ETH zürich

IT Services



Reproducible Scientific Computing and Data Analysis

Laura Marqués, Henry Lütcke
Scientific IT Services, ETH Zurich

October 28, 2026

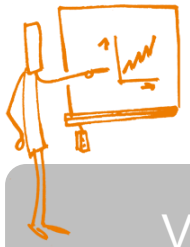
Slides & Materials: https://siscourses.ethz.ch/reproducible_computing/

CC BY NC SA

Overview of today's workshop



Setting the Scene



Version Control

git



Managing Dependencies & Computing Environments

CONDA



Virtualizing Computing Environments

docker



Literate Programming and Interactive Notebooks

quarto

jupyter



Reproducible Computing Platforms

binder renku

Image credit: Open Science Training Handbook





Setting the Scene

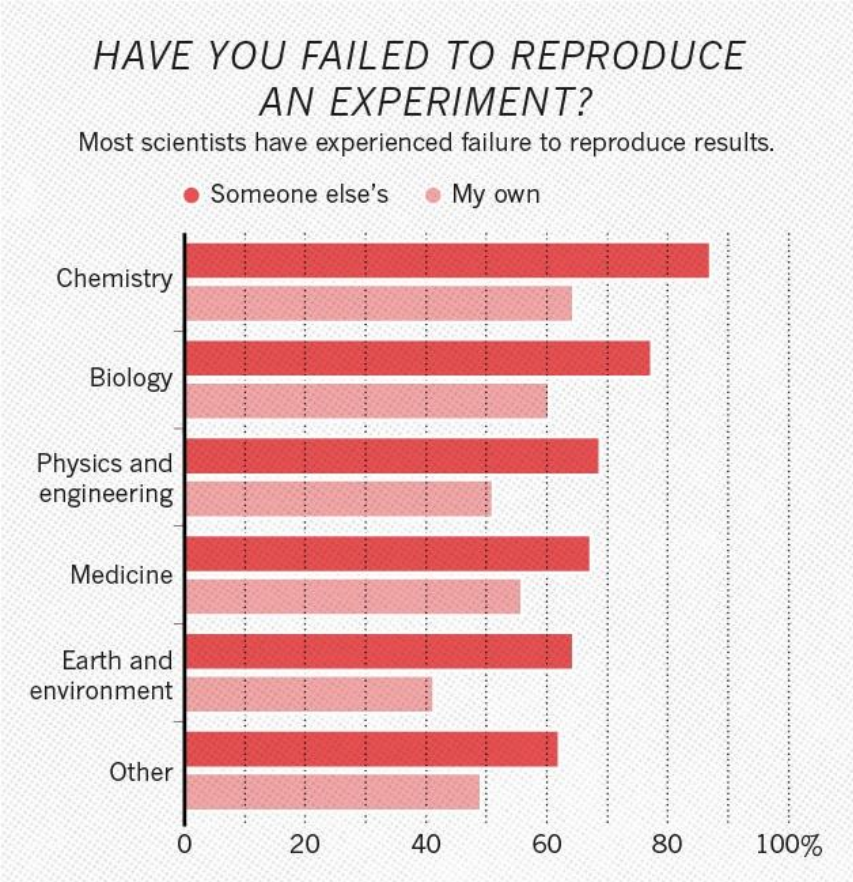


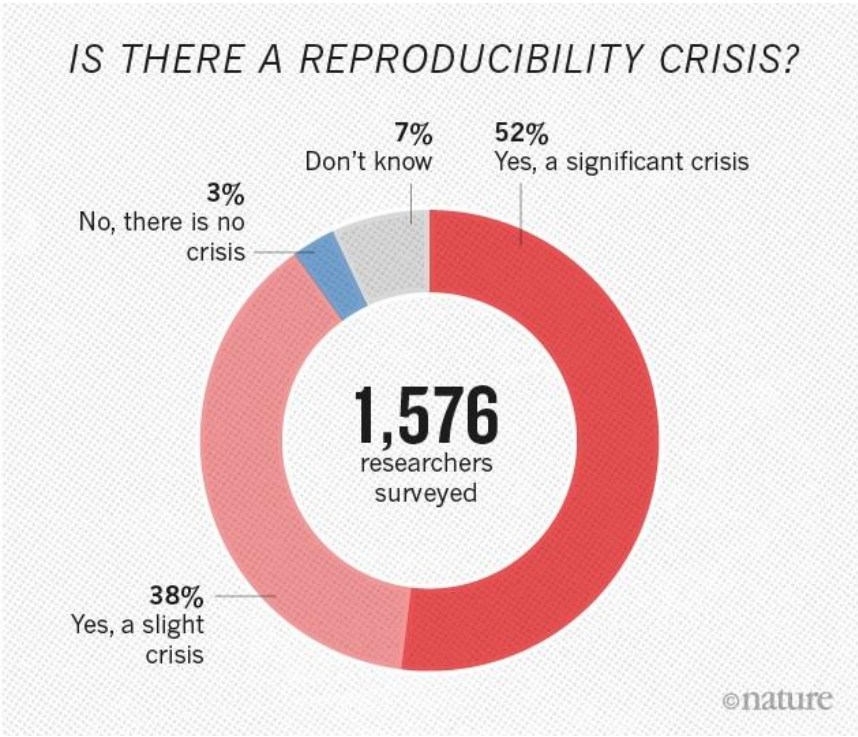




Reproducibility & Replicability in Science

Nature survey on reproducibility across all scientific domains





[Nature 533, 452–454 (26 May 2016) doi:10.1038/533452a]

Reproducibility & Replicability in Science

RESEARCH ARTICLE

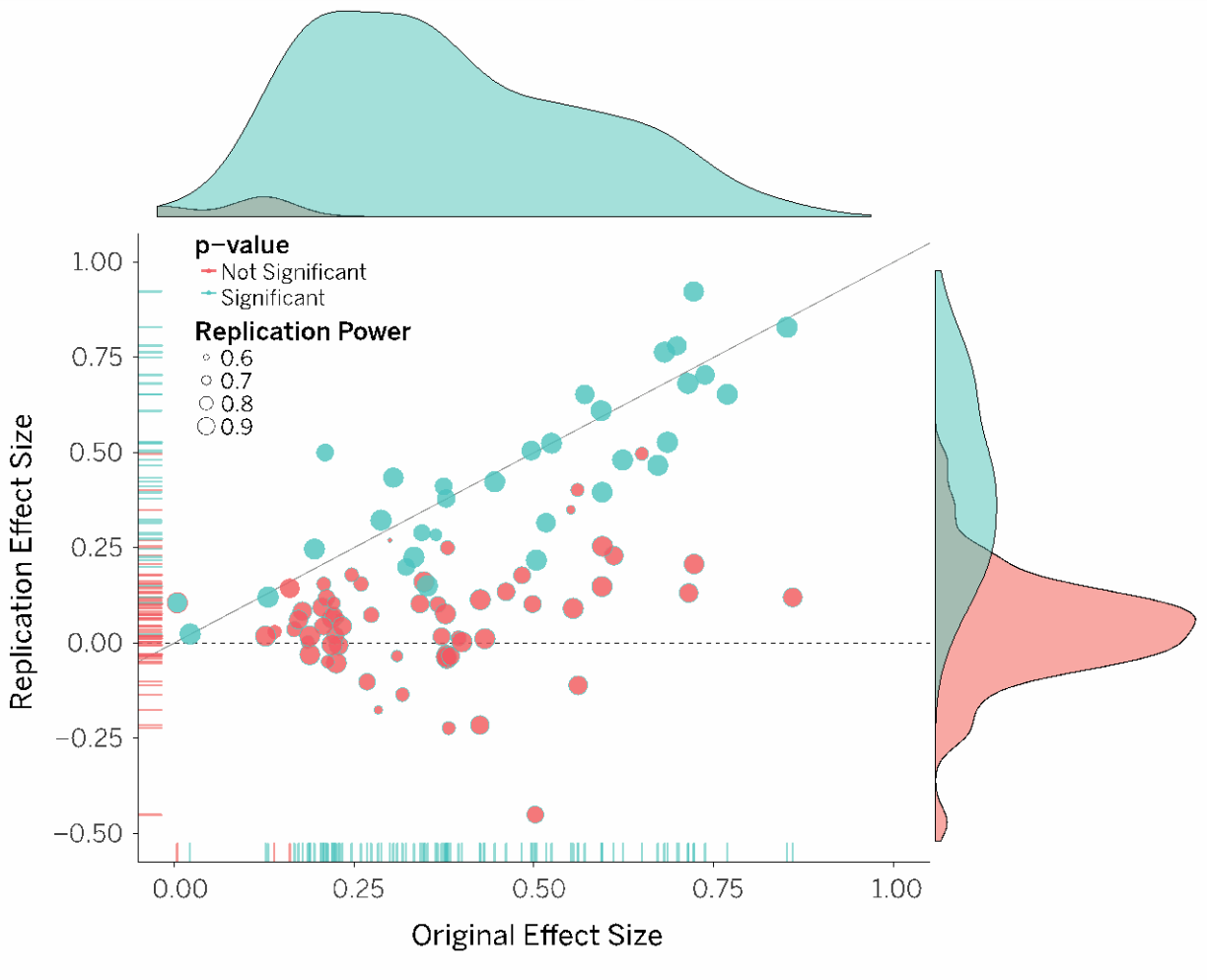
Estimating the reproducibility of psychological science

Open Science Collaboration*,†
+ See all authors and affiliations

Science 28 Aug 2015:
Vol. 349, Issue 6251, aac4716
DOI: 10.1126/science.aac4716

The *Reproducibility project*

- Replicate 100 experiments published in top psychology journals
- One-half to two-thirds of original findings could not be observed in the replication study



Reproducibility & Replicability in Science

RESEARCH ARTICLE

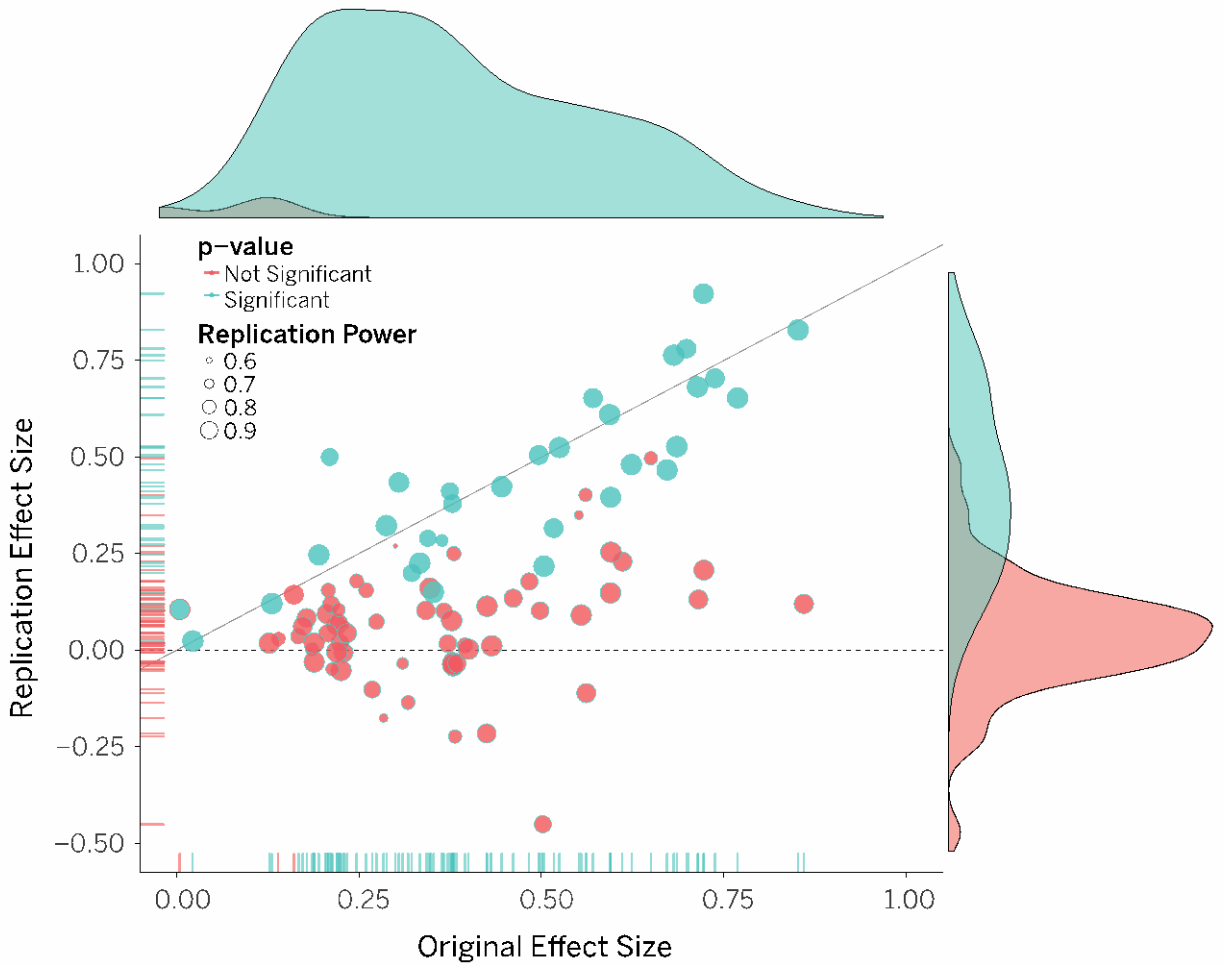
Estimating the reproducibility of psychological science

Open Science Collaboration*,†
+ See all authors and affiliations

Science 28 Aug 2015:
Vol. 349, Issue 6251, aac4716
DOI: 10.1126/science.aac4716

The *Reproducibility project*

- Replicate 100 experiments published in top psychology journals
- One-half to two-thirds of original findings could not be observed in the replication study

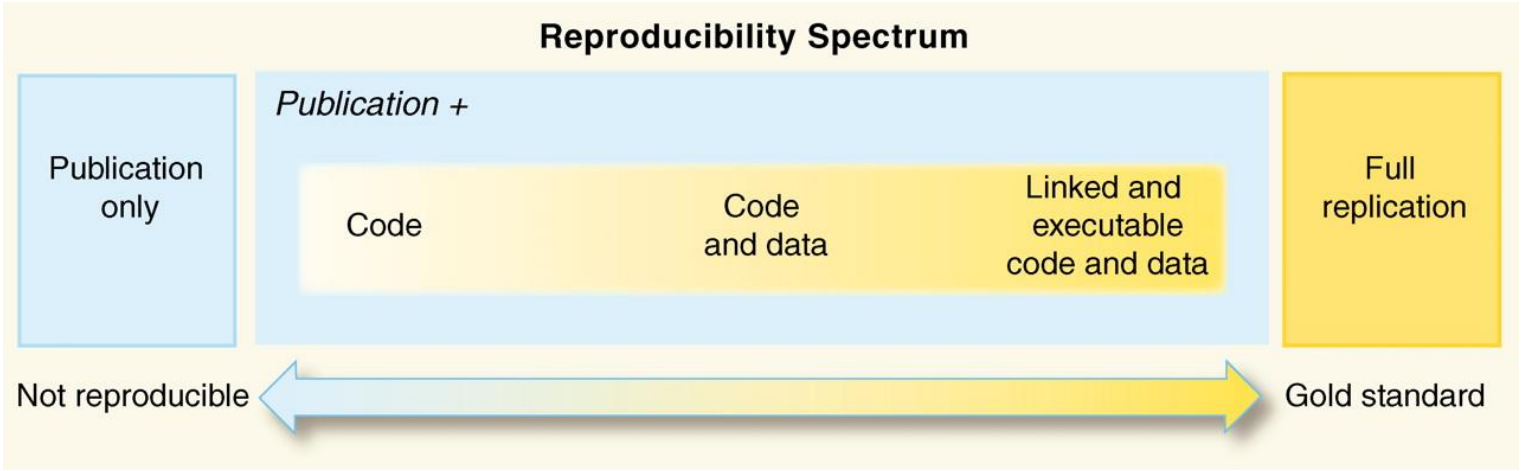


Reproducibility & Replicability in Science



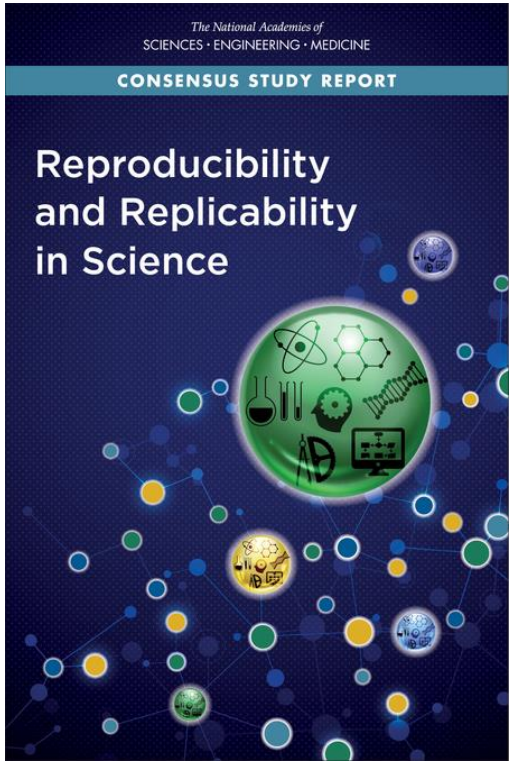
Replication:
new data and / or new method in independent study = same finding

Reproducible research:
same data + same method = same results



Peng (2011). doi:10.1126/science.1213847

Defining the Scope: Computational Reproducibility



«**Reproducibility** is obtaining consistent results using the same input data, computational steps, methods, and code and conditions of analysis. The term is synonymous with *"computational reproducibility"*... »

«To help ensure the reproducibility of computational results, researchers should **convey clear, specific, and complete information about any computational methods and data products that support their published results in order to enable other researchers to repeat the analysis**, unless such information is restricted by non-public data policies. That information should include the data, study methods, and **computational environment**. »

National Academies of Sciences, Engineering, and Medicine (2019). https://doi.org/10.17226/25303



Reproducibility ≠ Correctness

Computational Reproducibility: What can go wrong?

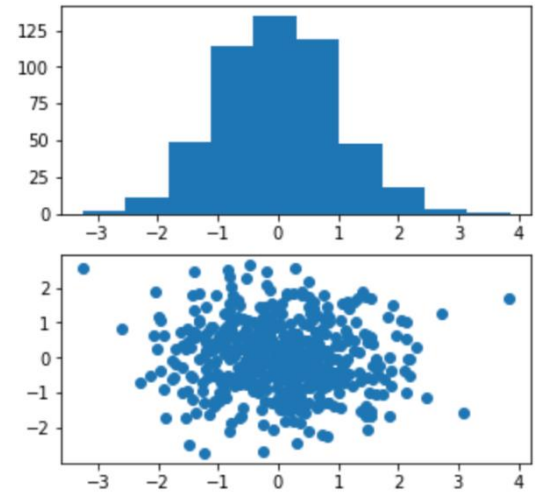
- Code only runs on specific **operating system**
  - Examples: Windows / Linux scripts, special programs (e.g. *SigmaPlot*)
- Code has specific **external dependencies**
  - Example: wget https://zenodo.org/record/1234567/files/dataset.zip
- Code has specific **internal dependencies** (libraries, modules etc.)

```python
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(42)
data = np.random.randn(2, 500)

fig, axs = plt.subplots(2, 1, figsize=(5, 5))
axs[0].hist(data[0])
axs[1].scatter(data[0], data[1])

plt.show()
```



Computational Reproducibility: What can go wrong?

