

DATALAD - AN INTRODDUCTION

Dr. Adina Wagner

 mas.to/@adswa

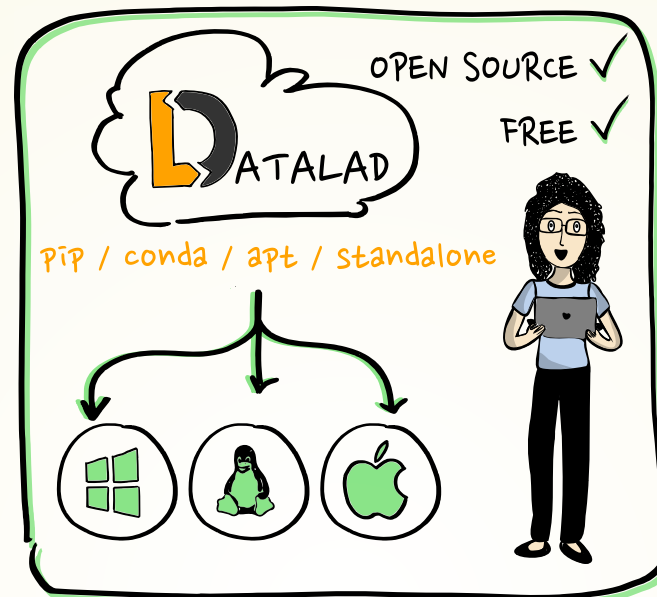
Institute of Neuroscience and Medicine,
Brain & Behavior (INM-7)
Research Center Jülich



Slides: DOI [10.5281/zenodo.13806404](https://doi.org/10.5281/zenodo.13806404)
files.inm7.de/adina/talks/html/andani.html

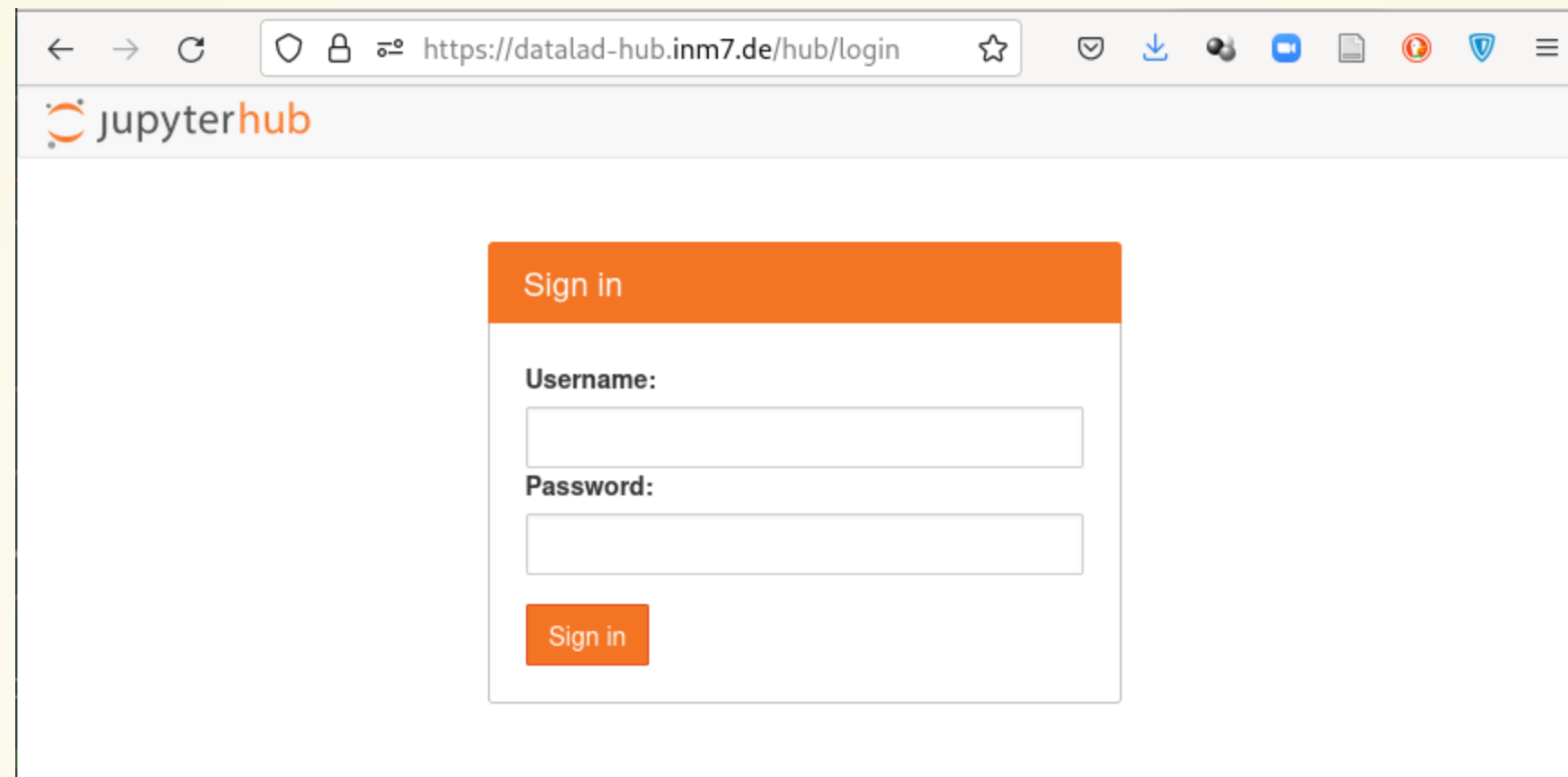
DATALAD

(**DATALAD.ORG**)



- 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 DATALAD



For convenience, we work online today:

datalad-hub.inm7.de

username:

The spice or herb you got as a user name

password:

Set at first login, at least 8 characters

On your own machines, DataLad is available via pip, conda, apt, brew; On all major operating systems: See handbook.datalad.org/r.html?install

ACKNOWLEDGEMENTS

DataLad software & ecosystem

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

Funders



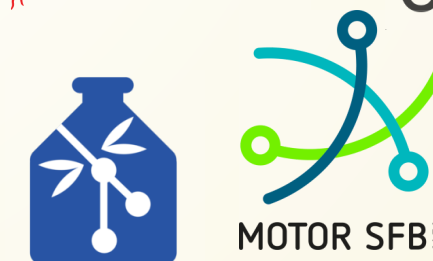
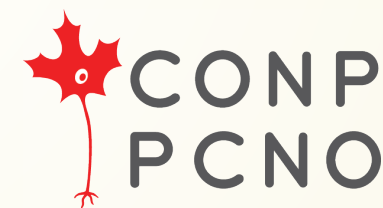
NSF 1429999



BMBF 01GQ1411



Collaborators



DATALAD USECASES

psychoinformatics-de / paper-remodnav

Public

Edit Pins 11 Unwatch 2 Fork 3 Star

Code Issues Pull requests Actions Projects Wiki Security Insights

master 6 branches 2 tags Go to file Add file Code

adswa Merge pull request #19 from psychoinformatics-de/reproducibility-im... 4edccb8 on Sep 14, 2021 451 commits

code	Generate figures with deterministic metadata for comp. reproducibility	10 months ago
data	Point to latest label dataset	3 years ago
img	Include original SVGs into the repo to allow immediate manuscript ren...	10 months ago
remodnav @ d289118	Update remodnav with latest test dataset	3 years ago
.gitignore	Prevent permanent rebuilds of the figures	3 years ago
.gitmodules	[DATALAD] modified subdataset properties	16 months ago
COPYING	Declare CC-BY license	3 years ago
Makefile	Specify Python package versions in the state of final submission June...	10 months ago
README.md	DOC: improve readme, differentiate between recompile and recompute	10 months ago

About

Code, data and manuscript for <https://doi.org/10.1101/619254>

Readme CC-BY-4.0 license 3 stars 11 watching 2 forks

Releases

2 tags

Create a new release

FURTHER RESOURCES AND STAY IN TOUCH

If you have questions after the session...

