BIDS and DATALAD in a nutshell

your experiment data, FAIR and reusable

Giorgio Marinato (@neurogima)

...and some slides-karaoke from the projects

Outline

- General motivation to start employing BIDS and data versioning system

Outline

- General motivation to start employing BIDS and data versioning system
- BIDS and MNE-BIDS data handling

Outline

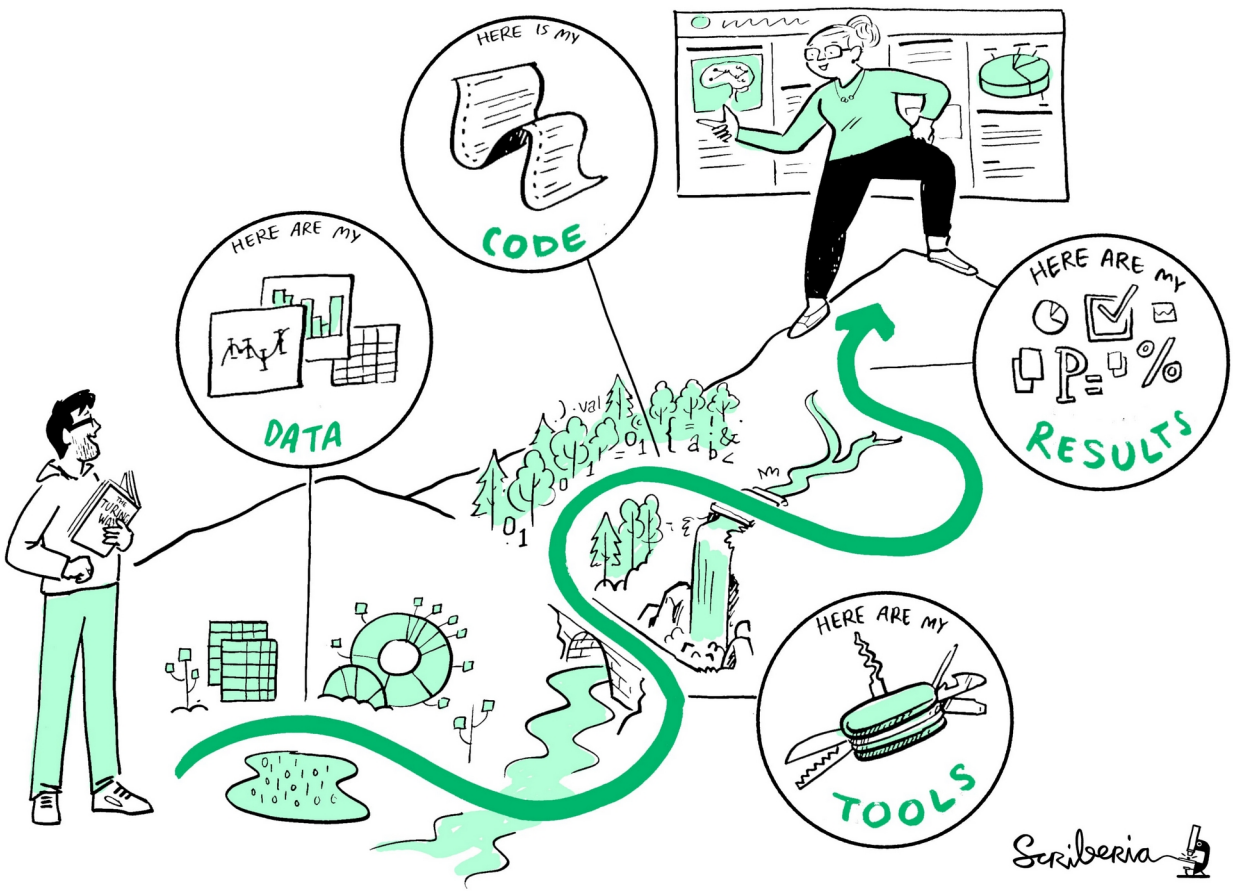
- General motivation to start employing BIDS and data versioning system
- BIDS and MNE-BIDS data handling
- DataLad intro and basic usage

Outline

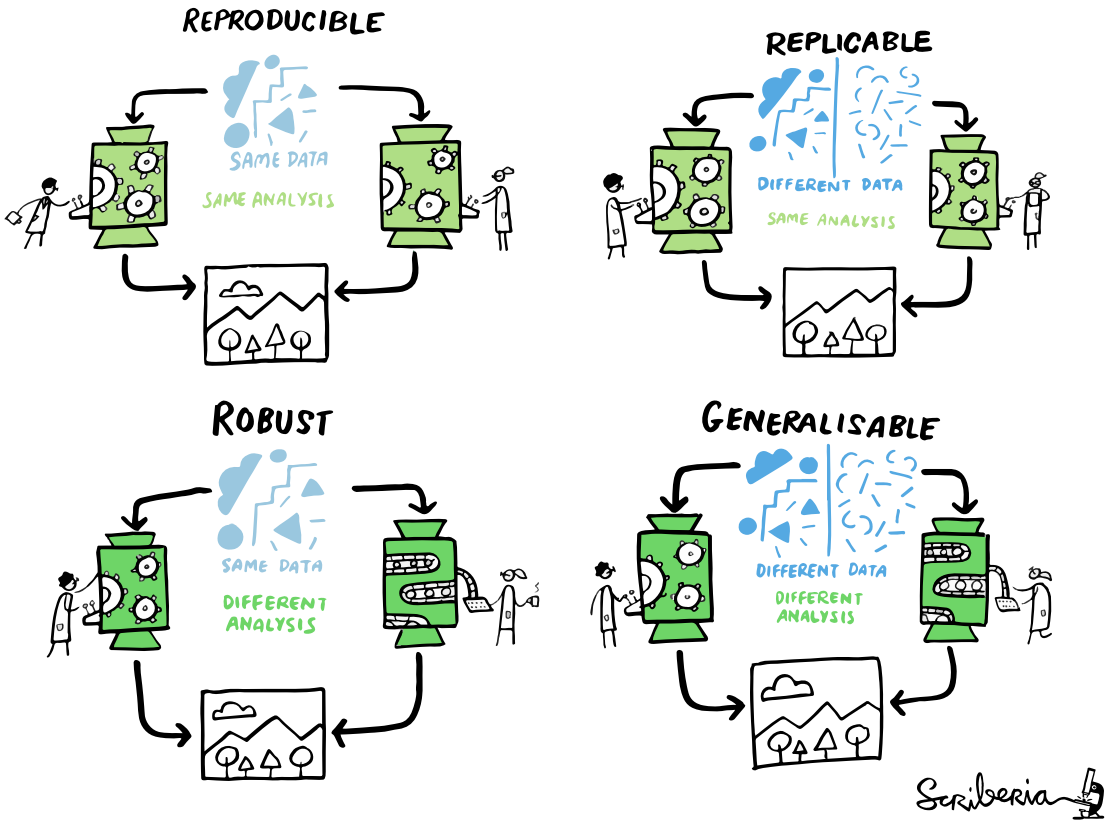
- General motivation to start employing BIDS and data versioning system
- BIDS and MNE-BIDS data handling
- DataLad intro and basic usage

With some live coding if you like and time permit...

