

RSE day 4

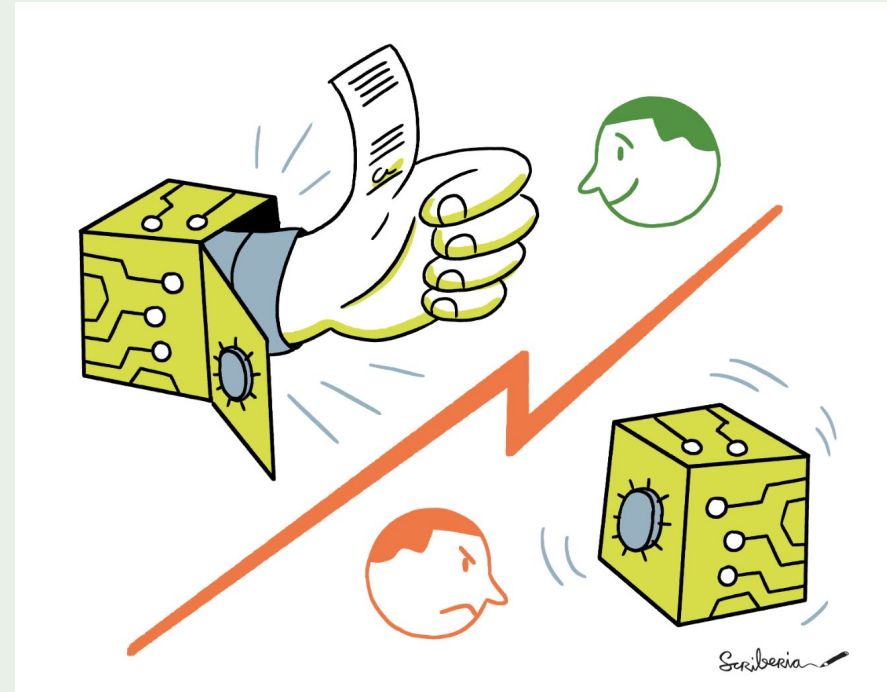


DIGITAL RESEARCH
ACADEMY

Outline

Day 4				
Topic	Time			Duration
Introduction	9:00	-	9:15	0:15
Software publication and licenses Activity 1: Create a license	9:15	-	9:45	0:30
Activity 2: Zenodo	9:45	-	10:00	0:15
Break	10:00	-	10:15	0:15
Brainstorming "What makes a good code project?" (preparation group work)	10:15	-	10:30	0:15
Group work: pitch prep on "What makes a good code project"-topics	10:30	-	11:05	0:35
Break	11:05	-	11:20	0:15
Pitches	11:20	-	11:45	0:25
Feedback, Wrap Up, Farewell	11:45	-	12:00	0:15

Licenses



The Turing Way Community, & Scriberia. (2024). *Illustrations from The Turing Way*: Shared under CC-BY 4.0 for reuse. Zenodo. [10.5281/ZENODO.3332807](https://doi.org/10.5281/ZENODO.3332807)

