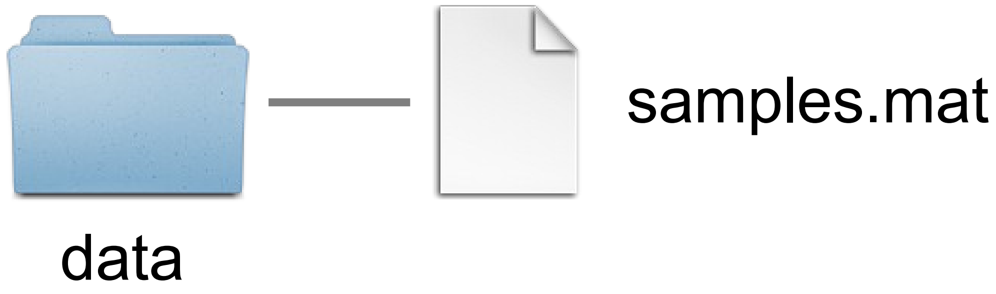
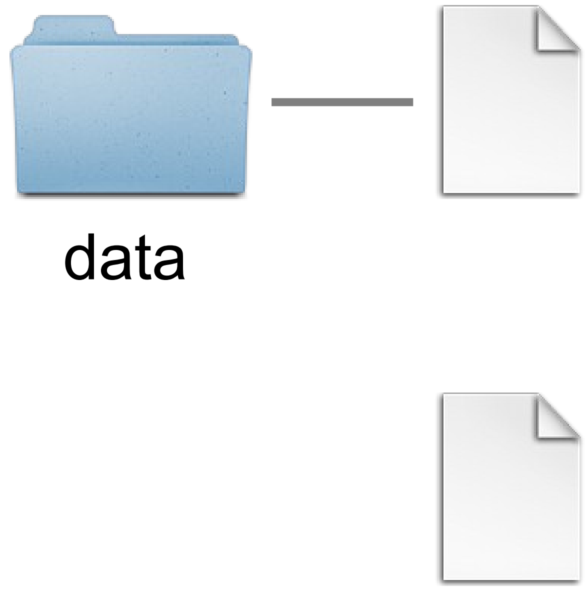
Data Management

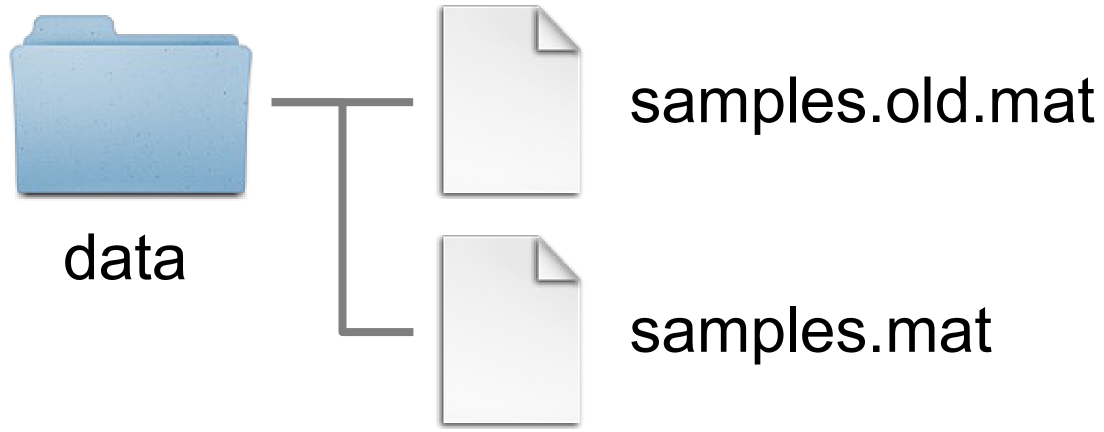
How do we manage multiple versions of data files?



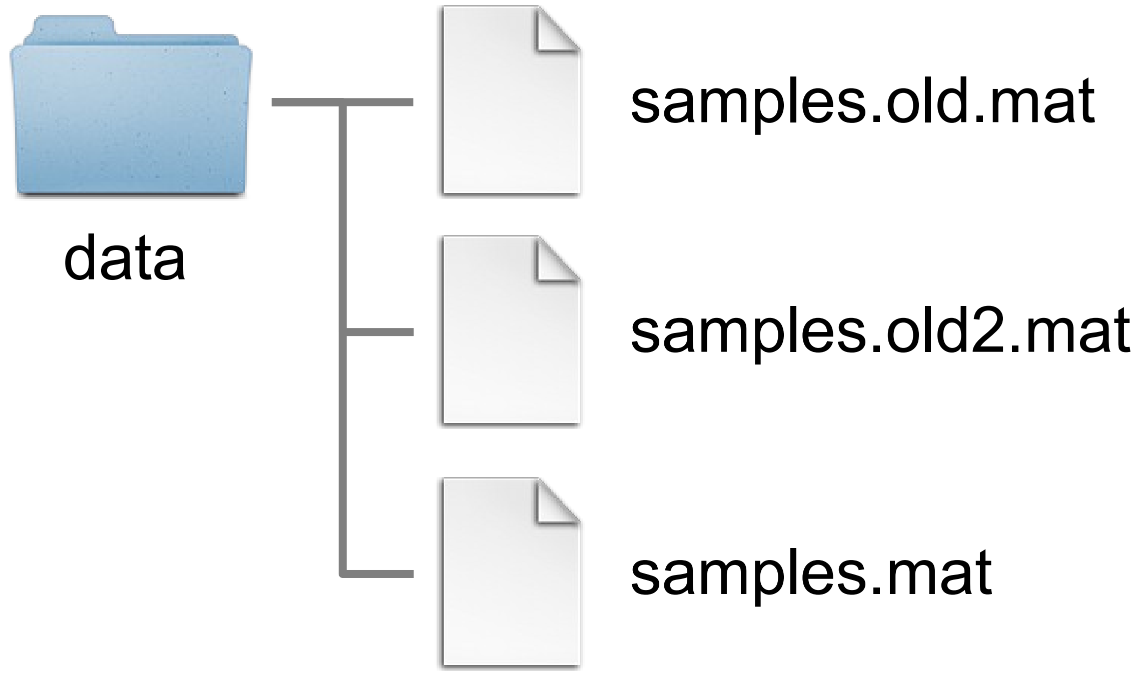
So far, so good…



Now what?



I guess this is alright?



