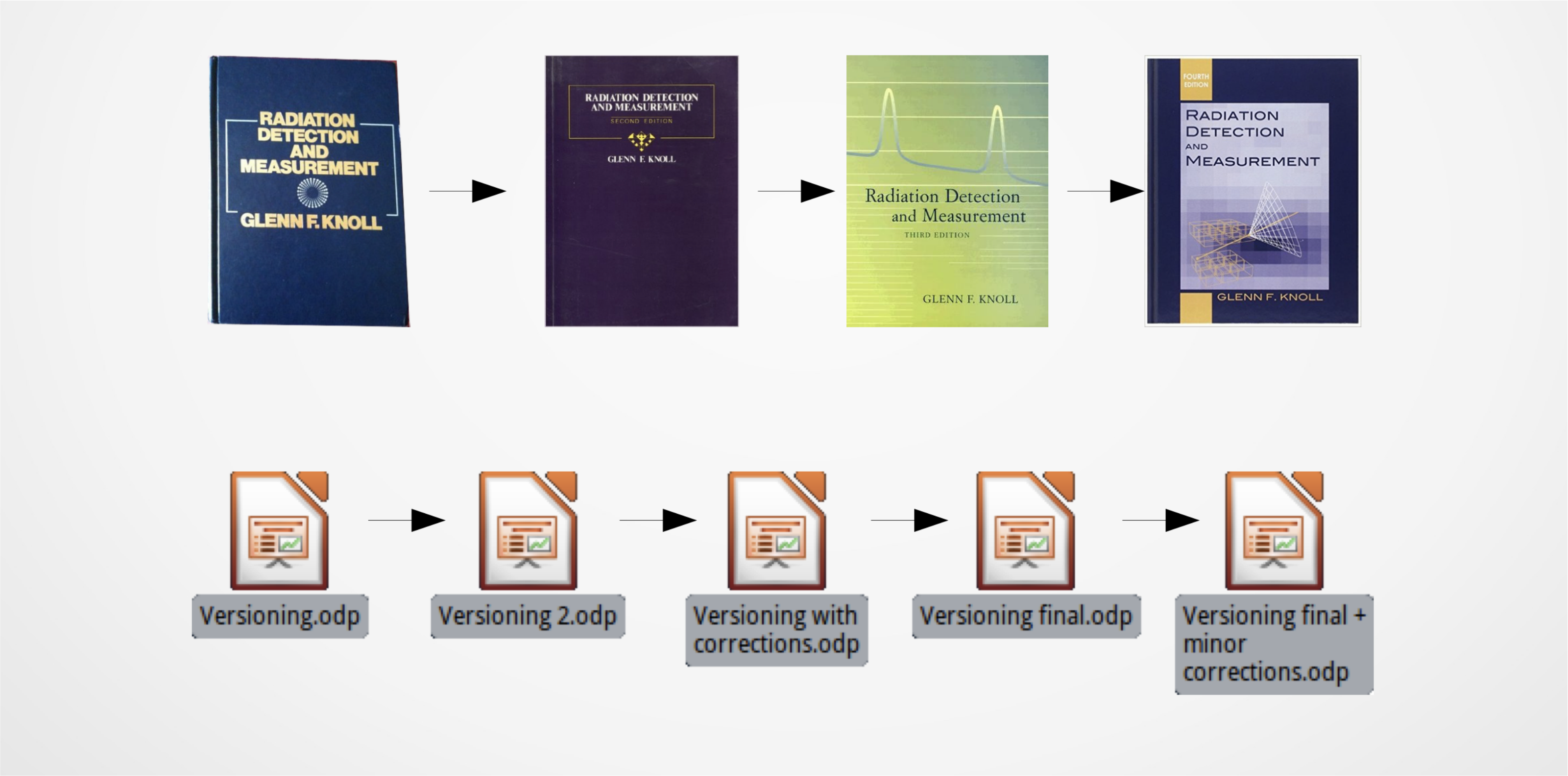
Introduction to data versioning

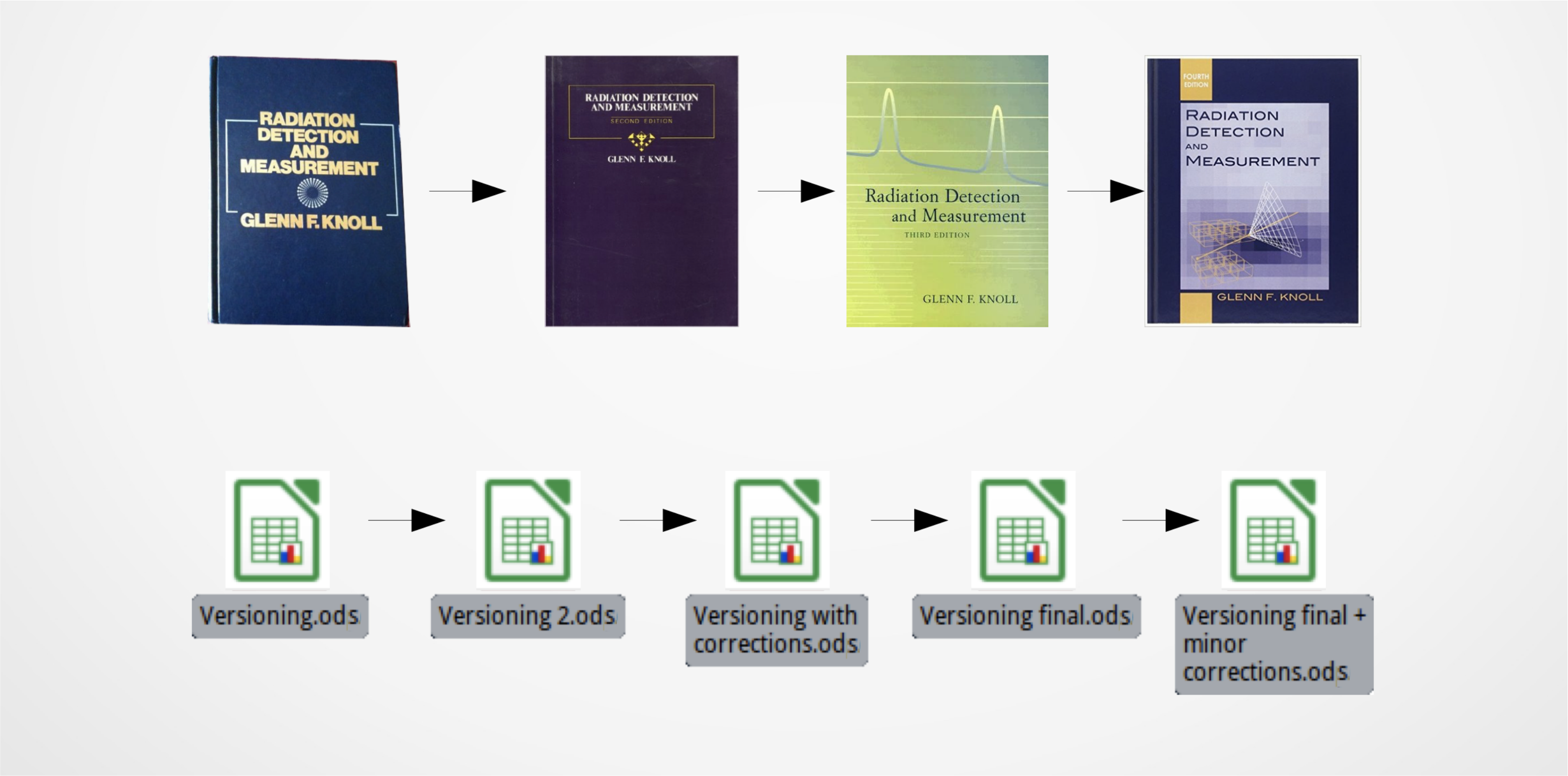
damien.francois@uclouvain.be

Code versioning:



RADIATION DETECTION AND MEASUREMENT
GLENN F. KNOLL
RADIATION DETECTION AND MEASUREMENT
SECOND EDITION
GLENN F. KNOLL
Radiation Detection and Measurement
THIRD EDITION
GLENN F. KNOLL
FOURTH EDITION
RADIATION DETECTION AND MEASUREMENT
GLENN F. KNOLL
Versioning.odp
Versioning 2.odp
Versioning with corrections.odp
Versioning final.odp
Versioning final + minor corrections.odp

Data versioning:


RADIATION DETECTION AND MEASUREMENT
GLENN F. KNOLL
RADIATION DETECTION AND MEASUREMENT
SECOND EDITION
GLENN F. KNOLL
Radiation Detection and Measurement
THIRD EDITION
GLENN F. KNOLL
FOURTH EDITION
RADIATION DETECTION AND MEASUREMENT
GLENN F. KNOLL
Versioning.ods
Versioning 2.ods
Versioning with corrections.ods
Versioning final.ods
Versioning final + minor corrections.ods

What ..

> Data versioning is the transposition of the ideas of code versioning to data files rather than source code.

More specifically:

> Data versioning is creating a *unique reference* for a collection of data. This reference most commonly is a timestamp or a version number and is associated a *comment* or annotation.

Why ..

Track provenance

- aid compliance and auditing
- revert errors to a known working state

Ensure reproducibility

- replicate previous experiments
- run expriments on multiple versions of the data

Ease collaboration

- centralize storage and manage multiple contributions
- communicate changes in the data

How ..

Source code : git is the standard

Data: why not use `git` ?

How ..

`git` and its whole ecosystem is designed to version **source code**, i.e. **small line-oriented text files**.

Data, by contrast,

- can be *large*
- can undergo *column*-oriented changes
- is not necessarily stored in a *text* file
- is not even necessarily stored in a *file*

No universal tool/method for data versioning

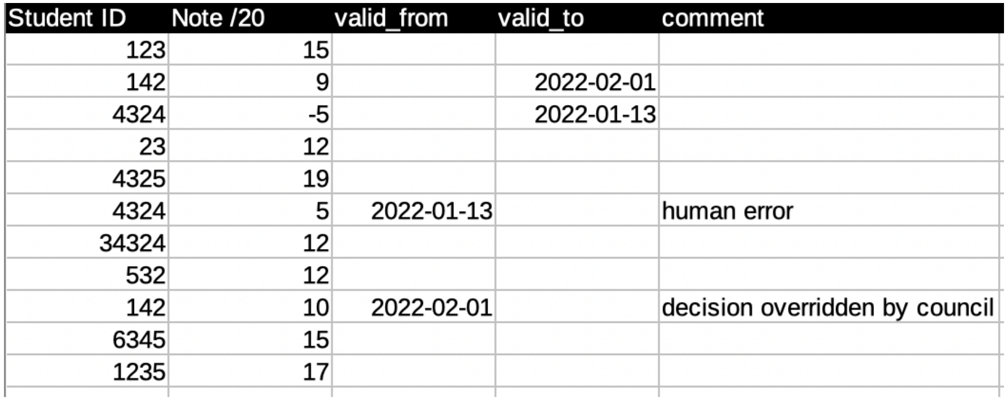
- Naive Approach: Full Duplication
- **Copy on write** approach: `Valid_from/to` Metadata
- Use a file-writing **library** with built-in versioning
- Use data a data **hosting service** that features versioning
- Version Control with **arbitrary file types**
    - Abuse `git`
        - Version `.csv` files
        - Version text dump of data
        - Version code that alters the data
    - Use `git` extension
    - Use dedicated tool

Copy on write

The idea:

> rather than overwriting a data element, append it and timestamp the modification

Referred to as *Type 2 slowly changing dimension (SCD)* in the database world. Notice the `valid_from`, `valid_to`, and `comment` columns.

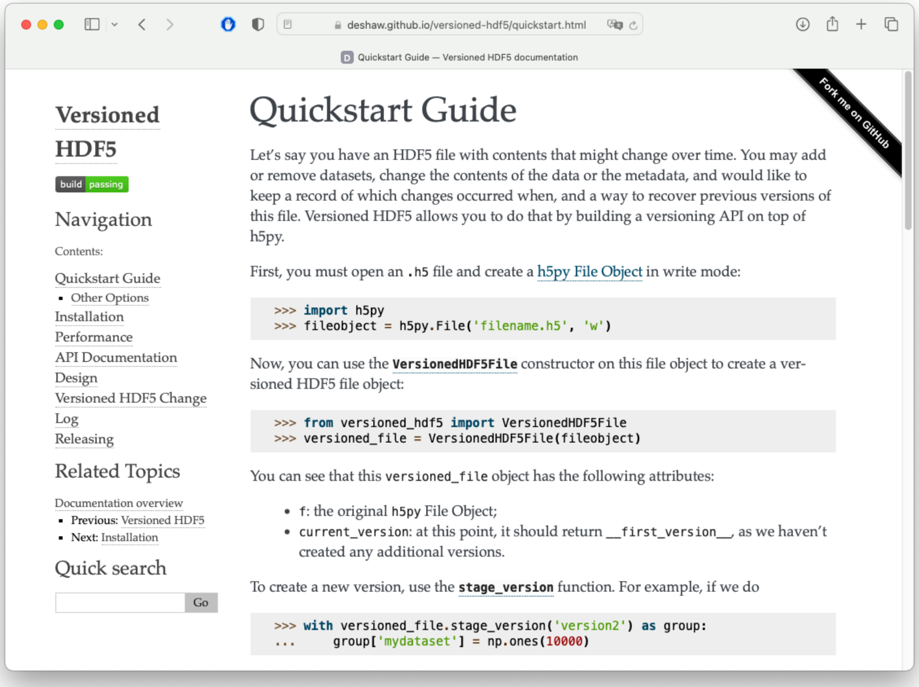


Student ID	Note /20	valid_from	valid_to	comment
123	15			
142	9		2022-02-01	
4324	-5		2022-01-13	
23	12			
4325	19			
4324	5	2022-01-13		human error
34324	12			
532	12			
142	10	2022-02-01		decision overridden by council
6345	15			
1235	17			

https://en.wikipedia.org/wiki/Slowly_changing_dimension

File-writing library with built-in versioning

e.g. Versioned HDF5:


deshaw.github.io/versioned-hdf5/quickstart.html
Quickstart Guide — Versioned HDF5 documentation
Fork me on GitHub
build passing
Go

