YOUR TURN!

Use what you already know about how and where to get help to complete these challenges on https://datalad-hub.inm7.de or on your own system:

1. **Create** dataset, add a file with the content "abc". Check the **status** of the dataset. Now **save** the dataset with a **commit message**. Check the status again.
2. **Create** a different dataset *outside* the first one.
3. **Clone** the first dataset into the second under the name "input".
4. Use datalad to capture the provenance of a data transformation that converts the content of the file created at (1) to all-uppercase and saves it in the dataset from (2). Hint the command

   ```
   sh -c 'tr "a-z" "A-Z" < inputpath > outputpath'
   ```

   can convert text in this fashion.
5. Check the **status** of the dataset. Now let DataLad show you the change to the dataset that running the `tr` command made.

A GUIDED CODE-ALONG THROUGH DATALAD'S BASICS AND INTERNALS

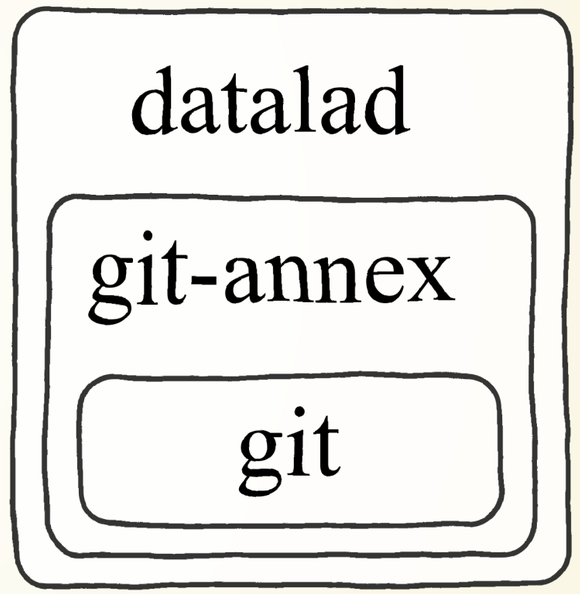
Code: psychoinformatics-de.github.io/rdm-course/01-content-tracking-with-datalad/index.html#getting-started-create-an-empty-dataset

DATALAD DATASETS

- DataLad's core data structure
  - Dataset = A directory managed by DataLad
  - Any directory of your computer can be managed by DataLad.
  - Datasets can be created (from scratch) or installed
  - Datasets can be nested: linked subdirectories

```
$ datalad create -c text2git my-dataset
```

DATALAD DATASETS



A DataLad dataset is a joined Git + git-annex repository

WHAT IS VERSION CONTROL?

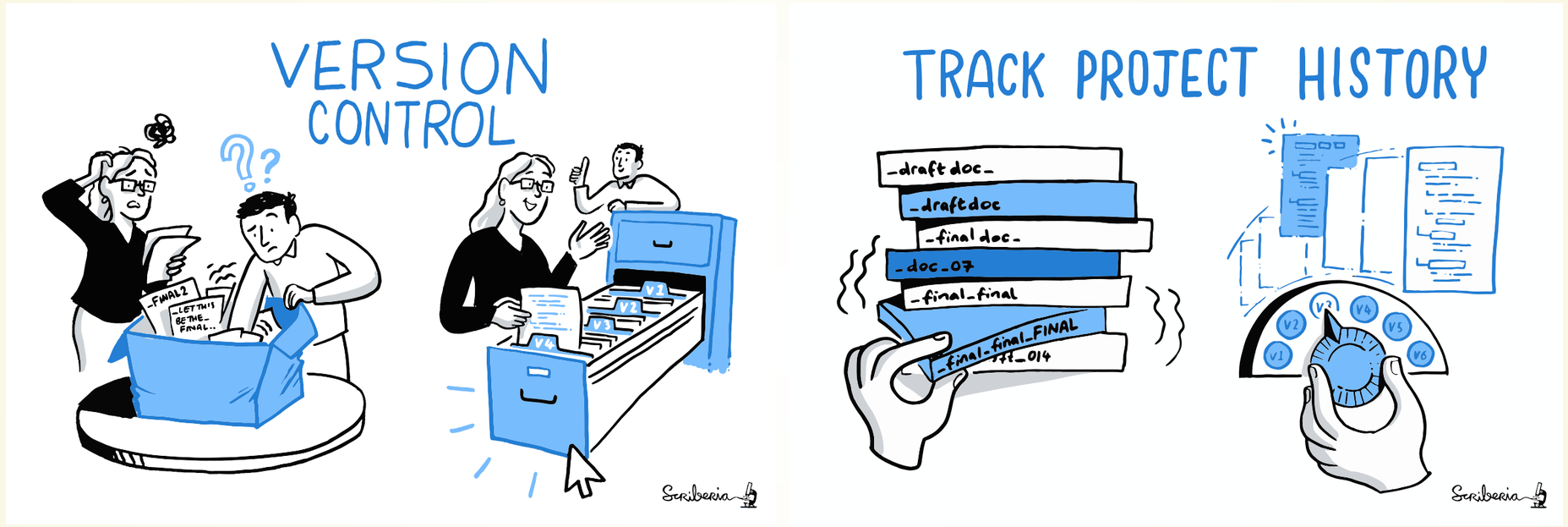
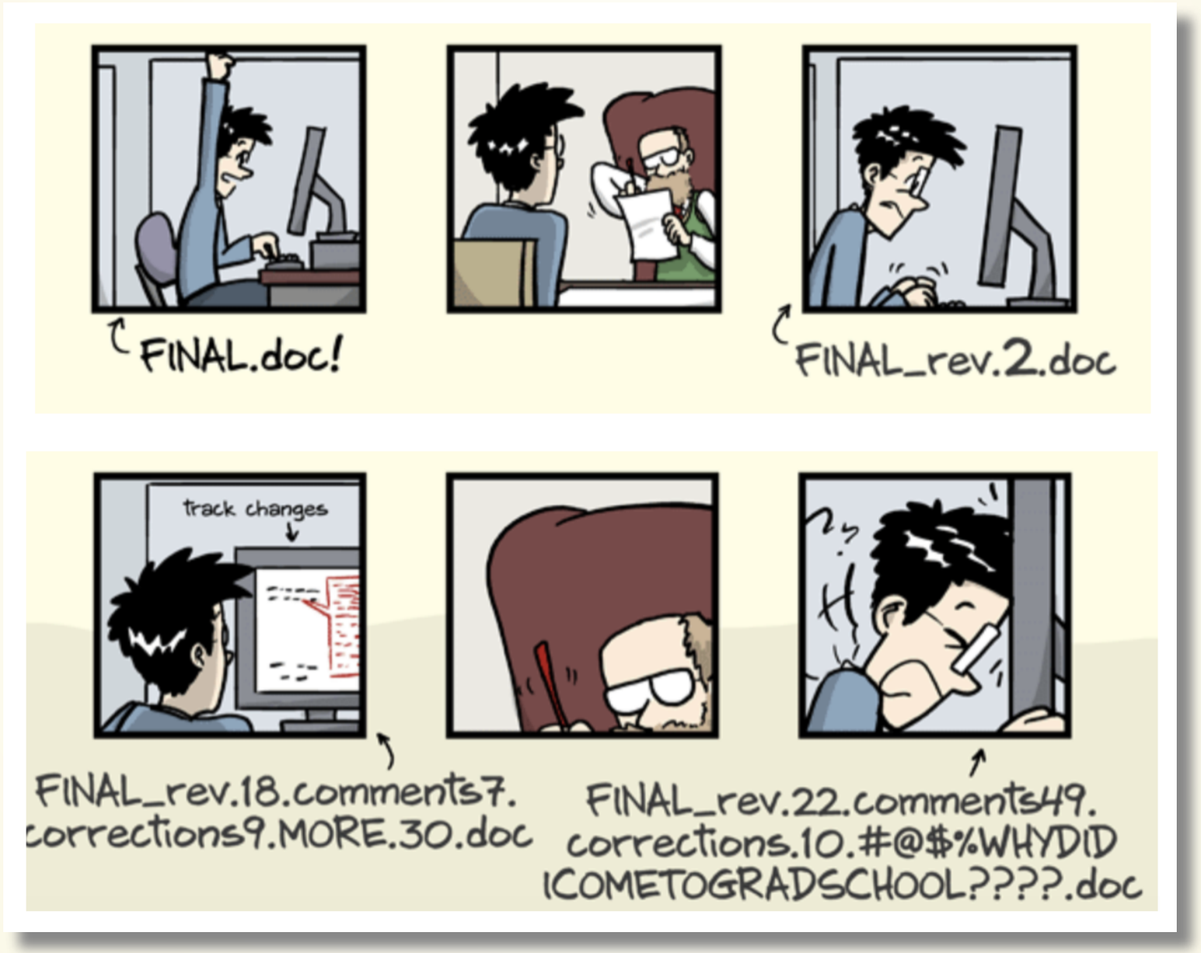


