Reproducible Data Management with Datalad

Part 1

David Deepwell and Pedro Martinez

June 12, 2024

UNIVERSITY OF CALGARY

Research Data Management

- Research projects are not static:
  - Research focus might change
  - New data is added
  - Data might be discarded
  - A project might split or join other projects
  - Software is updated
  - New analyses are tried
  - Data issues are found and dealt with

Research Data Management

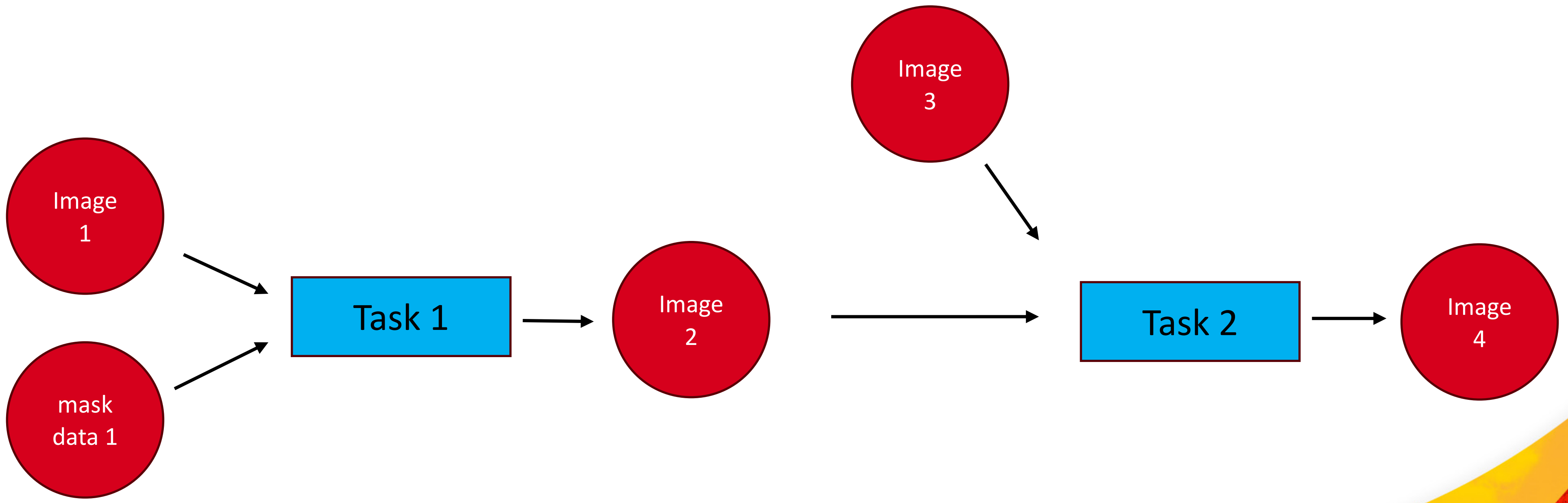
- Requires time to keep up with changes
- Management of:
  - Data (specifically, data versioning)
  - Research software
  - Scripts and workflows

Research Data Management

- How can RDM be made reproducible?
  - By recording data provenance
    - What was executed
    - On what data
    - By whom
    - When
    - And why

Data Provenance

- Data provenance
  - A record of file sources and subsequent modifications
  - Can be viewed as a graph