Okay…

One problem:

generate_results.sh

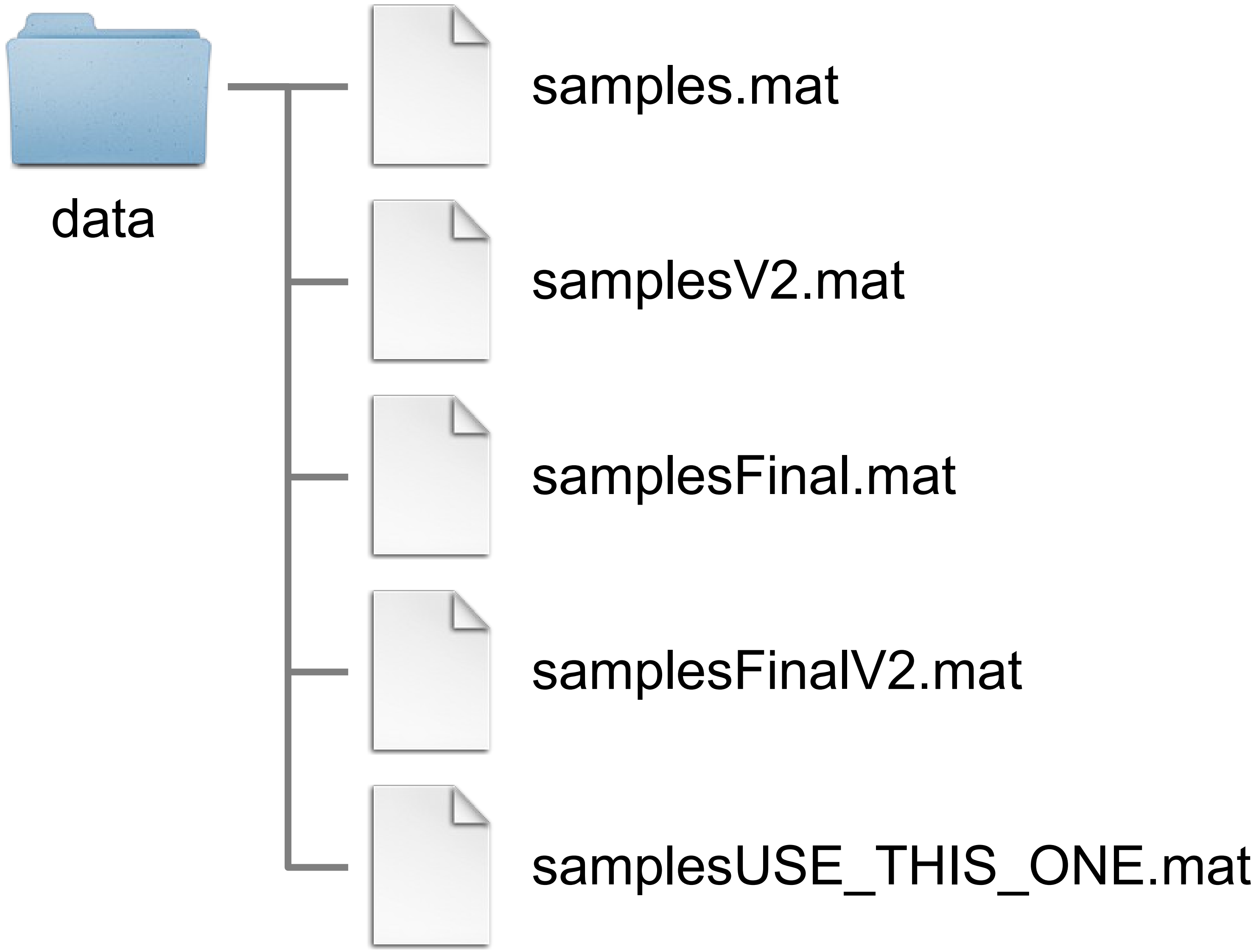
```
#!/usr/bin/env bash

cat data/samples.mat \
  | tr “^M” “\n” \
  | sort -k  2,2nr \
  | ...
  > results/samples.txt
```

Which version of samples.mat was used to generate our results?

In general, don’t move or rename files unless absolutely necessary.

-makes it harder to reproduce results
-makes it harder to find the data later
-breaks scripts
-breaks symbolic links (`ln -s`)

But adding new filenames without structure isn't necessarily any better…



Which one is the most recent?

You could…

1)look at the data and guess

2)check the last-modified date

And don't ever move or copy the data!

sounds like a job for…

sounds like a job for…

version control!

sounds like a job for…

version control!

…or is it?

Why shouldn’t we just use Subversion?

Subversion is good for:
-simultaneous modification
-archive
-merge
-branch
-diff

Why shouldn’t we just use Subversion?

Subversion is good for:
-simultaneous modification
-archive
-merge
-branch
-diff

what makes sense for data?

Why shouldn't we just use Subversion?

Subversion is good for:

-simultaneous modification
-archive
-merge
-branch
-diff

what makes sense for data?

What we need…

a clear, stable, accessible way to archive data

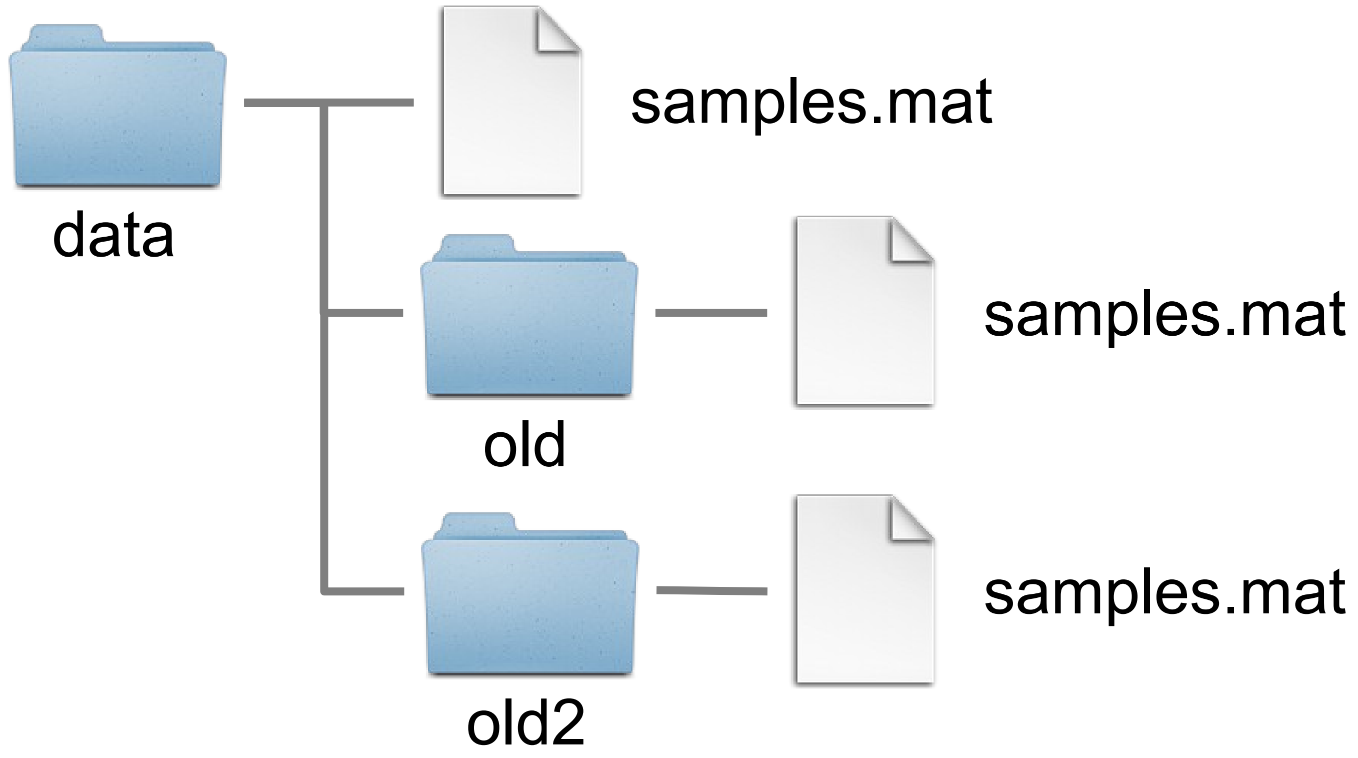
Performance reasons for not using version control on data

- We can no longer remove or compress old and obsolete data files.