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**

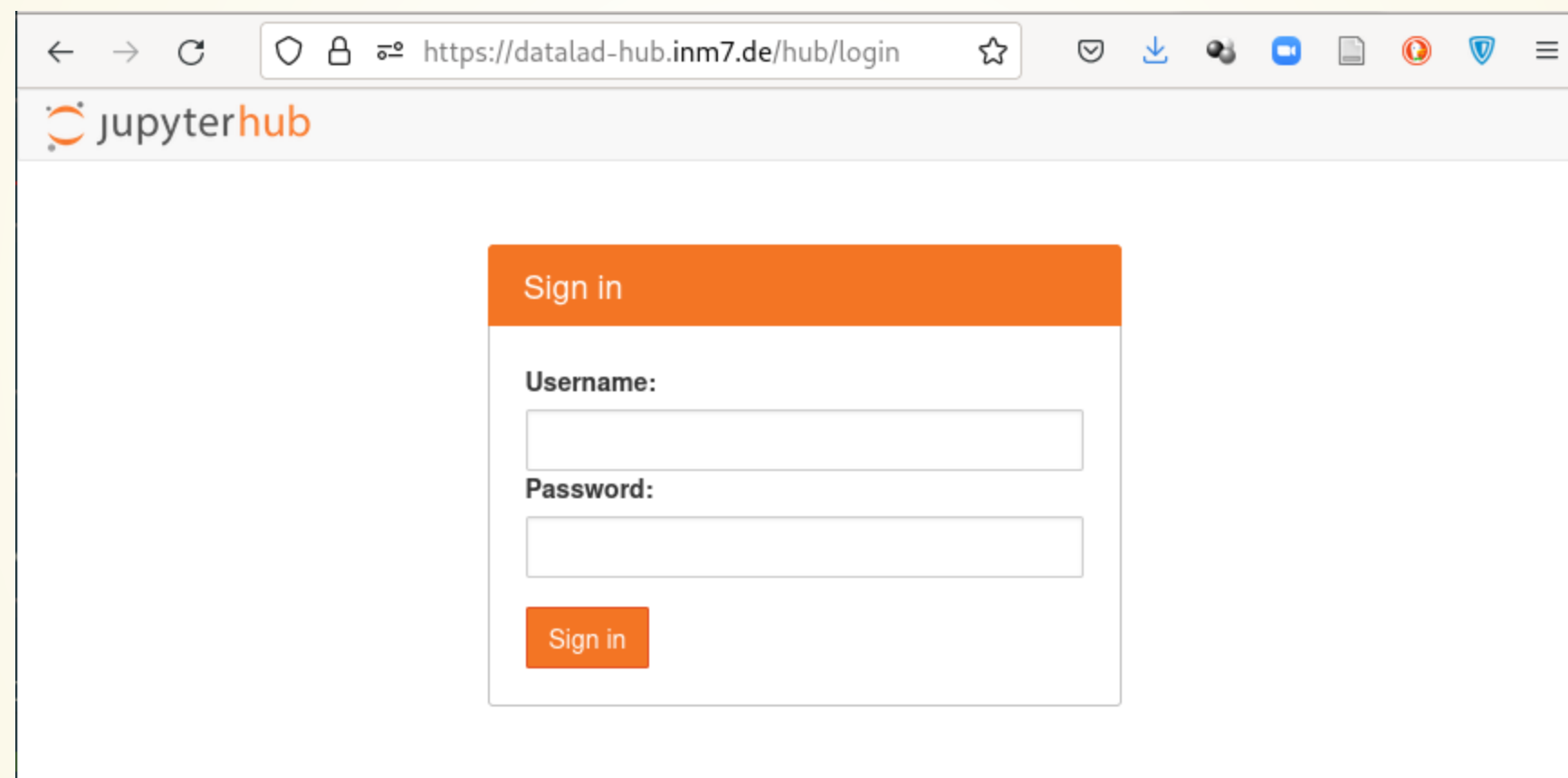
Reach out to the (Neuro-) user community with

- A question on **neurostars.org** with a data`lad` 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**

LET'S TRY DATALAD



A screenshot of a web browser showing the login page for JupyterHub at <https://datalad-hub.inm7.de/hub/login>. The page features the JupyterHub logo at the top left. In the center, there is a white box with an orange header labeled "Sign in". Below the header, there are two input fields: "Username:" and "Password:". At the bottom of the box is an orange button labeled "Sign in". The browser's address bar shows the URL, and the top of the browser window displays various icons and a hamburger menu.

datalad-hub.inm7.de

username:

The spice or herb you got as a user name

password:

Set at first login, at least 8 characters

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](#)

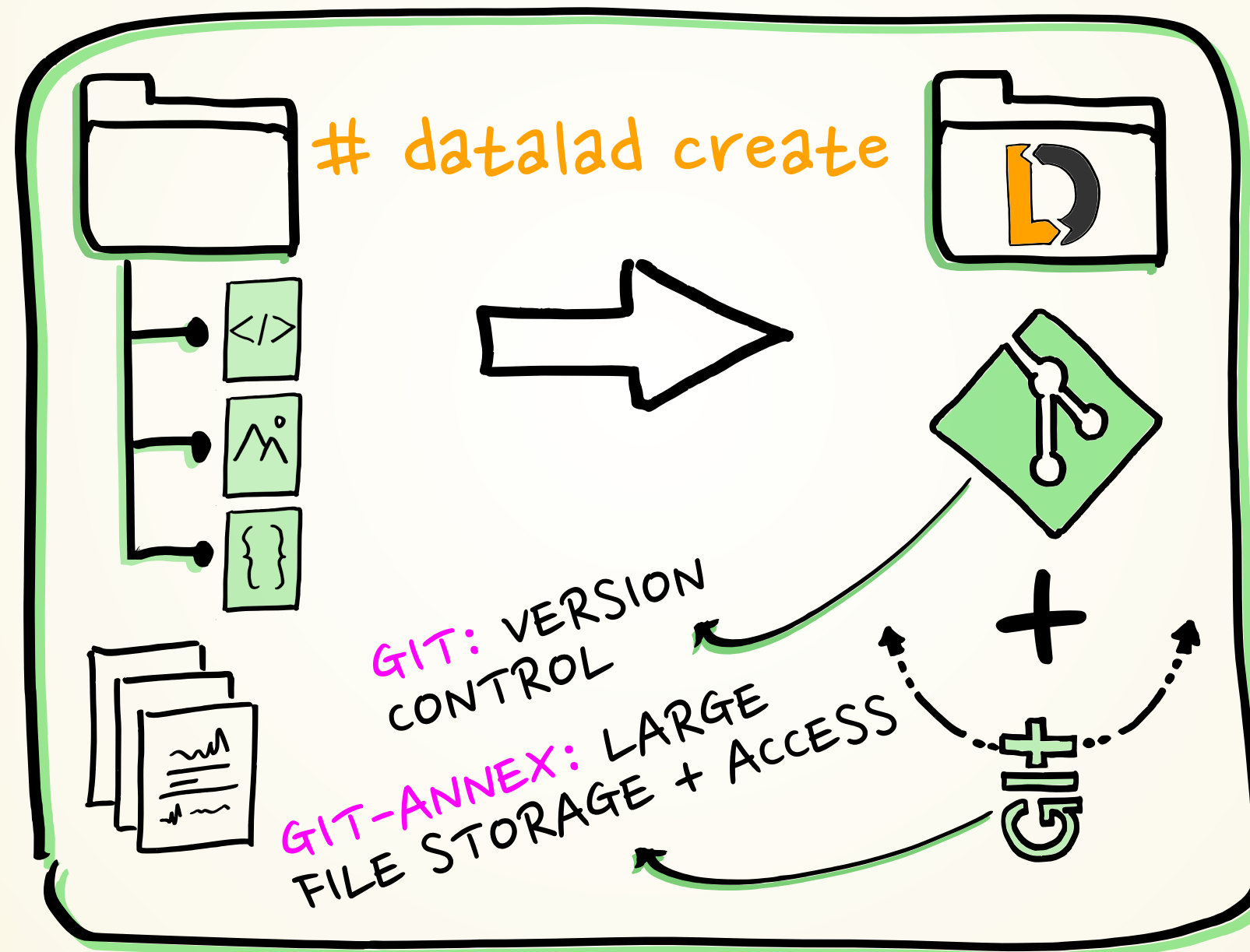
Exit the Python environment:

```
exit
```

[copy](#)

DATALAD DATASETS...