Image credit: Illustration adapted from Scriberia and The Turing Way

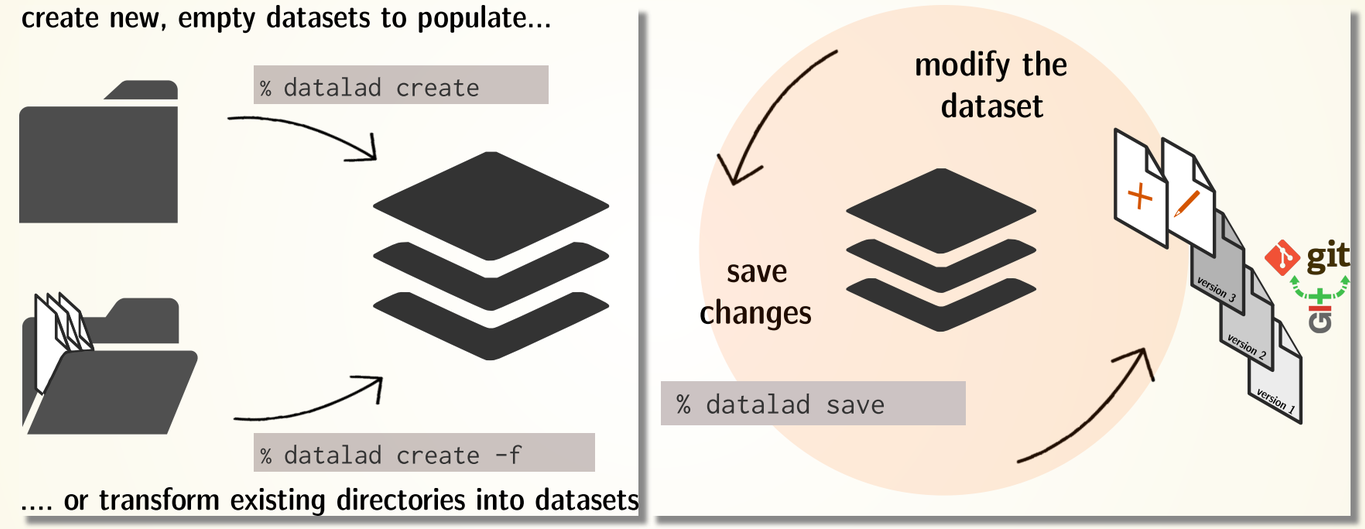
- keep things organized
- keep track of changes
- revert changes or go back to previous states

WHY VERSION CONTROL?



VERSION CONTROL

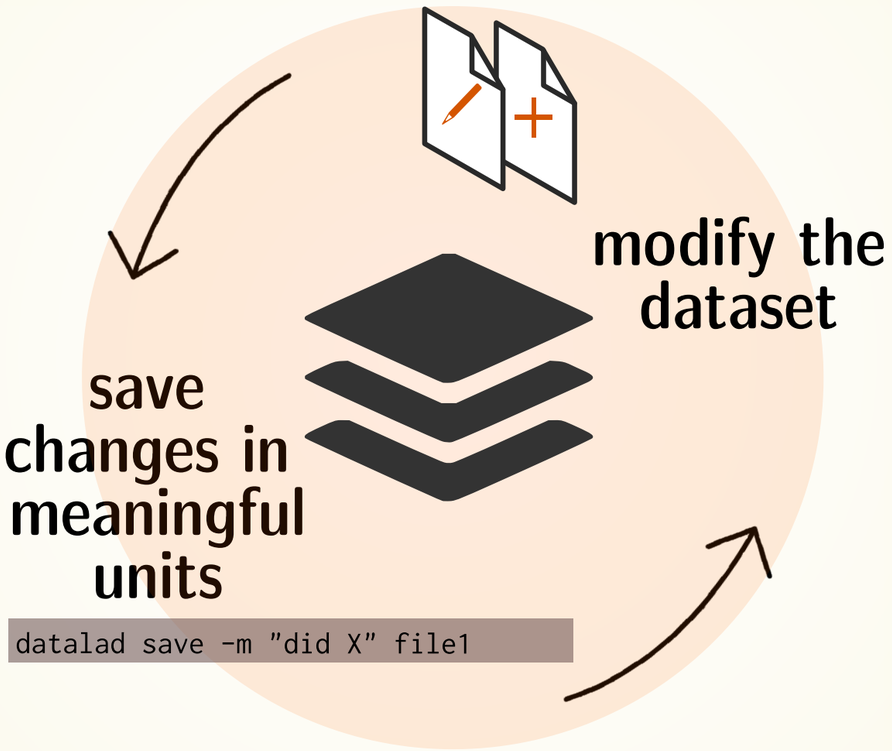
- DataLad knows two things: Datasets and files



- Every file you put into a in a dataset can be easily version-controlled, regardless of size, with the same command: *datalad save*
  - Pure Git/git-annex commands can be used as well

LOCAL VERSION CONTROL

Procedurally, version control is easy with DataLad!