- Code only runs on specific **operating system**
  - Examples: Windows / Linux scripts, special programs (e.g. *SigmaPlot*)
- Code has specific **external dependencies**
  - Example: wget https://zenodo.org/record/1234567/files/dataset.zip
- Code has specific **internal dependencies** (libraries, modules etc.)
- Code has specific **version dependencies**
- Code may rely on availability of specific **software licenses**
  - Example: fastaread function in the MATLAB Bioinformatics Toolbox

```
import numpy as np

print("Using Numpy %s" % np.__version__)

rng = np.random.default_rng(42)
rng.dirichlet((0.04, 0.03), 2)
```

```
Using Numpy 1.18.1
array([[2.10122596e-01, 7.89877404e-01],
       [1.99456813e-22, 1.00000000e+00]])
```

```
import numpy as np

print("Using Numpy %s" % np.__version__)

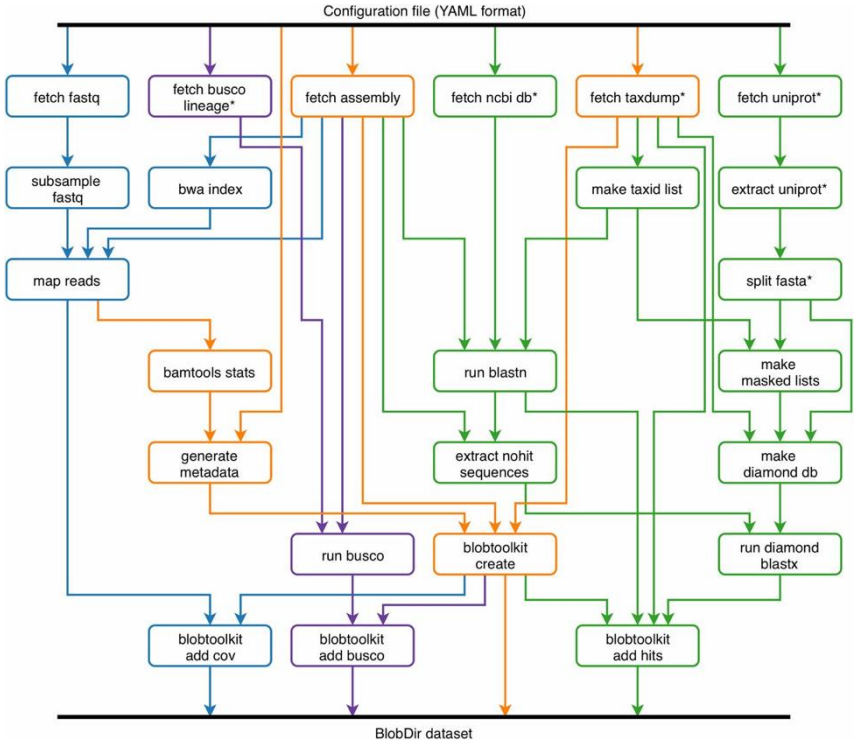
rng = np.random.default_rng(42)
rng.dirichlet((0.04, 0.03), 2)
```

```
Using Numpy 1.20.2
array([[9.99999999e-01, 7.24826532e-10],
       [9.99726345e-01, 2.73654825e-04]])
```

See *https://numpy.org/doc/stable/release/1.19.0-notes.html#changed-random-variate-stream-from-numpy-random-generator-dirichlet*

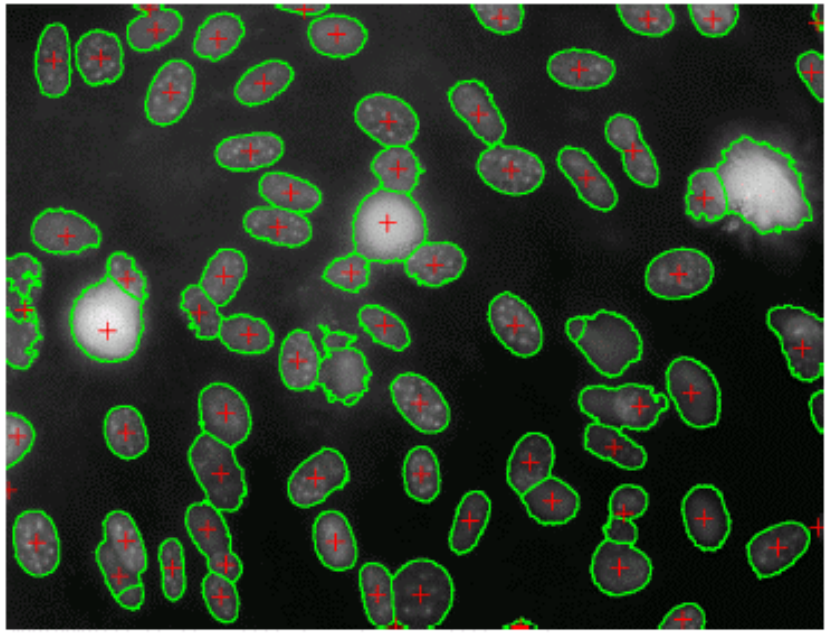
Computational Reproducibility: What can go wrong?

- Code only runs on specific **operating system**
  - Examples: Windows / Linux scripts, special programs (e.g. *SigmaPlot*)
- Code has specific **external dependencies**
  - Example: wget https://zenodo.org/record/1234567/files/dataset.zip
- Code has specific **internal dependencies** (libraries, modules etc.)
- Code has specific **version dependencies**
- Code may rely on availability of specific **software licenses**
  - Example: fastaread function in the MATLAB Bioinformatics Toolbox
- Code may be **incomprehensible** (complex, undocumented workflows)



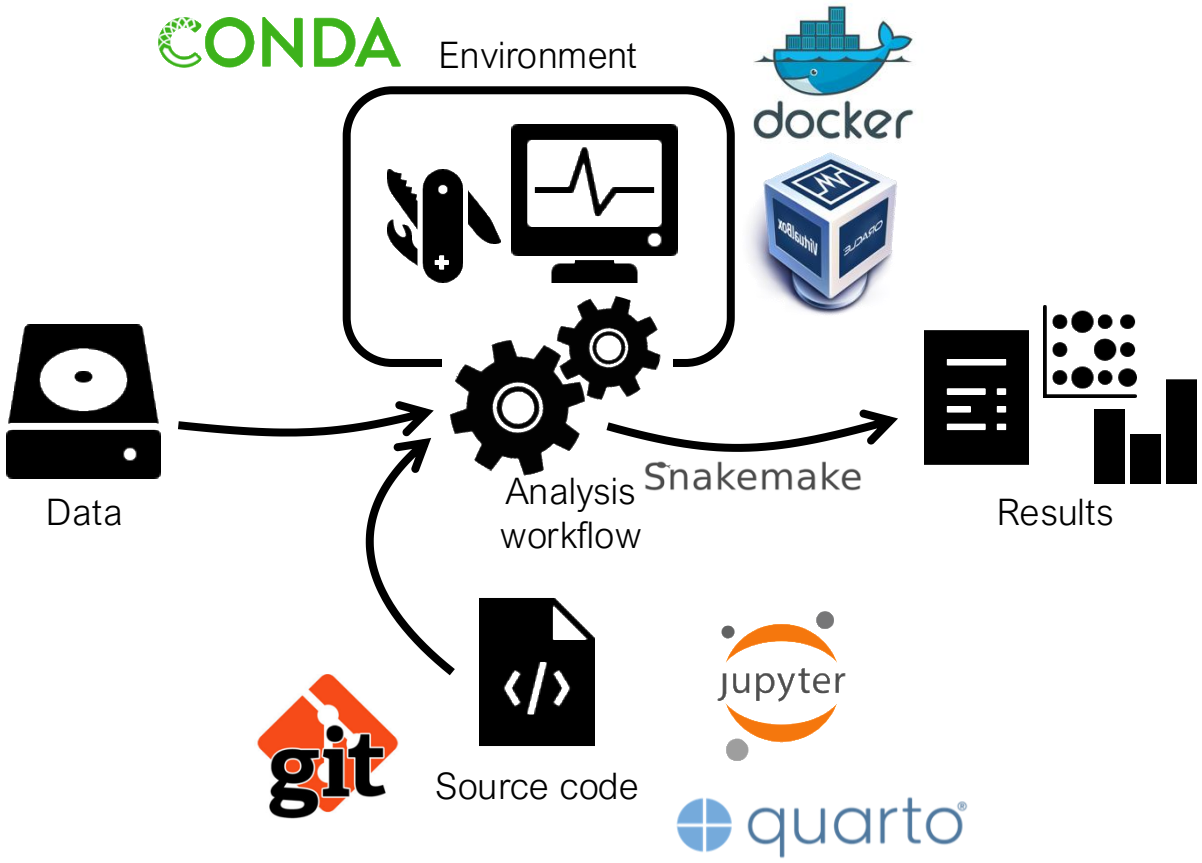
Computational Reproducibility: What can go wrong?

- Code only runs on specific **operating system**
  - Examples: Windows / Linux scripts, special programs (e.g. *SigmaPlot*)
- Code has specific **external dependencies**
  - Example: `wget` https://zenodo.org/record/1234567/files/dataset.zip
- Code has specific **internal dependencies** (libraries, modules etc.)
- Code has specific **version dependencies**
- Code may rely on availability of specific **software licenses**
  - Example: `fastaread` function in the MATLAB Bioinformatics Toolbox
- Code may be **incomprehensible** (complex, undocumented workflows)
- Analysis workflow may rely on **manual steps**



Computational Reproducibility: Pieces of the Puzzle

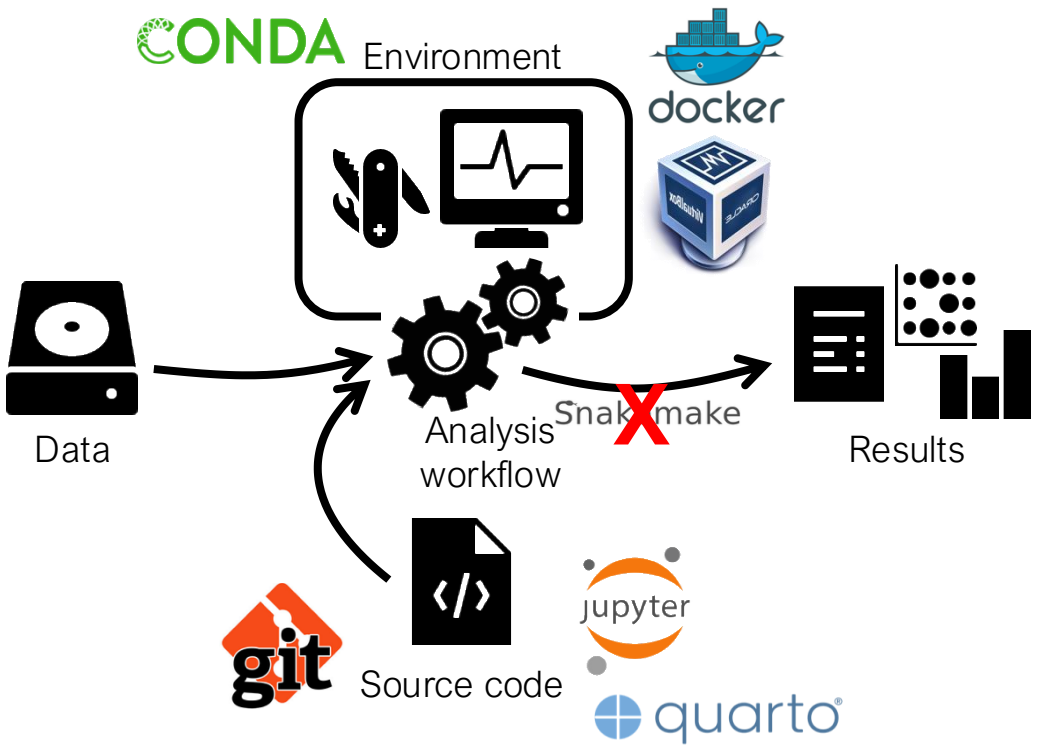
All parts of a computational analysis have to be reproducible!



Slide credit: L. Wigge, R. Ågren & J. Sundh (NBIS & SciLifeLab, Sweden)

Computational Reproducibility: Pieces of the Puzzle

What is covered in today's workshop?



Part 1 — **Version control**
Track and backup your project history

Part 2 — **Environment management**
Setup and manage your project environment

Part 3 — **Virtual machines / containers**
Make your project self-contained and distributable

Part 4 — **Notebooks**
Document your exploratory analysis

Part 5 — **Reproducible computing platforms**
Out-of-the-box computational reproducibility

What about AI?

Computational Reproducibility: Questions?



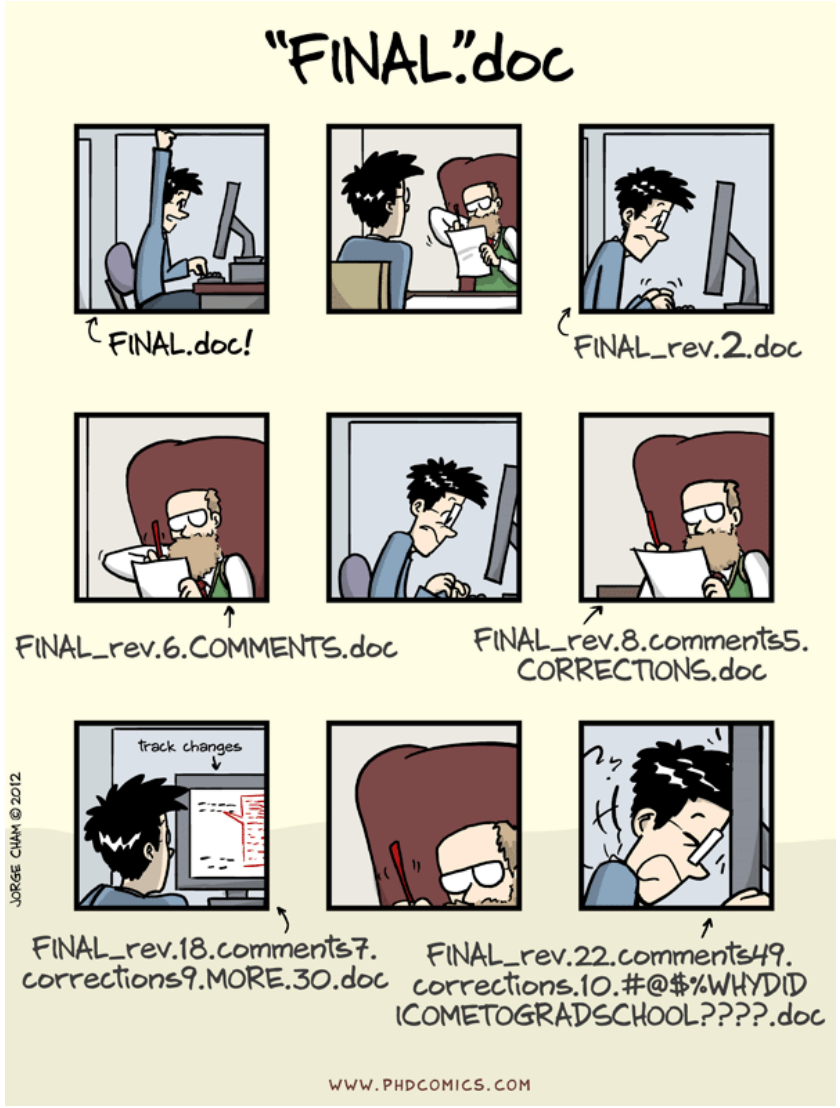
Tell us a bit about yourself



Version Control



Version Control



A **Version Control System** (VCS):

- **Tracks changes** in files over time.
- Allows you to **revert to earlier versions.**
- Makes **collaboration systematic** rather than chaotic.

Git is the most widely used version control system today.

- Git is a **distributed version control system**
  - Every contributor has a **local** copy of the project history.
  - Changes can be **synchronized** between **local** and **remote repositories** (e.g., using GitHub or GitLab).
  - Contributors can work on the same project across **multiple computers**.

- **Git** is free and open source.
- Install **Git** for Windows, Mac, and Linux.

How Git Works: Basic Workflow

A Git project typically has **four main components**:

Working Directory

The local folder where you edit your files



Staging Area

A holding space where you preview selected changes before committing them



Local repository (.git folder)

Where committed snapshots of your project are stored



Remote Repository

An online copy for collaboration and backup, hosted on a cloud service



Git can be used through:

- **Command-line interfaces** (CLI): e.g., the Terminal.
- **Graphical user interfaces** (GUI): e.g., GitHub Desktop, RStudio, etc. See the list of Git GUI Clients.
- **Web interfaces**: e.g., GitHub and GitLab in a browser.

