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
  - Ask questions any time!
  - Proper Zoom etiquette: 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:

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

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

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

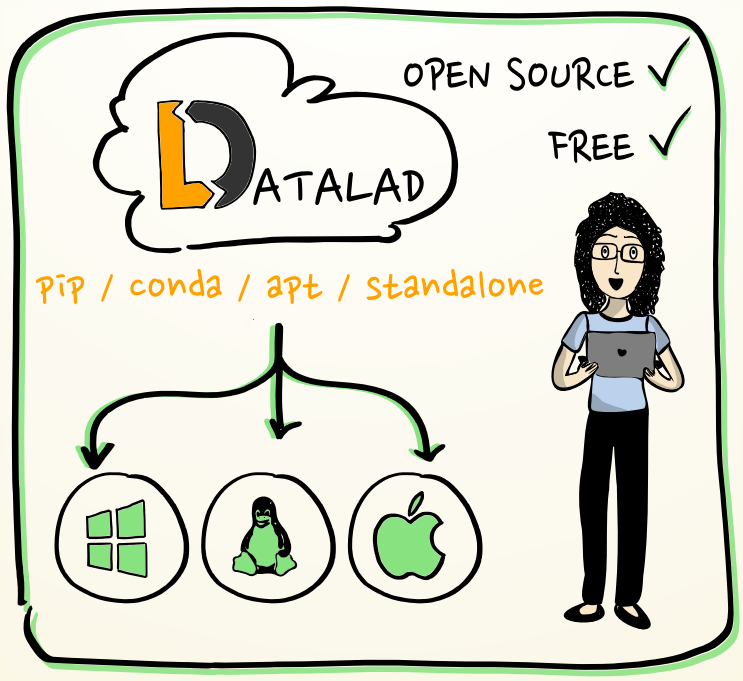
THIS IS: *DISTRIBUTED VERSION CONTROL*

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

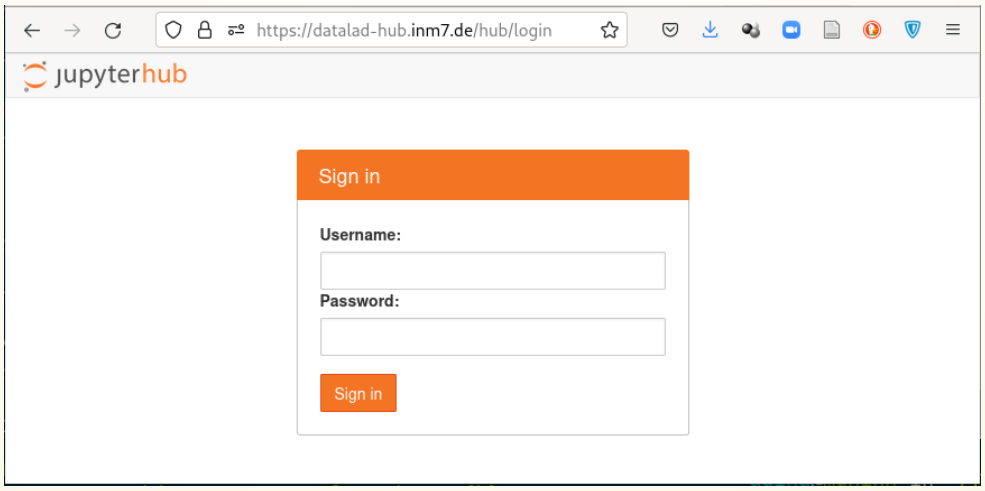
THIS IS: *DATA MANAGEMENT (WITH DATALAD 😀)*

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

LET'S TRY IT OUT



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 :)

DATALAD

Check the installed version:

```
datalad --version
```

copy

For help on using DataLad from the command line:

```
datalad --help
```

copy

For extensive info about the installed package, its dependencies, and extensions, use `wtf`:

```
datalad wtf
```