Advice:

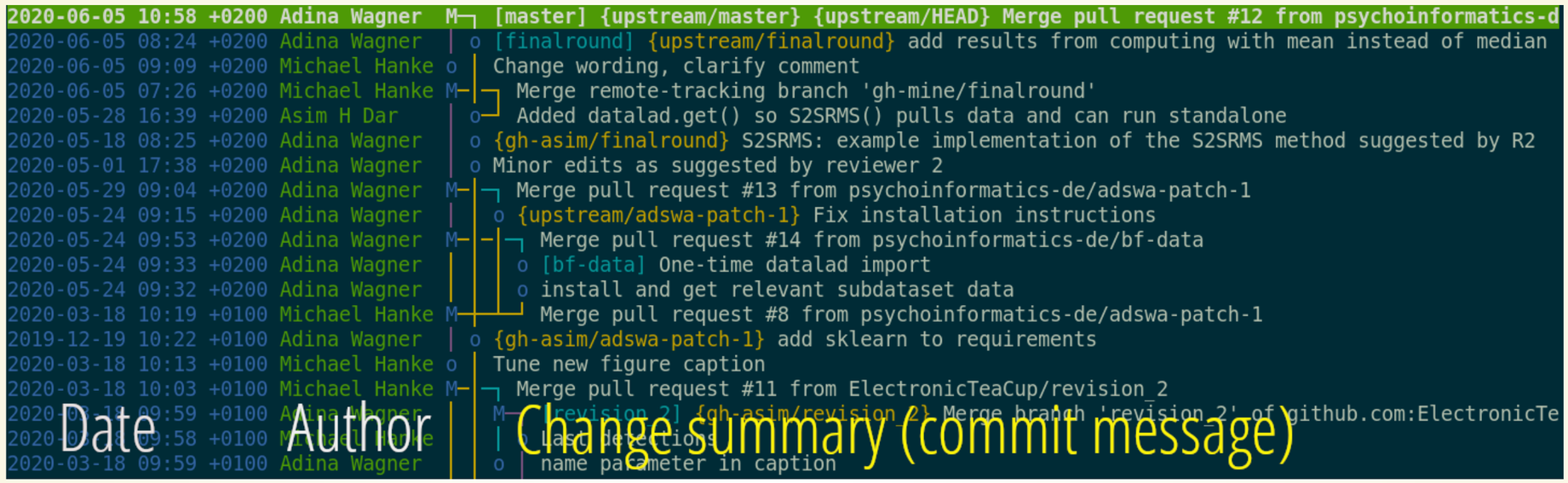
- Save meaningful units of change
- Attach helpful commit messages

THIS MEANS: YOU CAN ALSO VERSION CONTROL DATA!

```
$ 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)
```

VERSION CONTROL

- Your dataset can be a complete research log, capturing everything that was done, when, by whom, and how



- Interact with the history:
- reset your dataset (or subset of it) to a previous state,
- throw out changes or bring them back,
- find out what was done when, how, why, and by whom
- Identify precise versions: Use data in the most recent version, or the one from 2018, or...
- ...

PREVIEW: START TO RECORD PROVENANCE

- Have you ever saved a PDF to read later onto your computer, but forgot where you got it from?
- Digital Provenance = "The tools and processes used to create a digital file, the responsible entity, and when and where the process events occurred"
- The history of a dataset already contains provenance, but there is more to record - for example: Where does a file come from? `datalad download-url` is helpful

SUMMARY - LOCAL VERSION CONTROL

`datalad create` creates an empty dataset.

Configurations (**-c yoda**, **-c text2git**) are useful (details soon).

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.

QUESTIONS!

Awkward silence can be bridged with awkward MC questions :)

http://etc.ch/7YEk

TEASER: TIME-TRAVELLING

Code: psychoinformatics-de.github.io/rdm-course/01-content-tracking-with-datalad/index.html#breaking-things-and-repairing-them

Comprehensive walk-through handbook.datalad.org/basics/101-137-history.html

- Mistakes are not forever anymore: Past changes can transparently be undone
- Become a time-bender: Travel back in time or rewrite history
- Git's various identifiers:
  - Commit hash/Commit SHA: A 40-character string identifying each commit
  - Branch names, e.g., *main*
  - Tags, e.g., *v.0.1*
  - A pointer to the checked-out (current) commit on the current branch, *HEAD*

```
commit ac95f2fa94453e0e730f9183de7e37deb3f7b3df
Refs: [ux-exportfigshare], {gh-adswa/ux-exportfigshare}, 0.15.6-824-gac95f2fa9
Author:     Adina Wagner <adina.wagner@t-online.de>
AuthorDate: Mon Mar 14 10:47:42 2022 +0100
Commit:     Adina Wagner <adina.wagner@t-online.de>
CommitDate: Mon Mar 14 11:03:40 2022 +0100

    UX: Yield impossible result instead of RunTimeError in dirty ds

    This aims at standardizing result reporting/behavior over operations that
    refuse to operate in dirty datasets. export-to-figshare does it similarly
    to run now, and yields an impossible result. Fixes #6474

    Before:
    datalad export-to-figshare
    [ERROR  ] RuntimeError(Paranoid authors of DataLad refuse to proceed in a dir

    Now:
    datalad export-to-figshare
    export_to_figshare(impossible): /tmp/super (dataset) [clean dataset required
---
 datalad/distributed/export_to_figshare.py | 12 +++++++++---
```

SUMMARY: INTERACTING WITH GIT'S HISTORY (TEASER)

Interactions with Git's history require Git commands, but are immensely powerful

More in handbook.datalad.org/basics/101-137-history.html

`git restore` is a dangerous (!), but sometimes useful command:

It removes unsaved modifications to restore files to a past, saved state. What has been removed by it can not be brought back to life!

`git revert [hash]` transparently undoes a past commit

It will create a new entry in the revision history about this.

`git checkout`

lets you - among other things - time-travel.

Commands that are out of scope but useful to know:

`git rebase` changes and `git reset` rewinds history without creating a commit about it (see Handbook chapter for examples).

A life-saver that is not well-known: `git reflog`

A time-limited backlog of every past performed action, can undo every mistake except `git restore` and `git clean`.

QUESTIONS!

