DATA AND REPRODUCIBILITY MANAGEMENT WITH DATALAD

Adina Wagner
mas.to/@adswa

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

Slides: DOI 10.5281/zenodo.7627723 (Scan the QR code)

LOGISTICS

- Collaborative, public notes, networking, & anonymous questions at etherpad.wikimedia.org/p/love-your-data-datalad

- We are using a JupyterHub at datalad-hub.inm7.de. Your username is the email you registered with e.g., a.wagner@fz-juelich.de → a.wagner
  You can log in with a password of your choice.

- Format:
  - Mostly hands-on: Watch me live-code, and try out the software yourself in the browser. Conceptual wrap-up at the end.
  - Ask questions any time (But please mute yourself when you don't speak & make use of the "Raise hand" feature)
  - Quick ☕-break after ~1 hour

FURTHER RESOURCES AND STAY IN TOUCH

If you have questions after the workshop...

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 (Neuro-) 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
- In an overview of most tutorials, talks, videos at github.com/datalad/tutorials

ACKNOWLEDGEMENTS

Funders

NSF 1429999

BMBF 01GQ1411

JÜLICH Forschungszentrum

DFG

EUROPEAN UNION
European Regional Development Fund

cbbs center for behavioral brain sciences

SACHSEN-ANHALT
Ministerium für Wissenschaft und Wirtschaft

DataLad software & ecosystem

- Psychoinformatics Lab, Research center Jülich
- Center for Open Neuroscience, Dartmouth College
- Joey Hess (git-annex)
- *>100 additional contributors*

Collaborators

Human Brain Project

CONP PCNO

OpenNEURO

eBRAIN Health

MOTOR SFB 1451

brainlife.io

cbrain

VirtualBrainCloud

LET'S GET TO KNOW EACH OTHER

Please use your phone to scan to QR code, or open the link in a new browser window

http://etc.ch/aXaL

Direct Poll

A COMMON USECASE

- Alice is a PhD student in a research team.
- She works on a fairly typical research project: Data collection & processing.
- First sample → final result = complex process

HOW DOES ALICE GO ABOUT HER DAILY JOB?

A COMMON USECASE

- In her project, Alice likes to have an automated record of:
  - when a given file was last changed
  - where it came from
  - what input files were used to generate a given output
  - why some things were done.

- Even if she doesn't share her work, this is essential for her future self
- Her project is exploratory: Frequent changes to her analysis scripts
- She enjoys the comfort of being able to return to a previously recorded state

THIS IS: *LOCAL VERSION CONTROL*

A COMMON USECASE

- Alice's work is not confined to a single computer:
  - Laptop / desktop / remote server / dedicated back-up
  - Alice wants to automatically & efficiently synchronize

- Parts of the data are collected or analyzed by colleagues. This requires:
  - distributed synchronization with centralized storage
  - preservation of origin & authorship of changes
  - effective combination of simultaneous contributions

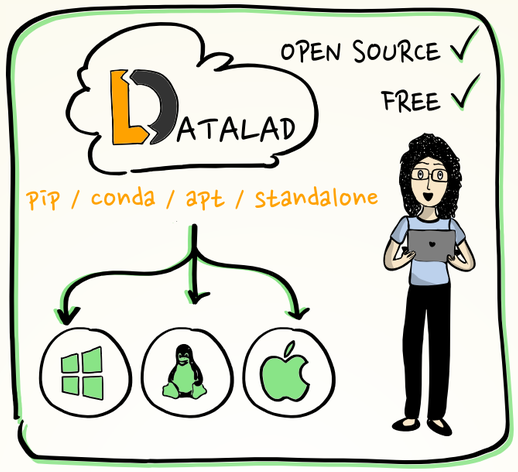
THIS IS: *DISTRIBUTED VERSION CONTROL*

A COMMON USECASE

- Alice applies local version control for her own work, and reproducibly records it
- She also applies distributed version control when working with colleagues and collaborators
- She often needs to work on a subset of data at any given time:
  - all files are kept on a server
  - a few files are rotated into and out of her laptop
- Alice wants to publish the data at project's end:
  - raw data / outputs / both
  - completely or selectively

THIS IS: *DATA MANAGEMENT (WITH DATALAD 😀)*

DATALAD



- Domain-agnostic **command-line tool** (+ **graphical user interface**), built on top of Git & Git-annex
- Major features:

  Version-controlling arbitrarily large content

  Version control data & software alongside to code!

  Transport mechanisms for sharing & obtaining data

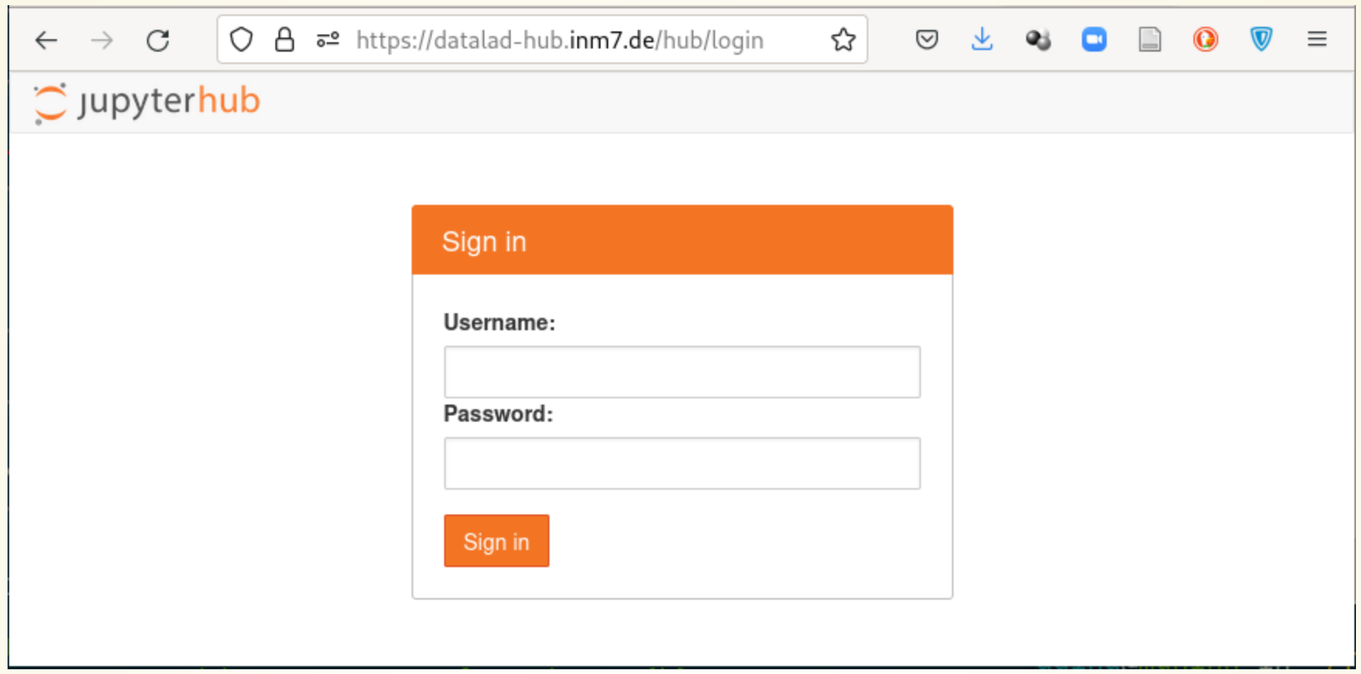
  Consume & collaborate on data (analyses) like software

  (Computationally) reproducible data analysis

  Track and share provenance of all digital objects

  (... and much more)

LET'S TRY IT OUT



datalad-hub.inm7.de

username:

email without `@domain` (a.wagner@fz-juelich.de -> a.wagner)
(must be the email you registered with for this workshop)

password:

Set at first login, at least 8 characters

Important! The Hub is a shared resource. Don't fill it up :)

GIT IDENTITY SETUP

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

Configure DataLad to use latest features:

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

copy

USING DATALAD IN A TERMINAL

Check the installed version:

```
datalad --version
```

copy

For help on using DataLad from the command line:

```
datalad --help
```

copy

```
The help may be displayed in a pager - exit it by pressing "q"
```

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

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')
```

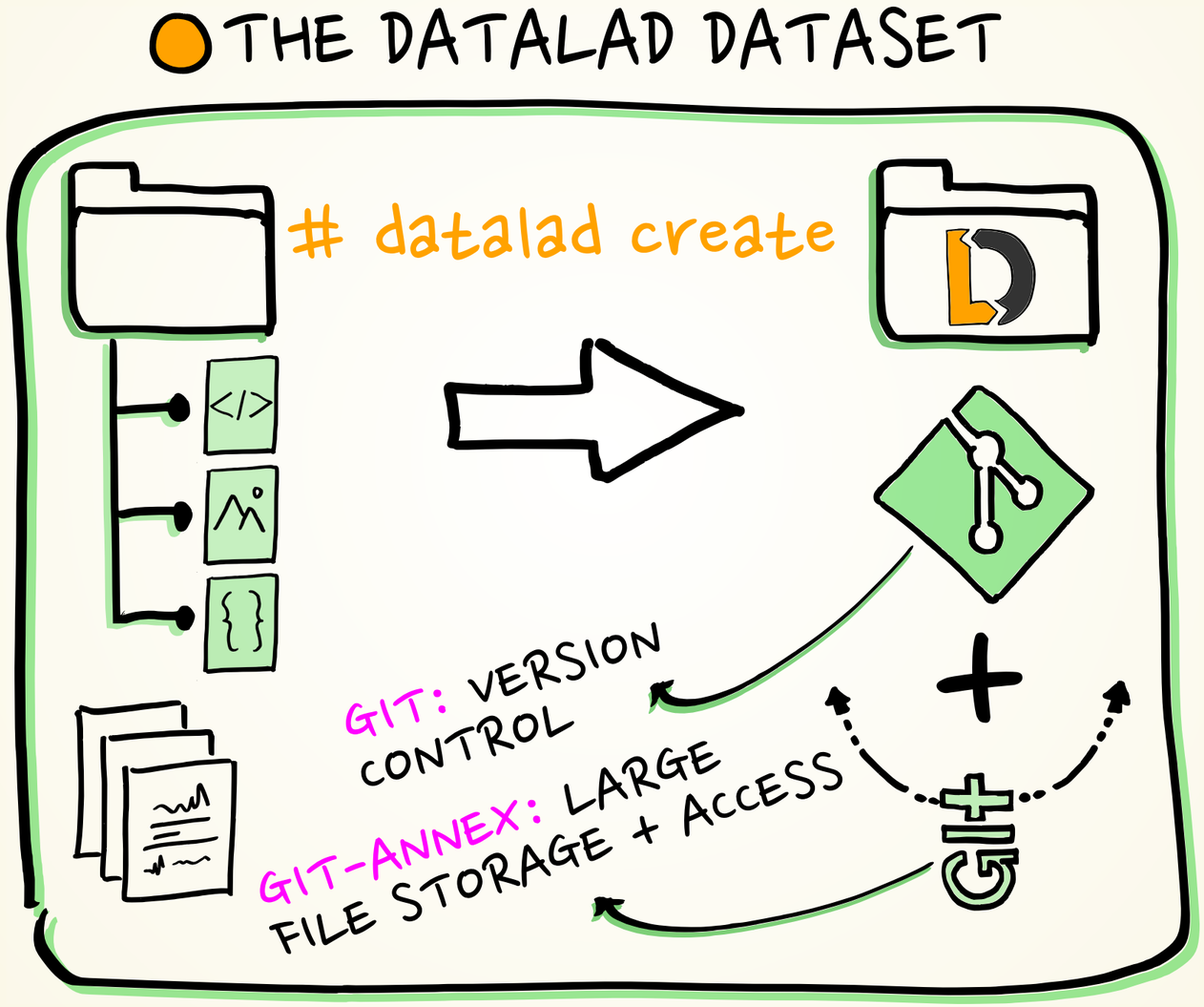
copy

Exit the Python environment:

```
exit
```

copy

DATALAD DATASETS...



...DATALAD DATASETS

Create a dataset (here, with the `yoda` configuration, which adds a helpful structure and configuration for data analyses):

```
datalad create -c yoda my-analysis
```

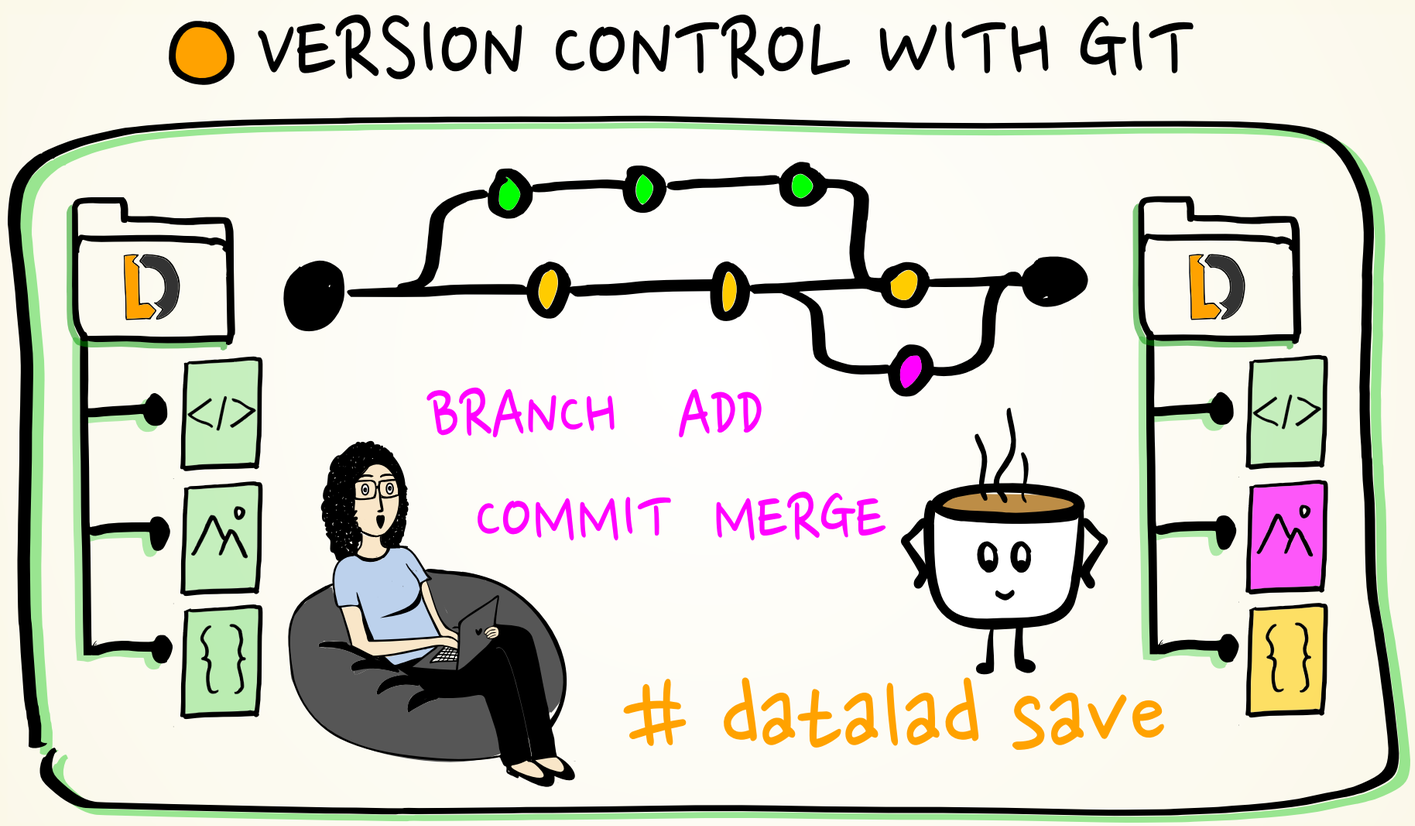
Let's have a look inside. Navigate using `cd` (change directory):

```
cd my-analysis
```

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

```
ls -la .
```

VERSION CONTROL...



...VERSION CONTROL

The yoda-configuration added a README placeholder in the dataset. Let's add Markdown text (a project title) to it:

```
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 "Add project title into the README"
```
copy

Further modifications:

```
echo "Contains a small data analysis for my project" >> 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)
```

Convenience functions make downloads easier. Let's add code for a data analysis from an external source:

```
datalad download-url -m "Add an analysis script" \
  -O code/classification_analysis.py \
  https://raw.githubusercontent.com/datalad-handbook/resources/master/classification_analysis.py
```

copy

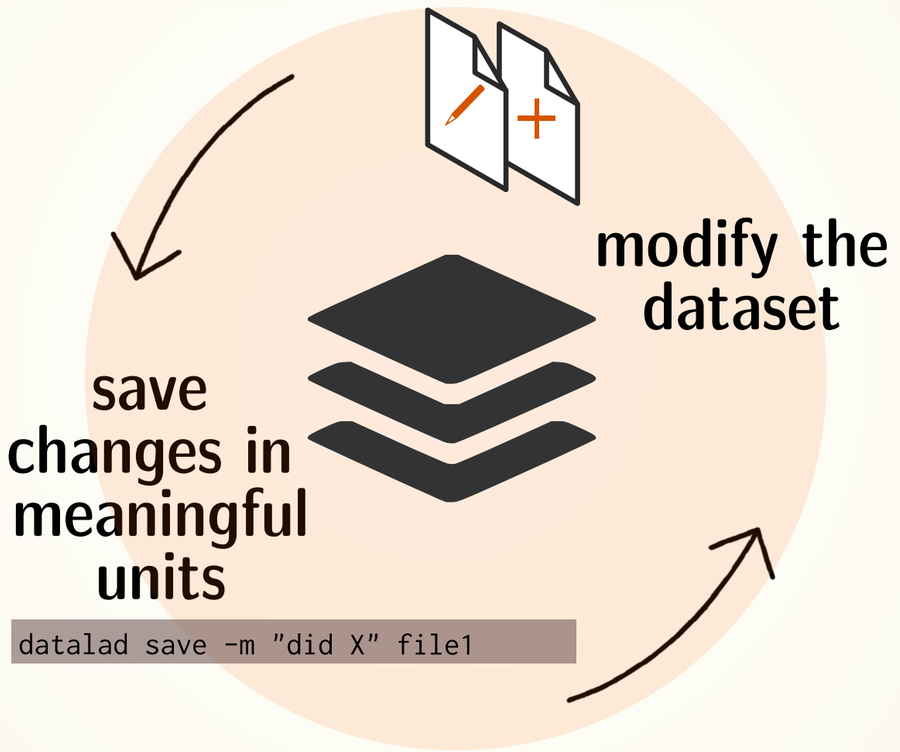
Check out the file's history:

```
git log code/classification_analysis.py
```

copy

LOCAL VERSION CONTROL

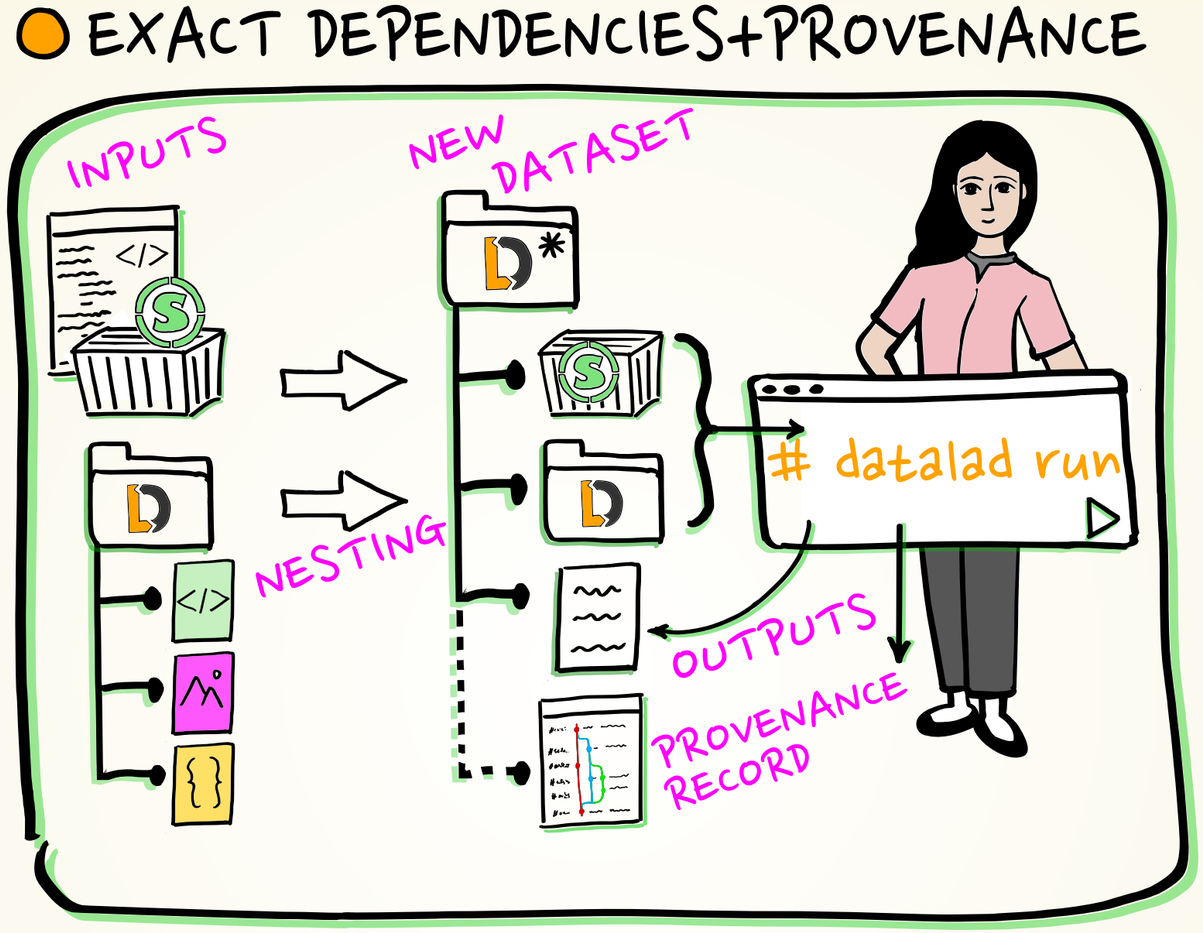
Procedurally, version control is easy with DataLad!



Advice:

- Save meaningful units of change
- Attach helpful commit messages

COMPUTATIONALLY REPRODUCIBLE EXECUTION I...



- which script/pipeline version
- was run on which version of the data
- to produce which version of the results?

... COMPUTATIONALLY REPRODUCIBLE EXECUTION I

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

