RESEARCH DATA MANAGEMENT

WITH DATALAD

Adina Wagner
mas.to/@adswa

Psychoinformatics lab,
Institute of Neuroscience and Medicine (INM-7)
Research Center Jülich

Slides: files.inm7.de/adina/talks/html/sfb-1280.html
Sources: https://github.com/datalad-handbook/datalad-course

WELCOME & LOGISTICS!

- A approximate schedule for today:
  - 1.00 pm: Introduction & Logistics
  - 1.30 pm: Overview of DataLad + break ☕
  - 2.00 pm: What's version control, and why should I care?
  - 2:45 pm: Reproducibility features + break
  - 3.30 pm: Data publication to the OSF + break ☕
  - 4.30 pm: Outlook and/or Your Questions and Usecases
- Collaborative notes & anonymous questions: etherpad.wikimedia.org/p/Datalad@sfb1280.
- Slides are CC-BY and will be shared after the workshop. Additional workshop contents: psychoinformatics-de.github.io/rdm-course
- Some guidelines for the virtual workshop venue...
  - Please mute yourself when you don't speak
  - Ask questions anytime, but make use of the "Raise hand" feature
  - Drop out and re-join as you please

QUESTIONS/INTERACTION THROUGHOUT THE WORKSHOP

- There are no stupid questions :)
- Lively discussions are wonderful - unless its interrupting others, please feel encouraged to unmute/turn on your video to interact.
- There is room discuss specific or advanced use cases at the end. Please make a note about them in the Etherpad.

QUESTIONS/INTERACTION AFTER THE WORKSHOP

If you have a question after the workshop, you can reach out for help:

Reach out to to the DataLad team via

- Matrix (free, decentralized communication app, no app needed). We run a weekly Zoom office hour (Tuesday, 4pm Berlin time) from this room as well.
- the development repository on GitHub

Reach out to the user community with

- A question on neurostars.org with a `datalad` tag

Find more user tutorials or workshop recordings

- On DataLad's YouTube channel
- In the DataLad Handbook
- In the DataLad RDM course
- In the Official API documentation

RESOURCES AND FURTHER READING

Comprehensive user documentation in the DataLad Handbook (handbook.datalad.org)

- High-level function/command overviews, Installation, Configuration, Cheatsheet

---

- Narrative-based code-along course
- Independent on background/skill level, suitable for data management novices

---

- Step-by-step solutions to common data management problems, like how to make a reproducible paper

Overview of most tutorials, talks, videos, ... at github.com/datalad/tutorials

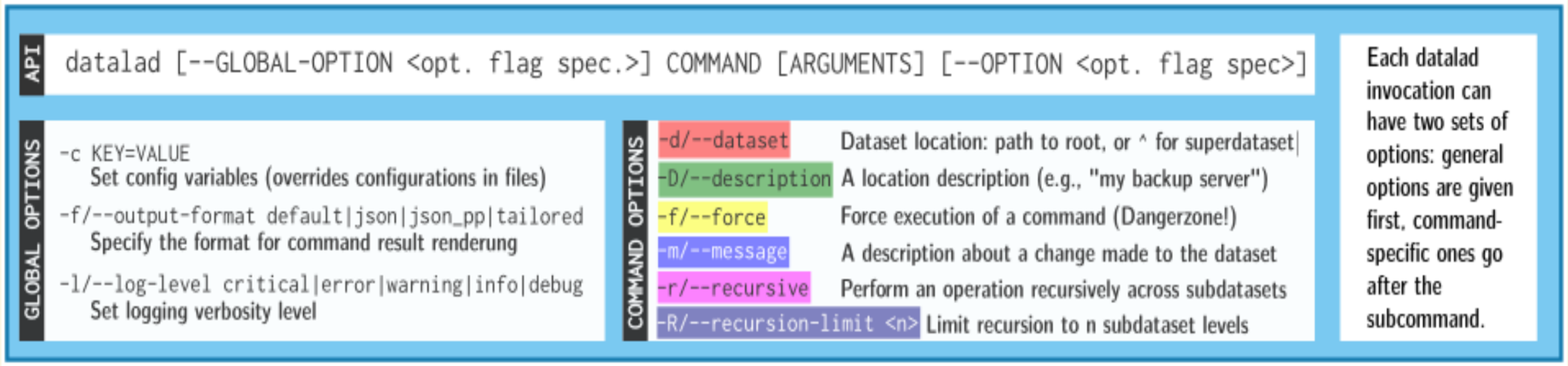
PRACTICAL ASPECTS

- We'll work in the browser on a cloud server with JupyterHub
- Cloud-computing environment:
  - datalad-hub.inm7.de
- We have pre-installed DataLad and other requirements
- We will work via the terminal
- Your username is all lower-case and follows this pattern: Firstname + Lastname initial (Adina Wagner -> adinaw)
- Pick any password with at least 8 characters at first log-in (and remember it)

Please try to log in now

PREREQUISITES: USING DATALAD

- Every DataLad command consists of a main command followed by a sub-command. The main and the sub-command can have options.



- Example (main command, subcommand, several subcommand options):

```
$ datalad save -m "Saving changes" --recursive
```

- Use *--help* to find out more about any (sub)command and its options, including detailed description and examples (*q* to close). Use *-h* to get a short overview of all options

```
$ datalad save -h
     Usage: datalad save [-h] [-m MESSAGE] [-d DATASET] [-t ID] [-r] [-R LEVELS]
                         [-u] [-F MESSAGE_FILE] [--to-git] [-J NJOBS] [--amend]
                         [--version]
                         [PATH ...]

Use '--help' to get more comprehensive information.
```

USING DATALAD IN THE TERMINAL

Check the installed version:

```
datalad --version
```

copy

For help on using DataLad from the command line (press q to exit):

```
datalad --help
```

copy

For extensive info about the installed package, its dependencies, and extensions, use `datalad wtf`. Let's find out what kind of system we're on:

```
datalad wtf -S system
```

copy

GIT IDENTITY

Check git identity:

```
git config --get user.name
git config --get user.email
```

copy

Configure git identity:

```
git config --global user.name "Adina Wagner"
git config --global user.email "adina.wagner@t-online.de"
```

copy

Use the latest datalad features:

```
git config --global --add datalad.extensions.load next
```

copy

USING DATALAD VIA ITS PYTHON API

Open a Python environment:

```
ipython
```

copy

Import and start using:

```
import datalad.api as dl
dl.create(path='mydataset')
```

copy

Exit the Python environment:

```
exit
```

copy

DIFFERENT WAYS TO USE DATALAD

- DataLad can be used from the command line

```
datalad create mydataset
```

- ... or with its Python API

```
import datalad.api as dl
dl.create(path="mydataset")
```

- ... and other programming languages can use it via system call

```
# in R
> system("datalad create mydataset")
```

- ... or via a graphical user interface "DataLad Gooey"

ACKNOWLEDGEMENTS

Software

- Joey Hess (git-annex)
- The DataLad team & contributors

Illustrations

- The Turing Way project & Scriberia

Funders

NSF 1912266

germany-usa

SPONSORED BY THE
Federal Ministry of Education and Research
BMBF 01GQ1905

JÜLICH Forschungszentrum

EUROPEAN UNION
European Regional Development Fund

cbbs center for behavioral brain sciences

SACHSEN-ANHALT
Ministerium für Wissenschaft und Wirtschaft

Collaborators

Human Brain Project

CONP PCNO

VirtualBrainCloud

OpenNEURO

cbrain

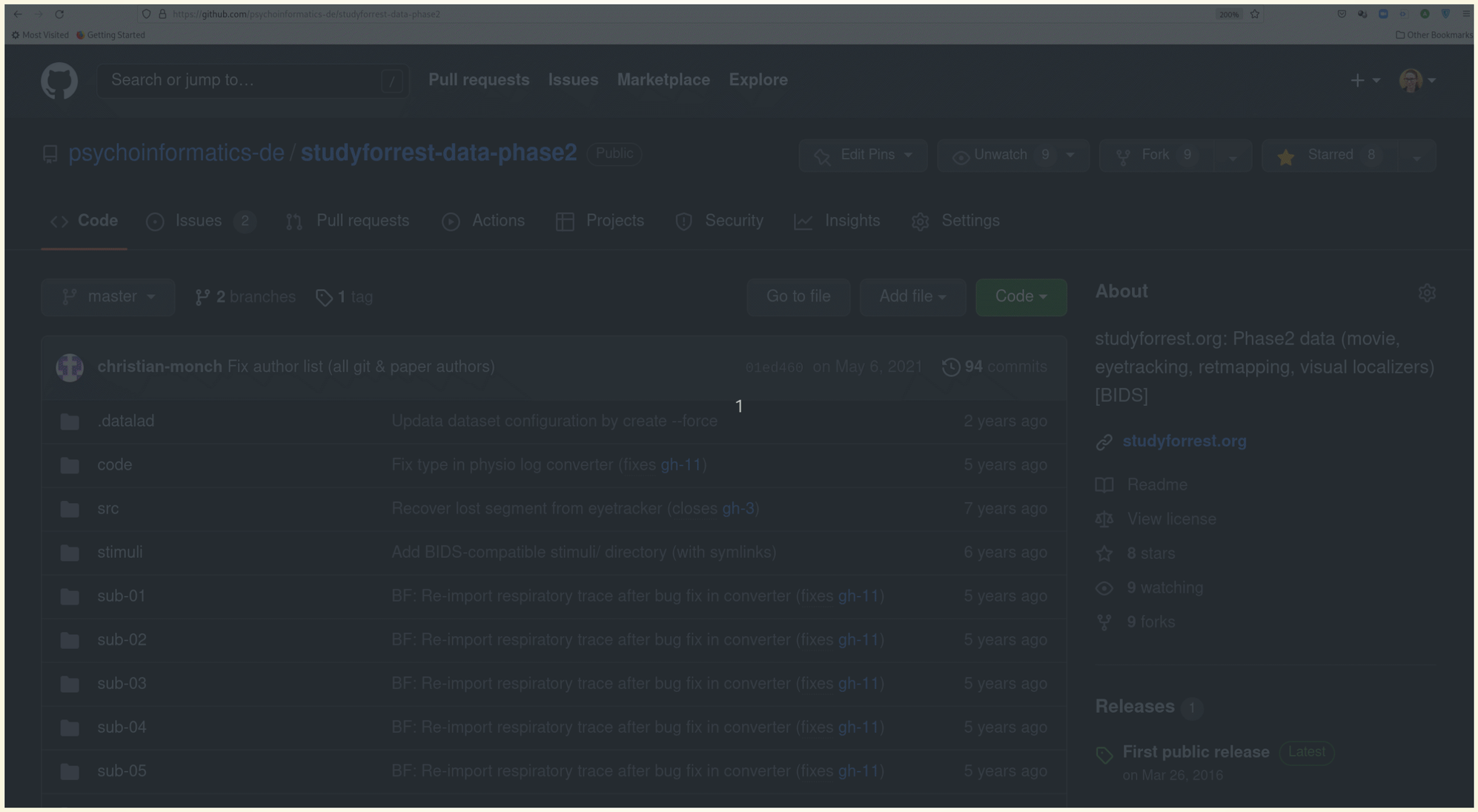
brainlife.io

DataLad CORE FEATURES:

- Joint **version control** (Git, git-annex): version control data & software alongside your code
- **Provenance capture**: Create and share machine-readable, re-executable provenance records for reproducible, transparent, and FAIR research
- Decentral **data transport** mechanisms: Install, share and collaborate on scientific projects; publish, update, and retrieve their contents in a streamlined fashion on demand, and distribute files in a decentral network on the services or infrastructures of your choice

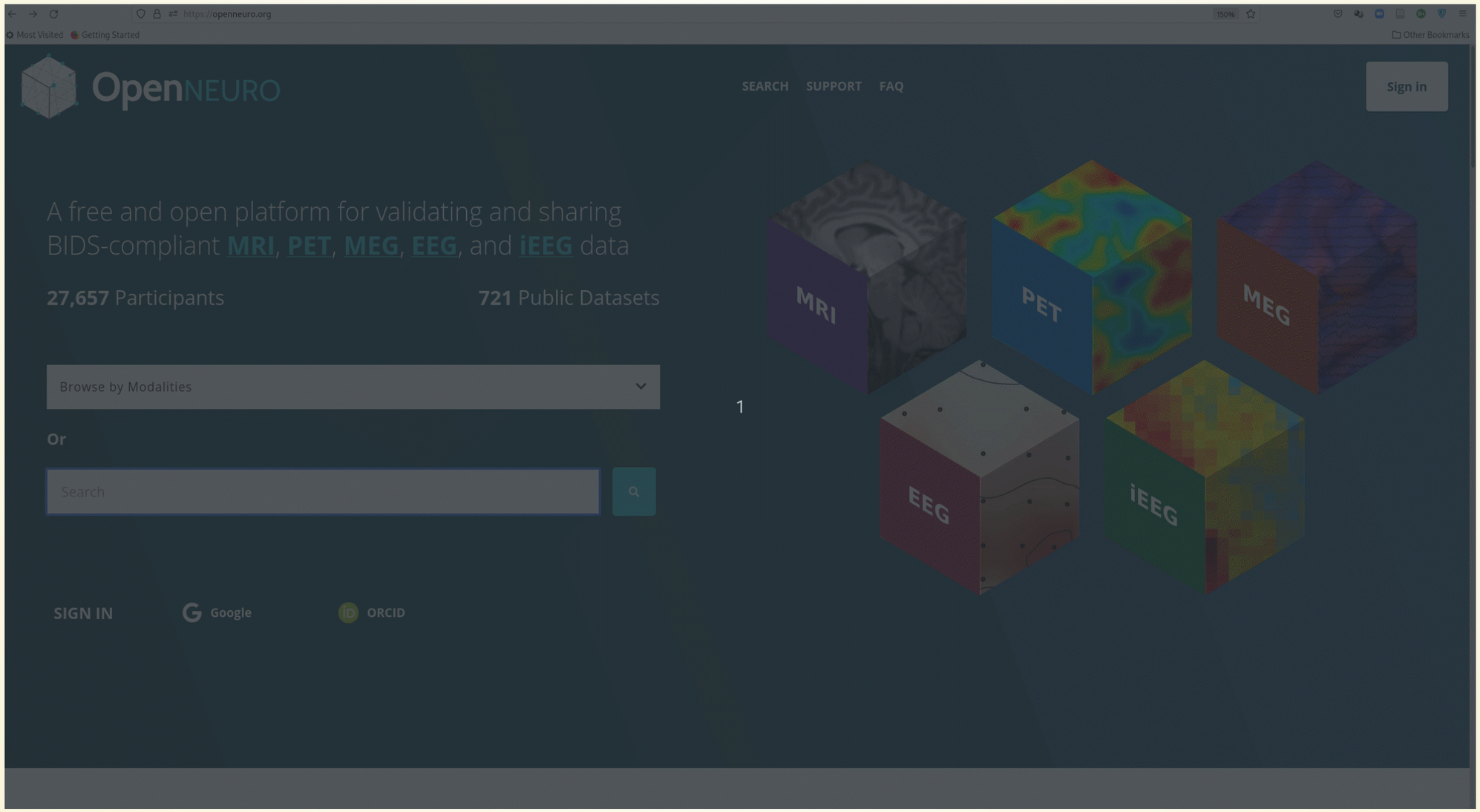
EXAMPLES OF WHAT DATALAD CAN BE USED FOR:

- **Publish or consume datasets** via GitHub, GitLab, OSF, the European Open Science Cloud, or similar services