Responsibility of reproducibility in data science and research



To achieve a scientific research that is:



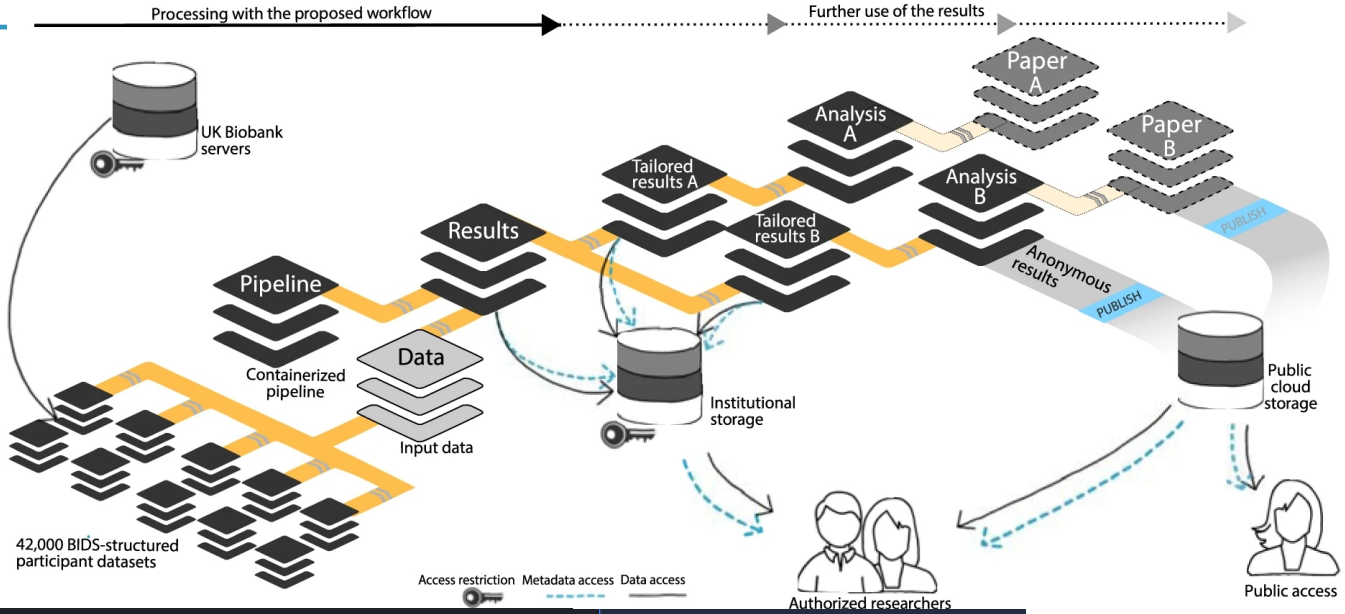
And to automatize advanced pipelines!

nature > scientific data > articles > article

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



What is MNE-BIDS-Pipeline?

MNE-BIDS-Pipeline is a full-flegded processing pipeline for your MEG and EEG data.

- It operates on data stored according to the Brain Imaging Data Structure (BIDS).
- Under the hood, it uses MNE-Python.