- Checking out the repository means copying *all* your data files.

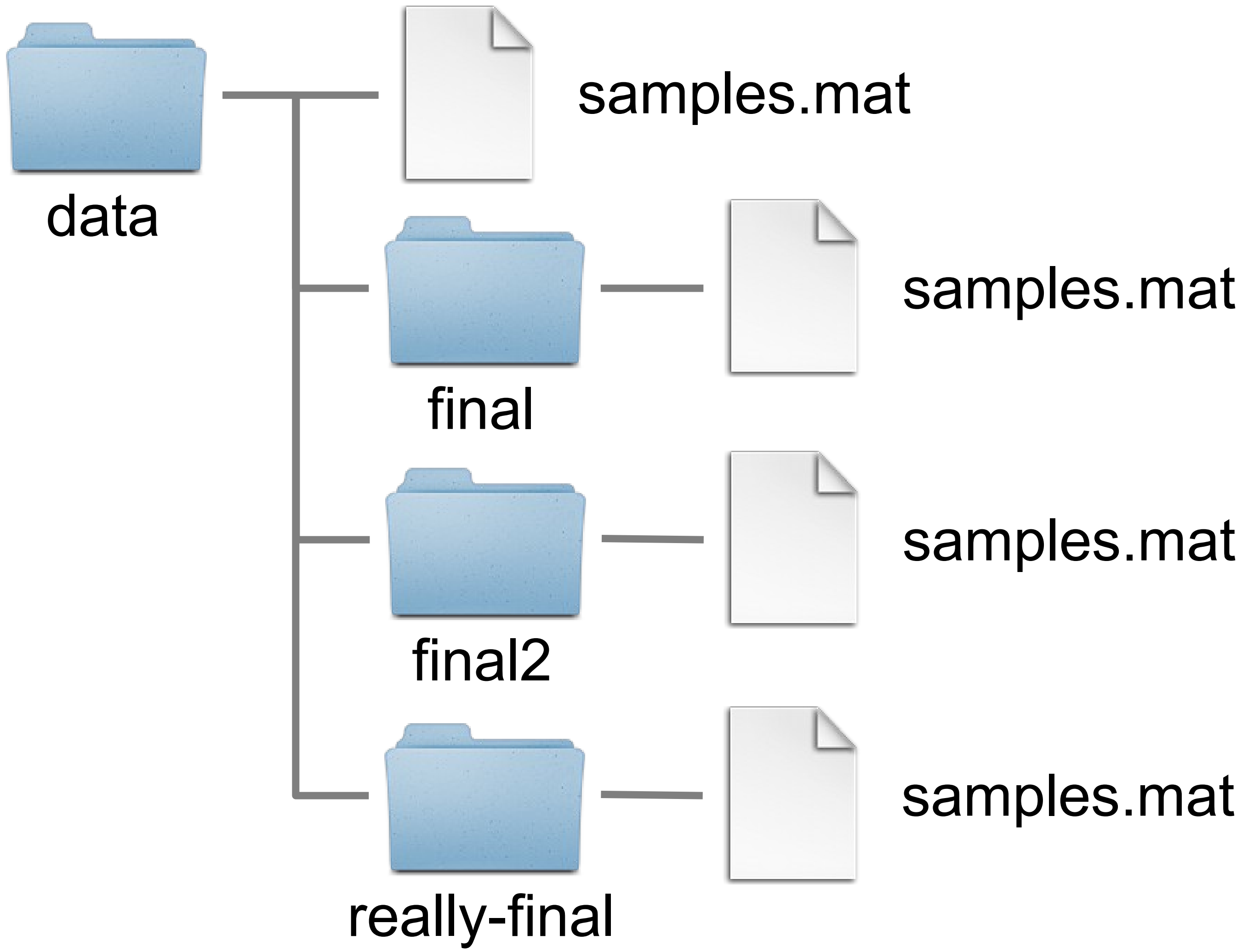
- Many version control tools store text-based differences between versions of a file. For binary files, they essentially store a new copy of the file for each version.

Common approach:



Archive old data into subdirectories:
-maintains file names
-keeps all data files in same “version” together
-still moves files, so scripts and history now refer to different data

Another common approach:



Two big problems still:

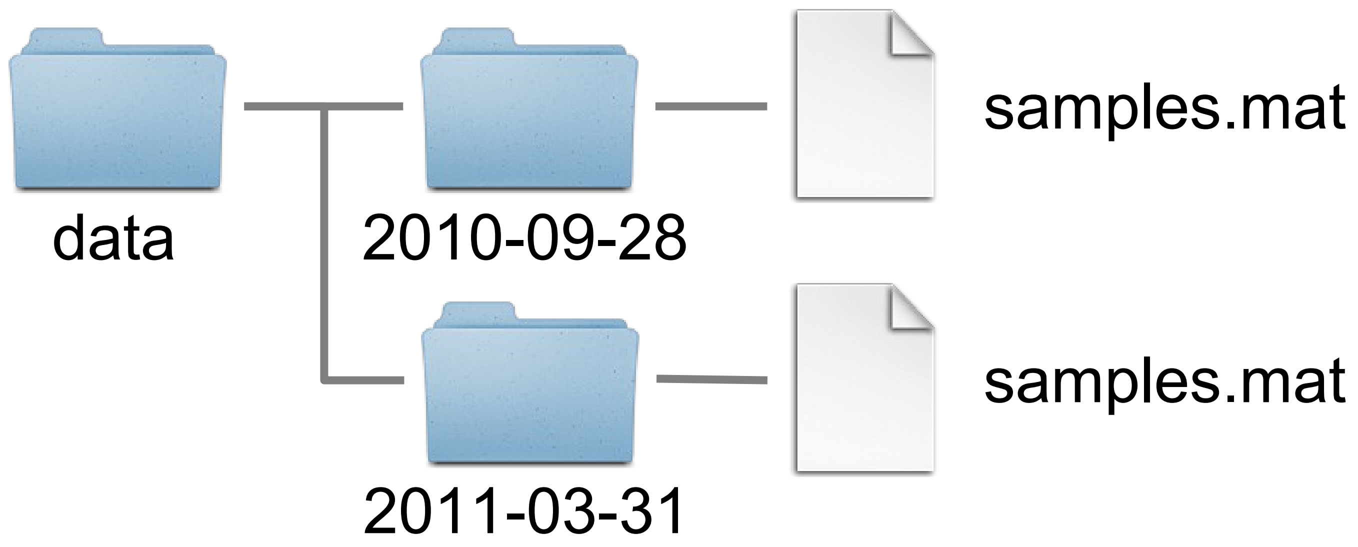
1) What order do they go in? The naming can easily become ambiguous.