How Git Works: Basic Workflow

Let's see an example: Drawing a robot

Working Directory

The local folder where you edit your files

Staging Area

A holding space where you preview selected changes before committing them

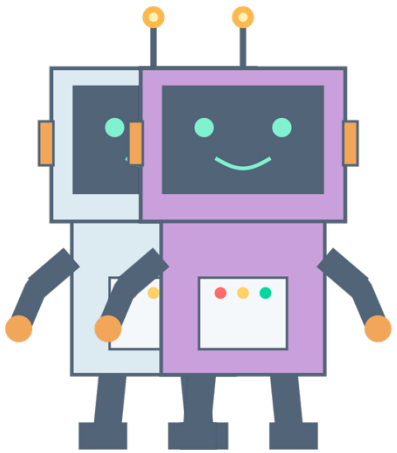
Local repository (.git folder)

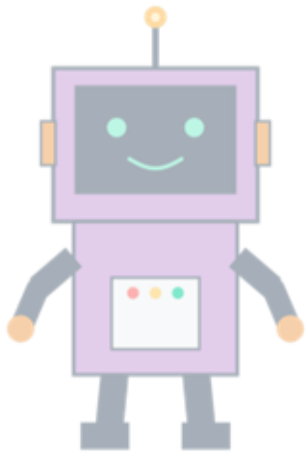
Where committed snapshots of your project are stored

`git init`

Remote Repository

An online copy for collaboration and backup, hosted on a cloud service





`git add [file]`

How Git Works: Basic Workflow

Let's see an example: Drawing a robot

Working Directory

The local folder where you edit your files

Staging Area

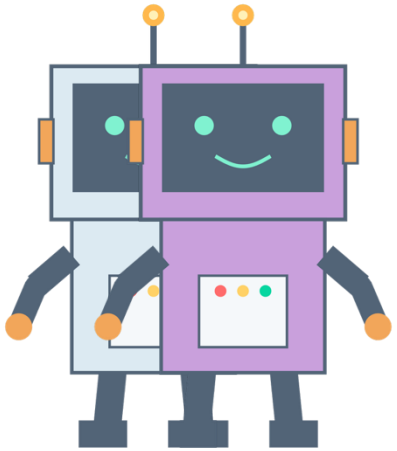
A holding space where you preview selected changes before committing them

Local repository (.git folder)

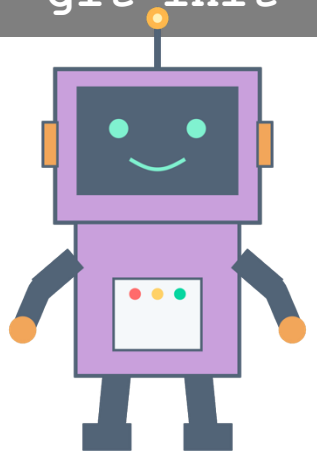
Where committed snapshots of your project are stored

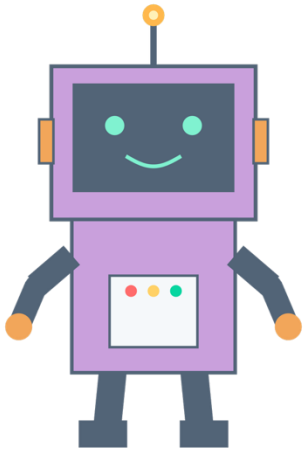
Remote Repository

An online copy for collaboration and backup, hosted on a cloud service



`git init`





`git add [file]` → `git commit -m [message]` → `git push`

How Git Works: Basic Workflow

Let's see an example: Drawing a robot

Working Directory

The local folder where you edit your files

Staging Area

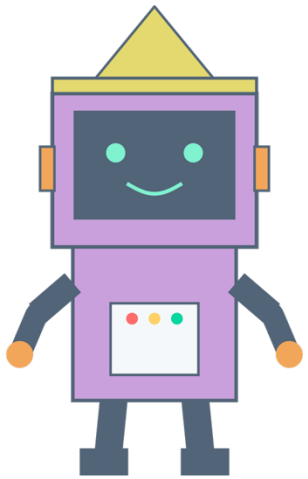
A holding space where you preview selected changes before committing them

Local repository (.git folder)

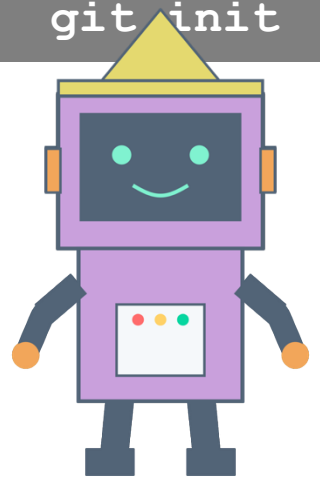
Where committed snapshots of your project are stored

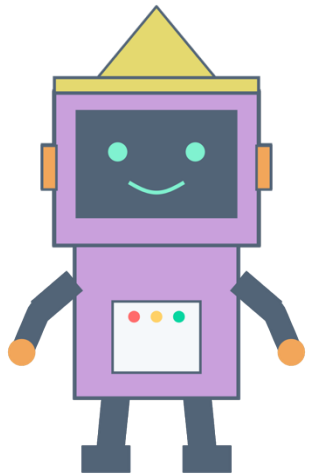
Remote Repository

An online copy for collaboration and backup, hosted on a cloud service



`git init`





`git add [file]`

`git commit -m [message]`

`git push`

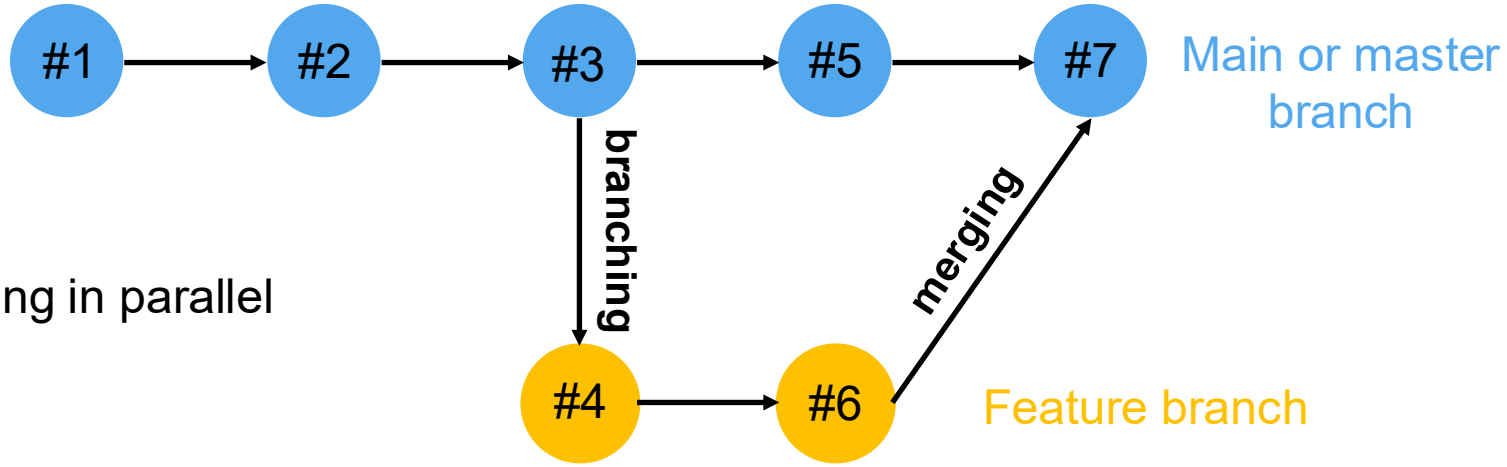
`git pull`

Git branches and collaborative workflows

A **branch** is an independent line of development within the same project.

Branches are used to:

- Test new analyses
- Prepare major updates
- Collaborate with colleagues working in parallel



Merging Branches

New **branches** can be **merged** back into the **main branch**.

Merge Conflicts: If the same part of a file was modified differently in two branches, Git will flag a conflict, which must be resolved manually. After merging, you must always rerun and validate your workflow.

Git practice

DEMO

In the Terminal



https://siscourses.ethz.ch/reproducible_computing/

Change Directory to the project folder

Check Git version installed

Initialize a new Git repository in the current directory

Check the current state of files in the working directory

Add all changes to the Staging Area

Check the current state of files in the working directory

Commit changes to the .git local repository

```
[lmarques ~ $ cd rdm_ws3_git
[lmarques rdm_ws3_git $ git --version
 git version 2.56.0
[lmarques rdm_ws3_git $ git init
 Initialized empty Git repository in /Users/lmarques/rdm_ws3_git/.git/
[lmarques rdm_ws3_git $ git status
 On branch main

 No commits yet

 Untracked files:
   (use "git add <file>..." to include in what will be committed)
         draw_robot.R
         rdm_ws3_git.Rproj

 nothing added to commit but untracked files present (use "git add" to track)
[lmarques rdm_ws3_git $ git add *
[lmarques rdm_ws3_git $ git status
 On branch main

 No commits yet

 Changes to be committed:
   (use "git rm --cached <file>..." to unstage)
         new file:   draw_robot.R
         new file:   rdm_ws3_git.Rproj

[lmarques rdm_ws3_git $ git commit -m "created git repo and added project"
 [main (root-commit) 338af64] created git repo and added project
  2 files changed, 150 insertions(+)
  create mode 100644 draw_robot.R
  create mode 100644 rdm_ws3_git.Rproj
```

Check the current state of files in the working directory

Open R project and run script to create figure

Check the current state of files in the working directory

Add changed file to the Staging Area

Check the current state of files in the working directory

Commit changes to the .git local repository

Check the current state of files in the working directory

```
[lmarques rdm_ws3_git $ git status
On branch main
nothing to commit, working tree clean
[lmarques rdm_ws3_git $ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        robot_antenna.png

nothing added to commit but untracked files present (use "git add" to track)
[lmarques rdm_ws3_git $ git add robot_antenna.png
[lmarques rdm_ws3_git $ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   robot_antenna.png

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

[lmarques rdm_ws3_git $  git commit -m "added figure robot_antena"
[main 96ebc1c] added figure robot_antena
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 robot_antenna.png
[lmarques rdm_ws3_git $ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

Display the history of commits (who changed what and when)

```
[lmarques rdm_ws3_git $ git log
commit 96ebc1ca6678aef58a92242a2ef0ab845a63cdbb (HEAD -> main)
Author: Laura Marques <laura.marques@ethz.ch>
Date:   Tue Oct 20 20:48:56 2026 +0200

    added figure robot_antena

commit 338af64c3c5d27f16b325b3668a36a5e598e0f92
Author: Laura Marques <laura.marques@ethz.ch>
Date:   Tue Oct 20 20:47:10 2026 +0200

    created git repo and added project
```

Add changed file to the Staging Area

```
[lmarques rdm_ws3_git $ git add .gitignore
```

Check the current state of files in the working directory

```
[lmarques rdm_ws3_git $ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
```

Commit changes to the .git local repository

```
[lmarques rdm_ws3_git $ git commit -m "added gitignore file"
[main 315b195] added gitignore file
 1 file changed, 4 insertions(+)
 create mode 100644 .gitignore
```

Check the current state of files in the working directory

```
[lmarques rdm_ws3_git $ git status
On branch main
nothing to commit, working tree clean
```

Show all branches

```
[lmarques rdm_ws3_git $ git branch
* main
```

Create a new branch

```
[lmarques rdm_ws3_git $ git branch test
```

Show all branches

```
[lmarques rdm_ws3_git $ git branch
* main
  test
```

Switch to a branch

```
[lmarques rdm_ws3_git $ git checkout test
Switched to branch 'test'
```

Show all branches

```
[lmarques rdm_ws3_git $ git branch
  main
* test
```

Edit R script and create new figure

Check the current state of files in the working directory

```
lmarques rdm_ws3_git $ git status
On branch test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   draw_robot.R

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        robot_hat.png

no changes added to commit (use "git add" and/or "git commit -a")
```

Add all changes to the Staging Area

```
lmarques rdm_ws3_git $ git add *
```

Check the current state of files in the working directory

```
lmarques rdm_ws3_git $ git status
On branch test
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   draw_robot.R
        new file:   robot_hat.png
```

Commit changes to the .git local repository

```
lmarques rdm_ws3_git $ git commit -m "added figure robot_hat"
[test 9a18978] added figure robot_hat
 2 files changed, 13 insertions(+), 13 deletions(-)
 create mode 100644 robot_hat.png
```

Check the current state of files in the working directory

```
lmarques rdm_ws3_git $ git status
On branch test
nothing to commit, working tree clean
```

Show all branches

```
lmarques rdm_ws3_git $ git branch
  main
* test
```

Switch to a branch

```
lmarques rdm_ws3_git $ git checkout main
Switched to branch 'main'
```

Show all branches

```
lmarques rdm_ws3_git $ git branch
* main
  test
```

Integrates changes from a branch into the current branch