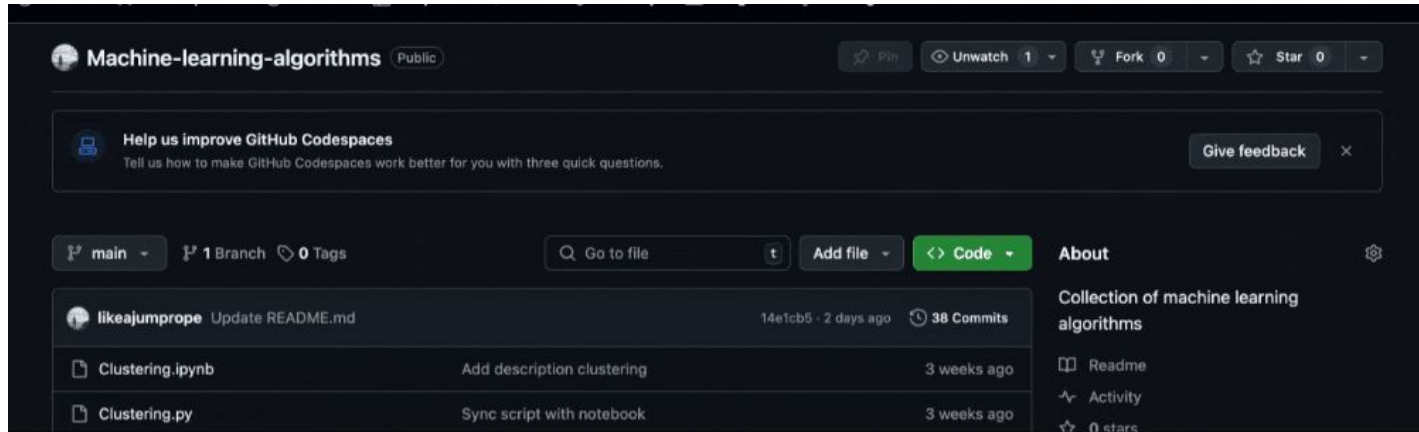
Licenses

Licenses

- You're under no obligation to create a license. However, without a license, the default copyright laws apply, meaning that you retain all rights to your source code and no one may reproduce, distribute, or create derivative works from your work.
- “Trap” lab or group repositories
 - Example: Lab PI creates repository without license, students contribute
 - Lack of license: Copyright of each contribution lies with the contributor
 - No one can make changes to the entire repository, INCLUDING LAB PI
 - This can be avoided using a license.
- Liability - most open source licenses protect the creator against liability claims

Licenses

- Licenses can be easily added to a github repository creating a file license.md
- Git will suggest license template text that you can copy and paste



Licenses

An overview about a large variety of licenses and comparisons are given here:

<https://choosealicense.com/appendix/>

<https://choosealicense.com/>

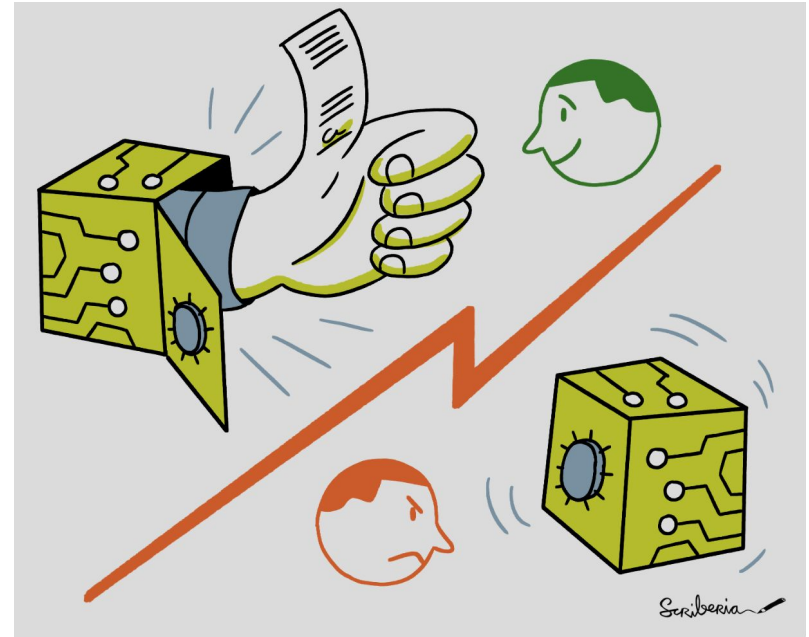
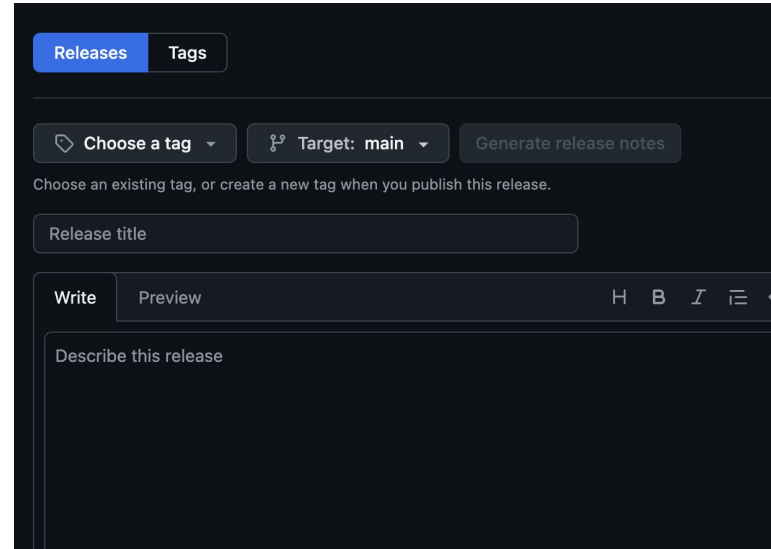