2) Hard to distinguish “top-level” data directories from archived or revised data directories.

Is data/saved/ an old version of all the data or a current version of “saved” data?

Both are easily solved with a little **thinking ahead**

From the start:



"yyyy-mm-dd" is a nice date format since:
-is easy to read (easier than yyyymmdd)
-the alphabetic ordering is also chronological

We’re still missing something **big**

- We give a collaborator access to the data folder
- A new student joins the project
- It’s three years later and we’ve forgotten the details of the project

We’re still missing something **big**

- We give a collaborator access to the data folder
- A new student joins the project
- It’s three years later and we’ve forgotten the details of the project

We need **context**. We need **metadata**.

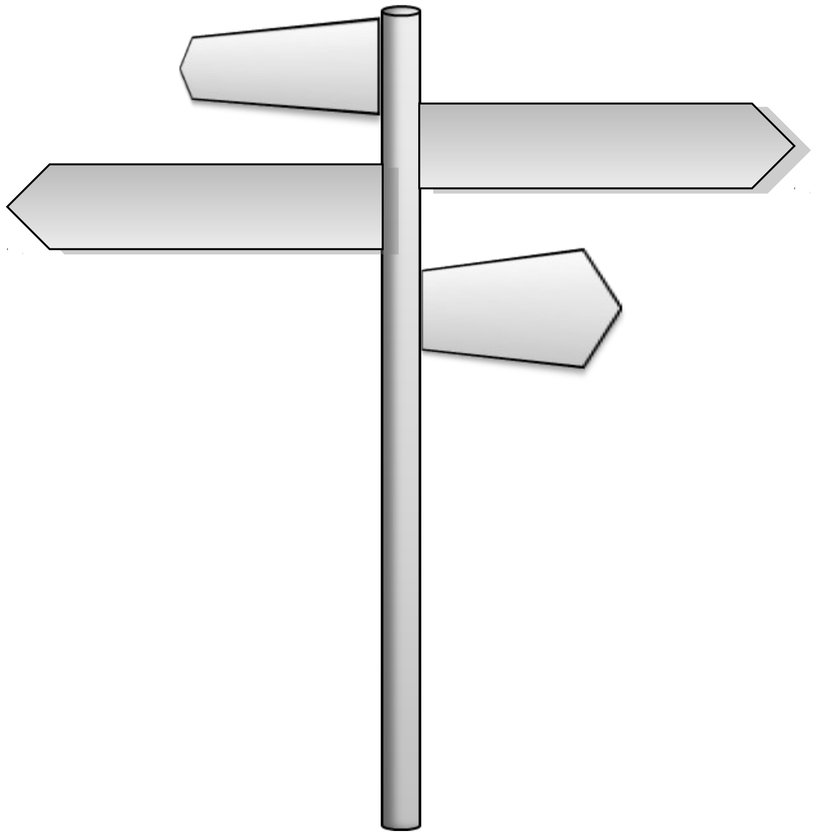
Metadata

- who is the data from?
- when was it generated?
- what were the experiment conditions?

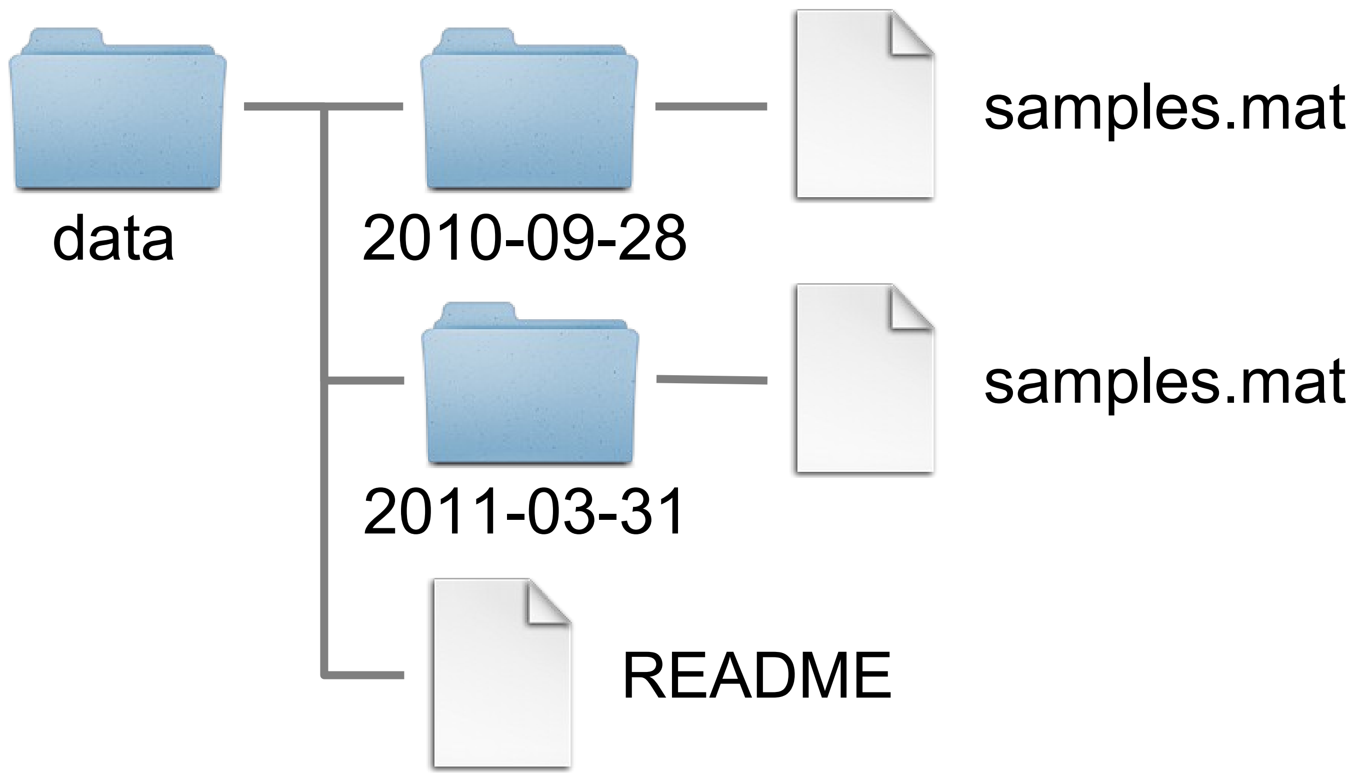
Header inside the file? (see Provenance essay)
-Doesn’t work well with binary files

Separate metadata file for each data file?
-Can quickly get out of sync or out of hand

