier to automate**: As part of pipeline/workflow tools.

Here is an example solution using Argparse:

```
from imgfilters.filters import pixelate
from imgfilters.file_io import read_image, save_image

import argparse


def parse_arguments():
    parser = argparse.ArgumentParser(description="Pixelate image")
    parser.add_argument(
        "--input", type=str, required=True, help="Path to the input image"
    )
    parser.add_argument(
        "--output", type=str, required=True, help="Path to the output image"
    )
    parser.add_argument(
        "--scale", type=float, default=0.05, help="Scale for pixelation"
    )

    return parser.parse_args()


args = parse_arguments()


image = read_image(args.input)
image_pixelated = pixelate(image, scale=args.scale, num_colors=8)
save_image(image_pixelated, args.output)
```

Instead of Argparse, we could have used Click or docopt or ...

More resources

- Command Line Interface Guidelines

Reproducible dependencies, environments, and workflows

Objectives

- Learn how to document dependencies in Python.
- Discuss how to document dependencies in other languages.
- Each (non-trivial) Python project from now on should have a `requirements.txt` or `environment.yml` file.

Have you ever experienced the following?

1. You try to run a code or notebook and it cannot find a module?

```
Traceback (most recent call last):
  File "/home/user/imgfilters/example.py", line 1, in <module>
    from imgfilters.filters import (
  File "/home/user/imgfilters/imgfilters/filters.py", line 1, in <module>
    import numpy as np
ModuleNotFoundError: No module named 'numpy'
```

2. You have installed the module but the code still does not work since it needs a different version of the module?
3. “It works on my machine 🤷” but not on your machine?

💬 Discussion

- How do you install and use dependencies in your projects?
- How do you document them?

Tracking dependencies in Python

There are two standard files to document dependencies in Python. Either use `requirements.txt` (together with pip) or `environment.yml` (Conda).

Let us study the requirements.txt from our project.

- How would we “pin” the versions?
- When should we “pin” the versions?

How would the corresponding `environment.yml` look like?

```
name: imgfilters
channels:
  - conda-forge
dependencies:
  - python=3.12
  - numpy
  - scikit-image
  - scikit-learn
  - pywavelets
  - matplotlib
  - jupyterlab
```

What if you don’t remember which dependencies you have installed?

Conda:

```
# list all dependencies and write them to file
$ conda env export > environment.yml

# omit detailed build information
$ conda env export --no-builds > environment.yml

# only list dependencies that were explicitly installed, not their dependencies
$ conda env export --from-history > environment.yml
```

pip:

```
$ python -m pip freeze > requirements.txt
```

My recommendation to install dependencies and document them at the same time:
Document them first and then install from file:

```
$ conda env create -f environment.yml
$ pip install -r requirements.txt
```

What if I need to record the entire operating system?

One solution can be: Containers (Apptainer/Singularity, Docker, ...) but the details of this are outside the scope of this short course.

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]

More reading

- Reproducible research
- The Turing Way: Guide for Reproducible Research

- Ten simple rules for writing Dockerfiles for reproducible data science
- Computing environment reproducibility

Choosing a software license

Objectives

- Knowing about what derivative work is and whether we can share it.
- Get familiar with terminology around licensing.
- We will add a license to our example project.

Copyright and derivative work: Sampling/remixing

[Midjourney, CC-BY-NC 4.0]

[Midjourney, CC-BY-NC 4.0]

- Copyright controls whether and how we can distribute the original work or the **derivative work**.
- In the **context of software** it is more about being able to change and **distribute changes**.
- Changing and distributing software is similar to changing and distributing music
- You can do almost anything if you don't distribute it

Often we don't have the choice:

- We are expected to publish software
- Sharing can be good insurance against being locked out

Can we distribute our changes with the research community or our future selves?