Fig. 35 Licensing. The Turing Way project illustration by Scriberia. Used under a CC-BY 4.0 licence. DOI: [10.5281/zenodo.3332807](https://doi.org/10.5281/zenodo.3332807).

	Free			Non-free		
	Public domain & equivalents	Permissive license	Copyleft (protective license)	Noncommercial license	Proprietary license	Trade secret
Description	Grants all rights	Grants use rights, including right to relicense (allows proprietization, license compatibility)	Grants use rights, forbids proprietization	Grants rights for noncommercial use only. May be combined with share-alike.	Traditional use of copyright ; certain rights may or may not be granted	No information made public
For software	PD, Unlicense , CC0	BSD , MIT , Apache	GPL , AGPL	JRL , AFPL	Proprietary software , no public license	Private, internal software
For other creative works	PD, CC0	CC BY	CC BY-SA , FAL	CC BY-NC	Copyright , no public license, CC BY-ND	Unpublished

Versioning software

- Versioning software on GitHub typically involves using **Git tags** to label specific commits with version numbers.
- Tags provide a snapshot of your codebase at a particular point in time, often corresponding to software releases.
- GitHub integrates these tags with release management tools, making it easier to distribute and document new versions of your software.



Versioning software

Semantic versioning:

MAJOR.MINOR.PATCH

- **MAJOR:** Increments for breaking changes.
- **MINOR:** Increments for backward-compatible feature additions.
- **PATCH:** Increments for bug fixes.

Releasing a version

1. Update the version number in your code (e.g., in `__version__` or [setup.py](#))
2. Commit with a meaningful message:

"Release v1.0.0"

3. Tag the commit:

```
git tag v1.0.0  
git push origin v1.0.0
```

4. Keep a CHANGELOG

```
## v1.1.0 - 2025-05-06  
- Added support for multiple inputs  
- Improved error messages  
- Fixed bug in file loader
```

What happens when we create a release?

Creating a specific pointer to a certain commit

- A **Git tag** is created — this is a named pointer to a specific commit.
- Tags are immutable: once created, they permanently point to that commit.

Creating an Archive Snapshot

- Zip or tarball the exact state of your code at that tag.
- Make it downloadable (for citations, installations, replication).

Attach Metadata (Changelog, Notes)

You can (and should) add:

- A **title** (v1.0.0)
- A **release description or changelog**
- Optional binaries, datasets, or wheels

Activity 1 (15'):

**Create a license file
and a version release**

- Do Activity 1.

**Sharing <insert
research output here>
openly**

Zenodo



- Free and open platform for sharing, preserving and citing research outputs
- Developed by European Organisation for Nuclear Research CERN, supported by the [OpenAIRE](#) initiative
- Used to store and share:
 - Software
 - Publications
 - Datasets
 - Other types of research software

Zenodo



- **Open Access Repository:** enables researchers to share their work openly, promoting transparency and collaboration.
- Every upload is assigned a **Digital Object Identifier (DOI)**, making it easy to cite and permanently reference the work.
- **Support for All Research Outputs:** Accepts a variety of file types, including datasets, code, publications, posters, presentations, and multimedia.
- **Versioning:** Allows for versioning of submissions, ensuring that updates to datasets or software can be tracked while maintaining the ability to reference specific versions.
- **Integration with GitHub:** Automates the archiving of GitHub repositories. Researchers can push their code or project directly from GitHub to Zenodo for publication and DOI generation.
- **Open Licenses:** Supports the application of open licenses (e.g., CC-BY, GPL) to promote reuse and clarity about usage rights.
- **Long-term Preservation:** Ensures long-term preservation of research outputs by leveraging CERN's robust storage infrastructure.
- **Metadata Standards:** Complies with metadata standards to improve discoverability and interoperability.
- Compatible with EU Standards

How to use Zenodo for sharing



1. Create an Account

- Go to [Zenodo](https://zenodo.org) and sign up using your ORCID or email address

2. Upload Research Outputs

- Click **Upload** and fill out the metadata fields (title, authors, description, etc.).
- Attach the files you want to share (datasets, code, PDFs, etc.).
- Choose a license (e.g., Creative Commons, GPL).
- Publish the record to generate a DOI.