```
black code/classification_analysis.py
```

copy

Version control makes changes transparent:

```
git diff
```

copy

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

```
git restore code/classification_analysis.py
```

copy

... and record precisely what we did

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

copy

let's take a look:

```
git show
```

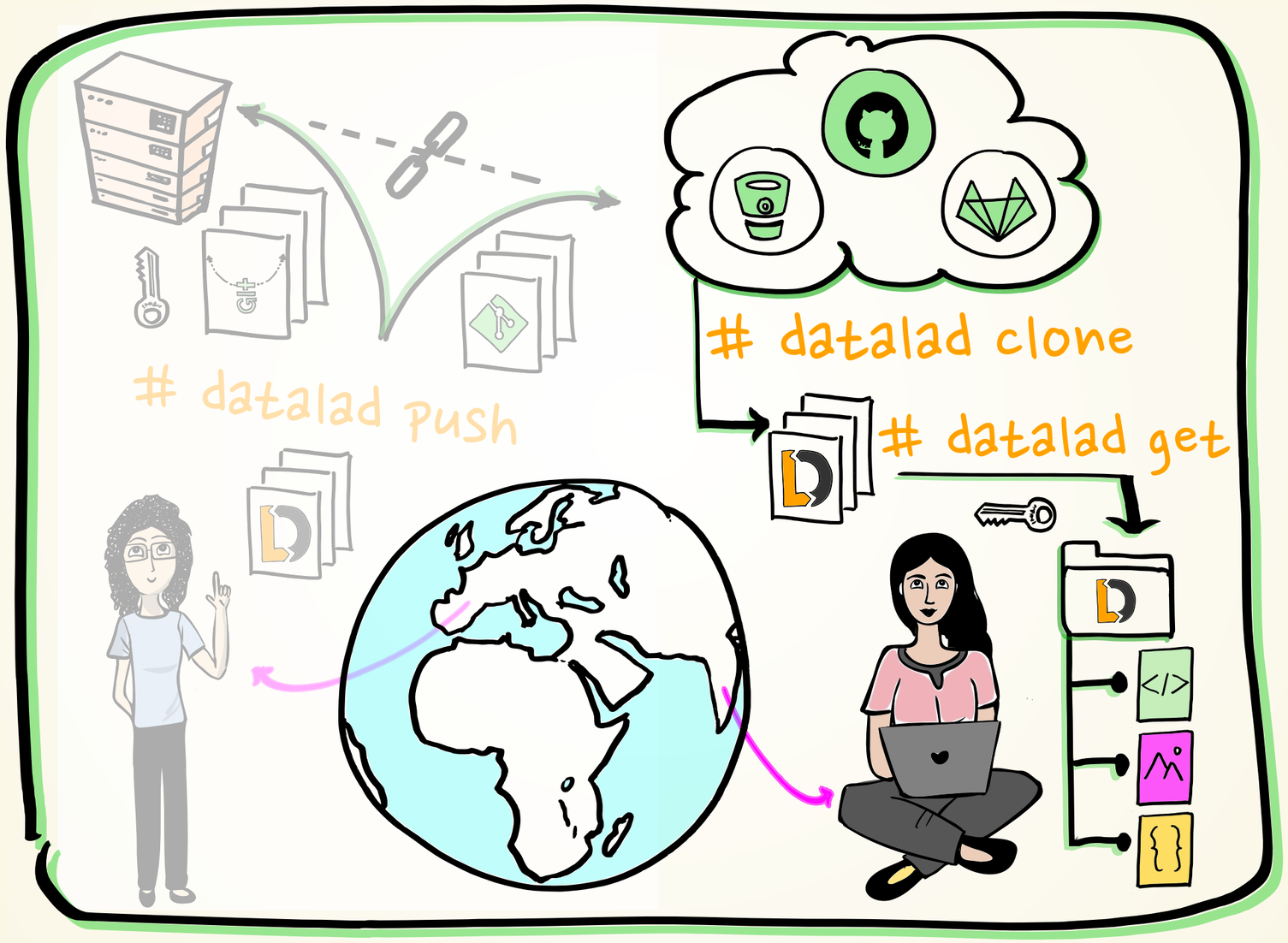
copy

... and repeat!

```
datalad rerun
```

copy

DATA CONSUMPTION & TRANSPORT...



...DATA CONSUMPTION & TRANSPORT...

You can install a dataset from remote URL (or local path) using `clone`. Either as a stand-alone entity:

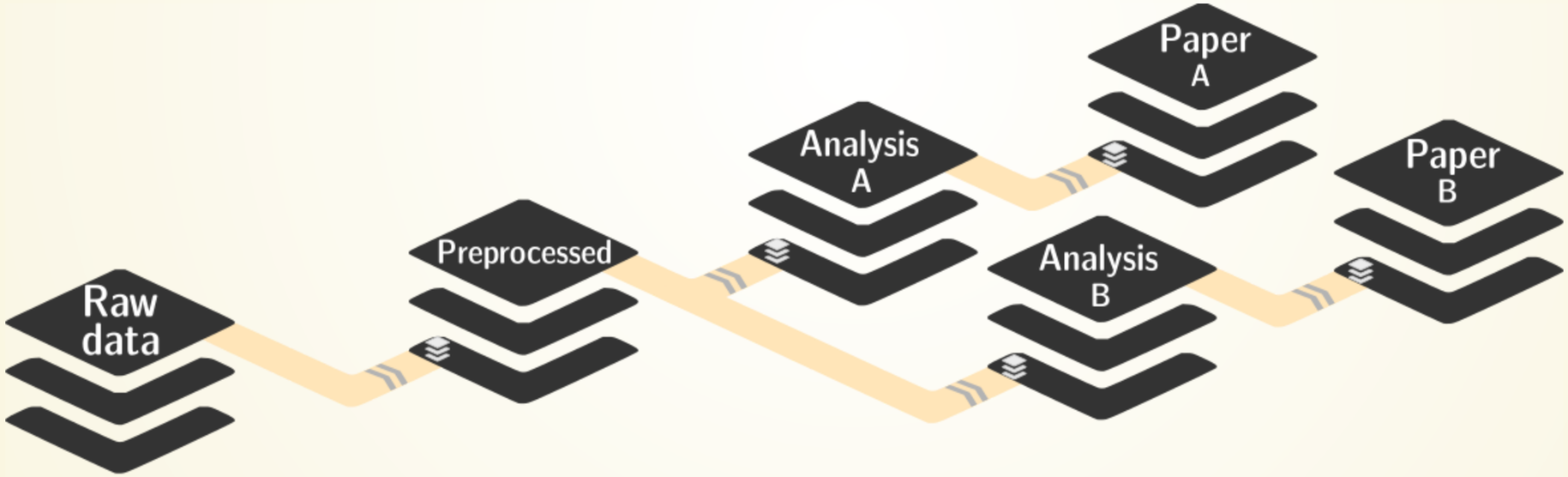
```
# just an example:
datalad clone \
https://github.com/psychoinformatics-de/studyforrest-data-phase2.git
```

copy

Or as linked dataset, nested in another dataset in a superdataset-subdataset hierarchy:

```
# just an example:
datalad clone -d . \
https://github.com/psychoinformatics-de/studyforrest-data-phase2.git
```

copy



- Helps with scaling (see e.g. the Human Connectome Project dataset)
- Version control tools struggle with >100k files
- Modular units improves intuitive structure and reuse potential
- Versioned linkage of inputs for reproducibility

...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 --dataset . \
   https://github.com/datalad-handbook/iris_data.git \
   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

Let's check try to print the file contents into the terminal (`cat`):

```
cat iris.csv
```

copy

...DATA CONSUMPTION & TRANSPORT

We can retrieve actual file content with `get`:

```
datalad get iris.csv
```

copy

If we don't need a file locally anymore, we can `drop` its content:

```
datalad drop iris.csv
```

copy

No need to store all files locally, or archive results with Giga/Terra-Bytes of source data:

```
dl.get('input/sub-01')
[really complex analysis]
dl.drop('input/sub-01')
```

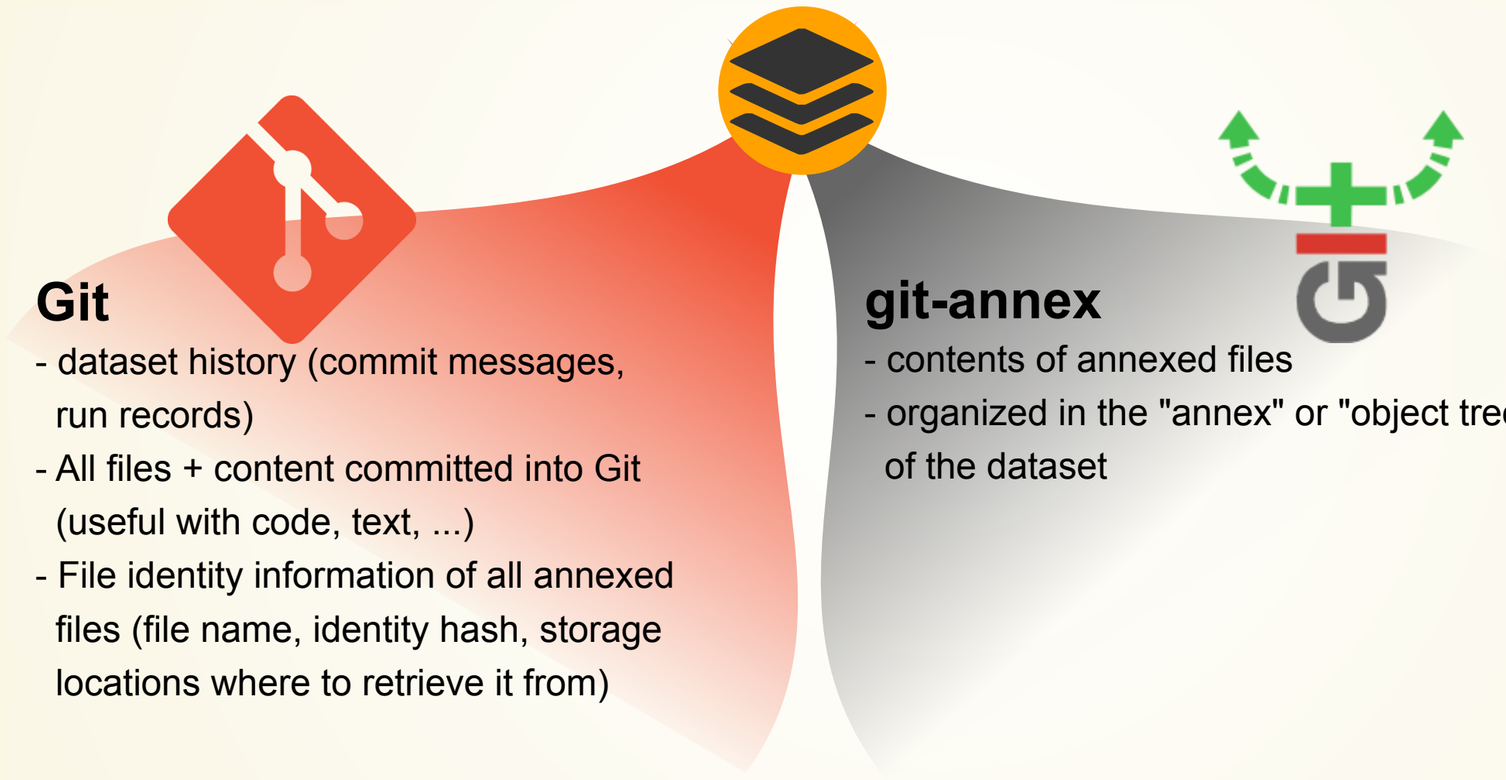
copy

If data is published anywhere, your data analysis can carry an actionable link to it, with barely any space requirements.

GIT VERSUS GIT-ANNEX

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

By default, everything is annexed, i.e., stored in a dataset annex by git-annex



- With annexed data, only content identity (hash) and location information is put into Git, rather than file content. The annex, and transport to and from it is managed with **git-annex**

GIT VERSUS GIT-ANNEX

Configurations (e.g., YODA), custom rules, or command parametrization determines if a file is annexed

Storing files in Git or git-annex has distinct advantages:

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

YODA configures the contents of the `code/` directory and the dataset descriptions (e.g., README files) to be in Git. There are many other configurations, and you can also write your own.

...COMPUTATIONALLY REPRODUCIBLE EXECUTION...

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

```
cd ..
python code/classification_analysis.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

...COMPUTATIONALLY REPRODUCIBLE EXECUTION...

- The `datalad run` can run any command in a way that links the command or script to the results it produces and the data it was computed from
- The `datalad rerun` can take this recorded provenance and recompute the command
- The `datalad containers-run` (from the extension "datalad-container") can capture software provenance in the form of software containers in addition to the provenance that datalad run captures

...COMPUTATIONALLY REPRODUCIBLE EXECUTION

With the `datalad-container` extension, we can add software containers to datasets and work with them. Let's add a software container with Python software to run the script

```
datalad containers-add python-env --url shub://adswa/resources:2
```

inspect the list of registered containers:

```
datalad containers-list
```

Now, let's try out the `containers-run` command:

```
datalad containers-run -m "run classification analysis in python environment" \
  --container-name python-env \
  --input "input/iris.csv" \
  --output "pairwise_relationships.png" \
  --output "prediction_report.csv" \
  "python3 code/classification_analysis.py {inputs} {outputs}"
```

What changed after the `containers-run` command has completed?
We can use `datalad diff` (based on `git diff`):

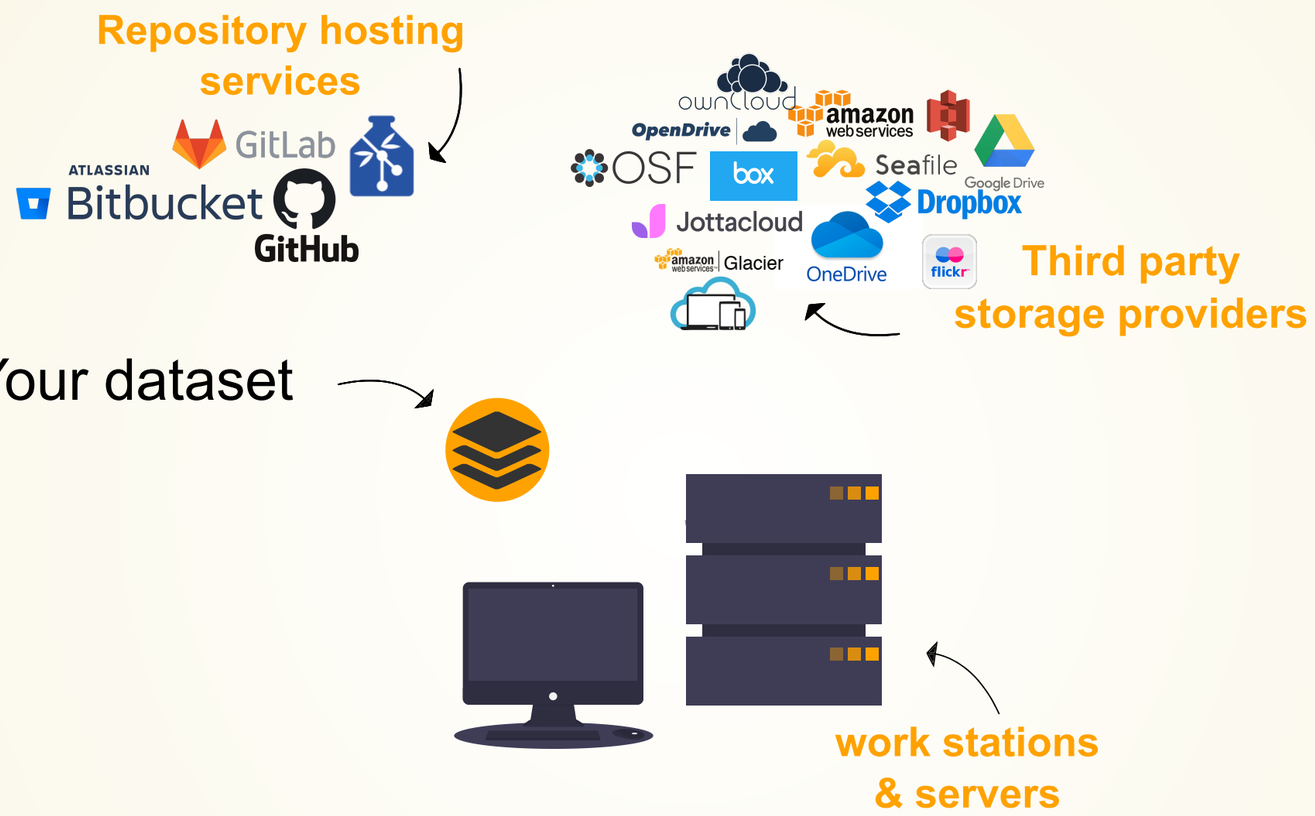
```
datalad diff -f HEAD~1
```

We see that some files were added to the dataset!
And we have a complete provenance record as part of the git history:

```
git log -n 1
```

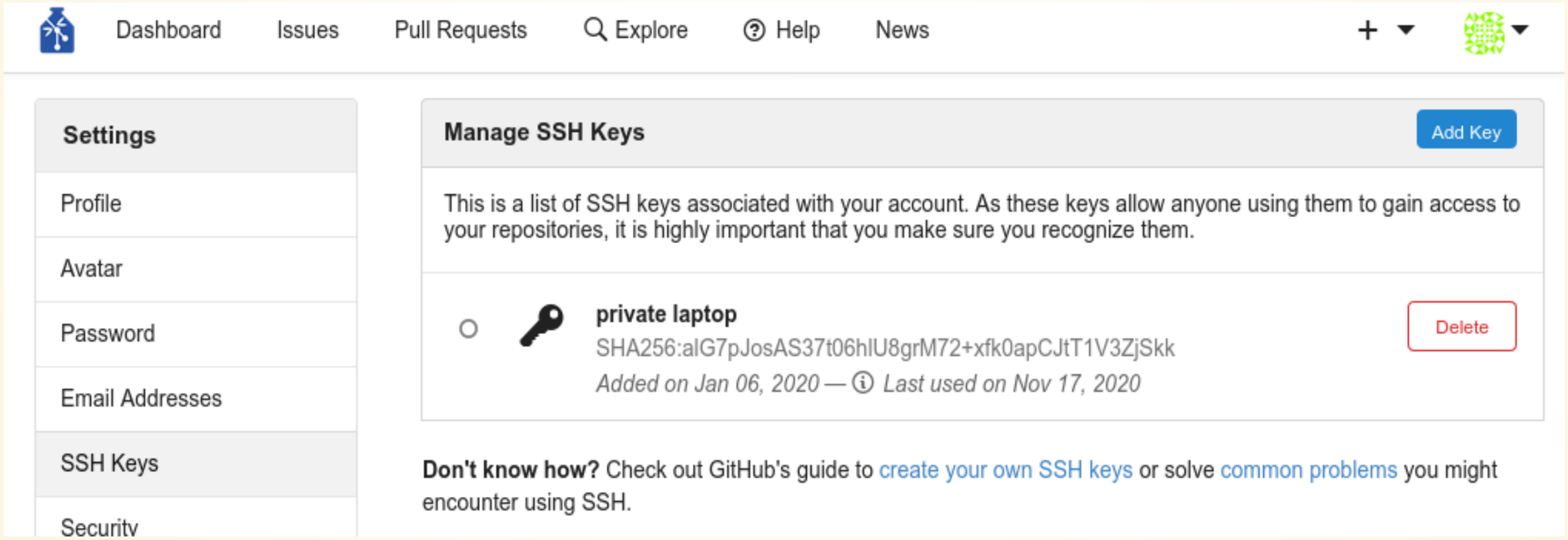
PUBLISHING DATASETS...