How I view metadata…

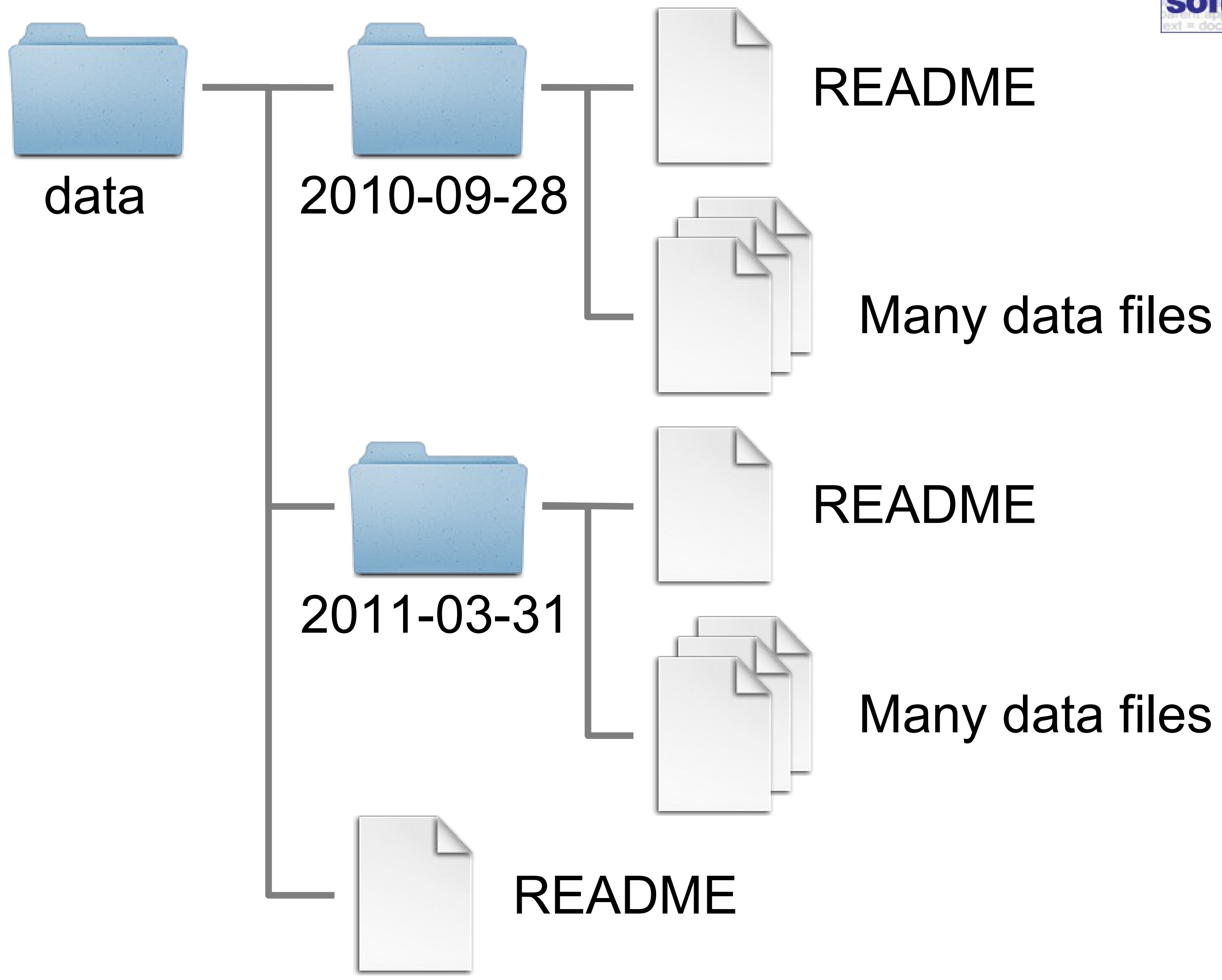


A list of attributes that I might want to search by later

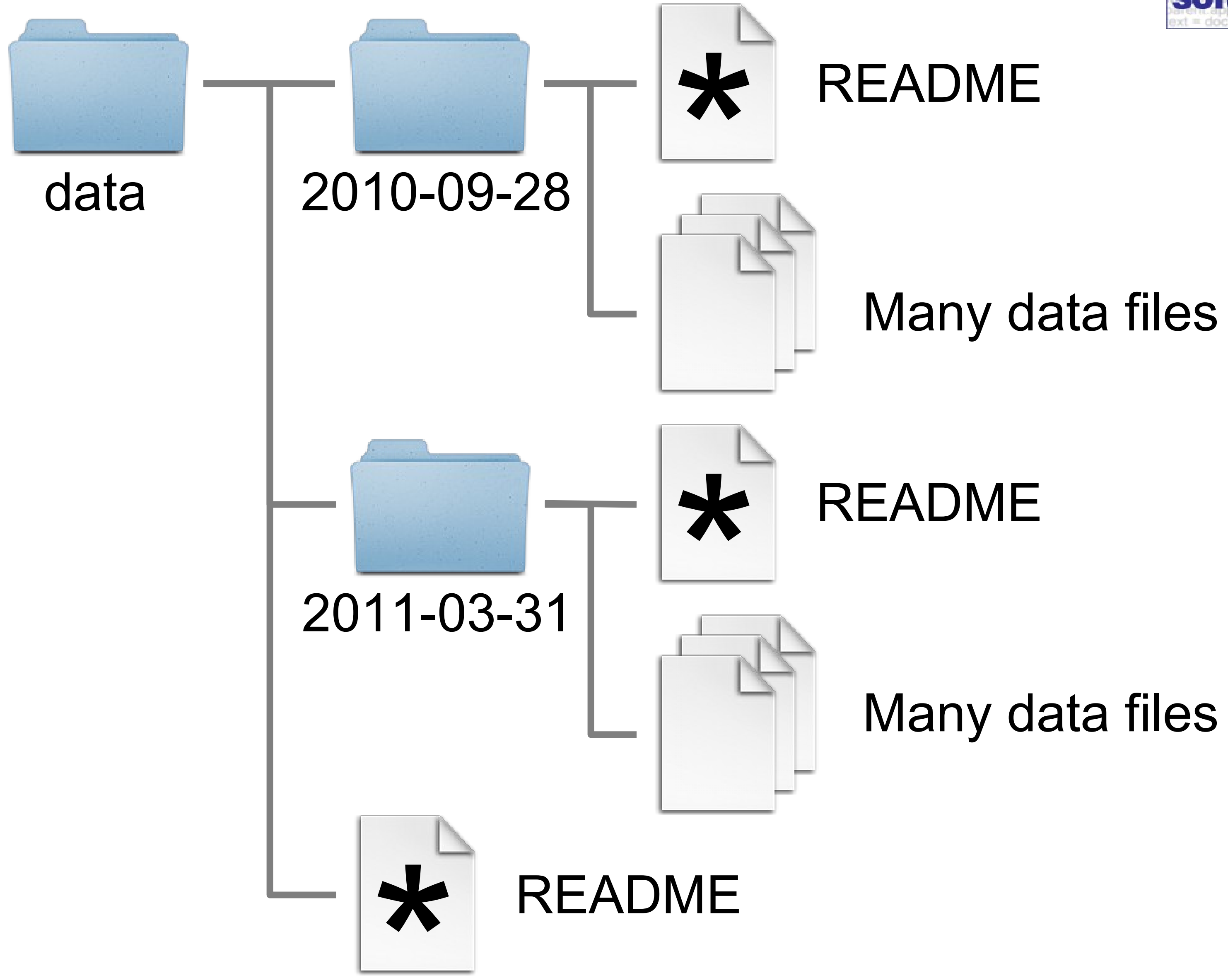


README

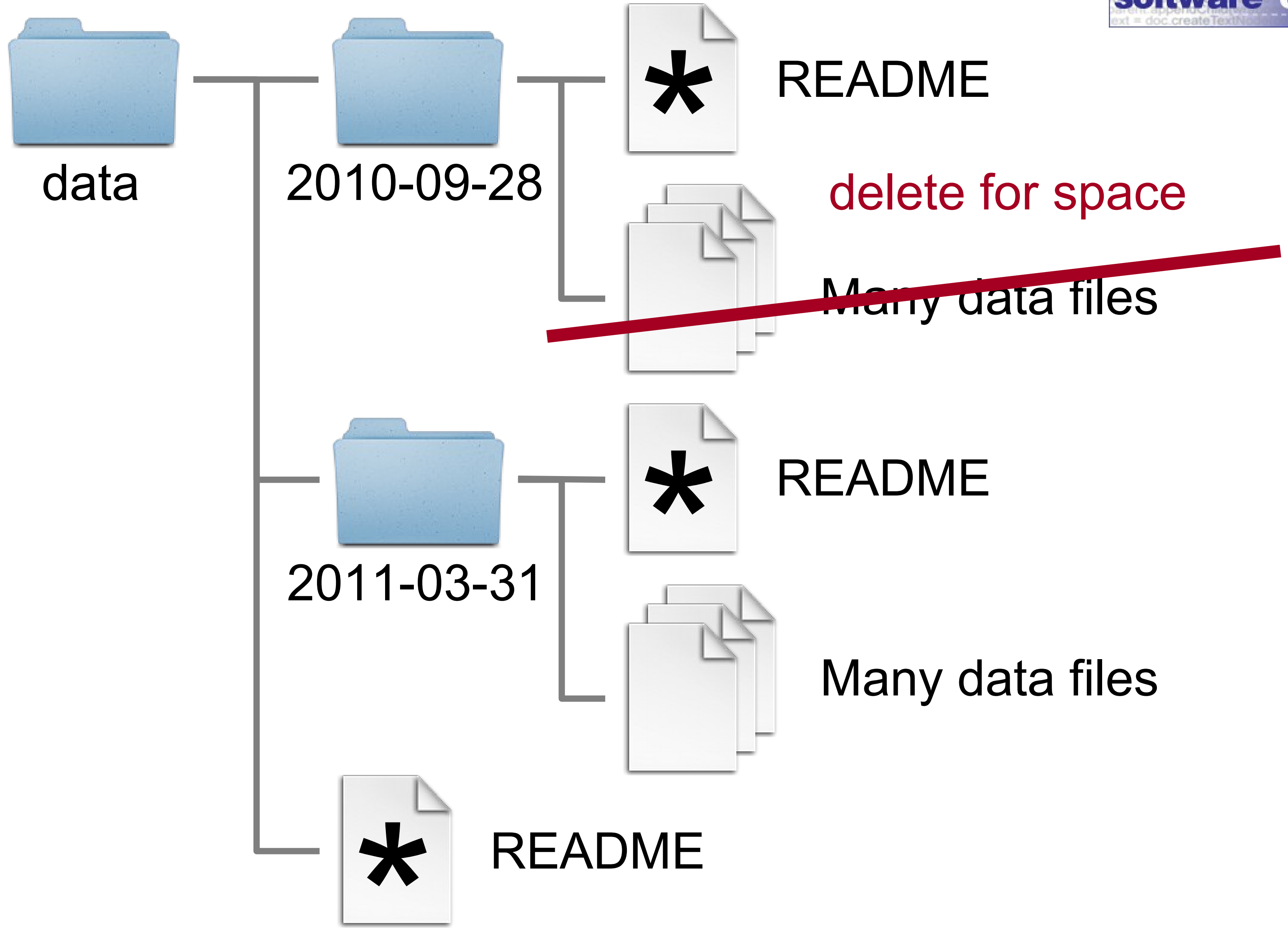
```
2010-09-28 - batch1 - 25*C - 5 hrs