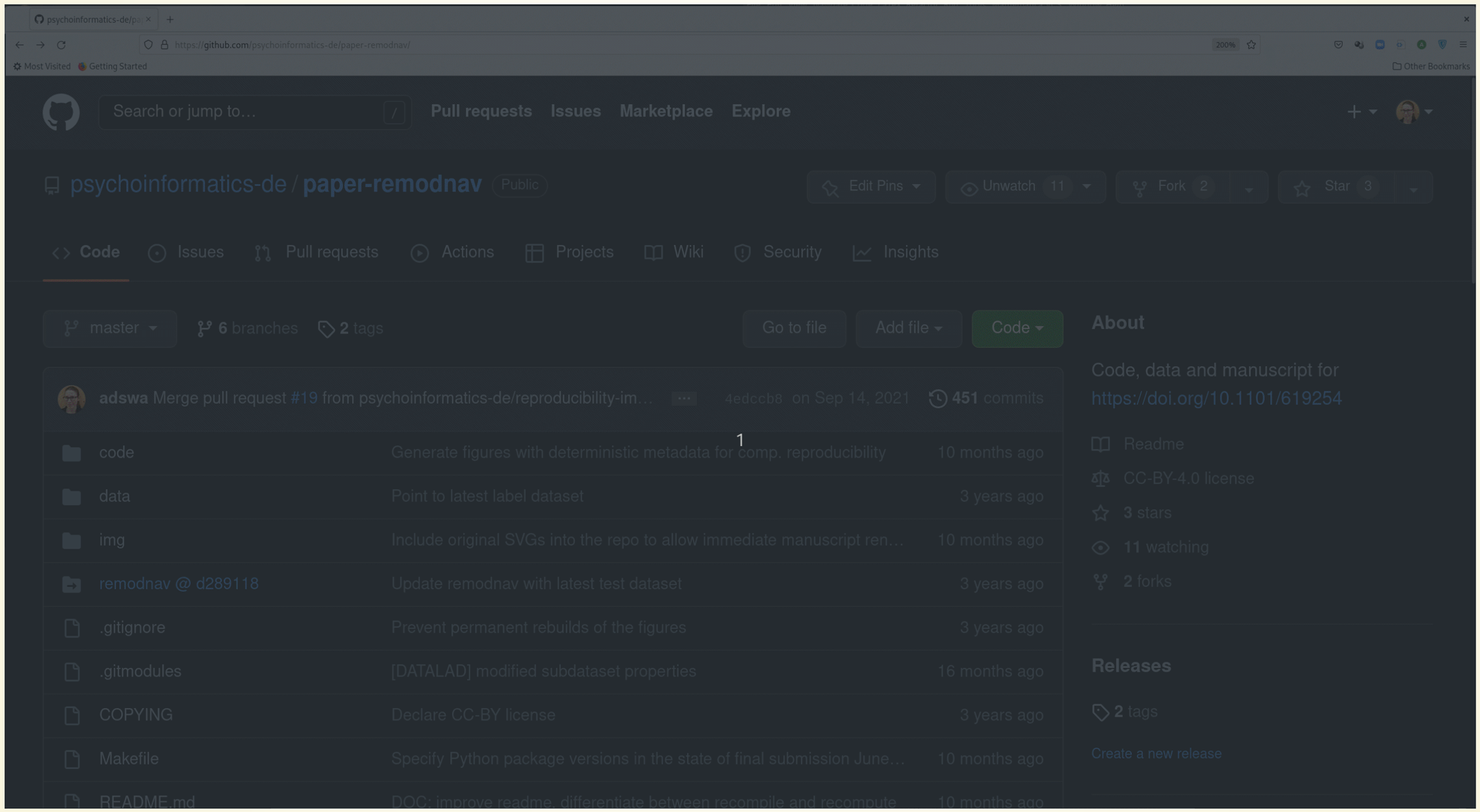
EXAMPLES OF WHAT DATALAD CAN BE USED FOR:

- Behind-the-scenes **infrastructure component for data transport and versioning** (e.g., used by OpenNeuro, brainlife.io , the Canadian Open Neuroscience Platform (CONP), CBRAIN)



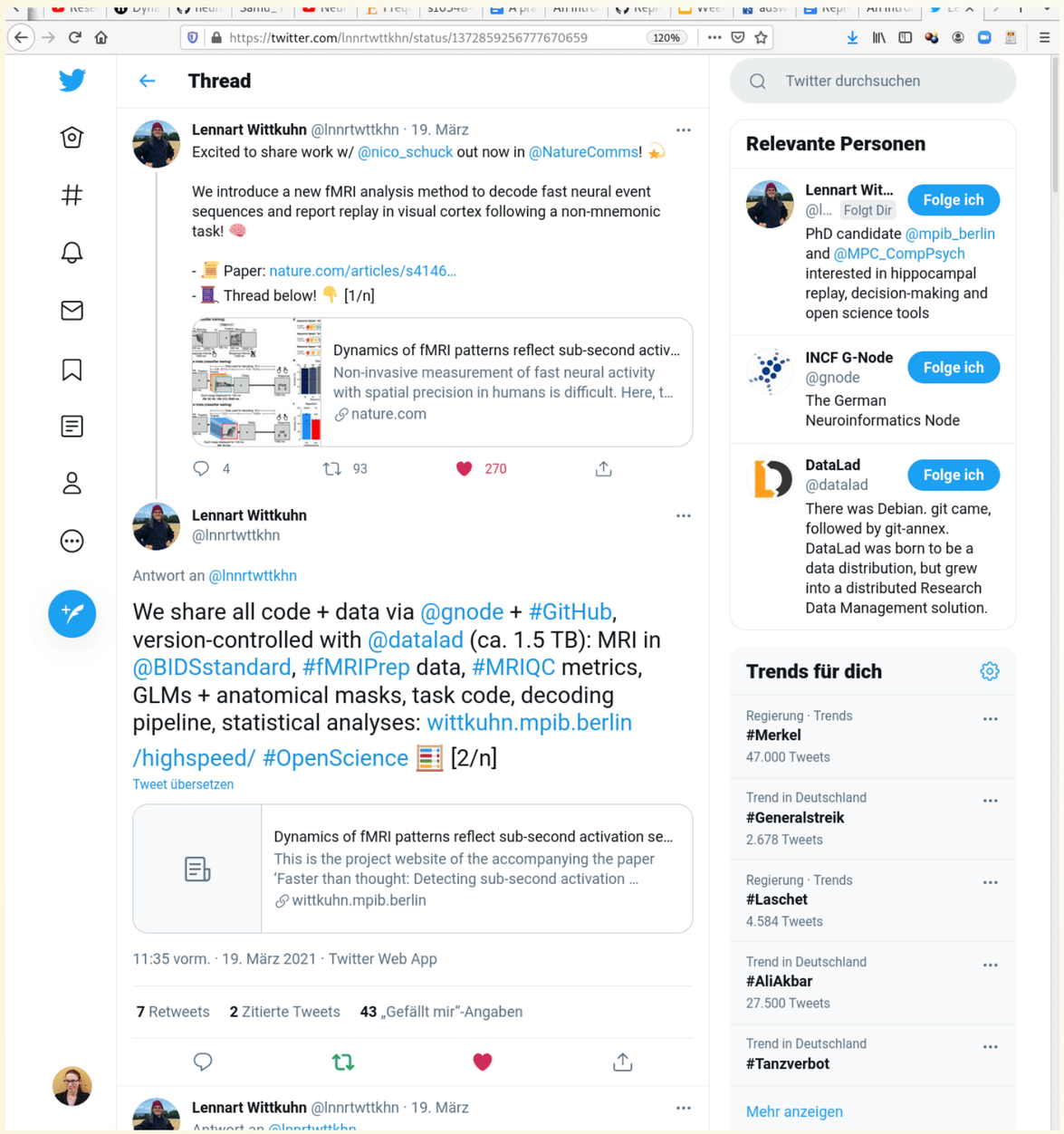
EXAMPLES OF WHAT DATALAD CAN BE USED FOR:

- **Creating and sharing reproducible, open science:** Sharing data, software, code, and provenance



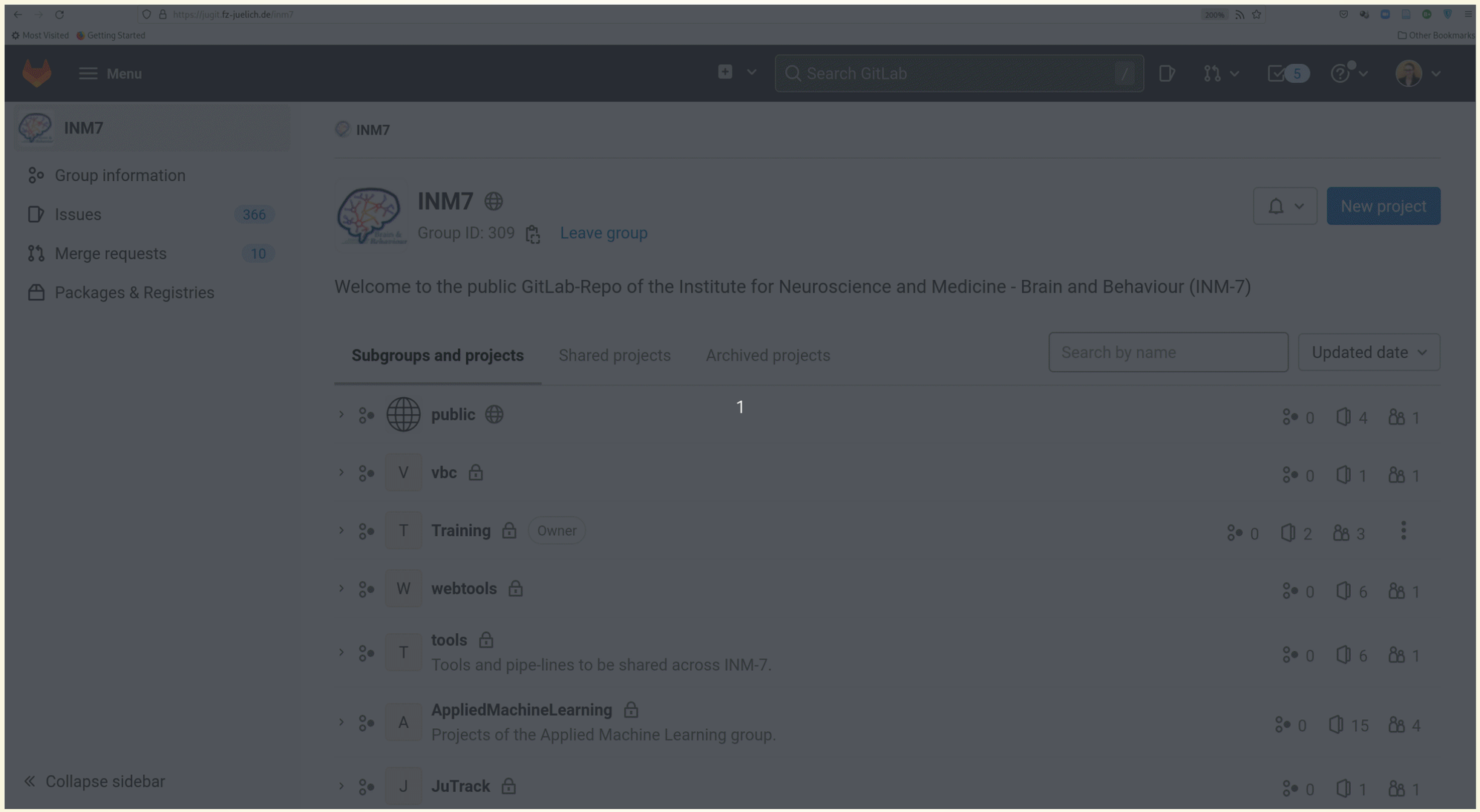
EXAMPLES OF WHAT DATALAD CAN BE USED FOR:

- **Creating and sharing reproducible, open science**: Sharing data, software, code, and provenance



EXAMPLES OF WHAT DATALAD CAN BE USED FOR:

- **Central data management** and archival system



EXAMPLES OF WHAT DATALAD CAN BE USED FOR:

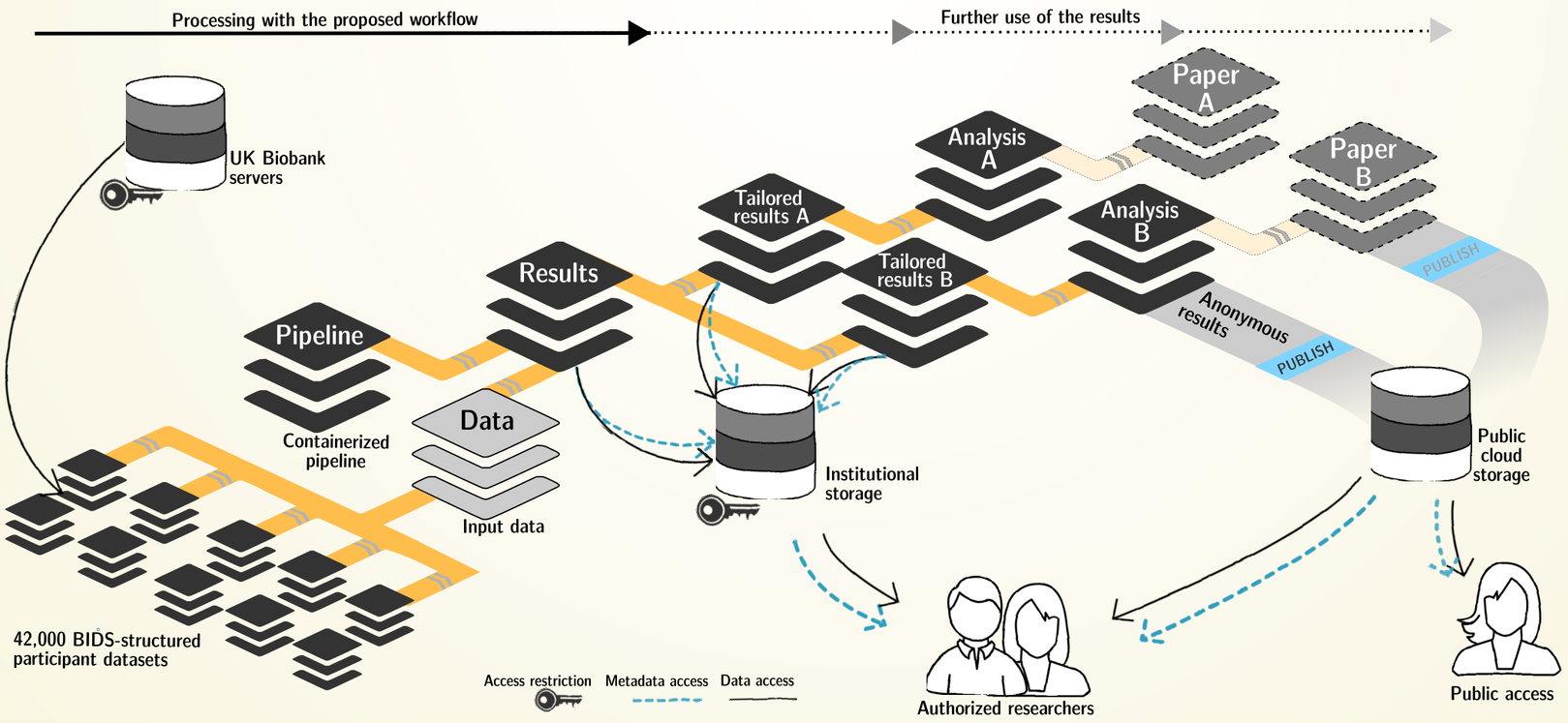
- **Scalable computing framework** for reproducible science

Article | Open Access | Published: 11 March 2022

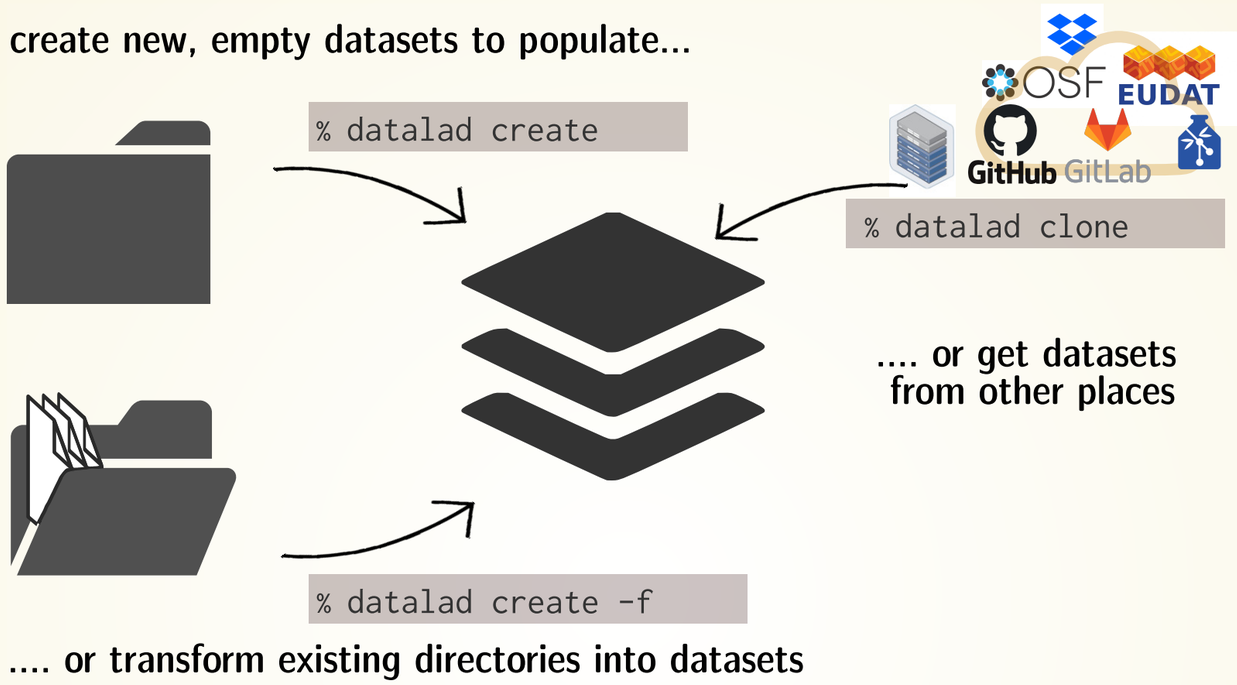
FAIRly big: A framework for computationally reproducible processing of large-scale data