1. Filesystem initialization and dataset inspection

Click to expand

2. Preprocessing

Click to expand

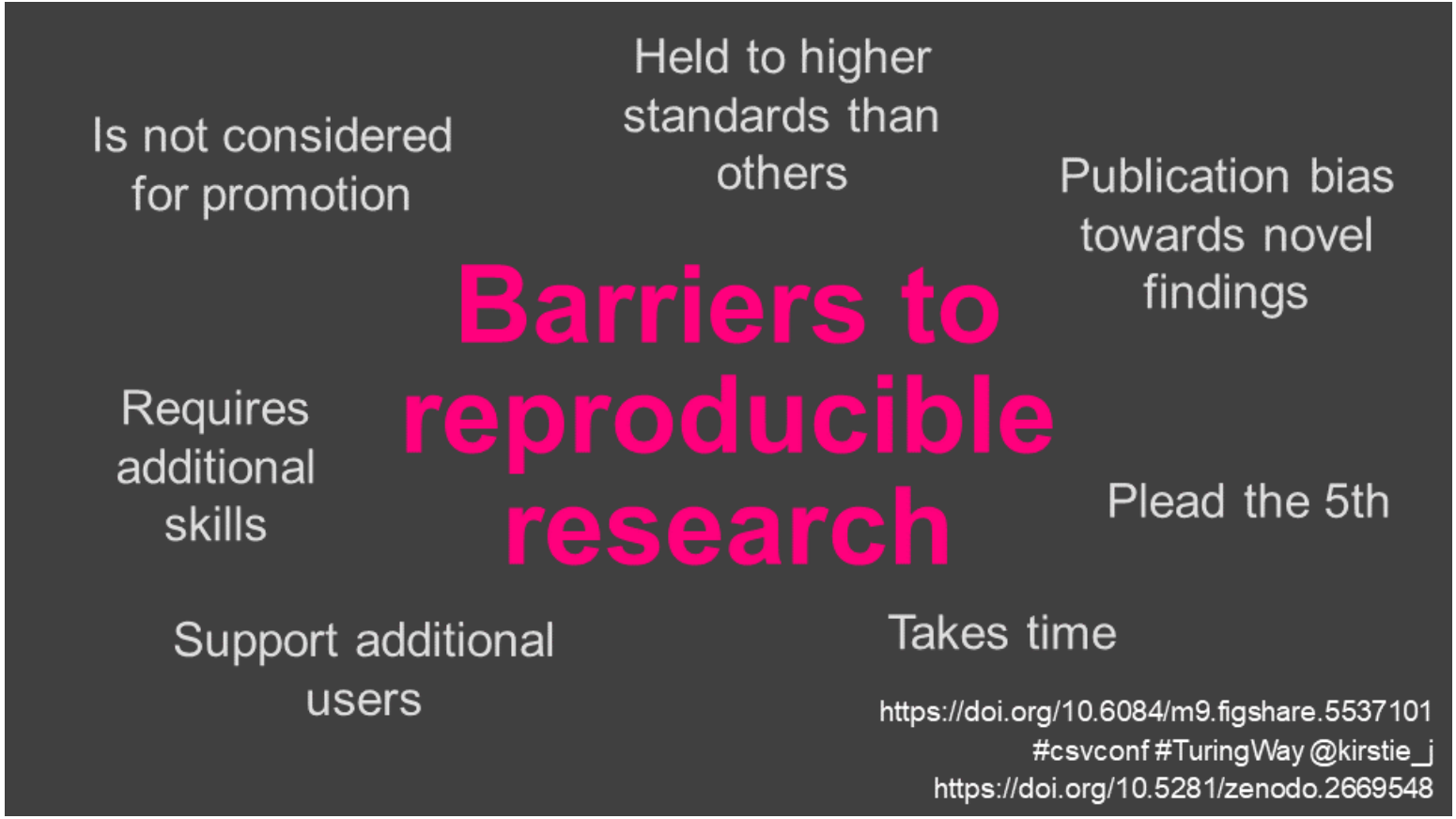
3. Sensor-space analysis

Click to expand

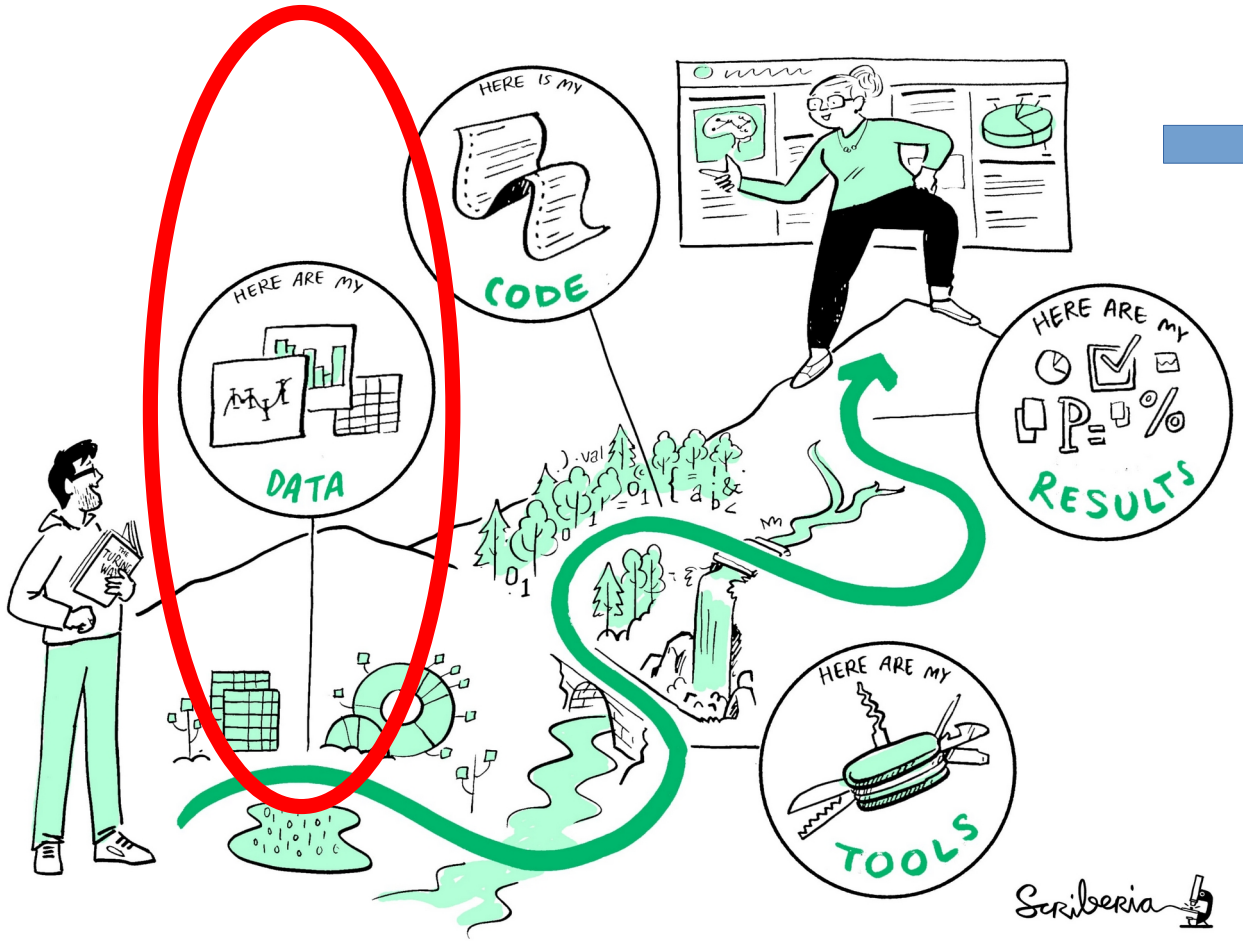
4. Source-space analysis

Click to expand

But we need to deconstruct some bias...