3. Share the DOI

- Use the DOI to cite your research outputs in publications, presentations, or online profiles.

Linking your Zenodo to your Github



1. Create an Account

- Go to [Zenodo](#) and sign up using your ORCID or email address

2. Integrate GitHub with Zenodo

- Link your GitHub account to Zenodo .
- Switch the flip: Enable archiving for specific repositories.
- When you create a release on GitHub, Zenodo automatically archives the repository and assigns a DOI.

3. Share the DOI

- Use the DOI to cite your research outputs in publications, presentations, or online profiles.

A screenshot of the Zenodo user interface. On the left is a 'Settings' sidebar with options: Profile, Change password, Security, Notifications, Linked accounts (highlighted in blue), Applications, and GitHub. The main content area is divided into two sections. The top section, 'Linked accounts', shows 'GitHub' with a green checkmark and 'ORCID' with a green checkmark. The bottom section, '1 Flip the switch', contains a toggle switch labeled 'ON' and '(example)', with text above it: 'Select the repository you want to preserve, and toggle the switch below to turn on automatic preservation of your software.' Below this is an 'Enabled Repositories' section showing a repository 'likeajumprope/Git_course' with a DOI '10.5281/ze'.

Other (often institutional) file sharing repositories

Figshare

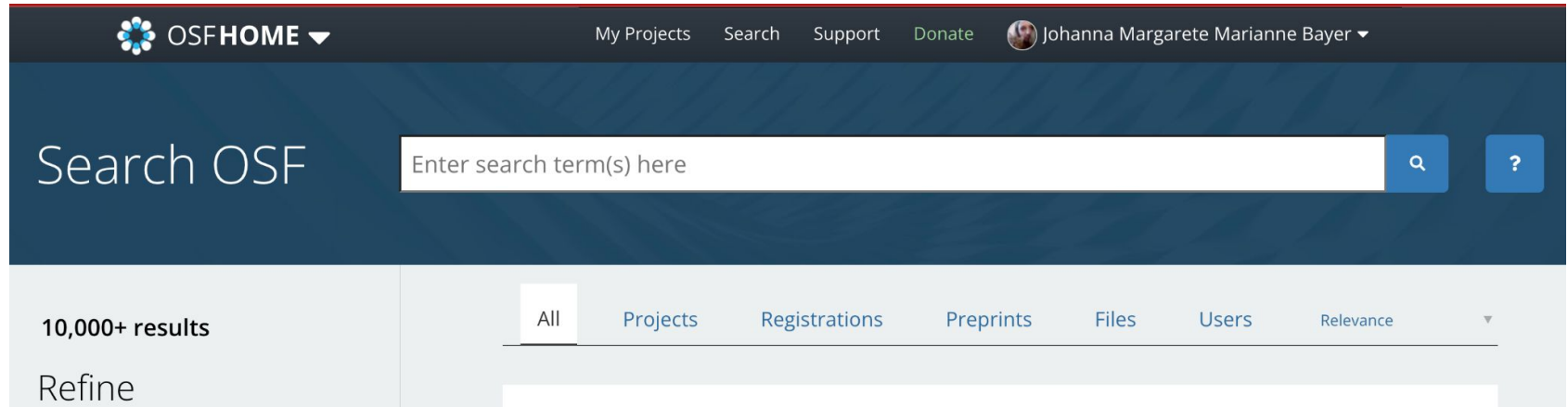
- [Figshare](#) is a cloud-based platform designed for storing, sharing, and managing **research outputs of all kinds**, including datasets, figures, images, code, and publications.
- However, **free accounts have limited storage**.