Adina S. Wagner ✉, Laura K. Waite, Małgorzata Wierzba, Felix Hoffstaedter, Alexander Q. Waite, Benjamin Poldrack, Simon B. Eickhoff & Michael Hanke

Scientific Data **9**, Article number: 80 (2022) | Cite this article



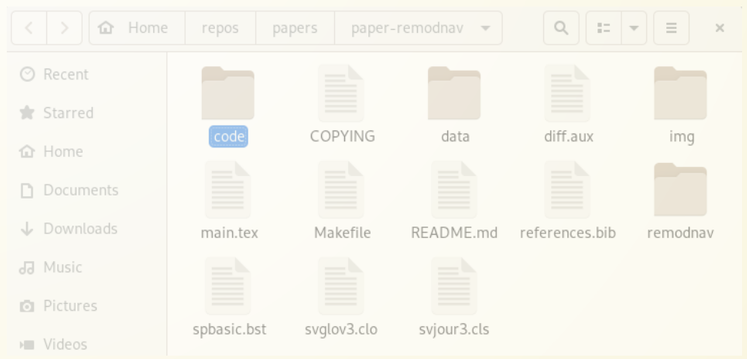
EVERYTHING HAPPENS IN DATALAD DATASETS



- Look and feel like a directory on your computer
- content agnostic
- no custom data structures

```
adina@muninn ~/repos/papers/paper-remodnav on master!
➜ ls
code  remodnav  main.tex   references.bib  svjour3.cls
data  COPYING   Makefile   spbasic.bst
img   diff.aux  README.md  svglov3.clo
```

Terminal view



File viewer

...DATALAD DATASETS

Create a dataset (here, with the `text2git` configuration, which adds a helpful configuration):

```
datalad create -c text2git my-analysis
```

copy

Let's have a look inside. Navigate using `cd` (change directory):

```
cd my-analysis
```

copy

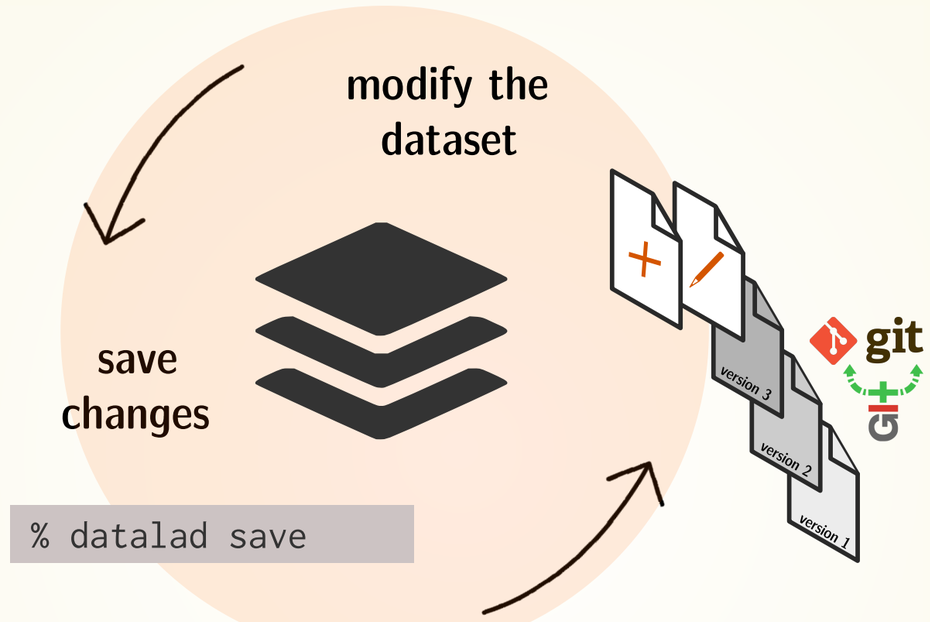
List the directory content, including hidden files, with `ls`:

```
ls -la .
```

copy

DATASET = GIT/GIT-ANNEX REPOSITORY

- version control files regardless of size or type



Stay flexible:

- Non-complex DataLad core API (easy for data management novices)
- Pure Git or git-annex commands (for regular Git or git-annex users, or to use specific functionality)

...VERSION CONTROL

Let's build a dataset for an analysis by adding a README. The command below writes a simple header into a new file README.md:

```
echo "# My example DataLad dataset" > README.md
```
copy

Now we can check the `status` of the dataset:

```
datalad status
```
copy

We can save the state with `save`

```
datalad save -m "Create a short README"
```
copy

Further modifications:

```
echo "This dataset contains a toy data analysis" >> README.md
```
copy

You can also checkout what has changed:

```
git diff
```
copy

Save again:

```
datalad save -m "Add information on the dataset contents to the README"
```
copy

...VERSION CONTROL

Now, let's check the dataset history:

```
git log
```

copy

We can also make the history prettier:

```
tig
```

copy

```
(navigate with arrow keys and enter, press "q" to go back and exit the program)
```

EXHAUSTIVE TRACKING

The building blocks of a scientific result are rarely static

Analysis code evolves

(Fix bugs, add functions, refactor, ...)

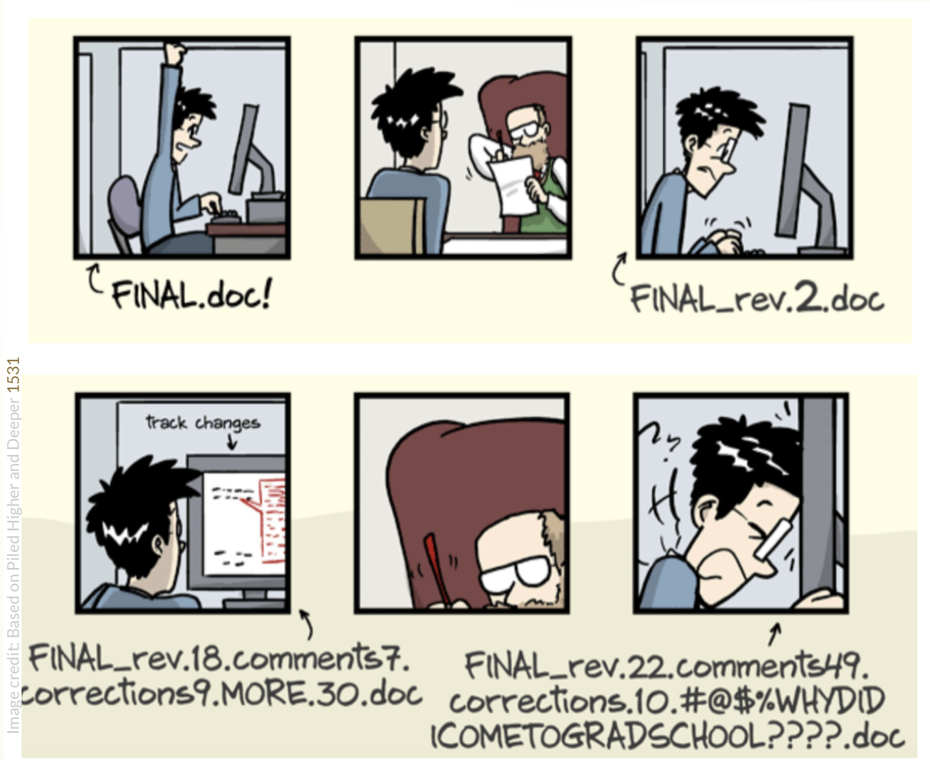


Image credit: Based on Piled Higher and Deeper 1531

EXHAUSTIVE TRACKING

The building blocks of a scientific result are rarely static

Data changes

(errors are fixed, data is extended, naming standards change, an analysis requires only a subset of your data...)

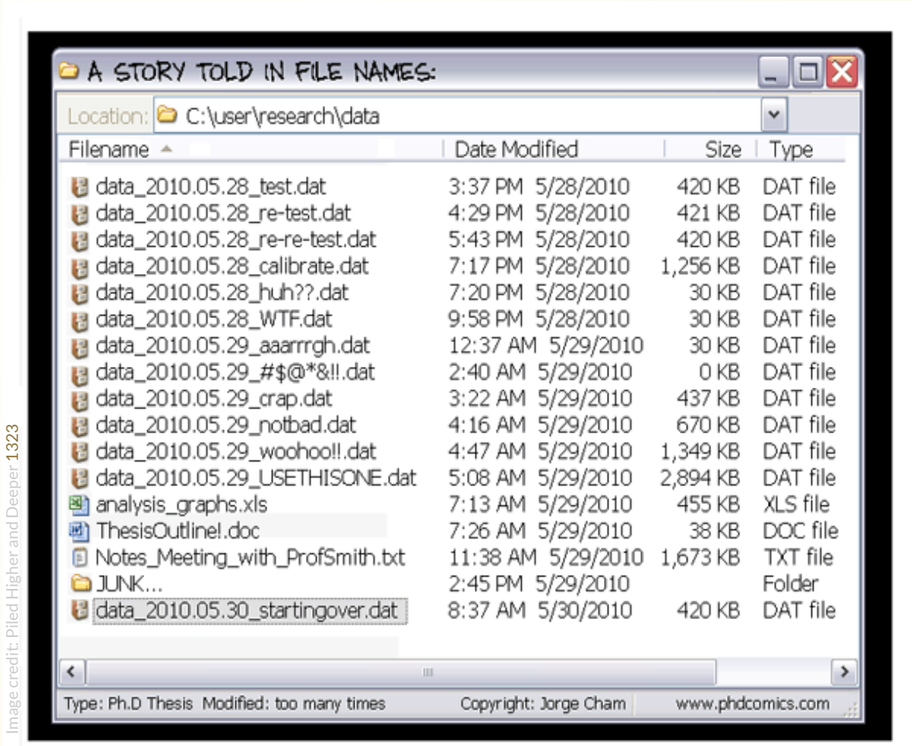


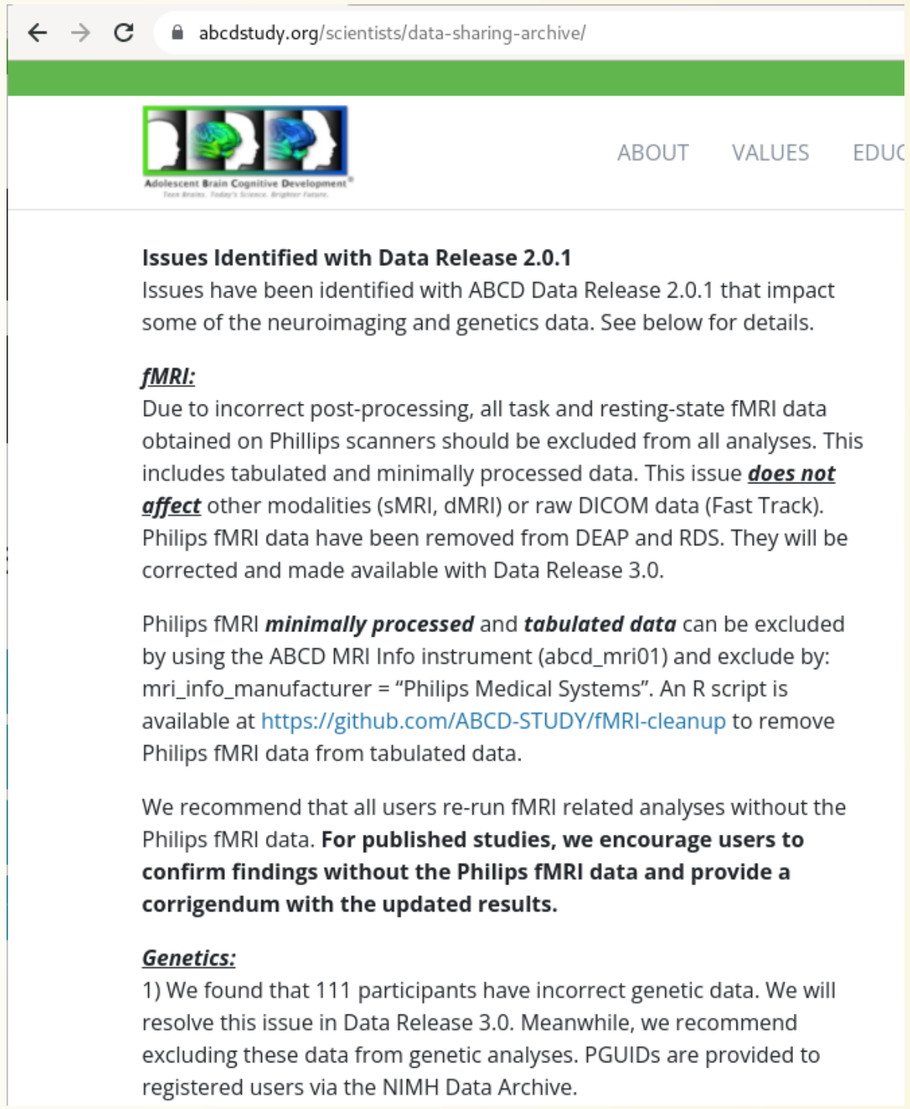
Image credit: Piled Higher and Deeper 1323

EXHAUSTIVE TRACKING

The building blocks of a scientific result are rarely static

Data changes (for real)

(errors are fixed, data is extended,
naming standards change, ...)



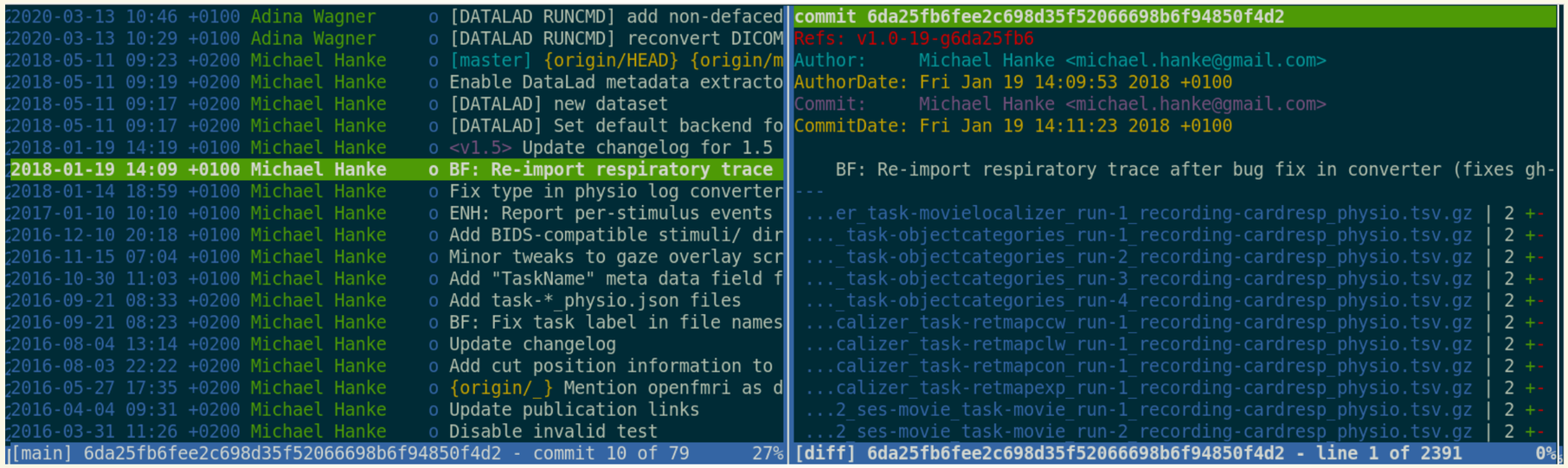
EXHAUSTIVE TRACKING

"Shit, which version of which script produced these outputs from which version of what data... and which software version?"