Versioned HDF5

Navigation

Contents:

- Quickstart Guide
  - Other Options
- Installation
- Performance
- API Documentation
- Design
- Versioned HDF5 Change Log
- Releasing

Related Topics

- Documentation overview
  - Previous: Versioned HDF5
  - Next: Installation

Quick search

Quickstart Guide

Let's say you have an HDF5 file with contents that might change over time. You may add or remove datasets, change the contents of the data or the metadata, and would like to keep a record of which changes occurred when, and a way to recover previous versions of this file. Versioned HDF5 allows you to do that by building a versioning API on top of h5py.

First, you must open an `.h5` file and create a h5py File Object in write mode:

```
>>> import h5py
>>> fileobject = h5py.File('filename.h5', 'w')
```

Now, you can use the **`VersionedHDF5File`** constructor on this file object to create a versioned HDF5 file object:

```
>>> from versioned_hdf5 import VersionedHDF5File
>>> versioned_file = VersionedHDF5File(fileobject)
```

You can see that this `versioned_file` object has the following attributes:

- `f`: the original `h5py` File Object;
- `current_version`: at this point, it should return `__first_version__`, as we haven't created any additional versions.

To create a new version, use the **`stage_version`** function. For example, if we do

```
>>> with versioned_file.stage_version('version2') as group:
...     group['mydataset'] = np.ones(10000)
```

https://deshaw.github.io/versioned-hdf5/quickstart.html

Multi-featured data hosting services

Cloud services: Designed for team collaboration on documents

- Google spreadsheet
- Dropbox and the likes

Backup systems: Designed to keep a history of files

- Borg https://www.borgbackup.org
- Duplicati https://www.duplicati.com

Multi-featured data hosting services (cont'd)

Data repositories: Designed to publish data alongside articles

- Dataverse https://dataverse.org
- Zenodo https://zenodo.org

Data lakes: Designed to foster team collaboration around files

- Neptune https://neptune.ai/home
- Pachyderm https://www.pachyderm.com
- Delta Lake https://delta.io
- lakeFS https://docs.lakefs.io
- Qri https://github.com/qri-io/qri

Multi-featured data hosting services (cont'd)

Workflow management systems: Designed to keep track of experiments

- DAGsHub https://dagshub.com
- MLflow https://mlflow.org
- ClearML https://clear.ml
- Fireworks https://materialsproject.github.io/fireworks/
- NextFlow https://nextflow.io

Version Control with arbitrary file types

Git

- The most used code versioning solution
- Distributed solution (no need for a main server)
- Can interact with code sharing websites (GitHub, GitLab)
- Mainly a command line tool
  - `git init` -- create a repository
  - `git commit` -- freeze a version with author+comment
  - `git push` -- share code
  - `git merge` -- merge code from multiple collaborators

https://indico.cism.ucl.ac.be/event/125/contributions/71/

Why not Git?

Using `git` with *text* data files (`.csv`, `tsv`, `.yml`, `.json`, `.xml`, etc.) *can* work.

But...

- `git` is designed for *small files*
  - `pull` operations assume enough local space
  - commit hashes files, which can be time-consuming
- `git` tools (merge, diff, etc.) are *line-oriented*
  - operations on columns create very large changesets
  - line breaks in the data are mis-interpreted
- What about *binary* files?

Workaround: Version a text copy of the data

- dump a text version of the data
  - `mysqldump > dump.sql`
  - `h5dump -x FILE > dump.xml`
  - `pickle2json.py FILE > dump.json`
  - ...
- version the files
  - `git add *.sql && git commit -m COMMENT`
  - `git add *.xml && git commit -m COMMENT`
  - `git add *.json && git commit -m COMMENT`
  - ...

Better Workaround: Co-version code and data

- insert version in filename, e.g.:
  - `data.v1.Rdata`
  - `data.v1.sqlite3`
  - …
- version code that generates the file
  - `git add create_data.R && git commit -m"v1"`
  - `git add create_data.sql && git commit -m"v1"`
  - …

Git extensions to manage (large) binary files

- `git-annex`
  - based on symbolic links
  - separate commands to manage the files
  - designed for one-to-one file exchanges
- `git-lfs`
  - based on placeholder files (*pointer* files)
  - uses filters so file management is transparent
  - requires specific hosting service for the files
- `git-fat`
  - based on placeholder files
  - uses filters so file management is transparent
  - relies on `rsync` to a central location

https://linuxaria.com/recensioni/storing-large-binary-files-in-git-repositories

Dedicated Git-like tools

- Datalad https://www.datalad.org/

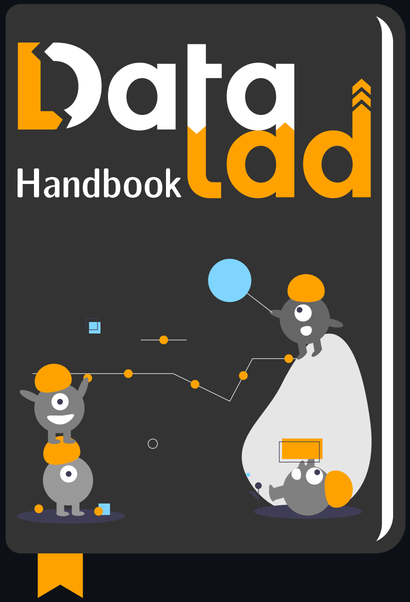
> DataLad is a free and open source distributed data management system that keeps track of your data, creates structure, ensures reproducibility, supports collaboration, and integrates with widely used data infrastructure.

Alternatives:

- DVC https://dvc.org/
- ArtiV https://artivc.io

Datalad tutorial

Based on Chapter 5 of the



http://handbook.datalad.org/en/latest/

Datalad

- command-line tool written in Python
- built upon `git` and `git-annex`
- workflow simplified compared to `git`
- no need to know `git` but `git` command will work
- file type and application domain agnostic
- works with arbitrarily large data
- minimum custom procedures and data strutures
- can interact with many other systems
- documentation is remarkable

Installation

From the doc:

> If you want to install DataLad on a machine you do not have root access to, DataLad can be installed with Miniconda.

```
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ bash Miniconda3-latest-Linux-x86_64.sh
# acknowledge license, keep everything at default
$ conda install -c conda-forge datalad
```

> This should install Git, git-annex, and DataLad. The installer automatically configures the shell to make conda-installed tools accessible, so no further configuration is necessary.