Today is data step



Findable **A**ccessible **I**nteroperable **R**eusable

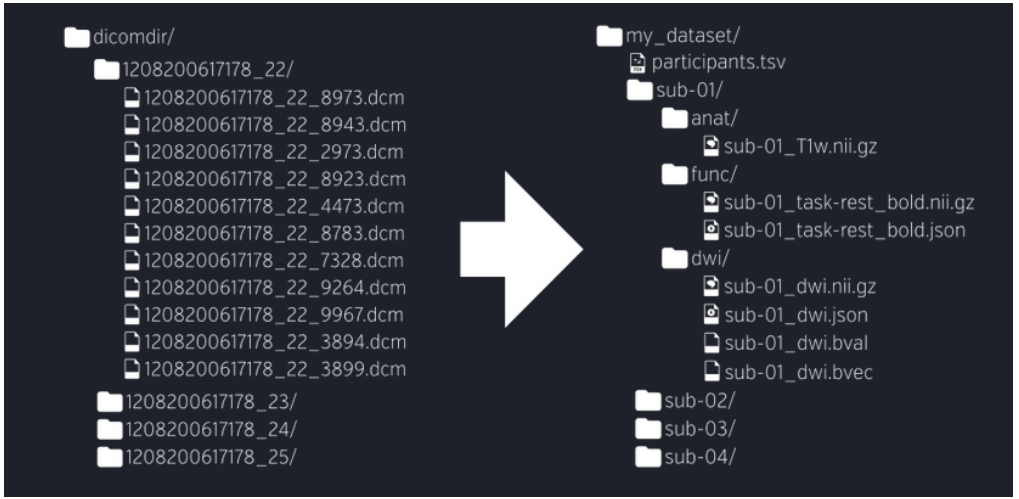
BIDS
BRAIN IMAGING DATA STRUCTURE

Datalad

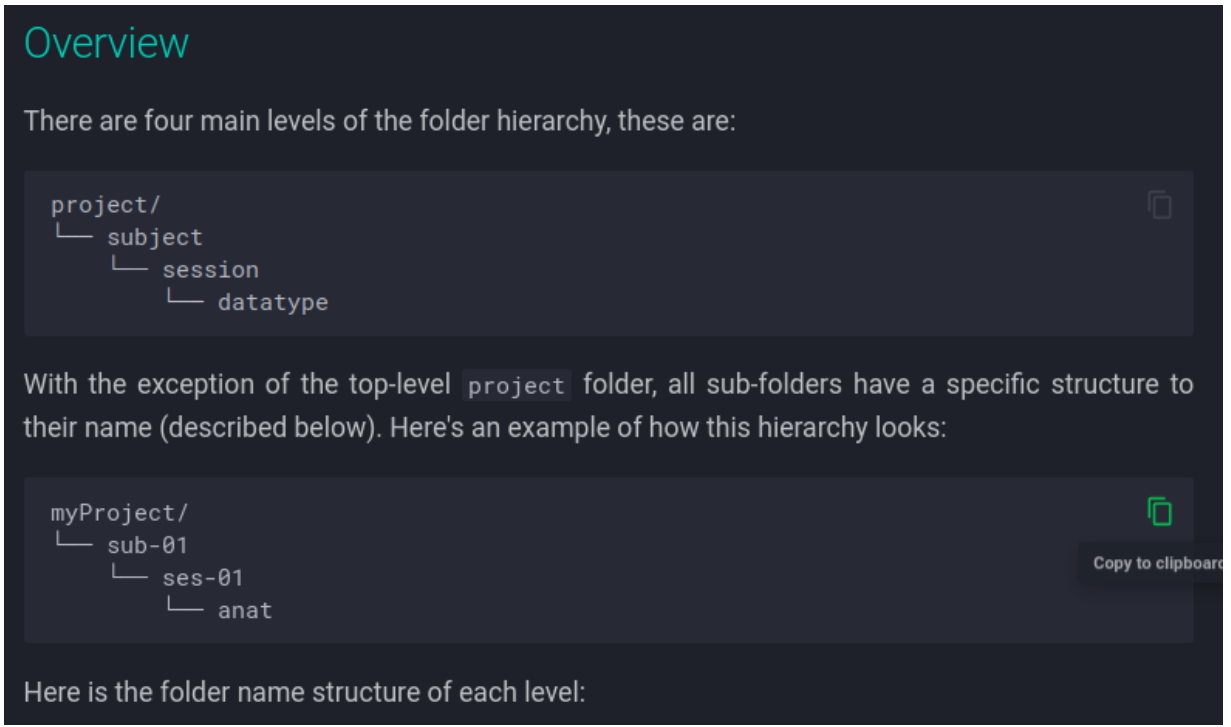
BIDS

a set of consensus-based rules to organize neuroimaging data

- Modularizes data
- Specifies a folder structure
- Names files in a human AND machine friendly way
- Uses standard interoperable file formats
- Documents metadata
- Minimizes duplication (inheritance principles)



BIDS folders



Overview

There are four main levels of the folder hierarchy, these are:

```
project/
└── subject
    └── session
        └── datatype
```

With the exception of the top-level `project` folder, all sub-folders have a specific structure to their name (described below). Here's an example of how this hierarchy looks:

```
myProject/
└── sub-01
    └── ses-01
        └── anat
```

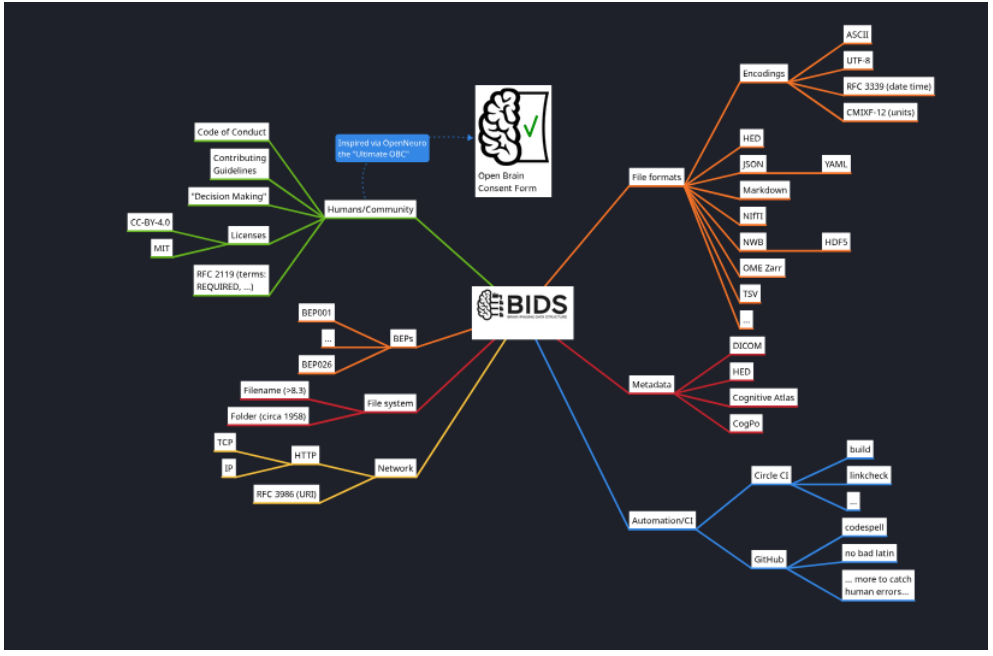
Here is the folder name structure of each level:

Separate folders for:
subject/[session]
and for datatypes:
anat
perf
dwi
func
beh
eeg
meg
ieeg
micr
pet

An example

```
ds001
├── dataset_description.json
├── participants.tsv
├── sub-01
│   ├── anat
│   │   ├── sub-01_inplaneT2.nii.gz
│   │   └── sub-01_T1w.nii.gz
│   └── func
│       ├── sub-01_task-balloonanalogrisktask_run-01_bold.nii.gz
│       ├── sub-01_task-balloonanalogrisktask_run-01_events.tsv
│       ├── sub-01_task-balloonanalogrisktask_run-02_bold.nii.gz
│       ├── sub-01_task-balloonanalogrisktask_run-02_events.tsv
│       ├── sub-01_task-balloonanalogrisktask_run-03_bold.nii.gz
│       └── sub-01_task-balloonanalogrisktask_run-03_events.tsv
├── sub-02
│   ├── anat
│   │   ├── sub-02_inplaneT2.nii.gz
│   │   └── sub-02_T1w.nii.gz
│   └── func
│       ├── sub-02_task-balloonanalogrisktask_run-01_bold.nii.gz
│       ├── sub-02_task-balloonanalogrisktask_run-01_events.tsv
│       ├── sub-02_task-balloonanalogrisktask_run-02_bold.nii.gz
│       ├── sub-02_task-balloonanalogrisktask_run-02_events.tsv
│       ├── sub-02_task-balloonanalogrisktask_run-03_bold.nii.gz
│       └── sub-02_task-balloonanalogrisktask_run-03_events.tsv
...
...
└── task-balloonanalogrisktask_bold.json
```