● THE DATALAD DATASET



...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
```

[copy](#)

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

```
cd my-analysis
```

[copy](#)

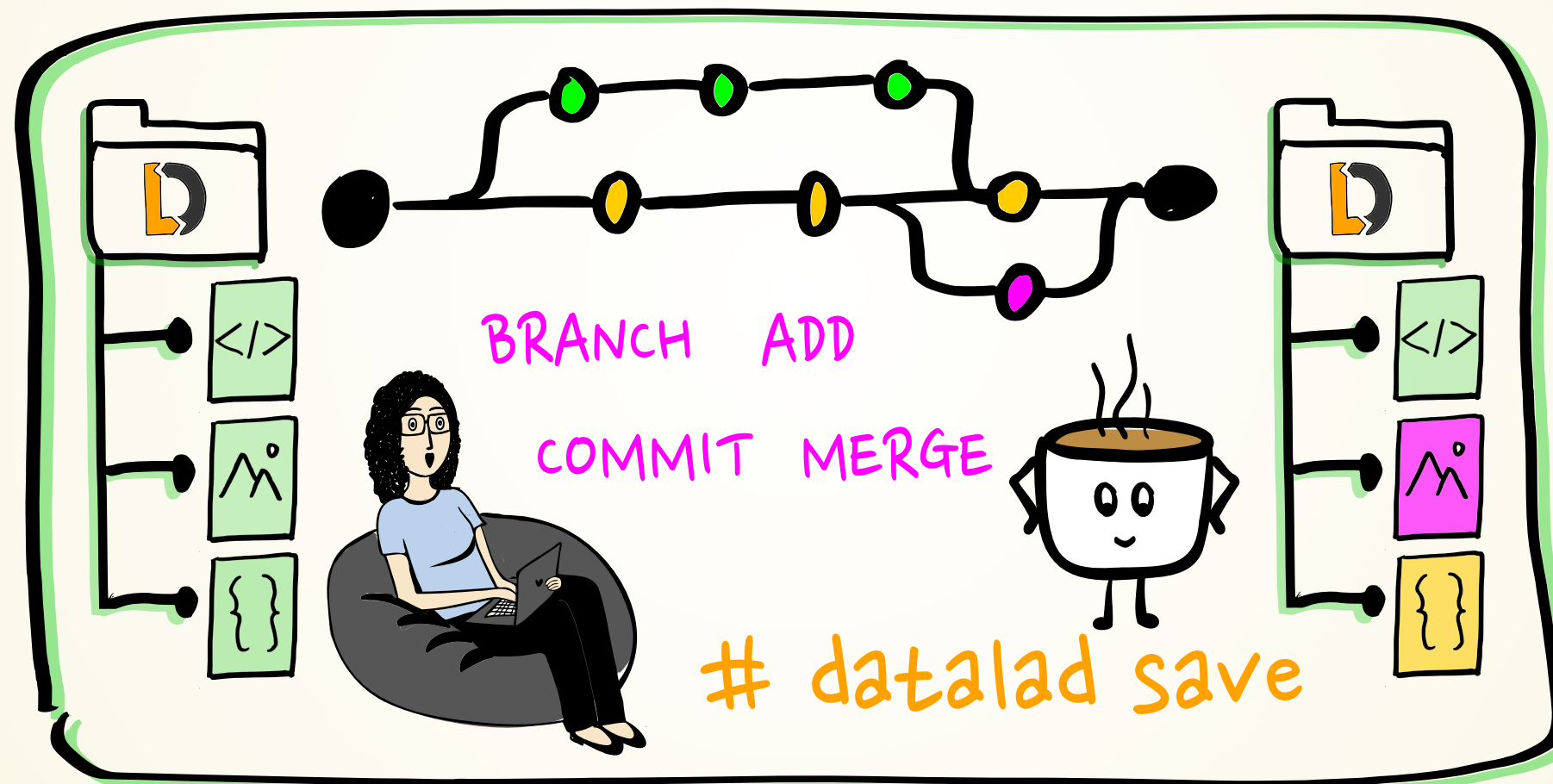
List the directory content, including hidden files, with `ls`:

```
ls -la .
```

[copy](#)

VERSION CONTROL...

● VERSION CONTROL WITH GIT



...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](#)

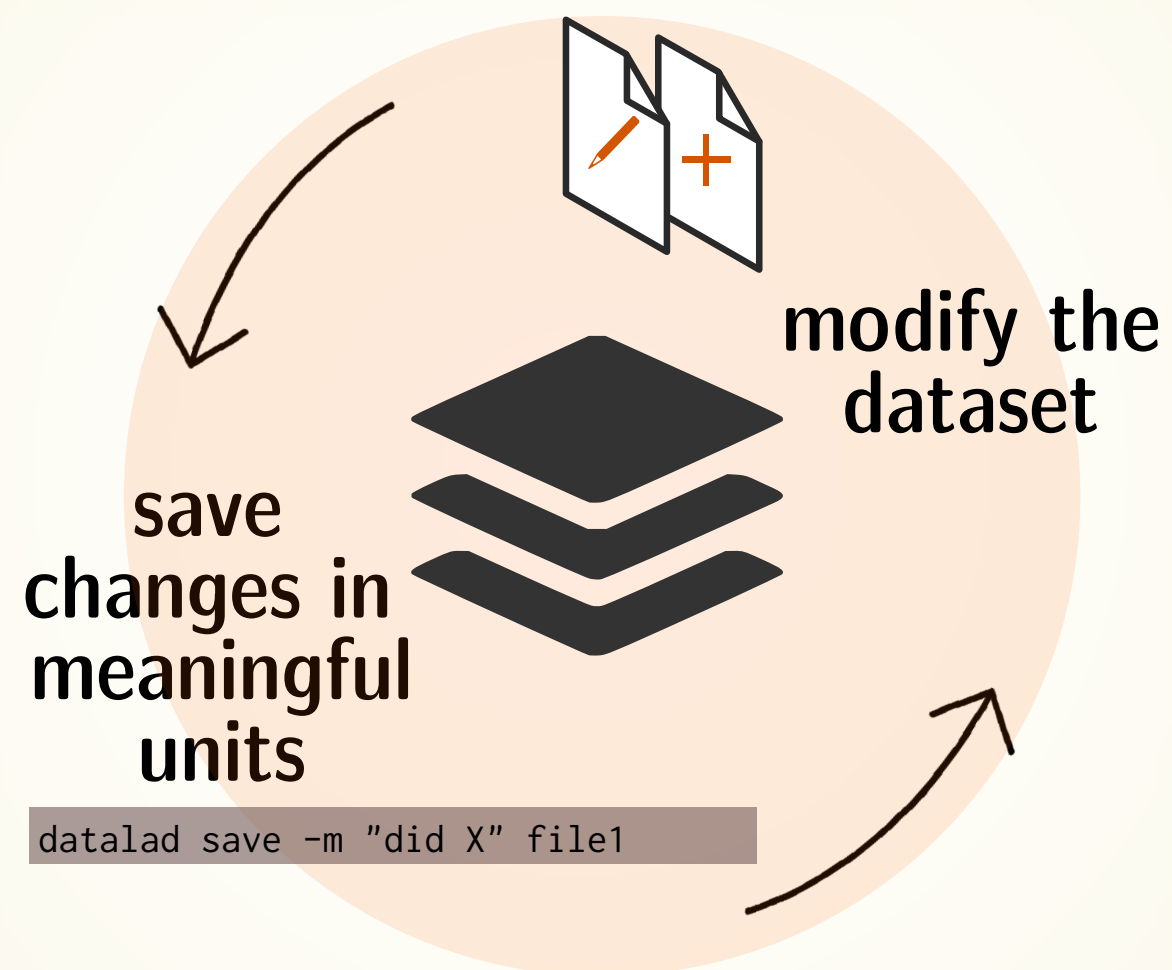
Check out the file's history:

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

[copy](#)

LOCAL VERSION CONTROL

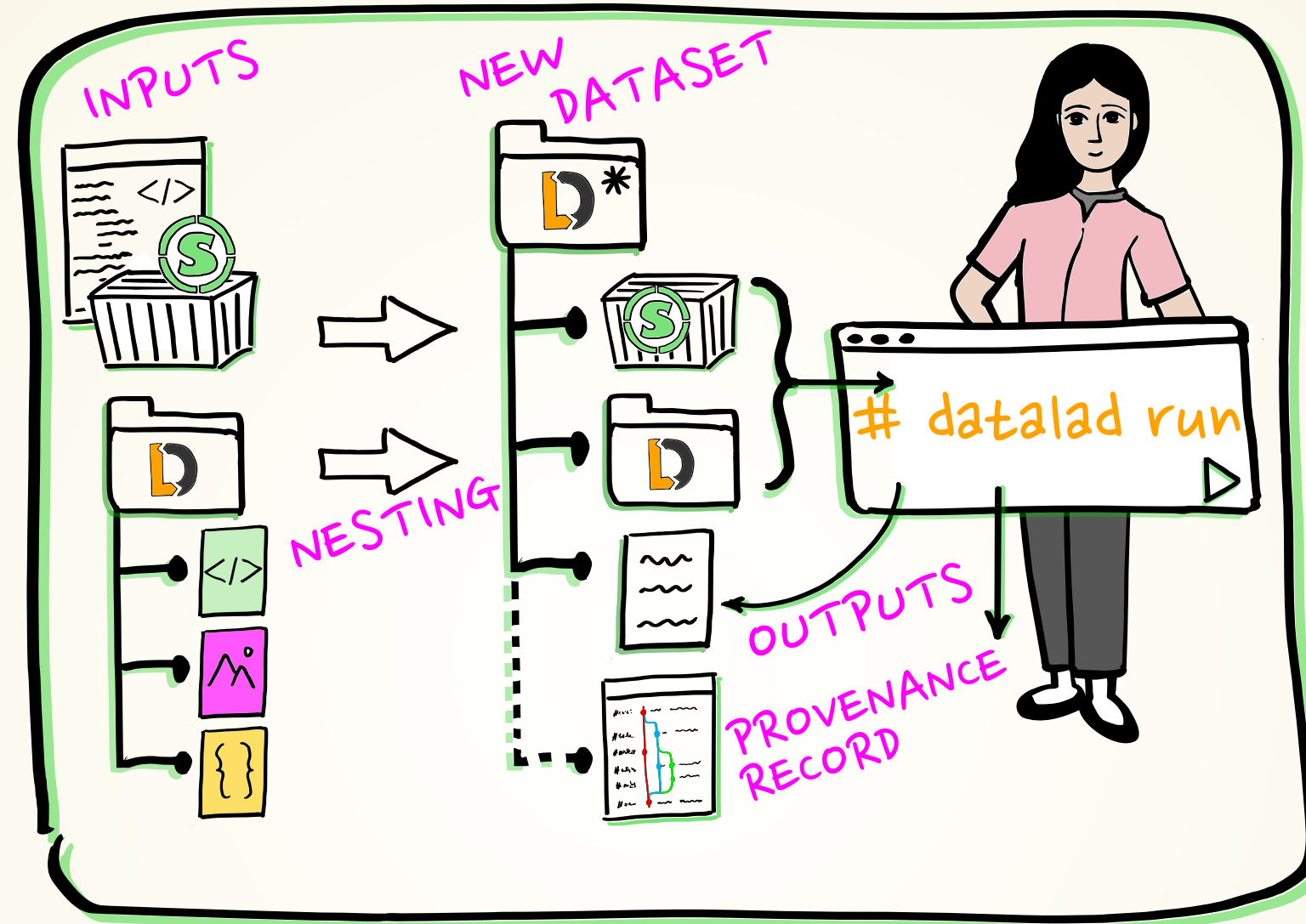
Procedurally, version control is easy with DataLad!



- Save meaningful units of change
- Advice:**
- Attach helpful commit messages

COMPUTATIONALLY REPRODUCIBLE EXECUTION I...

● EXACT DEPENDENCIES+PROVENANCE



- 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
```