Why software licenses matter

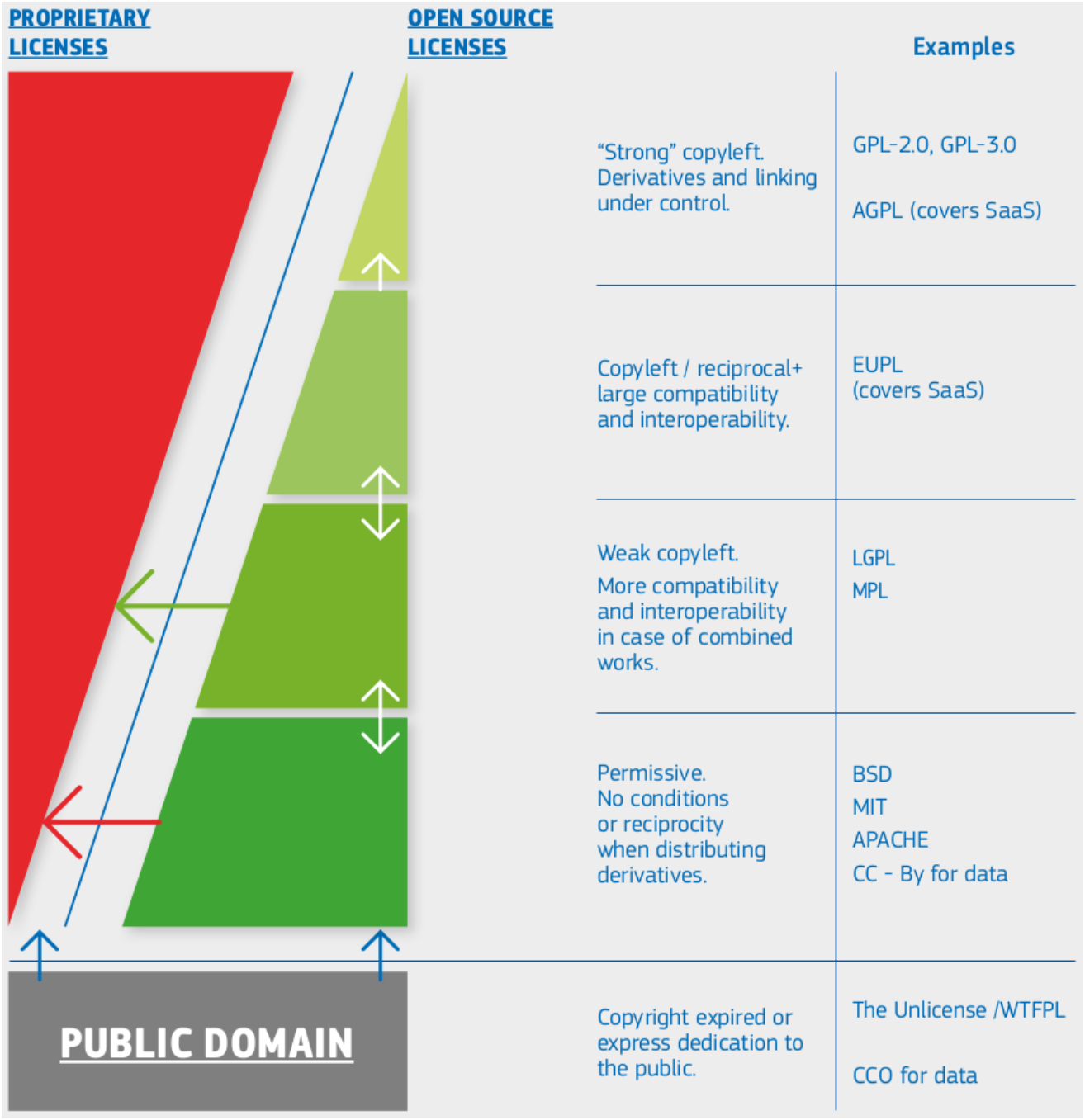
You find some great code that you want to reuse for your own publication.

- This is good for the original author - you will cite them. Maybe other people who cite you will cite them.
- You modify and remix the code.
- Two years later … ⏳
- Time to publish: You realize **there is no license to the original work** 😱

Now we have a problem:

- 😬 "Best" case: You manage to publish the paper without the software/data. Others cannot build on your software and data.
- 😱 Worst case: You cannot publish it at all. Journal requires that papers should come with data and software so that they are reproducible.

Taxonomy of software licenses



European Commission, Directorate-General for Informatics, Schmitz, P., European Union Public Licence (EUPL): guidelines July 2021, Publications Office, 2021, https://data.europa.eu/doi/10.2799/77160

Comments:

- Arrows represent compatibility (A -> B: B can reuse A)
- Proprietary/custom: Derivative work typically not possible (no arrow goes from proprietary to open)
- Permissive: Derivative work does not have to be shared
- Copyleft/reciprocal: Derivative work must be made available under the same license terms
- NC (non-commercial) and ND (non-derivative) exist for data licenses but not really for software licenses

Great resource for comparing software licenses: Joinup Licensing Assistant

- Provides comments on licenses
- Easy to compare licenses (example)
- Joinup Licensing Assistant - Compatibility Checker
- Not biased by some company agenda

Exercise/demo

✍ Exercise

- Let us choose a license for our example project.
- We will add a LICENSE to the repository.

