Reproducible research - Preparing code to be usable by you and others in the future

Have you ever spent days **trying to repeat the results from few weeks or months ago**? Or you have to do paper revisions, but you just can’t get the results to match up? It’s unpleasant for both you and science.

In this lesson we will explore different methods and tools for better reproducibility in research software and data. We will demonstrate how version control, workflows, containers, and package managers can be used to **record reproducible environments and computational steps** for our future selves and others.

Learning outcomes

By the end of this lesson, learners should:

- Be able to apply well organized directory structure for their project
- Understand that code can have dependencies, and know how to document them
- Be able to document computational steps, and have an idea when it can be useful
- Know about use cases for containers

Prerequisites

You need to install Git, Python, and Snakemake (part of CodeRefinery Conda environment).

If you wish to follow in the terminal and are new to the command line, we recorded a short shell crash course.

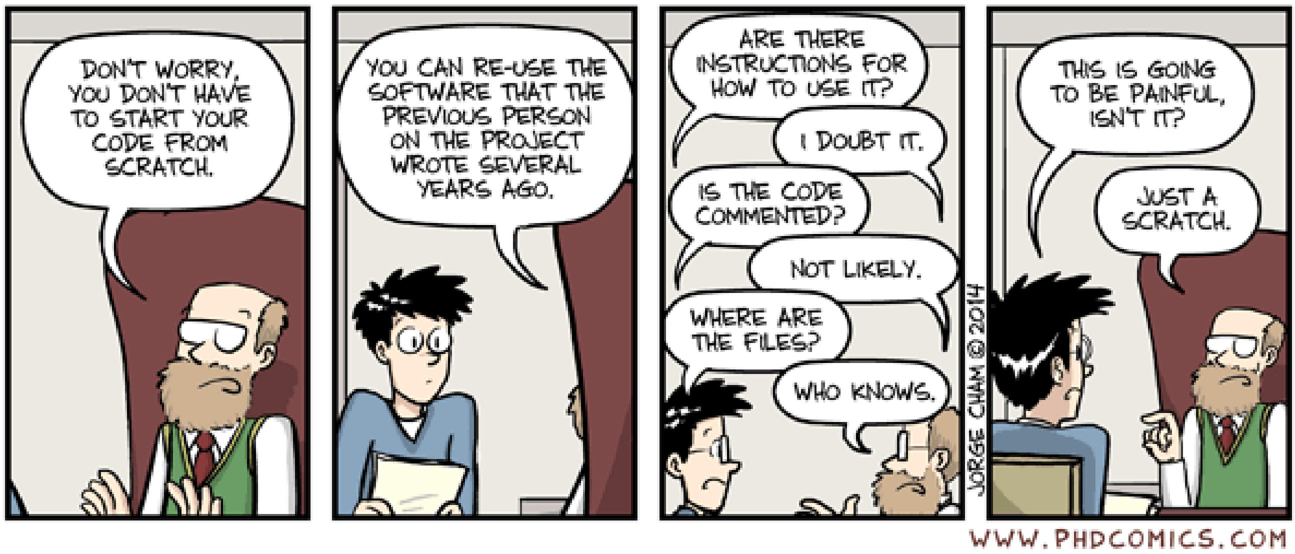
Motivation

Objectives

- Understand why we are talking about reproducibility in this workshop

Instructor note

- 5 min teaching/discussion

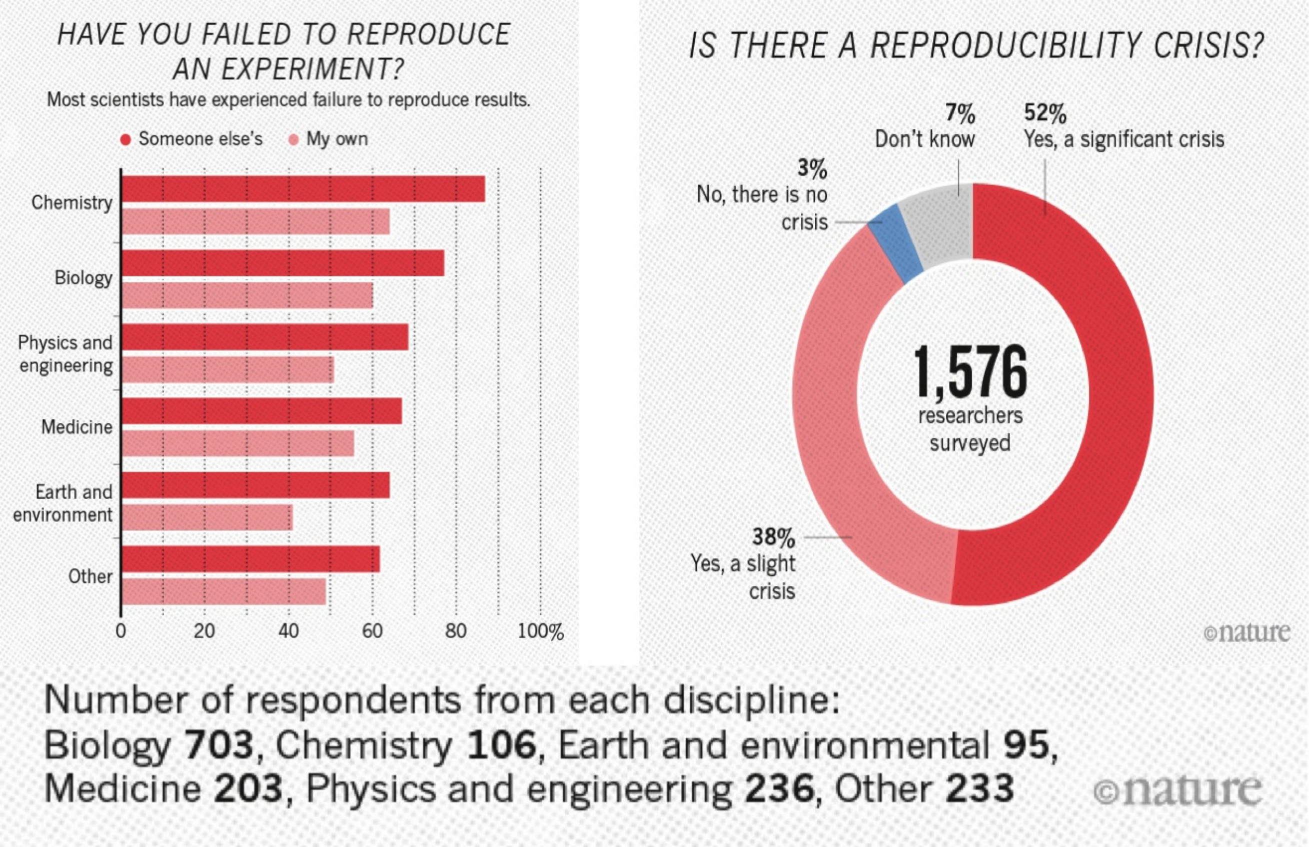

DON'T WORRY, YOU DON'T HAVE TO START YOUR CODE FROM SCRATCH.
YOU CAN RE-USE THE SOFTWARE THAT THE PREVIOUS PERSON ON THE PROJECT WROTE SEVERAL YEARS AGO.
ARE THERE INSTRUCTIONS FOR HOW TO USE IT?
I DOUBT IT.
IS THE CODE COMMENTED?
NOT LIKELY.
WHERE ARE THE FILES?
WHO KNOWS.
JORGE CHAM © 2014
THIS IS GOING TO BE PAINFUL, ISN'T IT?
JUST A SCRATCH.
WWW.PHDCOMICS.COM

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 ...

Why talking about reproducible research?

A 2016 survey in Nature revealed that irreproducible experiments are a problem across all domains of science:

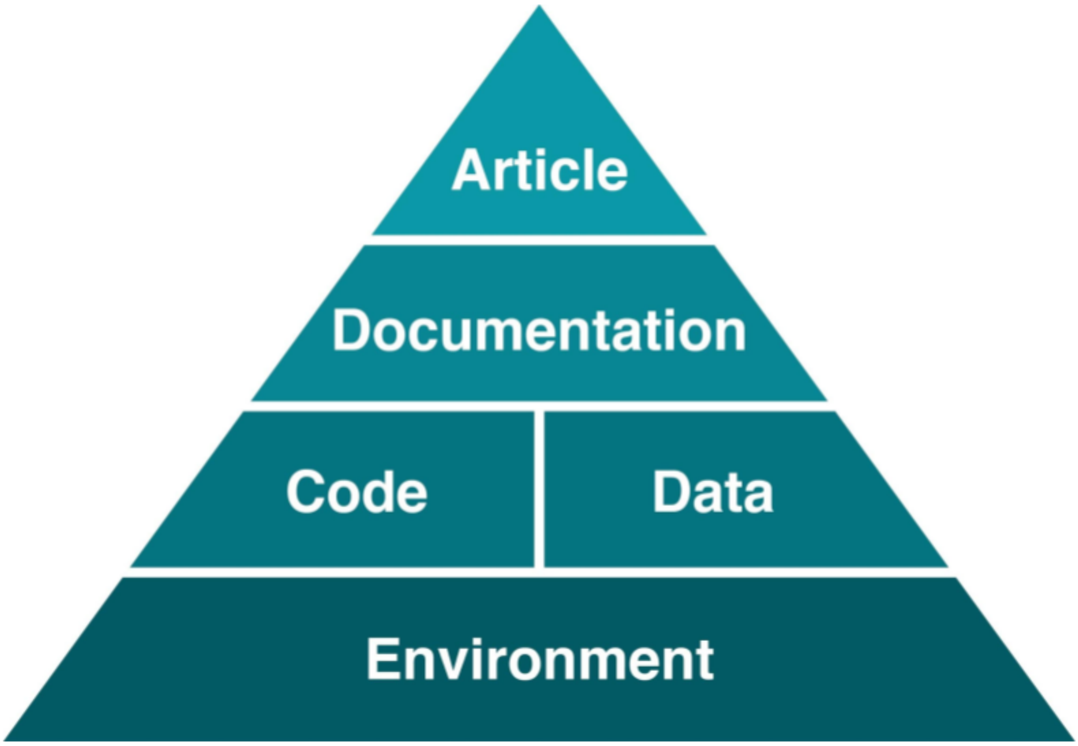


Number of respondents from each discipline: Biology **703**, Chemistry **106**, Earth and environmental **95**, Medicine **203**, Physics and engineering **236**, Other **233** ©nature

This study is now few years old but the highlighted problem did not get smaller.

Levels of reproducibility

A published article is like the top of a pyramid. It rests on multiple levels, each contributing to its reproducibility.