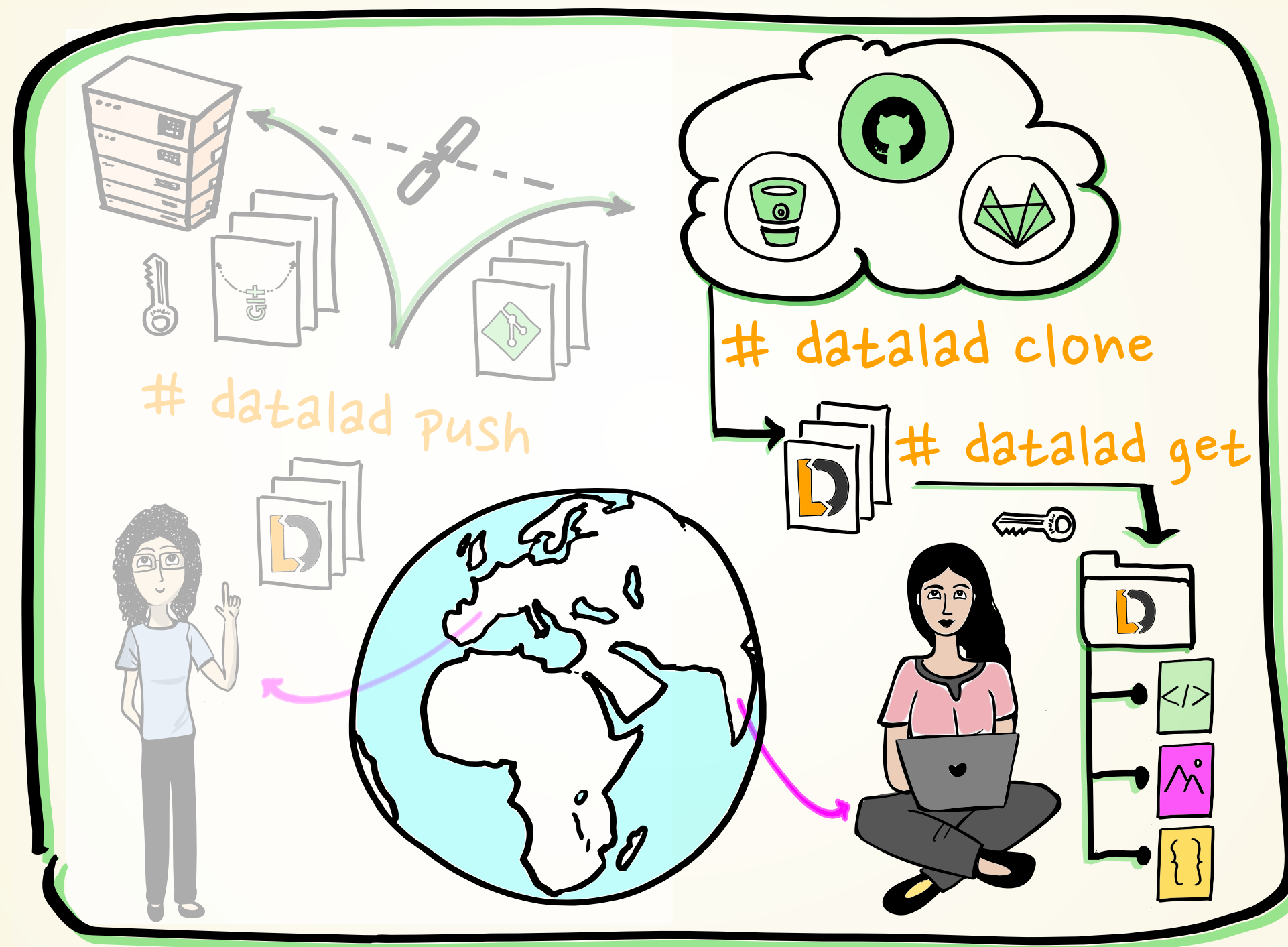
[copy](#)

... 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:

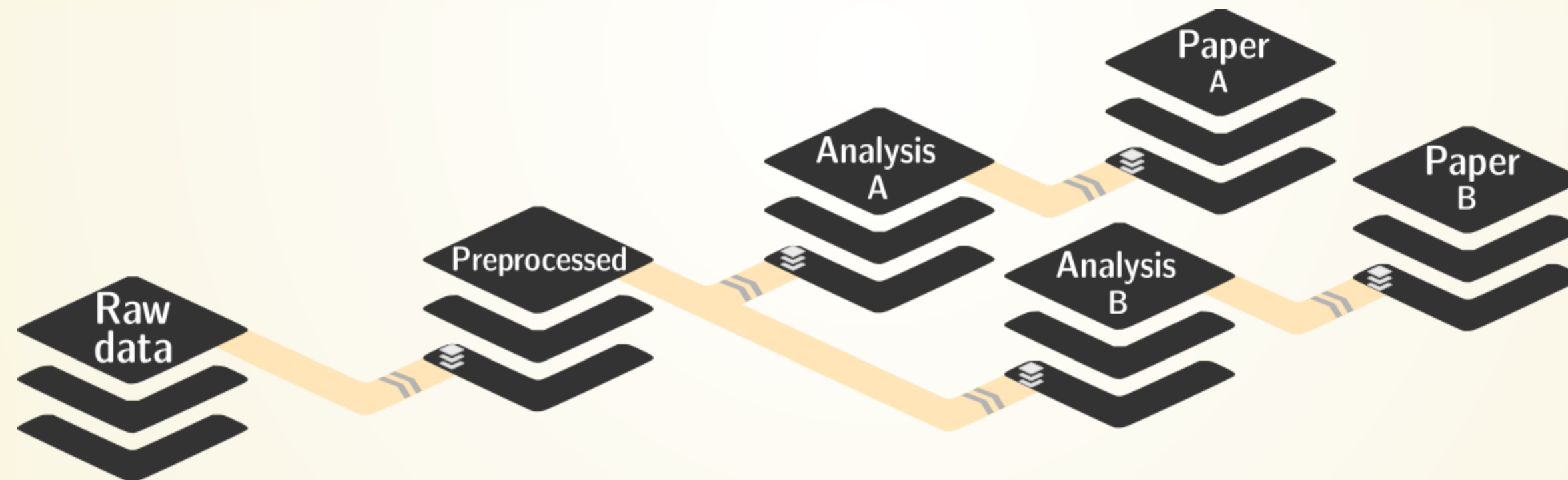
```
# just an example:  
datalad clone \  
https://github.com/psychoinformatics-de/studyforrest-data-phase2.git
```

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 *subdataset*:

```
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')
```

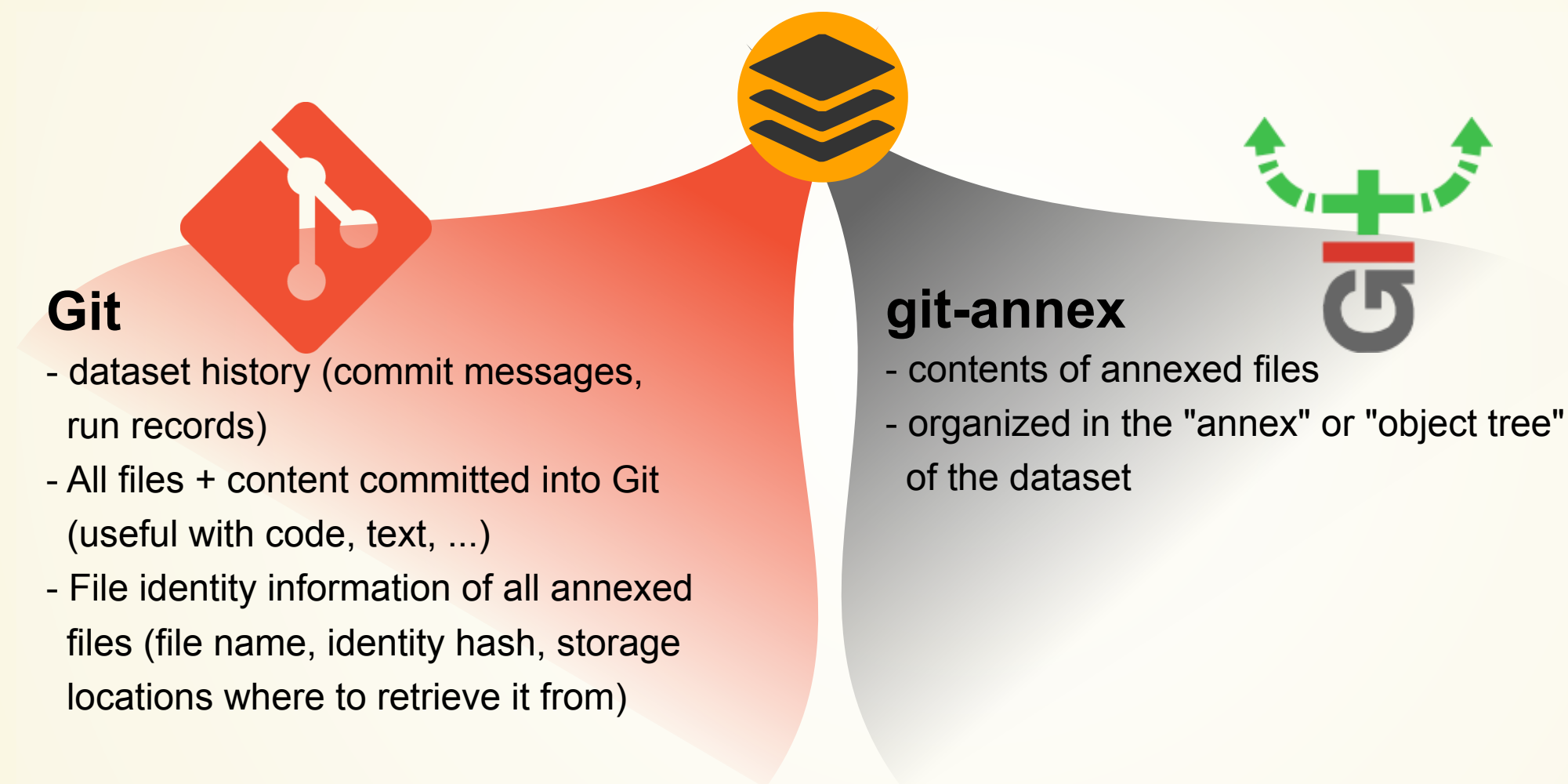
[copy](#)

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](https://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 `data-lad-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

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

[copy](#)

inspect the list of registered containers:

```
data-lad containers-list
```

[copy](#)

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

```
data-lad 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}"
```

[copy](#)

What changed after the `containers-run` command has completed?

We can use `data-lad diff` (based on `git diff`):

```
data-lad diff -f HEAD~1
```

[copy](#)