        Initial QC: samples 1328, 1422 poor quality
2011-03-31 - batch1 - 28*C - 10 hrs
        Initial QC: okay, but 1 min power failure @ t=2.5
```



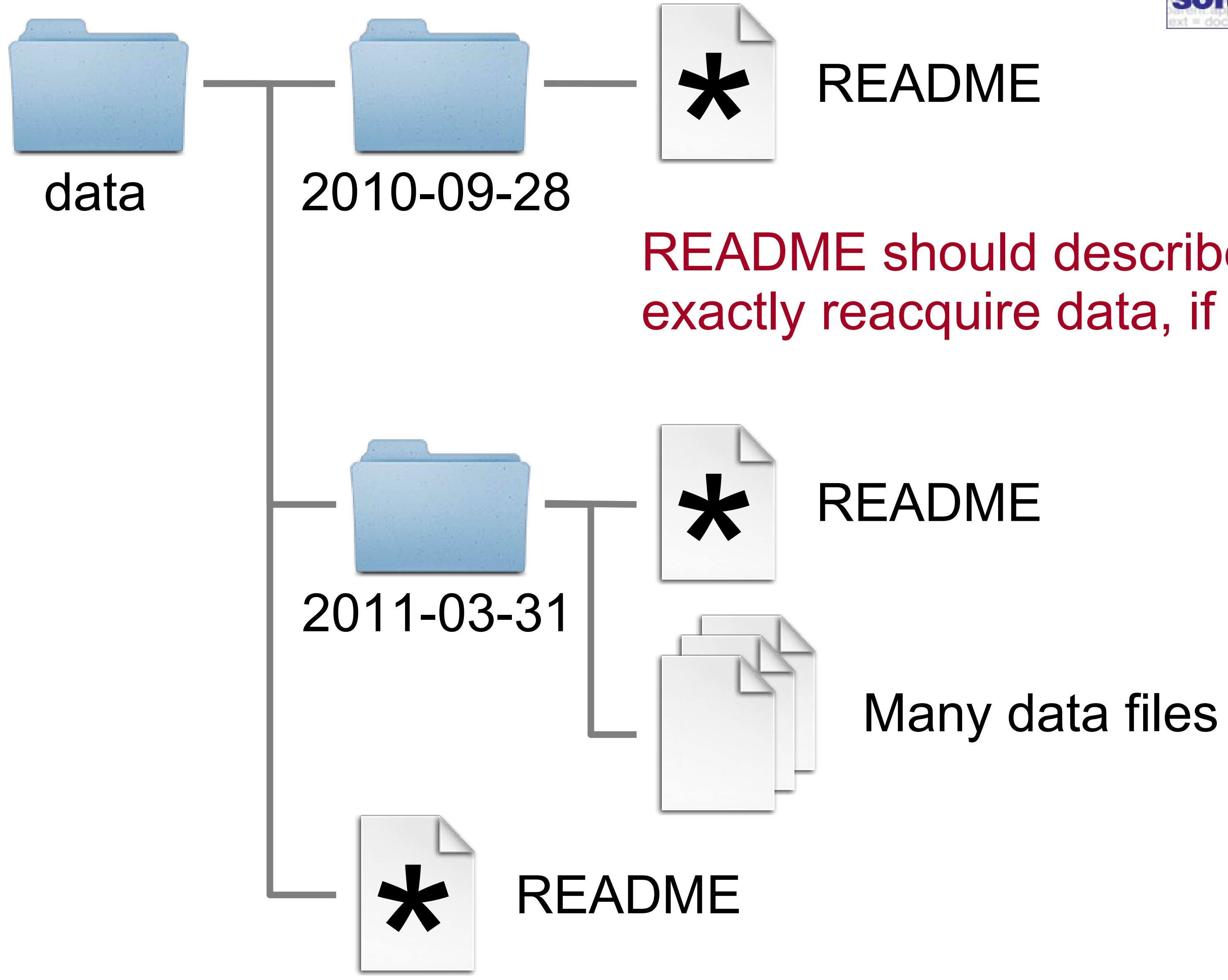
Additional READMEs can be necessary when you have many data files per directory



