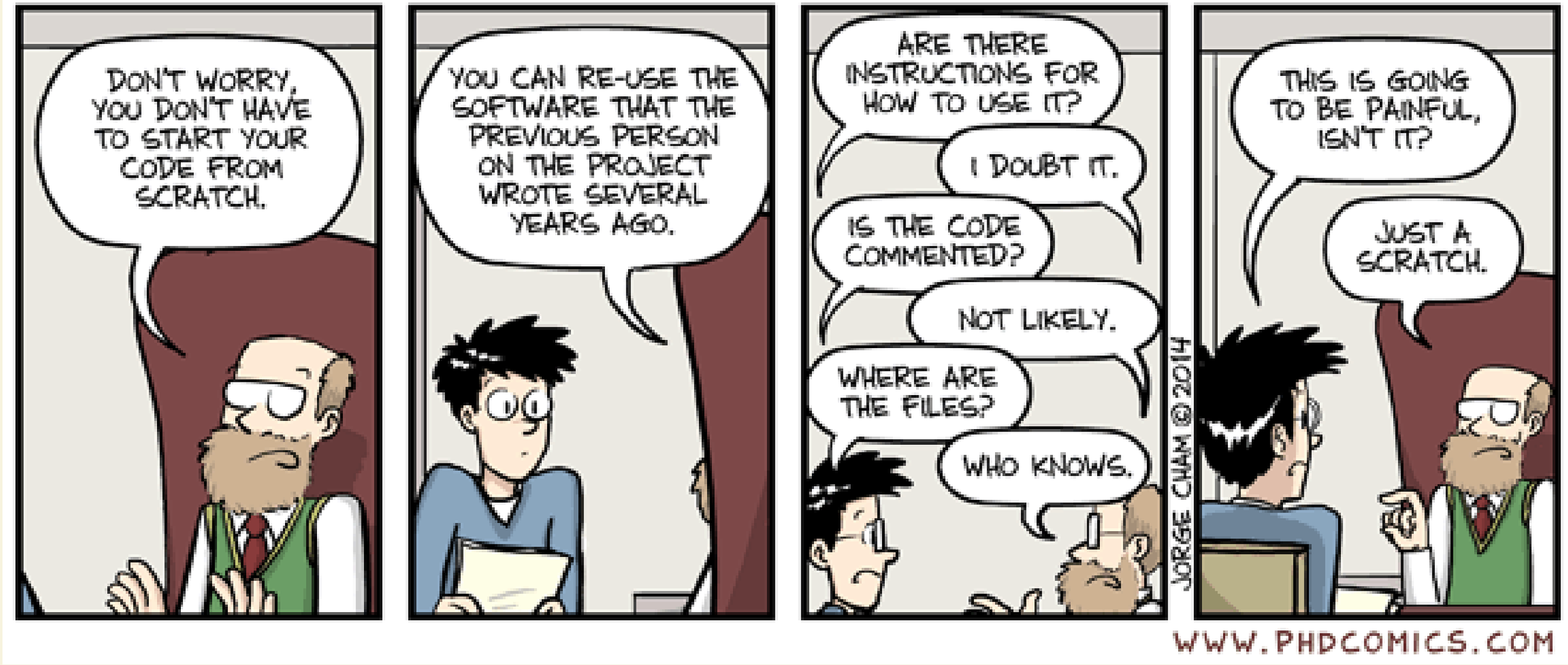
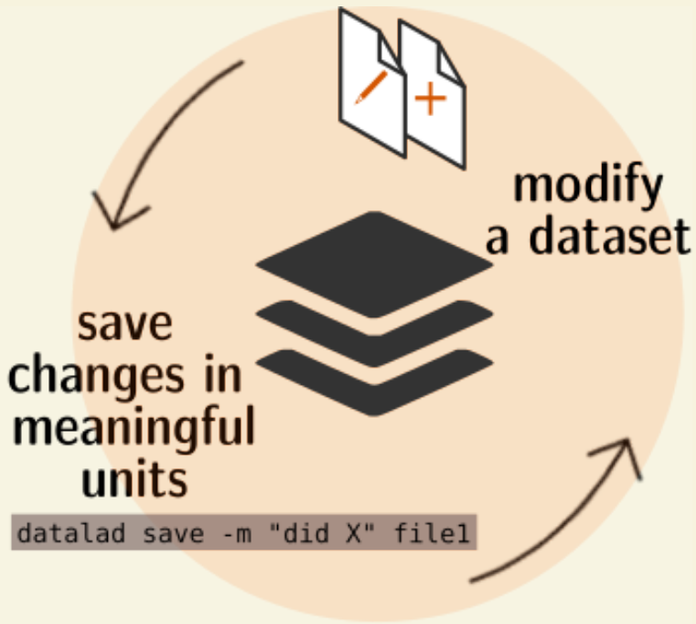
Data management

Reproducible analysis



LAST SESSION...

- **Local version control**: `datalad create`, `datalad save`
- **Datasets**: Exploring & nesting datasets
- **Practice**: Create, structure, and install datasets



CLARIFICATION

Do I have to do all modifications to files or the dataset from the comnand line?

CLARIFICATION

Do I have to do all modifications to files or the dataset from the comnand line?

No!

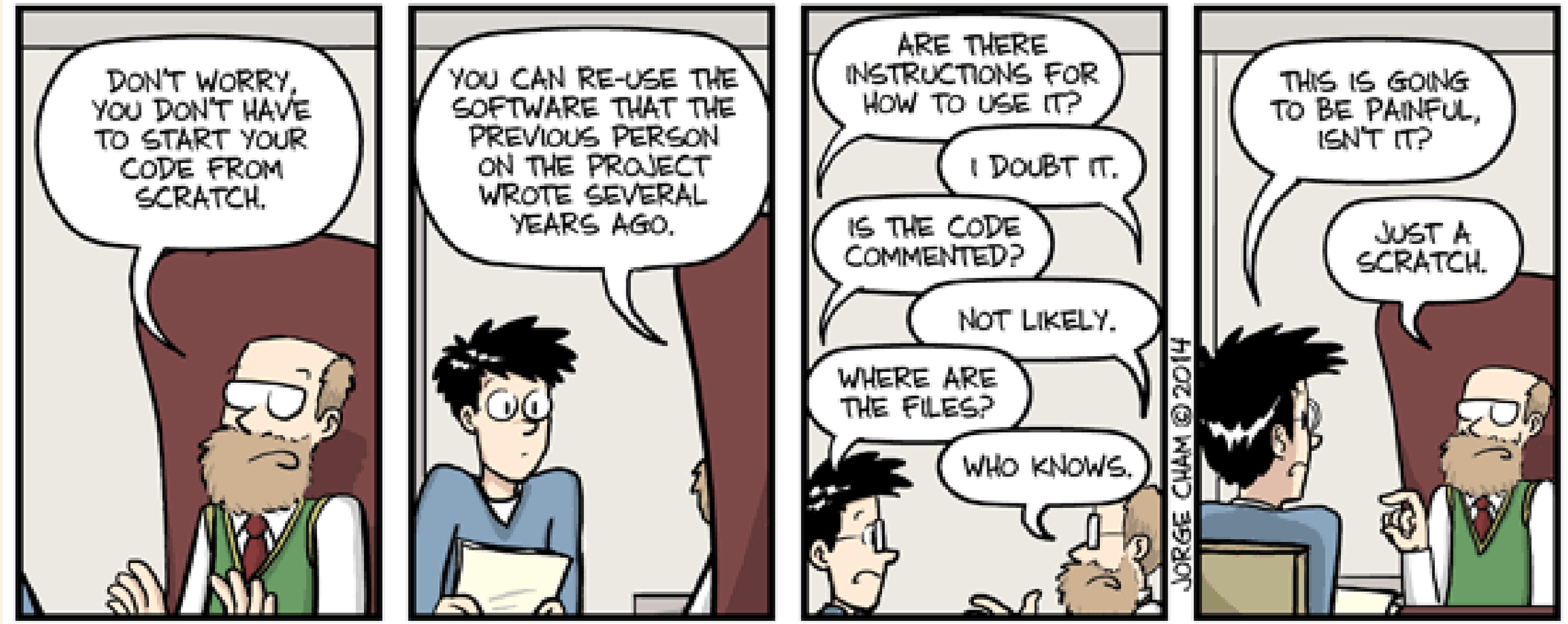
CLARIFICATION

Do I have to do all modifications to files or the dataset from the comnand line?

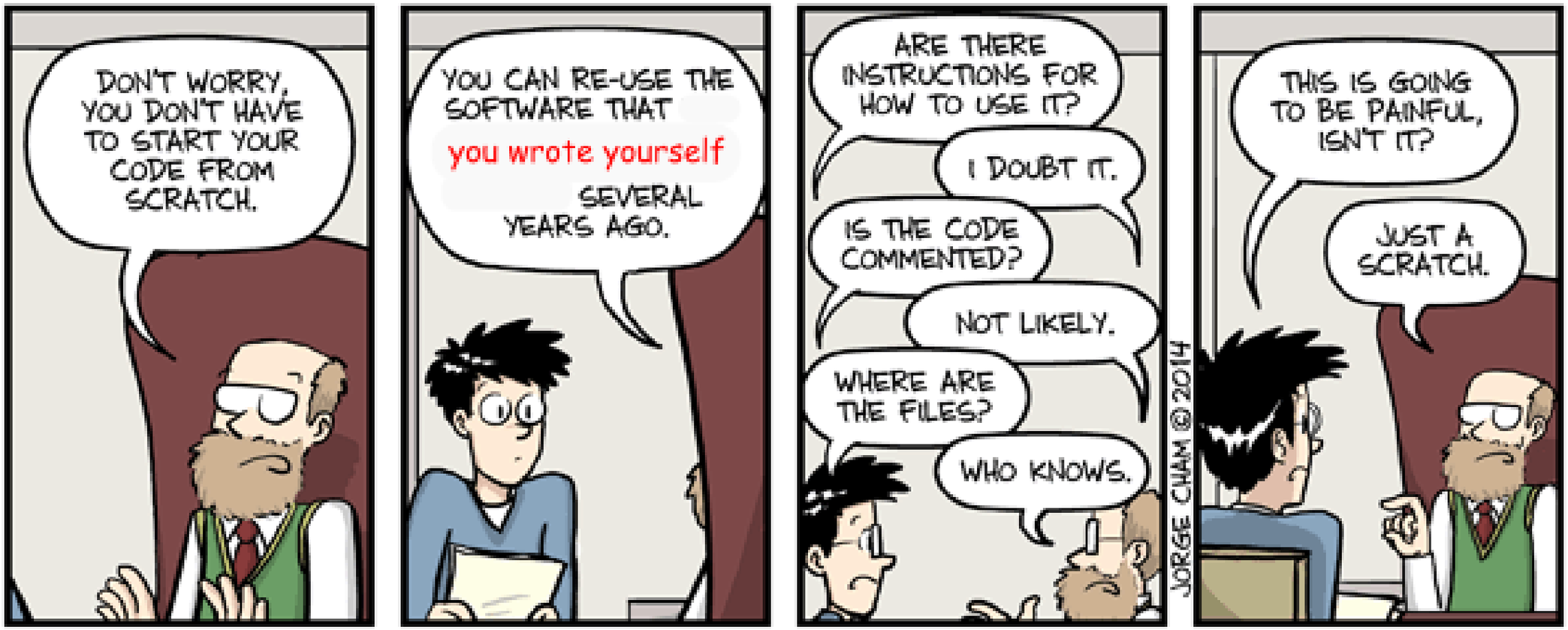
No!

ANY OTHER QUESTIONS?

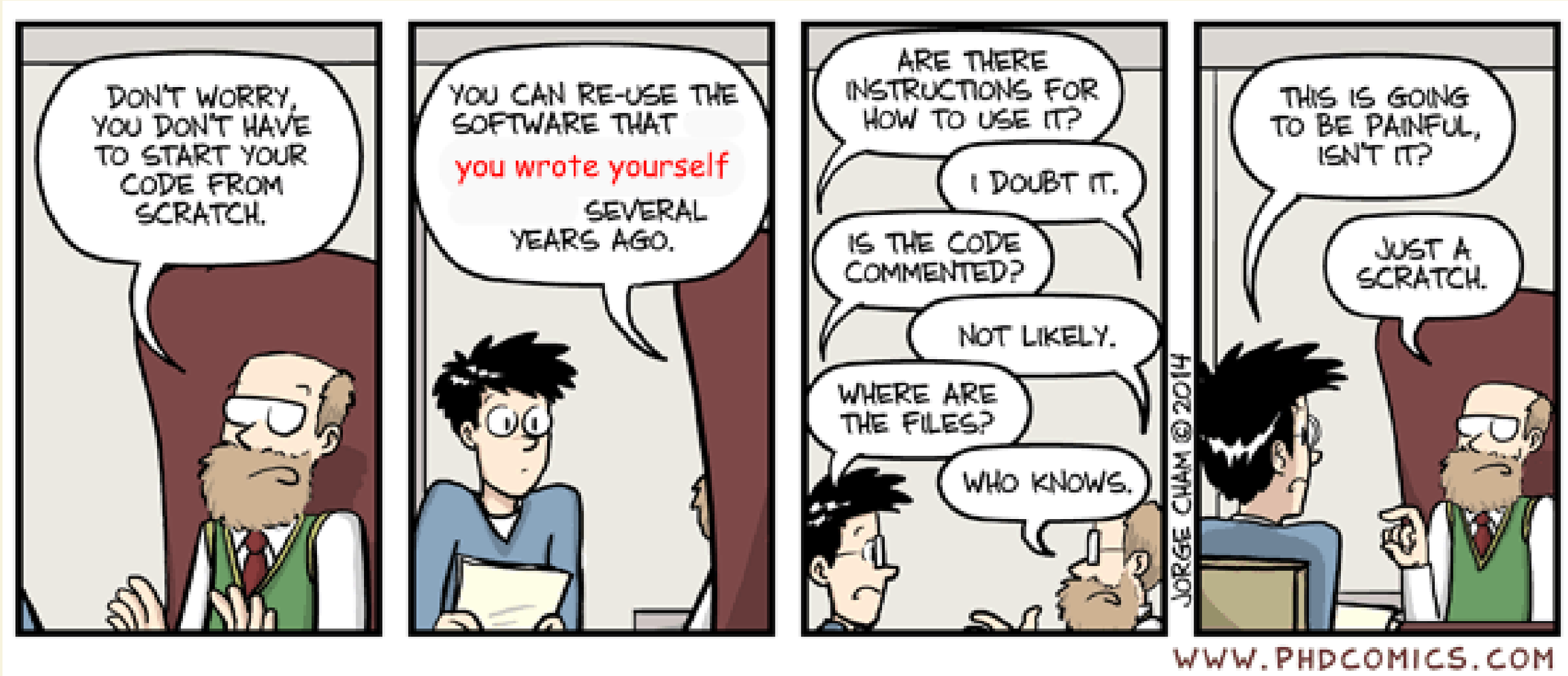
REPRODUCIBLE ANALYSES ARE HARD...