```
lmarques rdm_ws3_git $ git merge test
Updating 315b195..9a18978
Fast-forward
 draw_robot.R  |  26 ++++++++++++++----------
 robot_hat.png | Bin 0 -> 65419 bytes
 2 files changed, 13 insertions(+), 13 deletions(-)
 create mode 100644 robot_hat.png
```

Important Git Commands



`git init`	Initialize a new Git repository in the current directory
`git clone [url]`	Download an existing repository, including its full history
`git add [file]`	Add selected file changes to the staging area
`git commit -m [message]`	Save a snapshot of staged changes in the local repository
`git status`	Show the current state of files in the working directory
`git log`	Display the history of commits (who changed what and when)
`git pull`	Update your local repository with changes from the remote
`git push`	Upload your local commits to the remote repository
`git branch`	Show all branches
`git branch [branch-name]`	Create a new branch
`git checkout [branch-name]`	Switch to a branch
`git checkout -b [branch-name]`	Create and switch in one step
`git merge [branch-name]`	Integrates changes from a branch into the current branch

Collaboration Beyond Your Local Computer

Local repositories



PUSH / PULL



Remote repositories





github.com

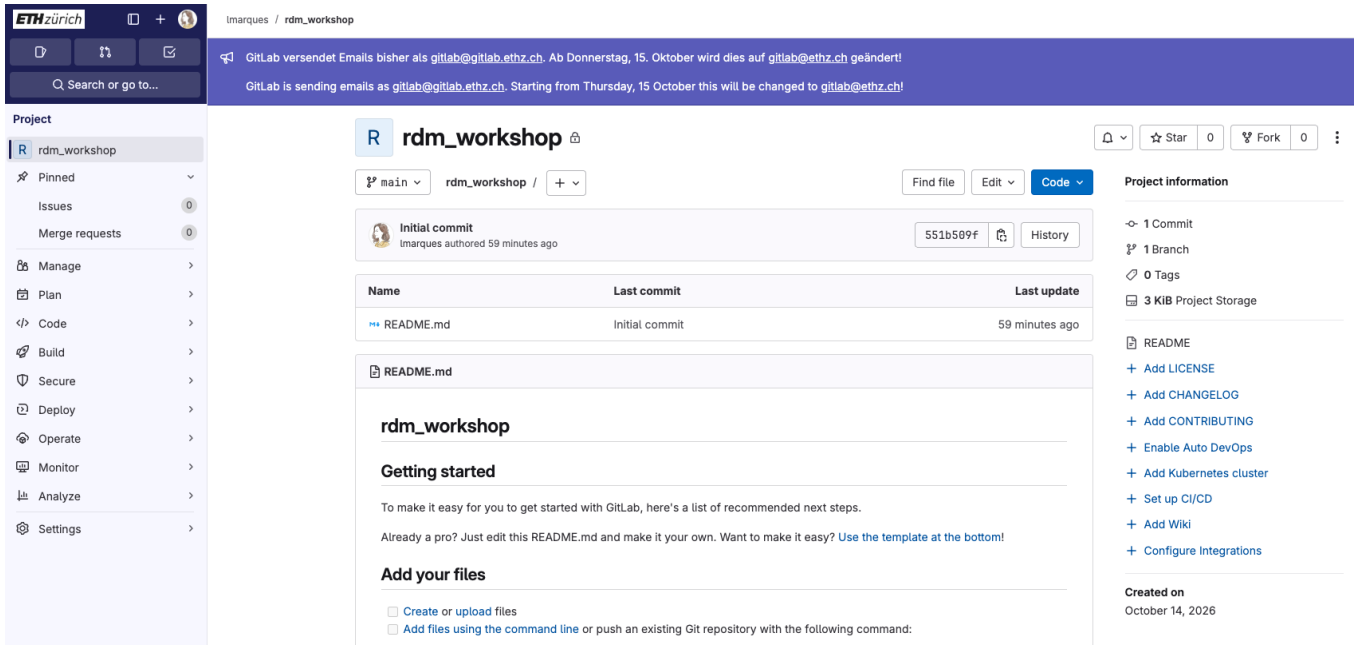
gitlab.com

- **Remote Backups:** Cloud repository hosting.
- **Transparent Tracking:** Full visibility into project history.
- **Collaborative workspace:** Integrated code reviews, issue tracking, and documentation.
- **Structured Teamwork:** Controlled access & role permissions.
- **Data Integrity:** Prevents accidental overwrites & lost work.
- **Open Science:** Facilitates FAIR principles & public code sharing.

ETH Zurich GitLab Service

An institutional GitLab instance gitlab.ethz.ch

- Recommended, and in some cases mandatory - particularly when working with sensitive data.
- Log in directly with your ETH credentials.
- Free for repositories up to 2 GB.
- Personal (private/public) and group repositories.
- Integrated Wiki: Built-in notebook for documentation.
- Container Registry: Build, store, and share containers within the platform.
- Easy export of repositories (e.g., to GitHub).



Git – General Recommendations



Good Git Practices

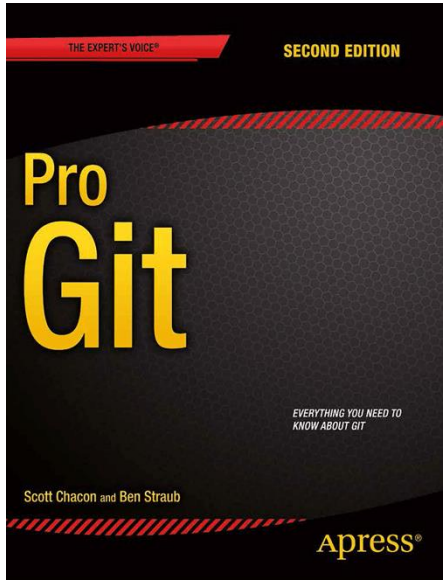
- Commit often and write clear commit messages.
- Avoid storing large data files:
  - e.g., images, movies, binary files.
  - Consider using Git Large File Storage (LFS).
- Synchronize carefully:
  - git pull before starting work.
  - git push when you are done.
- Use a .gitignore file to prevent unnecessary, large, or sensitive files from being tracked.
- When resolving and merging, verify your workflow.
- Git never forgets: Once committed, data remains in the project history, even if deleted later.

Git – Resources



Resources for getting started with Git

- Pro Git book by S. Chacon & B. Straub (free online)
- Numerous tutorials available on the web / YouTube
  - Software Carpentry Git course Online course
  - Swiss Institute of Bioinformatics (SIB) training: Version Control with Git (2-day Online Course with free course materials)
  - Happy Git and GitHub for the useR Online Book





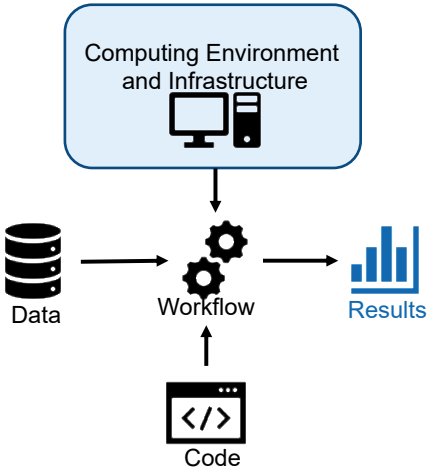
Version Control: Questions?



Managing Dependencies & Computing Environments

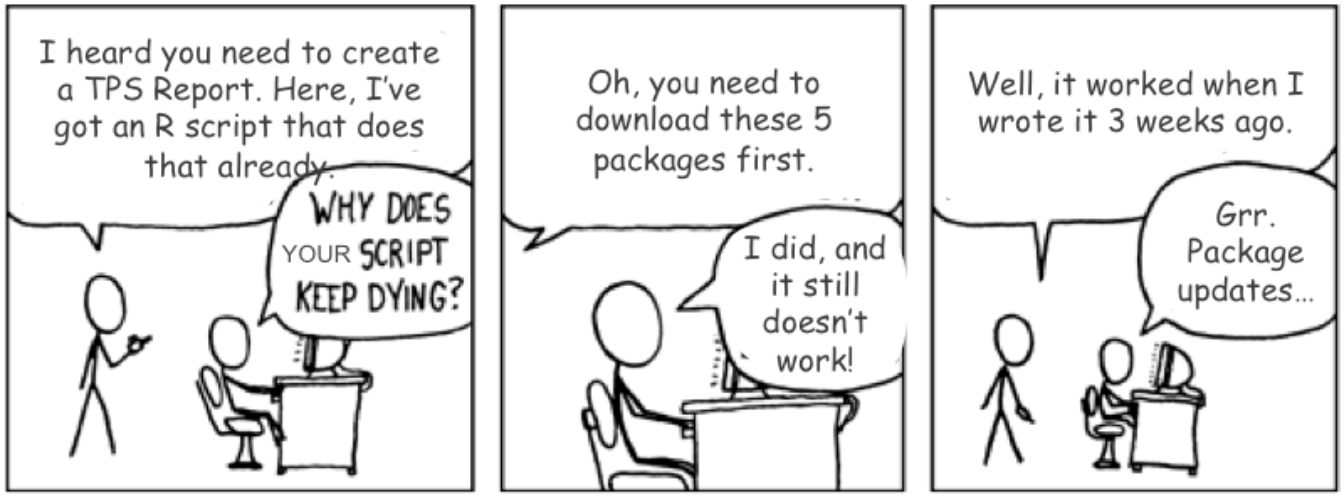


Reproducible Computing Environment

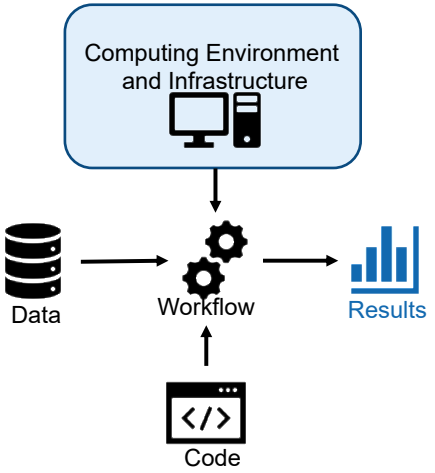


Problem:

Full reproducibility requires the possibility to recreate the system that was originally used to generate the results



Reproducible Computing Environment



Problem:

Full reproducibility requires the possibility to recreate the system that was originally used to generate the results

Solution:

Bundle your application and all dependencies

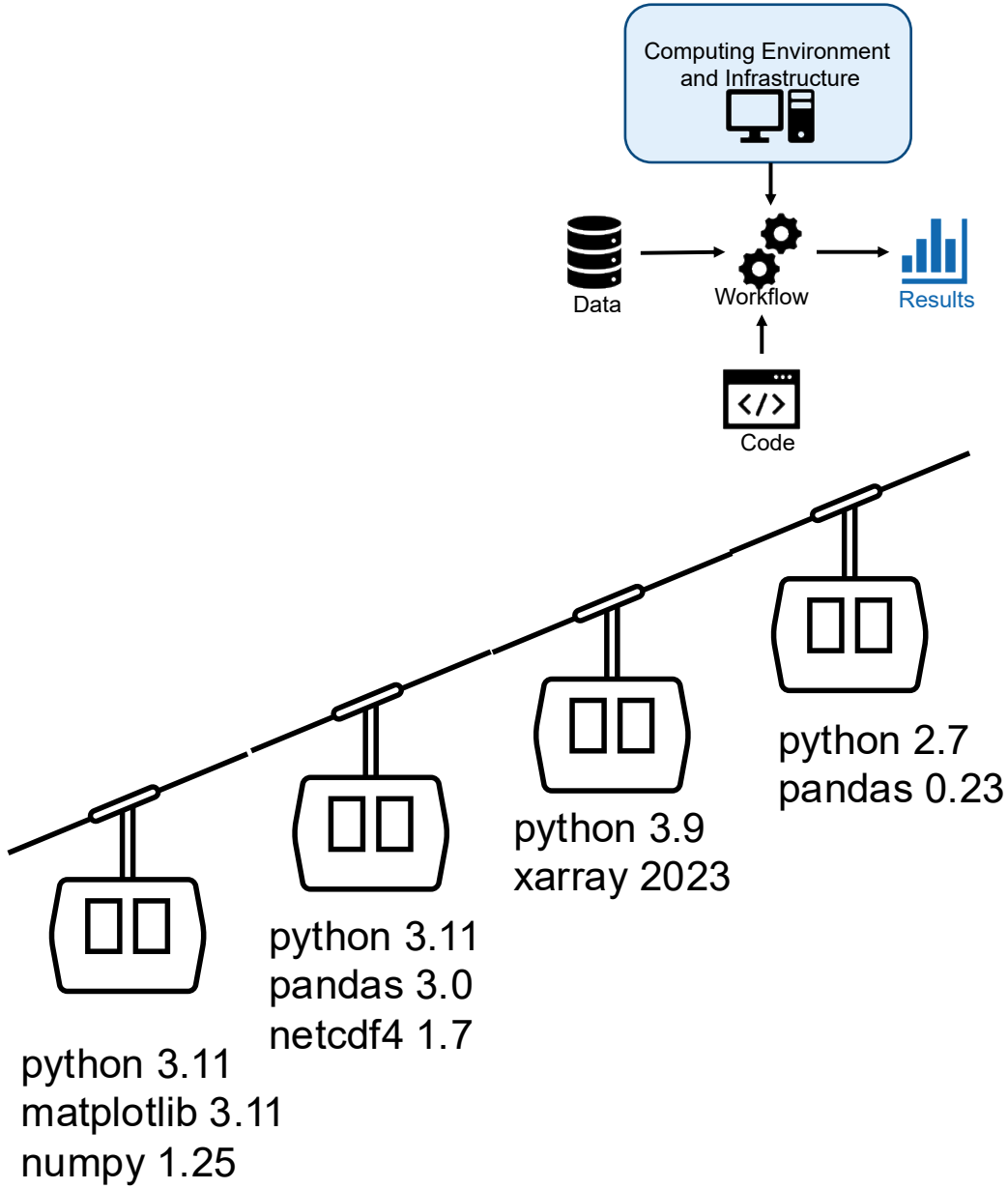
→ Environment Isolation & Dependency management

Tools:

- Application / software level: Conda, pip, virtualenv, renv, Devbox
- Containerization: Docker, Podman, Apptainer
- Virtualization (Virtual Machine, VM): VirtualBox, Vmware, Parallels

Virtual Computing Environments

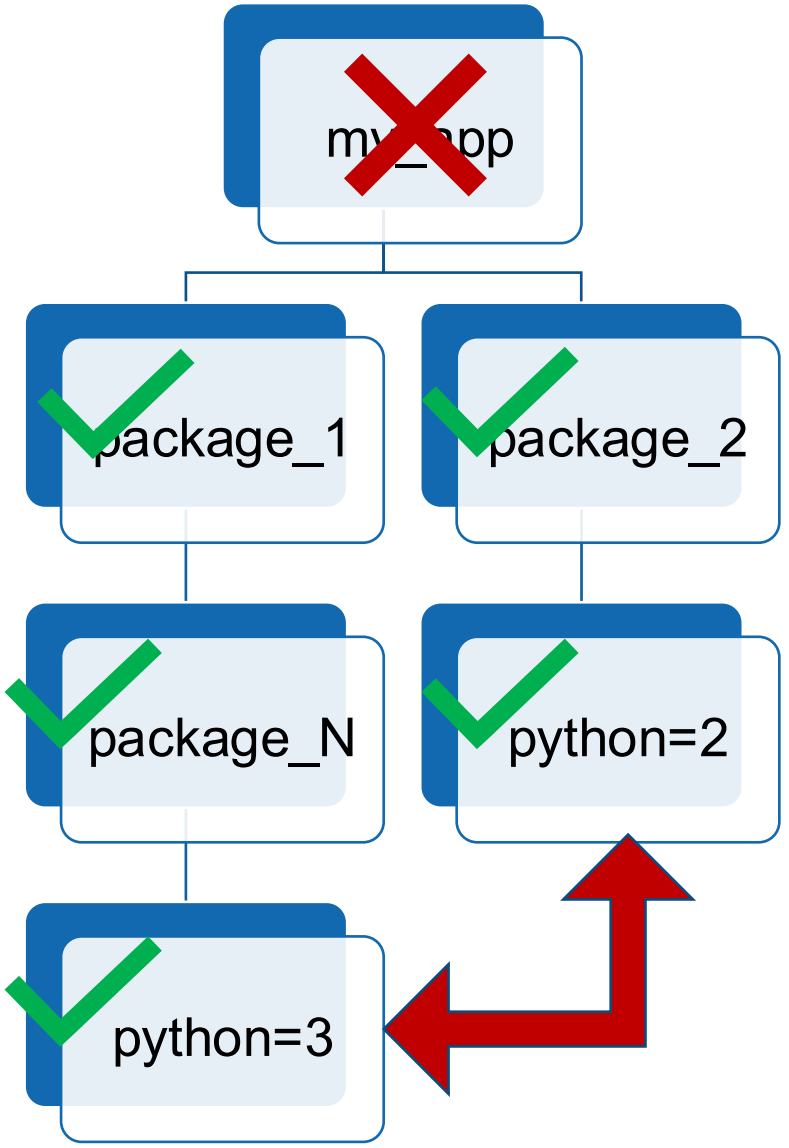
- Separated, self-contained builds of your coding language of choice (e.g. python, R, Julia, etc.)
- Great for projects that are solely in your coding language
- Some tools also work for projects with dependencies in multiple languages
- We recommend one virtual environment for each coding project



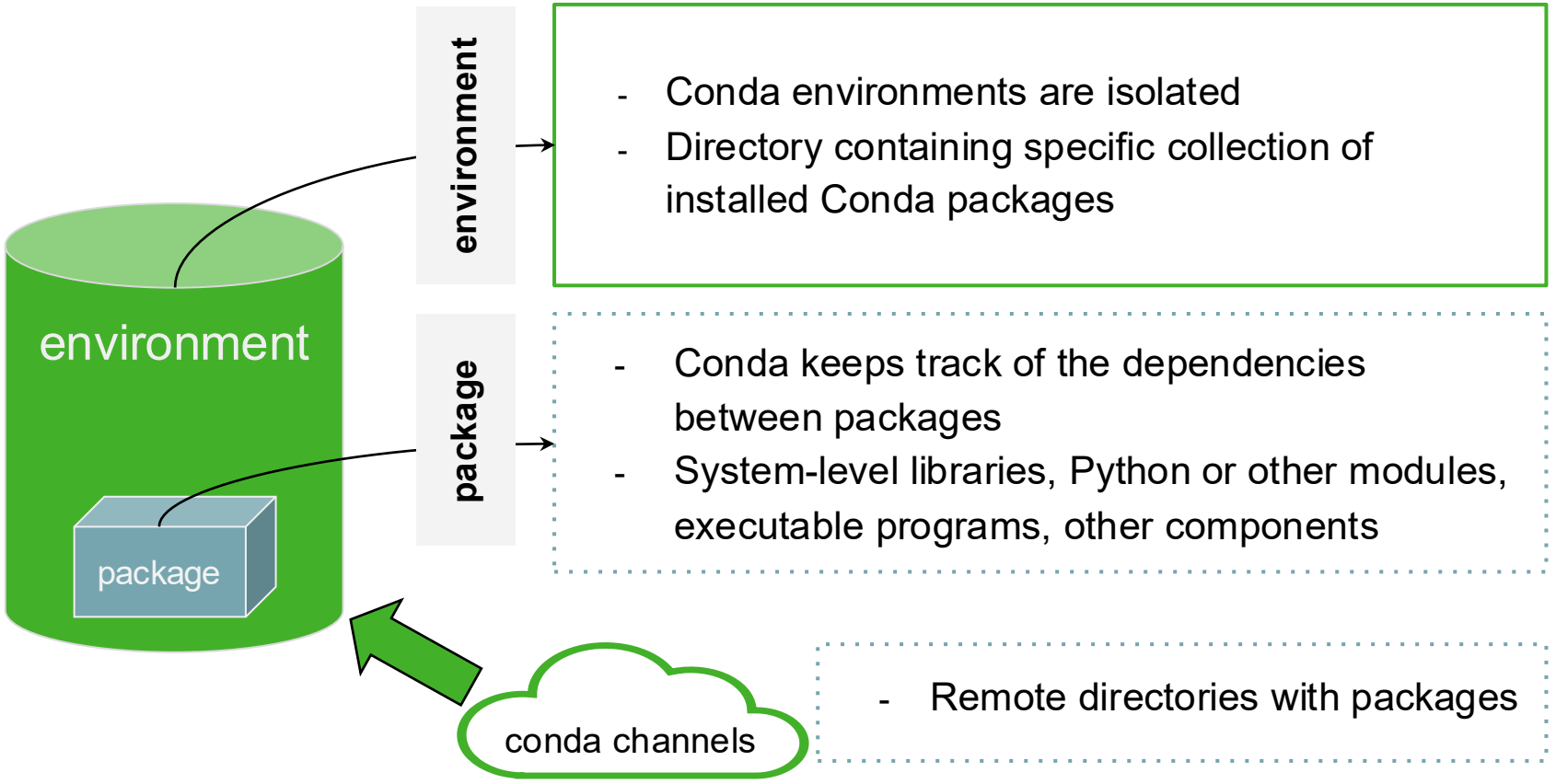
Slide credit: E. Knappe (Lib4RI)

Reproducible Environment for R and Python

- Open source: Anaconda, Miniconda, Miniforge
- Commercial support: Anaconda Enterprise
  - ***Note:*** *certain functionality requires a paid license outside education / academia*
- Multi-platform: Windows, macOS, Linux
- Environment Management System
  - Isolated computing environments on the same system
  - Documentation of the computing environment
- Package Management System
  - Supported programming Languages: Python, R, …
  - System libraries shipped in binary format
  - Resolve dependencies & conflicts between packages



Conda in a Nutshell



environment.yml