[Steeves, Vicky (2017) in “Reproducibility Librarianship,” Collaborative Librarianship: Vol. 9: Iss. 2, Article 4. Available at: https://digitalcommons.du.edu/collaborativelibrarianship/vol9/iss2/4]

This also means that you can think about it from the beginning of your research life cycle!

❗ Keypoints

- Without reproducibility in scientific computing, everyone would have to start a new project / code from scratch.

Organizing your projects

❗ Objectives

- Understand how to organize research projects
- What to do when and before you start a new research project at Aalto
- Get an overview of tools for collaborative and version controlled manuscripts

Instructor note

- 15 min teaching incl. discussions

Reproducible publications

Tools for collaborative writing and version control of manuscripts

On the very top of the pyramid we intoduced rests the paper that is the published result of your work. Even if it is based on a solid foundation, additional safeguards are important. -> Consider using **version control for manuscripts** as well. It may help you when keeping track of edits + if you sync it online then you don’t have to worry about losing your work.

Git **can** be used to collaborate on manuscripts written in, e.g., LaTeX and other text-based formats. However it might not always be the most convenient. Other tools exist to make the process more enjoyable:

You can **collaboratively gather notes** using self-hosted or public instances of tools like HedgeDoc and Etherpad or use online options like HackMD, Google Docs or the Microsoft online tools for easy and efficient collaboration. These tools are convenient for brainstorming or getting a quick place to share ideas also with external collaborators.

At Aalto we also have Sharepoint, which allows collaborative editing of MS Office documents, if this is your preferred format. Aalto also provides an Overleaf (LaTeX) subscription. Overleaf is a collaborative latex editor and Aalto provides several templates on the platform (e.g. Thesis templates, base Article templates etc).

The huge advantages:

- You can work in parallel without duplicating work (if using a collaborative editor)
- You have time stamps of when ideas have been put forward (when there is a contention of originality)

Executable manuscripts

You may also want to consider writing an executable manuscript using tools like Jupyter Notebooks hosted on Binder, Quarto, Authorea or Observable, to name a few.