💬 Discussion

- Our example code uses the libraries `scikit-image`, `scikit-learn`, `numpy`, `matplotlib`, and `pywavelets`. Do we need to look at their licenses? In other words, **is our project derivative work** of something else?

More resources

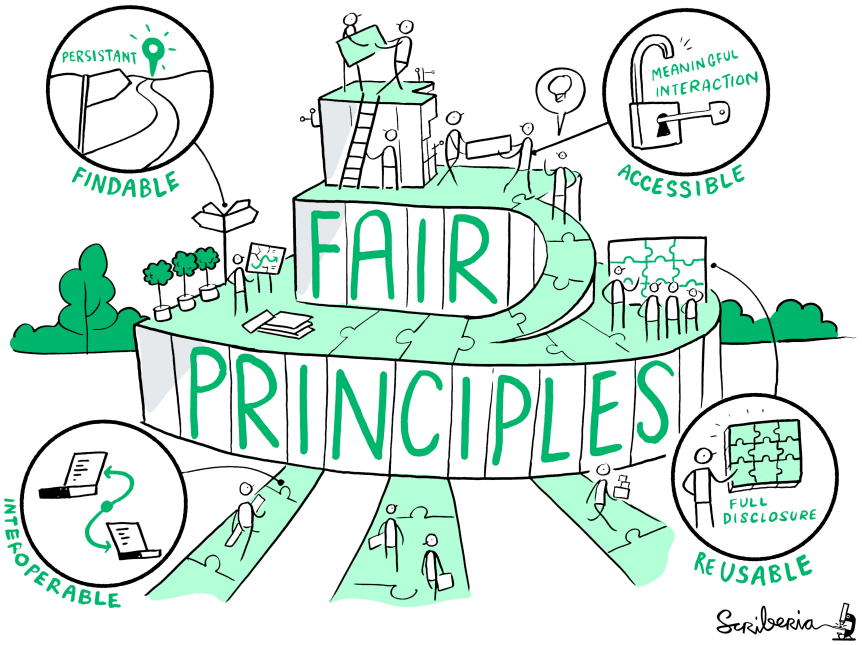
- Presentation slides “Practical software licensing” (R. Bast): https://doi.org/10.5281/zenodo.11554001
- Social coding lesson material
- UiT research software licensing guide (draft)
- Research institution policies to support research software (compiled by the Research Software Alliance)
- More reading material

How to publish your code

❗ Objectives

- Make our code citable and persistent.
- Make our Notebook reusable and persistent.

Is putting software on GitHub/GitLab/... publishing?



FAIR principles. (c) Scriberia for The Turing Way, CC-BY.

Is it enough to make the code public for the code to remain **findable and accessible**?

- No. Because nothing prevents me from deleting my GitHub repository or rewriting the Git history and we have no guarantee that GitHub will still be around in 10 years.
- **Make your code citable and persistent**: Get a persistent identifier (PID) such as DOI in addition to sharing the code publicly, by using services like Zenodo or similar services.

How to make your software citable

💬 Discussion (Citation-1): Explain how you currently cite software

- Do you cite software that you use? How?
- If I wanted to cite your code/scripts, what would I need to do?

Checklist for making a release of your software citable:

- Assigned an appropriate license
- Described the software using an appropriate metadata format
- Clear version number
- Authors credited
- Procured a persistent identifier
- Added a recommended citation to the software documentation

This checklist is adapted from: N. P. Chue Hong, A. Allen, A. Gonzalez-Beltran, et al., Software Citation Checklist for Developers (Version 0.9.0). Zenodo. 2019b. (DOI)

Our practical recommendations:

- Add a file called CITATION.cff (example).

- Get a digital object identifier (DOI) for your code on Zenodo (example).
- Make it as easy as possible: clearly say what you want cited.