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

USING DATALAD VIA ITS PYTHON API

Open a Python environment:

```
ipython
```
copy

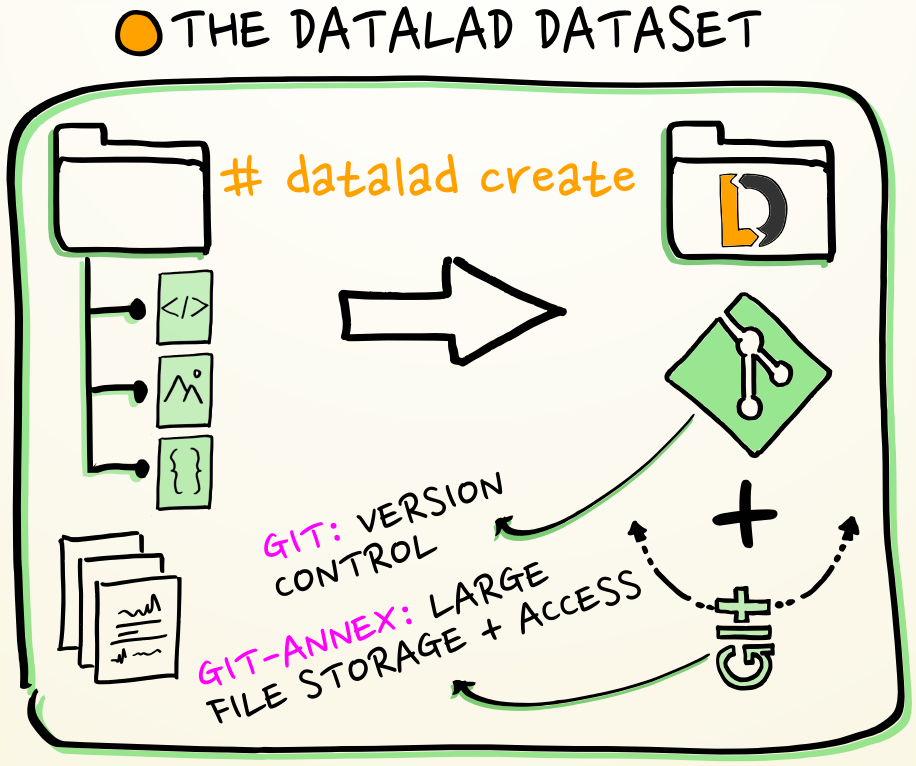
Import and start using:

```
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` config):

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

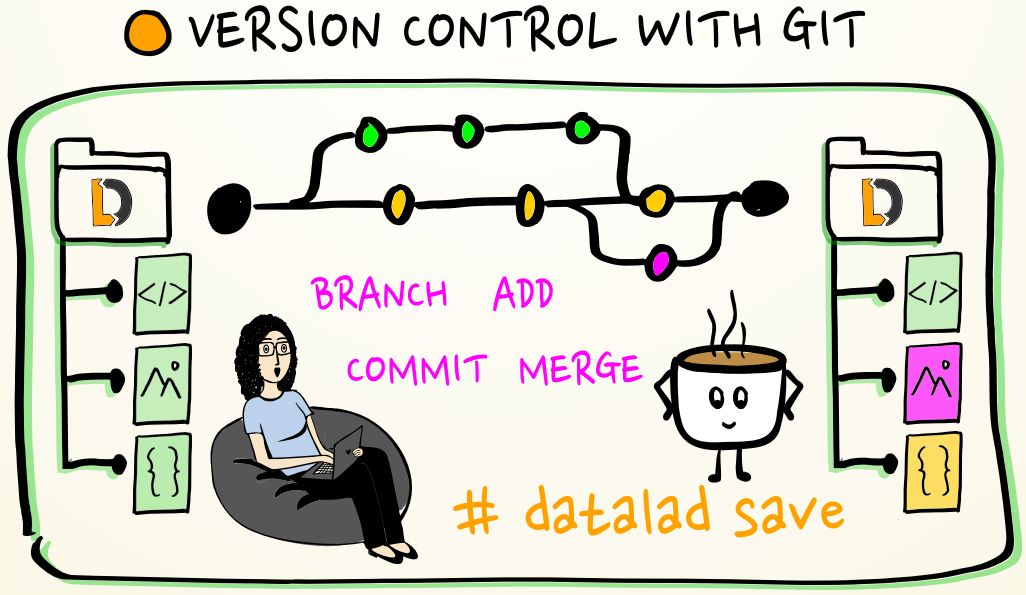
Let's have a look inside. Navigate using `cd` (change directory):

```
cd my-analysis
```
copy

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

```
ls -la .
```
copy

VERSION CONTROL...



...VERSION CONTROL

Let's add some Markdown text to a README file in the dataset

```
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 a short README"
```
copy

Further modifications:

```
echo "Contains a small data analysis for my project" >> README.md
```
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_an
```

copy

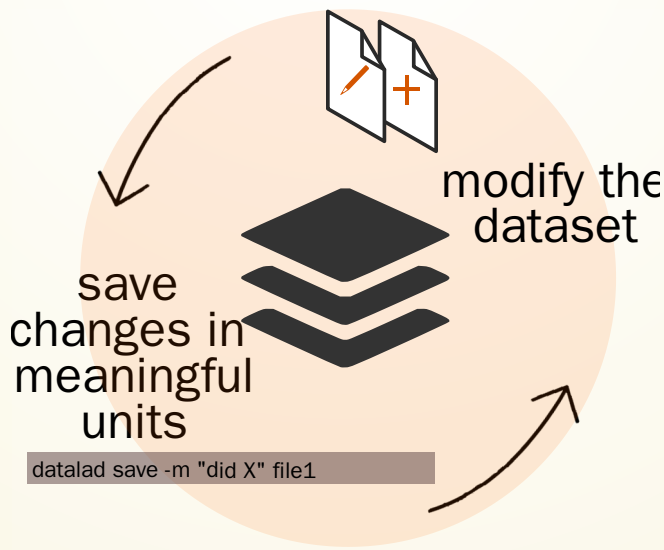
Check out the file's history:

```
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

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.

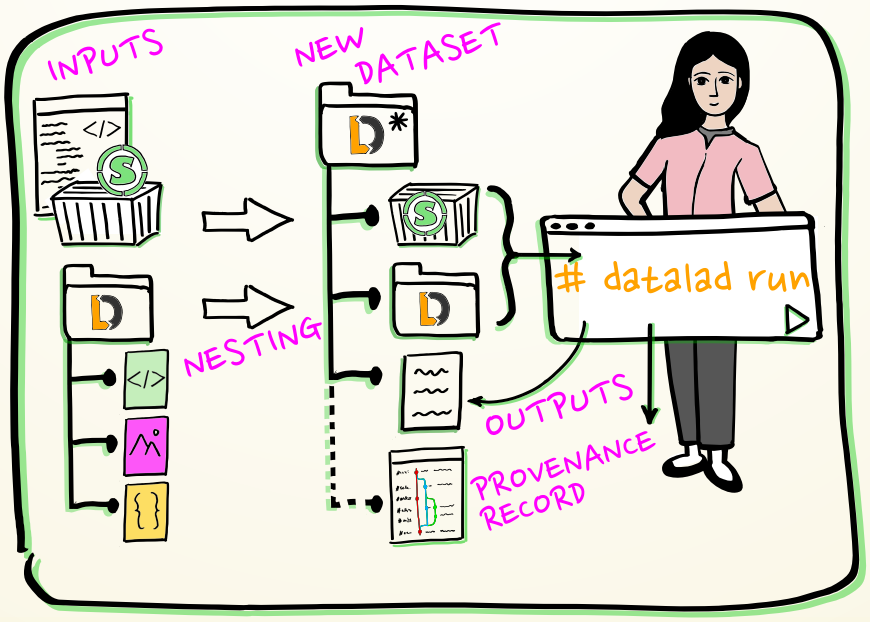
It even takes care of saving the change.

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

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

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
```