We will use GIN: gin.g-node.org:



PUBLISHING DATASETS...

- Create a GIN user account and log in: gin.g-node.org/user/sign_up
- Create and upload an SSH key to GIN



- Publish your dataset!

...PUBLISHING DATASETS

DataLad has convenience functions to create `sibling`-repositories on various infrastructure and third party services (GitHub, GitLab, OSF, WebDAV-based services, DataVerse, ...) , to which data can then be published with `push`.

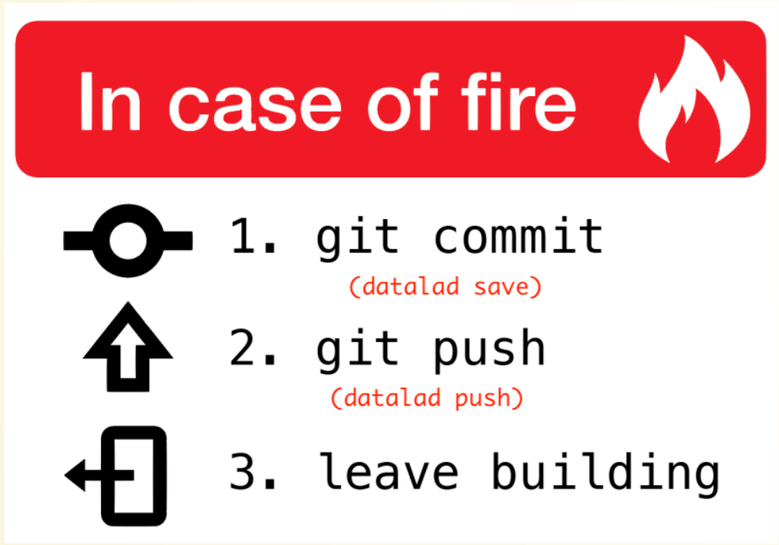
```
datalad create-sibling-gin example-analysis --access-protocol ssh
```
copy

You can verify the dataset's siblings with the `siblings` command:

```
datalad siblings
```
copy

And we can push our complete dataset (Git repository and annex) to GIN:

```
datalad push --to gin
```
copy



USING PUBLISHED DATA...

Let's see how the analysis feels like to others:

```
cd ../
datalad clone \
   https://gin.g-node.org/adswa/example-analysis \
   myclone
```
copy

```
cd myclone
```
copy

Get results:

```
datalad get prediction_report.csv
```
copy

```
datalad drop prediction_report.csv
```
copy

Or recompute results:

```
datalad rerun
```
copy

HOW DOES THIS RELATE TO REPRODUCIBILITY?

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...)

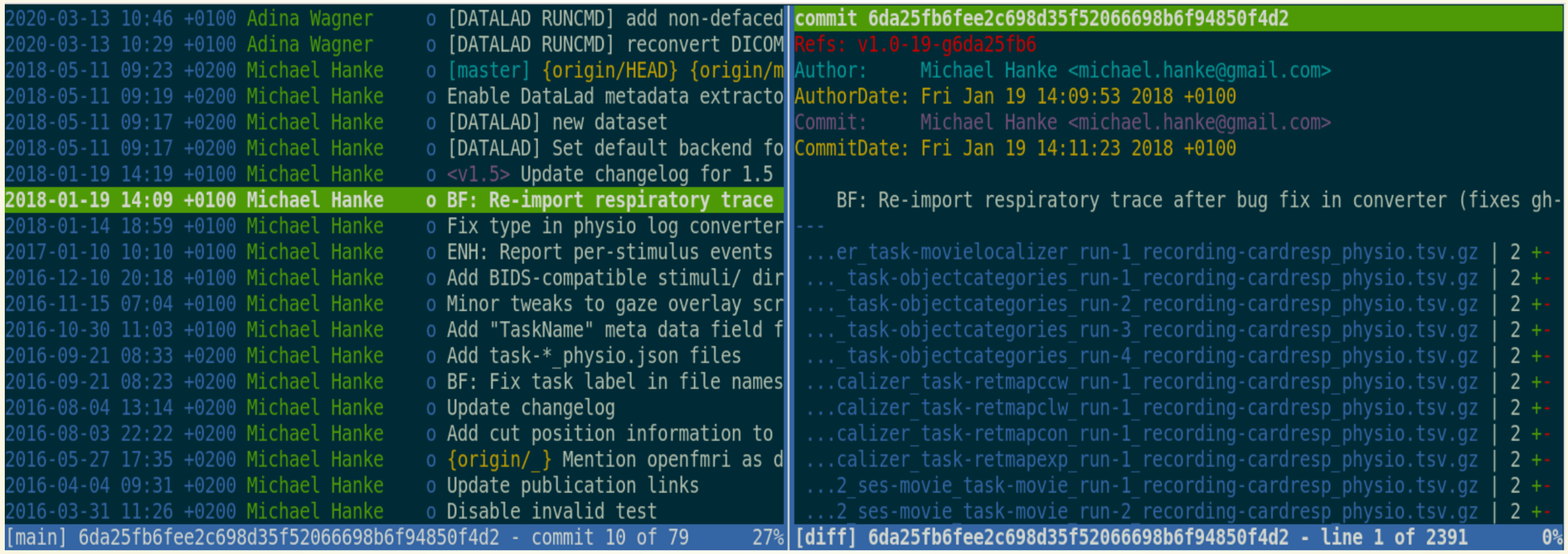
Image credit: Piled higher and Deeper 1323

EXHAUSTIVE TRACKING

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

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.



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?

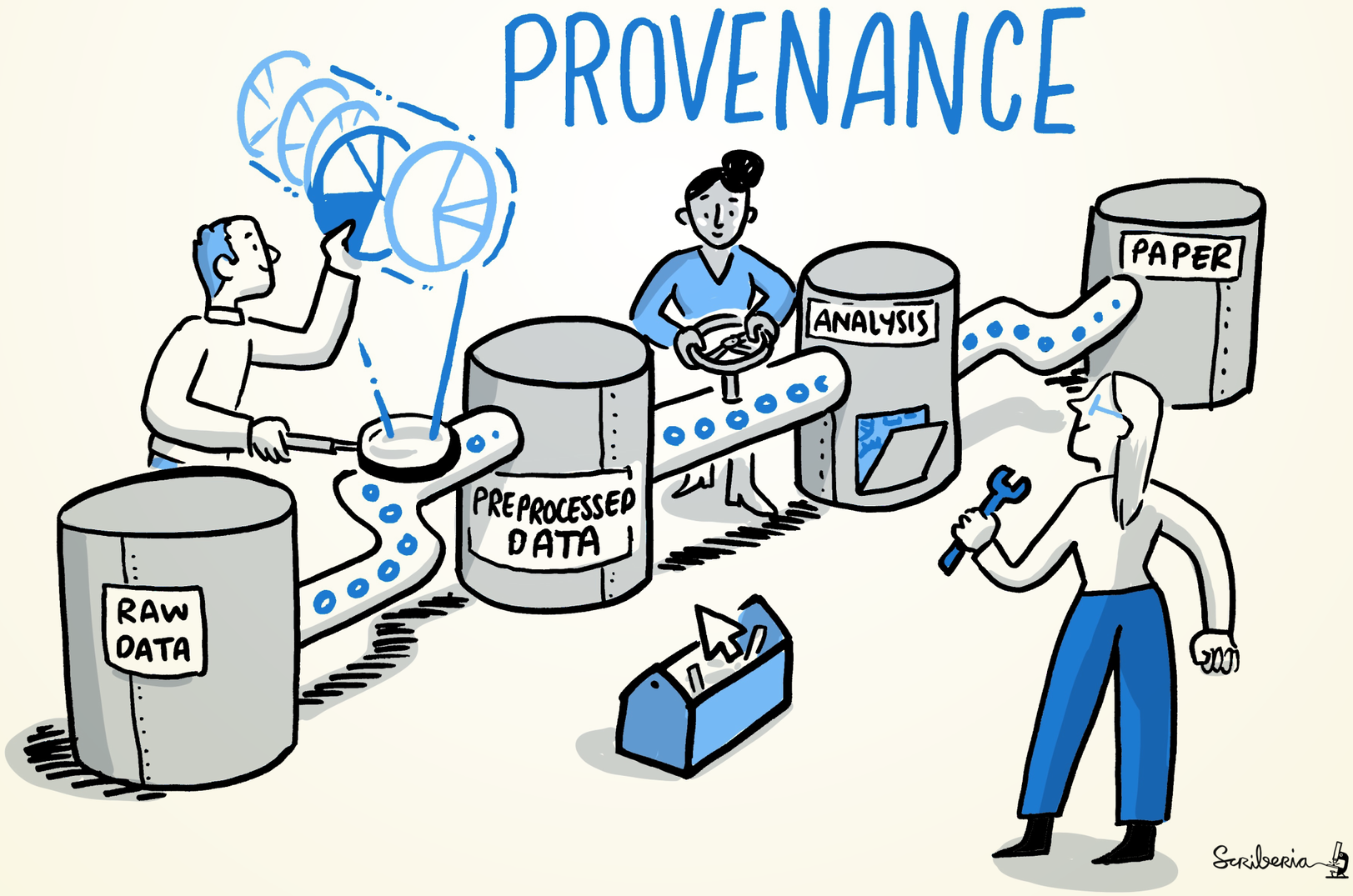
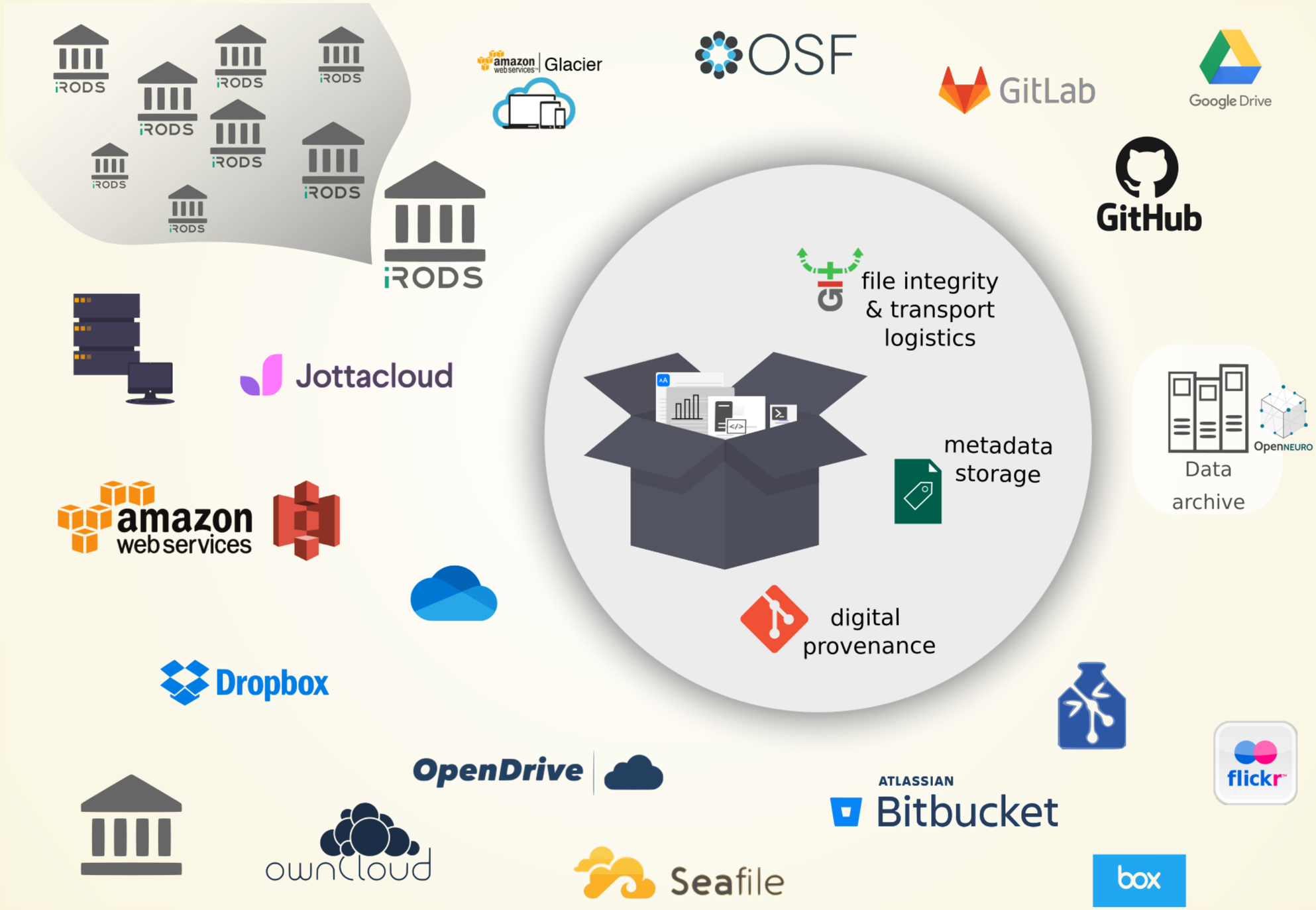