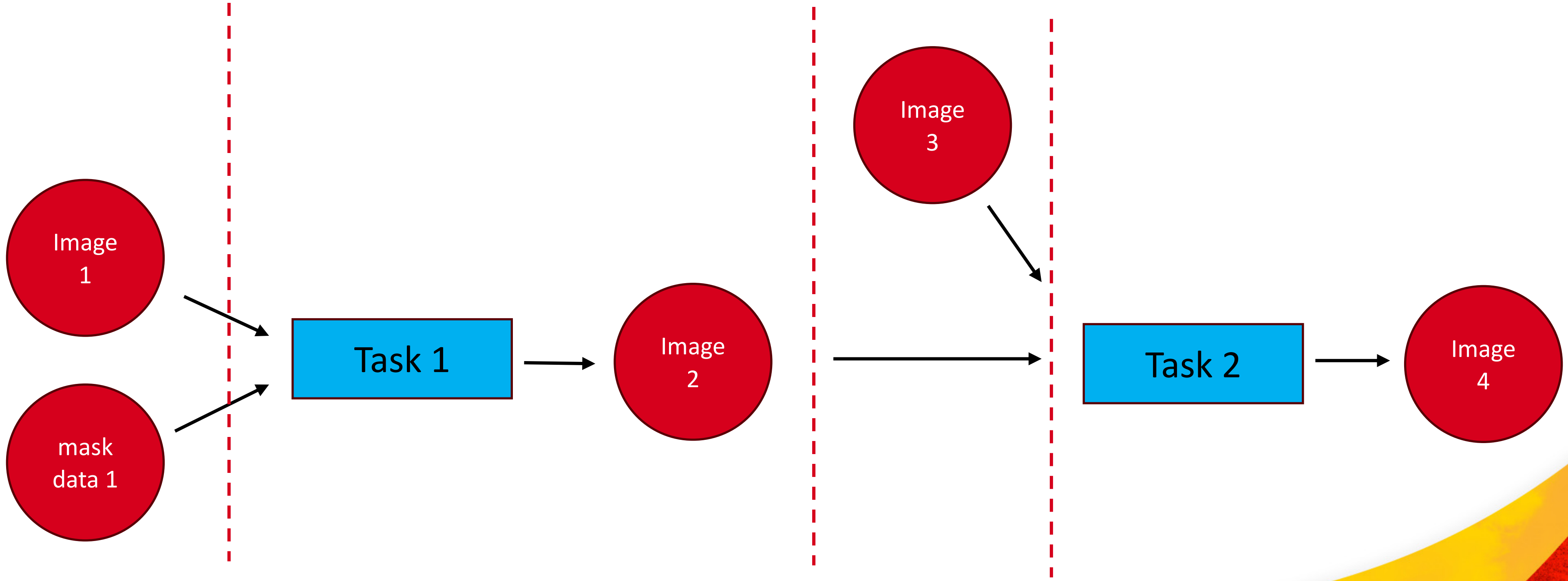
Data Provenance

Commit 1
Date: 25-05-2024
Author: Bob Smith
Message: Add data

Commit 2
Date: 26-05-2024
Author: Bob Smith
Message: Run Task 1

Commit 3
Date: 27-05-2024
Author: Alice Jones
Message: Add Image 3

Commit 4
Date: 28-05-2024
Author: Alice Jones
Message: Run Task 2



DataLad

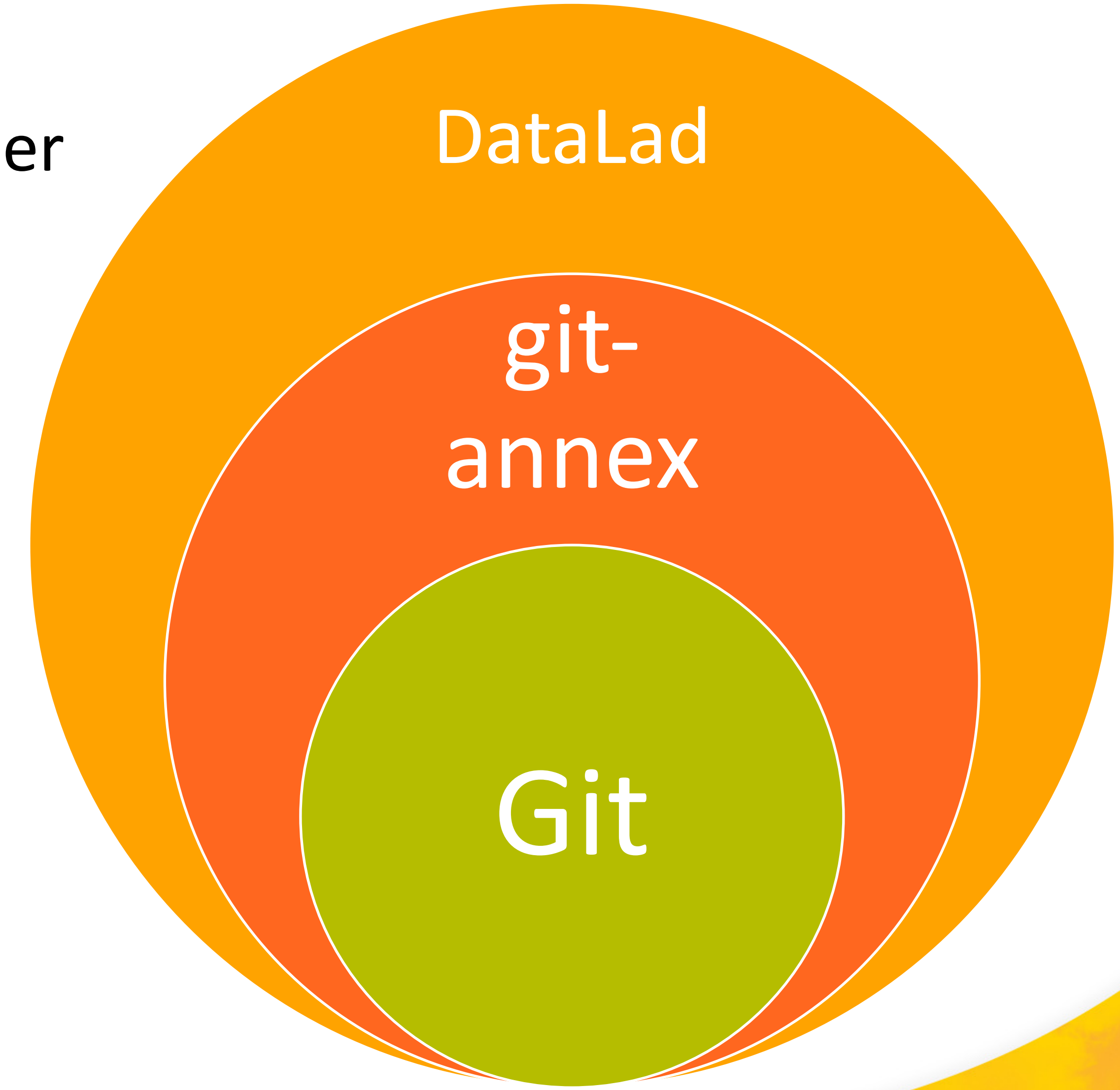
- DataLad is a distributed data management system built for research data management