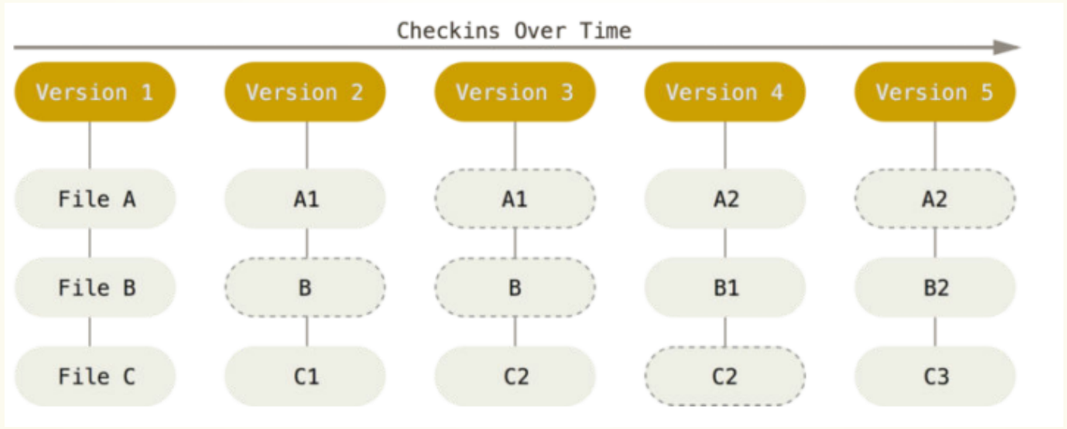
http://etc.ch/7YEk

A LOOK UNDERNEATH THE HOOD

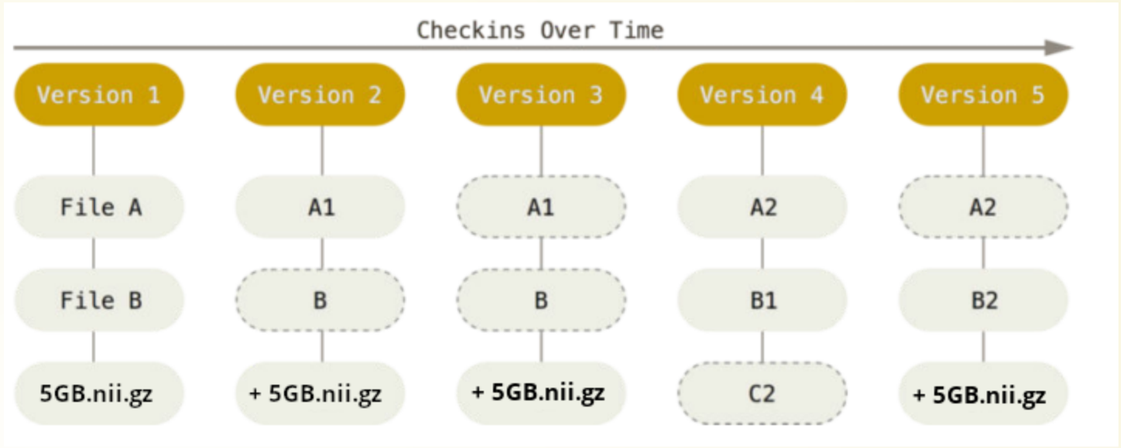
(IN-DEPTH EXPLANATIONS HOW AND WHY THINGS WORK, WITH PLENTY OF TEASERS TO ADDITIONAL FEATURES)

THERE ARE TWO VERSION CONTROL TOOLS AT WORK - WHY?

Git does not handle large files well.



THERE ARE TWO VERSION CONTROL TOOLS AT WORK - WHY?



Git does not handle large files well.

And repository hosting services refuse to handle large files:

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

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

CONSUMING DATASETS

- A dataset can be created from scratch/existing directories:

```
$ datalad create mydataset
[INFO] Creating a new annex repo at /home/adina/mydataset
create(ok): /home/adina/mydataset (dataset)
```

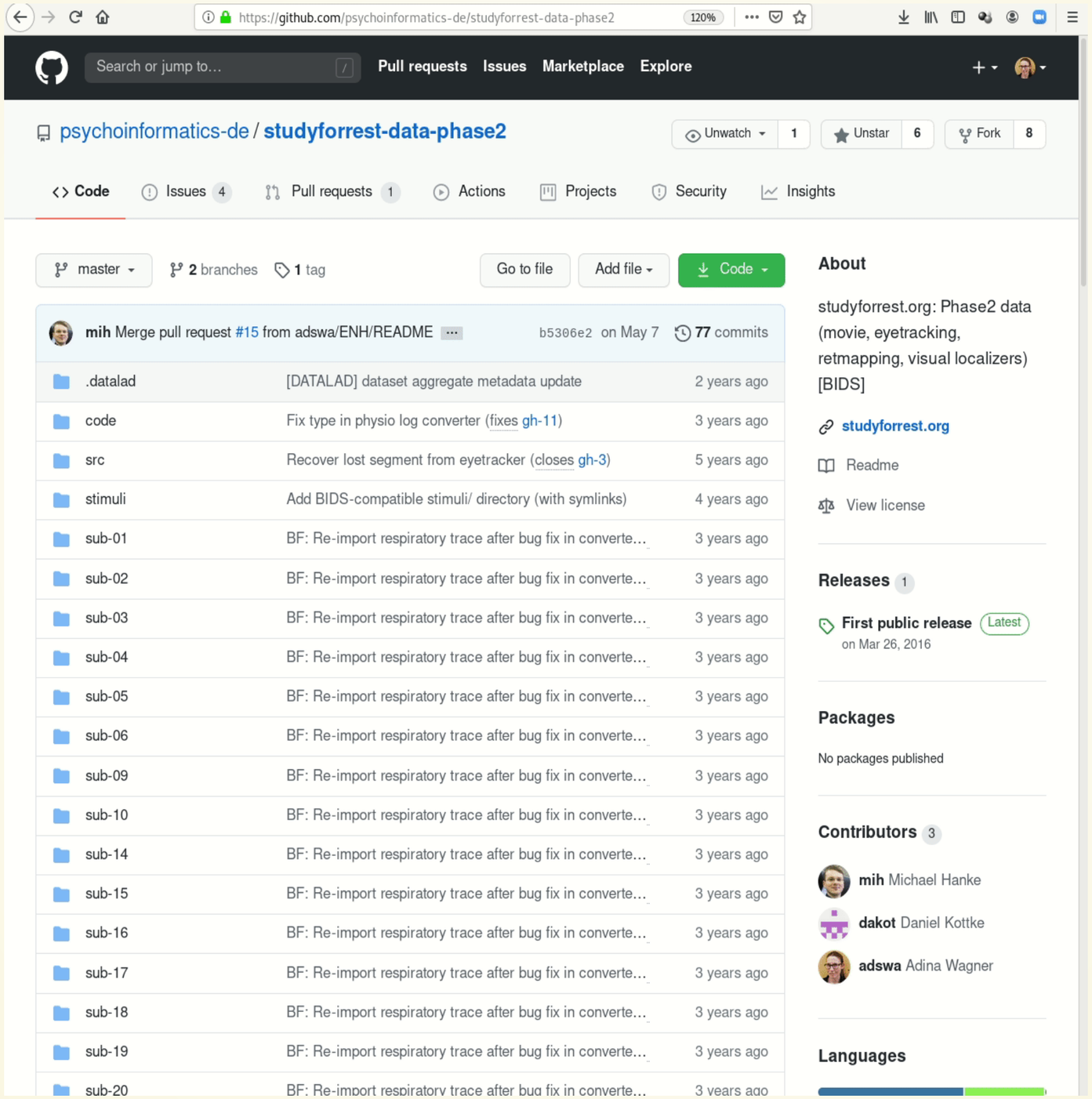
- but datasets can also be installed from paths or from URLs:

```
$ datalad clone https://github.com/datalad-datasets/human-connectome-project-openac
install(ok): /tmp/HCP (dataset)
```

Hint: Did you know that you can get the Human Connectome Project Open Access Data as a Dataset?