This is an example of a simple `CITATION.cff` file:

```yaml
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
```

More about `CITATION.cff` files:

- GitHub now supports CITATION.cff files
- Web form to create, edit, and validate CITATION.cff files
- Video: "How to create a CITATION.cff using cffinit"

Papers with focus on scientific software

Where can I publish papers which are primarily focused on my scientific software? Great list/summary is provided in this blog post: "In which journals should I publish my software?" (Neil P. Chue Hong)

How to cite software

Great resources

- A. M. Smith, D. S. Katz, K. E. Niemeyer, and FORCE11 Software Citation Working Group, "Software citation principles," PeerJ Comput. Sci., vol. 2, no. e86, 2016 (DOI)
- D. S. Katz, N. P. Chue Hong, T. Clark, et al., Recognizing the value of software: a software citation guide [version 2; peer review: 2 approved]. F1000Research 2021, 9:1257 (DOI)
- N. P. Chue Hong, A. Allen, A. Gonzalez-Beltran, et al., Software Citation Checklist for Authors (Version 0.9.0). Zenodo. 2019a. (DOI)
- N. P. Chue Hong, A. Allen, A. Gonzalez-Beltran, et al., Software Citation Checklist for Developers (Version 0.9.0). Zenodo. 2019b. (DOI)

Recommended format for software citation is to ensure the following information is provided as part of the reference (from Katz, Chue Hong, Clark, 2021 which also contains software citation examples):

- Creator

- Title
- Publication venue
- Date
- Identifier
- Version
- Type

Exercise/demo

✍️ Exercise

- We will add a `CITATION.cff` file to our example repository.
- We will get a DOI using the Zenodo sandbox:
  - We will log into the Zenodo sandbox using GitHub.
  - We will follow these steps and finally create a GitHub release and get a DOI.
- We will use the Binder badge on our example repository to run the Notebook in the cloud and discuss how we could make it persistent and citable.

💬 Discussion

- Why did we use the Zenodo sandbox and not the "real" Zenodo for our exercise?

More resources

- Social coding lesson material
- Sharing Jupiter Notebooks

Guide for instructors and contributors

Presentation style

When presenting the material we use the collaborative notes a lot and we use the example repository to show the concepts in practice.

Code review is the central tool to introduce and discuss changes.

For any change we create a new branch and we encourage creating new branches.

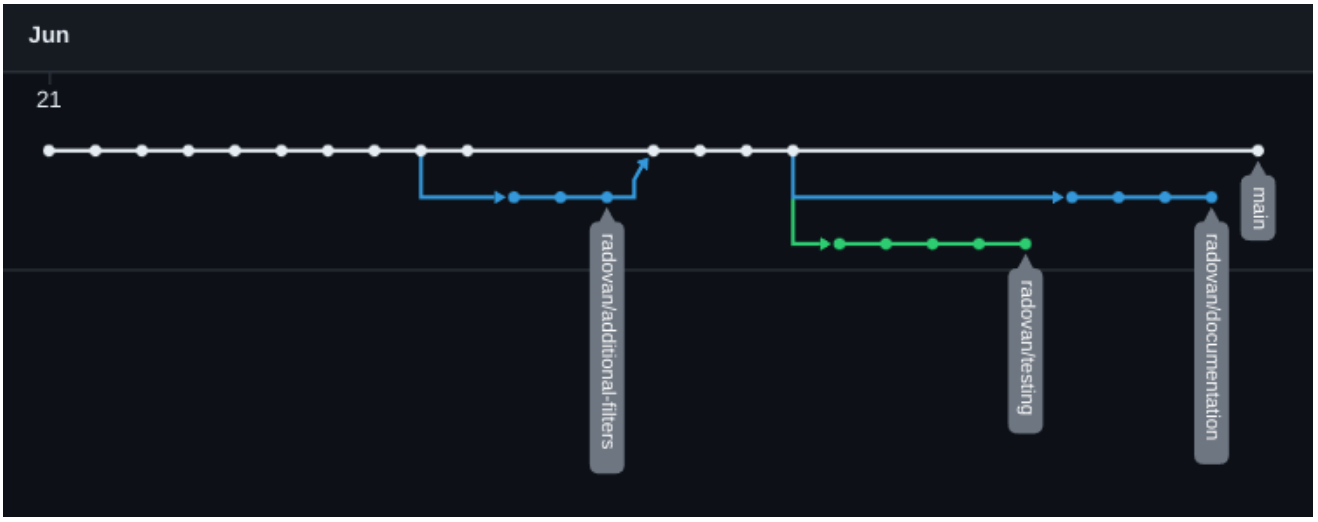
The presenter can either work within the example repository or fork it. We encourage participants to fork the repository and work on their own fork.

We encourage participants to open issues and send pull requests but the course assumes that some participants will only follow.

Preparation

Prepare a collaborative notes document which will be the central entry point.

Prepare the example repository. During the course the repository will receive pull requests and merged and before the next course it may need to be reset to the initial state (screenshot below).



Branch network of the example repository before the course.