DataLad

- DataLad is a distributed data management system built for research data management
- Tracks data provenance via git and git-annex (version control systems)
- Handles arbitrarily large files and indefinitely many files
- Example:
    - Share the largest library of MRI images in the world

DataLad

- DataLad: Data-oriented wrapper
- Git-annex: Tracks large files
- Git: Version control system



Value of DataLad

UNIVERSITY OF CALGARY

Record data provenance

Track the history and sources of data

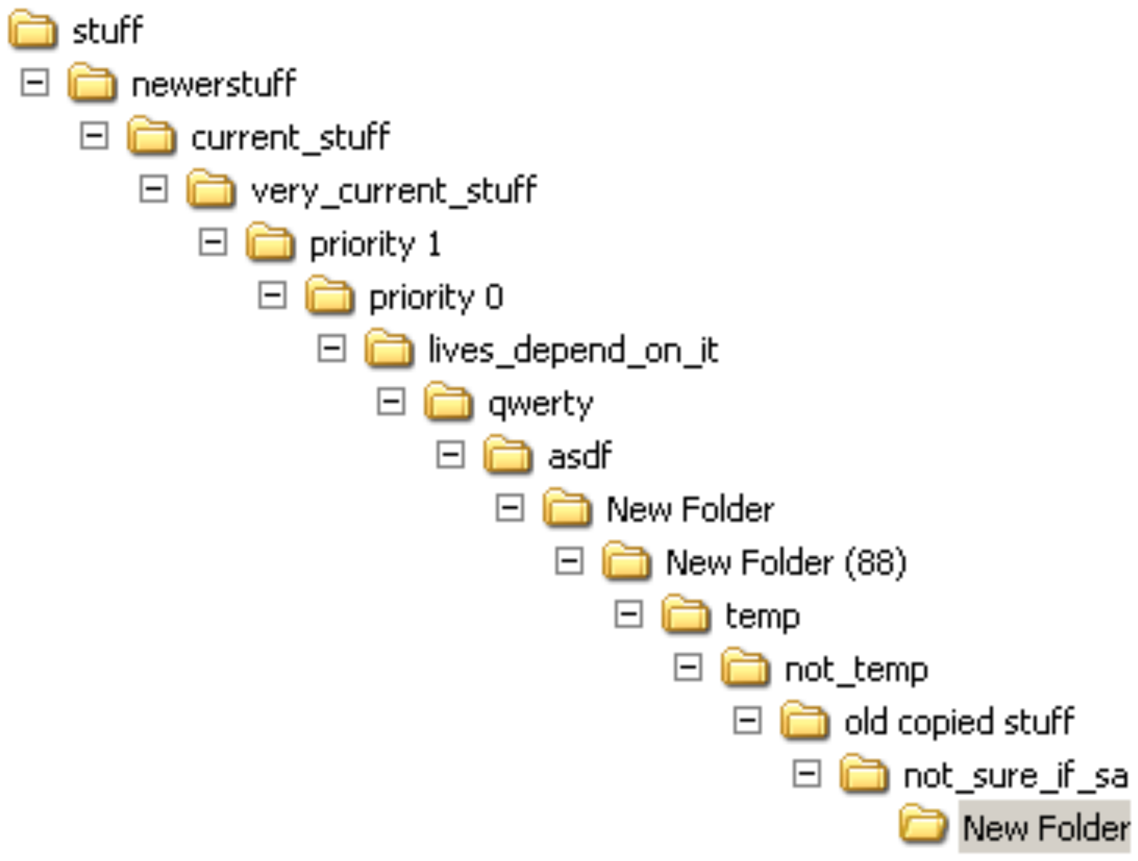
Ensure data reproducibility

Effortlessly rerun computations from a long time ago

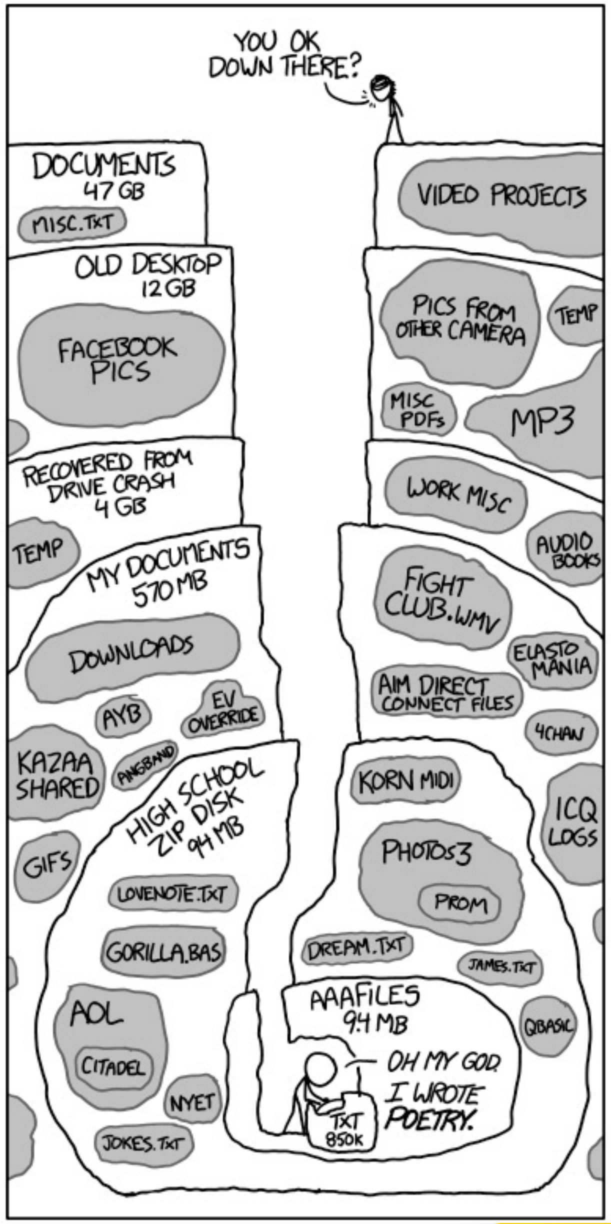
Support collaboration

Easily share data with colleagues

Project Structure

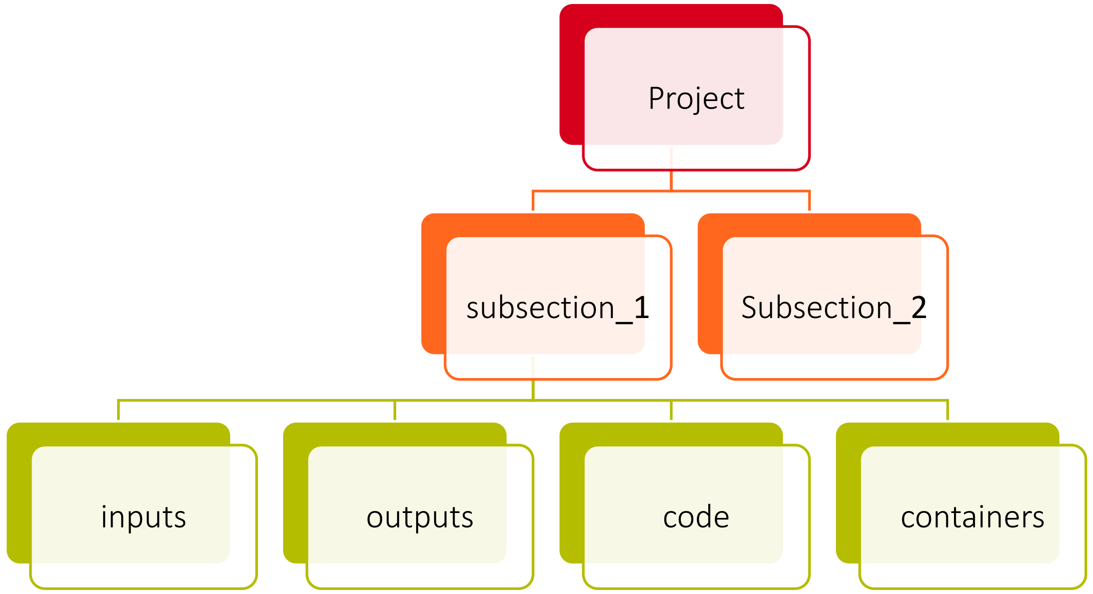