Image credit: CC-BY Scriberia and The Turing Way

EXHAUSTIVE TRACKING

Once you track changes to data with version control tools, you can find out *why* it changed, *what* has changed, *when* it changed, and *which version* of your data was used at which point in time.



EXHAUSTIVE TRACKING

With the `datalad-container` extension, we can not only add code or data, but also software containers to datasets and work with them. Let's add a software container with Python software for later:

```
datalad containers-add nilearn \
     --url shub://adswa/nilearn-container:latest
```

copy

inspect the list of registered containers:

```
datalad containers-list
```

copy

DIGITAL PROVENANCE

= "The tools and processes used to create a digital file, the responsible entity, and when and where the process events occurred"

- Have you ever saved a PDF to read later onto your computer, but forgot where you got it from? Or did you ever find a figure in your project, but forgot which analysis step produced it?

DIGITAL PROVENANCE

Imagine that you are getting a script from a colleague to perform your analysis, but they email it to you or upload it to a random place for to download:

```
wget -P code/ \
   https://raw.githubusercontent.com/datalad-handbook/resources/master/get_brainmask.py
```

The `wget` command downloaded a script for extracting a brain mask:

```
datalad status
```

Save it into your dataset to have the script ready:

```
datalad save -m "Adding a nilearn-based script for brain masking"
```

Convenience functions make downloads easier. Let's add a nilearn tutorial, and also register the original location of this file as digital provenance:

```
datalad download-url -m "Add a tutorial on nilearn" \
   -O code/nilearn-tutorial.pdf \
   https://raw.githubusercontent.com/datalad-handbook/resources/master/nilearn-tutorial.pdf
```

Notice how its automatically saved:

```
datalad status
```

Check out the file's history:

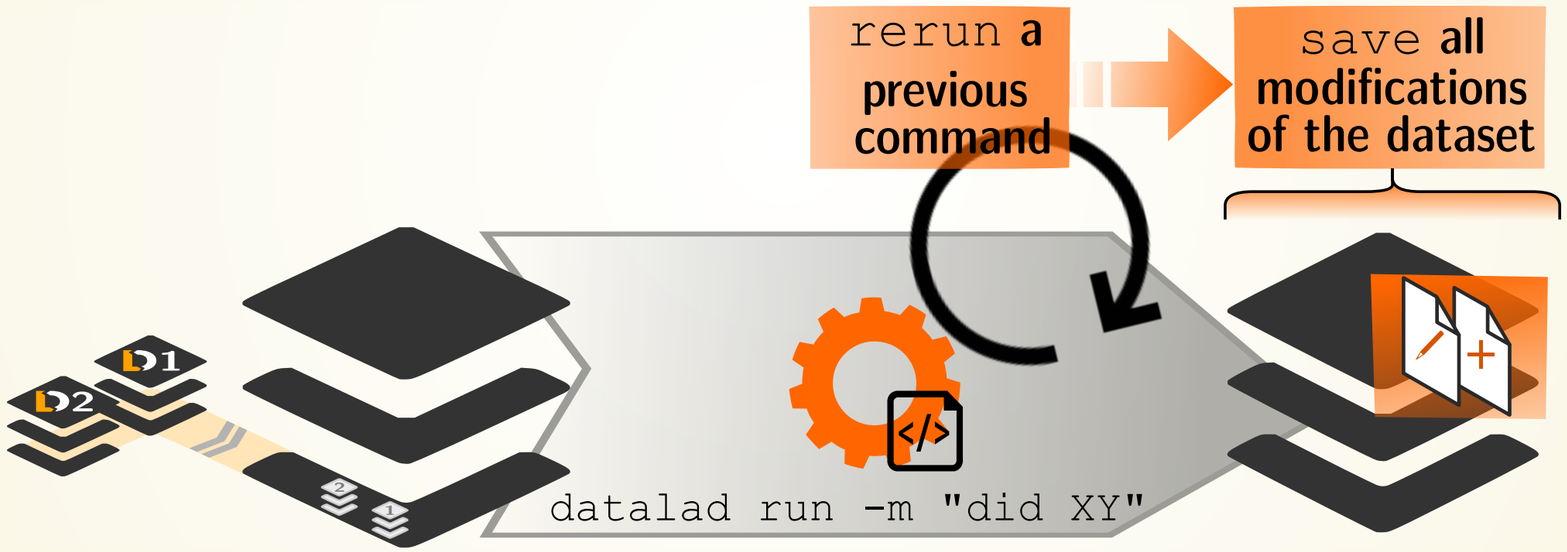
```
git log code/nilearn-tutorial.pdf
```

PROVENANCE AND REPRODUCIBILITY

datalad run wraps around anything expressed in a command line call and saves the dataset modifications resulting from the execution

PROVENANCE AND REPRODUCIBILITY

datalad rerun repeats captured executions.
If the outcomes differ, it saves a new state of them.



... COMPUTATIONALLY REPRODUCIBLE EXECUTION I

A variety of processes can modify files. A simple example: Code formatting

```
black code/get_brainmask.py
```

Version control makes changes transparent:

```
git diff
```

But its useful to keep track beyond that. Let's discard the latest changes...

```
git restore code/get_brainmask.py
```

... and record precisely what we did

```
datalad run -m "Reformat code with black" \
 "black code/get_brainmask.py"
```

let's take a look (press q to exit):

```
git show
```

... and repeat!

```
datalad rerun
```

SEAMLESS DATASET NESTING & LINKAGE

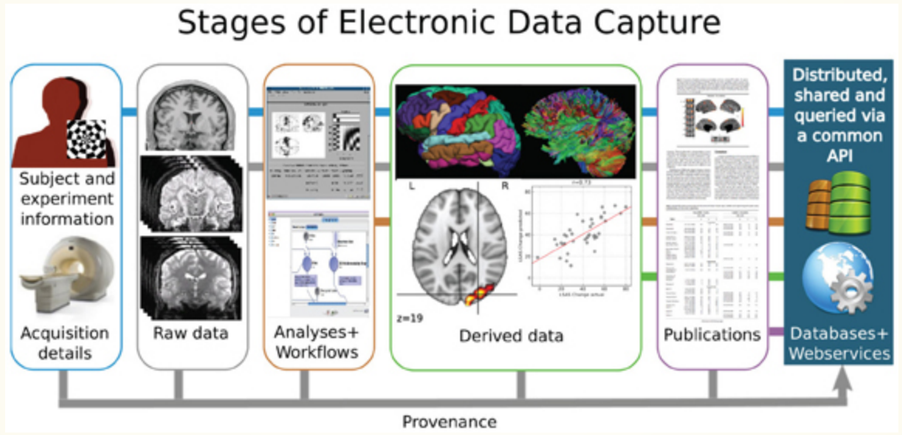
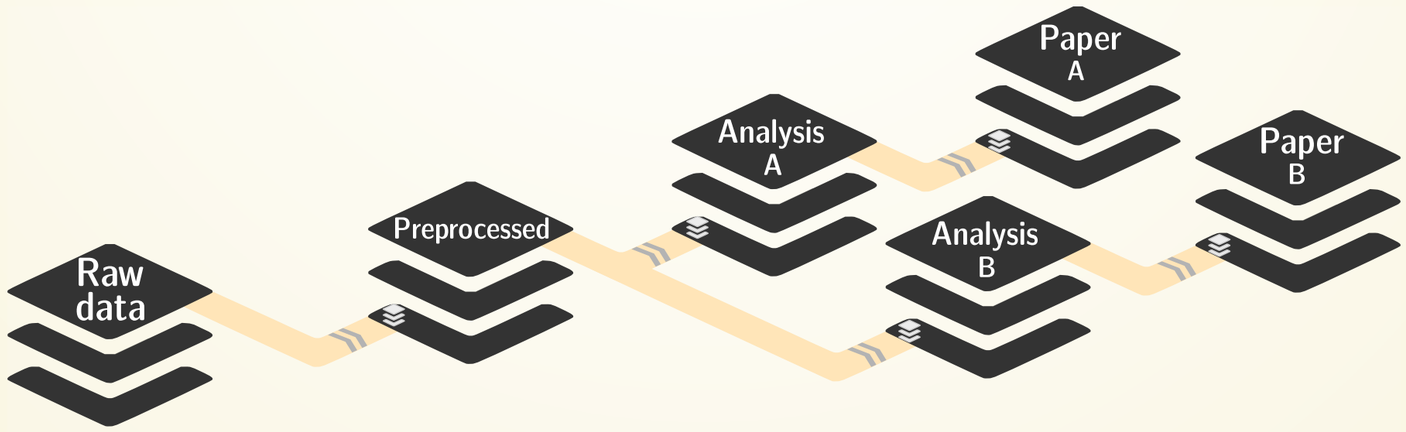
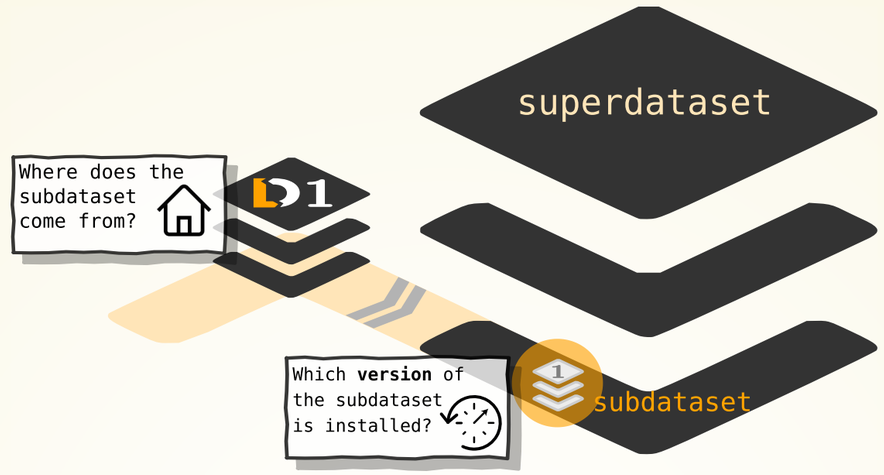


Image credit: Poline et al., 2011



Nest modular datasets to create a linked hierarchy of datasets, and enable recursive operations throughout the hierarchy

SEAMLESS DATASET NESTING & LINKAGE




```
$ datalad clone --dataset . http://example.com/ds inputs/rawdata
```

copy

```
$ git diff HEAD~1
diff --git a/.gitmodules b/.gitmodules
new file mode 100644
index 0000000..c3370ba
--- /dev/null
+++ b/.gitmodules
@@ -0,0 +1,3 @@
+[submodule "inputs/rawdata"]
+       path = inputs/rawdata
+       datalad-id = 68bdb3f3-eafa-4a48-bddd-31e94e8b8242
+       datalad-url = http://example.com/importantds
diff --git a/inputs/rawdata b/inputs/rawdata
new file mode 160000
index 0000000..fabf852
--- /dev/null
+++ b/inputs/rawdata
@@ -0,0 +1 @@
+Subproject commit fabf8521130a13986bd6493cb33a70e580ce8572
```

copy

...DATASET NESTING

Let's make a nest!
Clone a dataset with analysis data into a specific location ("input/") in the existing dataset, making it a *sub*dataset:

```
datalad clone -d . \
 https://gin.g-node.org/adswa/bids-data \
 input
```

copy

Let's see what changed in the dataset, using the `subdatasets` command:

```
datalad subdatasets
```

copy

... and also `git show`:

```
git show
```

copy

We can now view the cloned dataset's file tree:

```
cd input
ls
```

copy

...and also its history

```
tig
```

copy

Let's check the dataset size (with the du disk-usage command):

```
du -sh
```

copy

Let's check the *actual* dataset size:

```
datalad status --annex
```

copy

You can `get` or `drop` annexed file contents depending on your needs:

```
datalad get sub-02
```

copy

```
datalad drop sub-02
```

copy

...COMPUTATIONALLY REPRODUCIBLE EXECUTION...

Try to execute the downloaded analysis script. Does it work?

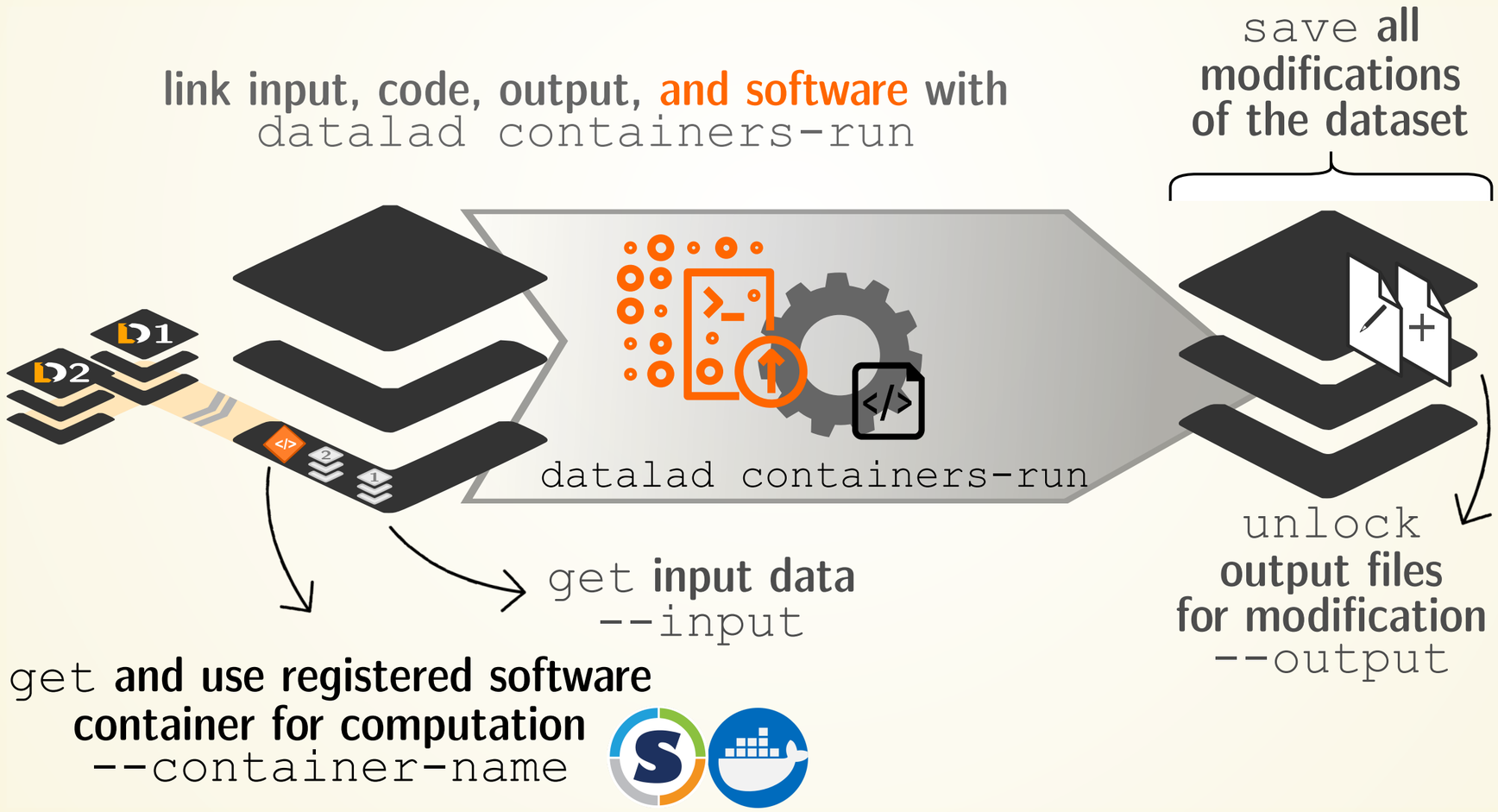
```
cd ..
datalad run -m "Compute brain mask" \
  --input input/sub-02/func/sub-02_task-oneback_run-01_bold.nii.gz \
  --output "figures/*" \
  --output "sub-02*" \
  "python code/get_brainmask.py"
```

copy

- Software can be difficult or impossible to install (e.g. conflicts with existing software, or on HPC) for you or your collaborators
- Different software versions/operating systems can produce different results: Glatard et al., doi.org/10.3389/fninf.2015.00012
- **Software containers** encapsulate a software environment and isolate it from a surrounding operating system. Two common solutions: Docker, Singularity