On Lemaitre3

```
[dfr@lemaitre3 ~]$ module load Python git-annex
[dfr@lemaitre3 ~]$ pip3 install datalad
Defaulting to user installation because normal site-packages is not writeable
Collecting datalad
[...]
Successfully installed annexremote-1.6.0 boto-2.49.0 datalad-0.18.0 [...]
[dfr@lemaitre3 ~]$
[dfr@lemaitre3 ~]$
[dfr@lemaitre3 ~]$ datalad
error: too few arguments, run with --help or visit https://handbook.datalad.org
usage: datalad [-c (:name|name=value)] [-C PATH] [--cmd] [-l LEVEL]
               [--on-failure {ignore,continue,stop}]
               [--report-status {success,failure,ok,notneeded,impossible,error}]
               [--report-type {dataset,file}]
               [-f {generic,json,json_pp,tailored,disabled,'<template>'}] [--dbg]
               [--idbg] [--version] [-h]
               command [command-opts]
```

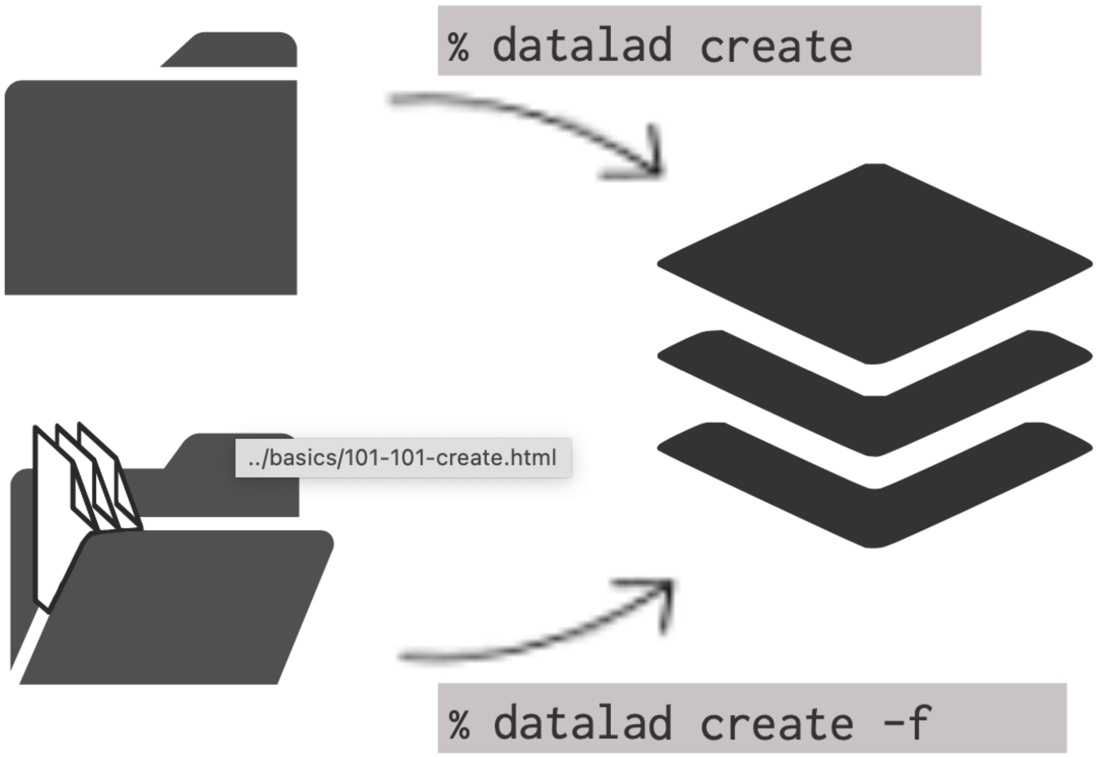
Initial configuration

Datalad needs `git` to be configured properly regarding your identity:

```
% git config --global --add user.name "John Doe"
% git config --global --add user.email jd@example.com
```

Create a dataset

create new, empty datasets to populate...



```
% datalad create
```

```
% datalad create -f
```

.... or transform existing directories into datasets

Create a dataset

- `datalad create` : create a dataset
- `datalad status` : show dataset status

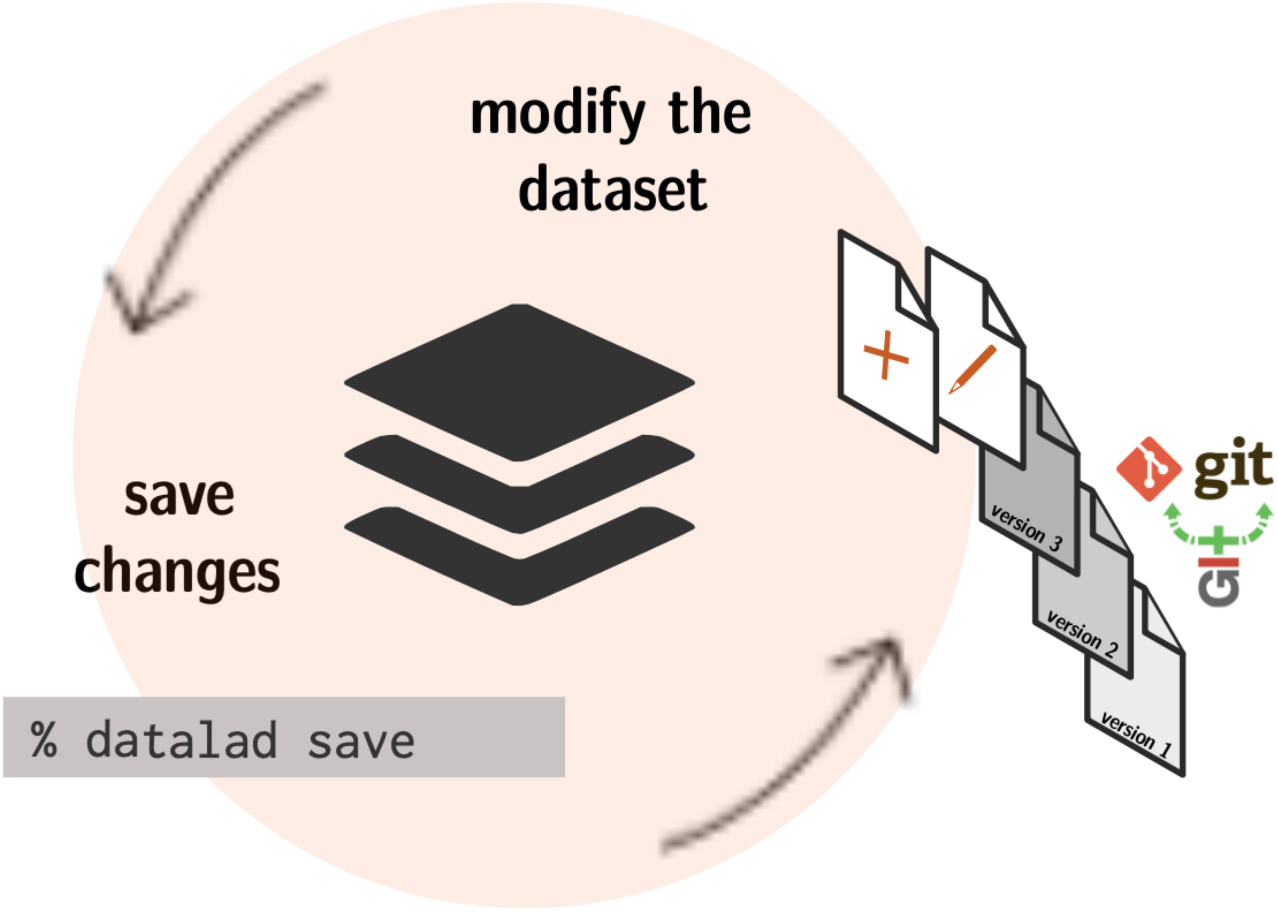
```
[dfr@lemaitre3 ~]$ datalad create --description "Test dataset for datalad" ./testdatalad
create(ok): /home/ucl/pan/dfr/testdatalad (dataset)
[dfr@lemaitre3 ~]$ ls -la ./testdatalad
total 109
drwxrwx---  4 dfr dfr   5 Jan 12 14:36 .
drwxr-x--x 64 dfr dfr 122 Jan 12 14:36 ..
drwxrwx---  2 dfr dfr   4 Jan 12 14:36 .datalad
drwxrwx---  9 dfr dfr  15 Jan 12 14:36 .git
-rw-rw----  1 dfr dfr  55 Jan 12 14:36 .gitattributes

[dfr@lemaitre3 ~]$ cd testdatalad/
[dfr@lemaitre3 testdatalad]$ datalad status
nothing to save, working tree clean
[dfr@lemaitre3 testdatalad]$
```