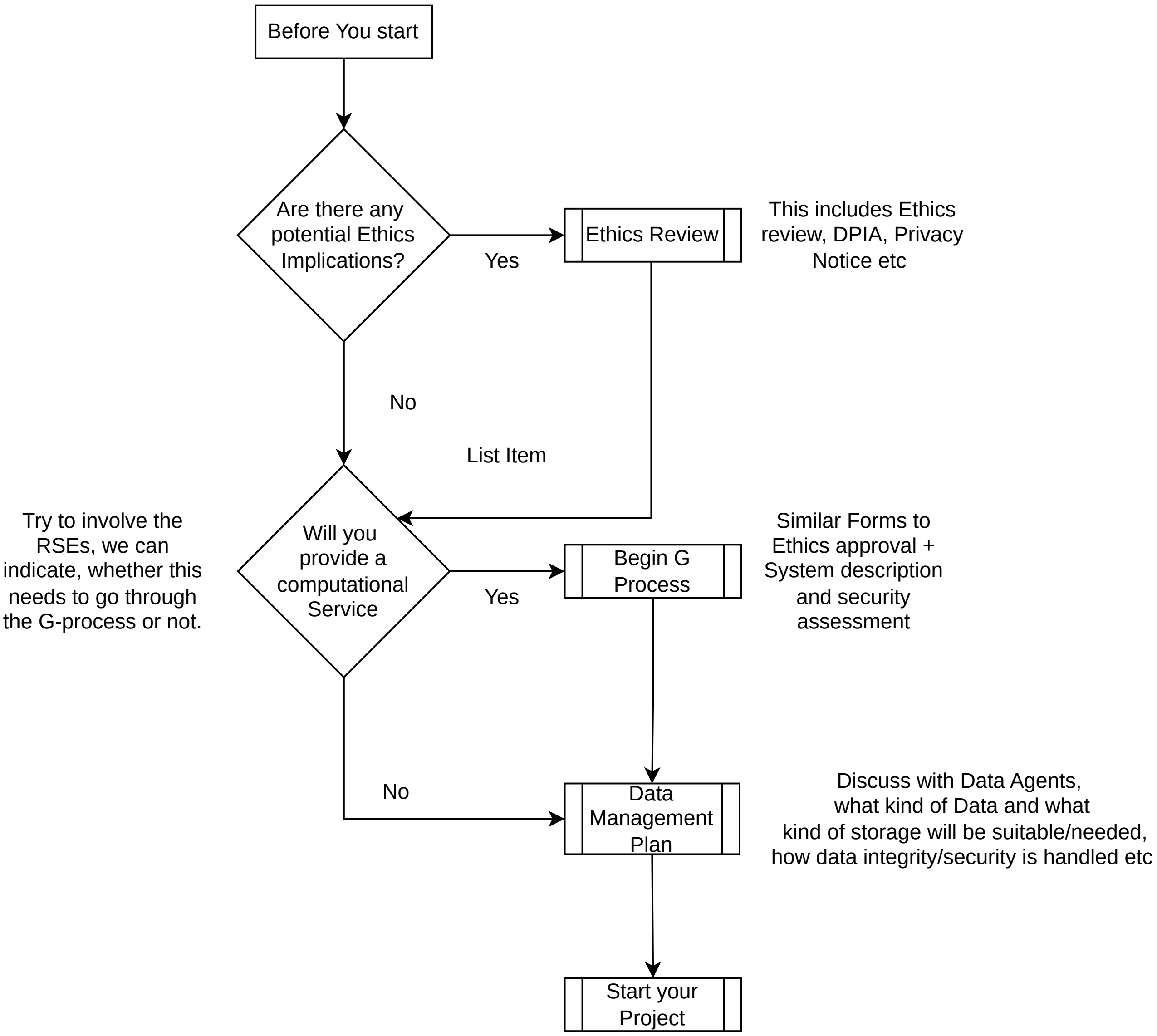
One of the first steps to make your work reproducible is to organize your projects well. Let's go over some of the basic things which people have found to work (and not to work).

Organizing projects

What to do when you start

Before you start

Here are some things that should be at least considered before starting any project at Aalto. It's important to not get scared by the amount of forms that you might encounter. There are people who are willing to help you like the Data Agents and Research Software Engineers