Dryad

- [Dryad](#) is an open-access repository specifically designed for **research data**. It focuses on datasets that accompany peer-reviewed publications, ensuring that they are preserved and accessible.
- However **limited to data sets** and it relies on a subscription model.

Open Science Framework (OSF)

- The **Open Science Framework (OSF)** is a free, open-source platform designed to support collaboration, transparency, and reproducibility throughout the research lifecycle. Researchers, educators, and professionals use OSF to organize and share their projects, data, and research outputs.
- It is maintained by the **Center for Open Science (COS)**, a nonprofit organization dedicated to increasing the openness and accessibility of scientific research.



The screenshot shows the OSF Home page. At the top, there is a dark blue header with the OSF logo and the text "OSFHOME" followed by a dropdown arrow. To the right of the header are links for "My Projects", "Search", "Support", and "Donate", along with a user profile for "Johanna Margarete Marianne Bayer" with a dropdown arrow. Below the header is a large dark blue banner with the text "Search OSF" on the left. To the right of this text is a search input field with the placeholder text "Enter search term(s) here". To the right of the input field are two blue buttons: one with a magnifying glass icon and one with a question mark icon. Below the banner is a light gray section. On the left, it says "10,000+ results" and "Refine". To the right of this are several tabs: "All", "Projects", "Registrations", "Preprints", "Files", "Users", and "Relevance" with a dropdown arrow. The "All" tab is currently selected.

OSF

- **Project Management:**
 - Create and manage research projects with a centralized workspace
 - Share and organize files, data, code, and other materials.
- **Collaboration:**
 - Add collaborators with flexible permissions (read-only, read/write, admin).
 - Work with team members from different institutions.
- **File Storage and Integration:**
 - Store files (up to 5 GB per file for free users), from storage services like Google Drive, Dropbox, and GitHub.
- **Pre-registration:**
 - Pre-register research plans and hypotheses to promote transparency and reduce publication bias.
- **Version Control:**
 - Track changes to files and materials, maintaining a clear history of updates.
- **Preprints:**
 - Publish preprints (early research drafts) and link them to your OSF project.
 - Access preprint services like **PsyArXiv**, **SocArXiv**, **BioRxiv**, and others.
- **DOI Assignment:**
 - Generate DOIs (Digital Object Identifiers) for projects and files to make them citable.

	OSF	Zenodo	Github
Purpose	Project management, collaboration, preprints	Data and software sharing	Code and small script version control
Open Access	yes	yes	Depends on repository
Pre-registration	yes	no	no
File storage	5 GB limit per file	50 GB limit per upload	File size and repos size limits
DOI supports	yes	yes	no
Integration with other tools	Zotero, Mendeley, Github, Dropbox	Github, OpenAire	Many others

Activity 2:

Zenodo & DOIs

- Link your Github to Zenodo
- Link your Github release to your zenodo profile and create a DOI
- Add the DOI as a badge on your Github repo

Break 15'

Data version control

Provenance on which data in which version was underlying which computation is crucial for reproducibility. The Turing Way project illustration by Scriberia. Used under a CC-BY 4.0 licence. DOI: [10.5281/zenodo.3332807](https://doi.org/10.5281/zenodo.3332807).