BIDS reuse standards and metadata ontologies

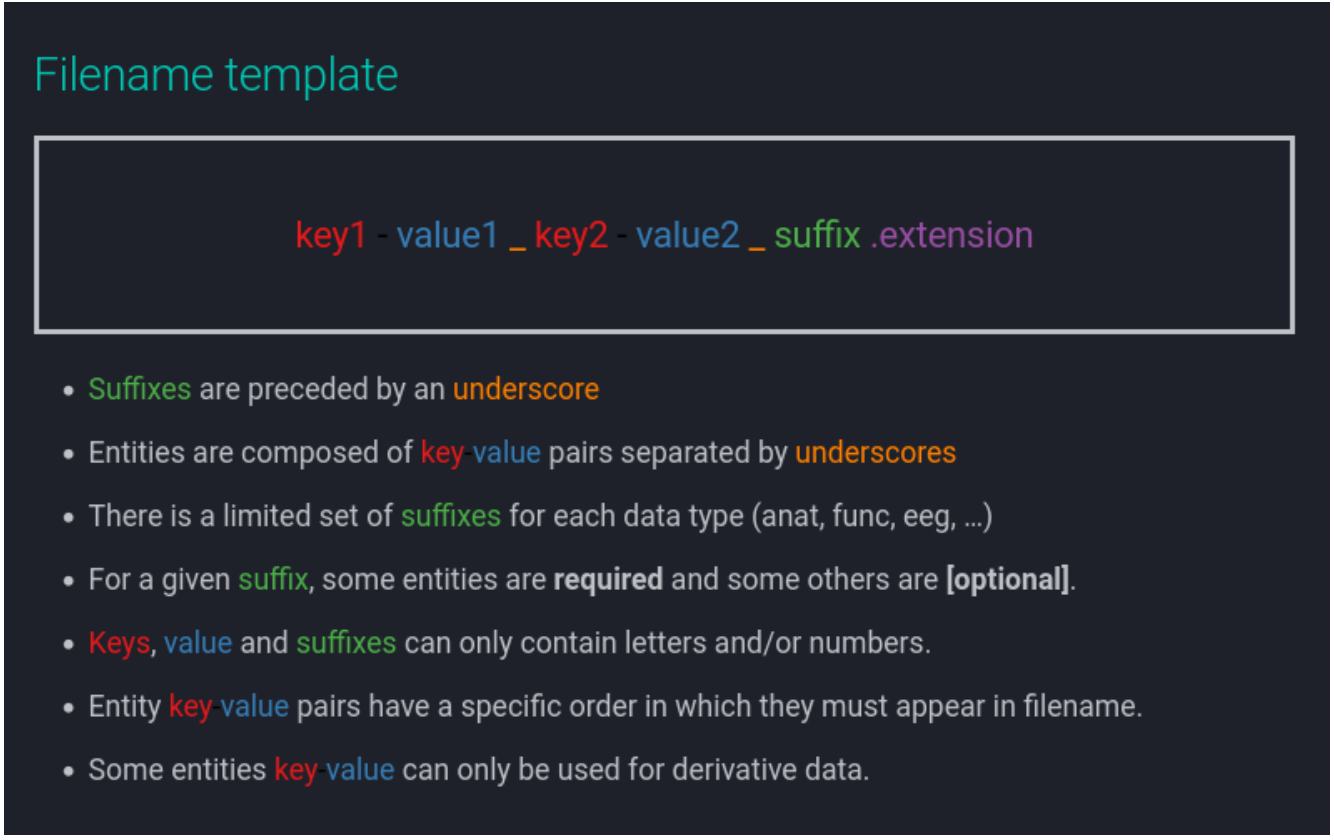


BIDS files

1. json files that contain key: value metadata

2. tsv files that contain tables of metadata

3. Raw data files (.fif for MEG)



BIDS MEG files

sub-<label>/
[ses-<label>/]
meg/
sub-<label>[_ses-<label>]_task-<label>[_acq-<label>][_run-<index>][_proc-<label>]_channels.json
sub-<label>[_ses-<label>]_task-<label>[_acq-<label>][_run-<index>][_proc-<label>]_channels.tsv
sub-<label>[_ses-<label>][_acq-<label>]_coordsystem.json
sub-<label>[_ses-<label>][_acq-<label>][_run-<index>][_proc-<label>][_space-<label>]_electrodes.json
sub-<label>[_ses-<label>][_acq-<label>][_run-<index>][_proc-<label>][_space-<label>]_electrodes.tsv
sub-<label>[_ses-<label>]_task-<label>[_acq-<label>][_run-<index>][_proc-<label>][_split-<index>]_meg.<extension>
sub-<label>[_ses-<label>]_task-<label>[_acq-<label>][_run-<index>][_proc-<label>][_split-<index>]_meg.json
sub-<label>[_ses-<label>]_acq-<calibration>_meg.dat
sub-<label>[_ses-<label>]_acq-<crosstalk>_meg.fif
sub-<label>[_ses-<label>][_acq-<label>]_headshape.*
sub-<label>[_ses-<label>][_acq-<label>]_headshape.pos
sub-<label>[_ses-<label>][_task-<label>][_acq-<label>][_space-<label>]_markers.mrk
sub-<label>[_ses-<label>][_task-<label>][_acq-<label>][_space-<label>]_markers.sqd
sub-<label>[_ses-<label>][_acq-<label>]_photo.jpg
sub-<label>[_ses-<label>][_acq-<label>]_photo.png
sub-<label>[_ses-<label>][_acq-<label>]_photo.tif
sub-<label>[_ses-<label>]_task-<label>[_acq-<label>][_run-<index>]_events.json
sub-<label>[_ses-<label>]_task-<label>[_acq-<label>][_run-<index>]_events.tsv
sub-<label>[_ses-<label>]_task-<label>[_acq-<label>][_run-<index>][_proc-<label>][_recording-<label>]_physio.json
sub-<label>[_ses-<label>]_task-<label>[_acq-<label>][_run-<index>][_proc-<label>][_recording-<label>]_physio.tsv.gz
sub-<label>[_ses-<label>]_task-<label>[_acq-<label>][_run-<index>][_proc-<label>][_recording-<label>]_stim.json
sub-<label>[_ses-<label>]_task-<label>[_acq-<label>][_run-<index>][_proc-<label>][_recording-<label>]_stim.tsv.gz

MNE-BIDS

An interface to BIDSify the data and interact along the analysis

It is all about mastering the *BIDSpath* object:

```python
# Create the BIDSPath object
bids_path = BIDSPath(
    subject=subject_id,
    session=session_id,
    task="SemanticScenes",
    run=run_id,
    root=bids_root,
    datatype='meg'
)
```

```python
# Write the BIDS data
write_raw_bids(
        raw,
        bids_path,
        empty_room=empty_room_raw,
        events=events,
        event_id=events_id,
        overwrite=True,
        format='auto')
```

MNE-BIDS

An interface to BIDSify the data and interact along the analysis

It is all about mastering the *BIDSpath* object:

```python
# Create the BIDSPath object
bids_path = BIDSPath(
    subject=subject_id,
    session=session_id,
    task="SemanticScenes",
    run=run_id,
    root=bids_root,
    datatype='meg'
)
```

```python
# Write the BIDS data
write_raw_bids(
        raw,
        bids_path,
        empty_room=empty_room_raw,
        events=events,
        event_id=events_id,
        overwrite=True,
        format='auto')
```

Let’s get out of the presentation and open the editor to code!