```
channels:
- conda-forge

dependencies:
- python=3.8
- jupyterlab
```

Conda automatically creates an environment file with packages and dependencies

Slide credit: L. Wigge, R. Ågren & J. Sundh (NBIS & SciLifeLab, Sweden)

Environment and Package Management Systems

Language	Environment Management	Package Management	Comments
Python 2 (not supported)	virtualenv, conda	pip, conda	
Python 3	venv, virtualenv, pipenv poetry, uv, conda	pip, pipenv, poetry, uv, conda	Not all can install different Python versions
R	renv, conda	renv, conda	only conda can install different R versions
Julia	Pkg, conda	Pkg, conda	conda provides outdated Julia versions
Matlab	N/A	Add-on manager, Matlab Package Manager (unofficial)	Matlab search path determines dependencies



Alternatives to Conda are emerging!

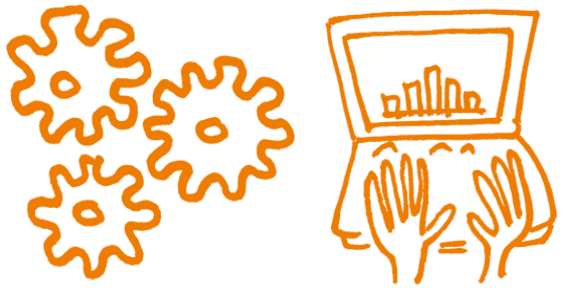


pixi

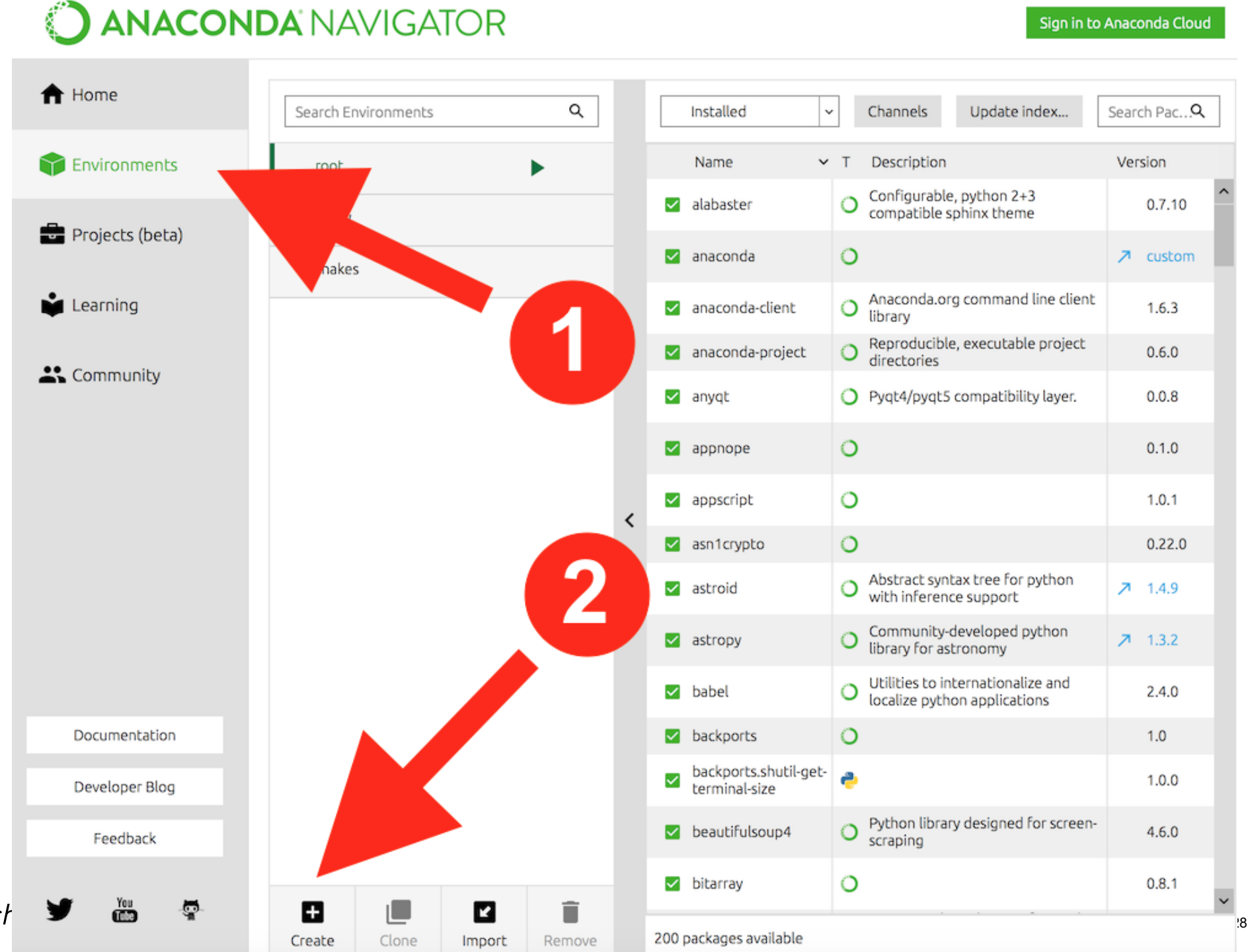


Devbox

Conda Hands-on Session



https://siscourses.ethz.ch/reproducible_computing/Conda.slidy.html


ANACONDA NAVIGATOR
Sign in to Anaconda Cloud
Home
Environments
Projects (beta)
Learning
Community
Documentation
Developer Blog
Feedback
Search Environments
root
nakes
Create
Clone
Import
Remove
Installed
Channels
Update index...
Search Pac...
Name
T
Description
Version
alabaster
Configurable, python 2+3 compatible sphinx theme
0.7.10
anaconda
custom
anaconda-client
Anaconda.org command line client library
1.6.3
anaconda-project
Reproducible, executable project directories
0.6.0
anyqt
Pyqt4/pyqt5 compatibility layer.
0.0.8
appnope
0.1.0
appscript
1.0.1
asn1crypto
0.22.0
astroid
Abstract syntax tree for python with inference support
1.4.9
astropy
Community-developed python library for astronomy
1.3.2
babel
Utilities to internationalize and localize python applications
2.4.0
backports
1.0
backports.shutil-get-terminal-size
1.0.0
beautifulsoup4
Python library designed for screen-scraping
4.6.0
bitarray
0.8.1
200 packages available
1
2

Conda - What can go wrong?

- The package metadata (dependency list) is updated (not very likely)
- The package is deleted by the owner
- The package is not available under another platform
- There is no conda package for what you are looking for
- Complex dependencies may fail or take a long time to resolve

Virtualizing Computing Environments





Conda - What can go wrong?

- The package metadata (dependency list) is updated (not very likely)
- **The package is deleted by the owner**
- **The package is not available under another platform**
- **There is no conda package for what you are looking for**
- Complex dependencies may fail or take a long time to resolve

Reproducible Environment

Problem:

Full reproducibility requires the possibility to recreate the system that was originally used to generate the results

Solution:

Bundle your application and all dependencies

→ Environment Isolation & Dependency management

Tools:

- Application / software level: Conda, pip, virtualenv, renv
- Containerization: Docker
- Virtualization (Virtual Machine, VM): VirtualBox, VMware, Parallels

Reproducible Environment – Virtual Machines



- A virtual machine (VM) is an operating system (“guest”) that runs inside another computing environment (“host”).

- **Advantages:**
  - Allows multiple OS environments on a single physical computer
  - VMs are widely available and are easy to manage, maintain and distribute
  - Offers application provisioning and disaster recovery options

- **Drawbacks:**
  - They are not as efficient as a physical computer because the hardware resources are distributed in an indirect way.
  - Multiple VMs running on a single physical machine can deliver unstable performance

Source: https://searchservervirtualization.techtarget.com/definition/virtual-machine

Reproducible Environment – Containers

- A mini, more flexible, virtual machine
- Great for projects that require specific system requirements / applications / tools / OS
- Larger than a virtual environment but even better for long-term reproducibility
- Docker / Podman / Colima / Apptainer are programs that help you create, run and deliver containers





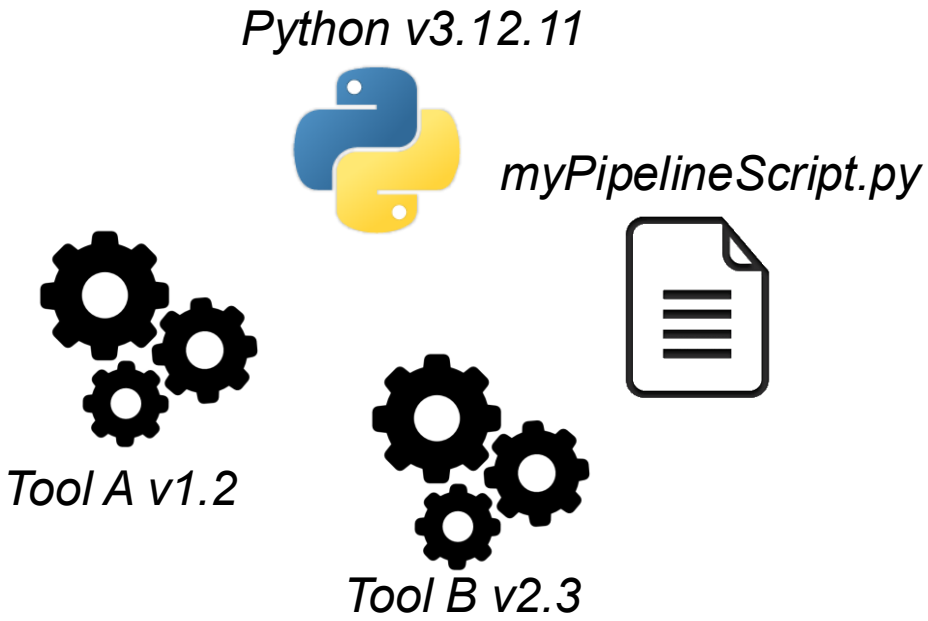
podman

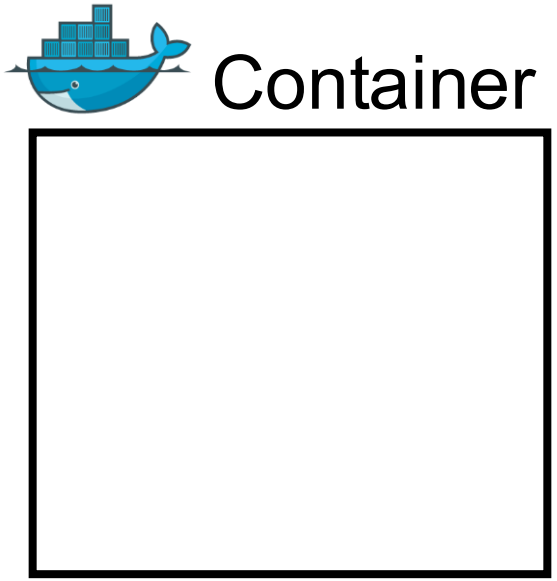




Reproducible Environment – Containers

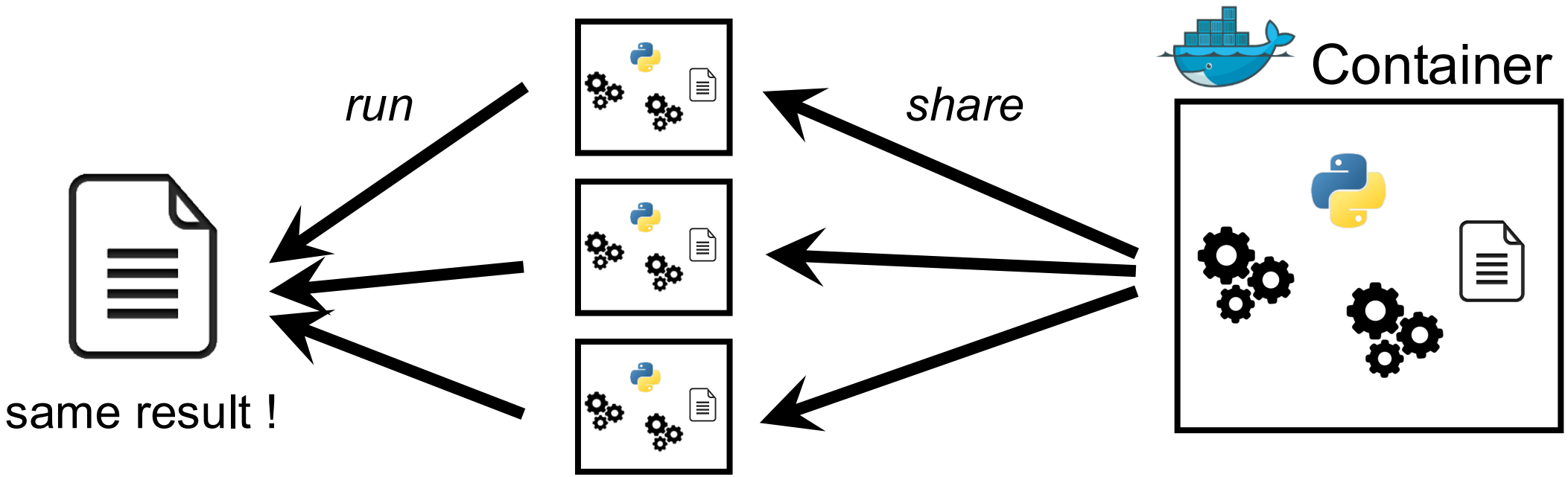
- **Containers** allow you to **package** your software / pipeline with the **dependencies** inside a **reproducible**, easy to **share**, **runnable** file
- Example: **Docker containers**





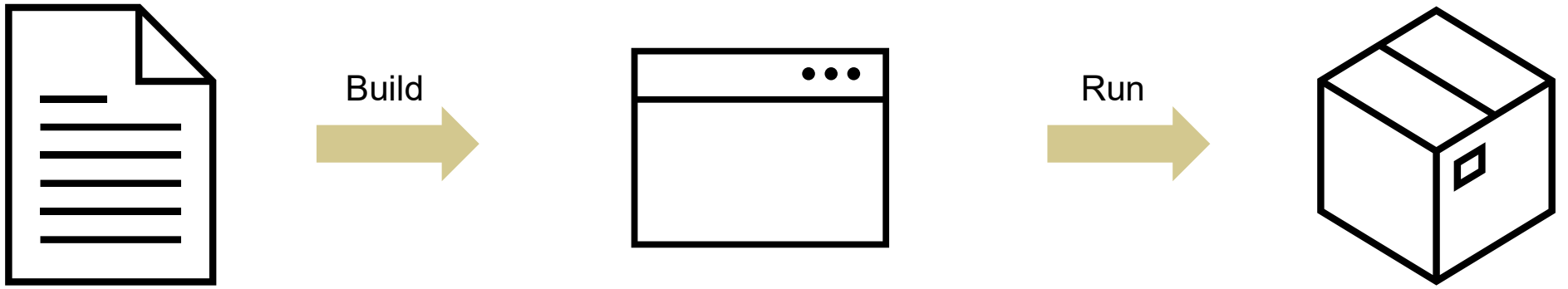
Reproducible Environment – Containers

- **Containers** allow you to **package** your software / pipeline with the **dependencies** inside a **reproducible**, easy to **share**, **runnable** file
- Example: **Docker containers**



Reproducible Environment – Containers

Pull back the curtain on containerization



Dockerfile

Docker image

Docker container

Small text file with instructions

Packaged snapshot of everything
(OS, packages, code) ready to be used

A running instance of your image,
Package of software

*an instance because you can run
multiple containers from one image

Slide credit: E. Knappe (Lib4RI)

Reproducible Environment – Containers

Pull back the curtain on containerization

- Write or use an existing Dockerfile

```
FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD [ "python", ".scripts/wind_speed_at_dorthys_house.py" ]
```


Slide credit: E. Knappe (Lib4RI)

Reproducible Environment – Containers

Pull back the curtain on containerization

- Write or use an existing Dockerfile

Dockerfile for pflotran



https://groundwater.usu.edu/

Slide credit: E. Knappe (Lib4RI)

Reproducible Environment – Containers

Pull back the curtain on containerization

- Write or use an existing Dockerfile
- Build the image
  - Or use an existing one: https://hub.docker.com/search?badges=official

```
docker build -t tornado_analysis
```


Slide credit: E. Knappe (Lib4RI)

Reproducible Environment – Containers

Pull back the curtain on containerization

- Write or use an existing Dockerfile
- Build the image
  - Or use an existing one: https://hub.docker.com/search?badges=official
- Run the container

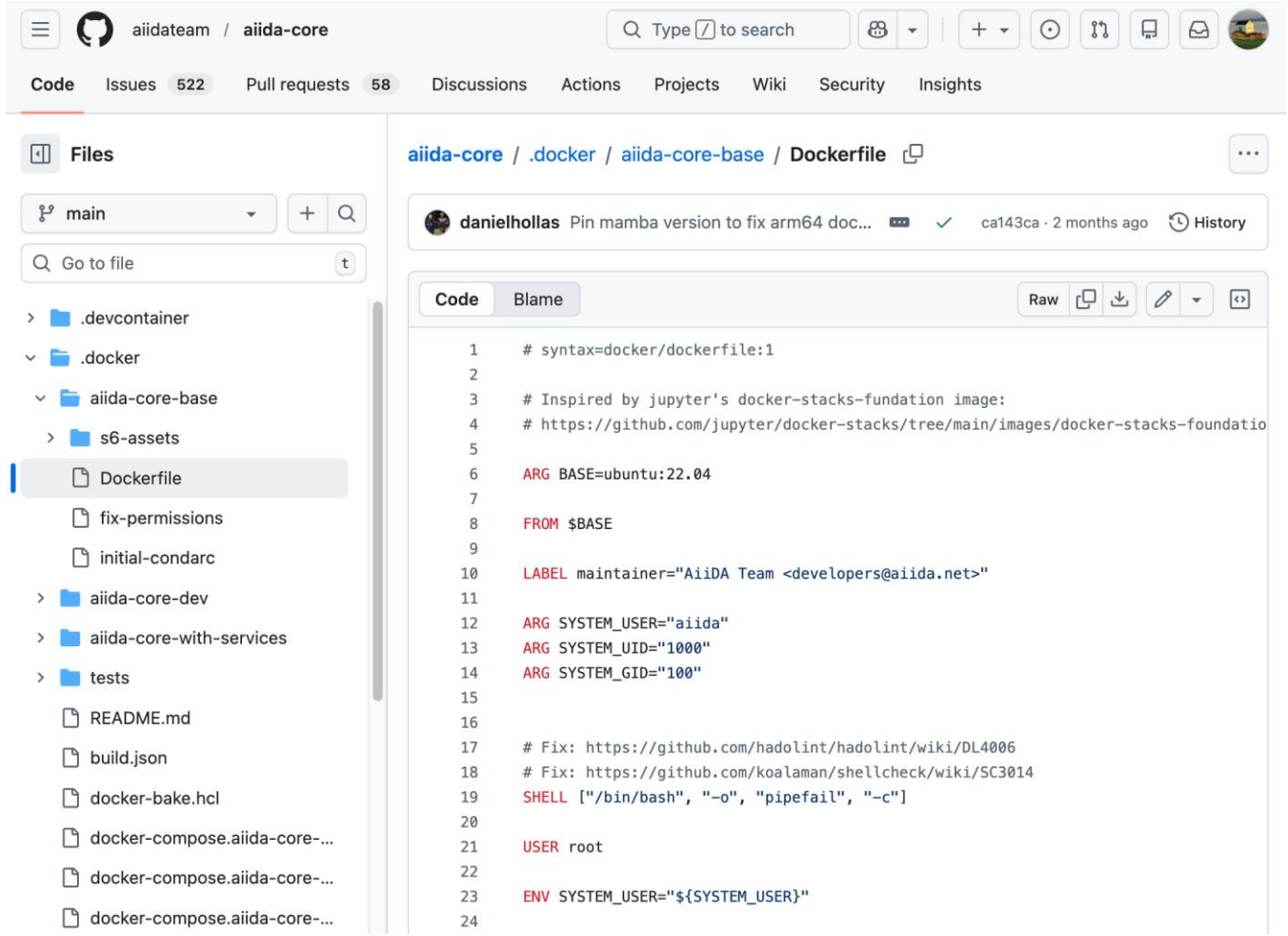
```
docker run tornado_analysis
```


Slide credit: E. Knappe (Lib4RI)

Reproducible Environment – Containers

Pull back the curtain on containerization

- Write or use an existing Dockerfile
- Build the image
  - Or use an existing one: https://hub.docker.com/search?badges=official
- Run the container
- Share your Dockerfile when you share your code

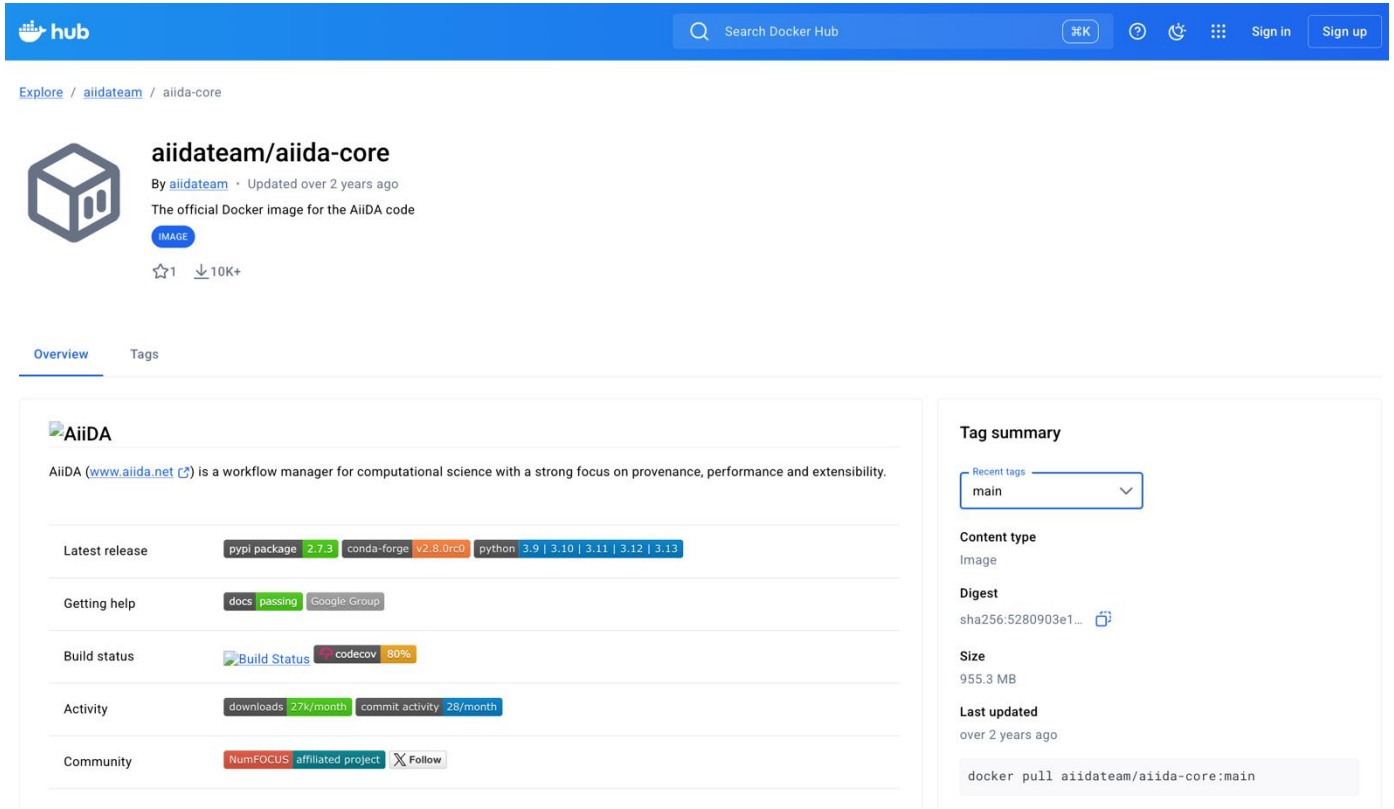


Dockerfile for AiiDA workflow manager on Github

Reproducible Environment – Containers

Pull back the curtain on containerization

- Write or use an existing Dockerfile
- Build the image
  - Or use an existing one: https://hub.docker.com/search?badges=official
- Run the container
- Share your Dockerfile when you share your code
- Archive your Docker **image** for long-term preservation



Docker images for AiiDA workflow manager on Docker Hub

Reproducible Environment – Containers

Difference between Dockerfile, Image and Container

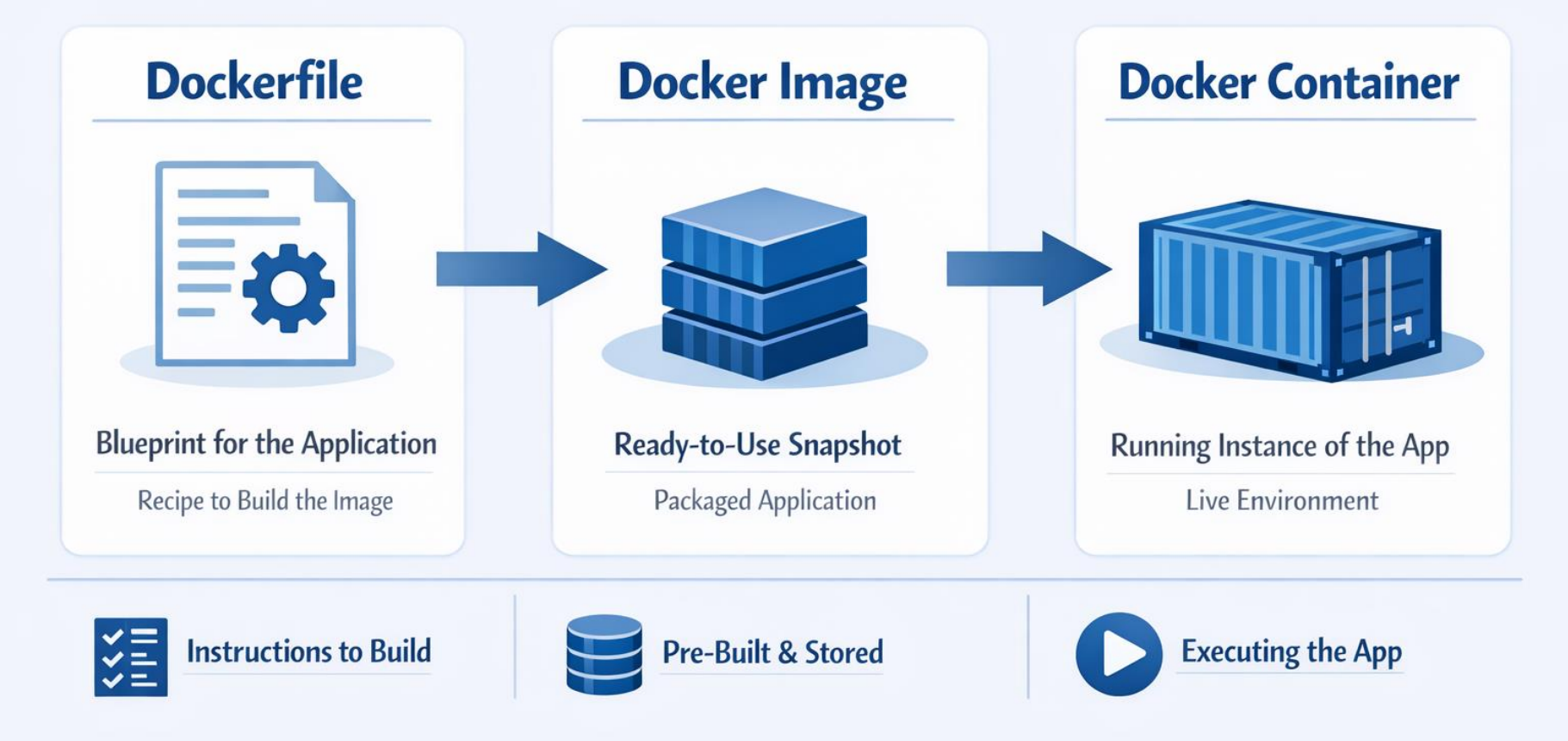


Image generated with DALL·E (OpenAI), March 2026.

Virtual Machines vs Containers

	VMs (Virtual Box)	Containers (Docker)
Use case	Complex Apps (GUI, …)	Data Analysis Scripts, Simple Apps, Microservices, Continuous Integration
Virtualization	Hardware-level	OS-level
Size	GB	MB
Startup time	Minutes	Seconds
Guest OS	Windows, macOS, Linux	Primarily Linux-based
Host OS	Windows, macOS, Linux	Linux, Windows 10 / macOS with hypervisor
Overhead (RAM, CPU)	High - reduced performance	Low - close to native performance
Security	Better (fully isolated)	Poorer (shared kernel)
How to use	Easy if you know to install OS	New things to learn
Getting started	www.virtualbox.org/manual/ch01.html	https://docs.docker.com/get-started/

Reproducible computational environment: Questions?



Literate Programming and Interactive Notebooks



R Markdown

from RStudio

quarto®

Literate Programming and Interactive Notebooks

Literate Programming

- Integrates text, computer code, and results/outputs into a single document.
- + Transparency: Code and outputs are together, making it clear how results were generated.
- + Reproducibility: Results are reproduced by running the entire workflow.

Interactive Notebooks: Digital documents to implement literate programming.



- Commercial
- First notebook interface
- Used in Mathematics



- Commercial
- Used in Engineering and Mathematics



Jupyter Notebook: One document
JupiyterLab: Multiple notebooks

- Open source
- Supports +40 languages (Python, R, Julia, C++, etc)
- Requires Python to run



- Open source
- Supports +40 languages
- Requires R to run



- Open source
- Multi-language native
- Standalone CLI (No language dependencies)

Interactive Notebooks

Quarto

- An open-source scientific and technical publishing system.
- Combines plain text with embedded code (Python, R, Julia) to produce HTML, PDF, and Word docs, etc.
- It can be run via the command line (CLI) or inside IDEs like RStudio, VS Code, Jupyter, and PyCharm.
- Download and install Quarto from quarto.org.