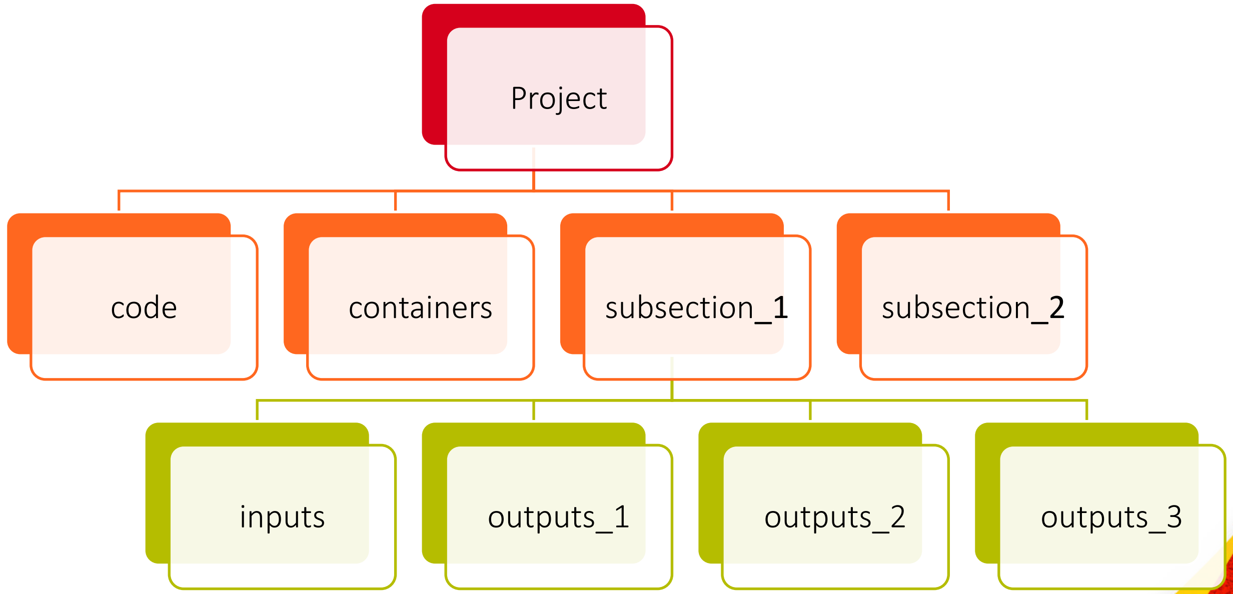
```
stuff
  newerstuff
    current_stuff
      very_current_stuff
        priority 1
          priority 0
            lives_depend_on_it
              qwerty
                asdf
                  New Folder
                    New Folder (88)
                      temp
                        not_temp
                          old copied stuff
                            not_sure_if_sa
                              New Folder
```



Project Structure

No one size fits all!





DataLad Dataset

UNIVERSITY OF CALGARY

- A collection of folders and files that serve a specific purpose
  - Is a git repo
  - The collection exists under a directory
    - Files/folders above this directory are not tracked by this dataset
  - DataLad utilizes the filesystem to track files and folders