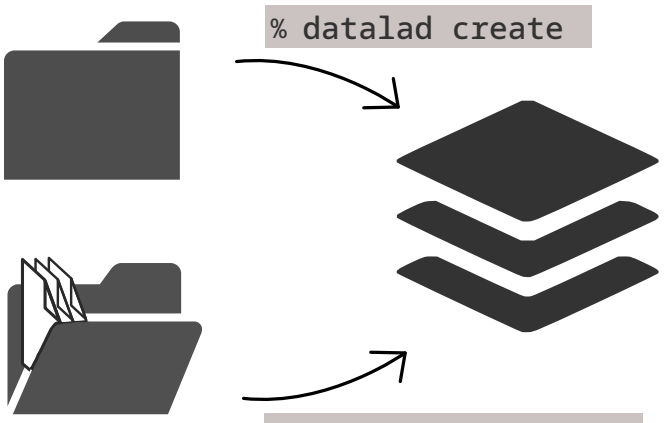
DataLad

Intro-duction

DataLad datasets

A dataset is a directory on a computer that DataLad manages

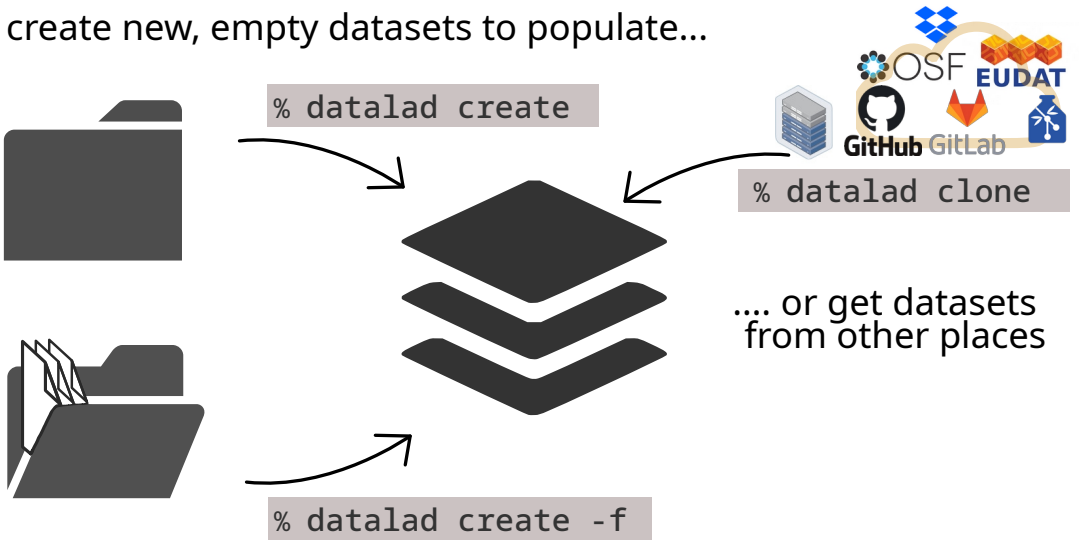
create new, empty datasets to populate...



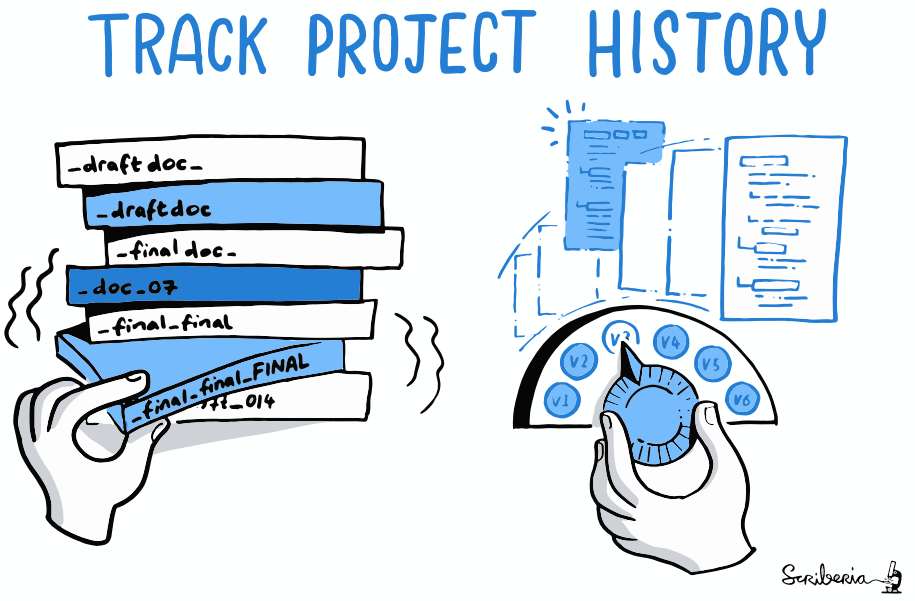
.... or transform existing directories into d

DataLad datasets

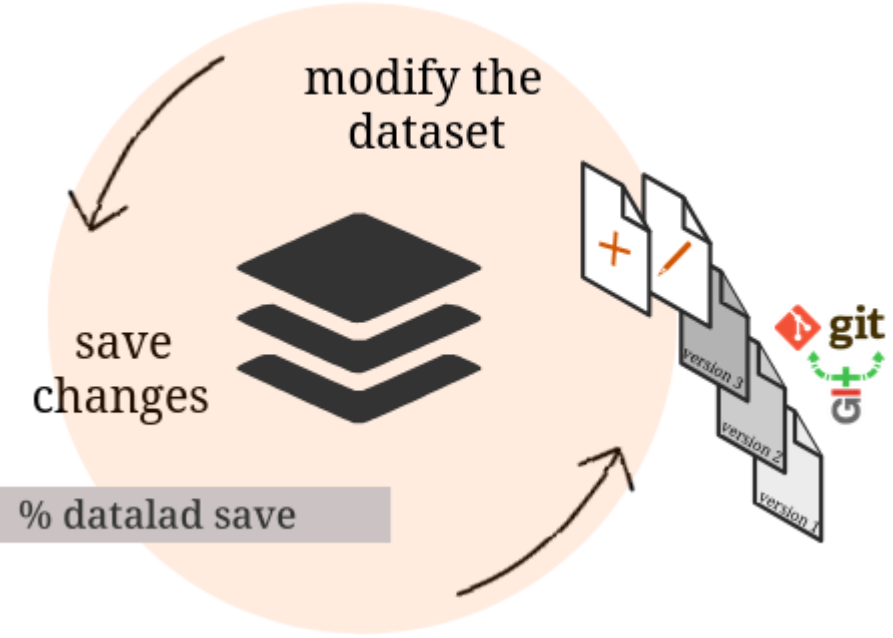
A dataset is a directory on a computer that DataLad manages



Version control

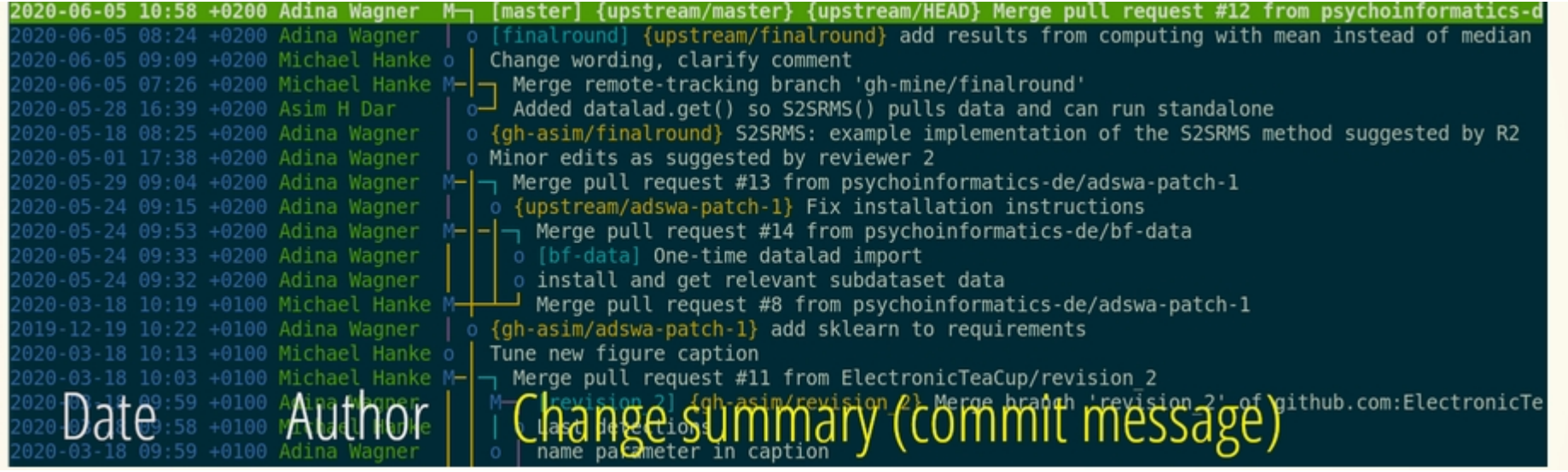


Data Version control (via git + git annex)