Add file to adataset

- `datalad save` : add a file to the dataset and commit

```
[dfr@lemaitre3 testdatalad]$ cp ~/data.tar.gz ./

[dfr@lemaitre3 testdatalad]$ datalad status
untracked: /home/ucl/pan/dfr/testdatalad/data.tar.gz (file)

[dfr@lemaitre3 testdatalad]$ datalad save -m "add initial version of data tar file"
add(ok): data.tar.gz (file)
save(ok): . (dataset)
action summary:
  add (ok: 1)
  save (ok: 1)

[dfr@lemaitre3 testdatalad]$ datalad status
nothing to save, working tree clean

[dfr@lemaitre3 testdatalad]$ ls -la
total 124
drwxrwx---  4 dfr dfr   6 Jan 12 14:42 .
drwxr-x--x 64 dfr dfr 122 Jan 12 14:36 ..
drwxrwx---  2 dfr dfr   4 Jan 12 14:36 .datalad
lrwxrwxrwx  1 dfr dfr 136 Jan 12 14:40 data.tar.gz -> .git/annex/[...].tar.gz
drwxrwx---  9 dfr dfr  15 Jan 12 14:42 .git
-rw-rw----  1 dfr dfr  55 Jan 12 14:36 .gitattributes
```

Update a file



```
% datalad save
```

Unlocking files

Files must be "unlocked" before they are modified.

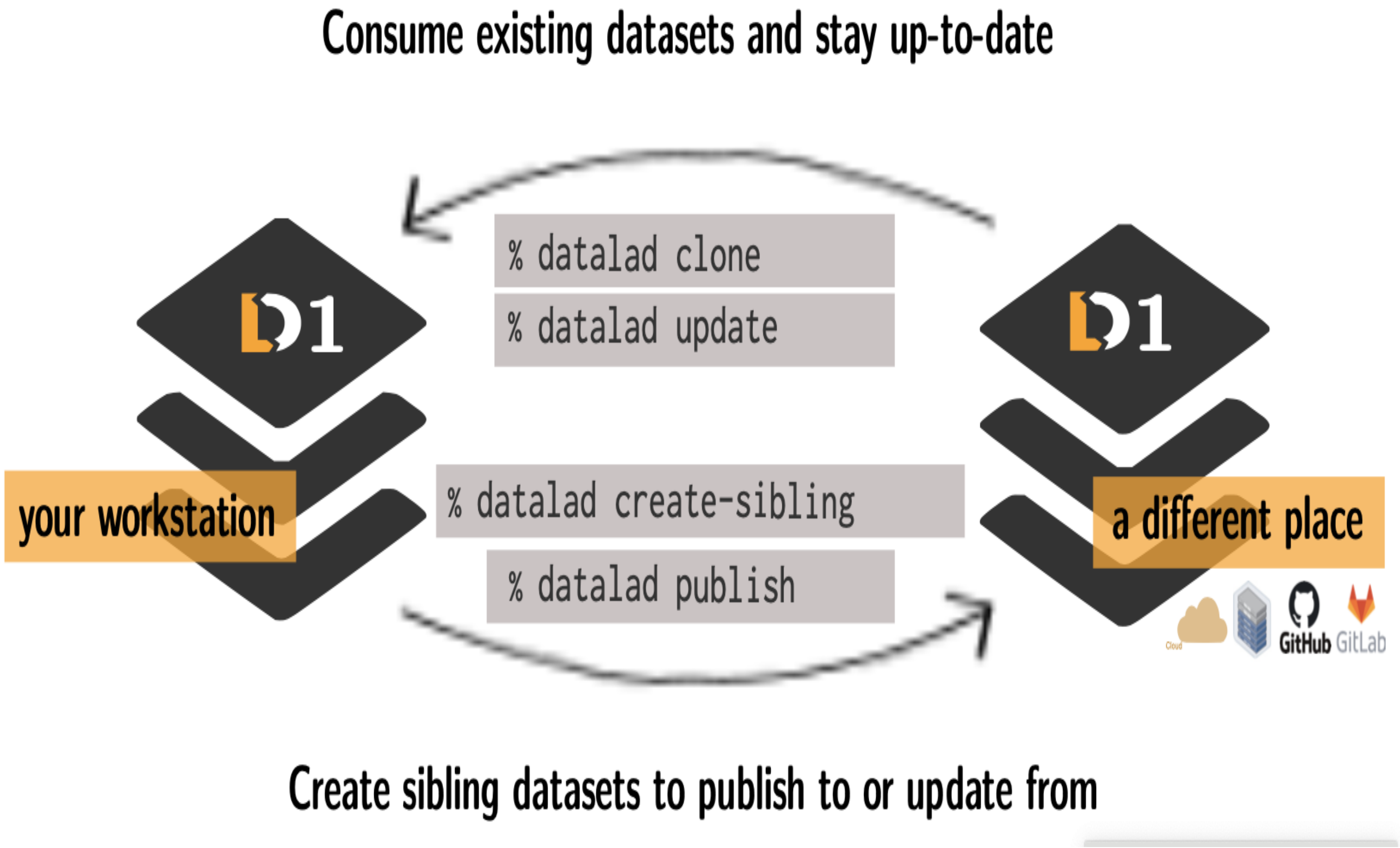
```
[dfr@lemaitre3 testdatalad]$ datalad unlock data.tar.gz
unlock(ok): data.tar.gz (file)
[dfr@lemaitre3 testdatalad]$ ll -la
total 51261
-rw-rw----  1 dfr dfr       55 Jan 12 14:36 .gitattributes
drwxrwx---  2 dfr dfr        4 Jan 12 14:36 .datalad
drwxr-x--x 64 dfr dfr      122 Jan 12 14:47 ..
-rw-r-----  1 dfr dfr 52901945 Jan 12 14:47 data.tar.gz
drwxrwx---  4 dfr dfr        6 Jan 12 14:56 .
drwxrwx---  9 dfr dfr       15 Jan 12 14:56 .git
```