Dataset Nesting

- Datasets can contain other datasets (a subdataset)
- Nesting can be arbitrarily:
  - Deep: dataset within dataset within dataset ...
  - Wide: dataset containing multiple datasets
- Datasets have independent history
  - A parent dataset specifies the commit of a subdataset to use
  - Care is needed to keep parent datasets up-to-date with modifications made within subdatasets (argument flags exists to help with this)

DataLad Usage

- The CLI (Command line interface)
  - Covered in these sessions
- Python API
  - For integrating directly into your software
  - Not covered
- A limited GUI also exists (datalad-gooey)
- WARNING: DataLad does not work well with Windows
  - If using Windows, it is recommended to use WSL

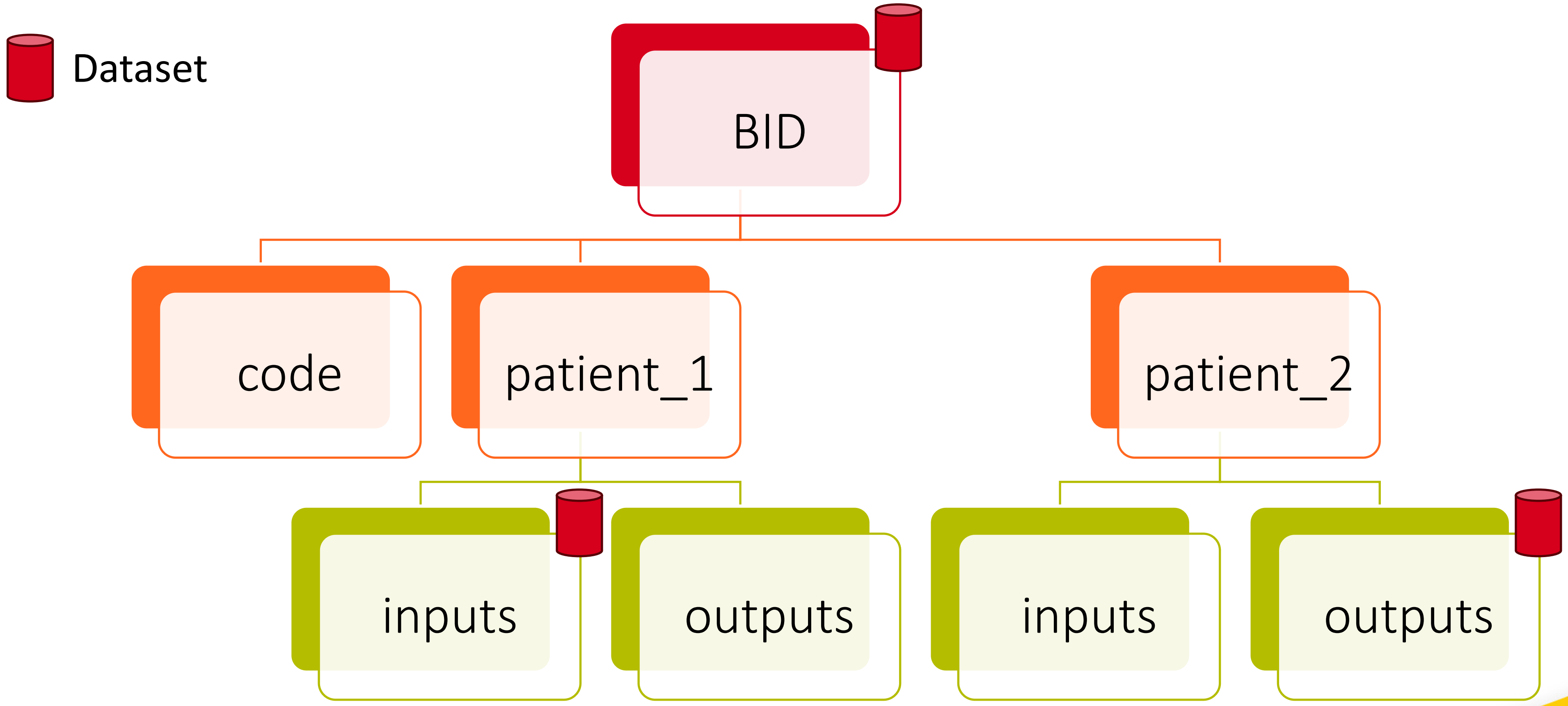
Demo!