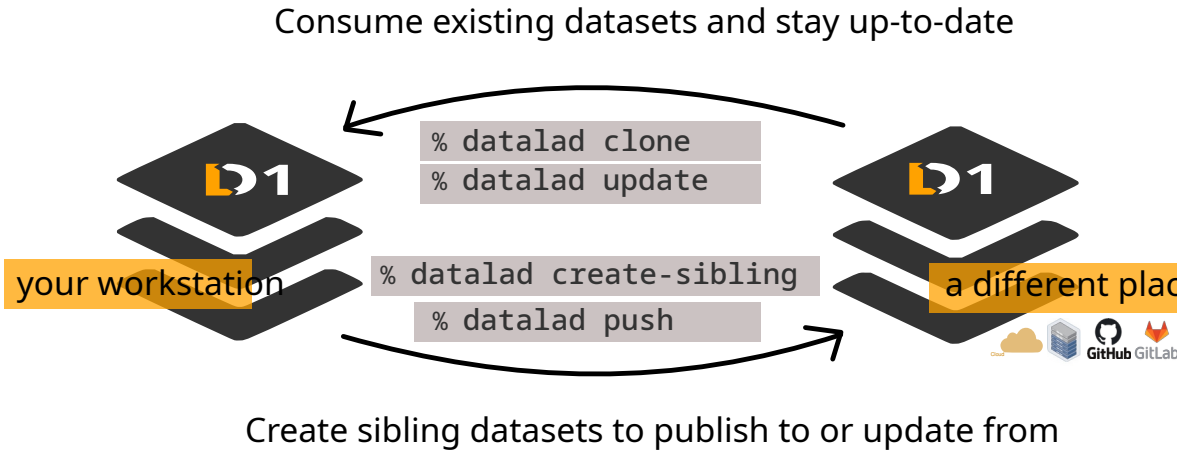

```
$ datalad save \
  -m "Adding raw data from neuroimaging study 1" \
  sub-*
  add(ok): sub-1/anat/T1w.json (file)
  add(ok): sub-1/anat/T1w.nii.gz (file)
  add(ok): sub-1/anat/T2w.json (file)
  add(ok): sub-1/anat/T2w.nii.gz (file)
  add(ok): sub-1/func/sub-1-run-1_bold.json (file)
  add(ok): sub-1/func/sub-1-run-1_bold.nii.gz (file)
  add(ok): sub-10/anat/T1w.json (file)
  add(ok): sub-10/anat/T1w.nii.gz (file)
  add(ok): sub-10/anat/T2w.json (file)
  add(ok): sub-10/anat/T2w.nii.gz (file)
 [110 similar messages have been suppressed]
  save(ok): . (dataset)
  action summary:
 add (ok: 120)
 save (ok: 1)
```

Data Version control (via git + git annex)

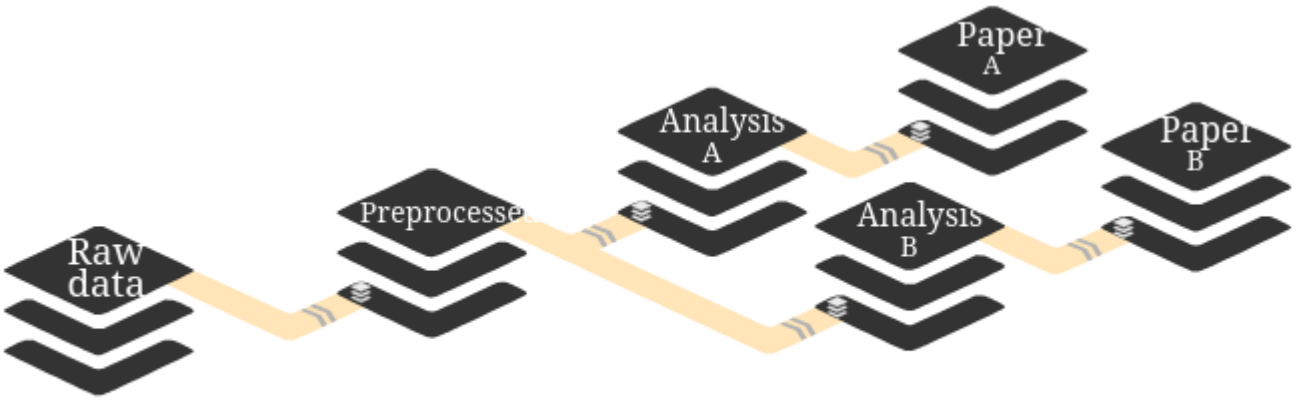


Consumption and collaboration

- Install existing datasets and update them from their sources
- create sibling datasets that you can publish updates to and pull updates from for collaboration and data sharing