Unlocking files

The `tar.gz` file can then be modified to add a `README` file, and then `saved` again.

```
[dfr@lemaitre3 testdatalad]$ datalad status
 modified: /home/ucl/pan/dfr/testdatalad/data.tar.gz (file)

[dfr@lemaitre3 testdatalad]$ datalad save -m 'update data archive with README file'
add(ok): data.tar.gz (file)
save(ok): . (dataset)
action summary:
  add (ok: 1)
  save (ok: 1)
[dfr@lemaitre3 testdatalad]$ datalad status
nothing to save, working tree clean

[dfr@lemaitre3 testdatalad]$ git log --oneline
3903790 (HEAD -> master) update data archive with README file
012d634 add initial version of data tar file
e53138a [DATALAD] new dataset
```

Pull/push data



Consume existing datasets and stay up-to-date

```
% datalad clone
% datalad update
```

```
% datalad create-sibling
% datalad publish
```

Create sibling datasets to publish to or update from

Pull data

Datalad `clone` is used to copy a dataset in another location.

```
[dfr@lemaitre3 testdatalad]$ cd $GLOBALSCRATCH
[dfr@lemaitre3 dfr]$ datalad clone ~/testdatalad
install(ok): /scratch/ucl/pan/dfr/testdatalad (dataset)

[dfr@lemaitre3 dfr]$ cd testdatalad/
[dfr@lemaitre3 testdatalad]$ ls
data.tar.gz
[dfr@lemaitre3 testdatalad]$ file data.tar.gz
data.tar.gz: broken symbolic link [...]

[dfr@lemaitre3 testdatalad]$ datalad status --annex all
1 annex'd file (0.0 B/85.7 MB present/total size)
nothing to save, working tree clean
```

Pull data

Datalad `get` is used to actually retrieve data files

```
[dfr@lemaitre3 testdatalad]$ datalad get data.tar.gz
get(ok): data.tar.gz (file) [from origin...]
[dfr@lemaitre3 testdatalad]$ datalad status --annex all
1 annex'd file (85.7 MB/85.7 MB present/total size)
nothing to save, working tree clean

[dfr@lemaitre3 testdatalad]$ datalad drop data.tar.gz
drop(ok): data.tar.gz (file)
[dfr@lemaitre3 testdatalad]$ datalad status --annex all
1 annex'd file (0.0 B/85.7 MB present/total size)
nothing to save, working tree clean
```

Push data

The copy can be "registered" back to the original location with `datalad siblings`:

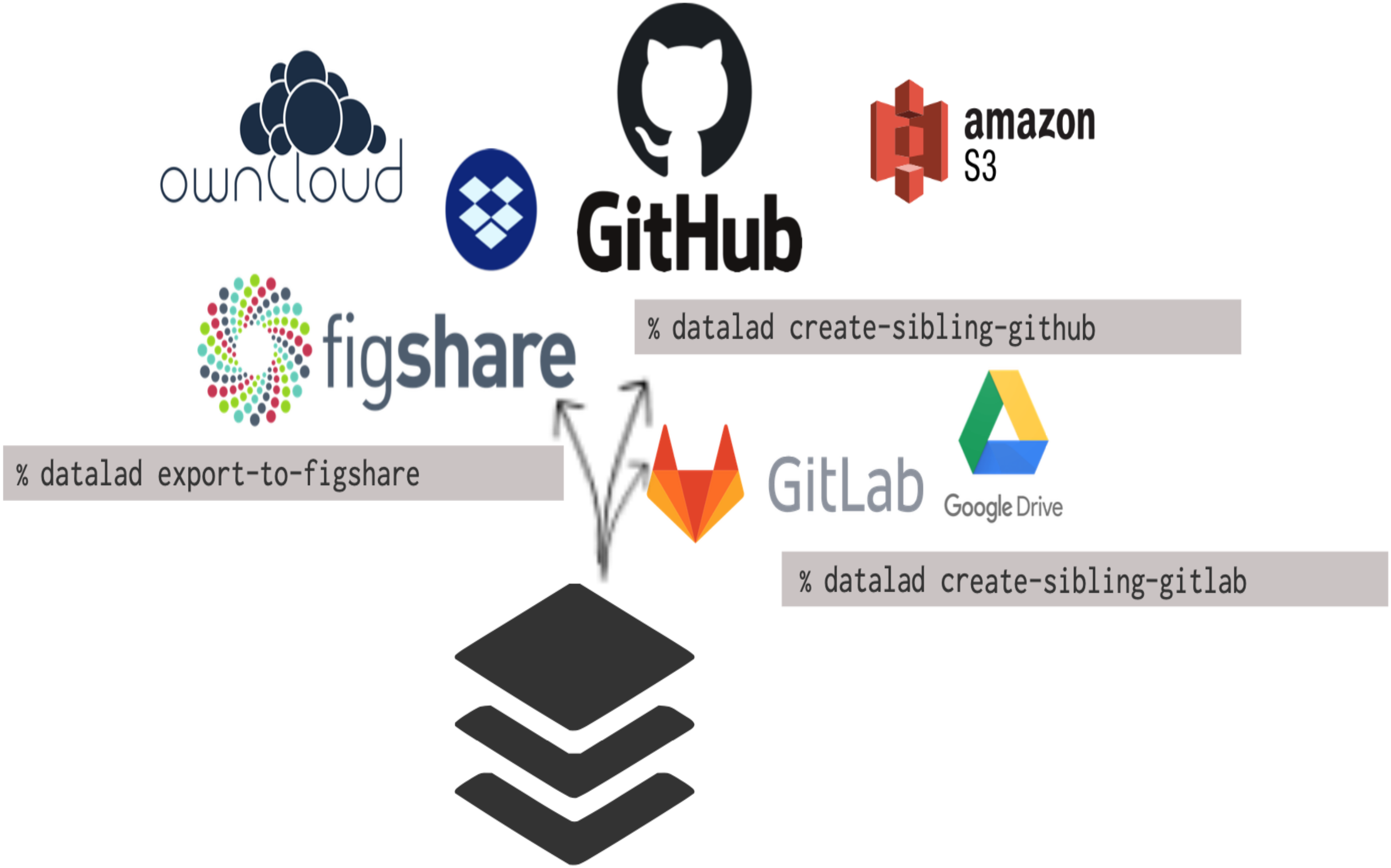
```
# In the original directory:
[dfr@lemaitre3 testdatalad]$ datalad siblings add -d . \
> --name scratch --url $GLOBALSCRATCH/testdatalad
.: scratch(+) [/scratch/ucl/pan/dfr/testdatalad (git)]
[dfr@lemaitre3 testdatalad]$ datalad siblings
.: here(+) [git]
.: scratch(+) [/scratch/ucl/pan/dfr/testdatalad (git)]

[dfr@lemaitre3 testdatalad]$ git remote -v
scratch /scratch/ucl/pan/dfr/testdatalad (fetch)
scratch /scratch/ucl/pan/dfr/testdatalad (push)
```