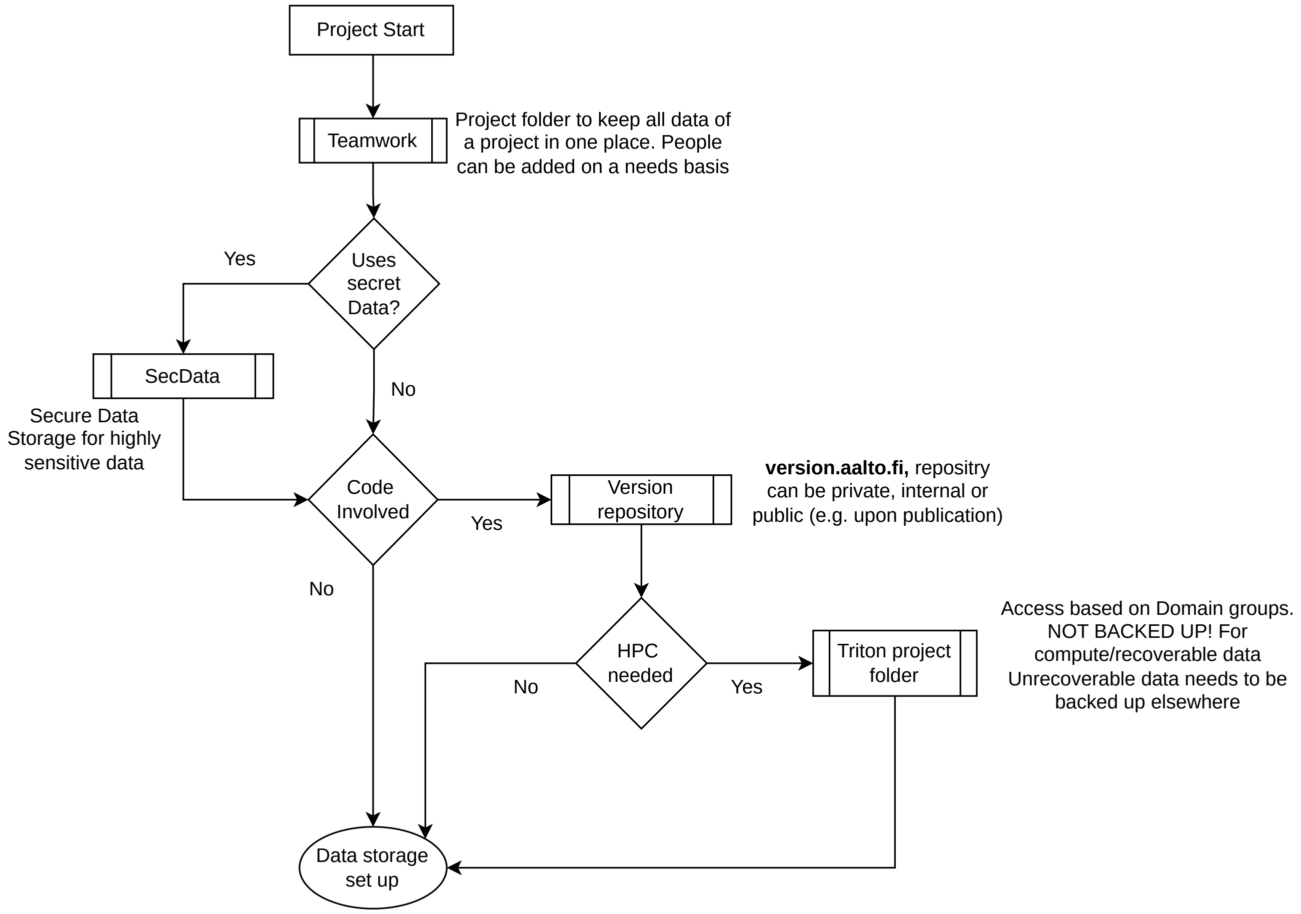

Before You start
Are there any potential Ethics Implications?
Yes
Ethics Review
This includes Ethics review, DPIA, Privacy Notice etc
No
List Item
Try to involve the RSEs, we can indicate, whether this needs to go through the G-process or not.
Will you provide a computational Service
Yes
Begin G Process
Similar Forms to Ethics approval + System description and security assessment
No
Data Management Plan
Discuss with Data Agents, what kind of Data and what kind of storage will be suitable/needed, how data integrity/security is handled etc
Start your Project

Relevant resources

- Research Ethics Committee
- G process for small scale development
- Data Management Plan

When you actually start

The following chart gives an overview of what data storage places you should set up / request when you start a new project, it can also help as a reference when setting up a Data management plan.


Project Start
Teamwork
Project folder to keep all data of a project in one place. People can be added on a needs basis
Uses secret Data?
Yes
SecData
Secure Data Storage for highly sensitive data
No
Code Involved
Yes
Version repository
version.aalto.fi, repositry can be private, internal or public (e.g. upon publication)
No
HPC needed
No
Yes
Triton project folder
Access based on Domain groups. NOT BACKED UP! For compute/recoverable data Unrecoverable data needs to be backed up elsewhere
Data storage set up

Relevant resources

- Data storage overview
- Teamwork Description
- Aalto Gitlab
- Triton Project Folder request
- SecData

It's important to note, that files in personal folders of students might be difficult to recover if the student leaves. Therefore always have project folders that are owned by the principal investigator and have your staff/students use these folders for storing project data or code.

Directory structure for projects

- Project files in a **single directory**, this is your teamwork folder.
- **Different projects** should have **separate directories**, different teamwork folders, different gitlab repositories
- Use **consistent and informative directory structure**
  - Avoid spaces in directory and file names – use `-`, `_` or CamelCase instead (nicer for computers to handle)
- If you need to separate public/private data in gitlab:
  - For secrets (e.g. API keys, access tokens or similar information):
    - use `.gitignore` to exclude the private information from being tracked
  - For code you don't want to have public yet:
    - create a separate public repository, which only contains the code that is publicly available.

        - Do NOT create a fork for this but a separate repository. Note, that this public repository is your “ground truth” for anyone else and should reflect the version in use for publications
- Add a **README file** to describe the project and instructions on reproducing the results
- If you want to use the **same code in multiple projects**, host it on GitHub (or similar) and clone it into each of your project directories

A project directory can look something like this:

```
project_name/
├── README.md             # overview of the project
├── data/                 # data files used in the project
│   ├── README.md         # describes where data came from
│   └── sub-directory/       # 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 (tracked with gitlab)
│   ├── LICENSE           # license for your code
│   ├── requirements.txt  # software requirements and dependencies
│   └── ...
└── doc/                  # documentation for your project
    ├── index.rst
    └── ...
```

Tracking source code, data, and results

- All code is version controlled and goes in the `src/` or `source/` 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 `processed_data/` and should be in `.gitignore`
- Consider using Git tags to mark specific versions of results (version submitted to a journal, dissertation version, poster version, etc.):

```
$ git tag -a thesis-submitted -m "this is the submitted version of my thesis"
```

Check the Git-intro lesson for a reminder.

Some tools and templates

- R devtools
- Python cookiecutter template

- Reproducible research template by the Turing Way

More tools and templates in Heidi Seibolds blog.

Resources on research compendia

- About research compendia at the Turing Way
- “Research compendia”: a set of good practices for reproducible data analysis in R, but much is transferable to other languages.
- rrtools: instructions, templates, and functions for writing a reproducible article or report with R.
- ...

Keypoints

- An organized project directory structure helps with reproducibility.
- Also think about version control for writing your academic manuscripts.

Recording computational steps

Objectives

- Know what is the minimal amount of information necessary to reproduce your results
- How can you make it easier for others (or your later self) to reproduce your work
- Understand why and when a workflow management tool can be useful

❓ Questions

- You have some steps that need to be run to do your work. How do you actually run them? Does it rely on your own memory and work, or is it reproducible? **How do you communicate the steps** for future you and others?
- How can we create a reproducible workflow?