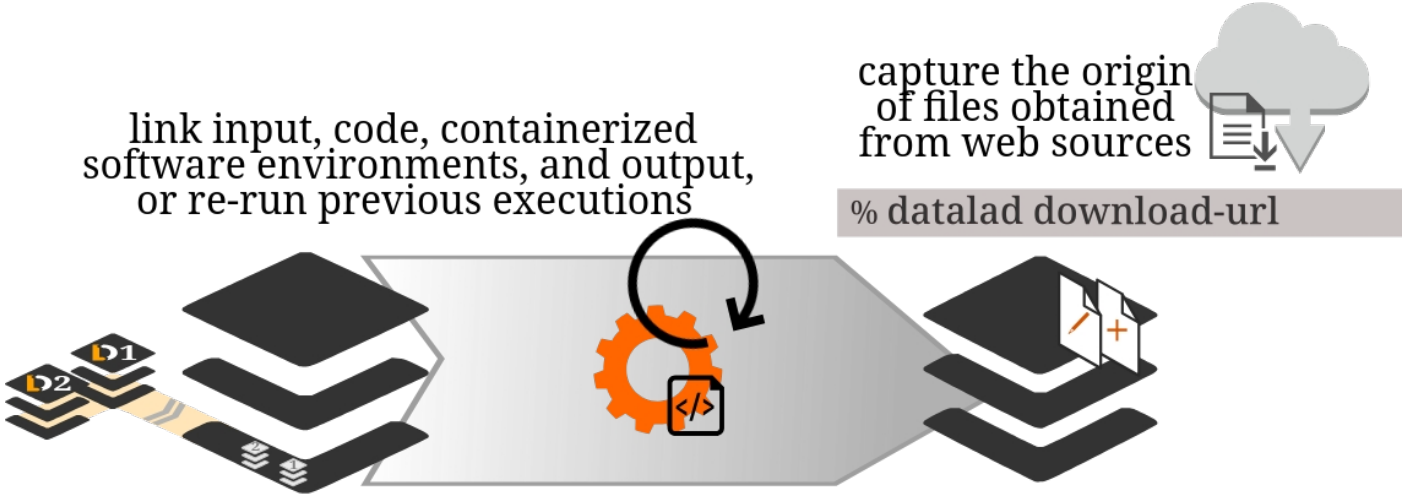


Dataset linkage



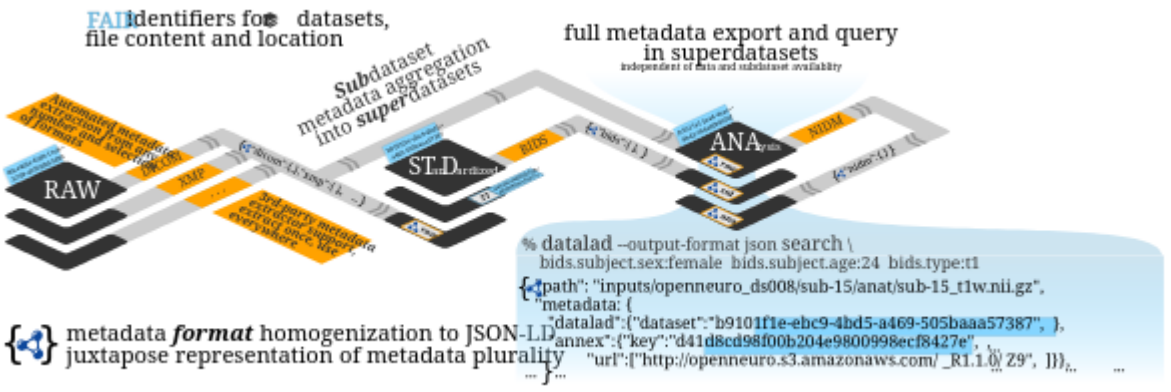
Nest modular datasets to create a linked hierarchy of datasets, and enable recursive operations throughout the hierarchy

Full provenance capture and reproducibility

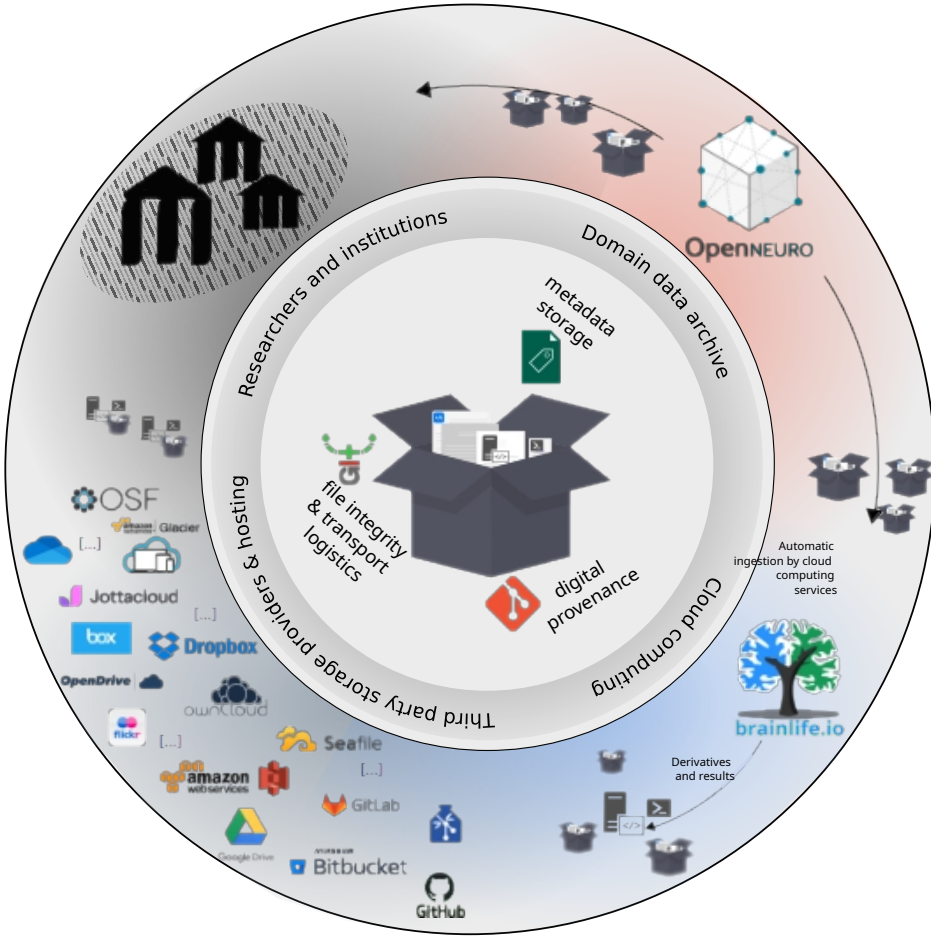


Third party service integration

Metadata handling



Wrap up


Researchers and institutions
Domain data archive
Cloud computing
Third party storage provideers & hosting
metadata storage
file integrity & transport logistics
digital provenance
OpenNEURO
Automatic ingestion by cloud computing services
brainlife.io
Derivatives and results
OSF
Glacier
Jottacloud
box
Dropbox
OpenDrive
ownCloud
flickr
Seafile
amazon web services
GitLab
Google Drive
Bitbucket
GitHub
[...]

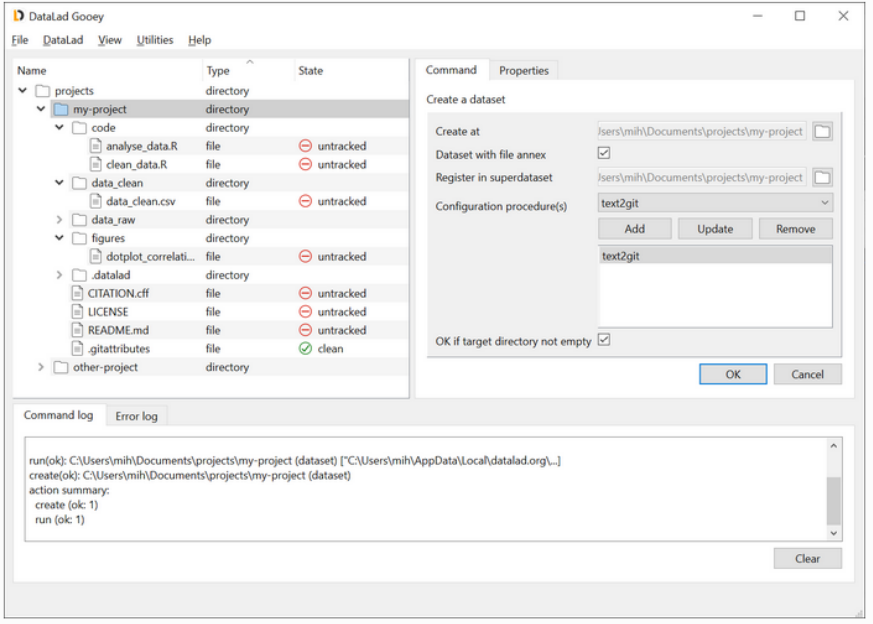
DataLad Tutorials, shall we?

- A typical collaborative data management workflow
- Basic provenance tracking
- Writing a reproducible paper
- Student supervision in a research project
- A basic automatically and computationally reproducible neuroimaging analysis
- An automatically and computationally reproducible neuroimaging analysis from scratch
- Scaling up: Managing 80TB and 15 million files from the HCP release
- Building a scalable data storage for scientific computing
- Using Globus as a data store for the Canadian Open Neuroscience Portal
- DataLad for reproducible machine-learning analyses
- Encrypted data storage and transport

DataLad Gooey :)

Welcome to DataLad Gooey's documentation!

DataLad Gooey is a Graphical User Interface (GUI) for using DataLad, a free and open source distributed data management tool. DataLad Gooey is compatible with all major operating systems and allows access to DataLad's operations via both a simplified and complete suite.



https://docs.datalad.org/projects/gooey/en/latest/