```
lmarques ~ $ quarto --version
1.10.19
lmarques ~ $ cd rdm_ws3_quarto
lmarques rdm_ws3_quarto $ quarto render demo_quarto.qmd

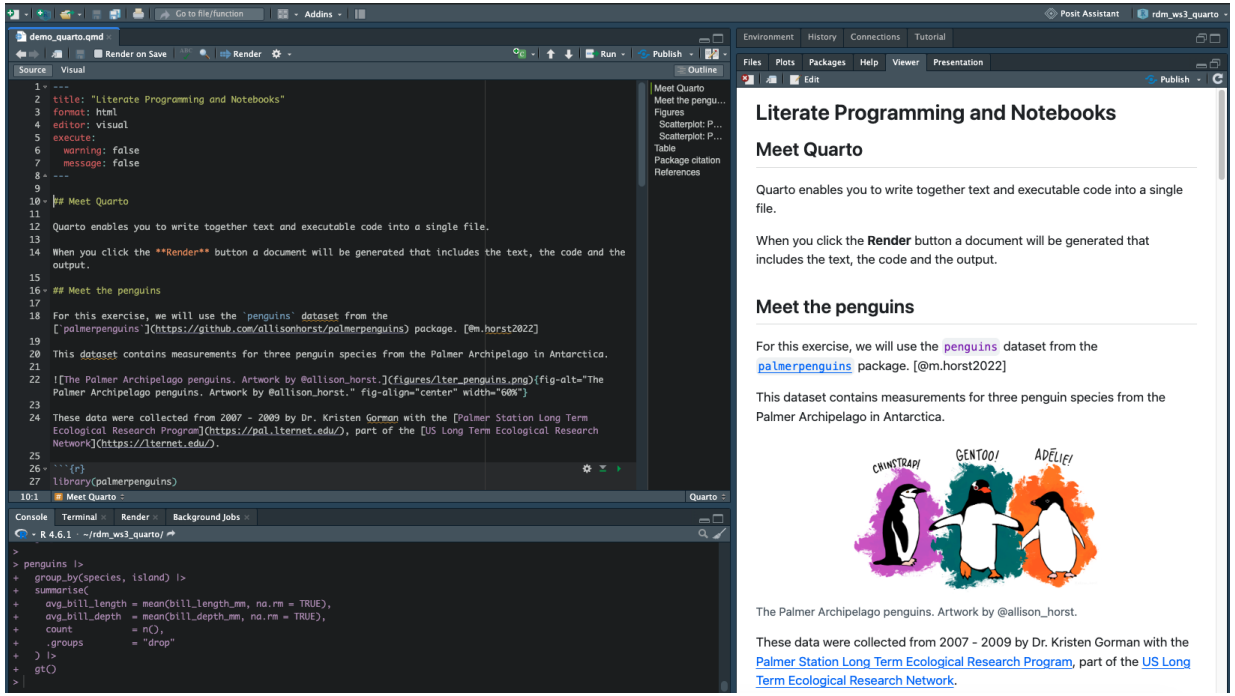

processing file: demo_quarto.qmd
1/13
2/13 [unnamed-chunk-1]
3/13
4/13 [unnamed-chunk-2]
5/13
6/13 [fig-penguins_01]
7/13
8/13 [plot-penguins_02]
9/13
10/13 [tbl-penguins-top10]
11/13
12/13 [unnamed-chunk-3]
13/13
output file: demo_quarto.knit.md

pandoc
  to: html
  output-file: demo_quarto.html
  standalone: true
  section-divs: true
  html-math-method: mathjax
  wrap: none
  default-image-extension: png

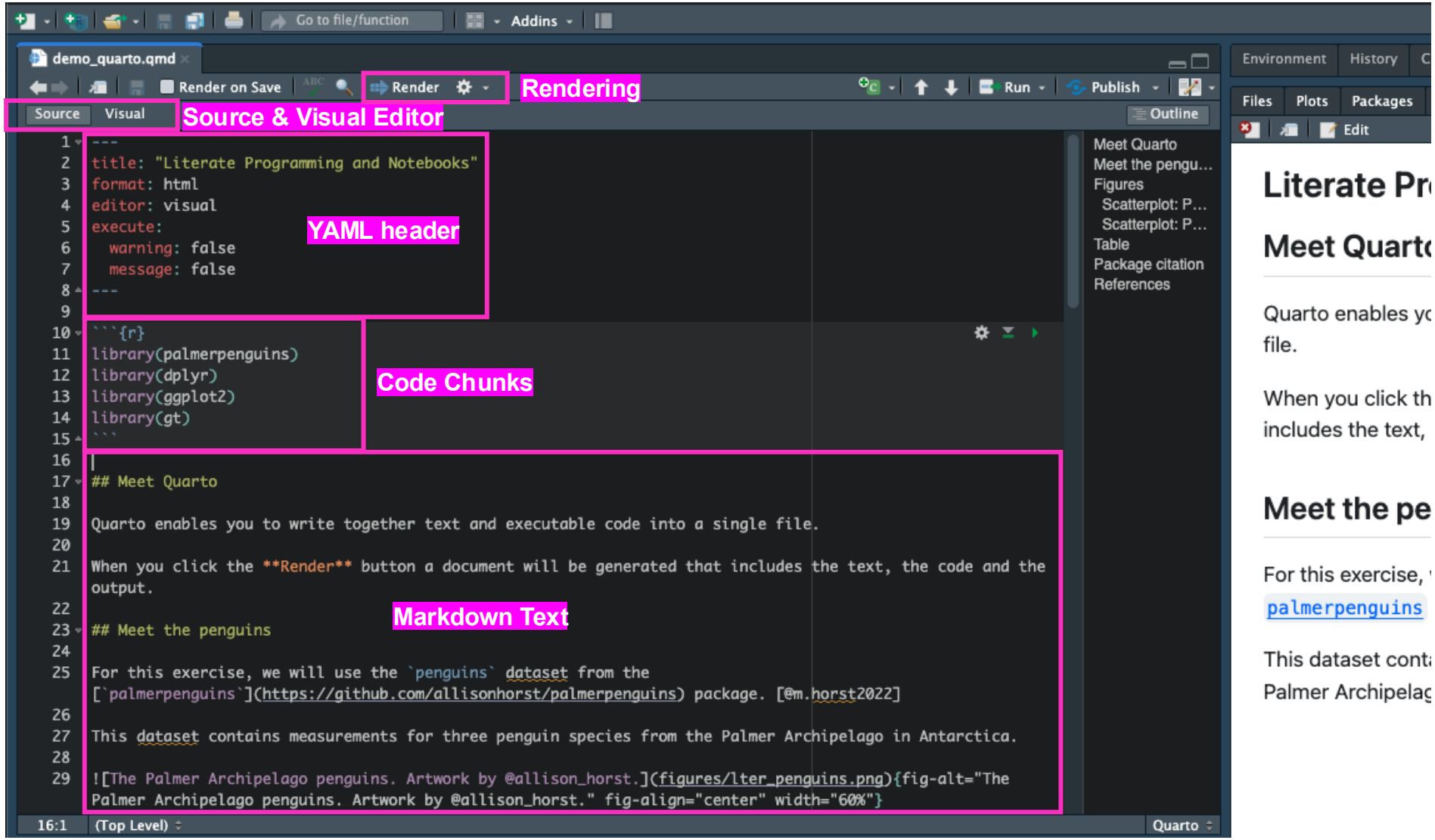
metadata
  document-css: false
  link-citations: true
  date-format: long
  lang: en
  title: Literate Programming and Notebooks
  editor: visual

Output created: demo_quarto.html

lmarques rdm_ws3_quarto $ quarto preview demo_quarto.qmd
```



Main components of Quarto

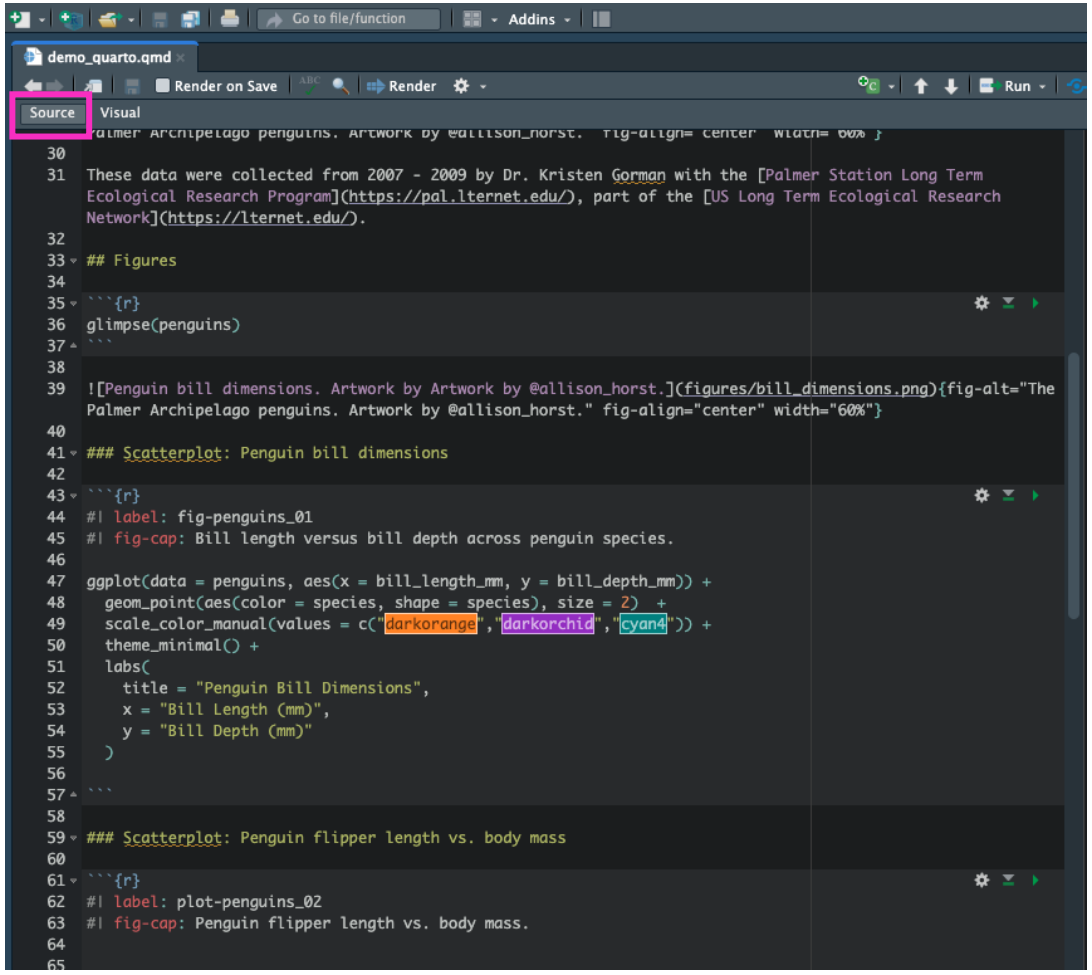


Source vs. Visual Editor in RStudio

Source - Raw Code View

Similar to classic R Markdown.

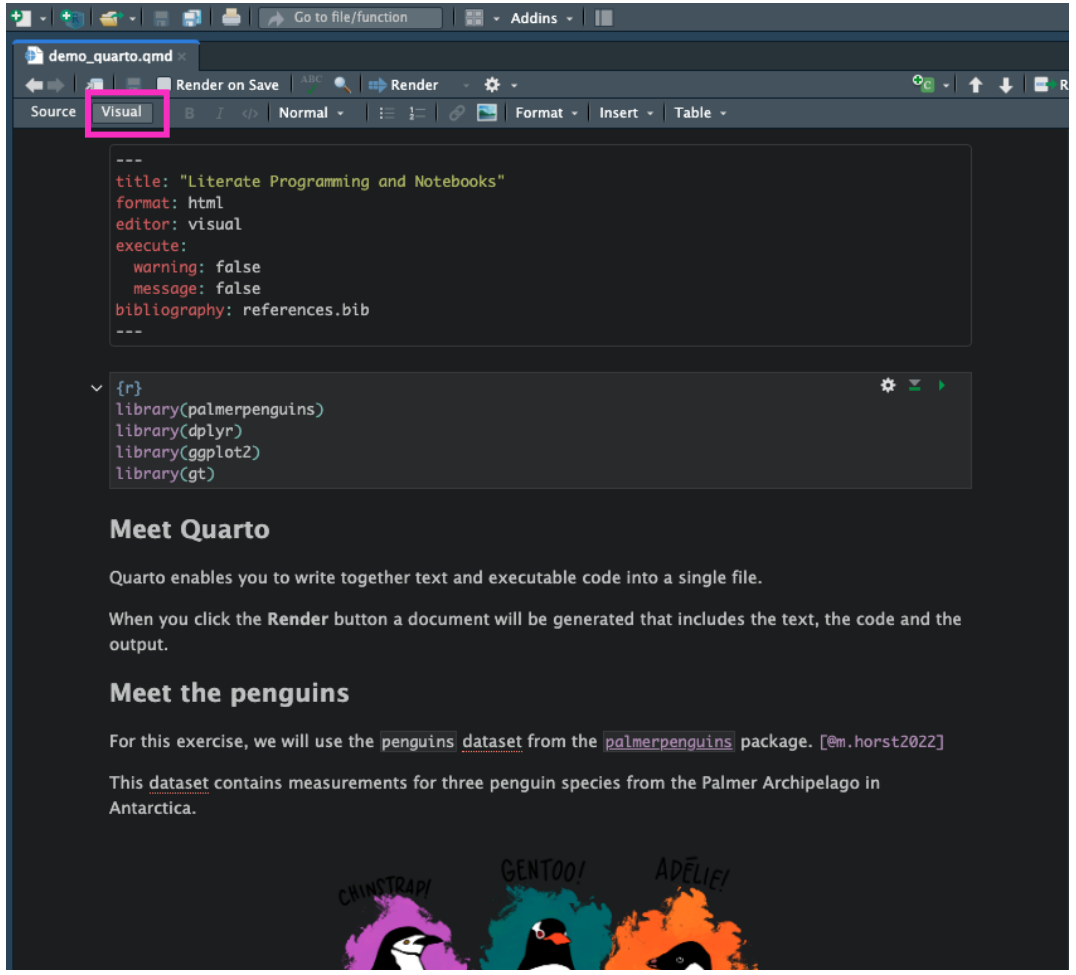
Fast for copy-pasting and precise editing.



Visual - WYSIWYG Mode

MS Word style. Includes UI Tools (point-and-click buttons).

Beginner-friendly: No need to memorize Markdown syntax.



The 3 Building Blocks of a .qmd File

YAML Header

- Defines the output formats: HTML, PDF, Word, etc.
- Controls document metadata and settings.
- Applies across the document.

Code Chunks

- Executable code.
- Generates tables and charts.
- Include results in the document.
- Code can be hidden.

Markdown Text

- Plain text and human-readable.
- Allows for headers, text formatting, images, links and citations.