Create a Dataset

- Situation:
  - A researcher is starting a new clinical study
  - Plan to record data provenance to track the changing states of the project
  - Subdatasets will be used to isolate “patient” data and enable their re-use in other projects

Part 1 Dataset End State

- BID: Brisk Instruction on DataLad



Create Project

- Create DataLad dataset:
  - `datalad create -c text2git $study_dir`
- Create subdataset
  - `datalad create -d .. -c text2git inputs`

- Show datasets
  - `datalad subdatasets`

Add and Commit Data

- Add data to project
- Before committing changes, check the state of the project to confirm changes
  - `datalad status`
- Commit changes
  - `datalad save -m "Add meaningful commit message"`

Commit in Subdatasets

- Subdatasets are separate git repos and require their own commits
- The parent dataset will need updating to the newest version of the subdataset

Add File to the Annex

- The previous data were text files and added to the git repo
- Add and commit a binary file (use recursive flag)
  - `datalad save -r`

- This file is protected and requires unlocking before making changes
  - `datalad unlock file.bin`
- Once finished changing, lock file
  - `git annex lock file.bin`

UNIVERSITY OF CALGARY

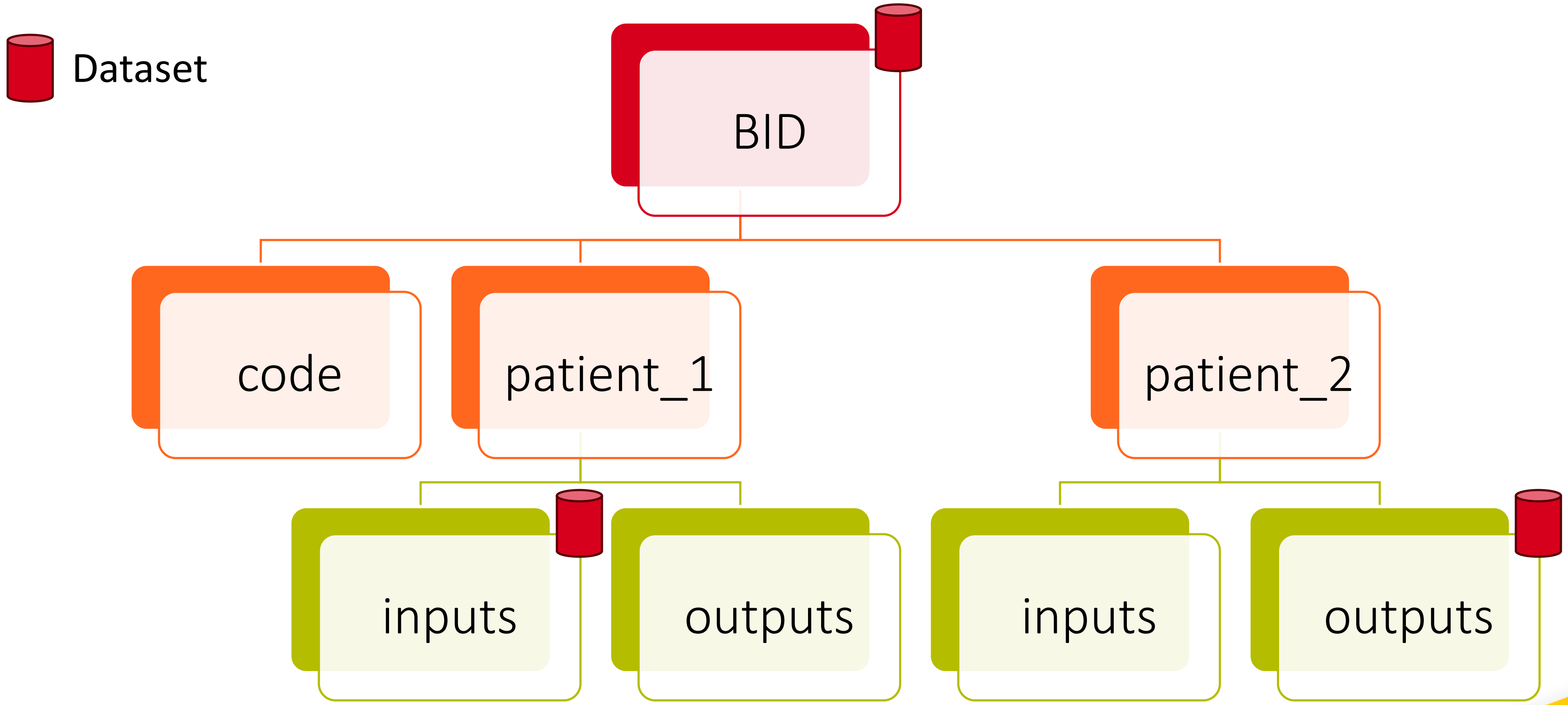
Datalad - Part 2

Background

- From Part 1:
  - Data provenance
  - DataLad purpose and goals
  - Dataset organization guidelines
  - Created and configured a parent dataset
  - Added subdatasets (each associated with a patient)
  - Populated subdatasets with data
  - Explored dataset organization and the data provenance log

Part 1 Dataset End State

- BID: Brisk Instruction on DataLad



Run a command

- Situation:
  - Record the execution of a script or software application
  - The record includes:
    - Input files used
    - Output files generated
    - Optional flags used by tool or software
  - DataLad command:
    - `datalad run`
  - Check the git log:
    - `git log`

Rerun a command

- Situation:
  - 6 months after running the analysis on case 1 you notice that the input data has an error
  - How do you rerun the analysis after fixing the data? Especially if there are many analysis routines which depend on that data.

Rerun a command

- Situation:
  - 6 months after running the analysis on case 1 you notice that the input data has an error
  - How do you rerun the analysis after fixing the data? Especially if there are many analysis routines which depend on that data.
  - Without DataLad:
    - Try to remember what command(s) used (and parameters to the command(s))
    - If recorded the command, do you remember where that record was? What if the colleague who initially ran the command is now gone?
  - With Datalad:
    - Everything is recorded in the dataset, so simply rerun the analysis

Rerun a command

- Scenario:
  - "Fix" the data error:
    - Replace all instances of 4 with 3 in inputs/sample_01.txt for patient 1
  - Save the “fixed” data
  - Find commit associated with the first run that had this file as an input
  - Rerun the analysis (in the subdataset):
    - `datalad rerun $RUN_COMMIT`
    - This will only rerun that commit, to rerun everything since that commit use:
      - `datalad rerun --since=$RUN_COMMIT`
  - Update the parent dataset to the new subdataset state

  - Check the git log:
    - `git log`

Add external dataset/files

- Situation:
  - Your research uses publicly available data
  - How is this data added to the dataset?

Add external dataset/files

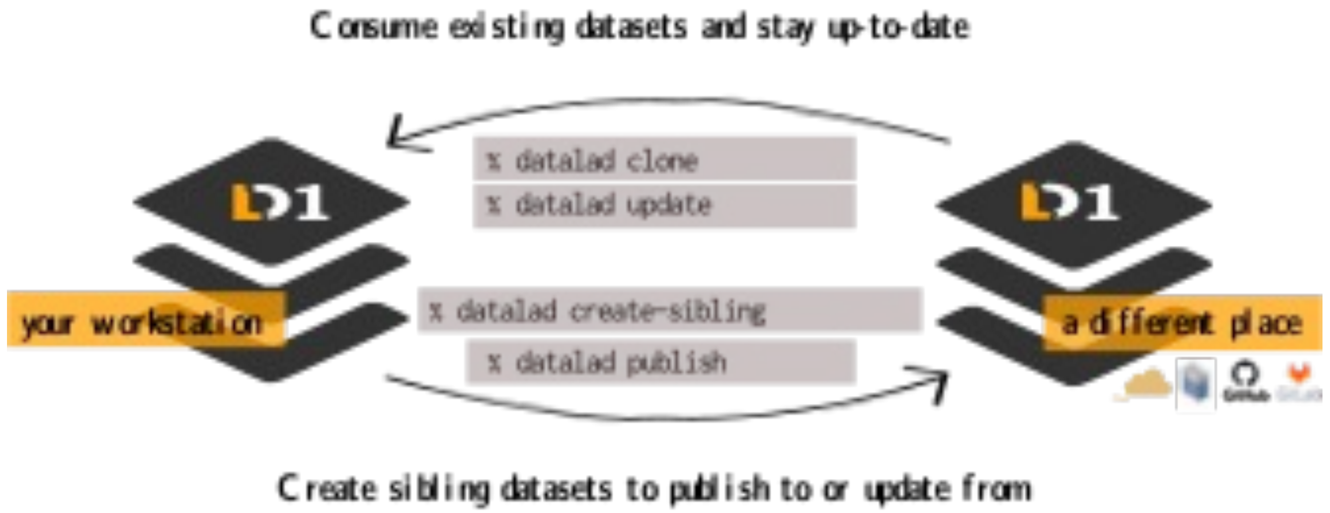
- Scenario 1 - Data is already a Datalad dataset:
  - Add dataset
    - `datalad install`
    - Only installs the dataset provenance and file references, does not add the annexed files
  - Get data
    - `datalad get`
  - View data
    - How much data is stored locally
      - `datalad status --annex all`
    - Where data is located
      - `git annex whereis`