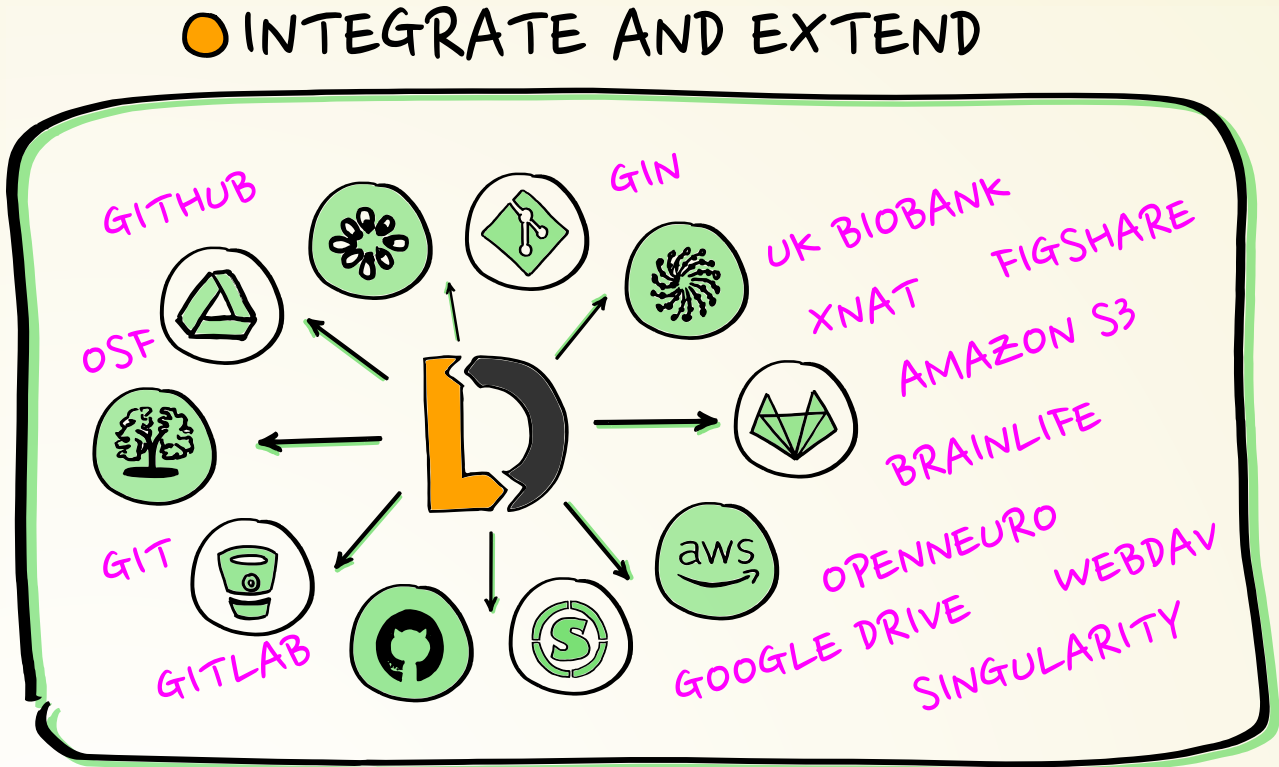
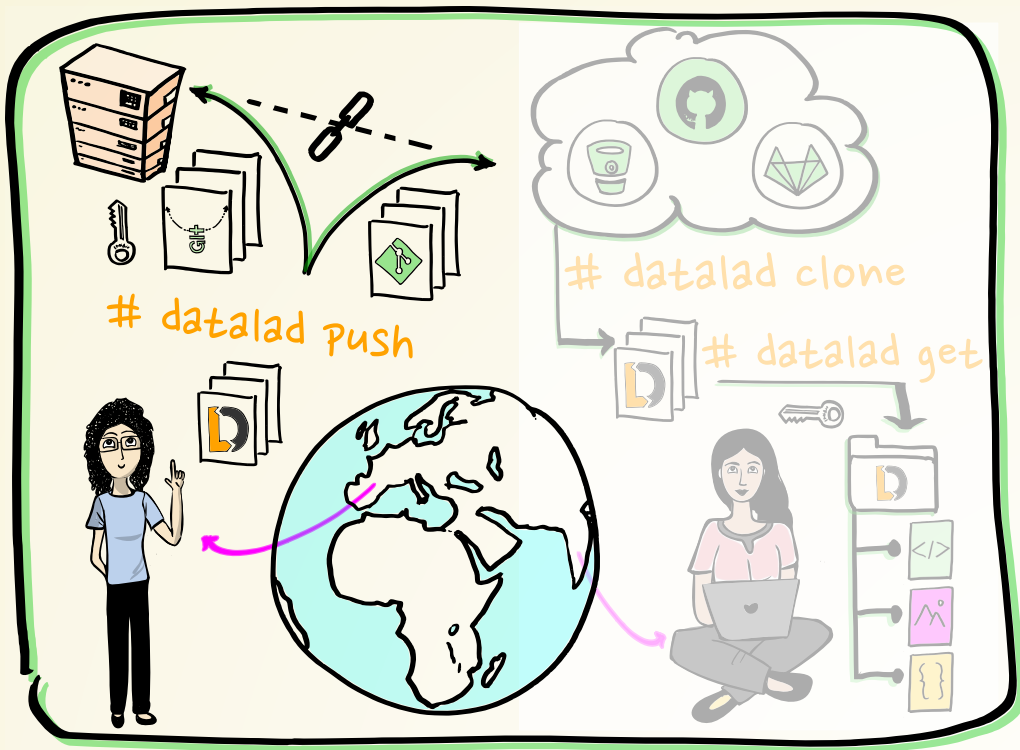
We see that some files were added to the dataset!

And we have a complete provenance record as part of the git history:

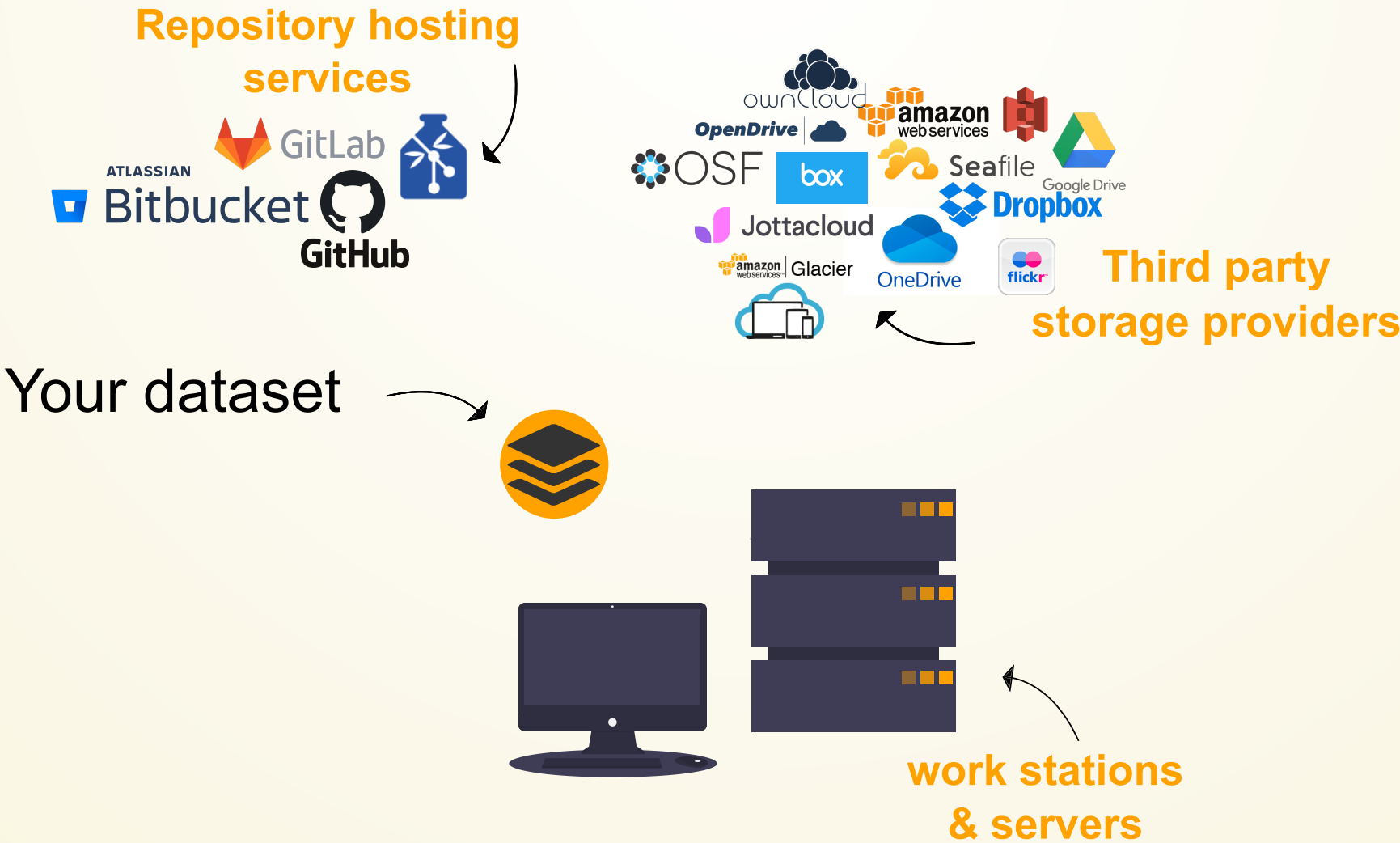
```
git log -n 1
```

[copy](#)

PUBLISHING DATASETS...