REPRODUCIBLE ANALYSES ARE HARD...



REPRODUCIBLE ANALYSES ARE HARD...



- Which script/unrecorded point-and-click in a GUI produced this figure?

REPRODUCIBLE ANALYSES ARE HARD...

- If I found the correct script, *what version* of this script produced the result?

REPRODUCIBLE ANALYSES ARE HARD...

- If I know the correct code and correct version, *what input data* are all these outputs based on?

REPRODUCIBLE ANALYSES ARE HARD...

- And finally: *How* do I execute the analysis script I want to recompute?

AFTER A WHILE...

AFTER A WHILE...

OBJECTIVES

Reproducible analysis

➲ Executing scripts and command line tools reproducibly with `datalad run`

OBJECTIVES

Reproducible analysis

- ➲ Executing scripts and command line tools reproducibly with `datalad run`
- ➲ Experience all the ways in which the command can fail

OBJECTIVES

Reproducible analysis

- ➲ Executing scripts and command line tools reproducibly with `datalad run`
- ➲ Experience all the ways in which the command can fail

Let's get into a terminal!

OBJECTIVES

Reproducible analysis

- ➲ Executing scripts and command line tools reproducibly with `datalad run`
- ➲ Experience all the ways in which the command can fail

Let's get into a terminal!

➲ handbook chapter

OBJECTIVES

Reproducible analysis

- ➲ Executing scripts and command line tools reproducibly with `datalad run`
- ➲ Experience all the ways in which the command can fail

Let's get into a terminal!

➲ handbook chapter

➲ code snippet list

REMINDER: DATALAD-101 NARRATIVE

REMINDER: DATALAD-101 NARRATIVE

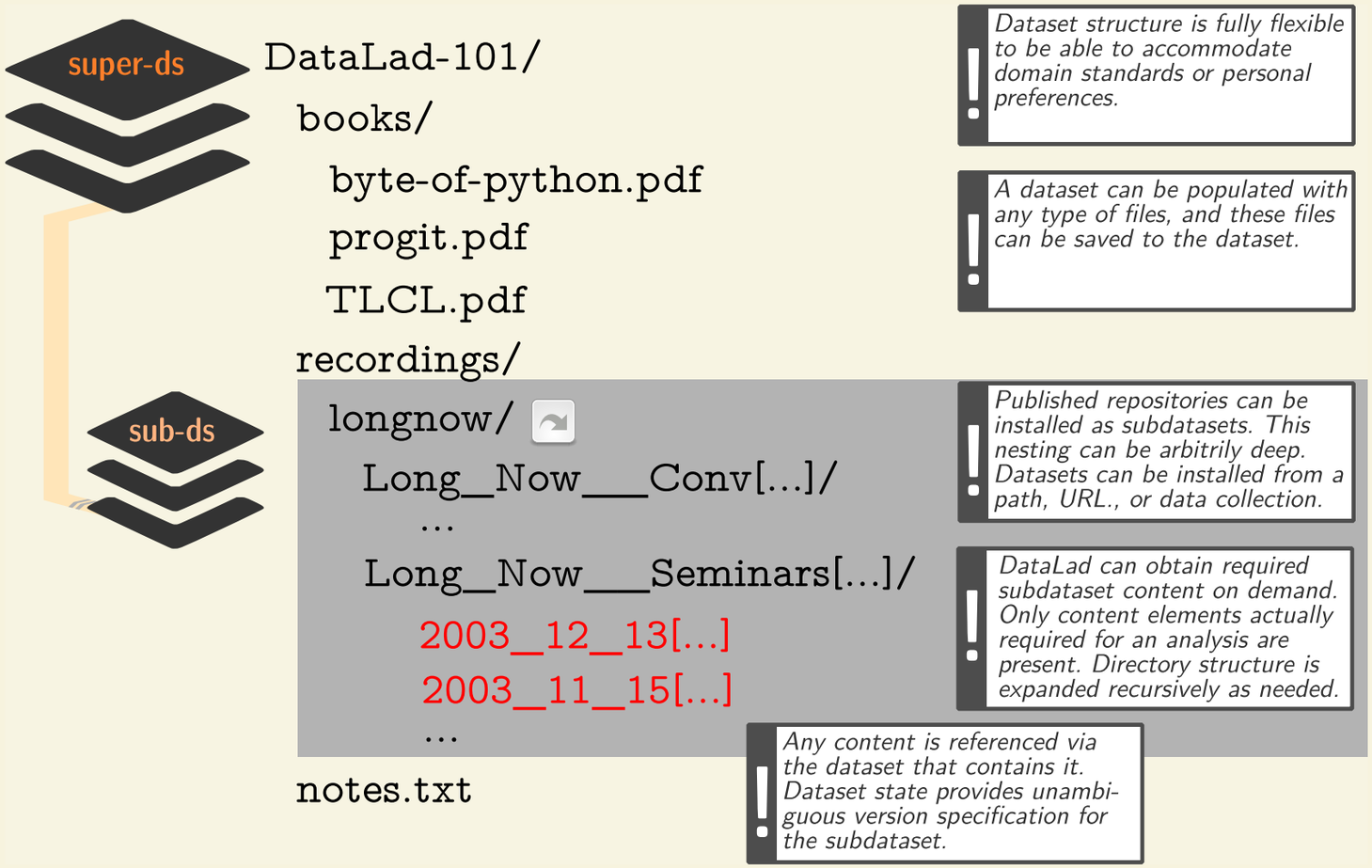
- Fictional educational course on DataLad

REMINDER: DATALAD-101 NARRATIVE

- Fictional educational course on DataLad
- Downloaded and saved PDFs, created and modified notes in text file, installed a subdataset with hundreds of podcasts.

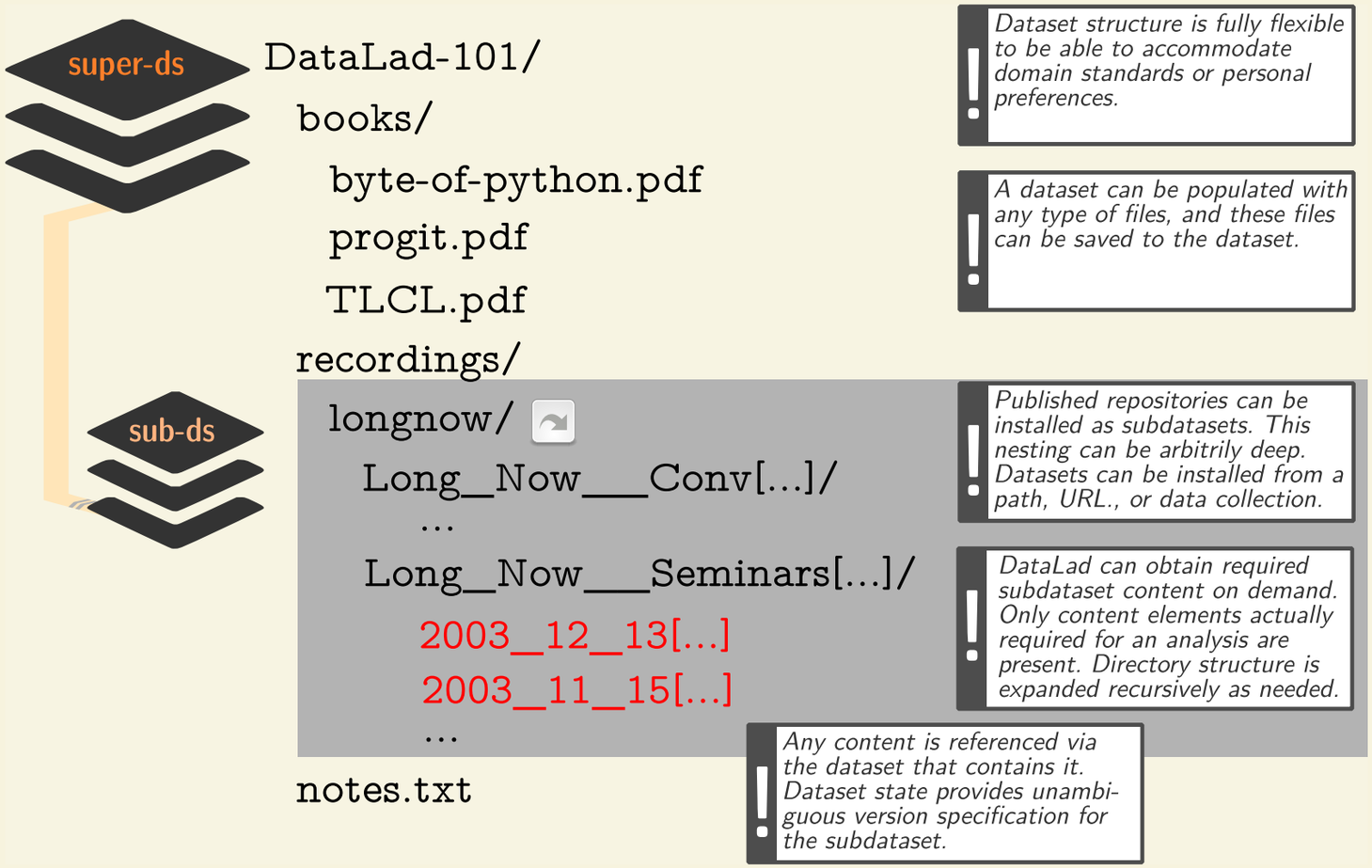
REMINDER: DATALAD-101 NARRATIVE

- Fictional educational course on DataLad
- Downloaded and saved PDFs, created and modified notes in text file, installed a subdataset with hundreds of podcasts.



REMINDER: DATALAD-101 NARRATIVE

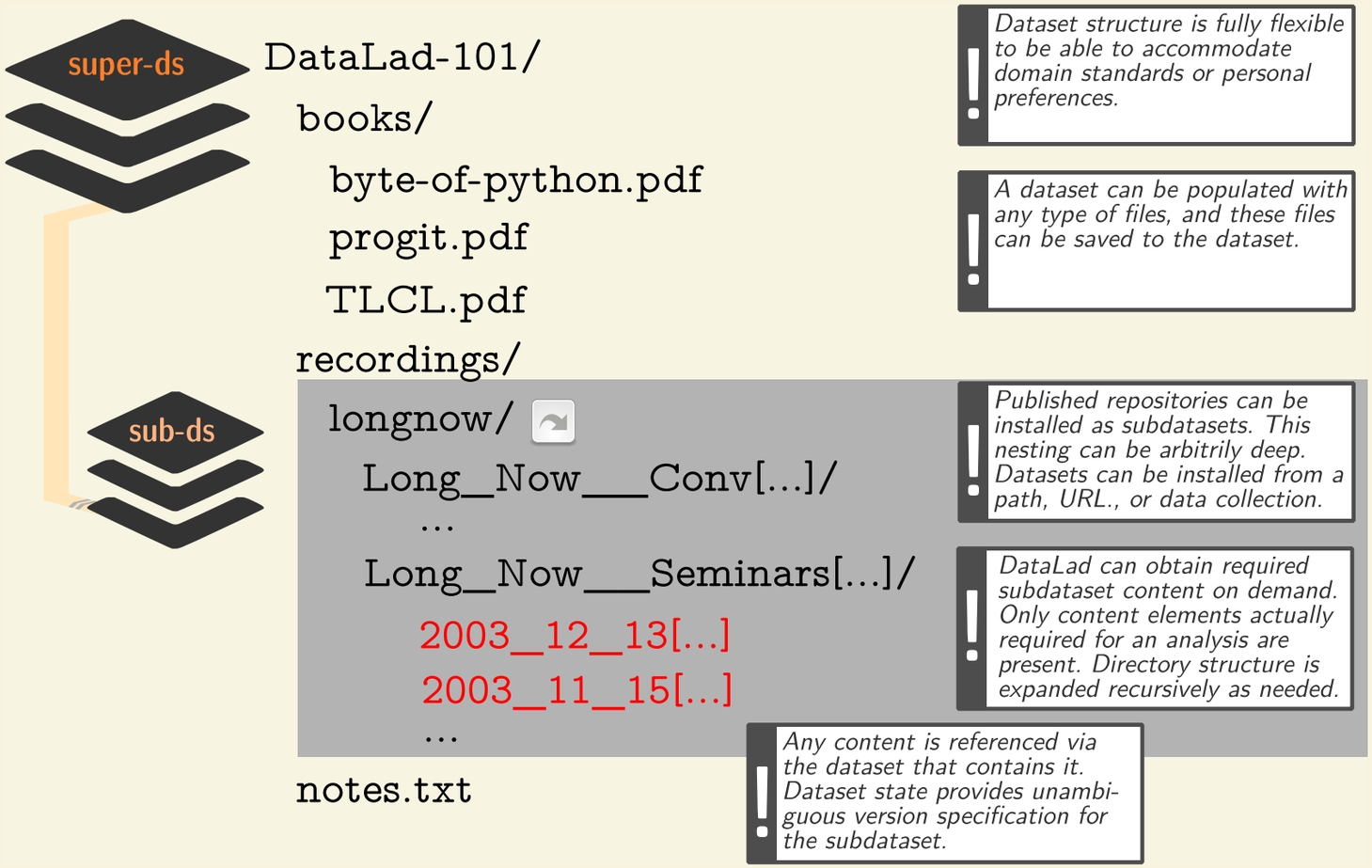
- Fictional educational course on DataLad
- Downloaded and saved PDFs, created and modified notes in text file, installed a subdataset with hundreds of podcasts.