Add external dataset/files

- Scenario 2 - Data is an internet file:
  - Add file
    - `datalad download-url`
    - Adds the file and records the url
    - Does not track the history of the file before it was posted to the web

Datalad siblings

- Datalad sibling: A known dataset clone of a DataLad dataset
    - Scenario 1: Distribute work between laptop and lab machine or HPC
    - Scenario 2: Collaborate with a colleague



Datalad sibling

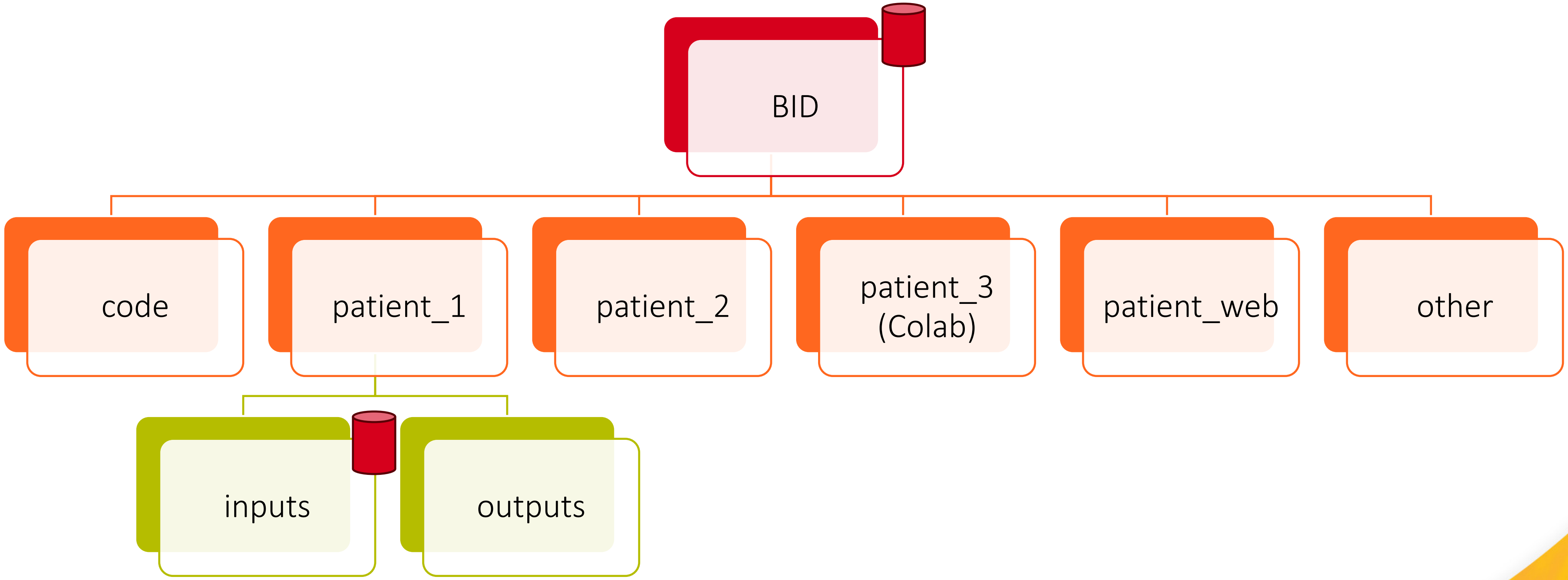
- Scenario:
  - Create a sibling
    - `datalad install` or `datalad clone`
  - On sibling:
    - Create case 3
    - Add and save data to case 3
    - Run analysis on case 3
  - On original dataset
    - View siblings (notice that the sibling is not present)
      - `datalad siblings`
    - Add sibling
      - `datalad siblings add`

Access content on sibling

- Scenario (on original dataset):
  - Make dataset aware of sibling modifications
    - `datalad update --sibling lab_computer --how fetch`
  - View changes between siblings
    - `datalad diff`
    - `git diff`
  - Merge changes from sibling
    - `datalad update --sibling lab_computer --how merge`
  - Get contents
    - `datalad get`

Part 2 Dataset end state

- BID: Brisk Instruction on DataLad



Review

- Provenance and data organization practices
- Created & configured an extendable dataset using subdatasets
- Added code and data to the subdatasets
- Recorded provenance of analysis scripts
- Viewed the data provenance
- Rerun scripts on “fixed” data
- Added an external dataset and data
- Created and shared data between siblings

Next Steps

- Containerized tasks
  - Wrap each tool into a container
  - Simplifies data reproducibility and tool sharing
- Workflows
  - Automate a collection of tasks
- RIA (Remote Indexed Archive)
  - Acts like Github/GitLab
- Alternative data storage
  - S3, Dropbox, Microsoft OneDrive, SMB

Additional support

- Learn more at www.datalad.org
- Request access to slides and Demo code:
  david.deepwell@ucalgary.ca
- For more research data management support:
  - Library: research.data@ucalgary.libanswers.com
  - RCS: support@hpc.ucalgary.ca