Push data

Or a copy can be created from the original location with `create-sibling`:

```
[dfr@lemaitre3 testdatalad]$ datalad create-sibling -s cecistorage  $CECIHOME/testdatalad
[INFO   ] Considering to create a target dataset [...]/testdatalad at /CECI/[...]/testdatalad
[INFO   ] Fetching updates for Dataset(/home/ucl/pan/dfr/testdatalad)
update(ok): . (dataset)
[INFO   ] Adjusting remote git configuration
[INFO   ] Running post-update hooks in all created siblings
create_sibling(ok): /home/ucl/pan/dfr/testdatalad (dataset)
[dfr@lemaitre3 testdatalad]$ git remote -v
cecistorage     /CECI/home/ucl/pan/dfr/testdatalad (fetch)
cecistorage     /CECI/home/ucl/pan/dfr/testdatalad (push)
scratch /scratch/ucl/pan/dfr/testdatalad (fetch)
scratch /scratch/ucl/pan/dfr/testdatalad (push)

# through SSH
[dfr@lemaitre3 testdatalad]$ datalad create-sibling -s workstation  workstation:testdatalad
```

Push data



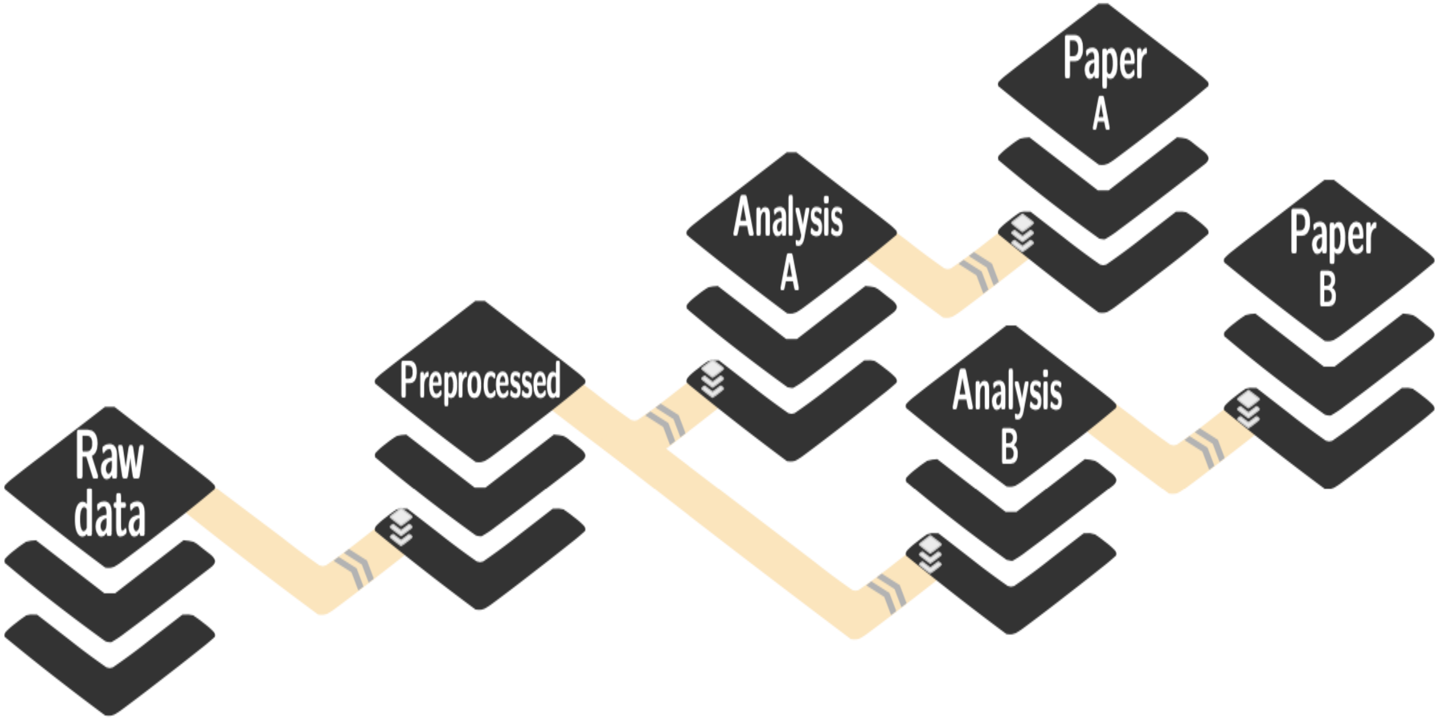
```
% datalad create-sibling-github
```

```
% datalad export-to-figshare
```

```
% datalad create-sibling-gitlab
```

https://git-annex.branchable.com/special_remotes/ | https://rclone.org

Nesting datasets



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

Nesting datasets

A foreign dataset can be "included" as a Git `submodule`:

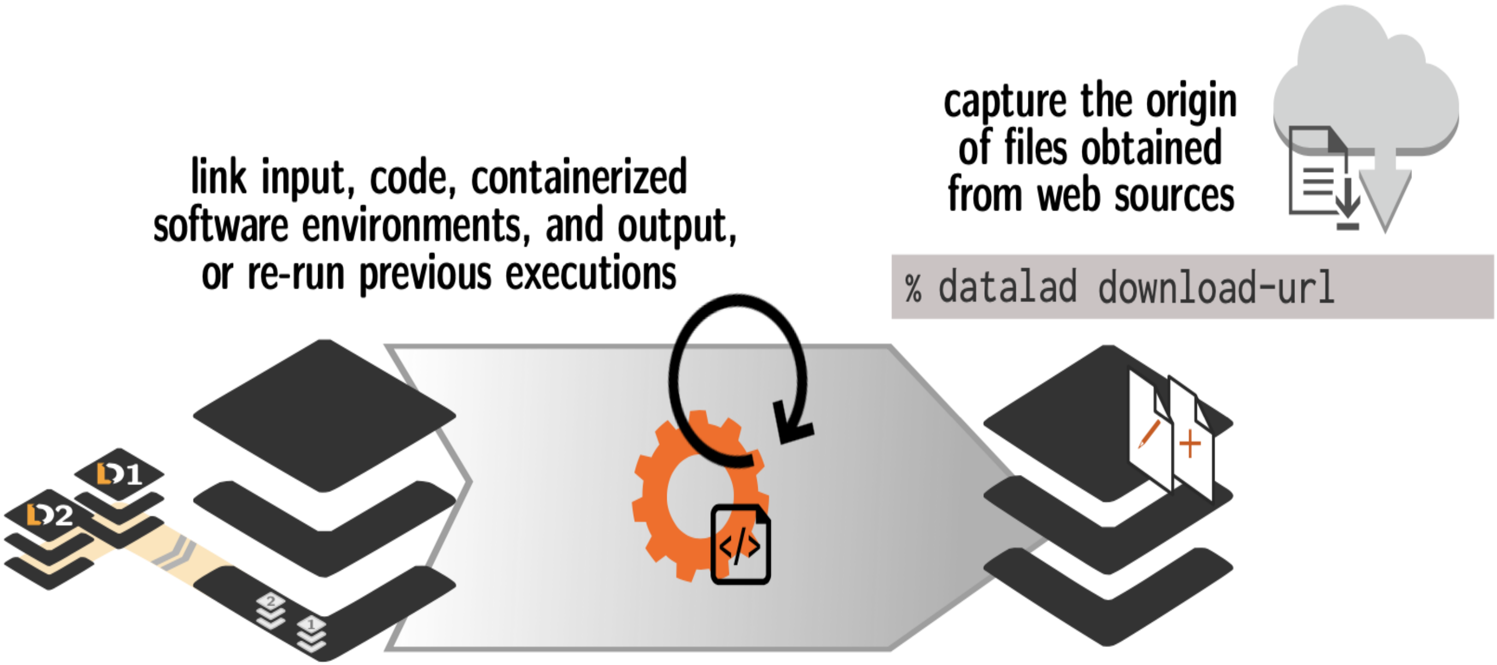
```
[dfr@lemaitre3 testdatalad]$ datalad clone --dataset . \
> https://github.com/damienfrancois/dataladset.git random
install(ok): random (dataset)
add(ok): random (dataset)
add(ok): .gitmodules (file)
save(ok): . (dataset)
add(ok): .gitmodules (file)
save(ok): . (dataset)
action summary:
  add (ok: 3)
  install (ok: 1)
  save (ok: 2)
[dfr@lemaitre3 testdatalad]$ ls
data.tar.gz  random


[dfr@lemaitre3 testdatalad]$ git submodule
 fc1b8ff45b59b2cb04f83fd13cffbd8603974ff5 random (heads/main)
```

Nesting datasets

The `save` command then freezes the version of the foreign dataset that is included.

```
[dfr@lemaitre3 testdatalad]$ cd random/
[dfr@lemaitre3 random]$ git log --oneline
fc1b8ff (HEAD -> main, origin/main, origin/HEAD) add random file
7eafa1d Initial commit

[dfr@lemaitre3 random]$ datalad status --annex all
1 annex'd file (0.0 B/95.4 MB present/total size)
nothing to save, working tree clean
[dfr@lemaitre3 random]$ datalad get random.dat
get(ok): random.dat (file) [from web...]

dfr@lemaitre3 testdatalad]$ datalad save -m"include random dataset from github"
add(ok): random (dataset)
add(ok): .gitmodules (file)
save(ok): . (dataset)
action summary:
  add (ok: 2)
  save (ok: 1)
```

Tracking experiments



link input, code, containerized software environments, and output, or re-run previous executions

capture the origin of files obtained from web sources

```
% datalad download-url
```

```
% datalad run
```

```
% datalad rerun
```

```
% datalad run-procedure
```

Tracking experiments

Use `datalad run` to run a command and commit the result.

```
[dfr@lemaitre3 testdatalad]$ datalad run -m "count number of 0's in the random file" \
> "grep -co \0 random/random.dat > ./count.txt"
[INFO   ] == Command start (output follows) =====
[INFO   ] == Command exit (modification check follows) =====
run(ok): /home/ucl/pan/dfr/testdatalad (dataset) [grep -co \0 random/random.dat > ./count....]
add(ok): count.txt (file)
save(ok): . (dataset)

[dfr@lemaitre3 testdatalad]$ cat count.txt
195367

[dfr@lemaitre3 testdatalad]$ git log --oneline
fdb7a81 (HEAD -> master) [DATALAD RUNCMD] count number of 0's in the random file
7065bd1 include random dataset from github
a63adf3 [DATALAD] Added subdataset
3903790 update data archive with README file
012d634 add initial version of data tar file
e53138a [DATALAD] new dataset
```

Tracking experiments

The command is stored in git.

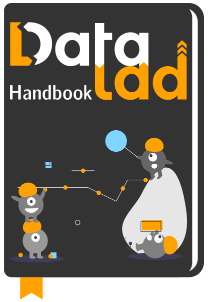
```
[dfr@lemaitre3 testdatalad]$ git log
commit fdb7a816cb5cae370d9ec39dd54e26b950ef5fa7
Author: Damien François <damien.francois@uclouvain.be>
Date:   Thu Jan 12 16:17:13 2023 +0100

    [DATALAD RUNCMD] count number of 0's in the random file

    === Do not change lines below ===
    {
     "chain": [],
     "cmd": "grep -co \\0 random/random.dat > ./count.txt",
     "dsid": "4383763f-7f51-4193-b57d-e96d029f3526",
     "exit": 0,
     "extra_inputs": [],
     "inputs": [],
     "outputs": [],
     "pwd": "."
    }
    ^^^ Do not change lines above ^^^
```

Tracking experiments

Experiments can be `rerun`:

```
[dfr@lemaitre3 testdatalad]$ datalad rerun fdb7a81
[INFO   ] run commit fdb7a81; (count number of 0...)
unlock(ok): count.txt (file)
[INFO   ] == Command start (output follows) =====
[INFO   ] == Command exit (modification check follows) =====
run(ok): /home/ucl/pan/dfr/testdatalad (dataset) [grep -co \0 random/random.dat > ./count....]
add(ok): count.txt (file)
action summary:
  add (ok: 1)
  run (ok: 1)
  save (notneeded: 2)
  unlock (ok: 1)
[dfr@lemaitre3 testdatalad]$
```

Further reading:



- https://www.datalad.org
- https://github.com/datalad/tutorials
- https://www.youtube.com/watch?v=QsAqnP7TwyY

Summary and conclusions

Data versioning

Same stakes/challenges as code versioning except with possibly large binary files and no single clear solution.

- Use a file-writing **library** with built-in versioning
- Use data a data **hosting service** that features versioning
- Version Control with **arbitrary file types**
    - Workaround with `git`
        - Version text dump of data
        - Version code that alters the data
    - `git` extensions
    - `datalad`

Further reading

- https://startupstash.com/data-versioning-tools/
- https://www.fuzzylabs.ai/guides/data-version-control

Choosing a versioning system

- http://calver.org
- http://semver.org

Tips for writing commit messages

- https://www.conventionalcommits.org
- https://git-scm.com/book/en/v2/Distributed-Git-Contributing-to-a-Project