```yaml
---
title: "Literate Programming and Notebooks"
format: html
editor: visual
execute:
  warning: false
  message: false
  echo: true
---
```

````
```{r}
#| label: fig-penguins_01
#| fig-cap: Bill length versus bill depth across penguin species.

ggplot(data = penguins, aes(x = bill_length_mm, y = bill_depth_mm)) +
  geom_point(aes(color = species, shape = species), size = 2)  +
  scale_color_manual(values = c("darkorange","darkorchid","cyan4")) +
  theme_minimal() +
  labs(
    title = "Penguin Bill Dimensions",
    x = "Bill Length (mm)",
    y = "Bill Depth (mm)"
  )

```
````

```
## Meet Quarto

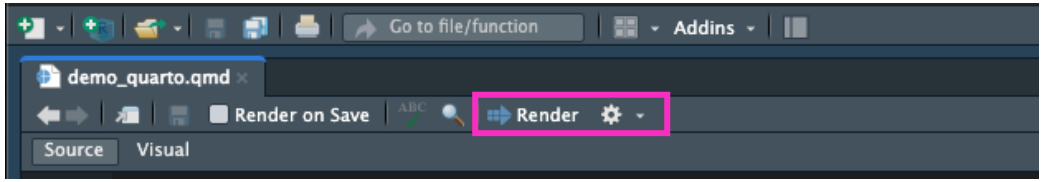
Quarto enables you to write together text and executable code into
a single file.

When you click the **Render** button a document will be generated
that includes the text, the code and the output.

## Meet the penguins

For this exercise, we will use the `penguins` dataset from the
[`palmerpenguins`](https://github.com/allisonhorst/palmerpenguins)
package. [@m.horst2022]
```

Rendering
What Happens When You Click "Render"?



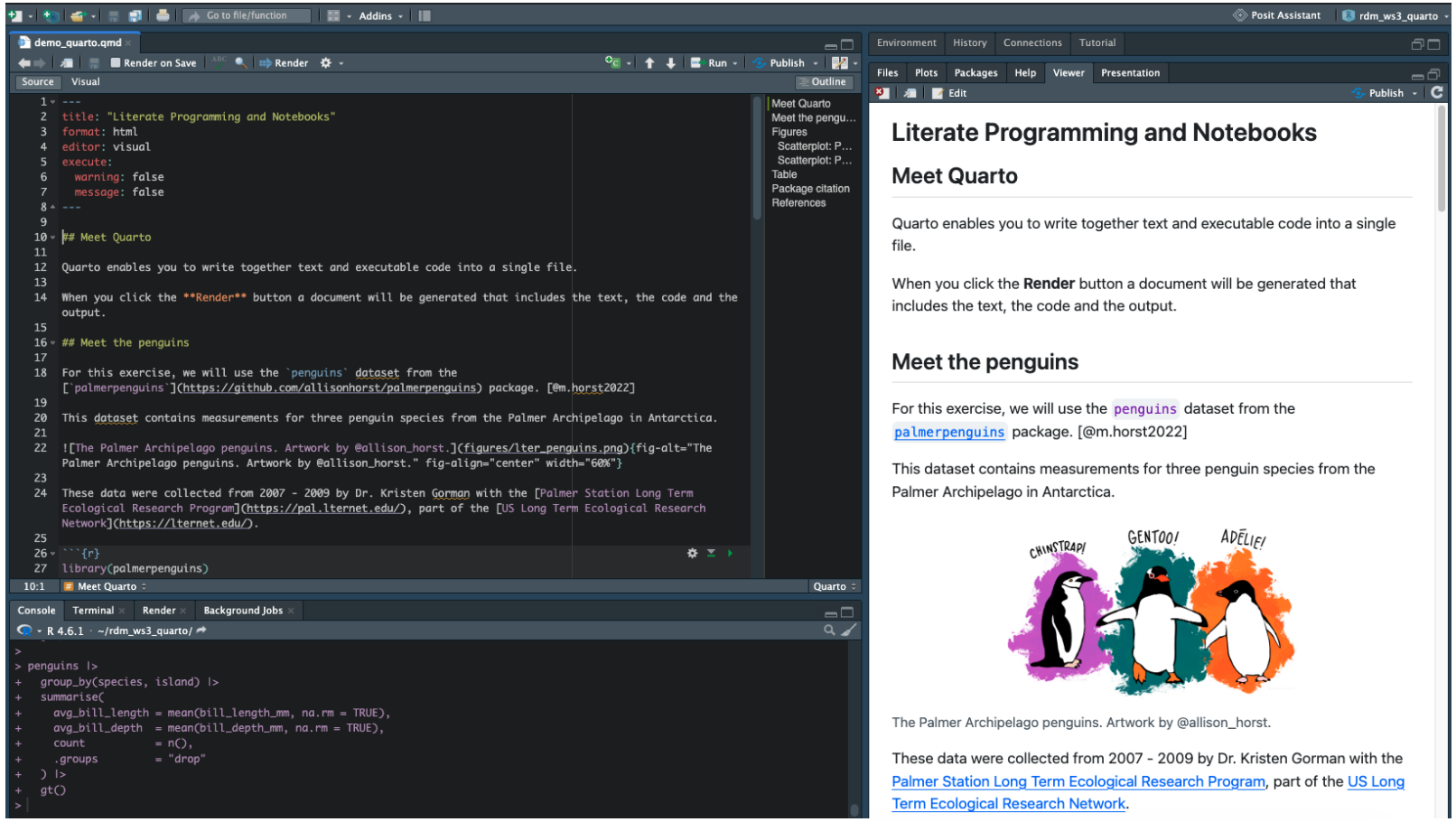
01 **Code Execution & Compilation**
Quarto runs all code top-to-bottom, captures plots and outputs, and knits them together.

02 **File Generation in Folder**
A standalone file (e.g., demo.html) is created right next to the .qmd file.

03 **Live Preview Display**
RStudio opens a live view in the Viewer Pane or an external browser window.

DEMO in RStudio

https://siscourses.ethz.ch/reproducible_computing/



DEMO in Terminal

Users > lmarques > rdm_ws3_quarto > demo_quarto.qmd > ...

```
1  ---
2  title: "Literate Programming and Notebooks"
3  format: html
4  editor: visual
5  execute:
6    warning: false
7    message: false
8    echo: true
9  ---
10
   ▷Run Cell | Run Next Cell
11 ```{r}
12 library(palmerpenguins)
13 library(dplyr)
14 library(ggplot2)
15 library(gt)
16 ```
17
18 ## Meet Quarto
19
20 Quarto enables you to write together text and executable code into a single file.
21
22 When you click the **Render** button a document will be generated that includes the
   text, the code and the output.
23
24 ## Meet the penguins
25
26 For this exercise, we will use the `penguins` dataset from the [`palmerpenguins`]
   (https://github.com/allisonhorst/palmerpenguins) package. [@m.horst2022]
27
```

```
[lmarques ~ $ quarto --version
1.10.19
[lmarques ~ $ cd rdm_ws3_quarto
[lmarques rdm_ws3_quarto $ quarto render demo_quarto.qmd


processing file: demo_quarto.qmd
1/13
2/13 [unnamed-chunk-1]
3/13
4/13 [unnamed-chunk-2]
5/13
6/13 [fig-penguins_01]
7/13
8/13 [plot-penguins_02]
9/13
10/13 [tbl-penguins-top10]
11/13
12/13 [unnamed-chunk-3]
13/13
output file: demo_quarto.knit.md

pandoc
  to: html
  output-file: demo_quarto.html
  standalone: true
  section-divs: true
  html-math-method: mathjax
  wrap: none
  default-image-extension: png

metadata
  document-css: false
  link-citations: true
  date-format: long
  lang: en
  title: Literate Programming and Notebooks
  editor: visual