COMPUTATIONAL PROVENANCE

- The `datalad-container` extension gives DataLad commands to register software containers as "just another file" to your dataset, and **datalad containers-run** analysis inside the container, capturing software as additional provenance



...COMPUTATIONALLY REPRODUCIBLE EXECUTION

Let's try out the `containers-run` command:

```
datalad containers-run -m "Compute brain mask" \
     -n nilearn \
     --input input/sub-02/func/sub-02_task-oneback_run-01_bold.nii.gz \
     --output "figures/*" \
     --output "sub-02*" \
     "python code/get_brainmask.py"
```
copy

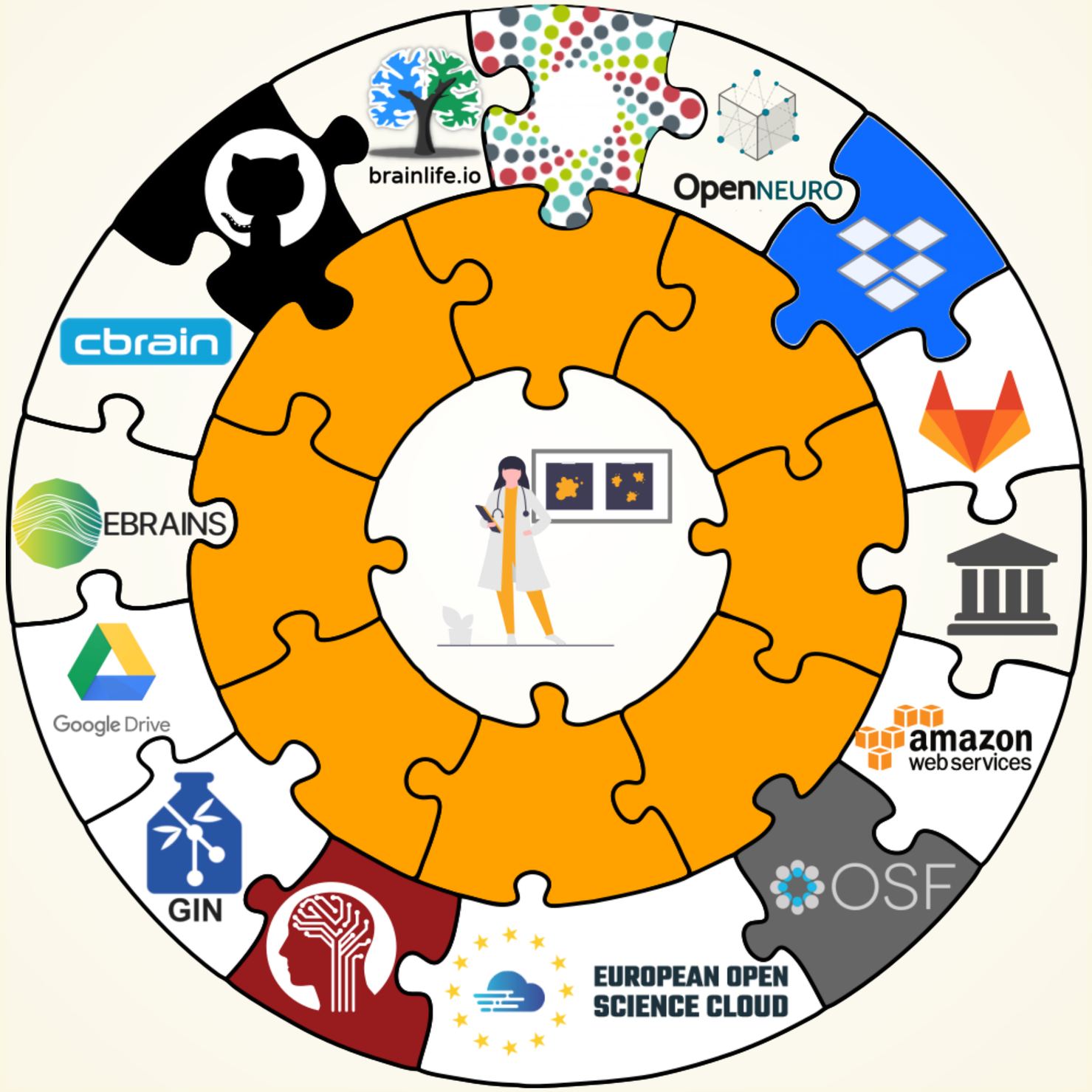
You can now query an individual file how it came to be...

```
git log sub-02_brain-mask.nii.gz
```
copy

... and the computation can be redone automatically and checked for computational reproducibility based on the recorded provenance using datalad rerun:

```
datalad rerun
```
copy

SHARING DATASETS



Apart from **local computing infrastructure** (from private laptops to computational clusters), datasets can be hosted in major **third party repository hosting and cloud storage** services. More info: Chapter on Third party infrastructure.

TRANSPORT LOGISTICS: LOTS OF DATA, LITTLE DISK-USAGE

- Cloned datasets are lean. "Meta data" (file names, availability) are present, but **no file content**:

```
$ datalad clone git@github.com:psychoinformatics-de/studyforrest-data-phase2.git
  install(ok): /tmp/studyforrest-data-phase2 (dataset)
$ cd studyforrest-data-phase2 && du -sh
  18M .
```

- files' contents can be retrieved on demand:

```
$ datalad get sub-01/ses-movie/func/sub-01_ses-movie_task-movie_run-1_bold.nii.gz
  get(ok): /tmp/studyforrest-data-phase2/sub-01/ses-movie/func/sub-01_ses-movie_task-movie_run-1
```

copy

- Have access to more data on your computer than you have disk-space:

```
# eNKI dataset (1.5TB, 34k files):
$ du -sh
1.5G  .
# HCP dataset (~200TB, >15 million files)
$ du -sh
48G .
```

copy

PLENTY OF DATA, BUT LITTLE DISK-USAGE

Drop file content that is not needed:

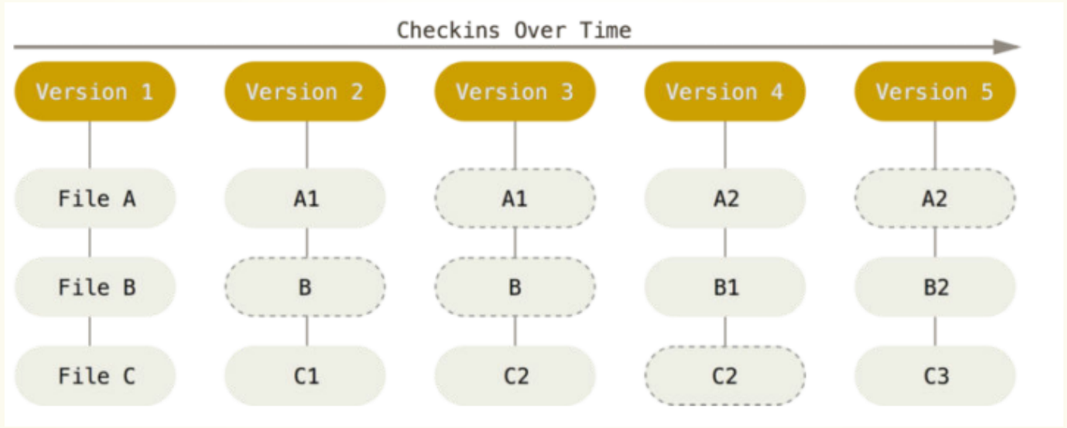
```
$ datalad drop sub-01/ses-movie/func/sub-01_ses-movie_task-movie_run-1_bold.nii.gz
drop(ok): /tmp/studyforrest-data-phase2/sub-01/ses-movie/func/sub-01_ses-movie_task-movie_run-1_
```

When files are dropped, only "meta data" stays behind, and they can be re-obtained on demand.

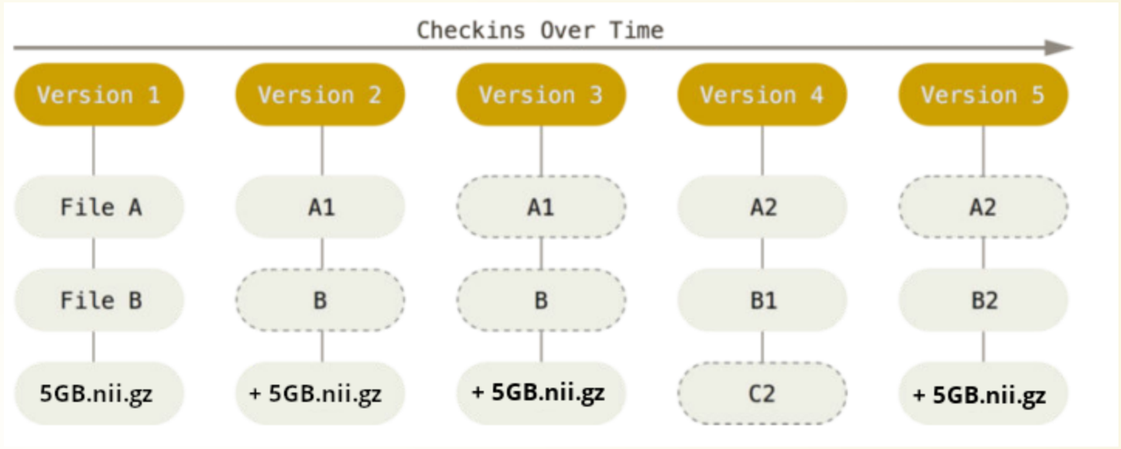
```
dl.get('input/sub-01')

[really complex analysis]

dl.drop('input/sub-01')
```

THERE ARE TWO VERSION CONTROL TOOLS AT WORK - WHY?

Git does not handle large files well.



THERE ARE TWO VERSION CONTROL TOOLS AT WORK - WHY?



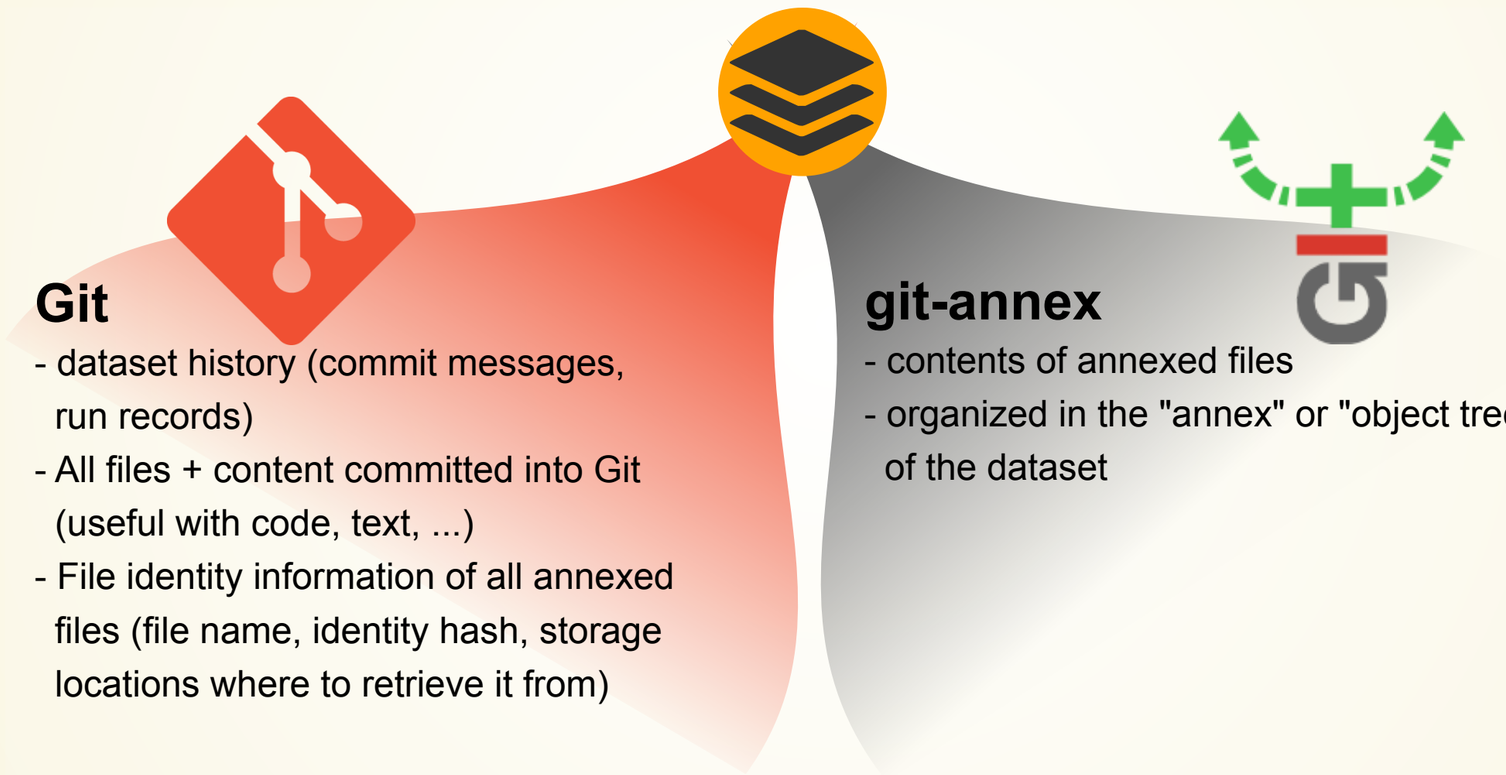
Git does not handle large files well.