* Under version control



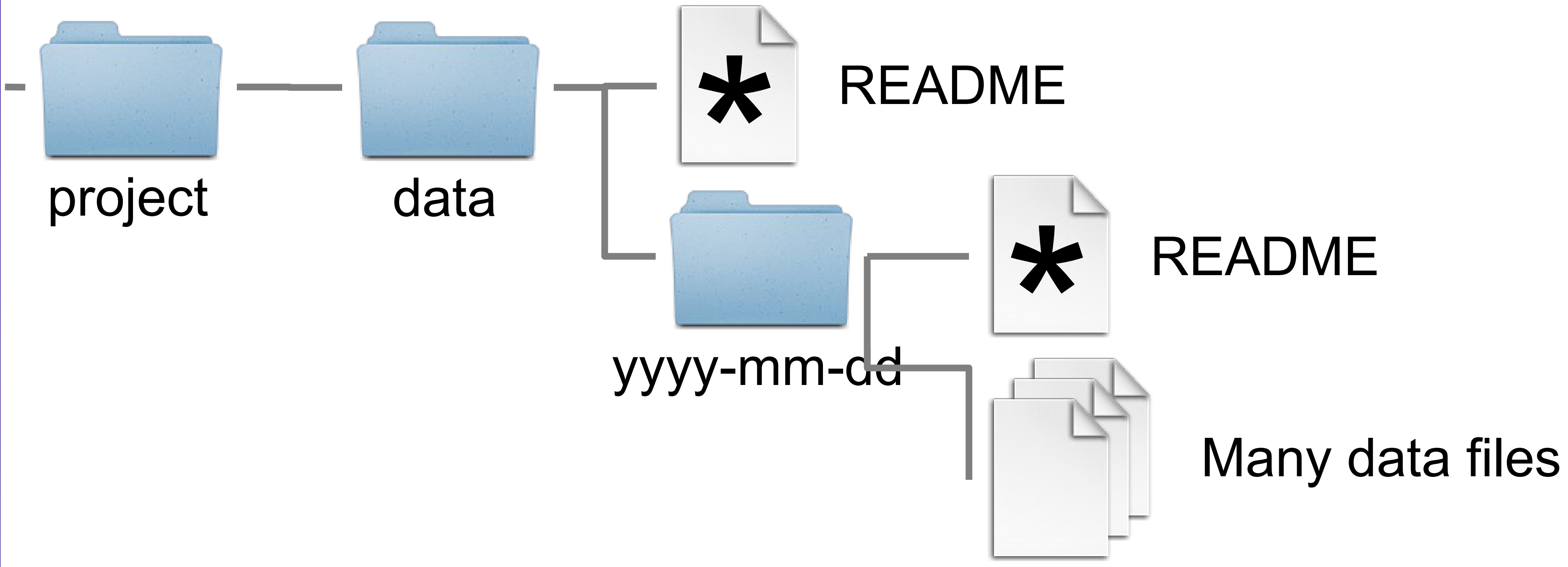
* Under version control



README should describe how to exactly reacquire data, if possible.