Instructor note

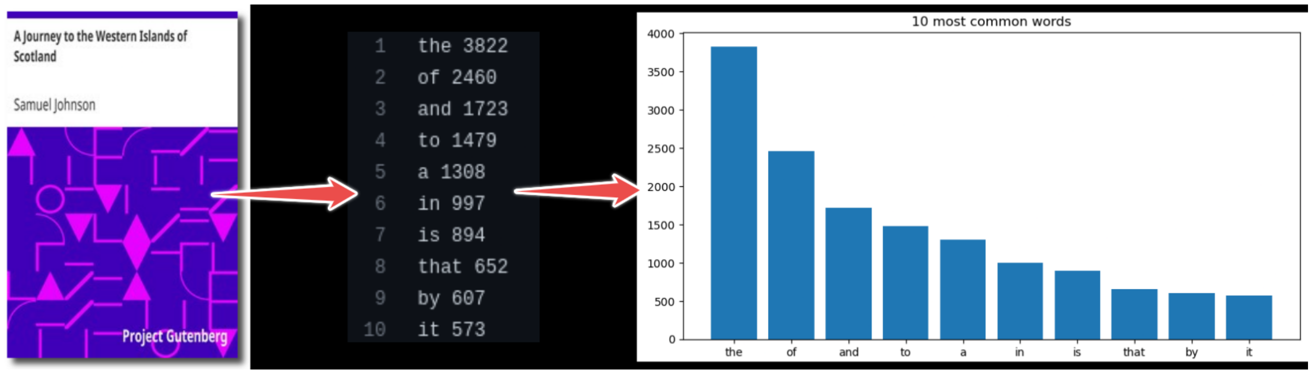
- 5 min teaching
- 5 min exercise/demo

Several steps from input data to result

The following material is partly derived from a HPC Carpentry lesson.

In this episode, we will use an example project which finds frequent words in books and plots the result from those statistics. In this example we wish to:

1. Analyze word frequencies using code/count.py for 4 books (they are all in the data directory).
2. Plot a histogram using plot/plot.py.



```
the 3822
of 2460
and 1723
to 1479
a 1308
in 997
is 894
that 652
by 607
it 573
```

Example (for one book only):

```
$ python code/count.py data/isles.txt > statistics/isles.data
$ python code/plot.py --data-file statistics/isles.data --plot-file plot/isles.png
```

Another way to analyze the data would be via a graphical user interface (GUI), where you can for example drag and drop files and click buttons to do the different processing steps.

We can also express the workflow for all books with a script. The repository includes such script called `run_all.sh`.

We can run it with:

```
$ bash run_all.sh
```

These approaches are fine, when only a few steps are needed, or only a few repetitons are required, but become infeasible when large amounts of data need to be processed. E.g. you don't want to do either approach with 500 different books. Clicking 500 times, or having 500 copies of lines with small modifications is bound to introduce typos or other errors. How could we deal with this?

- Loops with automated argument lists or other approaches to specify the inputs
- Workflow managers

The simpler way, to just get reproducible results is to have tools generate the inputs automatically e.g. using "one folder/file per input" approaches. This will lead to reproducible results, but requires that for every change everything has to be re-run. E.g. when adding another 10 datapoints, if your script just checks what is there, it will re-run the analysis for all 1000 other elements as well, and if you start adding more arguments to your script, you risk the forgetting elements or having typos again. The advantage of this approach, however, is that you can easily build a executable manuscript file from such a script (like jupyter notebooks, or matlab live code).

Workflow managers

Workflow managers in contrast create flows that, in general, keep track on what needs to be executed. E.g. snakemake will keep track of what has been processed and only re-run those parts of it's flow that need updates. If properly defined (e.g. source files being inputs of steps), it will re-run all analysis starting from a modified step, which makes results more dependable, since you can't forget to "run that one new pre-processing step" for some old input data. An example of how it can be used on triton (which is also generally applicable) can be found here

---

Additional Tools

- Make
- Nextflow
- Task
- Common Workflow Language
- Many specialized frameworks exist.
- Book on building reproducible analytical pipelines with R
- {targets} R package - make-like pipeline tool for R

❗ Keypoints

- Computational steps can be recorded in many ways.
- Workflow tools can help if there are many steps to be executed and/or many datasets to be processed.

Recording dependencies

❗ Objectives

- Understand what dependency management tools can be useful for
- Discuss environment/requirements files in the context of reusability and reproducibility

❓ Questions

- How can we communicate different versions of software dependencies?

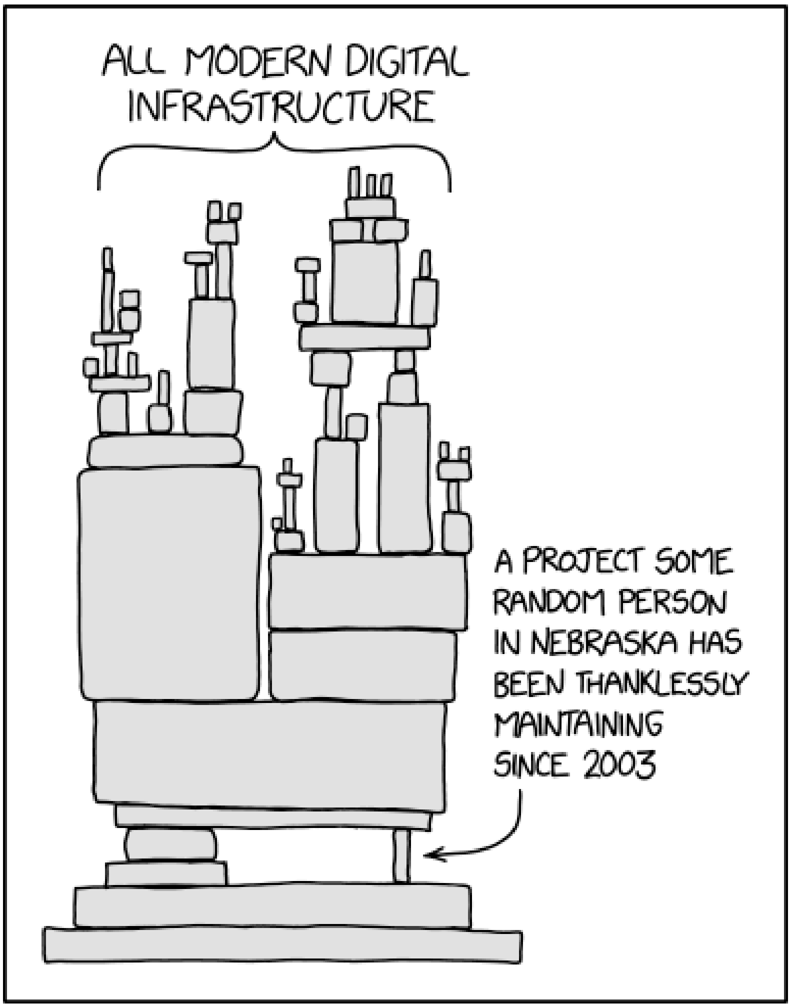
Instructor note

- 10 min teaching

- 10 min discussion

Our codes often depend on other codes that in turn depend on other codes …

- **Reproducibility**: We can version-control our code with Git but how should we version-control dependencies? How can we capture and communicate dependencies?
- **Dependency hell**: Different codes on the same environment can have conflicting dependencies.