And repository hosting services refuse to handle large files:

```
adina@muninn in /tmp/myresearch on git:master
❯ git push gh-adswa master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 497.87 KiB | 161.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote: error: Trace: 64a78dd41ece8e5493fe33f97397a7a90ef9c91260ba32786970dbdcf5c4e0dd
remote: error: See http://git.io/iEPt8g for more information.
remote: error: File output.dat is 500.00 MB; this exceeds GitHub's file size limit of 100.00 MB
remote: error: GH001: Large files detected. You may want to try Git Large File Storage - https://git-lfs.github.com.
To github.com:adswa/myresearch.git
 ! [remote rejected] master -> master (pre-receive hook declined)
error: failed to push some refs to 'github.com:adswa/myresearch.git'
```

git-annex to the rescue! Let's take a look how it works

GIT VERSUS GIT-ANNEX



Git

- dataset history (commit messages, run records)
- All files + content committed into Git (useful with code, text, ...)
- File identity information of all annexed files (file name, identity hash, storage locations where to retrieve it from)

git-annex

- contents of annexed files
- organized in the "annex" or "object tree" of the dataset

DATASET INTERNALS

- Where the filesystem allows it, annexed files are symlinks:

```
$ ls -l sub-02/func/sub-02_task-oneback_run-01_bold.nii.gz
lrwxrwxrwx 1 adina adina 142 Jul 22 19:45 sub-02/func/sub-02_task-oneback_run-01_bold.nii.gz ->
../../.git/annex/objects/kZ/K5/MD5E-s24180157--aeb0e5f2e2d5fe4ade97117a8cc5232f.nii.gz/MD5E-s24180
--aeb0e5f2e2d5fe4ade97117a8cc5232f.nii.gz
```

(PS: especially useful in datasets with many identical files)

- The symlink reveals this internal data organization based on identity hash:

```
$ md5sum sub-02/func/sub-02_task-oneback_run-01_bold.nii.gz
aeb0e5f2e2d5fe4ade97117a8cc5232f  sub-02/func/sub-02_task-oneback_run-01_bold.nii.gz
```

- The (tiny) symlink instead of the (potentially large) file content is committed - version controlling precise file identity without checking contents into Git

```
diff --git a/sub-02/func/sub-02_task-oneback_run-01_bold.nii.gz b/sub-02/func/sub-02_task-oneback_run-01_bold.nii.
new file mode 120000
index 0000000..398e7f1
--- /dev/null
+++ b/sub-02/func/sub-02_task-oneback_run-01_bold.nii.gz
@@ -0,0 +1 @@
+../../.git/annex/objects/kZ/K5/MD5E-s24180157--aeb0e5f2e2d5fe4ade97117a8cc5232f.nii.gz/MD5E-s24180157--aeb0e5f2e2
```

- File contents can be shared via almost all standard infrastructure. File availability information is a decentral network. A file can exist in multiple different locations.

```
$ git annex whereis code/nilearn-tutorial.pdf
whereis code/nilearn-tutorial.pdf (2 copies)
        cf13d535-b47c-5df6-8590-0793cb08a90a -- [datalad]
        e763ba60-7614-4b3f-891d-82f2488ea95a -- jovyan@jupyter-adswa:~/my-analysis [here]

  datalad: https://raw.githubusercontent.com/datalad-handbook/resources/master/nilearn-tutorial.pd
```

Delineation and advantages of decentral versus central RDM: Hanke et al., (2021). In defense of decentralized research data management

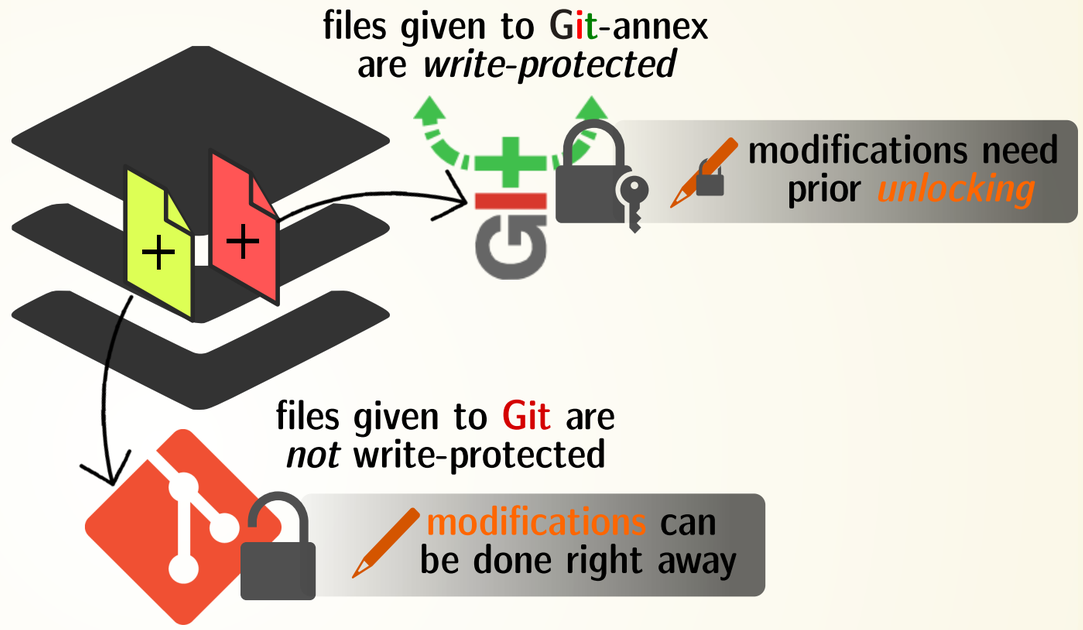
GIT VERSUS GIT-ANNEX

Data in datasets is either stored in Git or git-annex

By default, everything is annexed.

Two consequences:

- Annexed contents are not available right after cloning, only content identity and availability information (as they are stored in Git). Everything that is annexed needs to be retrieved with `datalad get` from whereever it is stored.
- Files stored in Git are modifiable, annexed files are protected against accidental modifcations



Git	git-annex
handles **small** files well (text, code)	handles **all** types and sizes of files well
file contents are in the Git history and will be **shared** upon git/datalad push	file contents are in the annex. Not necessarily shared
Shared with every dataset clone	**Can be kept private** on a per-file level when sharing the dataset
Useful: Small, non-binary, frequently modified, need-to-be-accessible (DUA, README) files	Useful: Large files, private files

Useful background information for demo later. Read this handbook chapter for details

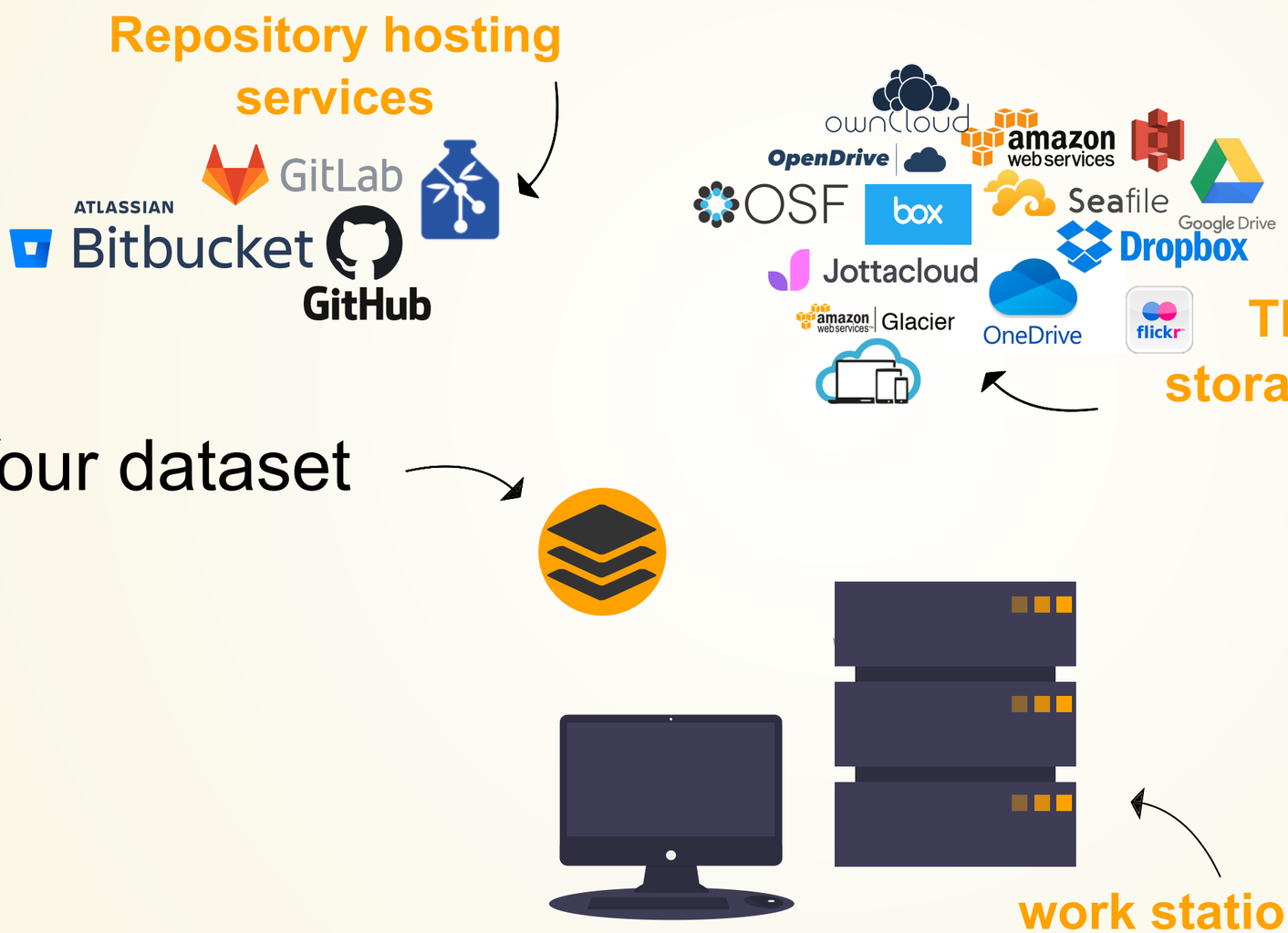
GIT VERSUS GIT-ANNEX

Users can decide which files are annexed:

- **Pre-made run-procedures**, provided by DataLad (e.g., `text2git`, `yoda`) or created and shared by users (Tutorial)
- Self-made configurations in `.gitattributes` (e.g., based on file type, file/path name, size, ...; rules and examples)
- Per-command basis (e.g., via `datalad save --to-git`)

PUBLISHING DATASETS

I have a dataset on my computer. How can I share it, or collaborate on it?



GLOSSARY

Sibling (remote)
: Linked clones of a dataset. You can usually update (from) siblings to keep all your siblings in sync (e.g., ongoing data acquisition stored on experiment compute and backed up on cluster and external hard-drive)

Repository hosting service
: Webservices to host Git repositories, such as GitHub, GitLab, Bitbucket, Gin, ...

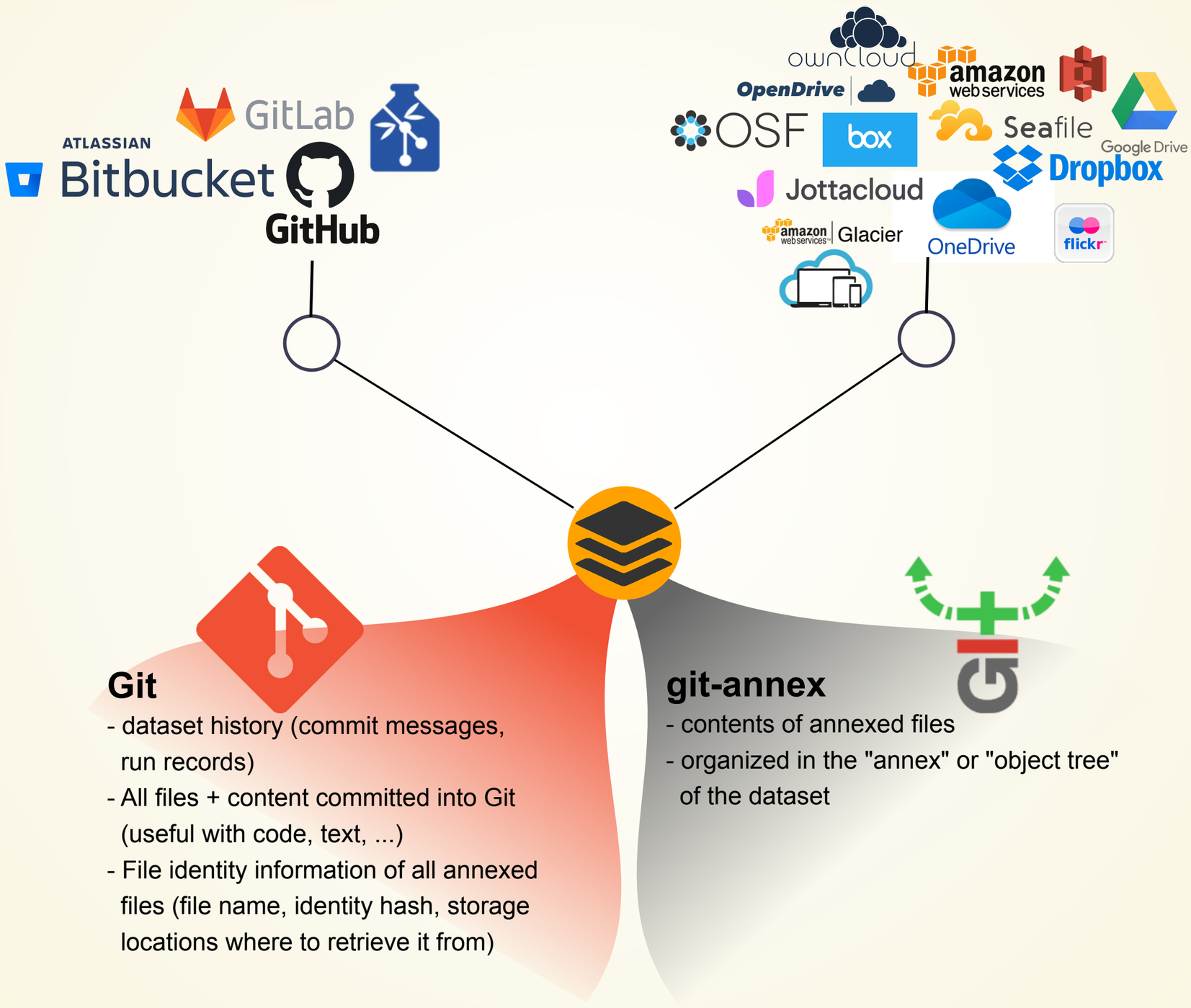
Third-party storage
: Infrastructure (private/commercial/free/...) that can host data. A "special remote" protocol is used to publish or pull data to and from it

Publishing datasets
: *Pushing* dataset contents (Git and/or annex) to a sibling using **datalad push**

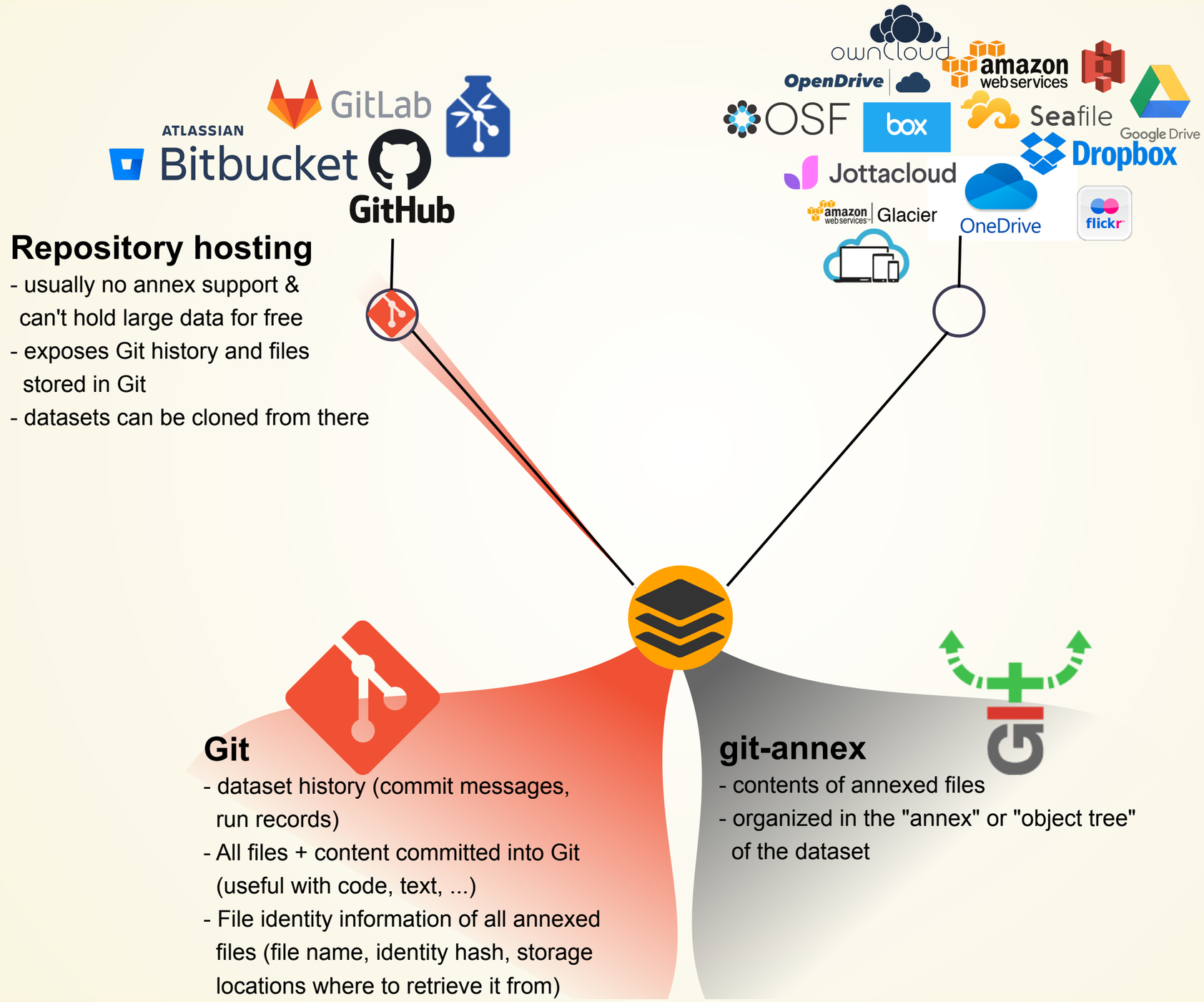
Updating datasets
: *Pulling* new changes from a sibling using **datalad update --merge**