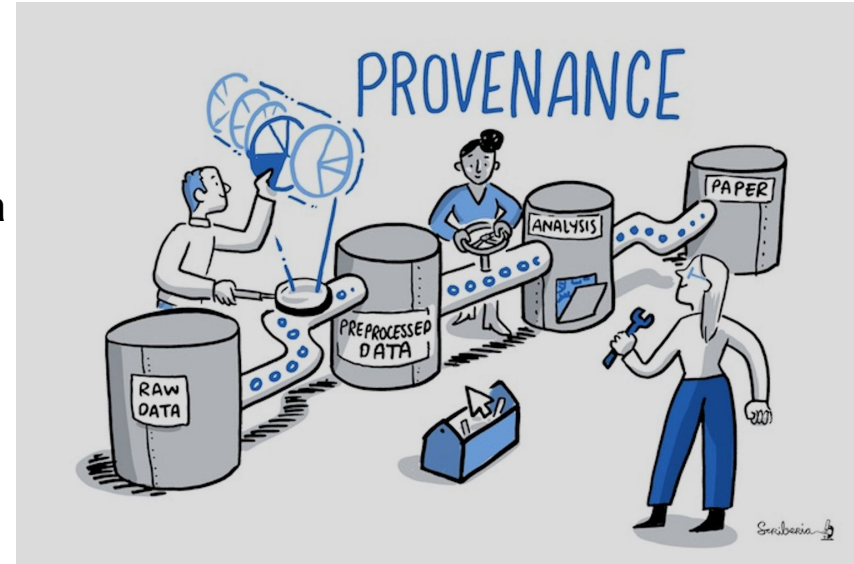
Data version control

Data is a living creature

- Throughout the research process, data can be:
 - datasets can be extended with new data
 - adapted to new naming schemes
 - reorganised into different file hierarchie
 - updated with new data points
 - modified to fix any errors.
- > threatens reproducibility

Provenance: describing the origin of an object

Data version
control



Provenance on which data in which version was underlying which computation is crucial for reproducibility. The Turing Way project illustration by Scriberia. Used under a CC-BY 4.0 licence. DOI: 10.5281/zenodo.3332807.

File organisation: Version control

Product	Short description	
Git/Github	Scripts and files (up to 100 MiB per file/ 50MB per repo)	https://github.com/
Git Large file Storage (LFS)	Creates pointers to large files that are stored elsewhere (i.e git server [paid], Dropbox)	https://git-lfs.com/
Datalad	Data sets and pipelines tracking, uses git annex & direct integration with git	https://www.datalad.org/
Data Version Control (DVC)	Data sets and (ML) pipeline tracking, ignores git	https://dvc.org/
ML flow (et al.)	(ML) Experiment tracking (python)	https://mlflow.org/

Git LFS

Git Large File Storage (Git LFS) is an open-source Git extension that improves how large files are handled in Git repositories. Instead of storing the content of large files directly in the repository, Git LFS replaces them with **lightweight pointers** and stores the actual file content in a separate storage location.

Git LFS comes with a **command-line extension to Git** and allows you to treat files of any size alike, using standard Git commands.

A major shortcoming, however, is that Git LFS is a centralised solution. Large files are not distributed but stored on a remote server. This usually requires setting up your server **or paying for a service** - which can make it very inaccessible.

Git LFS

How Git LFS Works

1. Pointers in the Repository:

- Git LFS replaces large files in the repository with small pointer files. These pointer files are tracked in Git instead of the actual file.

2. External Storage:

- The actual large files are stored in a separate Git LFS server or Git hosting service that supports Git LFS (e.g., GitHub, GitLab, Bitbucket).

3. Efficient Fetching:

- When you clone or pull a repository, Git LFS downloads the large files only when needed, reducing the size of the repository.

Data Version Control (DVC)

DVC (<https://dvc.org/>)

- A tool for version-controlling large datasets, machine learning models, and pipelines, specifically geared toward **machine learning and data science workflows**.
- DVC guarantees reproducibility by consistently maintaining a **combination of input data, configuration, and the code that was initially used to run an experiment**.
- Seamless integration with git



Data Version Control (DVC)

- Similar to LSF, uses pointers
- Tracks changes in data sets **using .dvc files.**
- Pipeline management:

```
### DVC
$ cat dvc.yaml
stages:
  prepare:
    cmd: python src/prepare.py
    deps:
      - data/raw
      - src/prepare.py
    outs:
      - data/prepared/test.csv
      - data/prepared/train.csv
```

- Experiment tracking

```
### DVC
$ dvc metrics show
Path          accuracy
metrics/accuracy.json 0.67934
```

Data version
control



Data Version Control (DVC)



Strengths:

1. Handles Large Files Efficiently

- Tracks large files (e.g., datasets, models) without storing them directly in Git, using .dvc pointer files to manage metadata.

2. Pipeline and Workflow Management

- Provides built-in tools (dvc.yaml) to define and track pipelines, ensuring reproducibility of data science and machine learning workflows.

3. Git Integration

- Integrates tightly with Git, enabling version control for both data and code in a single repository.