Analysis idea: Write a list of all podcast titles!

REMINDER: DATALAD-101 NARRATIVE

- Fictional educational course on DataLad
- Downloaded and saved PDFs, created and modified notes in text file, installed a subdataset with hundreds of podcasts.



Analysis idea: Write a list of all podcast titles!

If you do not have this directory set up from last time, here is the code that gets you there!

```
for i in recordings/longnow/Long_Now__Seminars*/*.mp3; do
   # get the filename
   base=\$(basename "\$i");
   # strip the extension
   base=\${base%.mp3};
   # date as yyyy-mm-dd
   printf "\${base%%__*}\t" | tr '_' '-';
   # name and title without underscores
   printf "\${base#*__}\n" | tr '_' ' ';
done
```

```
for i in recordings/longnow/Long_Now__Seminars*/*.mp3; do
   # get the filename
   base=\$(basename "\$i");
   # strip the extension
   base=\${base%.mp3};
   # date as yyyy-mm-dd
   printf "\${base%%__*}\t" | tr '_' '-';
   # name and title without underscores
   printf "\${base#*__}\n" | tr '_' ' ';
done
```

➲ A for loop in `shell`, will print each file name as **Date - Speaker - Title** to the terminal.

```
for i in recordings/longnow/Long_Now__Seminars*/*.mp3; do
   # get the filename
   base=\$(basename "\$i");
   # strip the extension
   base=\${base%.mp3};
   # date as yyyy-mm-dd
   printf "\${base%%__*}\t" | tr '_' '-';
   # name and title without underscores
   printf "\${base#*__}\n" | tr '_' ' ';
done
```

➲ A for loop in `shell`, will print each file name as **Date - Speaker - Title** to the terminal.

➲ Redirection to a file with > writes the stream to a file instead of the terminal.

```
for i in recordings/longnow/Long_Now__Seminars*/*.mp3; do
   # get the filename
   base=\$(basename "\$i");
   # strip the extension
   base=\${base%.mp3};
   # date as yyyy-mm-dd
   printf "\${base%%__*}\t" | tr '_' '-';
   # name and title without underscores
   printf "\${base#*__}\n" | tr '_' ' ';
done
```

➲ A for loop in `shell`, will print each file name as **Date - Speaker - Title** to the terminal.

➲ Redirection to a file with > writes the stream to a file instead of the terminal.

➲ Note: This could be any script or shell command!

A BASIC DATALAD RUN COMMAND

Wrapping any command* in a **datalad run** will record the command's impact on the dataset to the history.

* Running scripts from the command line, using tools from the command line, ...

RUN-RECORDS LINK DATASET MODIFICATIONS TO COMMANDS

```
commit f4a35c8841062eb58f65dbf3cde70ccdc3c9df68 (HEAD -> master)
Author: Adina Wagner adina.wagner@t-online.de
Date:   Mon Nov 11 09:55:02 2019 +0100

    [DATALAD RUNCMD] create a list of podcast titles

    === Do not change lines below ===
    {
     "chain": [],
     "cmd": "bash code/list_titles.sh > recordings/podcasts.tsv",
     "dsid": "02a84dae-faf5-11e9-ba9f-e86a64c8054c",
     "exit": 0,
     "extra_inputs": [],
     "inputs": [],
     "outputs": [],
     "pwd": "."
    }
    ^^^ Do not change lines above ^^^

diff --git a/recordings/podcasts.tsv b/recordings/podcasts.tsv
new file mode 100644
index 0000000..f691b53
--- /dev/null
+++ b/recordings/podcasts.tsv
@@ -0,0 +1,206 @@
+2003-11-15     Brian Eno  The Long Now
+2003-12-13     Peter Schwartz  The Art Of The Really Long View
+2004-01-10     George Dyson  There s Plenty of Room at the Top  Long term Thinking About Large scale Comp
[...]
```

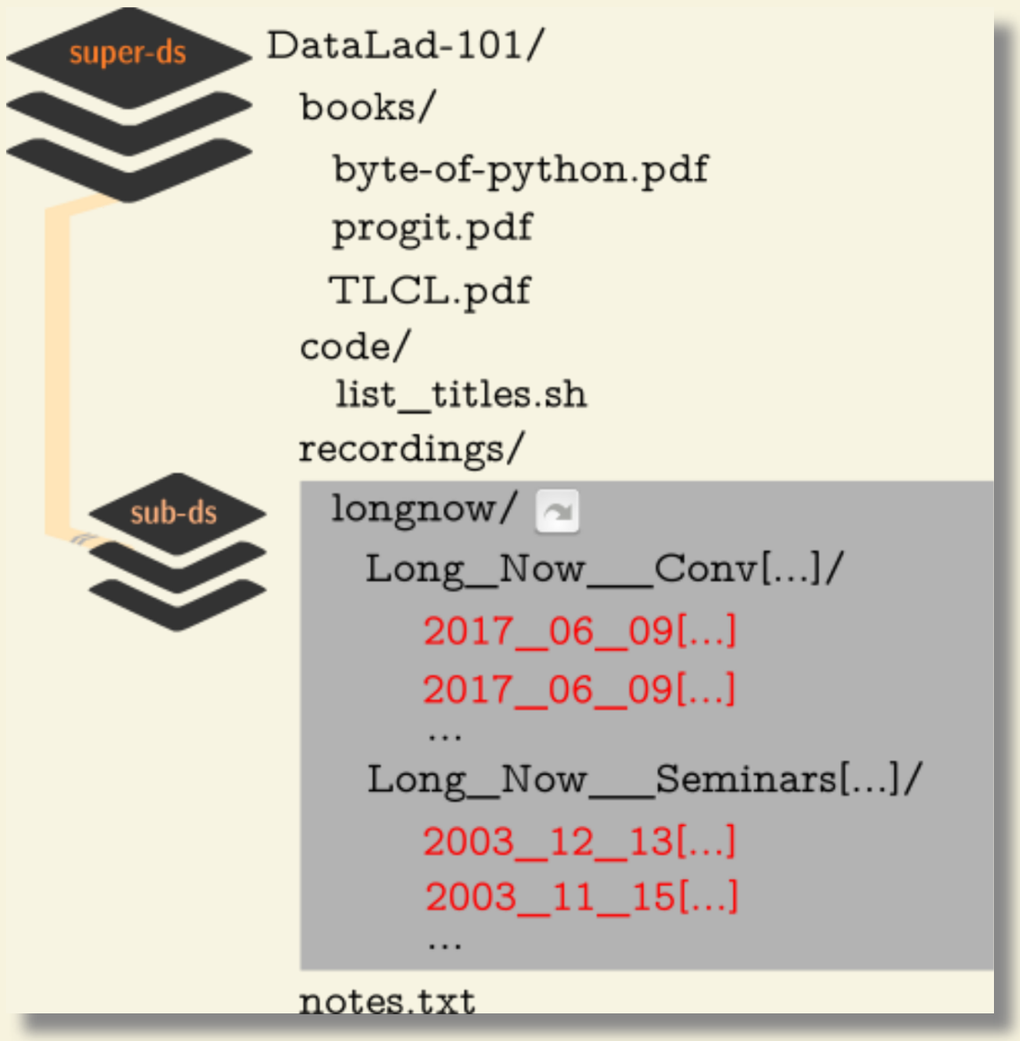
RUN-RECORDS LINK DATASET MODIFICATIONS TO COMMANDS

```
commit f4a35c8841062eb58f65dbf3cde70ccdc3c9df68 (HEAD -> master)
Author: Adina Wagner adina.wagner@t-online.de
Date:   Mon Nov 11 09:55:02 2019 +0100

    [DATALAD RUNCMD] create a list of podcast titles

    === Do not change lines below ===
    {
     "chain": [],
     "cmd": "bash code/list_titles.sh > recordings/podcasts.tsv",
     "dsid": "02a84dae-faf5-11e9-ba9f-e86a64c8054c",
     "exit": 0,
     "extra_inputs": [],
     "inputs": [],
     "outputs": [],
     "pwd": "."
    }
    ^^^ Do not change lines above ^^^

diff --git a/recordings/podcasts.tsv b/recordings/podcasts.tsv
new file mode 100644
index 0000000..f691b53
--- /dev/null
+++ b/recordings/podcasts.tsv
@@ -0,0 +1,206 @@
+2003-11-15     Brian Eno  The Long Now
+2003-12-13     Peter Schwartz  The Art Of The Really Long View
+2004-01-10     George Dyson  There s Plenty of Room at the Top  Long term Thinking About Large scale Comp
[...]
```

It follows logically: If a command does **not** lead to any modification in a dataset, it will not be recorded!

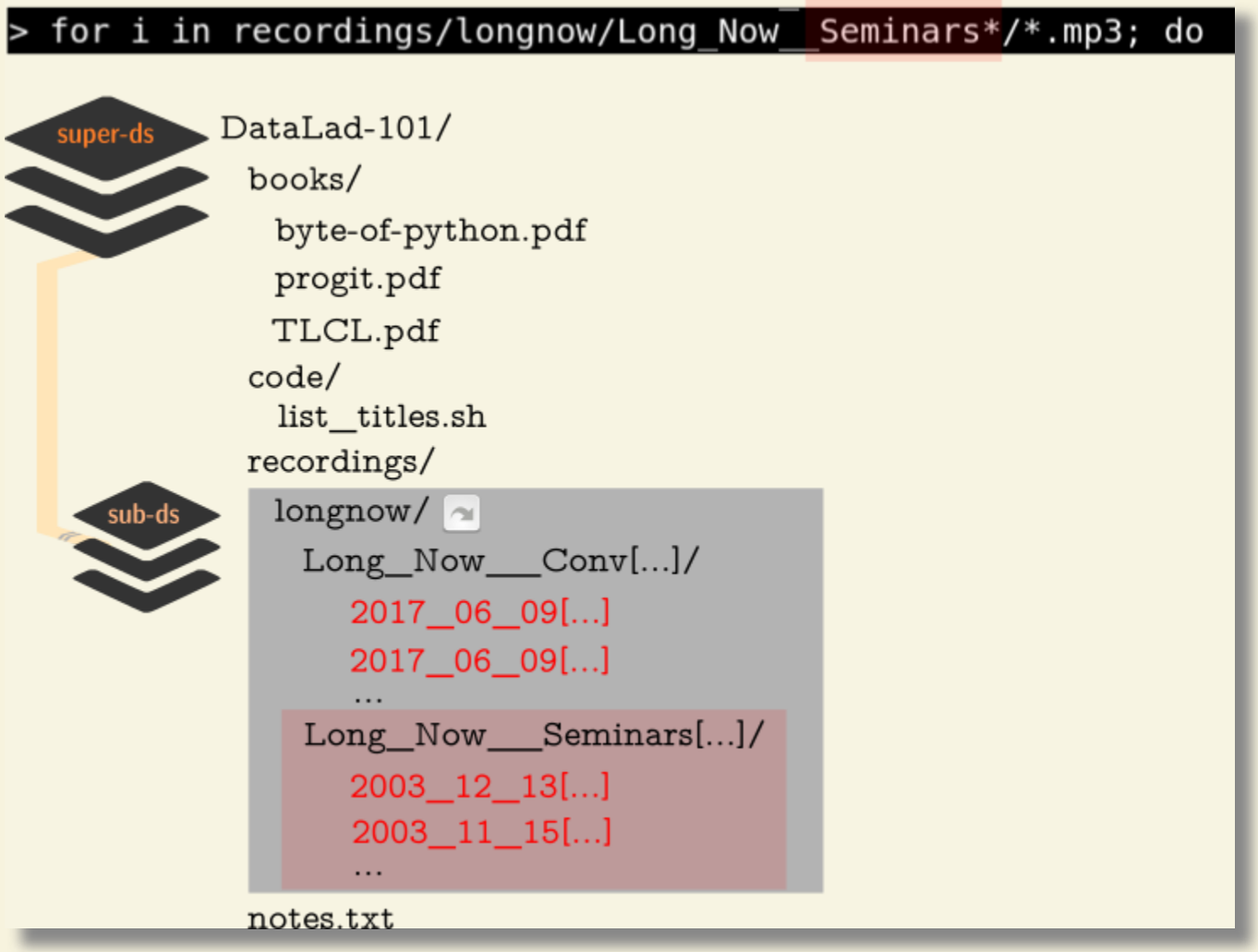
OH! AN ERROR IN THE CODE...