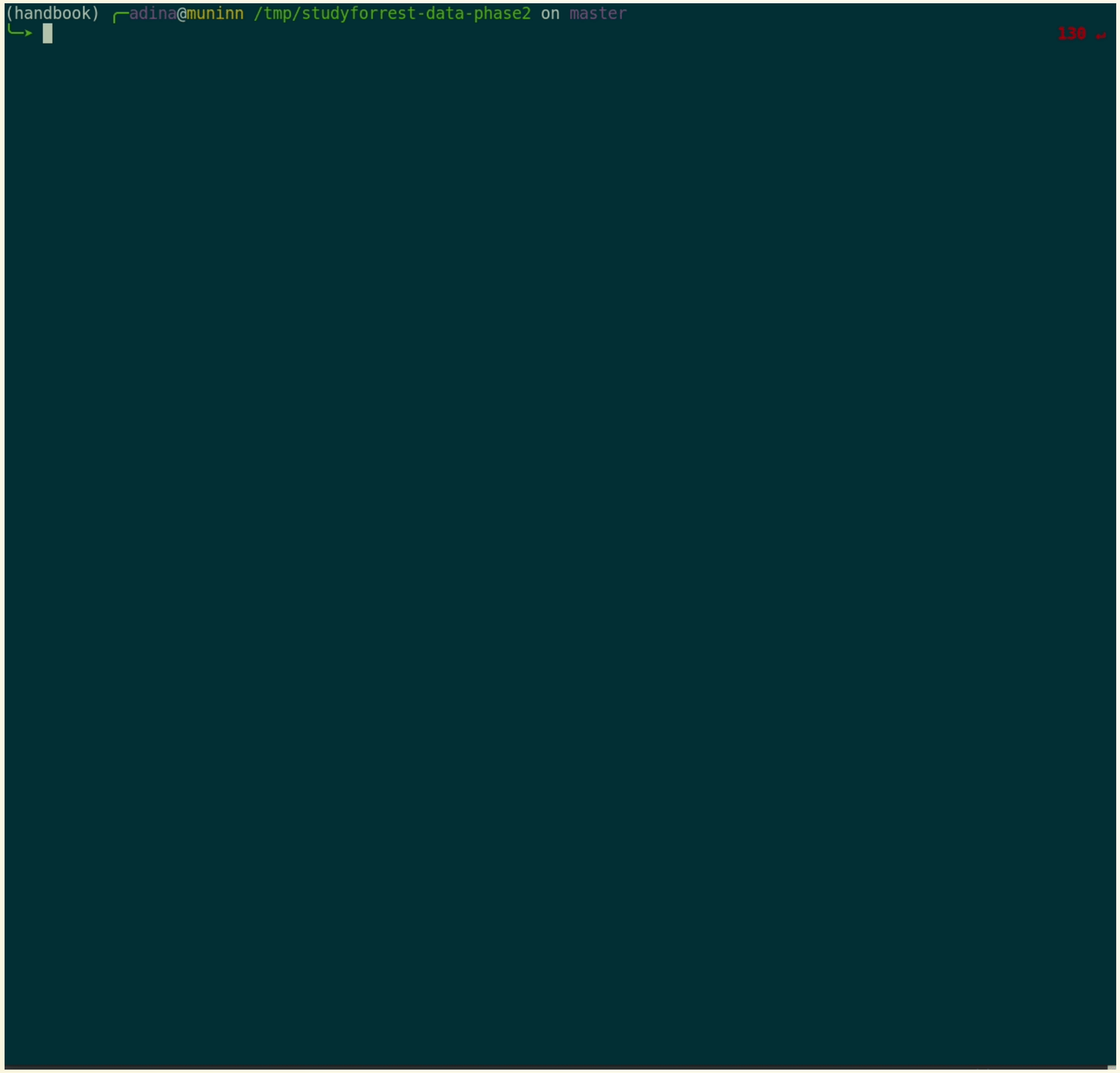
CONSUMING DATASETS

- Here's how to get a dataset:



CONSUMING DATASETS

- Here's how a dataset looks after installation:



Try it yourself with github.com/datalad-datasets/machinelearning-books.git

PLENTY OF DATA, BUT LITTLE DISK-USAGE

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

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

- files' contents can be retrieved on demand:

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

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

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

PLENTY OF DATA, BUT LITTLE DISK-USAGE

Drop file content that is not needed:

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

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

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

[really complex analysis]

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

GIT VERSUS GIT-ANNEX

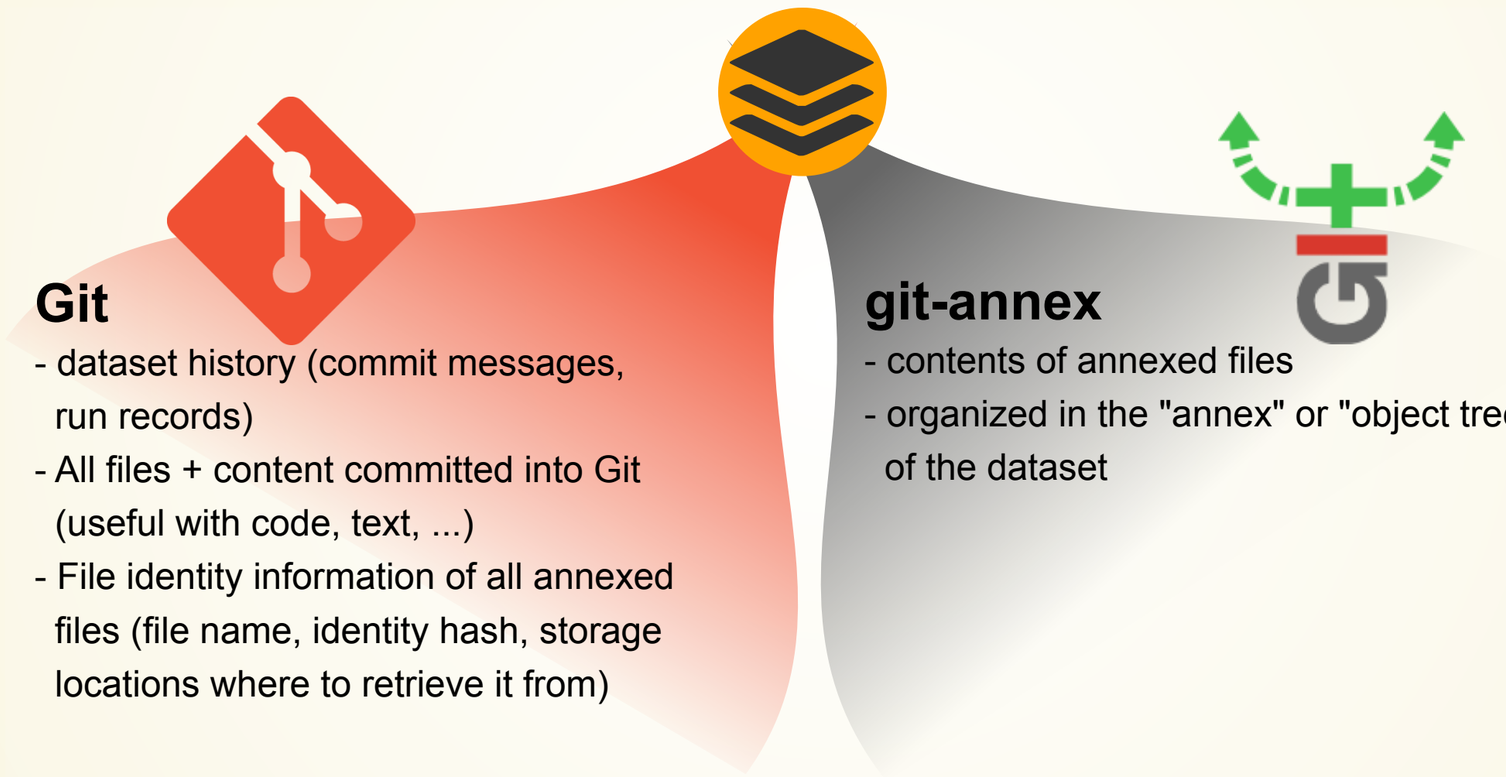
Data in datasets is either stored in Git or git-annex

By default, everything is annexed, i.e., stored in a dataset annex by git-annex & only content-identity is committed to Git.