Image credit: Scriberia and The Turing Way (CC-BY)

DATA TRANSPORT: SECURITY AND RELIABILITY - FOR DATA

Decentral version control for data integrates with a variety of services to let you store data in different places - creating a resilient network for data



"In defense of decentralized Research Data Management", doi.org/10.1515/nf-2020-0037

ULTIMATE GOAL: REUSABILITY

Teamscience on more than code:


```
2019-03-08 12:38 +0100 Richard Andersson M─┐ [master] {origin/master} {origin/HEAD} Merge pull request #3 from AdinaWagner/datafix
2019-03-08 11:35 +0100 Adina Wagner      │ o ENH/FIX: relabel erroneous saccades to fixations, closes #2.
2018-12-05 15:27 +0100 Richard Andersson o─┘ Uploaded a folder consting only of the data used in the original article
2017-08-22 19:40 +0200 Richard Andersson o Code added
2016-12-14 10:35 +0100 richardandersson  o Added currently shared data and stimuli.
2016-12-14 10:29 +0100 richardandersson  o Deleted -- to be reuploaded
2016-12-14 10:08 +0100 Richard Andersson o Add files via upload
2016-12-14 10:07 +0100 Richard Andersson o Add files via upload
2016-12-14 10:04 +0100 Richard Andersson I Initial commit
```

Fixed data

Original data

THE YODA PRINCIPLES

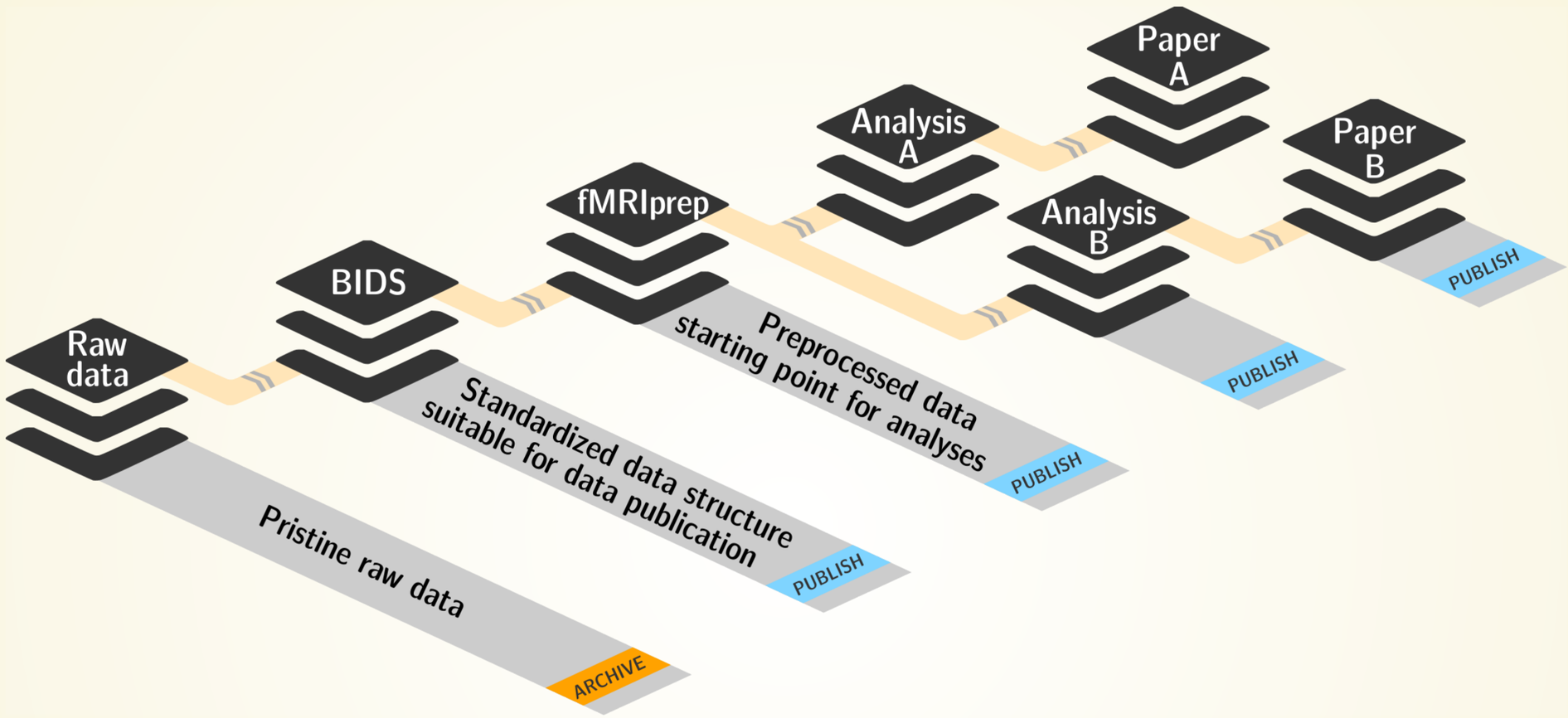
DATALAD DATASETS FOR DATA ANALYSIS

- A DataLad dataset can have any structure, and use as many or few features of a dataset as required.
- However, for **data analyses** it is beneficial to make use of DataLad features and structure datasets according to the **YODA principles**:

P1: One thing, one dataset
P2: Record where you got it from, and where it is now
P3: Record what you did to it, and with what

Find out more about the YODA principles in the handbook, and more about structuring dataset at psychoinformatics-de.github.io/rdm-course/02-structuring-data

P1: ONE THING, ONE DATASET



- Create **modular** datasets: Whenever a particular collection of files could anyhow be useful in more than one context (e.g. data), put them in their own dataset, and install it as a subdataset.
- Keep everything **structured**: Bundle all components of one analysis into one superdataset, and within this dataset, separate code, data, output, execution environments.
- Keep a dataset **self-contained**, with relative paths in scripts to subdatasets or files. Do not use absolute paths.

WHY MODULARITY?

- 1. Reuse and access management
- 2. Scalability
- 3. Transparency

Original:

```
/dataset
├── sample1
│   └── a001.dat
├── sample2
│   └── a001.dat
...
```

Without modularity, after applied transform (preprocessing, analysis, ...):

```
/dataset
├── sample1
│   ├── ps34t.dat
│   └── a001.dat
├── sample2
│   ├── ps34t.dat
│   └── a001.dat
...
```

Without expert/domain knowledge, no distinction between original and derived data possible.

WHY MODULARITY?

- 3. Transparency

  Original:

```
/raw_dataset
├── sample1
│   └── a001.dat
├── sample2
│   └── a001.dat
...
```

With modularity after applied transform (preprocessing, analysis, ...)

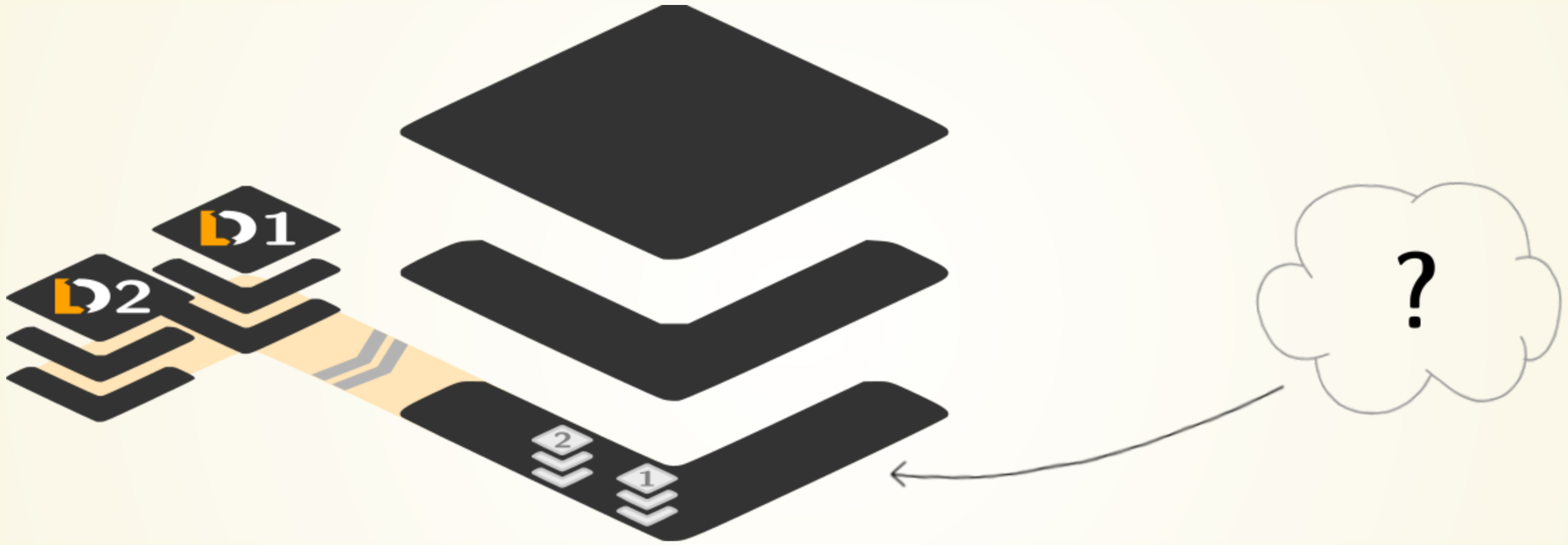
```
/derived_dataset
├── sample1
│   └── ps34t.dat
├── sample2
│   └── ps34t.dat
├── ...
└── inputs
    └── raw
        ├── sample1
        │   └── a001.dat
        ├── sample2
        │   └── a001.dat
        ...
```

Clearer separation of semantics, through use of pristine version of original dataset within a *new*, *additional* dataset holding the outputs.