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



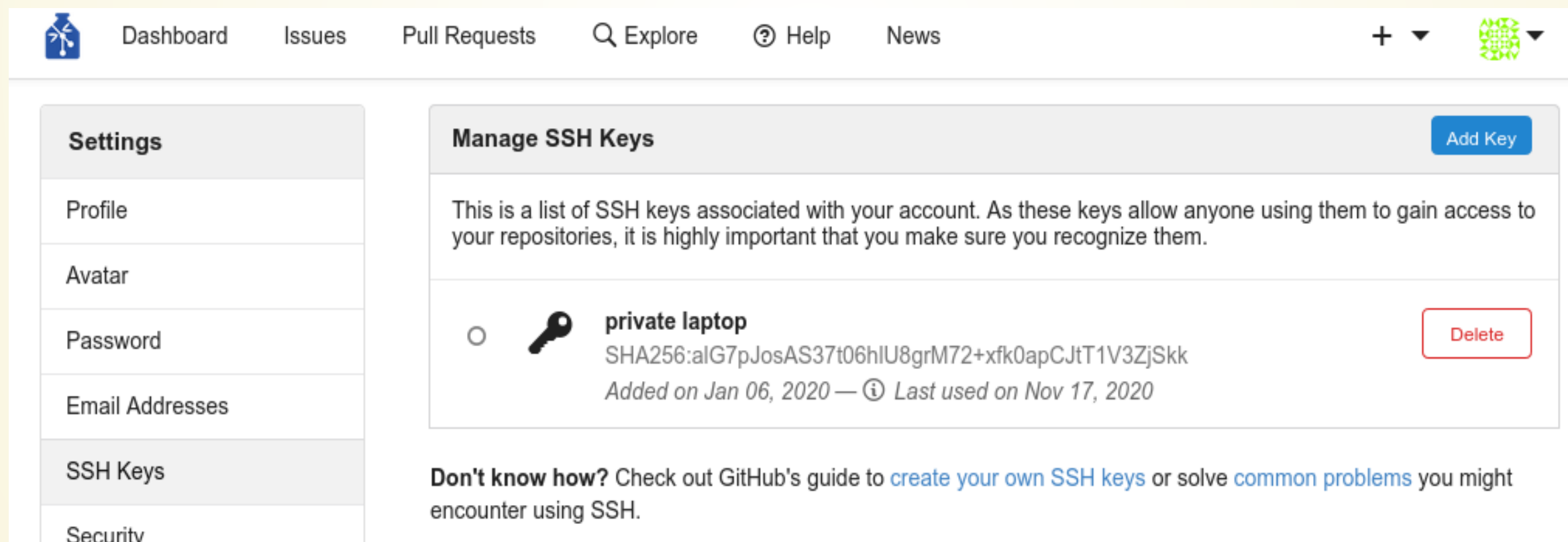
PUBLISHING DATASETS...

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

```
ssh-keygen -t ed25519 -C "your-email"  
eval "$(ssh-agent -s)"  
ssh-add ~/.ssh/id_ed25519
```

- upload the SSH key to GIN

```
cat ~/.ssh/id_ed25519.pub
```



The screenshot shows the GitHub 'Manage SSH Keys' interface. The top navigation bar includes links for Dashboard, Issues, Pull Requests, Explore, Help, and News. The left sidebar contains a 'Settings' menu with options for Profile, Avatar, Password, Email Addresses, SSH Keys (which is currently selected), and Security. The main content area is titled 'Manage SSH Keys' and features an 'Add Key' button. Below this, a message states: 'This is a list of SSH keys associated with your account. As these keys allow anyone using them to gain access to your repositories, it is highly important that you make sure you recognize them.' A single SSH key is listed with the title 'private laptop', a key icon, and a SHA256 fingerprint: 'SHA256:alG7pJosAS37t06hIU8grM72+xfk0apCJtT1V3ZjSkk'. It also shows the key was 'Added on Jan 06, 2020' and 'Last used on Nov 17, 2020'. A 'Delete' button is located to the right of the key entry. At the bottom, a note says: 'Don't know how? Check out GitHub's guide to [create your own SSH keys](#) or solve [common problems](#) you might encounter using SSH.'

- 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](#)

In case of fire



1. `git commit`

(`datalad save`)



2. `git push`

(`datalad push`)



3. `leave building`

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](#)

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**:



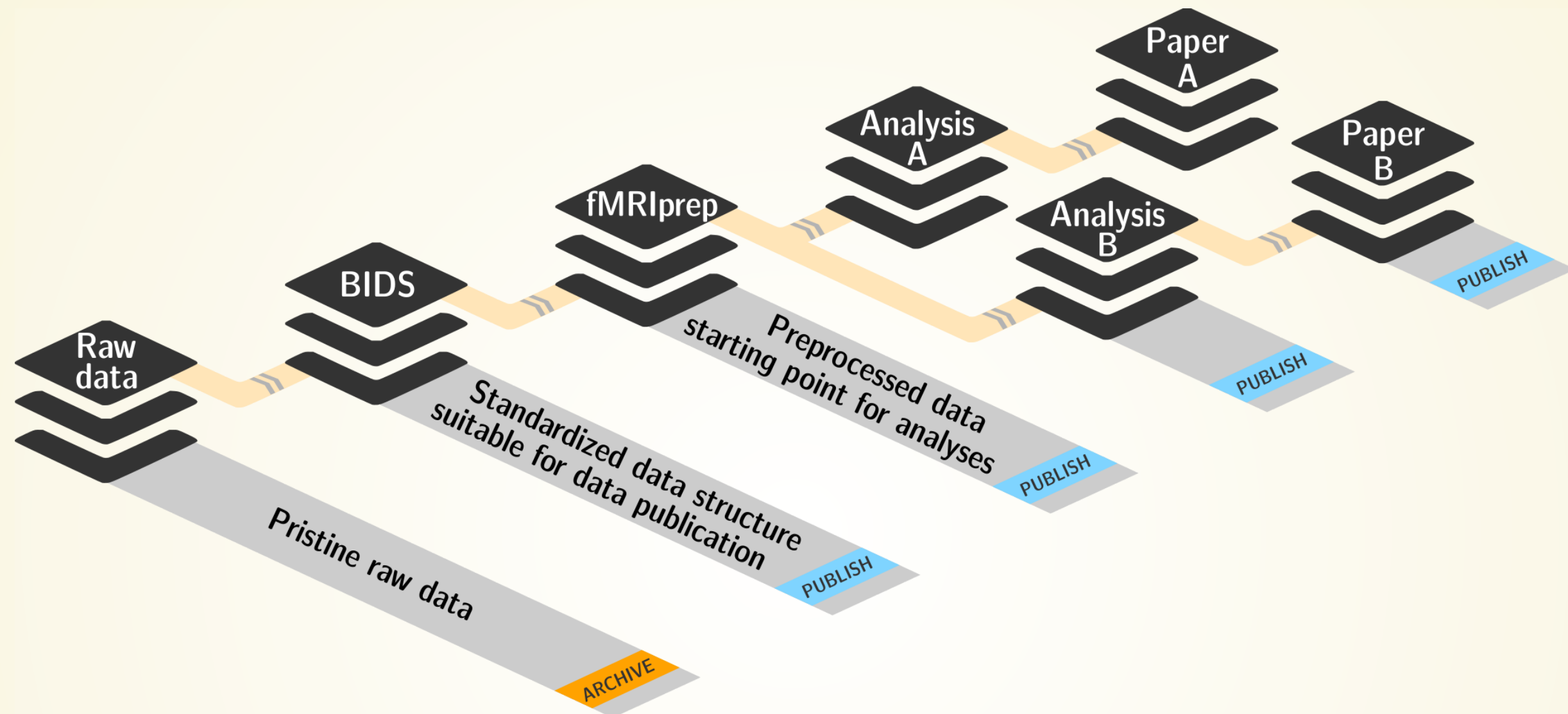
P1: One thing, one dataset

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, ...)

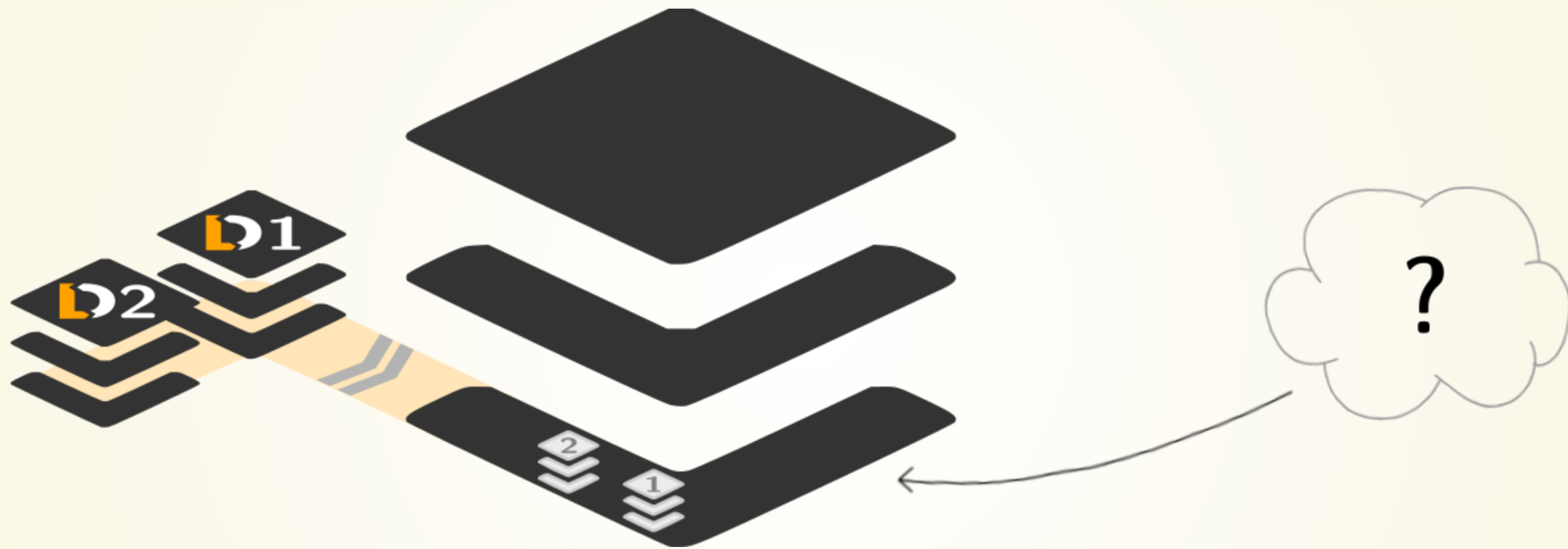
```
/derived_dataset
├── sample1
│   └── ps34t.dat
├── sample2
│   └── ps34t.dat
├── ...
└── inputs
    └── raw
        ├── sample1
        │   └── a001.dat
        ├── sample2
        │   └── a001.dat
        └── ...
```