DataLad-101 layout:

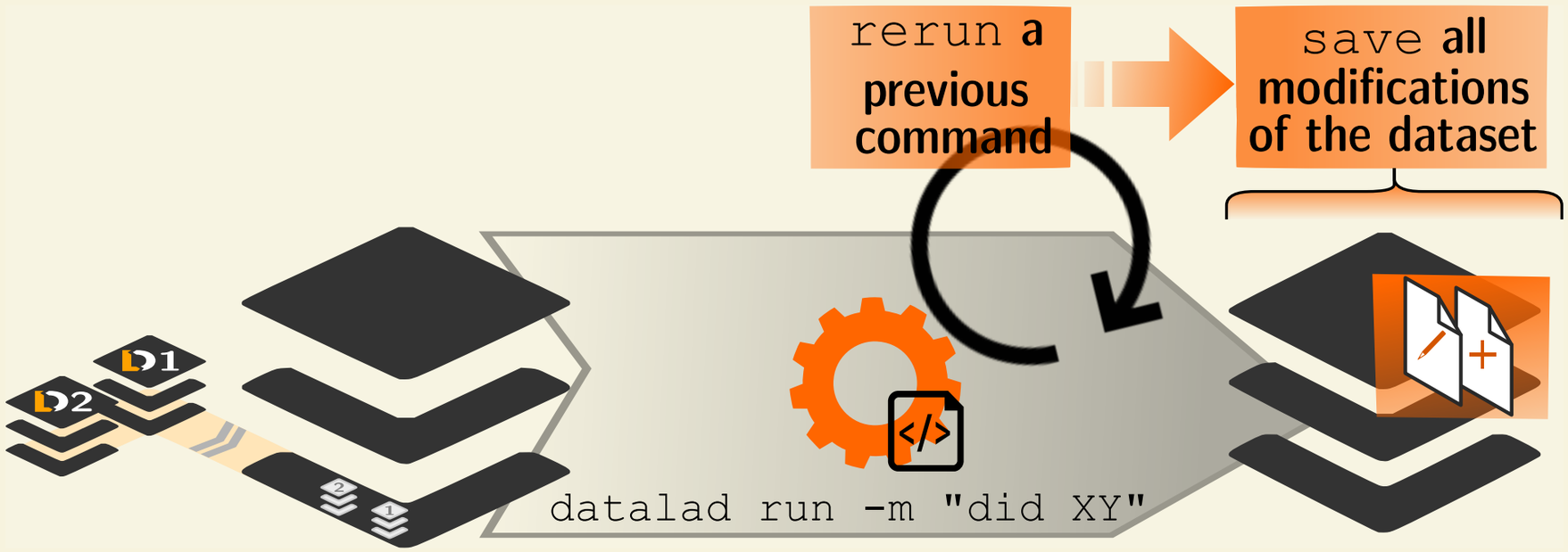


OH! AN ERROR IN THE CODE...

DataLad-101 layout:



DATALAD RERUN

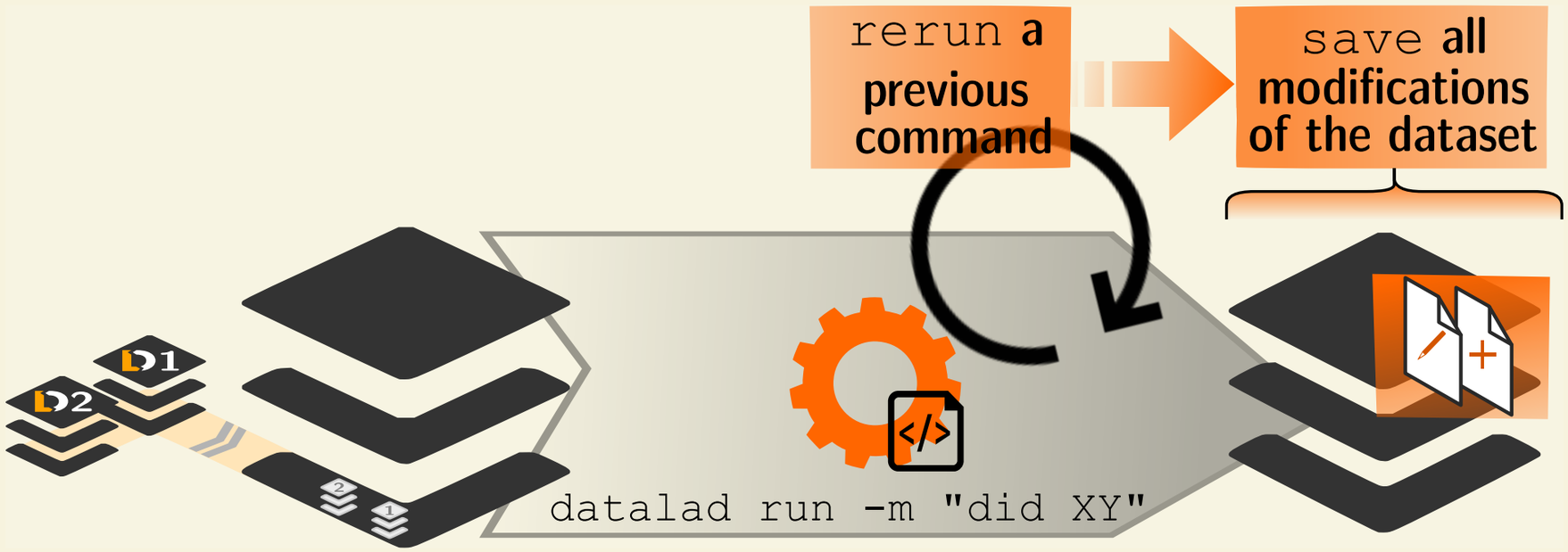


Re-execute previous datalad run commands

What shall be rerun can be specified via its commit hash:

```
datalad rerun f4a35c884106
```

DATALAD RERUN



Re-execute previous datalad run commands

What shall be rerun can be specified via its commit hash:

```
datalad rerun f4a35c884106
```

... but also via tag, revision specifications with HEAD, ..., or by giving a range of commits.

SUMMARY - BASIC DATALAD RUN

SUMMARY - BASIC DATALAD RUN

`datalad run` records a commands impact on a dataset.

SUMMARY - BASIC DATALAD RUN

`datalad run` records a commands impact on a dataset.

A record is only made if the command leads to dataset modifications

SUMMARY - BASIC DATALAD RUN

`datalad run` records a commands impact on a dataset.

A record is only made if the command leads to dataset modifications

The command captures provenance for humans and machines

SUMMARY - BASIC DATALAD RUN

`datalad run` records a commands impact on a dataset.

A record is only made if the command leads to dataset modifications

The command captures provenance for humans and machines

a machine-readable **runrecord** is automatically created, *you* need to provide a **commit message**.

SUMMARY - BASIC DATALAD RUN

`datalad run` records a commands impact on a dataset.

A record is only made if the command leads to dataset modifications

The command captures provenance for humans and machines

a machine-readable **runrecord** is automatically created, *you* need to provide a **commit message**.

`datalad rerun` can take any previous `datalad run` commit hash and re-execute it.

SUMMARY - BASIC DATALAD RUN

`datalad run` records a commands impact on a dataset.

A record is only made if the command leads to dataset modifications

The command captures provenance for humans and machines

a machine-readable **runrecord** is automatically created, *you* need to provide a **commit message**.

`datalad rerun` can take any previous `datalad run` commit hash and re-execute it.

This saves you the need to remember!

SUMMARY - BASIC DATALAD RUN

`datalad run` records a commands impact on a dataset.

A record is only made if the command leads to dataset modifications

The command captures provenance for humans and machines

a machine-readable **runrecord** is automatically created, *you* need to provide a **commit message**.

`datalad rerun` can take any previous `datalad run` commit hash and re-execute it.

This saves you the need to remember!

`datalad diff` and `git diff` are useful helpers to explore changes between version states of a dataset.

SUMMARY - BASIC DATALAD RUN

`datalad run` records a commands impact on a dataset.

A record is only made if the command leads to dataset modifications

The command captures provenance for humans and machines

a machine-readable **runrecord** is automatically created, *you* need to provide a **commit message**.

`datalad rerun` can take any previous `datalad run` commit hash and re-execute it.

This saves you the need to remember!

`datalad diff` and `git diff` are useful helpers to explore changes between version states of a dataset.

... but there is more that this command can do for you:

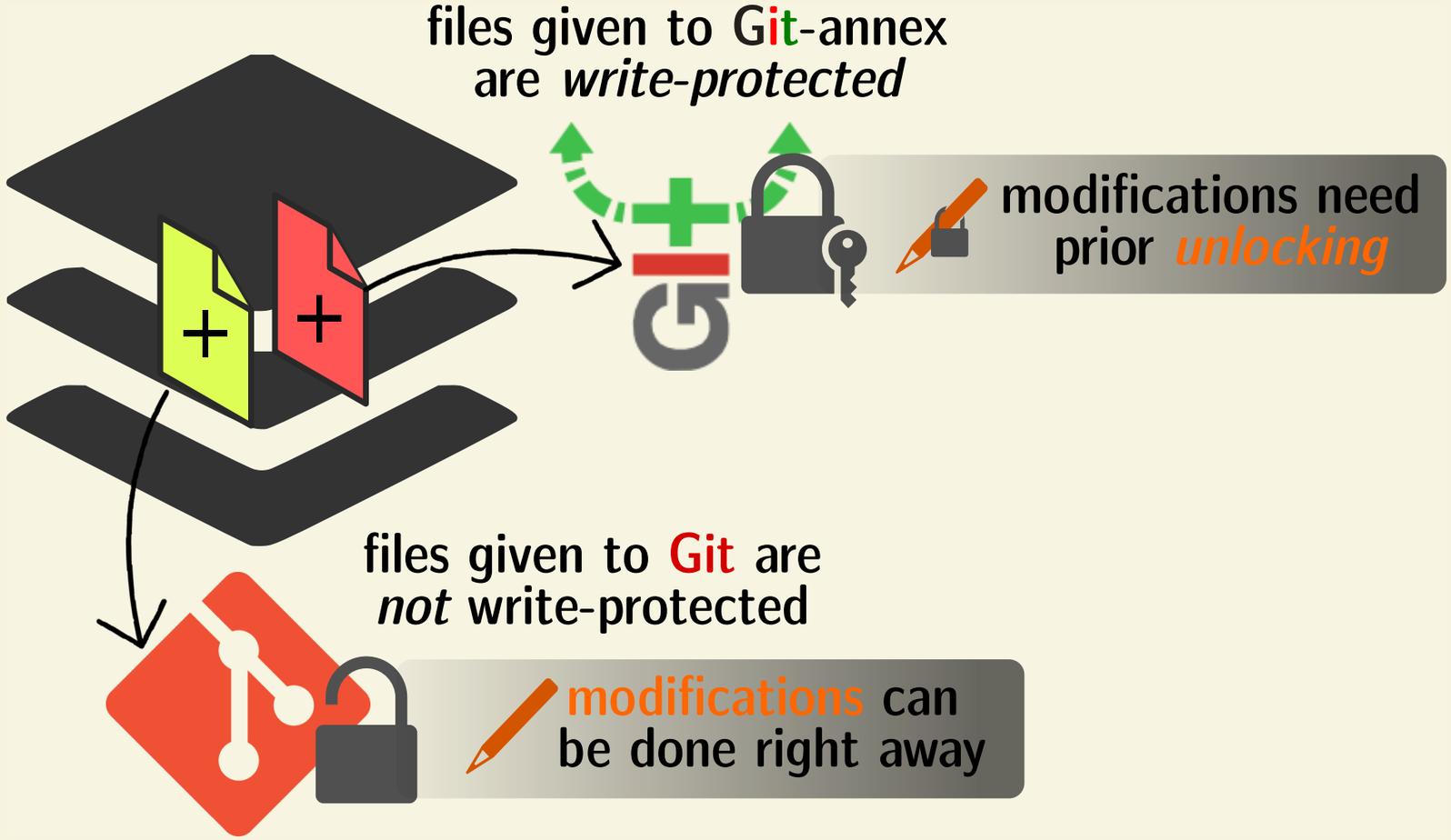
THE ANATOMY OF DATALAD ERROR MESSAGES

```
"convert -resize 400x400 recordings/longnow/.datalad/feed_metadata/logo_salt.jpg recordings/salt_logo_smal
[INFO   ] == Command start (output follows) =====
convert-im6.q16: unable to open image `recordings/longnow/.datalad/feed_metadata/logo_salt.jpg': No such f
convert-im6.q16: no images defined `recordings/salt_logo_small.jpg' @ error/convert.c/ConvertImageCommand/
[INFO   ] == Command exit (modification check follows) =====
[INFO   ] The command had a non-zero exit code. If this is expected, you can save the changes with 'datala
CommandError: command 'convert -resize 400x400 recordings/longnow/.datalad/feed_metadata/logo_salt.jpg rec
Failed to run 'convert -resize 400x400 recordings/longnow/.datalad/feed_metadata/logo_salt.jpg recordings/
```