WHEN TO MODULARIZE?

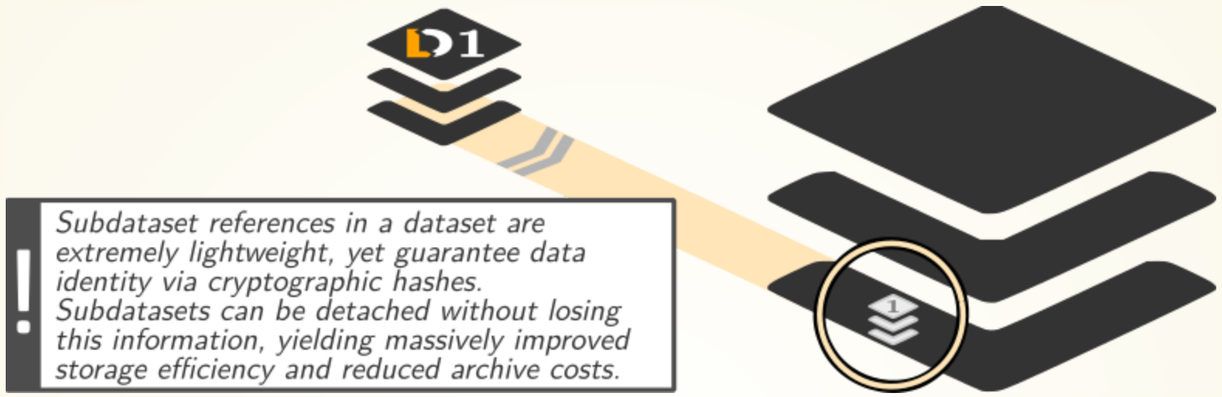
- Target audience is different
  - public vs. private
  - domain specific vs. domain general
- Pace of evolution is different
  - "factual" raw data vs. choices of (pre-)processing
  - completed acquisition vs. ongoing study
- Size impacts I/O and logistics
  - Git can struggle with 1M+ files
  - filesystems (licensing) can struggle with large numbers of inodes
  - More infos: Go Big or Go Home chapter
- Legal/Access constraints
  - personal vs. anonymized data

P2: RECORD WHERE YOU GOT IT FROM, AND WHERE IT IS NOW



- **Link** individual datasets to declare data-dependencies (e.g. as subdatasets).
- **Record data's origin** with appropriate commands, for example to record access URLs for individual files obtained from (unstructured) sources "in the cloud".
- Share and **publish** datasets for collaboration or back-up.