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

WHEN TO MODULARIZE?

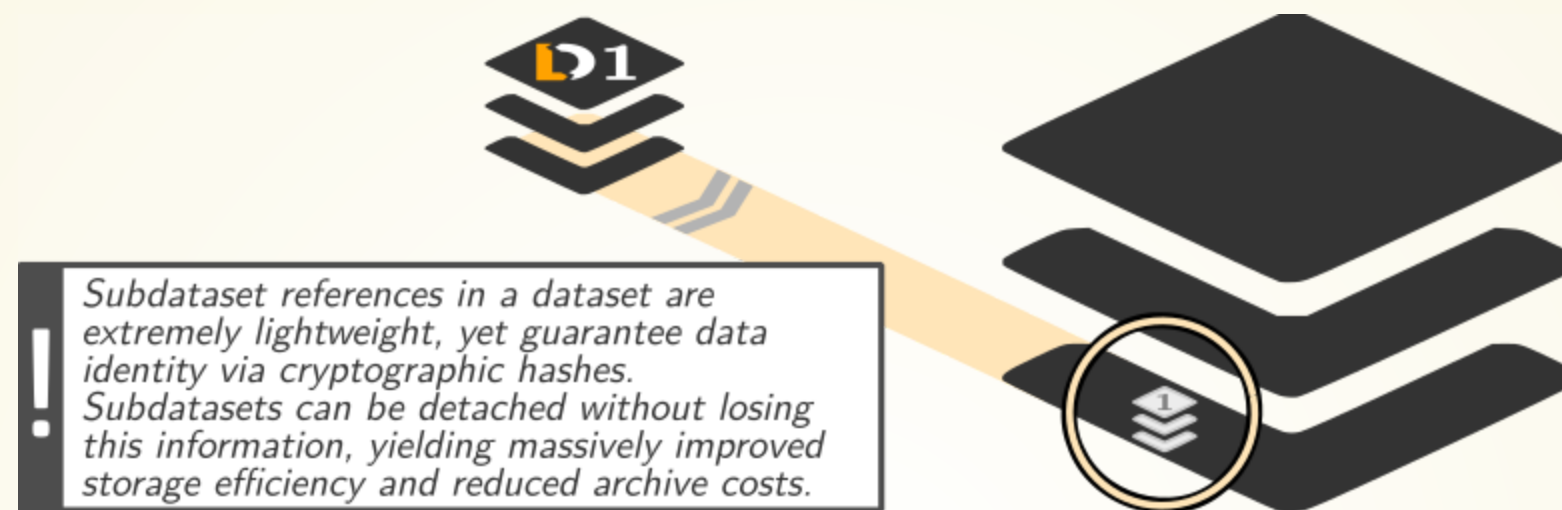
- Target audience is different
 - public vs. private
 - domain specific vs. domain general
- Pace of evolution is different
 - "factual" raw data vs. choices of (pre-)processing
 - completed acquisition vs. ongoing study
- Size impacts I/O and logistics
 - Git can struggle with 1M+ files
 - filesystems (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
```

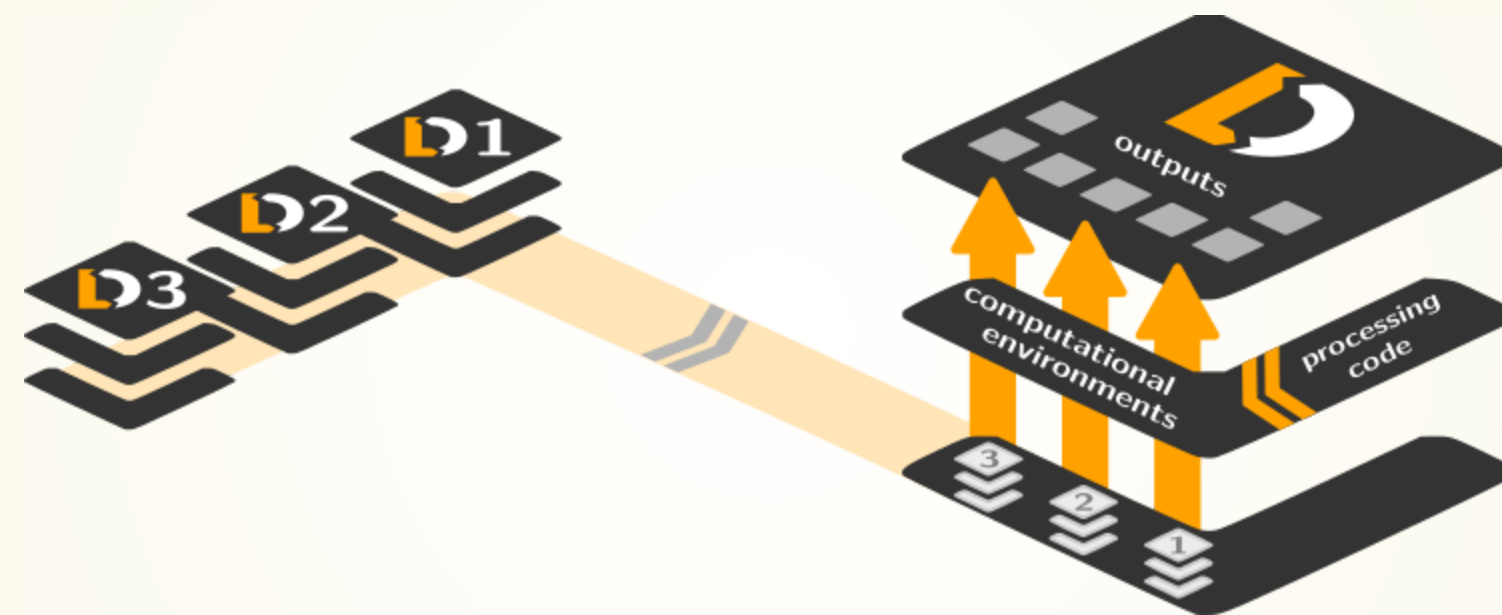
copy

```
$ 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

What does DataLad add to Git and git-annex?

- Simple(r) core API to unify Git and git-annex functionality

- Ability to record provenance

- Support for software container solutions (Singularity, Docker)

- Subdatasets and linkage with a mono-repo-like user-experience

- Interoperability adapters to publish to a variety of hosting services

- Open Data Distribution: To date, more than 600TB of open neuro data are available via [via datasets.datalad.org](https://datasets.datalad.org)

THANK YOU FOR YOUR ATTENTION!



Slides: DOI [10.5281/zenodo.10118794](https://doi.org/10.5281/zenodo.10118794) (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

data`lad` 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`).

data`lad` save records the dataset or file state to the history.

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

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

It even takes care of saving the change.

data`lad` status reports the current state of the dataset.

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

SUMMARY - DATASET CONSUMPTION & NESTING

`data-lad 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, `data-lad get` downloads file content on demand.

Datasets preserve their history.

The superdataset records only the version state of the subdataset.

SUMMARY - REPRODUCIBLE EXECUTION

data`lad` 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.

data`lad` 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

data`lad` rerun can automatically re-execute run-records later.

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