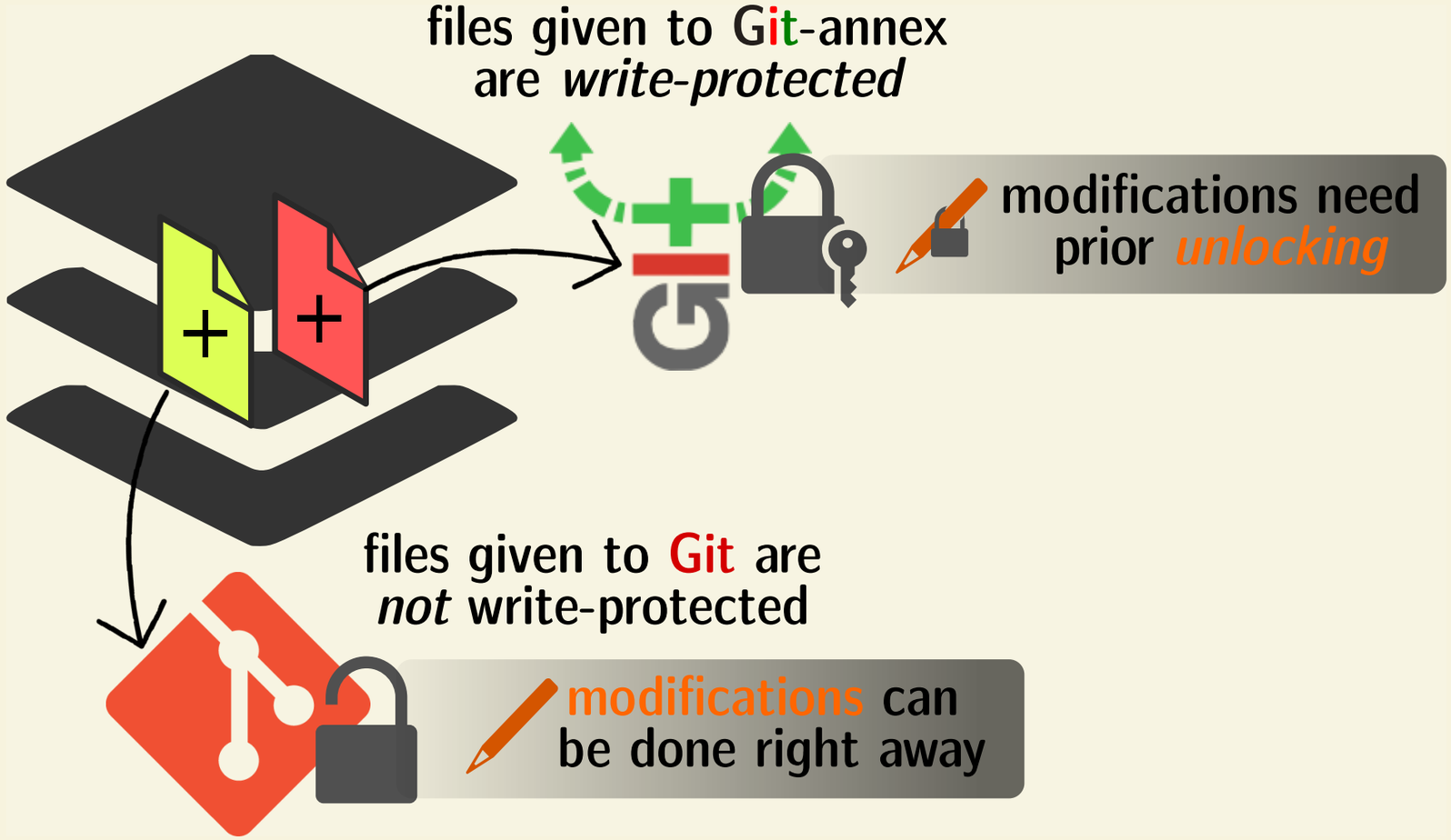
--INPUT IN DATALAD RUN

Files provided with the --input option are automatically retrieved with **datalad get**, if necessary.

CONTENT-LOCKED FILES (VASTLY SIMPLIFIED)

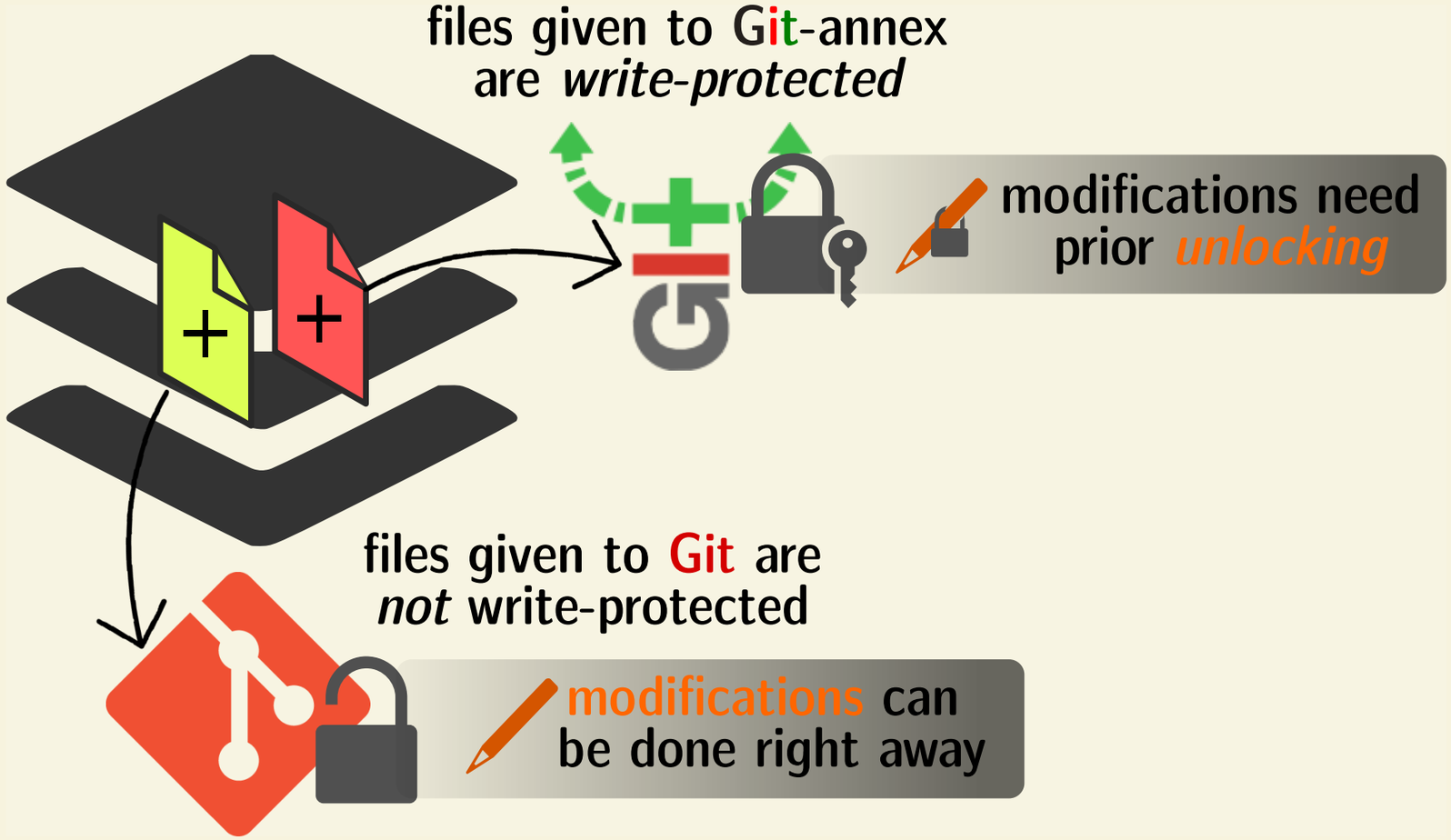


CONTENT-LOCKED FILES (VASTLY SIMPLIFIED)



Files are given to Git-annex or Git

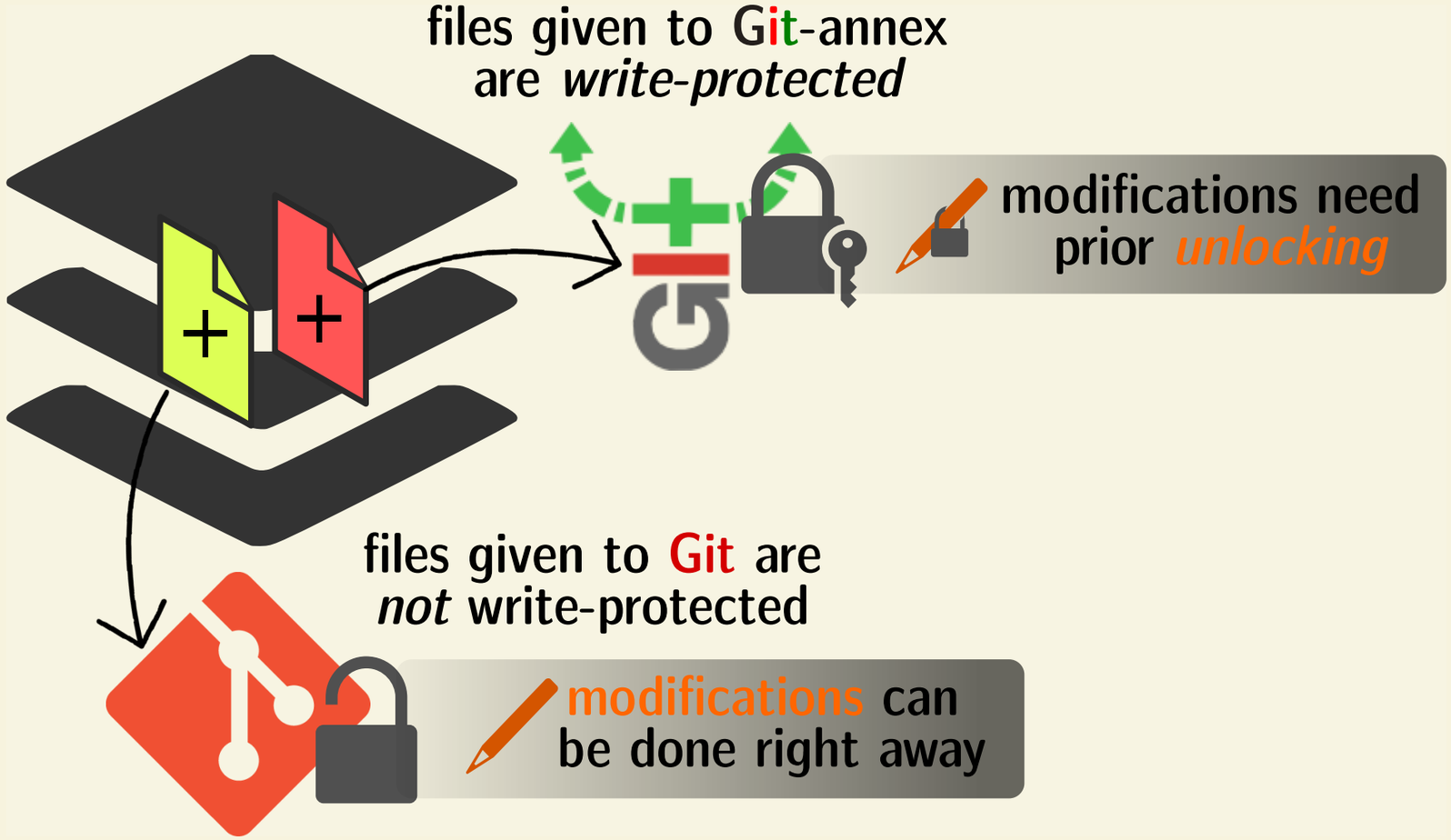
CONTENT-LOCKED FILES (VASTLY SIMPLIFIED)



Files are given to Git-annex or Git

Based on dataset configuration about file *type*, *size*, or *name*.

CONTENT-LOCKED FILES (VASTLY SIMPLIFIED)

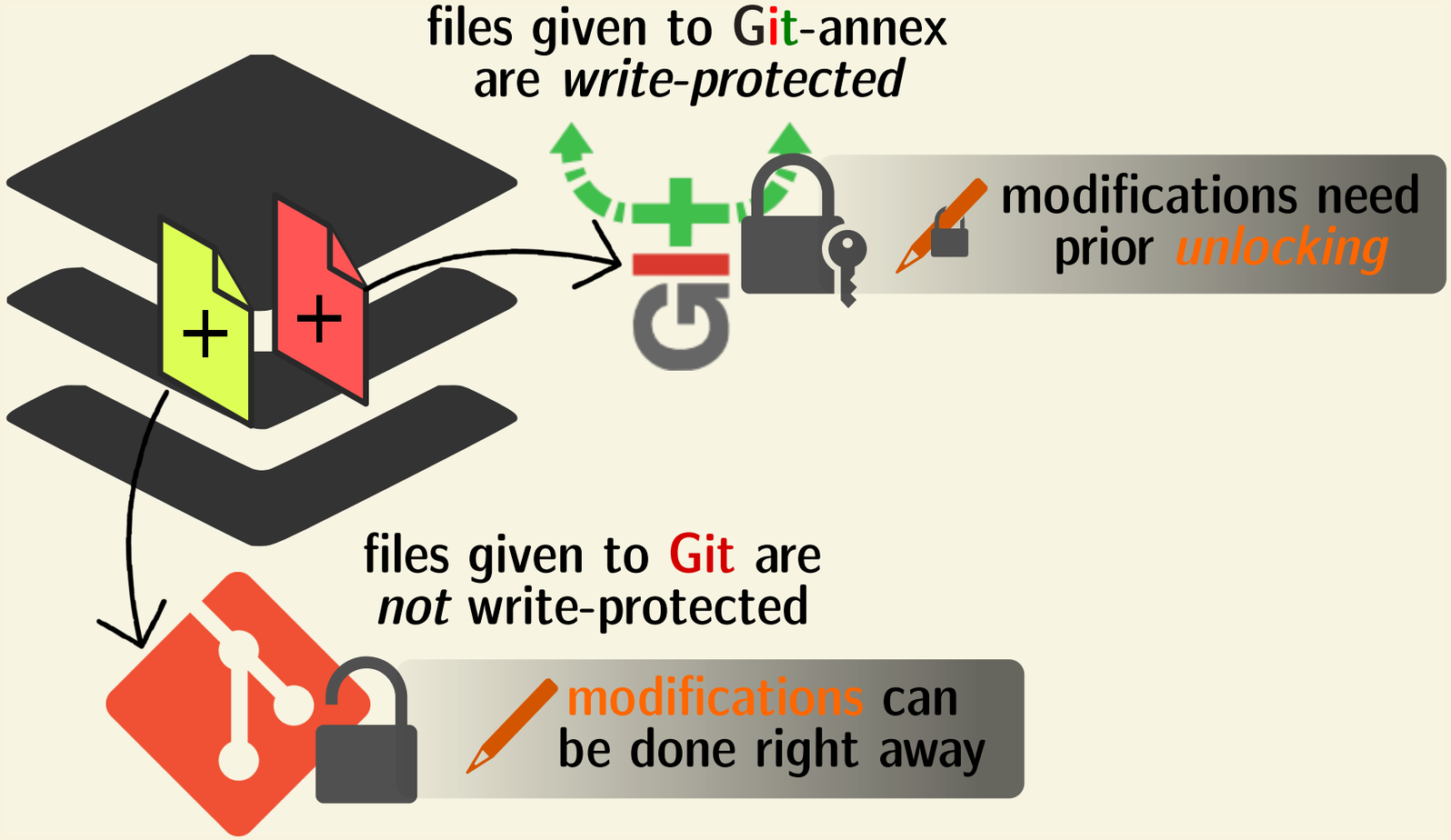


Files are given to Git-annex or Git

Based on dataset configuration about file *type*, *size*, or *name*.