DATASET LINKAGE

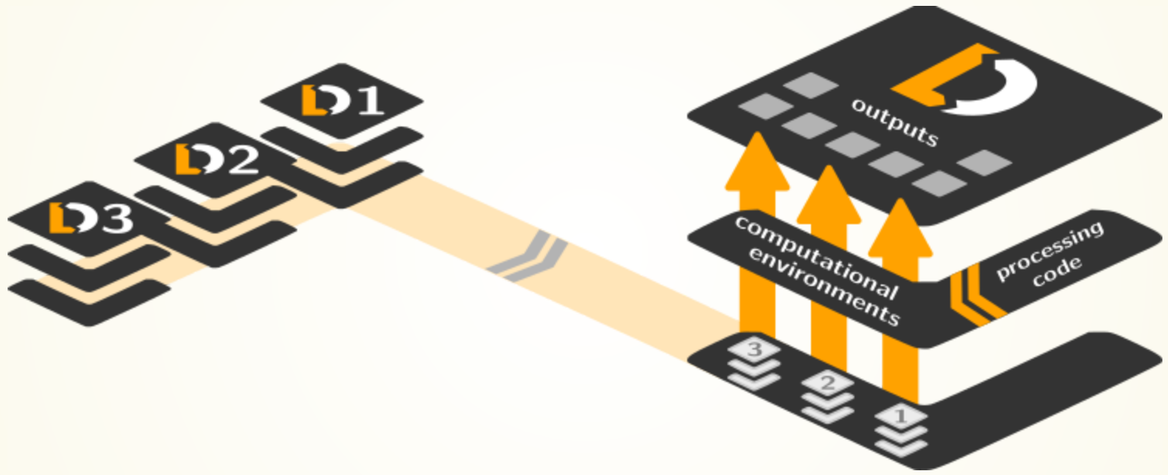



```
$ datalad clone --dataset . http://example.com/ds inputs/rawdata
```
copy

```
$ 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
+       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

Each (sub)dataset is a separately, but jointly version-controlled entity. If none of its data is retrieved, subdatasets are an extremely **lightweight** data dependency and yet **actionable** (**datalad get** retrieves contents on demand)

P3: RECORD WHAT YOU DID TO IT, AND WITH WHAT



- Collect and store **provenance** of all contents of a dataset that you create
- "Which script produced which output?", "From which data?", "In which **software environment**?" ... Record it in an ideally machine-readable way with **datalad (containers-)run**

TAKE HOME MESSAGES

Data deserves version control

It changes and evolves just like code, and exhaustive tracking lays a foundation for reproducibility

Reproducible science relies on good data management

But effort pays off: Increased transparency, better reproducibility, easier accessibility, efficiency through automation and collaboration, streamlined procedures for synchronizing and updating your work, ...

DataLad can help with some things

Have access to more data than you have disk space

Who needs short-term memory when you can have automatic provenance capture?

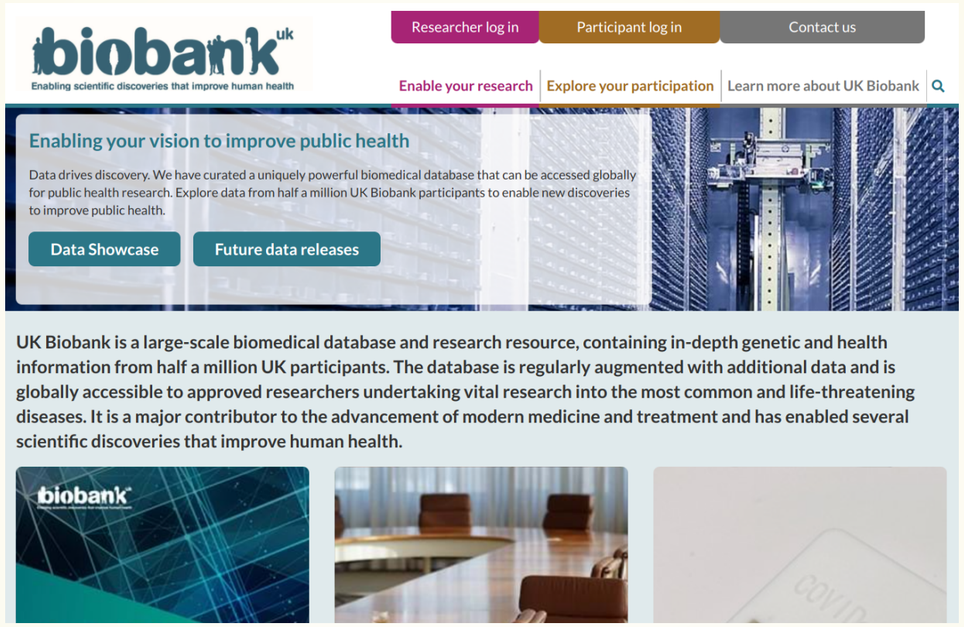
Link versioned data to your analysis at no disk-space cost

...

SCALABILITY

FAIRLY BIG: SCALING UP

Objective: Process the UK Biobank (imaging data)

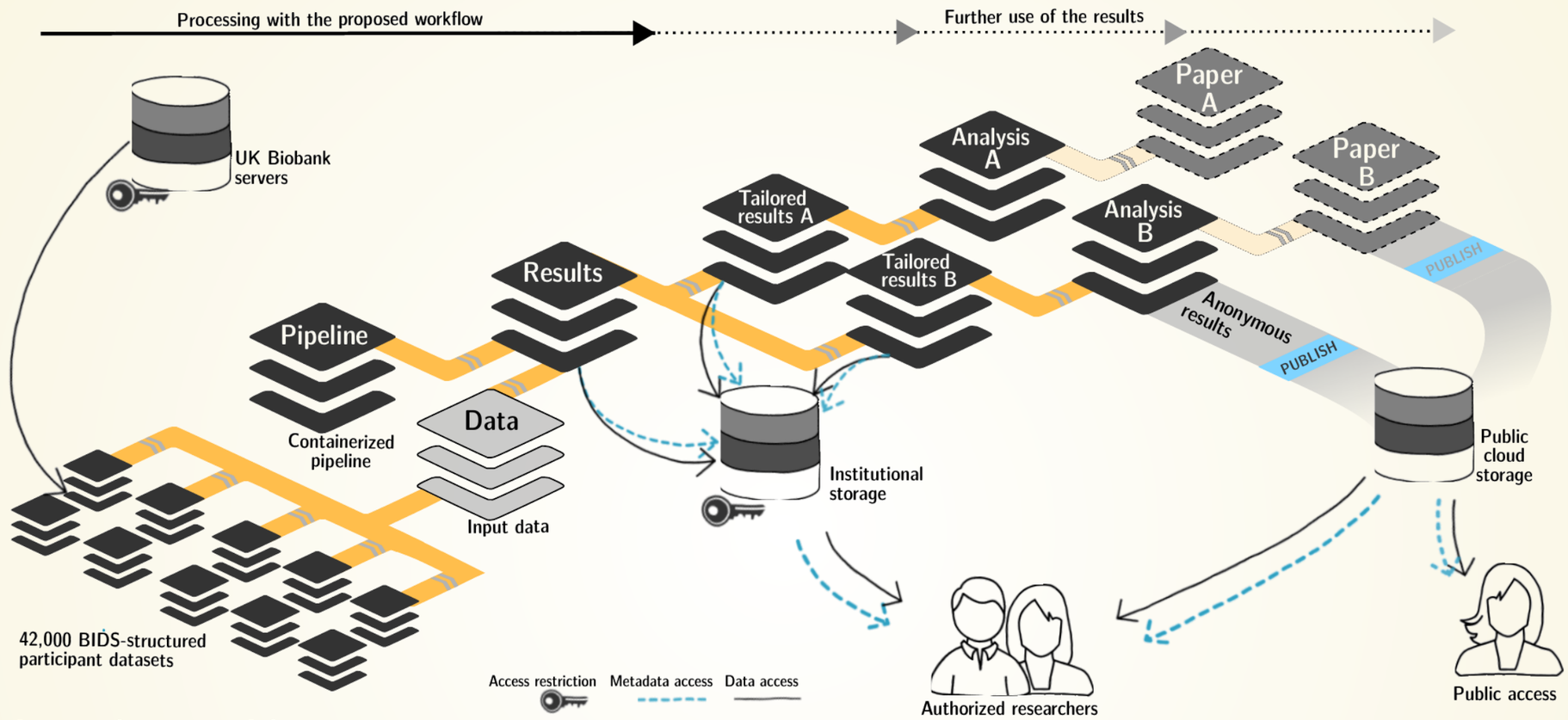


- 76 TB in 43 million files in total
- 42,715 participants contributed personal health data
- Strict DUA
- Custom binary-only downloader
- Most data records offered as (unversioned) ZIP files

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

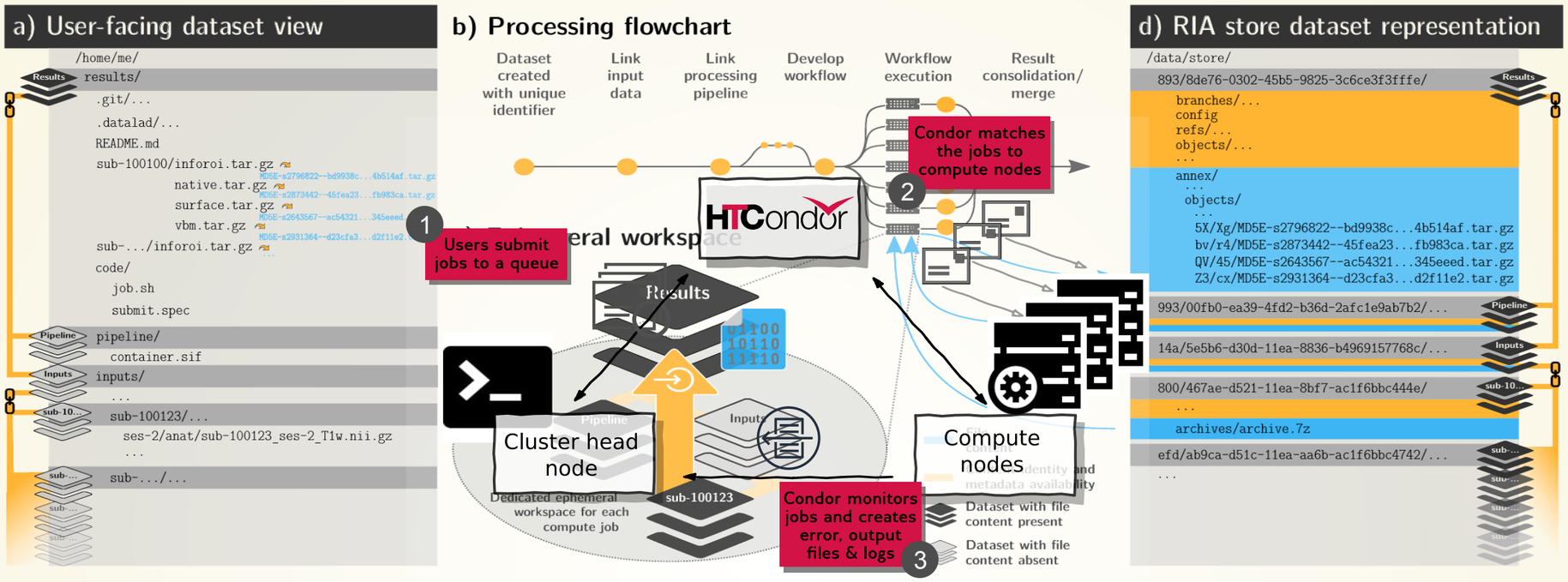


Exhaustive tracking

- 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

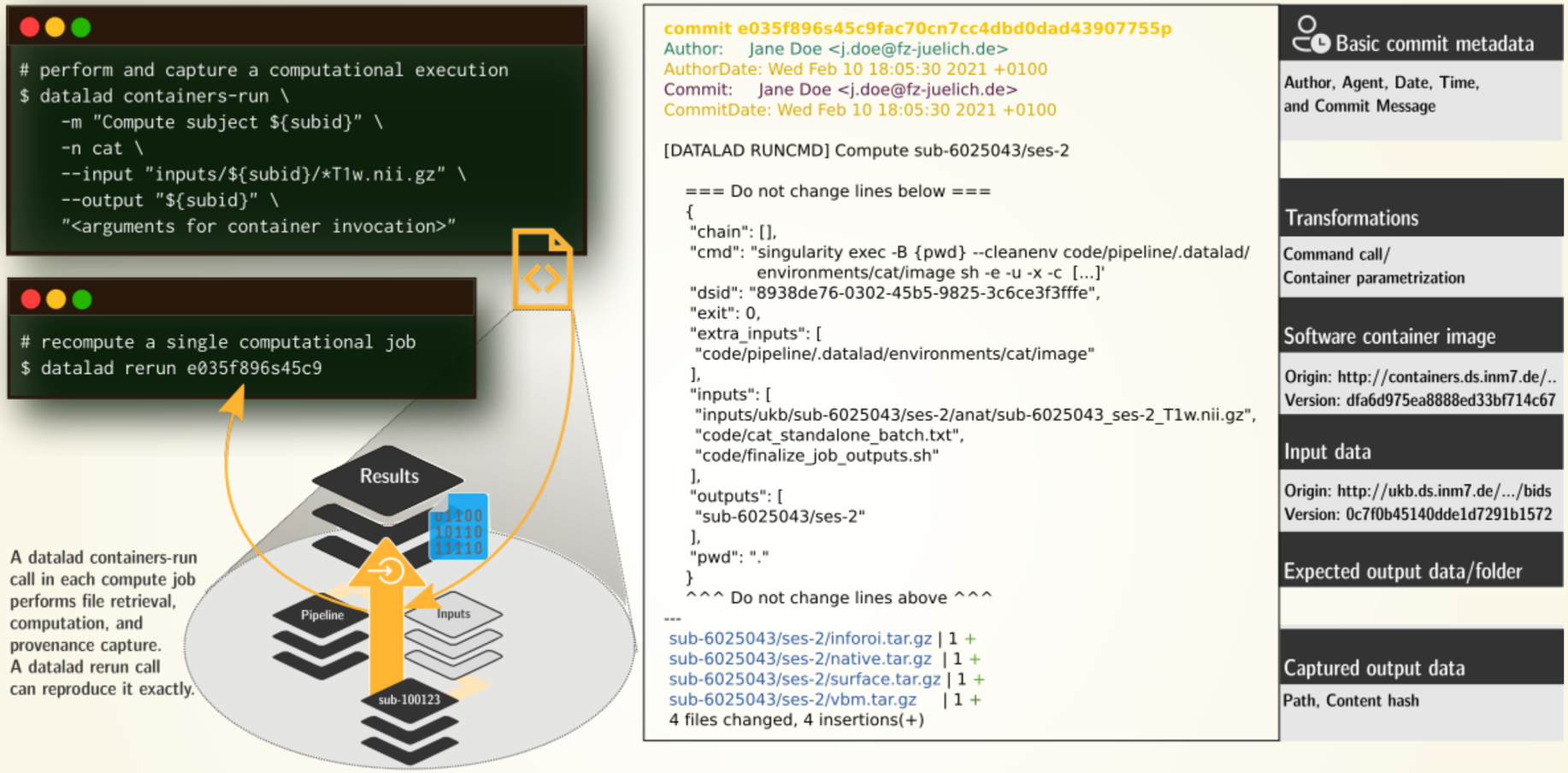


portability

- 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

THANK YOU FOR YOUR ATTENTION!

Slides: DOI 10.5281/zenodo.7627723 (Scan the QR code)

Women neuroscientists are underrepresented in neuroscience. You can use the Repository for Women in Neuroscience to find and recommend neuroscientists for conferences, symposia or collaborations, and help making neuroscience more open & divers.

COMMAND SUMMARIES

SUMMARY - LOCAL VERSION CONTROL

`datalad create` creates an empty dataset.

Configurations (**-c yoda**, **-c text2git**) add useful structure and/or configurations.

A dataset has a history to track files and their modifications.

Explore it with Git (**git log**) or external tools (e.g., **tig**).

`datalad save` records the dataset or file state to the history.

Concise **commit messages** should summarize the change for future you and others.

`datalad download-url` obtains web content and records its origin.

It even takes care of saving the change.

`datalad status` reports the current state of the dataset.

A clean dataset status (no modifications, not untracked files) is good practice.

SUMMARY - DATASET CONSUMPTION & NESTING

`datalad clone` installs a dataset.

It can be installed "on its own": Specify the source (url, path, ...) of the dataset, and an optional **path** for it to be installed to.

Datasets can be installed as subdatasets within an existing dataset.

The **--dataset/-d** option needs a path to the root of the superdataset.

Only small files and metadata about file availability are present locally after an install.

To retrieve actual file content of annexed files, `datalad get` downloads file content on demand.

Datasets preserve their history.

The superdataset records only the version state of the subdataset.

SUMMARY - REPRODUCIBLE EXECUTION

`datalad run` records a command and its impact on the dataset.

All dataset modifications are saved - use it in a clean dataset.

Data/directories specified as `--input` are retrieved prior to command execution.

Use one flag per input.

Data/directories specified as `--output` will be unlocked for modifications prior to a rerun of the command.

Its optional to specify, but helpful for recomputations.

`datalad containers-run` can be used to capture the software environment as provenance.

Its ensures computations are ran in the desired software set up. Supports Docker and Singularity containers

`datalad rerun` can automatically re-execute run-records later.

They can be identified with any commit-ish (hash, tag, range, ...)