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 • Files stored in Git are modifiable, files stored in Git-annex are content-locked • Annexed contents are not available right after cloning, only content identity and availability information (as they are stored in Git). Everything that is annexed needs to be retrieved with `datalad get` from whereever it is stored.	Useful: Large files, private files

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

GIT VERSUS GIT-ANNEX



Git

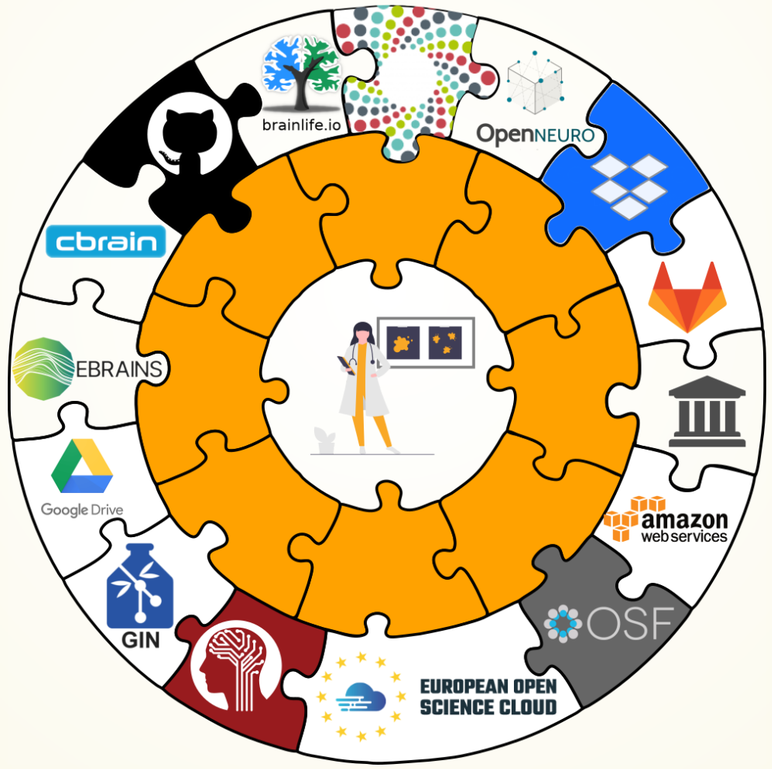
- dataset history (commit messages, run records)
- All files + content committed into Git (useful with code, text, ...)
- File identity information of all annexed files (file name, identity hash, storage locations where to retrieve it from)

git-annex

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

GIT VERSUS GIT-ANNEX

When sharing datasets with someone without access to the same computational infrastructure, annexed data is not necessarily stored together with the rest of the dataset (more tomorrow in the **session on publishing**).



Transport logistics exist to interface with all major storage providers. If the one you use isn't supported, let us know!

GIT VERSUS GIT-ANNEX

Users can decide which files are annexed:

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

TEXT VERSUS BINARY FILES

The *text2git* procedure affects text files. Can you identify them?

http://etc.ch/7YEk

An overview of text- versus binary files and implications for version control is in psychoinformatics-de.github.io/rdm-course/02-structuring-data/index.html#file-types-text-vs-binary

DISTRIBUTED AVAILABILITY

- git-annex conceptualizes file availability information as a decentral network. A file can exist in multiple different locations. *git annex whereis* tells you which are known:

```
$ git annex whereis inputs/images/chinstrap_02.jpg
whereis inputs/images/chinstrap_02.jpg (1 copy)
  00000000-0000-0000-0000-000000000001 -- web
  c1bfc615-8c2b-4921-ab33-2918c0cbfc18 -- adina@muninn:/tmp/my-dataset [here]

  web: https://unsplash.com/photos/8PxCm4HsPX8/download?force=true
ok
```

- If a file has no other known storage locations, *drop* will warn
  - Here is a file with a registered remote location (the web)

```
$ datalad drop inputs/images/chinstrap_02.jpg
drop(ok): /home/my-dataset/inputs/images/chinstrap_02.jpg (file)
$ datalad get inputs/images/chinstrap_02.jpg
get(ok): inputs/images/chinstrap_02.jpg (file)
```

  - Here is a file without a registered remote location (the web)

```
$ datalad drop inputs/images/chinstrap_01.jpg
drop(error): inputs/images/chinstrap_01.jpg (file)
             [unsafe; Could only verify the existence of 0 out of 1 necessary copy;
             (Use --reckless availability to override this check, or adjust numcopies.)]
```

Delineation and advantages of decentral versus central RDM: In defense of decentralized research data management

DATA PROTECTION

Why are annexed contents write-protected? (part I)

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

```
$ ls -l inputs/images/chinstrap_01.jpg
lrwxrwxrwx 1 adina adina 132 Apr  5 20:53 inputs/images/chinstrap_01.jpg -> ../../.git/annex/objects/1z/
xP/MD5E-s725496--2e043a5654cec96aadad554fda2a8b26.jpg/MD5E-s725496--2e043a5654cec96aadad554fda2a8b26.jpg
```

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

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

```
$ md5sum inputs/images/chinstrap_01.jpg
2e043a5654cec96aadad554fda2a8b26  inputs/images/chinstrap_01.jpg
```

- git-annex write-protects files to keep this symlink functional - Changing file contents without git-annex knowing would make the hash change and the symlink point to nothing
- To (temporarily) remove the write-protection one can *unlock* the file

DETOUR & TEASER: REPRODUCIBLE DATA ANALYSIS

Your past self is the worst collaborator:

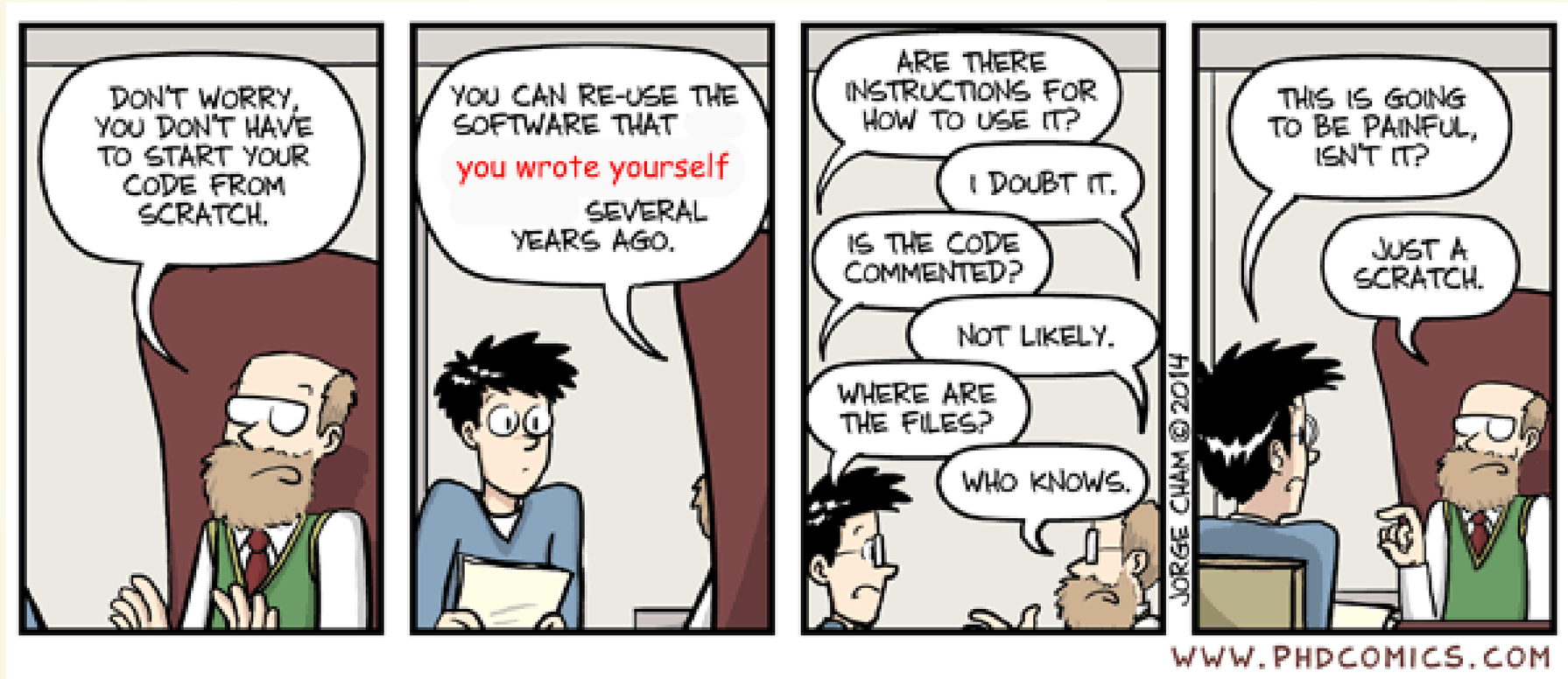


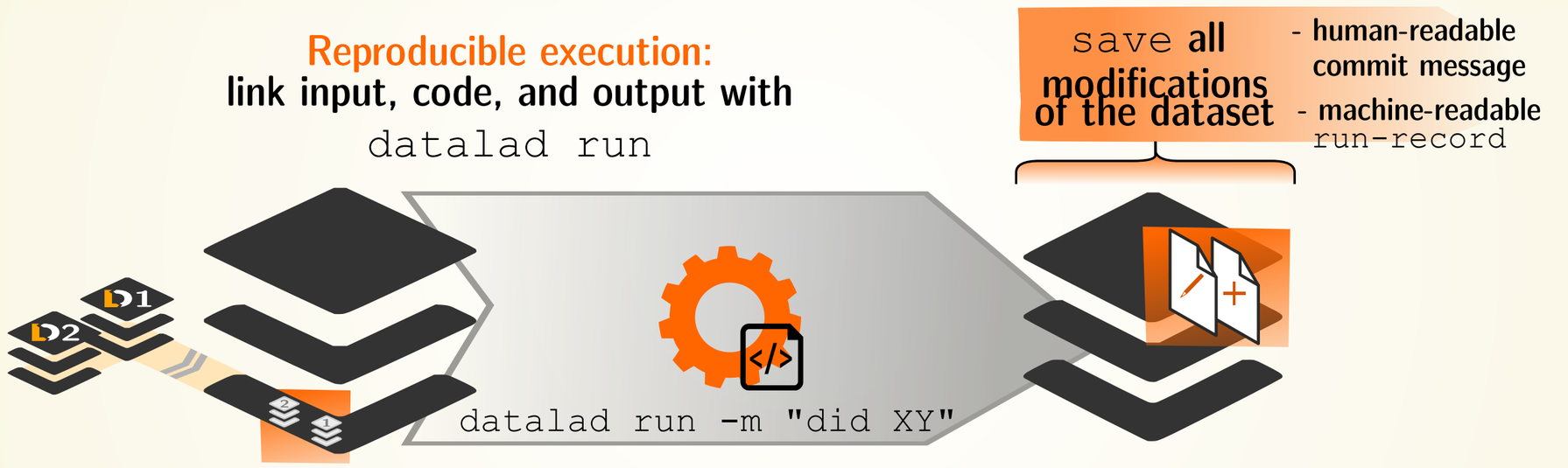
Image credit: Full comic at http://phdcomics.com/comics.php?f=1979

Code: psychoinformatics-de.github.io/rdm-course/01-content-tracking-with-datalad/index.html#data-processing

REPRODUCIBLE EXECUTION & PROVENANCE CAPTURE

datalad run wraps a command execution and records its impact on a dataset.

Reproducible execution:
link input, code, and output with
`datalad run`



REPRODUCIBLE EXECUTION & PROVENANCE CAPTURE

datalad run wraps a command execution and records its impact on a dataset.

```
commit 9fbc0c18133aa07b215d81b808b0a83bf01b1984 (HEAD -> main)
Author: Adina Wagner [adina.wagner@t-online.de]
Date:   Mon Apr 18 12:31:47 2022 +0200

    [DATALAD RUNCMD] Convert the second image to greyscale

    === Do not change lines below ===
    {
     "chain": [],
     "cmd": "python code/greyscale.py inputs/images/chinstrap_02.jpg outputs/im>
     "dsid": "418420aa-7ab7-4832-a8f0-21107ff8cc74",
     "exit": 0,
     "extra_inputs": [],
     "inputs": [],
     "outputs": [],
     "pwd": "."
    }
    ^^^ Do not change lines above ^^^

diff --git a/outputs/images_greyscale/chinstrap_02_grey.jpg b/outputs/images_gr>
new file mode 120000
index 0000000..5febc72
--- /dev/null
+++ b/outputs/images_greyscale/chinstrap_02_grey.jpg
@@ -0,0 +1 @@
+../../.git/annex/objects/19/mp/MD5E-s758168--8e840502b762b2e7a286fb5770f1ea69.>
\ No newline at end of file
```

The resulting commit's hash (or any other identifier) can be used to automatically re-execute a computation (more on this tomorrow)

DATA PROTECTION

Why are annexed contents write-protected? (part 2)

- When you try to modify an annexed file without unlocking you will see "Permission denied" errors.

```
Traceback (most recent call last):
  File "/home/bob/Documents/rdm-warmup/example-dataset/code/greyscale.py", line 20, in module
    grey.save(args.output_file)
  File "/home/bob/Documents/rdm-temporary/venv/lib/python3.9/site-packages/PIL/Image.py", line 2232, in save
    fp = builtins.open(filename, "w+b")
PermissionError: [Errno 13] Permission denied: 'outputs/images_greyscale/chinstrap_02_grey.jpg'
```