Git-annex removes write permission from the file content it stores.

CONTENT-LOCKED FILES (VASTLY SIMPLIFIED)



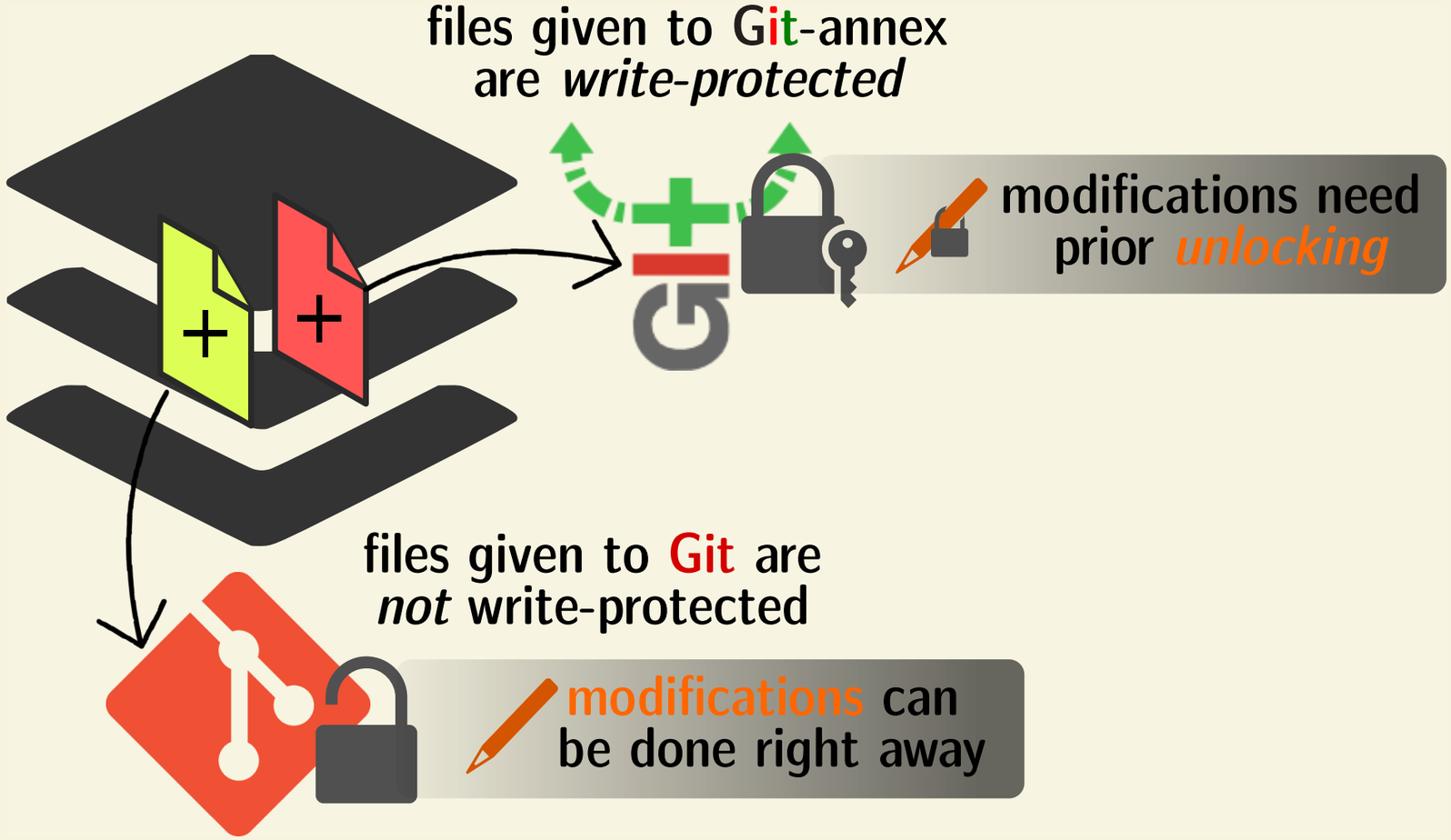
Files are given to Git-annex or Git

Based on dataset configuration about file *type*, *size*, or *name*.

Git-annex removes write permission from the file content it stores.

This prevents accidental modifications.

CONTENT-LOCKED FILES (VASTLY SIMPLIFIED)



Files are given to Git-annex or Git

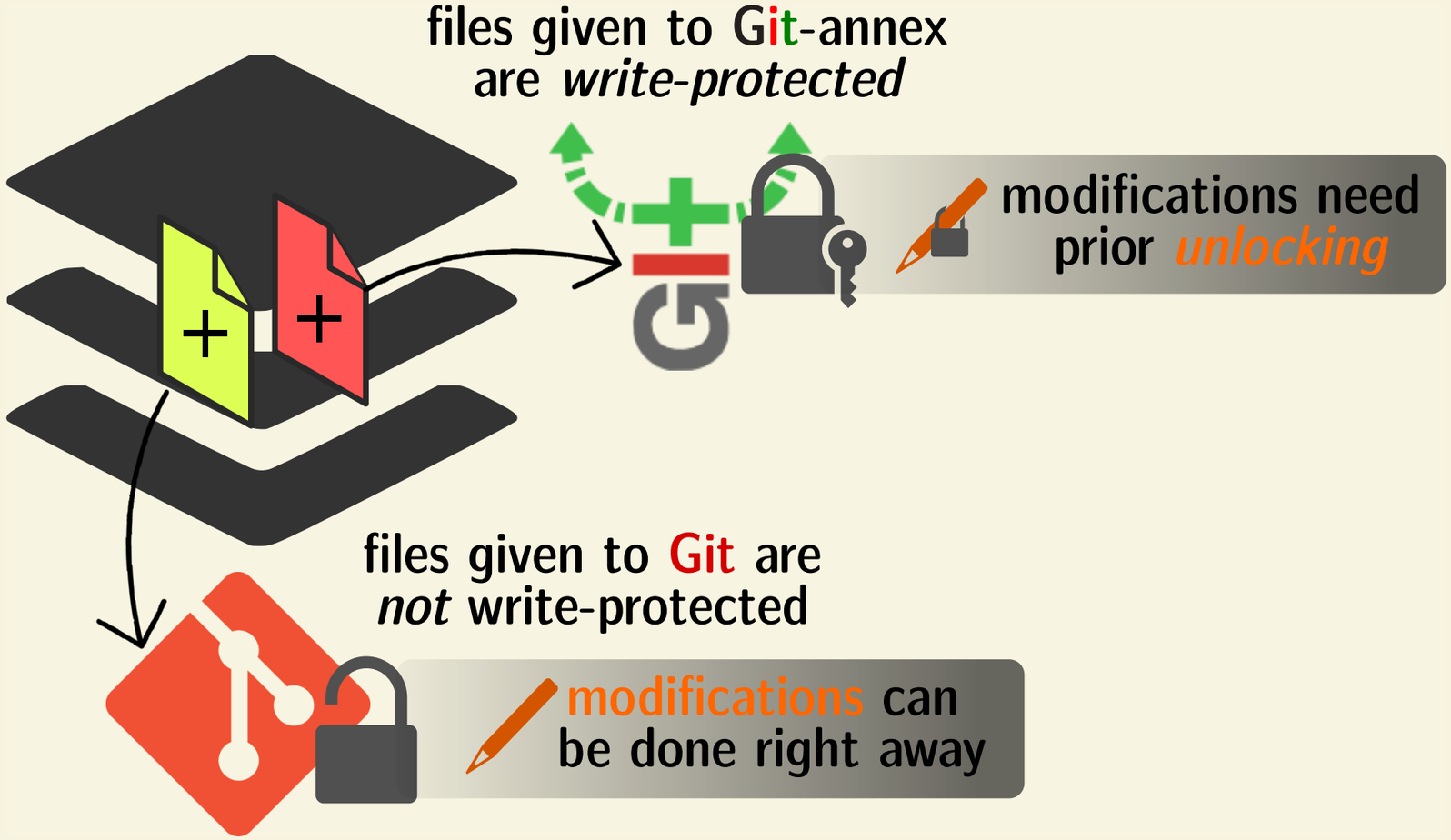
Based on dataset configuration about file *type*, *size*, or *name*.

Git-annex removes write permission from the file content it stores.

This prevents accidental modifications.

`datalad unlock` can unlock content for modification.

CONTENT-LOCKED FILES (VASTLY SIMPLIFIED)



Files are given to Git-annex or Git

Based on dataset configuration about file *type*, *size*, or *name*.

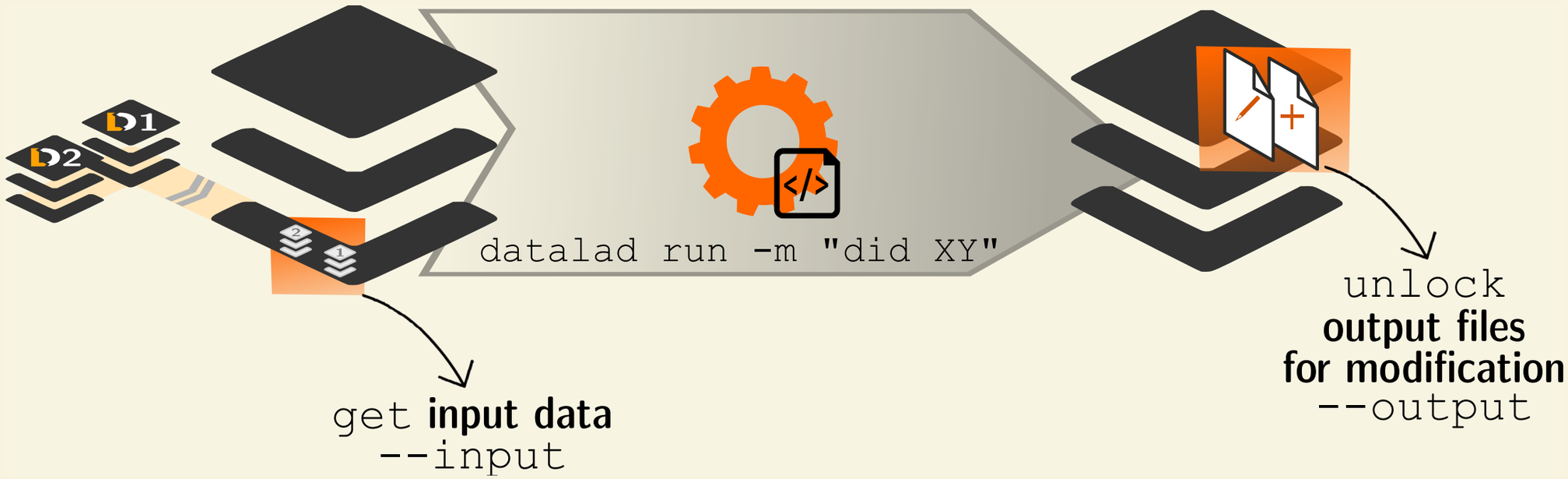
Git-annex removes write permission from the file content it stores.

This prevents accidental modifications.

`datalad unlock` can unlock content for modification.

`datalad save` will lock content again.

--OUTPUT IN DATALAD RUN



Files provided with the --output option are automatically unlocked for modification with **datalad unlock**, if necessary.

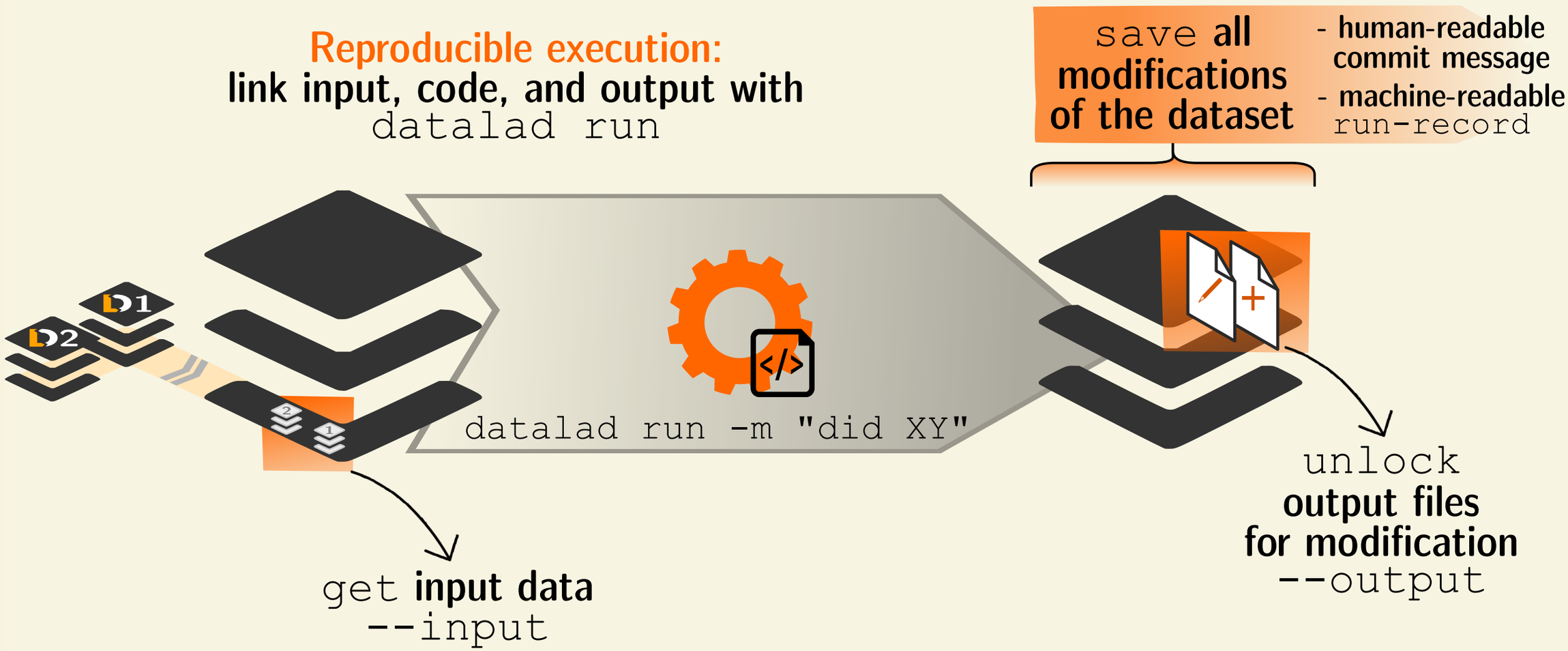
ANALYSIS PROVENANCE CAPTURE

Easy provanance capture!