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

From xkcd - dependency. Another image that might be familiar to some of you working with Python can be found on xkcd - superfund.

Dependency and environment management

Tools like **Conda, Anaconda, pip, virtualenv, Pipenv, pyenv, Poetry, renv** and files to record dependencies like **requirements.txt** and **environment.yml** try to solve the following problems:

- **Defining a specific set of dependencies**
- **Installing those dependencies** mostly automatically
- **Recording the versions** for all dependencies
- **Isolate environments**
  - On your computer for projects, so they can use different software

    - Isolate environments on computers with many users (and allow self-installations)
- Using **different package versions** per project (also, e.g., Python/R versions)
- Provide tools and services to **share packages**

Isolated environments are also useful because they help you make sure that you know your dependencies!

If things go wrong, you can delete and re-create - much better than debugging. The more often you re-create your environment, the more reproducible it is.

Examples of recorded dependencies

✍ Dependencies-1: Time-capsule of dependencies

Situation: 5 researchers (A, B, C, D, E) wrote code that depends on a couple of libraries. They uploaded their projects to GitHub. We now travel 3 years into the future and find their GitHub repositories from the respective publications. We would like to try to re-run their code before adapting it. Which of the following do you think you will get to work?

Conda Python virtualenv R

A: You find a couple of library imports across the code but that’s it.

B: The README file lists which libraries were used.

C: You find a `environment.yml` file with:

```
name: student-project
channels:
  - conda-forge
dependencies:
  - scipy
  - numpy
  - sympy
  - click
  - python
  - pytorch
  - pip
  - pip:
    - git+https://github.com/someuser/someproject.git@master
    - git+https://github.com/anotheruser/anotherproject.git@master
```

D: You find a `environment.yml` file with:

```
name: student-project
channels:
  - conda-forge
dependencies:
  - scipy=1.3.1
  - numpy=1.16.4
  - sympy=1.4
  - click=7.0
  - python=3.8
  - pytorch=1.10
  - pip
  - pip:
    - git+https://github.com/someuser/someproject.git@d7b2c7e
    - git+https://github.com/anotheruser/anotherproject.git@sometag
```

E: You find a `environment.yml` file with:

```
name: student-project
channels:
  - conda-forge
dependencies:
  - scipy=1.3.1
  - numpy=1.16.4
  - sympy=1.4
  - click=7.0
  - python=3.8
  - pytorch=1.10
  - someproject=1.2.3
  - anotherproject=2.3.4
```

✔ Solution

A: It will be tedious to collect the dependencies one by one. And after the tedious process you will still not know which versions they have used.

B: If there is no standard file to look for and look at, it might become very difficult to create the software environment required to run the software. At least we know the list of libraries, but we don't know the versions.

C: Having a standard file listing dependencies is definitely better than nothing. However, if the versions are not specified, you or someone else might run into problems with dependencies, deprecated features, changes in package APIs, etc.

D and **E**: In both of these cases exact versions of all dependencies are specified and one can recreate the software environment required for the project. One problem with the dependencies that come from GitHub is that they might have disappeared (what if their authors deleted these repositories?).

E is slightly preferable because version numbers are easier to understand than Git commit hashes or Git tags, but is most often out of scope for research projects, as it requires a significant overhead to submit these versions to the respective repositories

The necessary information for reproducibility of computational results

When it comes to computational results there are several factors that can come into play

1. Software
2. Hardware

Hardware

If possible (e.g. the code was run on a local machine), provide the make/model of the CPU and GPU used for the computations. unfortunately those can have an effect on the actual results, and it allows others to estimate run times on their hardware.

Software

For all of the following assume that we also want version numbers.

- Operating System (e.g. Windows 10, Windows 11, Ubuntu Linux 22.04, Red Hat Linux 9.3, Mac OS X 15...)
- Compilers/Interpreters (e.g. gcc, python, matlab etc)
- Libraries (what we talked about above).

If you use conda/mamba, there are some very useful functions that can extract what is currently in your environment.

1. `conda env export` This command will export the precise list of conda controlled packages in your environment down to the build version. For reproducibility I always recommend to provide this file, since it is the only version that shows someone precisely what you used. However, this export is often OS specific, if third-language libraries are used because the builds need to be created per operating system.
2. `conda env export --no-builds` This command lists the environment stripping the build version and should (often) be installable on any operating system However some dependencies are operating system dependent and thus even this version might not be installable on a differen OS.
3. `conda env export --from-history` This will list everything that you actively installed into the environment, but not the dependencies of what you wanted. This often leads to a minimal environment file, which is most likely installable anywhere. However, this does NOT list versions, as it simply repeats what you had in your environment file and any addiitonal `conda install xyz` or `pip install xyz` commands you ran.

In conclusion, We recommend to provide:

1. The environment file generated from `conda env export`.
2. The environment file generated from `conda env export --from-history` modified such, that the versions (not builds) listed in the full export are added to the file.

The former contains all information available about what you were running, the latter is a simple way to reproduce your code with the minimal amount of information required to obtain all relevant dependencies.

❗ Keypoints

- Recording dependencies with versions can make it easier for the next person to execute your code.
- There are many tools to record dependencies and separate environments.

Recording environments

❗ Objectives

- Understand what containers are and what they are useful for
- Discuss container definitions files in the context of reusability and reproducibility

Instructor note

- 5 min teaching/discussion
- 5 min demo

What is a container?

Imagine if you didn’t have to install things yourself, but instead you could get a computer with the exact software for a task pre-installed. Containers effectively do that, with various advantages and disadvantages. They are **like an entire operating system with software installed, all in one file**.

From reddit.

From definition files to container images to containers

- Containers can be built to bundle *all the necessary ingredients* (data, code, environment, operating system).
- A container image is like a piece of paper with all the operating system on it. When you run it, a transparent sheet is placed on top to form a container. The container runs and writes only on that transparent sheet (and what other mounts have been layered on top). When you are done, the transparent sheet is thrown away. This can be repeated as often as you want, and base is always the same.
- Definition files (e.g., Dockerfile or Singularity definition file) are text files that contain a series of instructions to build container images.