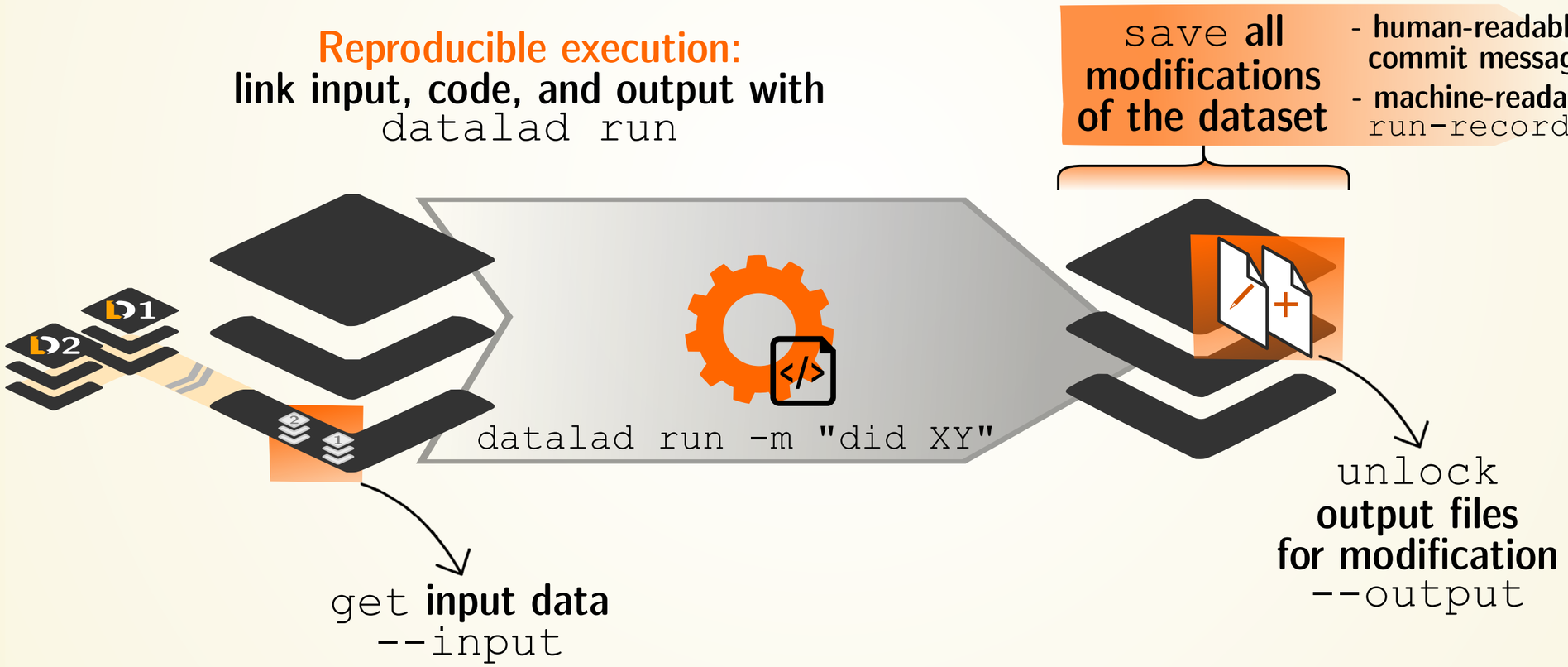
- Use *datalad unlock* to make the file modifiable. Underneath the hood (given the file system initially supported symlinks), this removes the symlink:

```
$ datalad unlock outputs/images_greyscale/chinstrap_02_grey.jpg
$ ls outputs/images_greyscale/chinstrap_02_grey.jpg
-rw-r--r-- 1 adina adina 758168 Apr 18 12:31 outputs/images_greyscale/chinstrap_02_grey.jpg
```

- *datalad save* locks the file again. Locking and unlocking ensures that git-annex always finds the right version of a file.

REPRODUCIBLE EXECUTION & PROVENANCE CAPTURE

datalad run wraps a command execution and records its impact on a dataset.
In addition, it can take care of data retrieval and unlocking



DATALAD RERUN

- `datalad rerun` is helpful to spare others and yourself the short- or long-term memory task, or the forensic skills to figure out how you performed an analysis
- But it is also a digital and machine-reable provenance record
- Important: The better the run command is specified, the better the provenance record
- Note: run and rerun only create an entry in the history if the command execution leads to a change.

- Task: Use `datalad rerun` to rerun the script execution. Find out if the output changed

SUMMARY - UNDERNEATH THE HOOD

Files are either kept in Git or in git-annex.

datalad save is used for both, but configurations (e.g., *text2git*), dataset rules (e.g., in a *.gitattributes* file, or flags change the default behavior of annexing everything

Annexed files behave differently from files kept in Git:

They can be retrieved and dropped from local or remote locations, they are write-protected, their content is unkown to Git (and thus easy to keep private).

***datalad clone* installs datasets from URLs or local or remote paths**

Annexed files contents can be retrieved or dropped on demand, file contents of files stored in Git are available right away.

***datalad unlock* makes annexed files modifiable, *datalad save* locks them again.**

(It is generally easier to get accidentally saved files out of the annex than out of Git - see handbook.datalad.org/basics/101-136-filesystem.html for examples)

***datalad run* records the impact of any command execution in a dataset.**

Data/directories specified as `--input` are retrieved prior to command execution, data/directories specified as `--output` unlocked.

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

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

QUESTIONS!

Awkward silence can be bridged with awkward MC questions :)

http://etc.ch/7YEk

DROPPING AND REMOVING STUFF

What to do with files you don't want to keep

`datalad drop` and `datalad remove`

Code: psychoinformatics-de.github.io/rdm-course/92-filesystem-operations

DROP & REMOVE

- Try to remove (*rm*) one of the pictures in your dataset. What happens?
- Version control tools keep a revision history of your files - file contents are not actually removed when you *rm* them. Interactions with the revision history of the dataset can bring them "back to life"

DROP & REMOVE

- Clone a small example dataset to drop file contents and remove datasets:

```
$ datalad clone https://github.com/datalad-datasets/machinelearning-books.git
$ cd machinelearning-books
$ datalad get A.Shashua-Introduction_to_Machine_Learning.pdf
```

- **datalad drop** removes annexed file contents from a local dataset annex and frees up disk space. It is the antagonist of **get** (which can get files and subdatasets).

```
$ datalad drop A.Shashua-Introduction_to_Machine_Learning.pdf
drop(ok): /tmp/machinelearning-books/A.Shashua-Introduction_to_Machine_Learning.pdf (file)
          [checking https://arxiv.org/pdf/0904.3664v1.pdf...]
```

- But: Default safety checks require that dropped files can be re-obtained to prevent accidental data loss. **git annex whereis** reports all registered locations of a file's content
- **drop** does not only operate on individual annexed files, but also directories, or globs, and it can uninstall subdatasets:

```
$ datalad clone https://github.com/datalad-datasets/human-connectome-project-openaccess.git
$ cd human-connectome-project-openaccess
$ datalad get -n HCP1200/996782
$ datalad drop --what all  HCP1200/996782
```