Version control makes changes transparent:

```
git diff
```

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

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

... and record precisely what we did

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

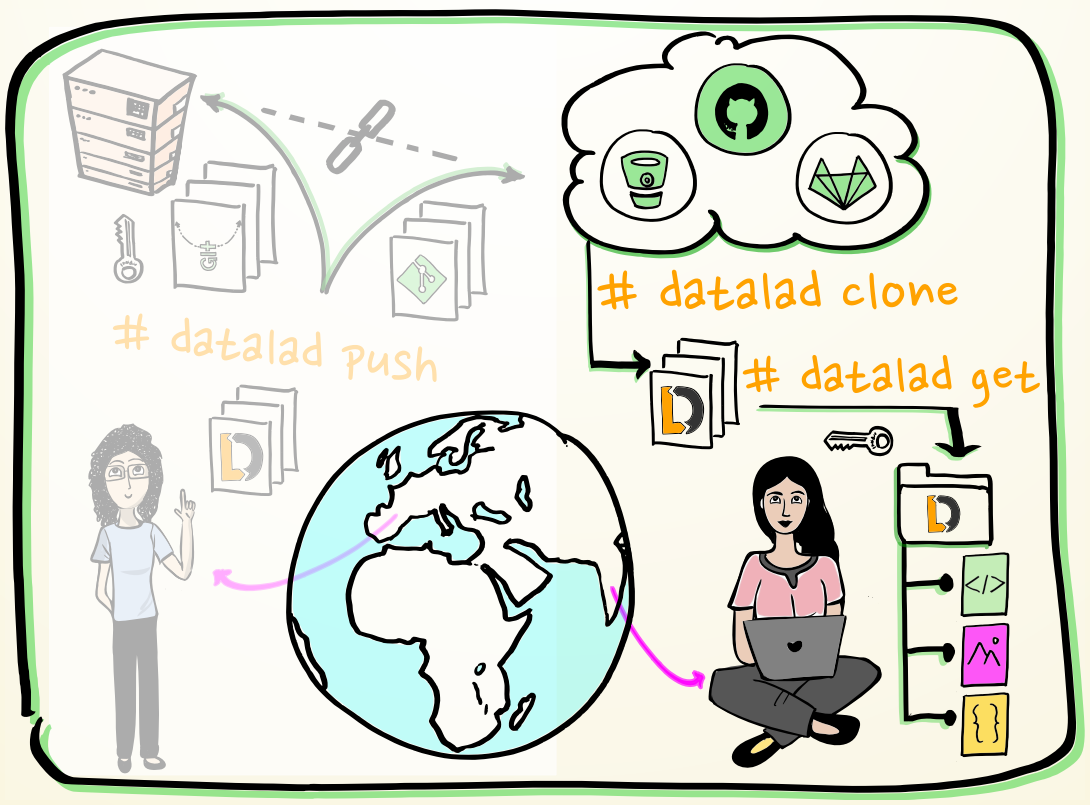
let's take a look:

```
git show
```

... and repeat!

```
datalad rerun
```

DATA CONSUMPTION & TRANSPORT...



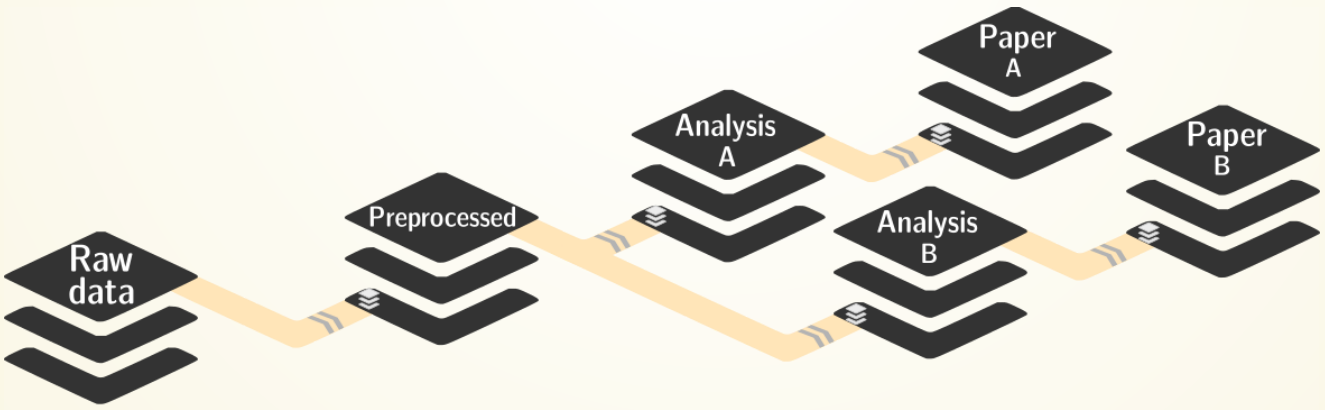
...DATA CONSUMPTION & TRANSPORT...

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
```

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
```



- 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!
We clone a dataset with data for an analysis into a specific location in the file tree of the existing dataset ("input"), 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

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:

```
du -sh # this will print the size of the directory in human readable sizes
```

copy

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

```
cat iris.csv
```

copy

Let's check the *actual* dataset size (i.e. Git repository + annexed content):

```
datalad status --annex
```