You may have use for containers in different ways

- **Installing a certain software is tricky**, or not supported for your operating system? - Check if an image is available and run the software from a container instead!
- You want to make sure your colleagues are using the **same environment** for running your code? - Provide them an image of your container!
    - If this does not work, because they are using a different architecture than you do? - Provide a definition file for them to **build the image suitable for their computers**. This does not create the exact environment you have, but in most cases a similar enough

one.

Popular container implementations

- Docker
- Singularity (popular on high-performance computing systems)
- Apptainer (popular on high-performance computing systems, fork of Singularity)
- podman

They are to some extent interoperable:

- podman is very close to Docker
- Docker images can be converted to Singularity/Apptainer images
- Singularity Python can convert Dockerfiles to Singularity definition files

Pros and cons of containers

Containers are popular for a reason - they solve a number of important problems:

- Allow for seamlessly **moving workflows across different platforms**.
- Can solve the **“works on my machine”** situation.
- For software with many dependencies, in turn with its own dependencies, containers offer possibly the only way to preserve the computational experiment for **future reproducibility**.
- A mechanism to “send the computer to the data” when the **dataset is too large** to transfer.
- **Installing software into a file** instead of into your computer (removing a file is often easier than uninstalling software if you suddenly regret an installation).

However, containers may also have some drawbacks:

- Can be used to hide away software installation problems and thereby **discourage good software development practices**.
- Instead of “works on my machine” problem: **“works only in this container”** problem?
- They can be **difficult to modify**.
- Container **images can become large**.

Danger

Use only **official and trusted images**! Not all images can be trusted! There have been examples of contaminated images, so investigate before using images blindly. Apply the same caution as when installing software packages from untrusted package repositories.

✍ (optional) Containers-3: Explore two really useful Docker images

You can try the below if you have Docker installed. If you have Singularity/Apptainer and not Docker, the goal of the exercise can be to run the Docker containers through Singularity/Apptainer.

1. Run a specific version of *Rstudio*:

   ```
   $ docker run --rm -p 8787:8787 -e PASSWORD=yourpasswordhere rocker/rstudio
   ```

   Then open your browser to http://localhost:8787 with login rstudio and password "yourpasswordhere" used in the previous command.

   If you want to try an older version you can check the tags at https://hub.docker.com/r/rocker/rstudio/tags and run for example:

   ```
   $ docker run --rm -p 8787:8787 -e PASSWORD=yourpasswordhere rocker/rstudio:3.3
   ```

2. Run a specific version of *Anaconda3* from https://hub.docker.com/r/continuumio/anaconda3:

   ```
   $ docker run -i -t continuumio/anaconda3 /bin/bash
   ```

Resources for further learning

- Carpentries incubator lesson on Docker
- Carpentries incubator lesson on Singularity/Apptainer

Keypoints

- Containers can be helpful if complex setups are needed to run a specific software.
- They can also be helpful for prototyping without "messing up" your own computing environment, or for running software that requires a different operating system than your own.

Where to go from here

Objectives

- Understand when tools discussed in this episode can be useful

Instructor note

- 10 min discussion

This episode presents a lot of different tools and opportunities for your research software project. However, you will not always need all of them. As with so many things, it again depends on your project.

Workflow tools will maybe make sense in the future

- In many cases, it is probably not needed
- You will want to consider workflow tools:
  - When processing many files with many steps
  - Steps or files may change
  - Your main script, connecting your steps, gets very long
  - You are still collecting your input data
  - …

Containers seem amazing, but do I have use for them?

- Maybe not yet, but knowing that you can …

  - Run Linux tools on your Windows computer
  - Run different versions of same software on your computer
  - Follow the "easy installation instructions" for an operating system that is not your own
  - Get a fully configured environment instead of only installing a tool
  - Share your setup and configurations with others

  … can be very beneficial :)

Important for every project

- A clear directory/file structure for your project.
- Record your workflow and write it down in a script file.
- Create a dependency list and keep it updated, optimally in an environment file.
- At least consider the possibility that someone, maybe you, may want to reproduce your work:
  - Can you do something (small) to make it easier?
  - If you have ideas, but no time: add an issue to your repository; maybe someone else wants to help.

Further reading

- The Turing Way handbook to reproducible, ethical and collaborative data science

- Reproducible research policies and software/data management in scientific computing journals: a survey, discussion, and perspectives
- Applying the FAIR Principles to computational workflows
- Recommendations for the packaging and containerizing of bioinformatics software (most of the recommendations and concepts should be transferable to other fields)
- …

➡ See also

Do you want to practice your reproducibility skills and get inspired by working with other people's code/data? Join a ReproHack event!

❗ Keypoints

- Not everything in this lesson might be useful right now, but it is good to know that these things exist if you ever get in a situation that would require such solutions.
- Caring about reproducibility makes work easier for the next person working on the project - and that might be you in a few years!

List of exercises

Full list

This is a list of all exercises and solutions in this lesson, mainly as a reference for helpers and instructors. This list is automatically generated from all of the other pages in the lesson. Any single teaching event will probably cover only a subset of these, depending on their interests.

Instructor guide

This sections is about the more extensive CR workshop, but is useful.

Detailed day schedule

Some example schedules for this lesson:

2024 edition plan (times in EET, Helsinki time), **no exercises**, just demos:

- 09:50 - 10:00 Soft start and icebreaker question
  - Page: collaborative notes document
  - Give more space to the icebreaker and see what people are writing and talk about our own experiences
- 10:00 - 10:03 Collab document intro
- 10:03 - 10:05 Learning outcomes: https://coderefinery.github.io/reproducible-research/
- 10:05 - 10:10 Overview of CR and how it all fits together
  - Page: https://coderefinery.github.io/reproducible-research/intro