PUBLISHING DATASETS

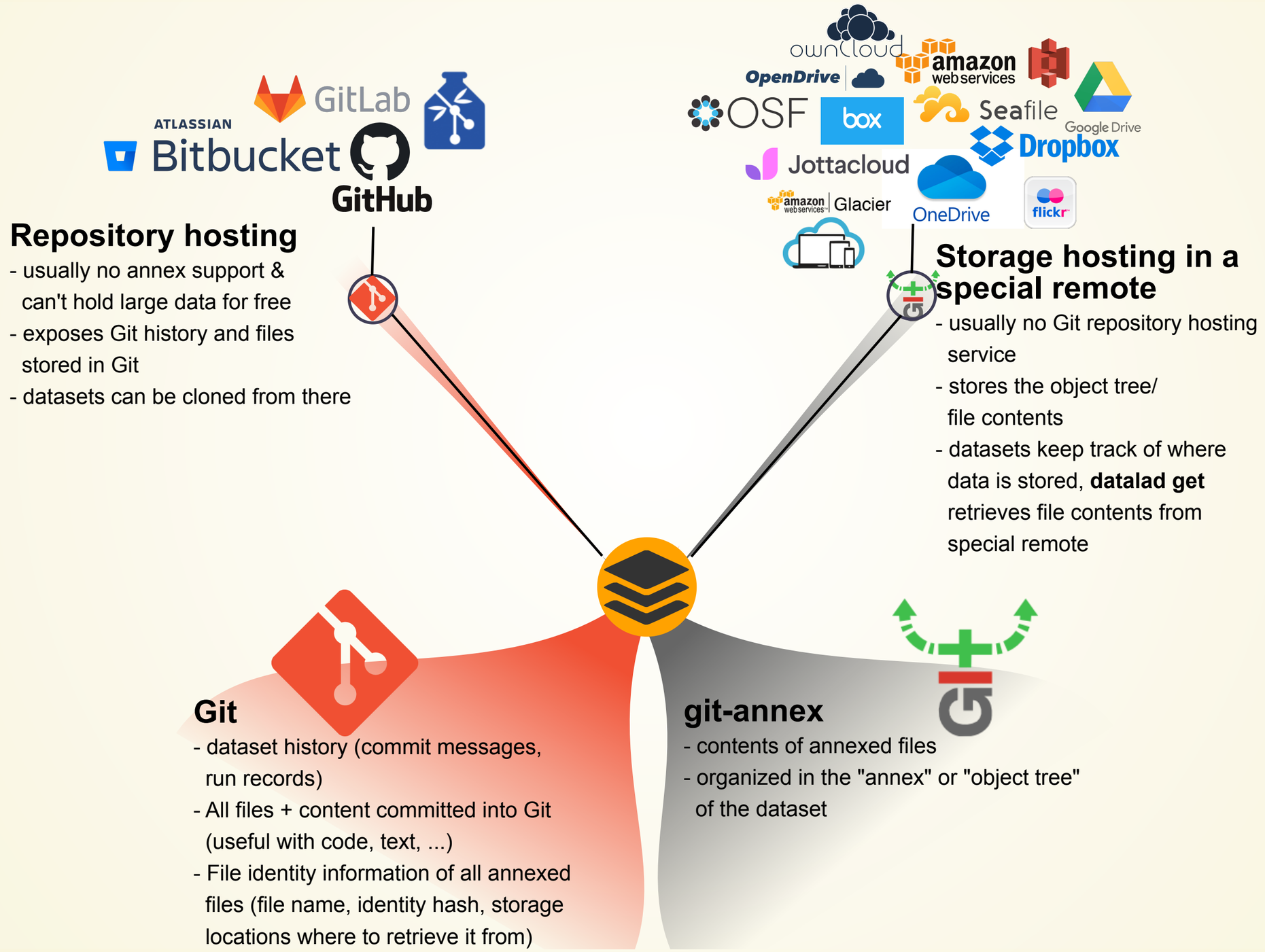
- Most public datasets separate content in Git versus git-annex behind the scenes



PUBLISHING DATASETS



PUBLISHING DATASETS



Repository hosting

- usually no annex support & can't hold large data for free
- exposes Git history and files stored in Git
- datasets can be cloned from there

Storage hosting in a special remote

- usually no Git repository hosting service
- stores the object tree/ file contents
- datasets keep track of where data is stored, **datalad get** retrieves file contents from special remote

Git

- dataset history (commit messages, run records)
- All files + content committed into Git (useful with code, text, ...)
- File identity information of all annexed files (file name, identity hash, storage locations where to retrieve it from)

git-annex

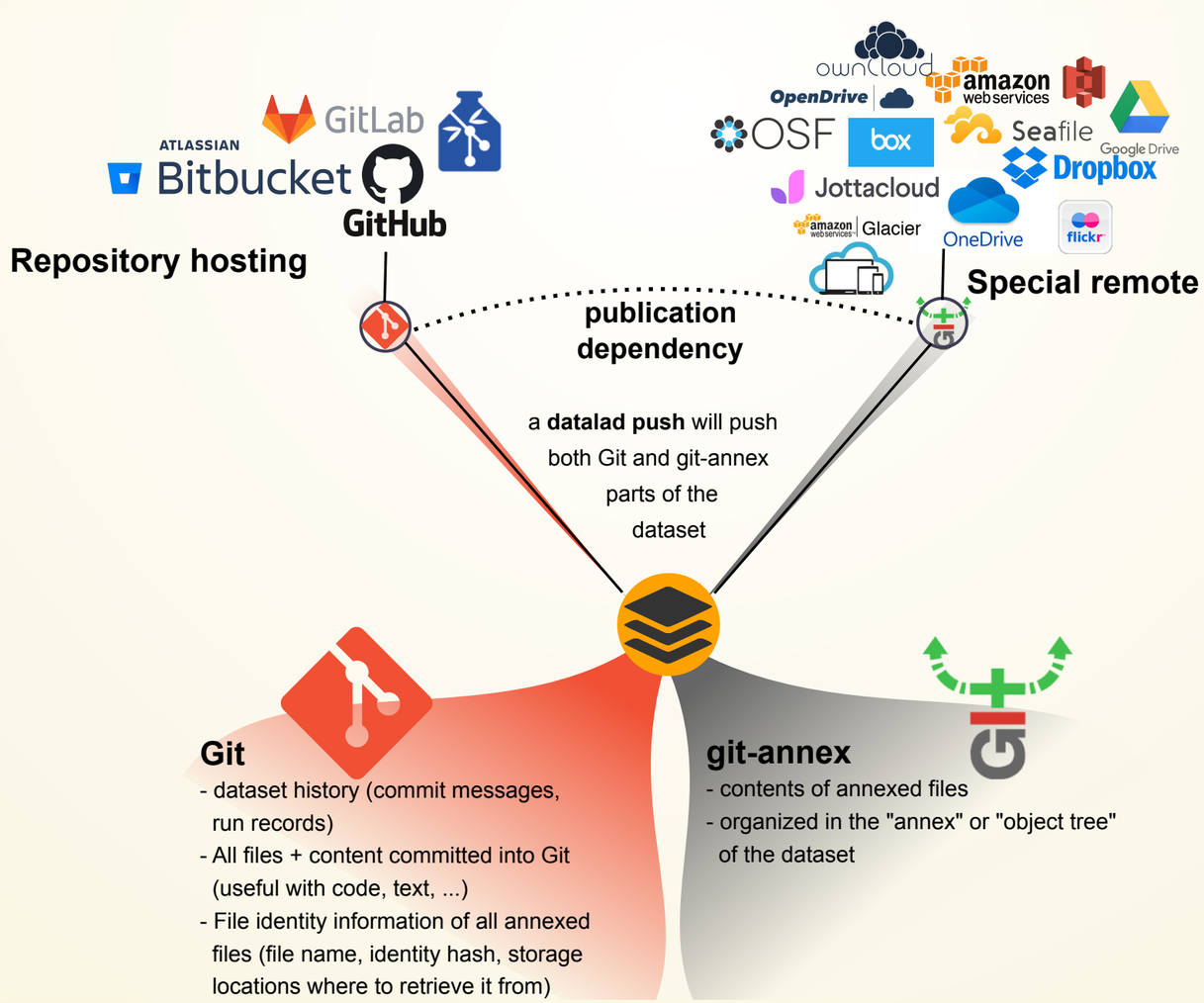
- contents of annexed files
- organized in the "annex" or "object tree" of the dataset

PUBLISHING DATASETS

Typical case:

- Datasets are exposed via a private or public repository on a repository hosting service
- Data can't be stored in the repository hosting service, but can be kept in almost any third party storage
- Publication dependencies automate pushing to the correct place, e.g.,

```
$ git config --local remote.github.datalad-publish-depends gdrive
# or
$ datalad siblings add --name origin --url git@git.jugit.fzj.de:adswa/experiment-data.git --publish-depends s3
```

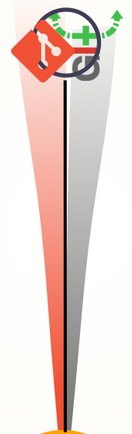


PUBLISHING DATASETS

Special case 1: repositories with annex support

Repositories with annex support

- examples: GIN (gin.g-node.org), GitLab instances with enabled annex support
- can hold large data for free
- exposes Git history and all files + content
- datasets can be cloned from there



Git

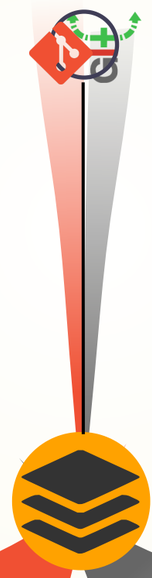
- dataset history (commit messages, run records)
- All files + content committed into Git (useful with code, text, ...)
- File identity information of all annexed files (file name, identity hash, storage locations where to retrieve it from)

git-annex

- contents of annexed files
- organized in the "annex" or "object tree" of the dataset

PUBLISHING DATASETS

Special case 2: Special remotes with repositories



Publishing both components to a special remote

- performed with DataLad extensions (e.g., datalad-osf, datalad-next), or helpers (e.g., git-remote-rclone)
- publishes the Git repository in conjunction with annexed data
- allows cloning from a special remote

Git- dataset history (commit messages, run records)- All files + content committed into Git (useful with code, text, ...)- File identity information of all annexed files (file name, identity hash, storage locations where to retrieve it from)

git-annex- contents of annexed files- organized in the "annex" or "object tree" of the dataset

Publishing to OSF

https://osf.io/

CREATE-SIBLING-OSF (docs)

Requires the DataLad extensions `datalad-osf` and `datalad-next`

Prerequisites:

1. Log into OSF
2. Create personal access token
3. Enter credentials using `datalad osf-credentials`:

```
datalad osf-credentials
```
copy

CREATE-SIBLING-OSF (docs)

Create the sibling in your dataset (different modes are possible):

```
datalad create-sibling-osf -d . -s my-osf-sibling \
--title 'my-osf-project-title' --mode export --public
```

copy

Push to the sibling:

```
datalad push -d . --to my-osf-sibling
```

copy

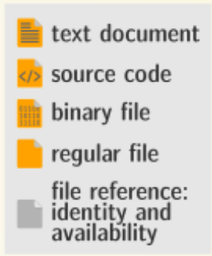
Clone from the sibling:

```
cd ..
datalad clone osf://my-osf-project-id my-osf-clone
```

copy

SUMMARY AND TAKE-HOME MESSAGES

EXHAUSTIVE TRACKING OF RESEARCH COMPONENTS

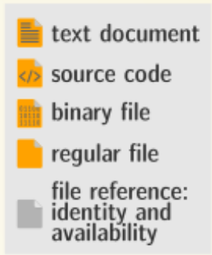


Well-structured datasets (using community standards), and portable computational environments — and their evolution — are the precondition for reproducibility

```
# turn any directory into a dataset
# with version control


% datalad create <directory>
```

```
# save a new state of a dataset with
# file content of any size


% datalad save
```

CAPTURE COMPUTATIONAL PROVENANCE



Which data was needed at which version, as input into which code, running with what parameterization in which computional environment, to generate an outcome?

```
# execute any command and capture its output
# while recording all input versions too


% datalad run --input ... --output ... <command>
```

EXHAUSTIVE CAPTURE ENABLES PORTABILITY

Precise identification of data and computational environments combined with provenance records form a comprehensive and portable data structure, capturing all aspects of an investigation.

```
# transfer data and metadata to other sites and services
# with fine-grained access control for dataset components


% datalad push --to <site-or-service>
```

REPRODUCIBILITY STRENGTHENS TRUST

Outcomes of computational transformations can be validated by authorized 3rd-parties. This enables audits, promotes accountability, and streamlines automated "upgrades" of outputs

```
# obtain dataset (initially only identity,
# availability, and provenance metadata)


% datalad clone <url>
```

```
# immediately actionable provenance records
# full abstraction of input data retrieval


% datalad rerun <commit|tag|range>
```

ULTIMATE GOAL: (RE-)USABILITY

Verifiable, portable, self-contained data structures that track all aspects of an investigation exhaustively can be (re-)used as modular components in larger contexts — propagating their traits

```
# declare a dependency on another dataset and
# re-use it a particular state in a new context


% datalad clone -d <superdataset> <url> <path-in-dataset>
```

YOUR QUESTIONS AND USECASES

POST-WORKSHOP CONTACT

- Slides are CC-BY. They will stay online and will be made available as a PDF as well
- Contact the DataLad Team anytime via GitHub issue, Matrix chat message, or in our office hour video call
- Find more DataLad content and tutorials at handbook.datalad.org

- Join us at our first conference for distributed data management: distribits.live (April 2024, registration closes October 15th)

THANKS FOR YOU ATTENTION!