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
```

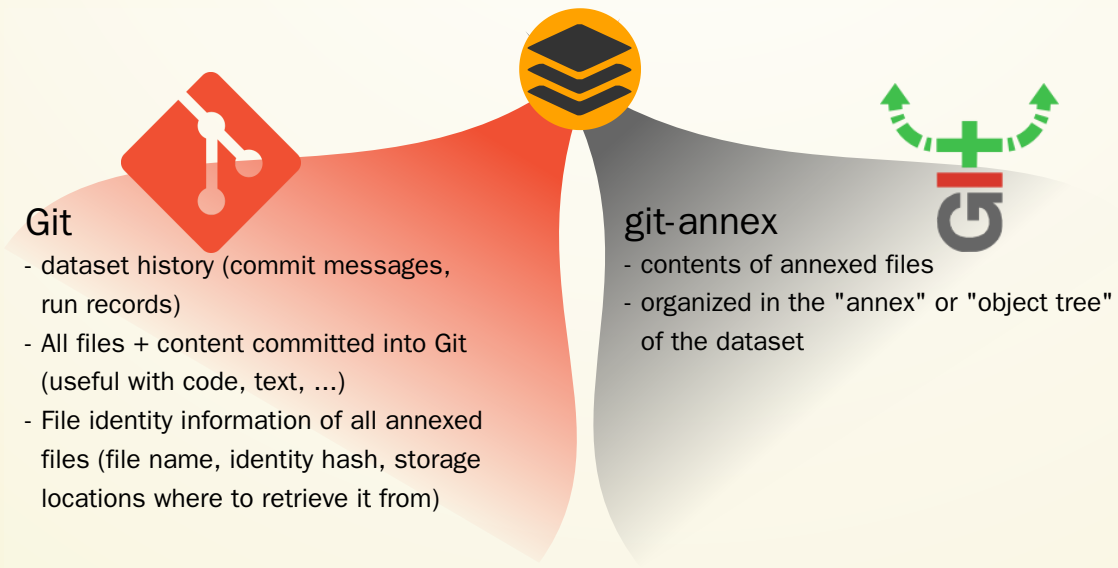
copy

Therefore: no need to store all files locally. Data just needs to be available from at least one location, then you can `get` what you want when you need it, and `drop` the rest.

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**

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.

...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

```
# back into my-analysis
cd ../
```
copy

Let's add a software container

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

inspect the list of registered containers:

```
datalad containers-list
```
copy

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

```
datalad containers-run -m "rerun analysis in container" \
  --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 `datalad diff` (based on `git diff`):

```
datalad 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:

1. Create a GIN user account and log in
2. Create and upload an SSH key to GIN
3. Create a new ***empty*** repository named "bids-data" and copy it's SSH URL

...PUBLISHING DATASETS

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

```
datalad create-sibling-gin examples-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 -d . \
https://gin.g-node.org/adswa/example-dataset \
myclone
```

copy

Recompute results:

```
cd myclone
datalad rerun
```

copy

BASIC ORGANIZATIONAL PRINCIPLES FOR DATASETS

Keep everything clean and modular

- An analysis is a superdataset, its components are subdatasets, and its structure modular

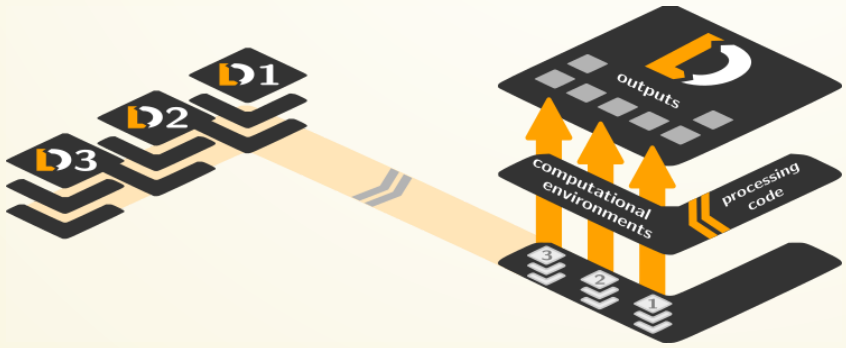
```
├── code/
│   ├── tests/
│   └── myscript.py
├── docs
│   ├── build/
│   └── source/
├── envs
│   └── Singularity
├── inputs/
│   └── data/
│       ├── dataset1/
│       │   └── datafile
│       └── dataset2/
│           └── datafile
├── outputs/
│   └── important_result
│       └── figures/
└── README.md
```

- do not touch/modify raw data: save any results/computations outside of input datasets
- Keep a superdataset self-contained: Scripts reference subdatasets or files with relative paths

BASIC ORGANIZATIONAL PRINCIPLES FOR DATASETS

Record where you got it from, where it is now, and what you do to it

- Link datasets (as subdatasets), record data origin
- Collect and store provenance of all contents of a dataset that you create



Document everything:

- Which script produced which output? From which data? In which software environment?

Find out more about organizational principles in the YODA principles!

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.

ACKNOWLEDGEMENTS

Funders

JÜLICH Forschungszentrum

DFG

EUROPEAN UNION
European Regional Development Fund

cbbs
center for behavioral brain sciences

SACHSEN-ANHALT
Ministerium für Wissenschaft und Wirtschaft

Collaborators

DataLad software & ecosystem

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

Human Brain Project

CONP PCNO

OpenNEURO

eBRAIN Health

MOTOR SFB

brainlife.io

cbrain

VirtualBrainCloud