Output created: demo_quarto.html

[lmarques rdm_ws3_quarto $ quarto preview demo_quarto.qmd
```

Quarto – General Recommendations

Quarto Best Practices & Advantages

- Render frequently to catch errors early.
- Choose the right editor mode for your workflow: Source or Visual
- Avoid large, single-file documents that turn into "spaghetti code":
  - Keep code chunks short, focused, and dedicated to one outcome (e.g., result, figure, table).
  - Break complex analyses into modular documents or Quarto projects (Books/Websites).

- Quarto uses plain-text .qmd files to improve reproducibility over notebook formats (like .ipynb):
  - Enforces clean top-to-bottom execution.
  - Makes version control and tracking changes in Git clean and simple.

Quarto – Resources

Resources for getting started with Quarto

- Tutorials on using quarto with different tools (RStudio, Jupyter, etc)
- Quarto Guide from the official website
- Quarto: The Practical Guide – online book
- Numerous tutorials available on the web / YouTube
  - Software Carpentry Quarto course
  - Swiss Institute of Bioinformatics (SIB) training: Make Your Research FAIRer with Quarto, GitHub and Zenodo (1-day online course with free course materials)



Literate Programming and Interactive Notebooks: Questions?





Reproducible Computing Platforms

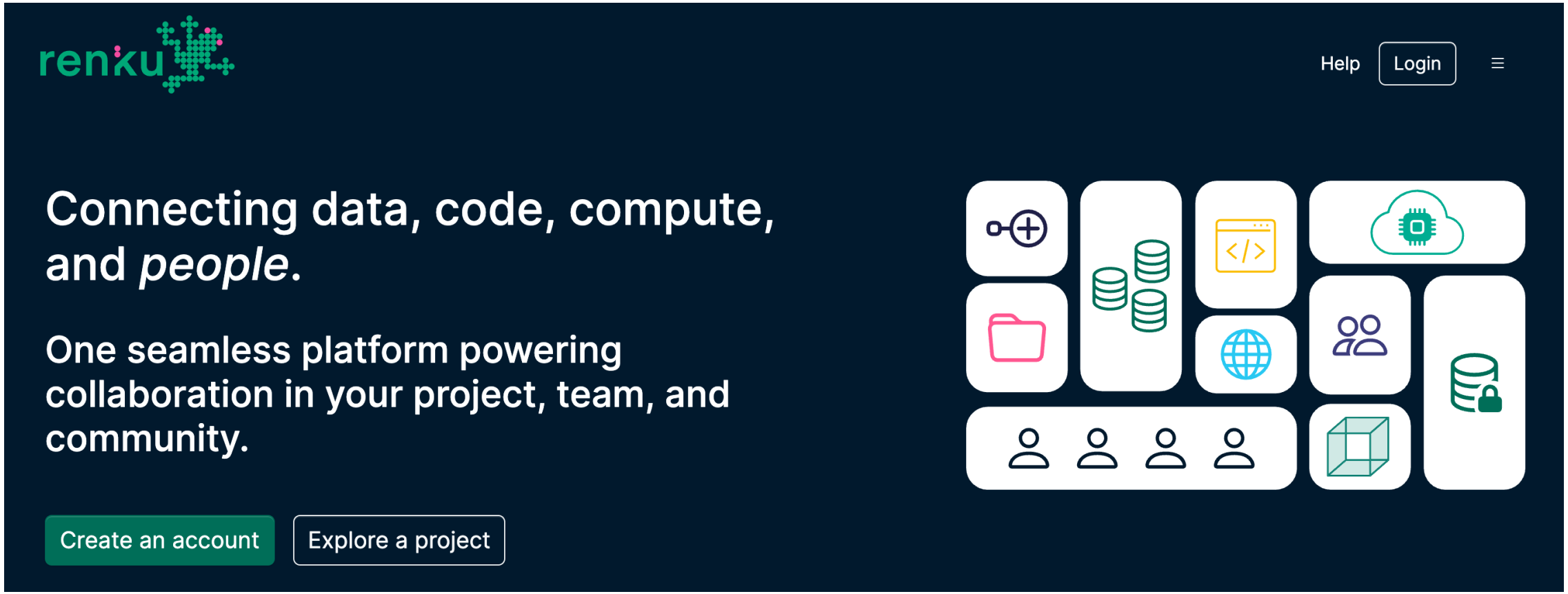


Reproducible Computing Platforms

- Integrated, **web-based** solutions for **reproducible** and **collaborative** data analysis and **computing**
- Usually built upon **proven open-source technologies** (Git, Conda, Docker etc.)
- Technical **complexity hidden** from user (or made easily accessible)
- Platforms provide **low entry barrier** access to fully reproducible computing
- **Commercial platforms**
  - Examples: Code Ocean, Google Colaboratory, …
  - Costs are incurred by usage of underlying cloud infrastructure (storage, compute, data transfer!)
  - Beware of data ownership, licensing issues and general T&Cs
- **Community platforms**
  - Examples: mybinder, Renkulab.io
  - Usually free of charge but resources are limited

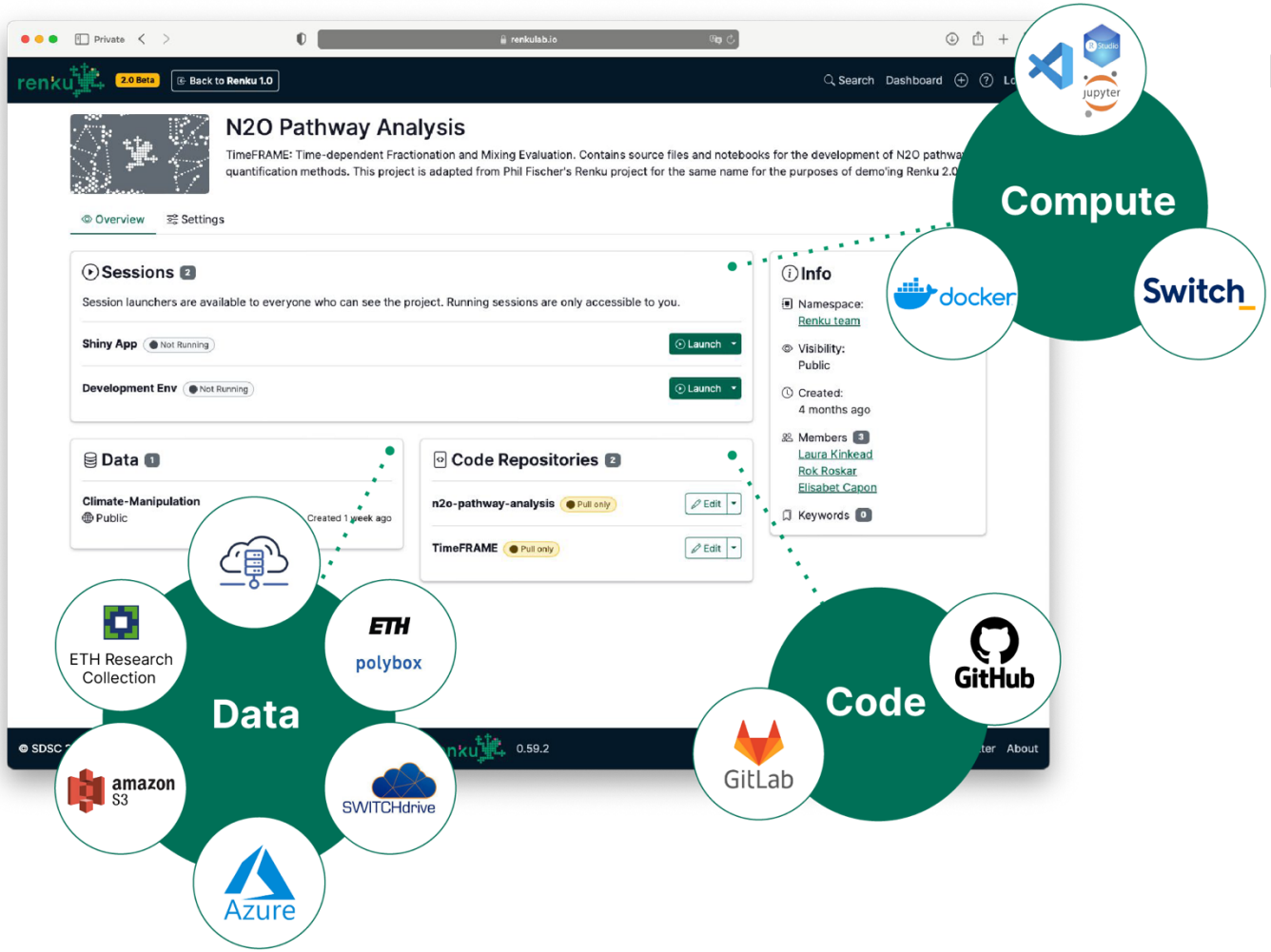
Reproducible Computing Platforms: *renkulab.io*

- Renkulab is a **platform for collaborative data science** from the Swiss Data Science Center (SDSC)



Reproducible Computing Platforms: *renkulab.io*

- Renkulab is a **platform for collaborative data science** from the Swiss Data Science Center (SDSC)

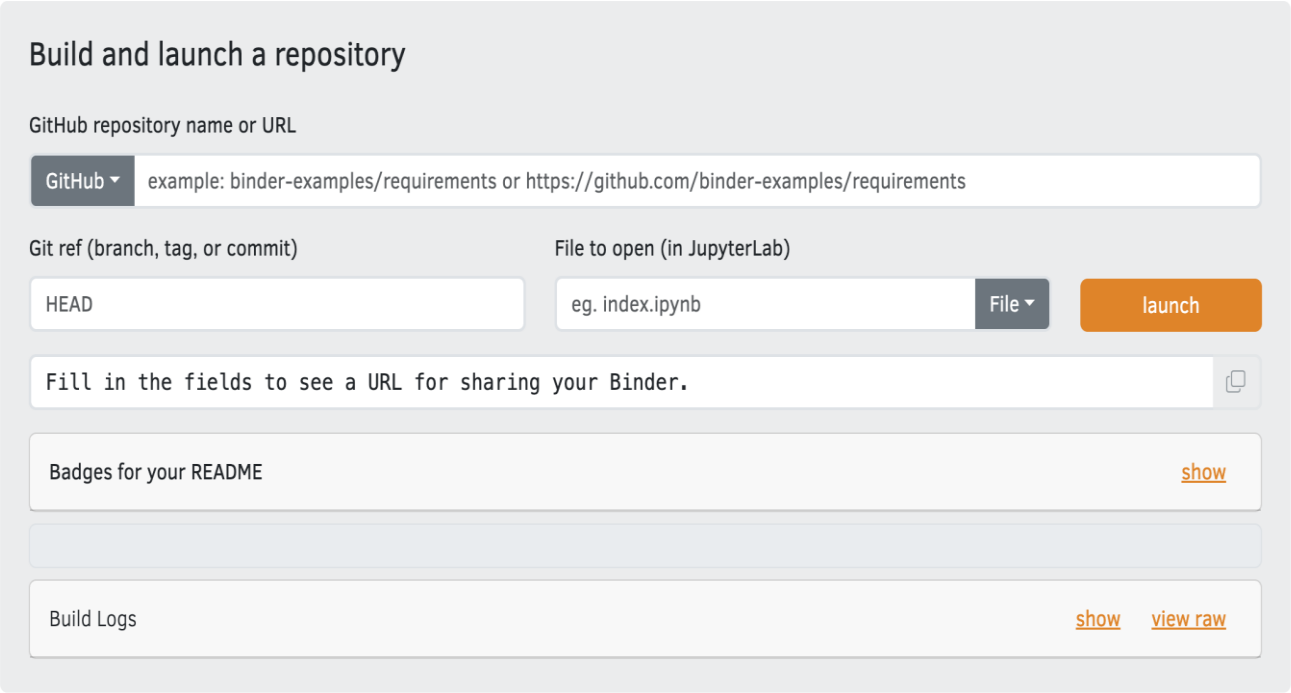


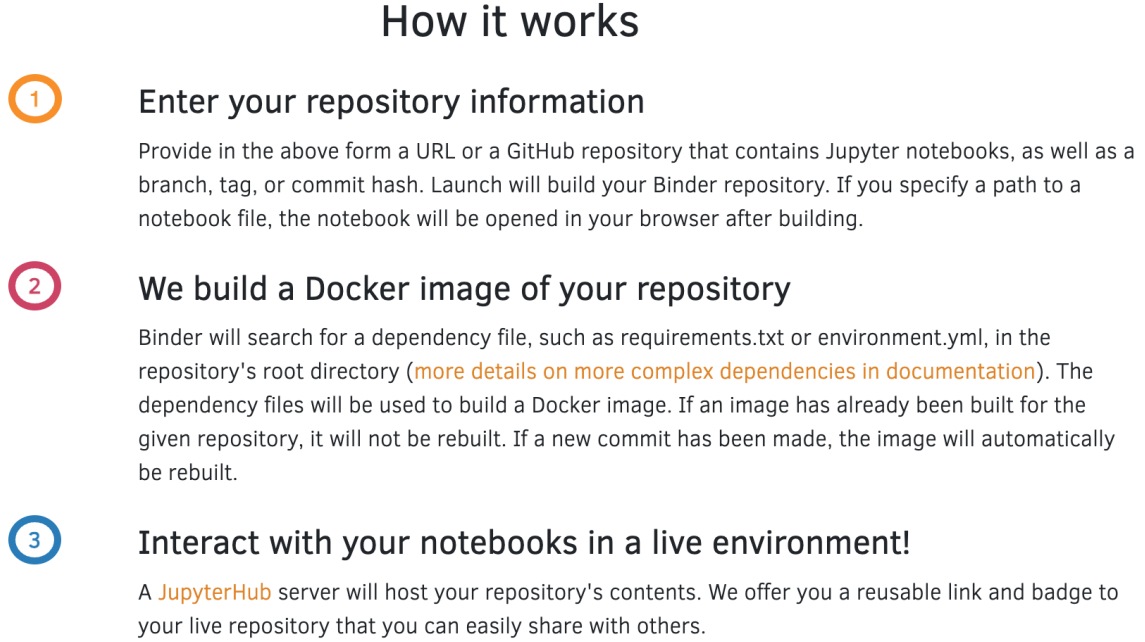
Learning more about Renkulab

- Login with your Switch edu-ID (or create a new account)
- Getting Started Tutorial
- More Renku Tutorials
- Renku How-To Guides

Reproducible Computing Platforms: *mybinder.org*

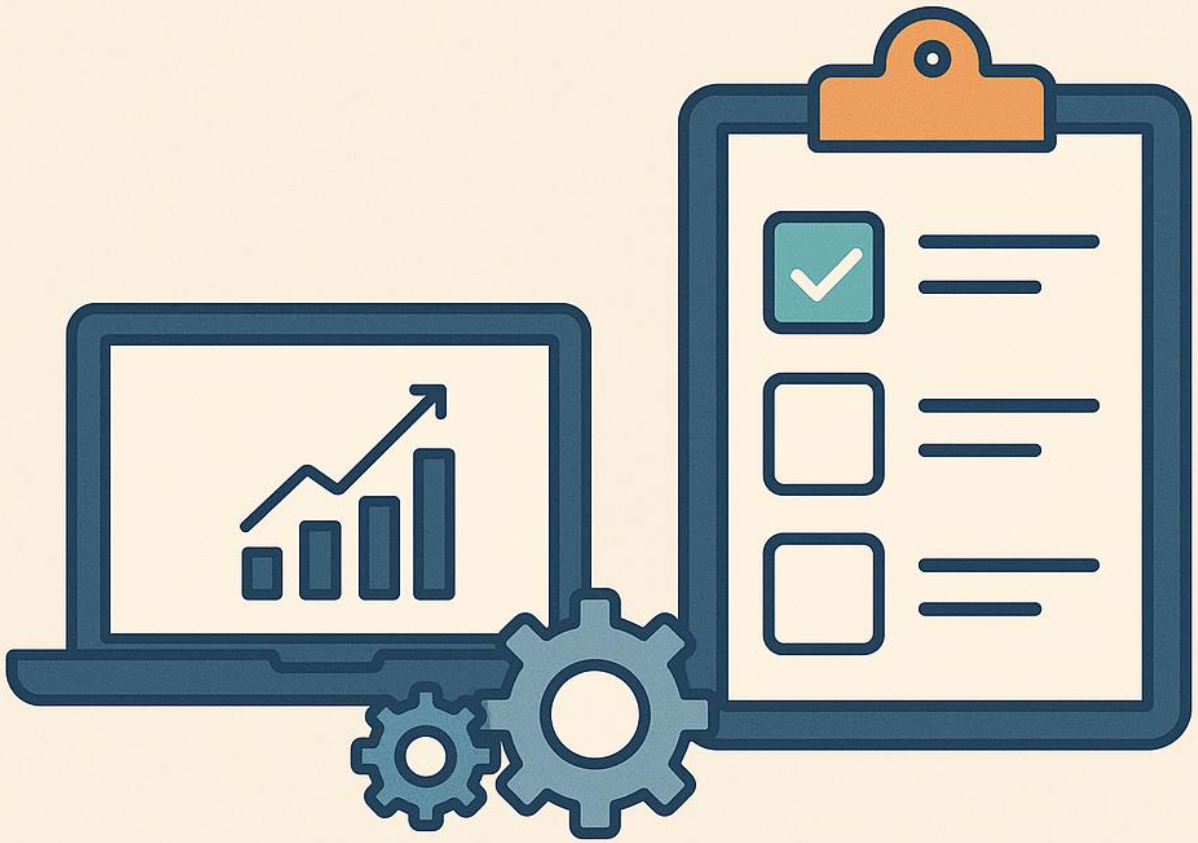
- Binder converts a Git repository into a collection of interactive notebooks
- Notebooks open in an executable environment → code becomes reproducible by anybody, anywhere



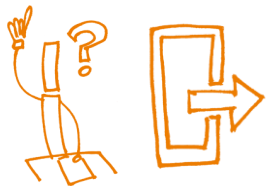


- Live example: https://github.com/hluetck/mybinder-demo-project

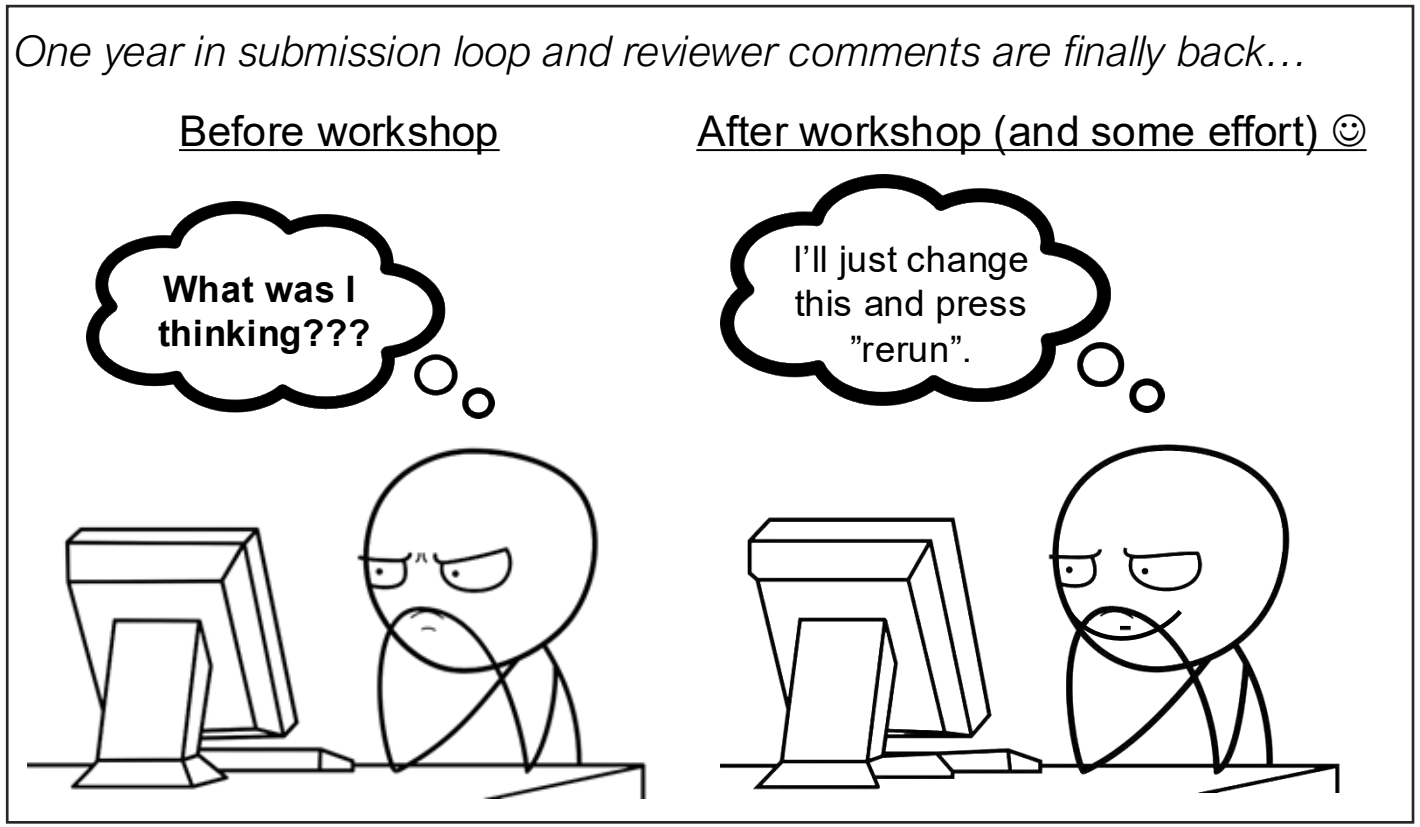
QUIZ TIME ...



Wrap-up & Discussion



What's in it for me?



At the start of the project

- Forced to think about scope and limitations
- Improved structure and organization

During the project

- Easier to rerun experiments and analysis
- Closer interaction between collaborators
- Much of the manuscript "writes itself"

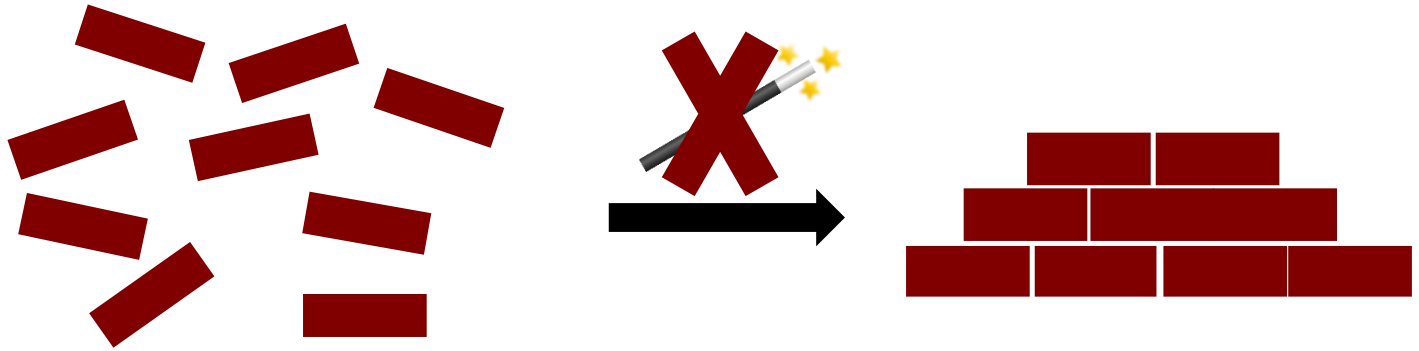
After the end of the project

- Faster resumption of research by others (or your future self), thereby increasing the impact of your work
- Increased visibility in the scientific community

Slide credit: L. Wigge, R. Ågren & J. Sundh (NBIS & SciLifeLab, Sweden)

What's in it for me?

- Aim for improvement, not perfection!
- RDM requires **WORK** & **TIME**, but the time spent on this is an **investment** for the future!

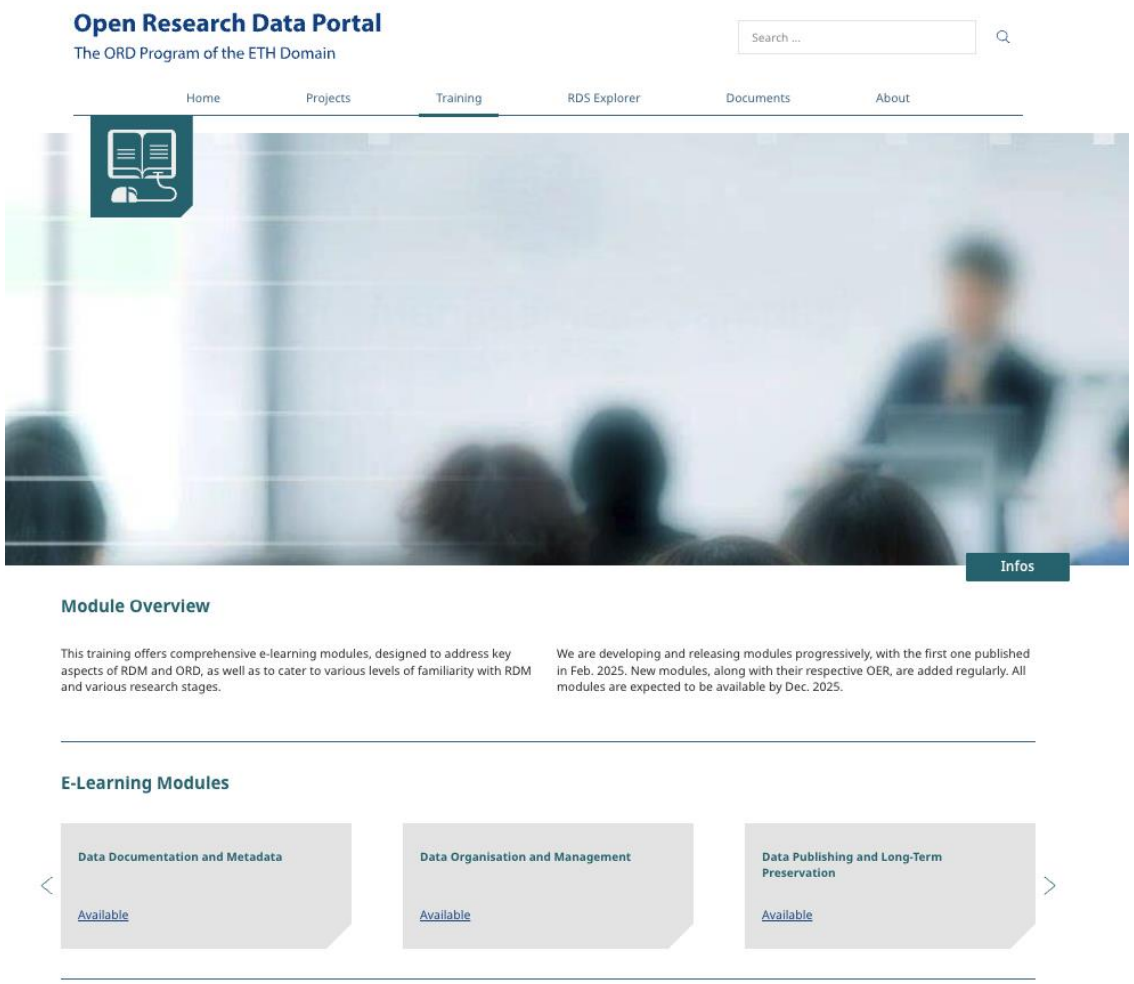


Contact us for consultations / trainings on data management, version control, reproducible computational workflows or data science support

sis.helpdesk@ethz.ch







https://open-research-data-portal.ch/training/

Contacts

Laura Marqués

laura.marques@id.ethz.ch

Henry Lütcke

henry.lutcke@id.ethz.ch

https://siscourses.ethz.ch/reproducible_computing/

sis.helpdesk@ethz.ch

https://sis.id.ethz.ch/

Feedback: https://www.umfrageonline.ch/c/scientificcomputing

Any final questions on what we have discussed this morning?

https://siscourses.ethz.ch/reproducible_computing/



Feedback: https://www.umfrageonline.ch/c/scientificcomputing