ANALYSIS PROVENANCE CAPTURE

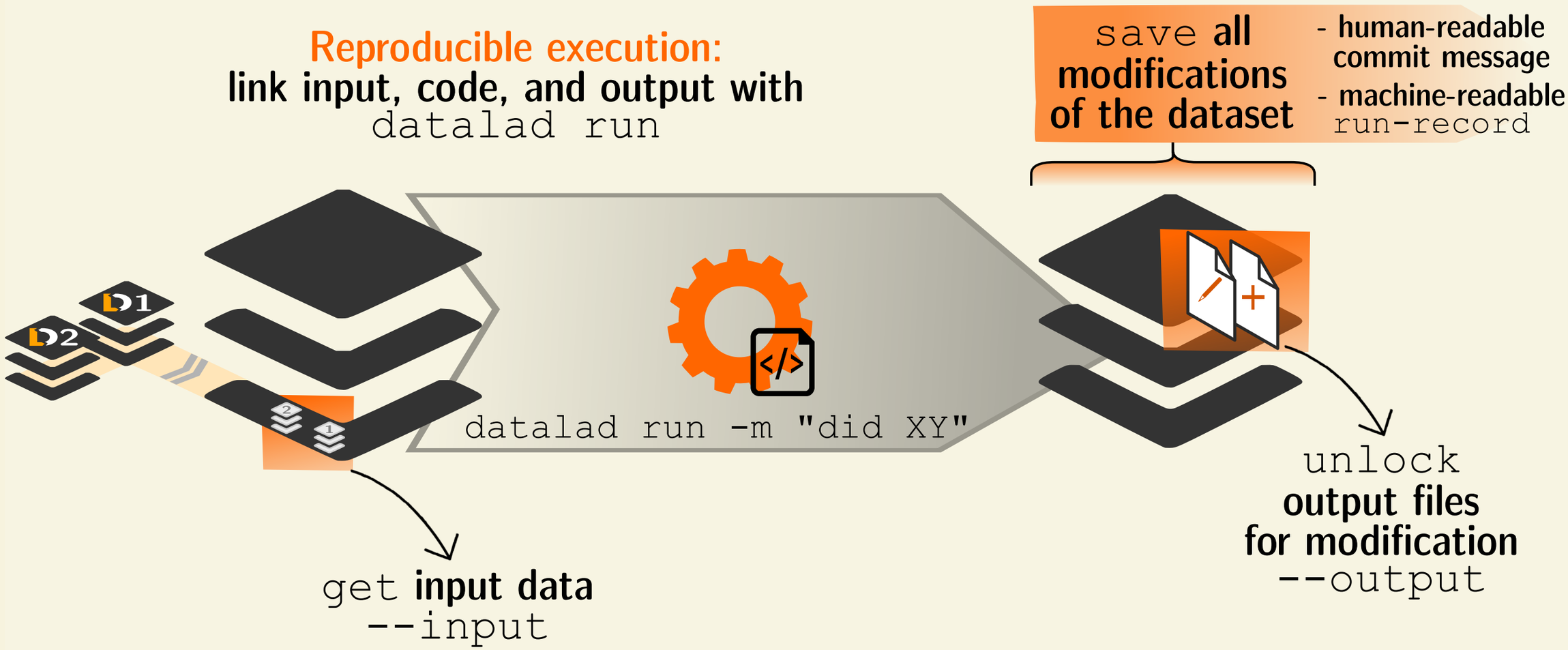
Easy provanance capture!

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



ANALYSIS PROVENANCE CAPTURE

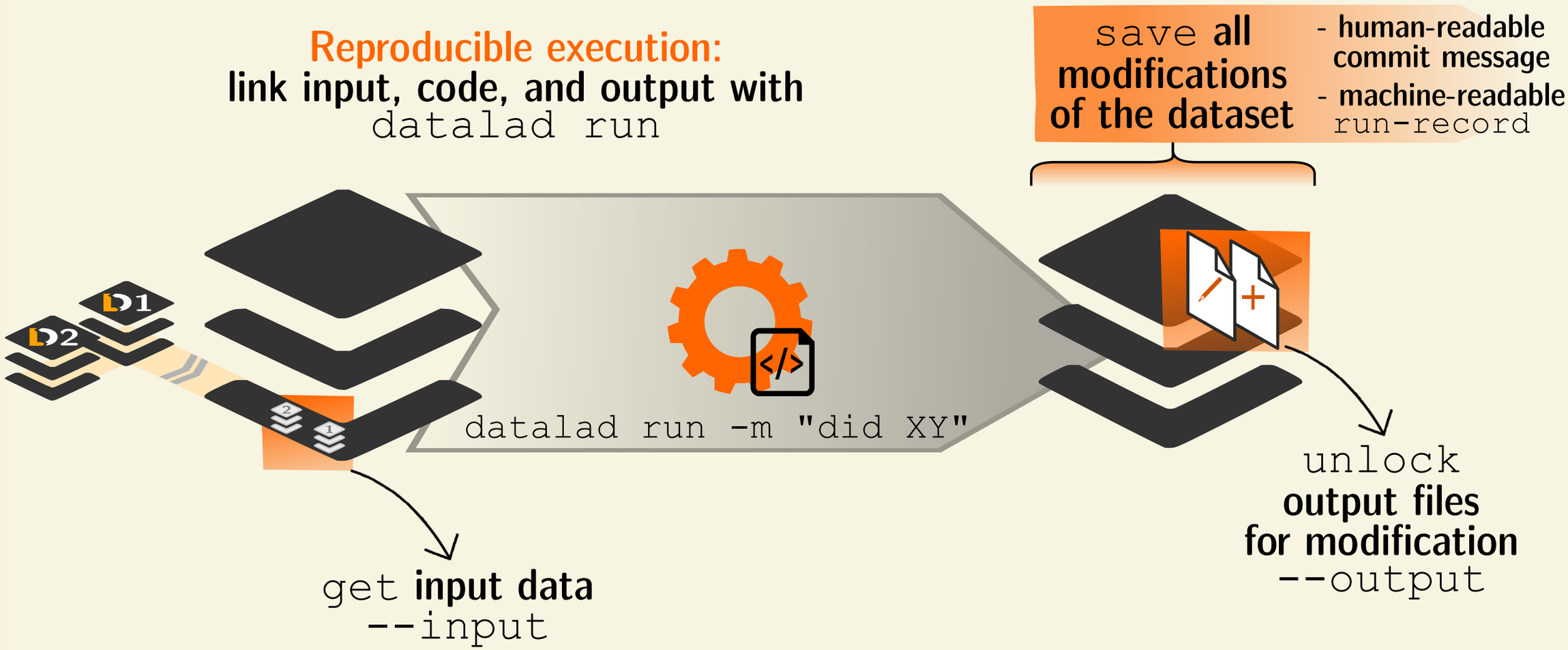
Easy provanance capture!



Advice:

ANALYSIS PROVENANCE CAPTURE

Easy provanance capture!

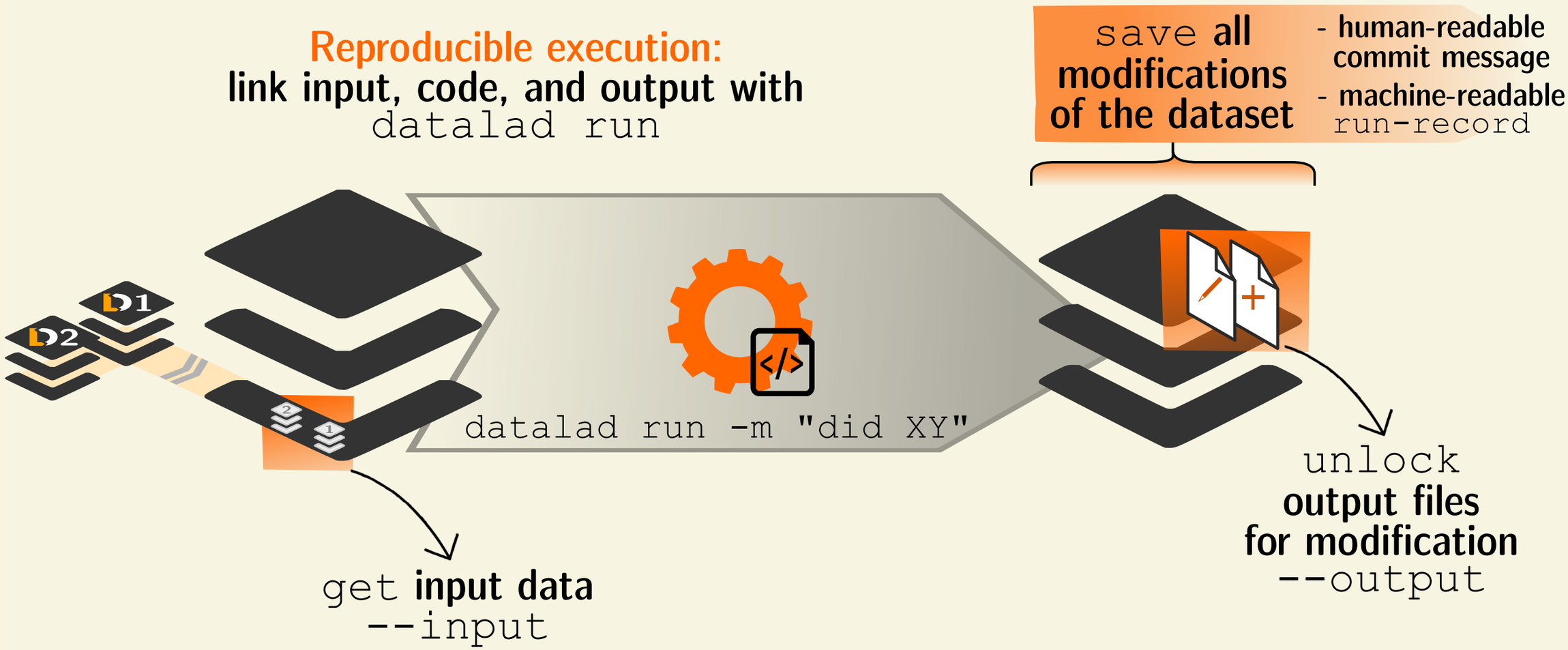


- use `--input` and `--output`

Advice:

ANALYSIS PROVENANCE CAPTURE

Easy provanance capture!

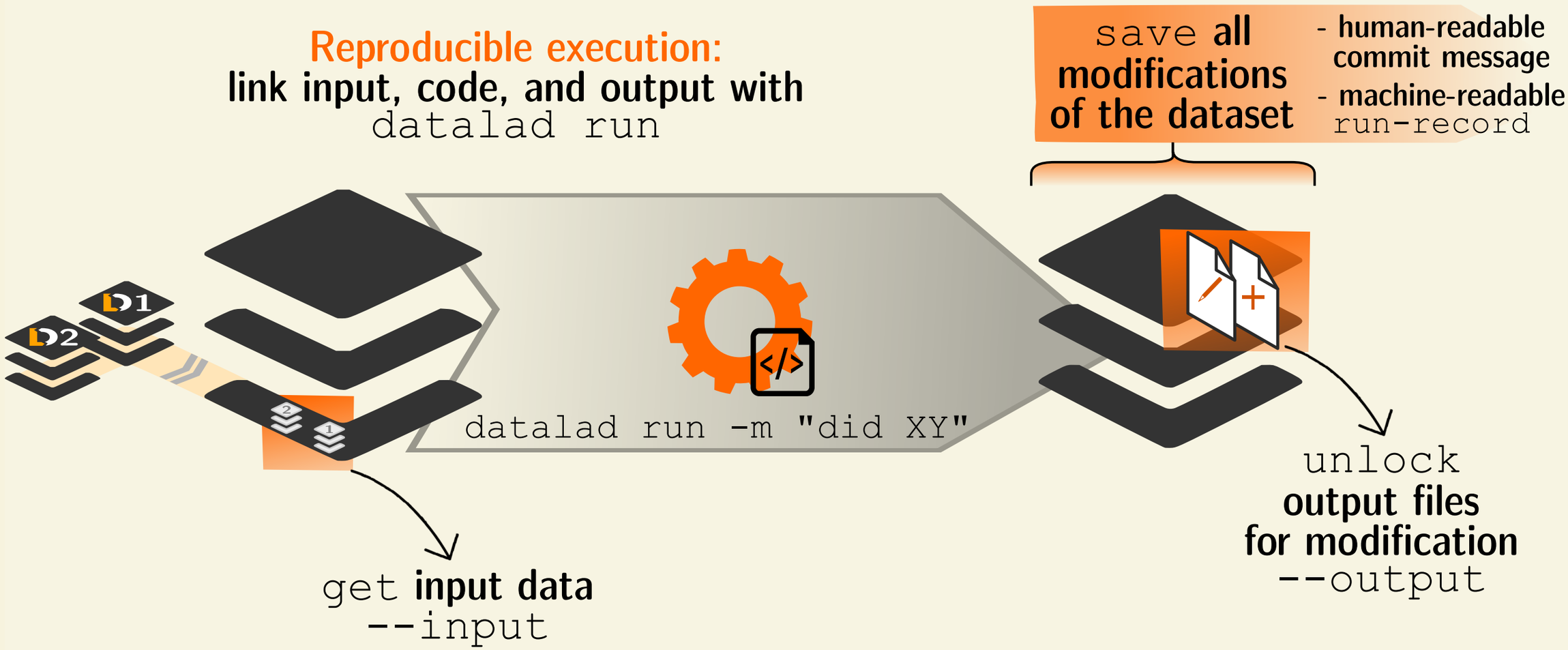


Advice:

- use `--input` and `--output`
- Attach helpful commit messages

ANALYSIS PROVENANCE CAPTURE

Easy provanance capture!