LIST OF INSTALLED SOFTWARE ON JUPYTER

The JupyterHub runs on Ubuntu 22.04 via an AWS EC2 instance. The following packages were installed with different package managers:

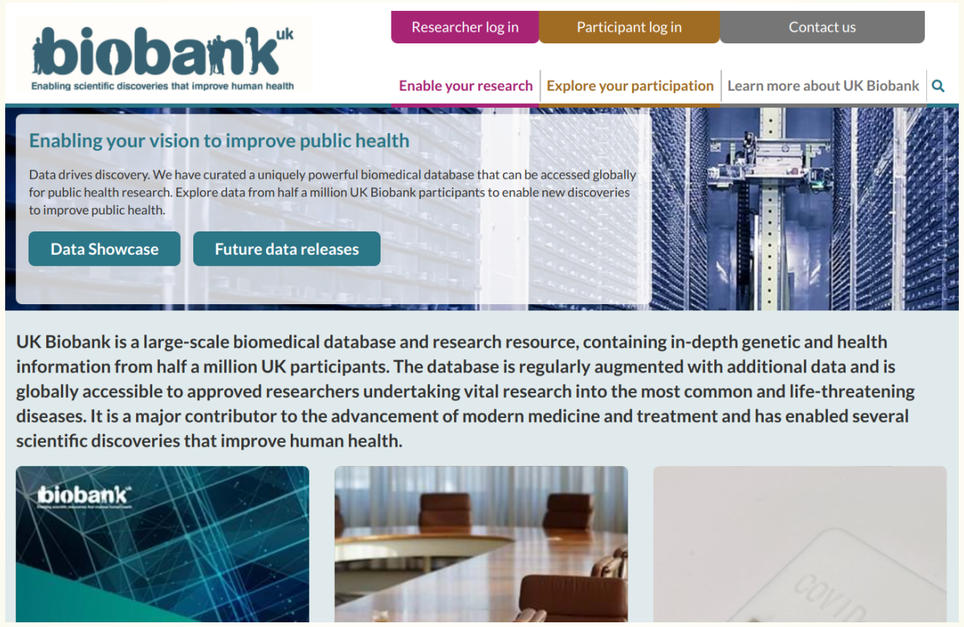
- apt: Git, git-annex, tree, tig, zsh, singularity
- pip: datalad, datalad-next, datalad-container, datalad-osf, black

Instructions to set up and configure your own JupyterHub are publicly available at psychoinformatics-de.github.io/rdm-course/for_instructors

OUTLOOK

FAIRLY BIG: SCALING UP

Objective: Process the UK Biobank (imaging data)

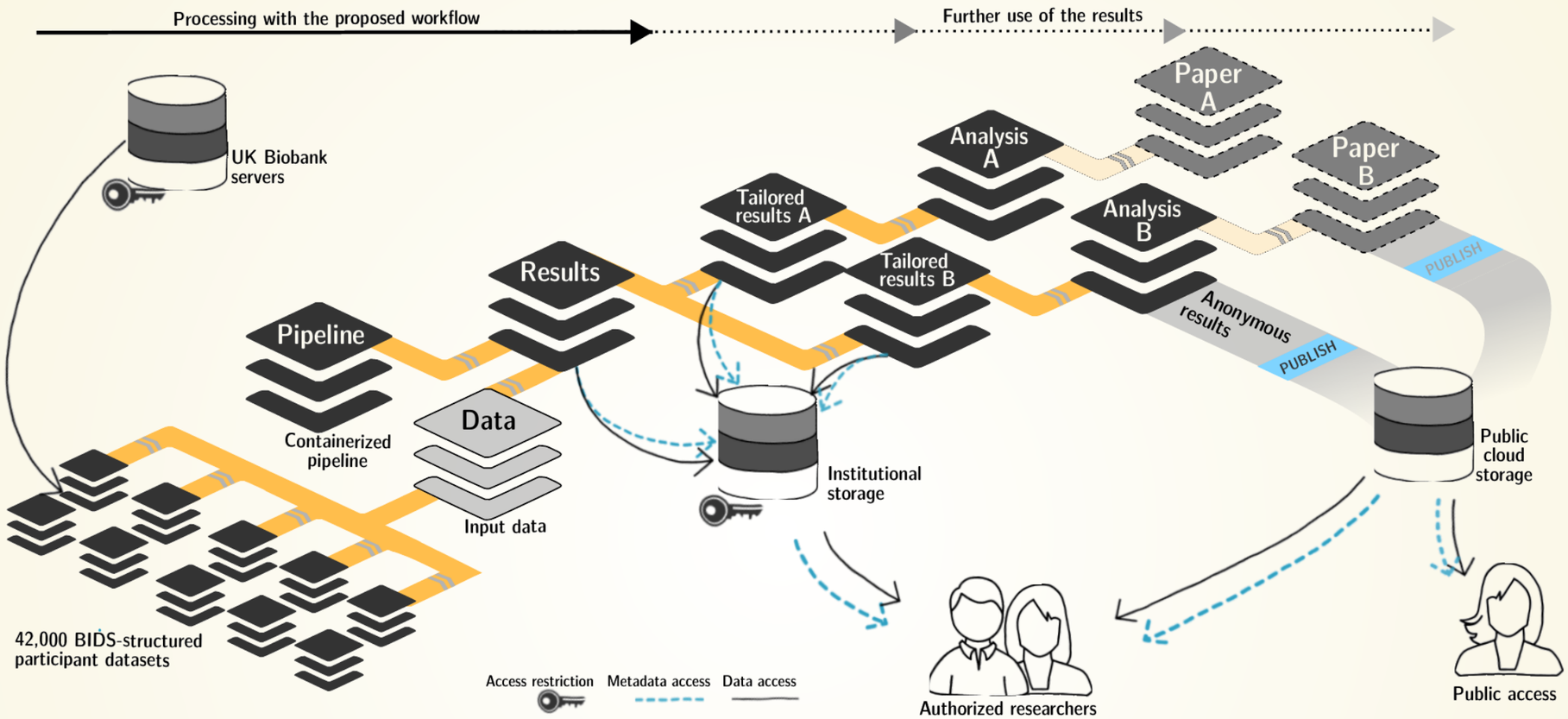


- 76 TB in 43 million files in total
- 42,715 participants contributed personal health data
- Strict DUA
- Custom binary-only downloader
- Most data records offered as (unversioned) ZIP files

CHALLENGES

- Process data such that
  - Results are computationally reproducible (without the original compute infrastructure)
  - There is complete linkage from results to an individual data record download
  - It scales with the amount of available compute resources
- Data processing pipeline
  - Compiled MATLAB blob
  - 1h processing time per image, with 41k images to process
  - 1.2 M output files (30 output files per input file)
  - 1.2 TB total size of outputs

FAIRLY BIG SETUP

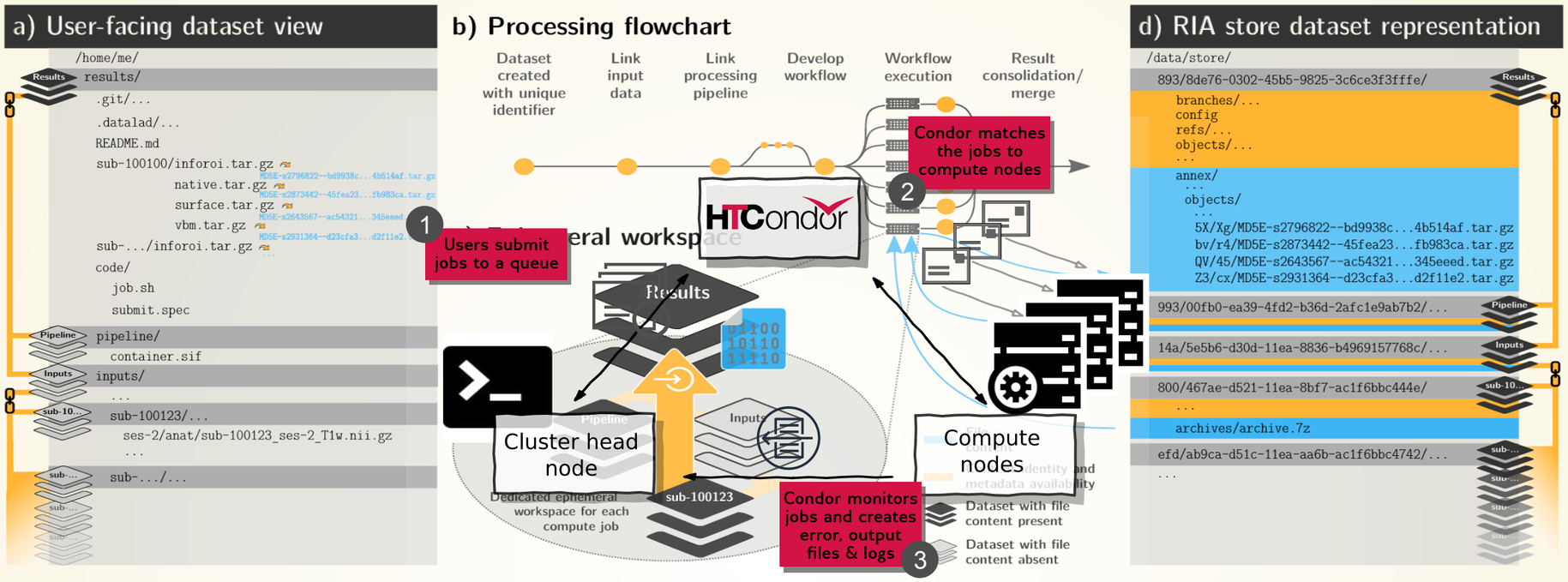


Exhaustive tracking

- datalad-ukbiobank extension downloads, transforms & track the evolution of the complete data release in DataLad datasets
- Native and BIDSified data layout (at no additional disk space usage)
- Structured in 42k individual datasets, combined to one superdataset
- Containerized pipeline in a software container
- Link input data & computational pipeline as dependencies

Wagner, Waite, Wierzba et al. (2021). FAIRly big: A framework for computationally reproducible processing of large-scale data.

FAIRLY BIG WORKFLOW

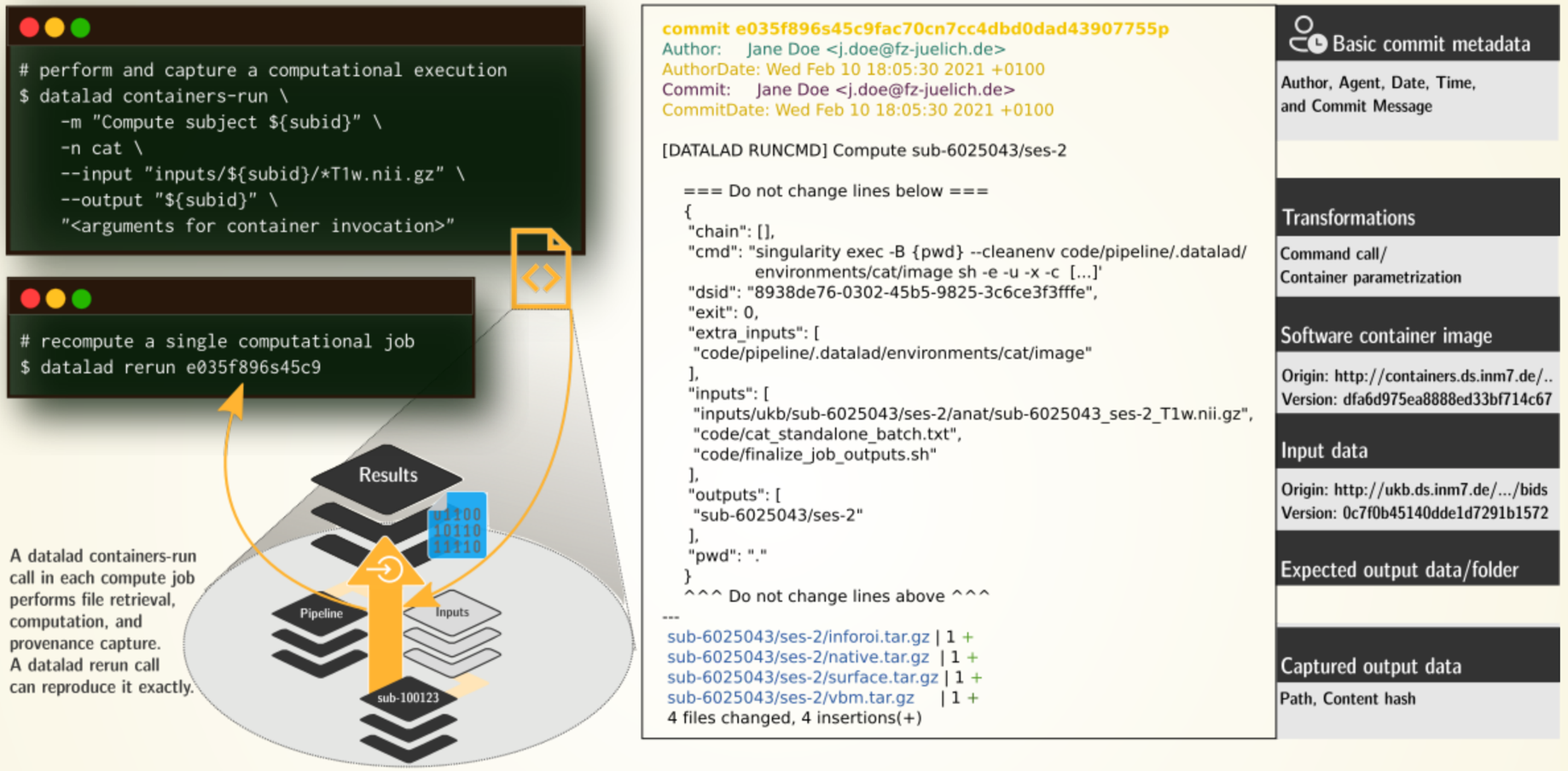


portability

- Parallel processing: 1 job = 1 subject (number of concurrent jobs capped at the capacity of the compute cluster)
- Each job is computed in a ephemeral (short-lived) dataset clone, results are pushed back: Ensure exhaustive tracking & portability during computation
- Content-agnostic persistent (encrypted) storage (minimizing storage and inodes)
- Common data representation in secure environments

Wagner, Waite, Wierzba et al. (2021). FAIRly big: A framework for computationally reproducible processing of large-scale data.

FAIRLY BIG PROVENANCE CAPTURE



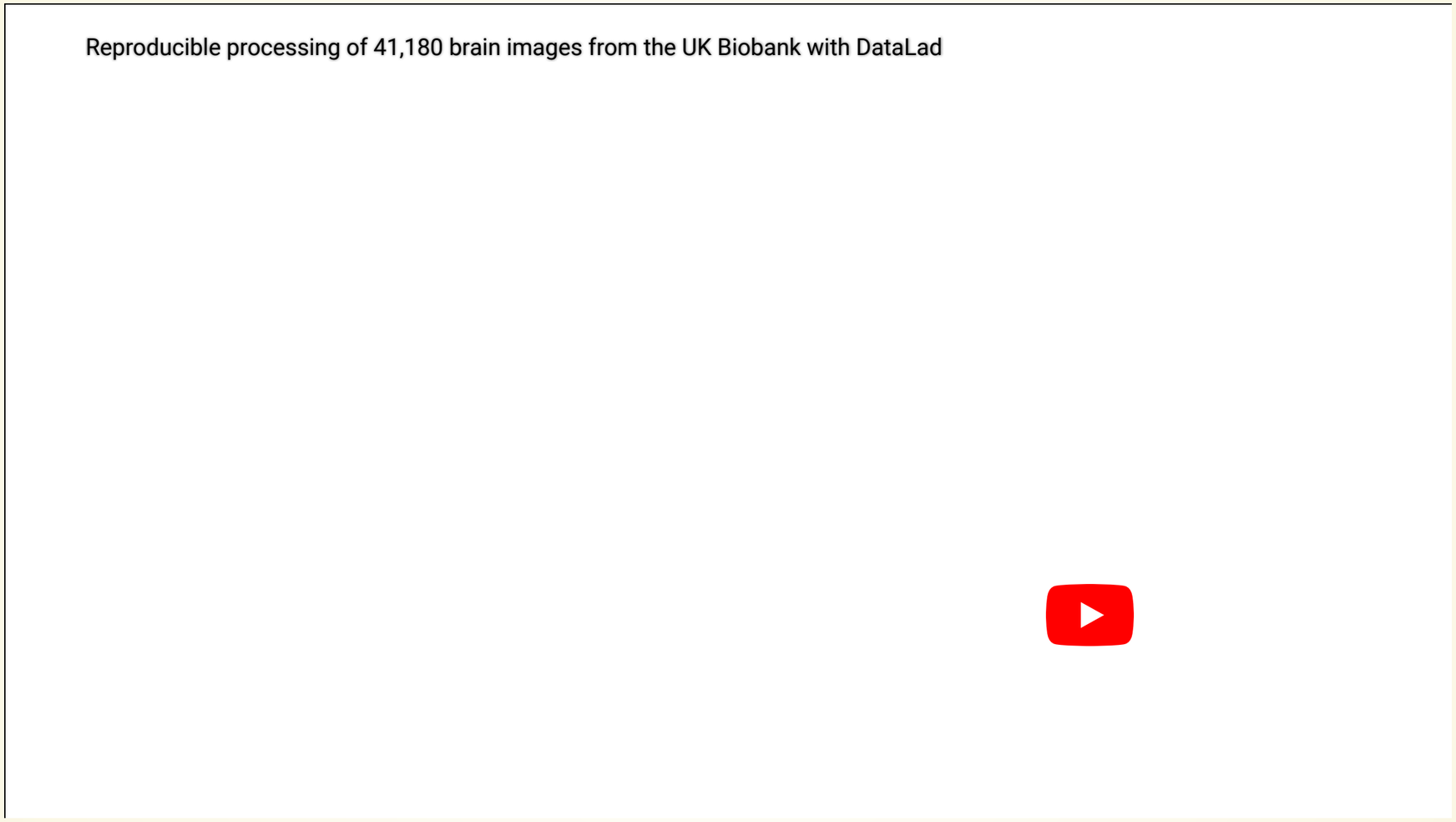
Provenance

- Every single pipeline execution is tracked
- Execution in ephemeral workspaces ensures results individually reproducible without HPC access

Wagner, Waite, Wierzba et al. (2021). FAIRly big: A framework for computationally reproducible processing of large-scale data.

FAIRLY BIG MOVIE

Reproducible processing of 41,180 brain images from the UK Biobank with DataLad



- Two computations on clusters of different scale (small cluster, supercomputer). Full video: https://youtube.com/datalad
- Two full (re-)computations, programmatically comparable, verifiable, reproducible -- on any system with data access