Weaknesses:

- **Learning curve:** Using DVC effectively requires understanding both Git workflows and DVC-specific commands for managing pipelines and datasets.

Git submodules

- **Git submodules** allows to include one Git repository (the **submodule**) as a subdirectory within another Git repository (the **parent repository**).
- useful when you want to incorporate **another repository's code** into your project while keeping it as a separate entity.
- Issue: Git submodules **do not update** with the parent library

For example:

- You might include a shared library or module from another repository in your project without copying the files directly.
- Submodules allow you to **track a specific version** of the included repository while maintaining a connection to its original repository.

Git submodules

Strength

1. **Version Control:** Submodules allow you to pin dependencies to specific commits.
2. **Modularity:** Keeps projects modular by separating the submodule's codebase.
3. **Avoid Code Duplication:** Submodules link to the original repository without copying its files.
4. **Works with Git Workflow:** Submodules integrate seamlessly with Git, leveraging Git's tools and workflows.

Weaknesses:

1. **Complexity:** Submodules can complicate workflows, especially for new Git users.
2. **Manual Updates:** Updating a submodule requires manual intervention (e.g., `git submodule update`).
3. **Nested Submodules:** Managing nested submodules can be particularly challenging.
4. **Merge Conflicts:** Merge conflicts involving submodules can be harder to resolve.
5. **Dependency Awareness:** All team members need to understand how submodules work to avoid issues during collaboration



Git-annex

- [Git Annex](#) is an open-source tool that extends Git's functionality to manage large files or collections of files across multiple locations. Instead of storing large files directly in the Git repository, Git Annex stores **symlinks (symbolic links)** in the repository while managing the actual files in a separate annex.
- It is especially useful for managing large datasets, media files, or files that need to be distributed across multiple devices or storage systems.

Strengths:

1. **Distributed and Decentralized:** Ideal for teams with varying levels of connectivity or where central hosting is impractical.
2. **Customizable Storage:** Can use multiple storage backends simultaneously.
3. **Data Redundancy:** Ensures data integrity by storing files across multiple locations.
4. **Offline First:** Files and changes can be queued for syncing when online.
5. **Advanced Rules:** Configure rules for where files should be stored or downloaded based on system requirements.



Git-annex

Weaknesses:

1. **Complexity:**

- Configuration and usage can be more complicated than Git LFS.
- Requires knowledge of symlinks and Git Annex-specific commands.

2. **Not Seamlessly Integrated with Git Hosting Services:**

- Unlike Git LFS, Git Annex does not natively integrate with platforms like GitHub or GitLab.

3. **No Built-in CI/CD Integration:**

- While it works well for distributed storage, it lacks native integration with DevOps pipelines.

Datalad



DataLad

DataLad, builds upon git and git-annex. Like [git-annex](#), it allows you to version control data and share it via third-party providers but simplifies and extends this functionality.

- In addition to sharing and version controlling large files; it allows recording, sharing, and using software environments, recording and re-executing commands or data analyses, and operating seamlessly across a hierarchy of repositories.

```
$ datalad containers-run -n software \  
-m "Evaluate SGD classifier on test data" \  
--input 'data/raw/val/n03445777' \  
--input 'data/raw/val/n03888257' \  
--output 'accuracy.json' \  
"python3 code/evaluate.py"  
[INFO] Making sure inputs are available (this may take some time)  
[INFO] == Command start (output follows) =====  
[INFO] == Command exit (modification check follows) =====  
{'accuracy': 0.7363751584283904}  
run(ok): /home/me/usecases/ml-project (dataset) [singularity exec  
save(ok): . (dataset)  
action summary:  
  add (ok: 2)  
  get (notneeded: 4)  
  run (ok: 1)  
  save (notneeded: 1, ok: 1)
```

<https://www.datalad.org/>

Data version
control

Datalad - strength



Handles Large Datasets Efficiently

- **Uses Git-Annex:** Tracks large files without bloating the repository by storing content in external locations and maintaining lightweight metadata in Git.
- **Remote Backends:** Supports multiple storage backends (local storage, SSH servers, cloud storage like AWS S3, and more).
- **Partial Data Access:** Allows downloading specific parts of a dataset without needing the entire dataset (on-demand access).