    - Learning outcomes from index
- 10:10 - 10:20 Reproducible research, Motivation
    - Exercise in notes doc with the discussions in bottom of motivation page
    - Page: https://coderefinery.github.io/reproducible-research/motivation/
- 10:20 - 10:30 Organizing your projects
    - Copy the discussion on the notes and if we have time we can highlight some answers
    - Page: https://coderefinery.github.io/reproducible-research/organizing-projects/
- 10:30 - 10:35 ask in collab document and discuss
    - https://coderefinery.github.io/reproducible-research/organizing-projects/#discussion-on-reproducibility
        - Are you using version control for academic papers?
            - ...
            - ...
        - How do you handle collaborative issues e.g. conflicting changes?
            - ...
            - ...
- 10:35 - 10:55 Recording computational steps
    - Page: https://coderefinery.github.io/reproducible-research/workflow-management/
- 10:55 - 11:05 Real break
- 11:05 - 11:25 Recording dependencies
    - https://coderefinery.github.io/reproducible-research/dependencies/#exercises
        - ask first one in collab doc and discuss on stream
        - show difference between created env from env file vs exported env file on stream
- 11:25 - 11:30 ask in collaborative document
    - Are you using any dependency and/or environment management tool in your work?
        - No: o
            - why not?
                - ..
                - ..
        - Yes: o
            - which?
                - ..
                - ..
    - Have you heard about or been in contact with containers (docker, singularity, podman) in your work? How did you come across them?
        - No: o
        - Yes:
            - ..
            - ..
            - ..
- 11:30 - 11:50 Recording environments
    - The first contact with containers is often: Take this and run this command and then when you need to share/build.
    - Discuss setup issues, permissions if docker wants root, bandwidth, etc
    - Pros and cons of containers

    - Demo of two pre-made containers e.g. expand the R studio optional exercise?
- 11:50 - 12.00 Wrapup
    - where to go from here: idea would be to give it more practical focus: what to do with these tools? Project level reproducibility. Time-scales of what changes (short time changes of code, long time years changes of OS-s, libraries).
    - Bring your code session advertisement
    - Material + recording available
- 12:00 - long break starts

This is the planned schedule for the workshop in September 2023 (2 hours and 5 minutes including 10 min break) ; note that for this workshop, sharing code and data was moved to social coding lesson:

- 08:50 - 09:00 Soft start and icebreaker question
- 09:00 - 09:10 Overview of CR and how it all fits together
- 09:10 - 09:20 Reproducible research, Motivation
- 09:20 - 09:27 Organizing your projects
- 09:27 - 09:35 Recording computational steps - discussion
- 09:35 - 10:00 Snakemake exercise (25 min)
- 10:00 - 10:10 Break
- 10:10 - 10:15 Summary of workflows and the exercise
- 10:15 - 10:30 Recording dependencies
- 10:30 - 10:40 Recording environments
- 10:40 - 11:00 Container-1 exercise (20 min)
- 11:00 - 11.05 Wrapup

This was the schedule at workshop in March 2023 (2 hours and 15 minutes including 2x 10 min break):

- 08:50 - 09:00 Soft start and icebreaker question
- 09:00 - 09:10 Interview with an invited guest
- 09:10 - 09:20 Motivation
- 09:20 - 09:30 Organizing your projects
- 09:30 - 10:00 Recording dependencies
    - discussion (5 min)
    - exercise (20 min)
    - discussion (5 min)
- 10:00 - 10:10 Break
- 10:10 - 10:40 Recording computational steps
    - discussion (5 min)
    - exercise (20 min)
    - discussion (5 min)
- 10:40 - 10:50 Recording environments
    - an exercise exists but is typically not done as part of a standard workshop
- 10:50 - 11:05 Sharing code and data

    - demo (15 min)
- 11:05 - 11:15 Break

Why we teach this lesson

Reproducibility in research is something that publishers, funding agencies, universities, research leaders and the general public worries about and much is being written about it. It is also something that researchers care deeply about - this lesson is typically one of the most popular lessons in the pre-workshop survey.

Even though most PhD students, postdocs and researchers (i.e. typical workshop participants) know about the importance of reproducibility in research, they often lack both a general overview of what different aspects there are to reproducibility, and the knowledge of specific tools that can be used for improving reproducibility.

Many participants may not adhere to good practices when organizing their projects, and the "Organizing your projects" episode is meant to encourage participants to structure their projects better. This may be obvious to some participants but it doesn't harm to preach to the choir.

Even though many participants know that code can have many dependencies (e.g. they may have experienced difficulties in getting other people's code to run), they often don't know or use good practices when it comes to recording dependencies. Most participants also don't use isolated environments for different projects and don't know why that can be important. The episode "Recording dependencies" tries to convey the importance of recording dependencies accurately for your projects, and shows how tools like conda can be used both as a package and software environment manager.

Many participants have heard about containers and find them interesting, but lack an understanding of how they work or how they can be used. The episode "Recording environments" introduces the concept of containers, and the optional episode "Creating and sharing a container image" goes into details.

Many participants use complicated series of computational steps in their research without realizing that this work falls into the category of "scientific workflows", and that there actually exist tools that help make such workflows reproducible. The episode "Recording computational steps" introduces the concept of scientific workflows, discusses various ways of managing workflows with varying degrees of reproducibility, and shows how tools like Snakemake can be used to both simplify workflows and make them more reproducible.

How to teach this lesson

How to start

Everyone knows that scientific results need to be reproducible, but not everyone is using appropriate tools to ensure this. Here we're going to get to know tools which help with preserving the provenance of data and reproducibility on different levels, ranging from workflow automation to software environment (containers).

Focus on concepts, and when to use which tool

Try to explain better what the different tools are useful for, but don't go into details. In this lesson we are not trying to gain expertise in the various tools and master the details but rather we want to give an overview and show that many tools exist and try to give participant the right feel for which set of tools to approach for which type of problem.

Typical pitfalls

Indentation in Snakefiles

- the body of a rule and the body of an input keyword need to be indented, but the number of spaces doesn't matter This works:

```
rule all:
    input:
        expand('statistics/{book}.data', book=DATA),
        expand('plot/{book}.png', book=DATA)
```

but this doesn't work:

```
rule all:
    input:
    expand('statistics/{book}.data', book=DATA),
    expand('plot/{book}.png', book=DATA)
```

nor this:

```
rule all:
input:
    expand('statistics/{book}.data', book=DATA),
    expand('plot/{book}.png', book=DATA)
```

Field reports

2022 September

We used the strategy "absolutely minimal introductions, most time for exercise". Overall, it was probably the right thing to do since there is so little time and so much to cover.

There wasn’t enough time for the conda exercise (we could give only 7 minutes), but also I wonder how engaging it is. We should look at how to optimize the start of that episode.

The Snakemake episode went reasonably well. Our goal was 5 minutes intro, long exercise, 5 minutes outro. The intro was actually a bit longer, and there was the comment that we didn’t really explain what Snakemake was before it started (though we tried). The start of this episode should get particular focus in the future, since this is the main exercise of the day.