Advice:

- use `--input` and `--output`
- Attach helpful commit messages
- Make sure to have a clean dataset state

SUMMARY - REPRODUCIBLE EXECUTION WITH DATALAD RUN

SUMMARY - REPRODUCIBLE EXECUTION WITH DATALAD RUN

`datalad run` records a commands impact on a dataset.

SUMMARY - REPRODUCIBLE EXECUTION WITH DATALAD RUN

`datalad run` records a commands impact on a dataset.

This usually requires a "clean" dataset status (no unsaved modifications)

SUMMARY - REPRODUCIBLE EXECUTION WITH DATALAD RUN

`datalad run` records a commands impact on a dataset.

This usually requires a "clean" dataset status (no unsaved modifications)

--input to the datalad run command gets retrieved (if necessary) prior to command execution.

SUMMARY - REPRODUCIBLE EXECUTION WITH DATALAD RUN

`datalad run` records a commands impact on a dataset.

This usually requires a "clean" dataset status (no unsaved modifications)

--input to the datalad run command gets retrieved (if necessary) prior to command execution.

This is done with a **datalad get** in the background.

SUMMARY - REPRODUCIBLE EXECUTION WITH DATALAD RUN

`datalad run` records a commands impact on a dataset.

This usually requires a "clean" dataset status (no unsaved modifications)

--input to the datalad run command gets retrieved (if necessary) prior to command execution.

This is done with a **datalad get** in the background.

--output to the datalad run command gets unlocked (if necessary) for modification prior to command execution.

SUMMARY - REPRODUCIBLE EXECUTION WITH DATALAD RUN

`datalad run` records a commands impact on a dataset.

This usually requires a "clean" dataset status (no unsaved modifications)

--input to the datalad run command gets retrieved (if necessary) prior to command execution.

This is done with a **datalad get** in the background.

--output to the datalad run command gets unlocked (if necessary) for modification prior to command execution.

This is done with a **datalad unlock** in the background.

OUTLOOK: COMPUTATIONAL REPRODUCIBILITY

- It may not be enough to record inputs, code, and outputs of an analysis!
- Without sufficient information about **required software (versions)**, analyses may fail to reproduce *or even run*.
- The DataLad extension datalad containers can also capture complete software environments.
- Get a preview soon: chapters on extensions is close to being finished

NOW WHAT I CAN DO WITH THAT?

Reproducible analysis with datalad run

- Record all provenance by wrapping any command in a datalad run
- Link input, output, and code
- Rerun computations with a single command

PRACTICE @HOME

- Wrap any simple shell command (e.g., `cp`) in a datalad run, and (later) also scripts of yours

FURTHER READING

A walk-through on `datalad run`:

- Chapter DataLad, Run! in the handbook.

More on the configurations that determine whether a file is managed by Git or Git-annex:

- Chapter Tuning datasets to your needs in the handbook

How to get help on commands and their options:

- Section How to get help in the handbook

OUTLINE: WHAT COMES NEXT?

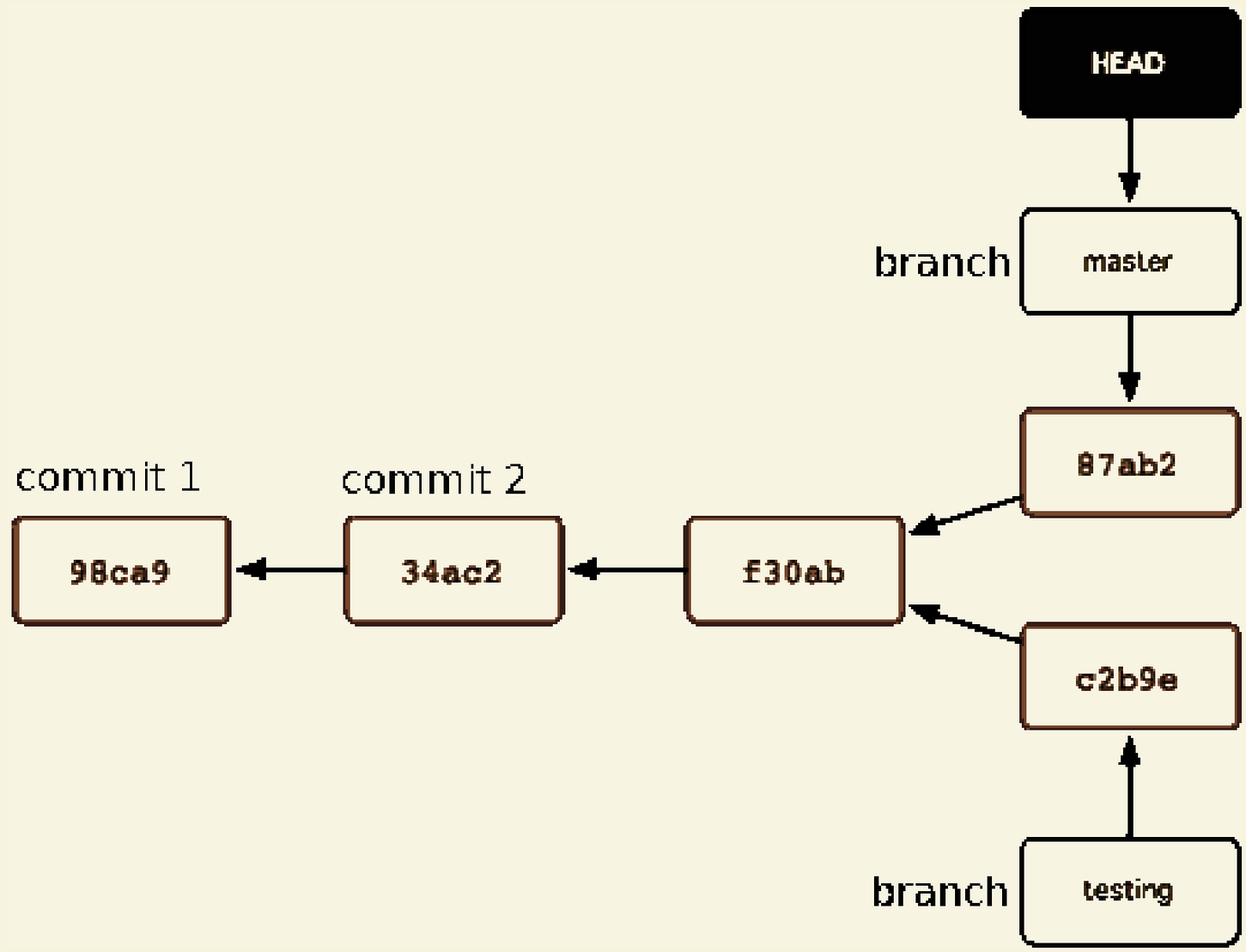
- Symlinks, data integrity, data security (chapter Under the hood: Git-annex in the handbook).
- **Which date is suitable?** > Doodle poll <

OPEN QUESTION SESSION

BACKUP SLIDES FOR ANTICIPATED QUESTIONS

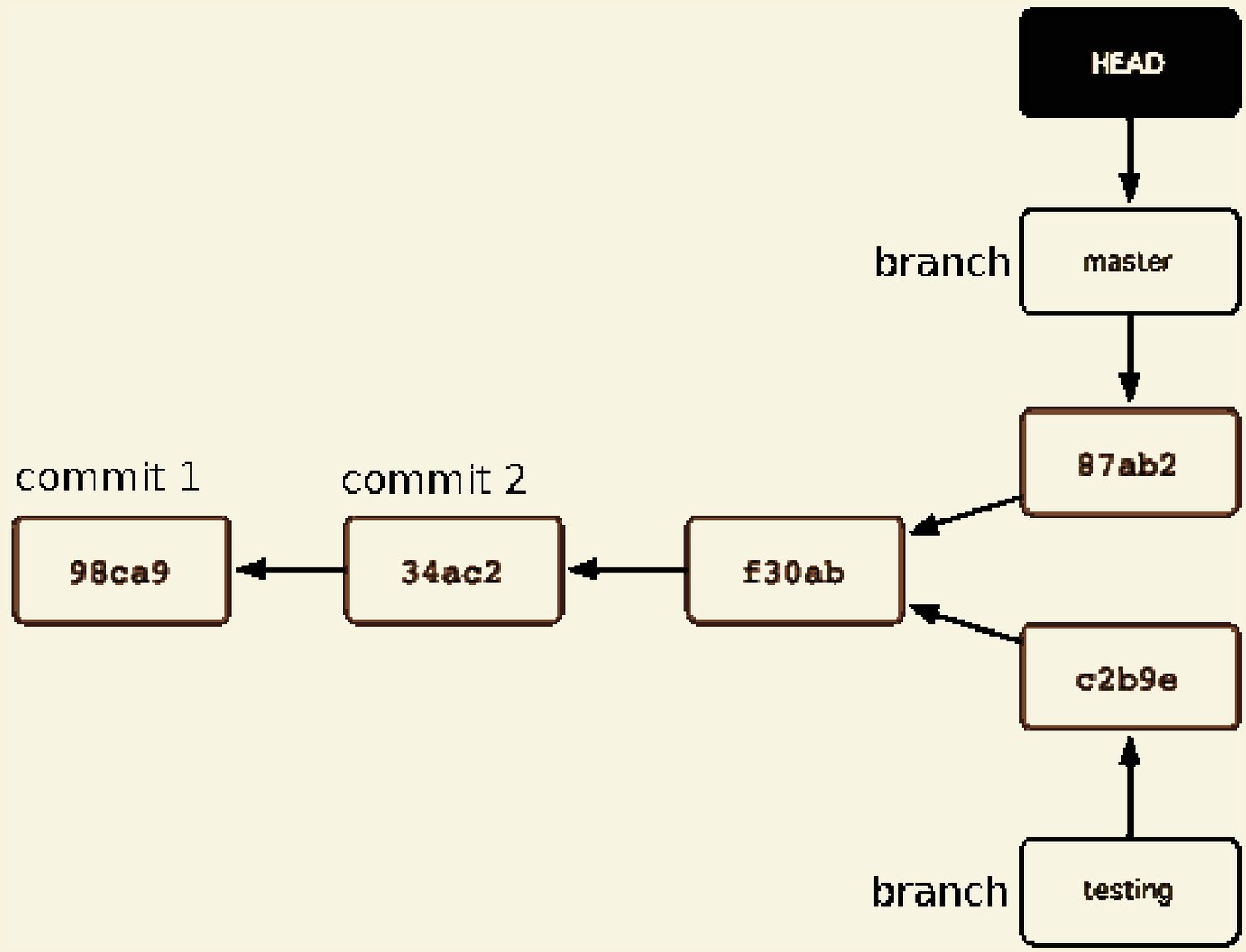
WHAT IS HEAD?

- A git repository is build up as a *tree* of **commits** (history entries).
- A **branch** is a named pointer (reference) to a commit, and allows to isolate developments. The default branch is called `master`.
- HEAD is a pointer to the branch you are on, and thus to the last commit in the given branch.



WHAT IS HEAD?

- A git repository is build up as a *tree* of **commits** (history entries).
- A **branch** is a named pointer (reference) to a commit, and allows to isolate developments. The default branch is called `master`.
- HEAD is a pointer to the branch you are on, and thus to the last commit in the given branch.



If you'd be on branch "testing", which commit would HEAD point to?

HOW DOES A HERE-DOCUMENT WORK?

```
$ cat << EOT > notes.txt
One can create a new dataset with 'datalad create [--description] PATH'.
The dataset is created empty

EOT
```

- Two *delimiting identifiers* (EOT) wrap any amount of text into a stream
- The << characters *redirect* the stream into *standard input* for the `cat` command
- The > character *redirects* the *standard output* of `cat` and writes it into a new file notes.txt

HOW DOES A HERE-DOCUMENT WORK?

```
$ cat << EOT > notes.txt
One can create a new dataset with 'datalad create [--description] PATH'.
The dataset is created empty

EOT
```

- Two *delimiting identifiers* (EOT) wrap any amount of text into a stream
- The << characters *redirect* the stream into *standard input* for the `cat` command
- The > character *redirects* the *standard output* of `cat` and writes it into a new file `notes.txt`

Why is it used?

HOW DOES A HERE-DOCUMENT WORK?

```
$ cat << EOT > notes.txt
One can create a new dataset with 'datalad create [--description] PATH'.
The dataset is created empty

EOT
```

- Two *delimiting identifiers* (EOT) wrap any amount of text into a stream
- The << characters *redirect* the stream into *standard input* for the `cat` command
- The > character *redirects* the *standard output* of `cat` and writes it into a new file `notes.txt`

Why is it used?

- Allows pretty formating (e.g., line breaks)
- Allows writing documents from the terminal