Reproducibility

- **Version Control for Data:** Enables tracking of changes to datasets, ensuring reproducibility across different research stages.
- **Snapshots:** Provides a history of dataset versions, making it easy to revert to a previous state.

Integration with Git

- Fully integrates with Git workflows, allowing you to track and version both datasets and code.
- Works well in collaborative environments using Git-based tools (e.g., GitHub, GitLab).

FAIR Principles Compliance

Metadata Management

- Enables adding rich metadata to datasets, improving their discoverability and usability.

Dataset Discovery and Sharing

- Provides tools for discovering and sharing datasets, particularly useful in scientific research.
- Supports distributed sharing via Git-Annex, enabling dataset collaboration.

Automation-Friendly

- Designed for use in both interactive and automated environments.
- CLI and Python API support enable integration into pipelines and workflows.

Datalad -weaknesses



1. Steep Learning Curve

- The combination of Git, Git-Annex, and Datalad concepts can be challenging for new users, especially those unfamiliar with Git workflows.
- Requires understanding both Git and Git-Annex to fully utilize Datalad's capabilities.

2. Dependency on Git-Annex

- Relies heavily on Git-Annex, which itself has a learning curve and specific limitations (e.g., requires additional setup for certain remotes).
- Git-Annex symlinks may cause confusion for users not familiar with its architecture.

3. Limited Mainstream Adoption

- While powerful, Datalad is not as widely adopted as other tools (e.g., DVC or Git LFS) in some domains, which might limit its community support and available resources.

4. Performance on Very Large Repositories

- Handling metadata for extremely large repositories with many files can become slow, as Git and Git-Annex are not optimized for such scenarios.

5. Complex Setup for Non-Technical Users

- Installing and configuring Datalad, especially in multi-user or multi-institution setups, can be challenging.
- Requires additional configuration for integrating with cloud storage or other remotes.

ML flow (python)

MLflow is an open-source platform for managing the **end-to-end machine learning (ML) lifecycle**. It provides tools to streamline **experimentation, reproducibility, and deployment of machine learning models**. MLflow integrates well with existing ML libraries and frameworks, making it a versatile tool for practitioners.

The screenshot displays the MLflow 2.18.0 web interface. The top navigation bar includes the MLflow logo, version 2.18.0, and tabs for 'Experiments' and 'Models'. The 'Experiments' sidebar on the left contains a search bar and a list of experiments, with 'Default' selected. The main content area shows the 'Default' experiment details, including a 'Share' button and tabs for 'Runs', 'Evaluation', 'Experimental', and 'Traces'. The 'Runs' tab is active, showing a search bar with the query 'metrics.rmse < 1 and params.model = "tree"', filters for 'Time created', 'State: Active', and 'Datasets', and a '+ New run' button. Below these are sorting and column options. A table lists three runs:

<input type="checkbox"/>	Run Name	Created	Dataset	Duration	Source	Models
<input type="checkbox"/>	luminous-flea-485	29 seconds ago	-	1.3s	mlflow_...	sklearn
<input type="checkbox"/>	grandiose-loon-828	2 minutes ago	-	1.6s	mlflow_...	sklearn
<input type="checkbox"/>	gentle-stork-276	4 minutes ago	-	1.8s	mlflow_...	-

Data version
control

Comparison

- **DVC:** Best for machine learning workflows where data versioning, pipeline management, and reproducibility are needed.
- **Git LFS:** Simplest for versioning large files directly with Git.
- **Git-Annex:** Flexible for managing large datasets with multiple storage options (i.e Dropbox).
- **Datalad:** Research-focused, built on Git-Annex for scientific data management. Pipeline management, retrieving and sharing data
- **MLflow:** Focused on tracking machine learning experiments and managing models.
- **Git Submodules:** Ideal for managing reusable Git-based code dependencies.

IMPLEMENTATION/ GROUP WORK

The background features a light green rectangular block on the left and a pink rectangular block on the right, both positioned below the main text.

Activity 3

Option 1: Create a checklist

*What should one think about
when setting up a code project?*

**Option 2:
Implement
some of the
here learned on
your own
project**

Github, tests, automation etc.

Share out

Wrap up