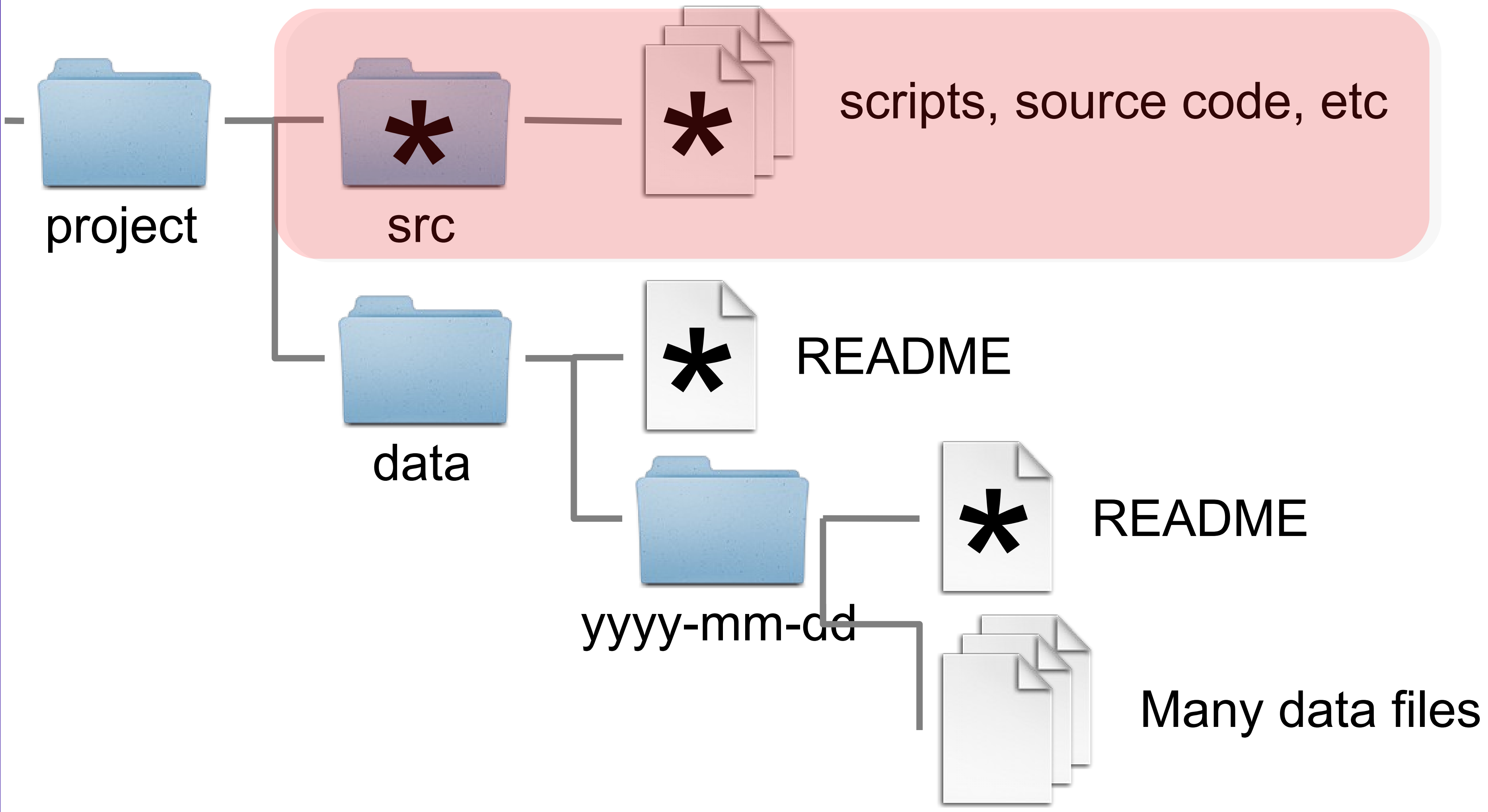
* Under version control

The big picture



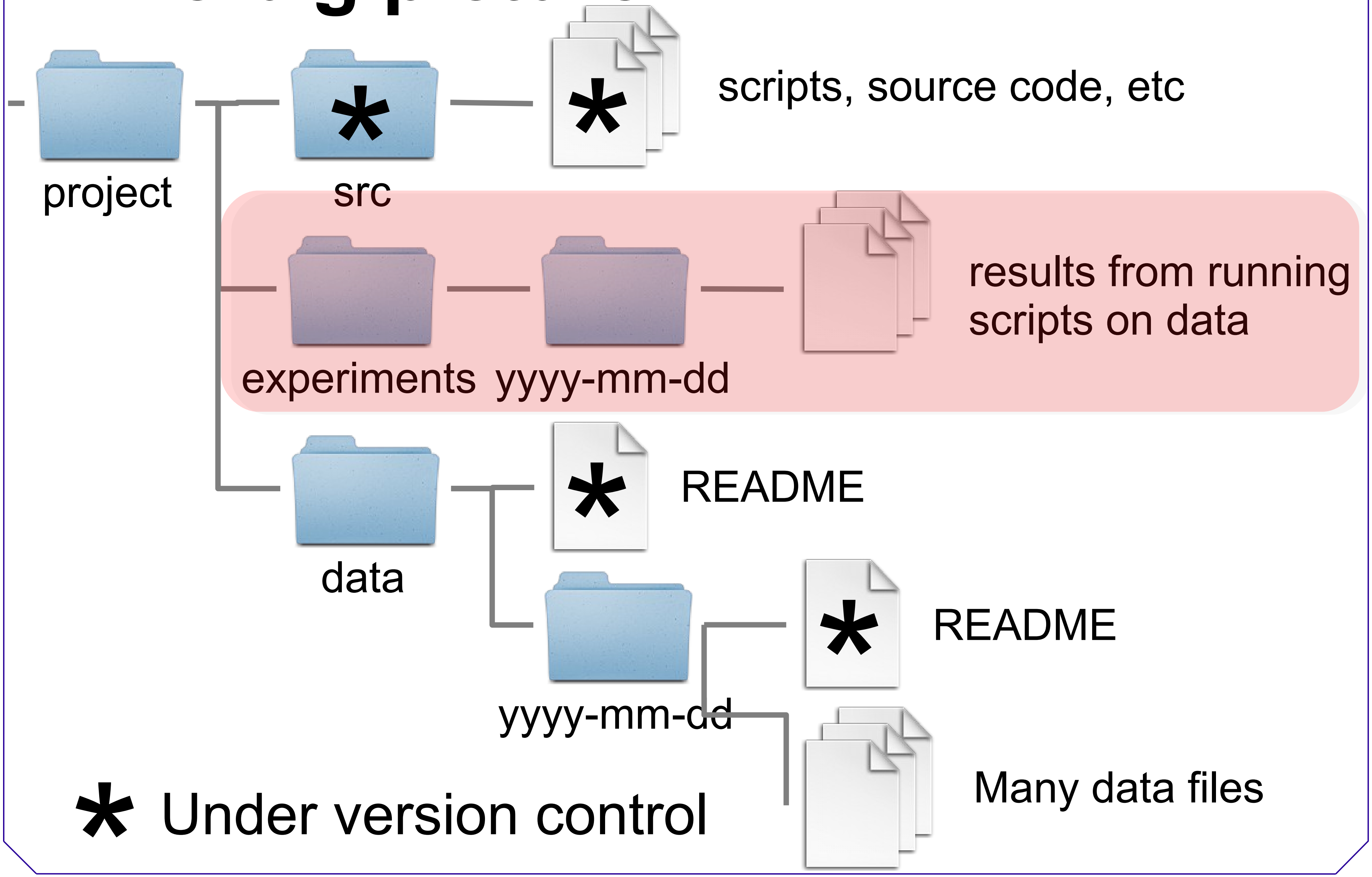
* Under version control

The big picture



* Under version control

The big picture



* Under version control

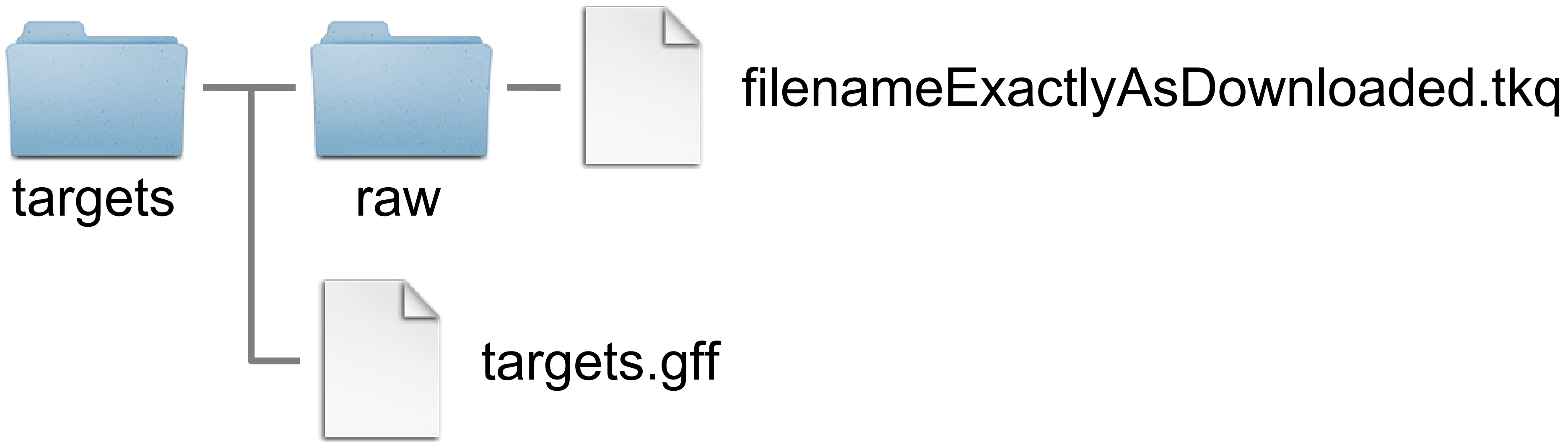
Why all the fuss?

Sensible archiving:

-make it clear and obvious for someone else

-“someone else” will likely be **you**, several months or years from now

Why all the fuss?



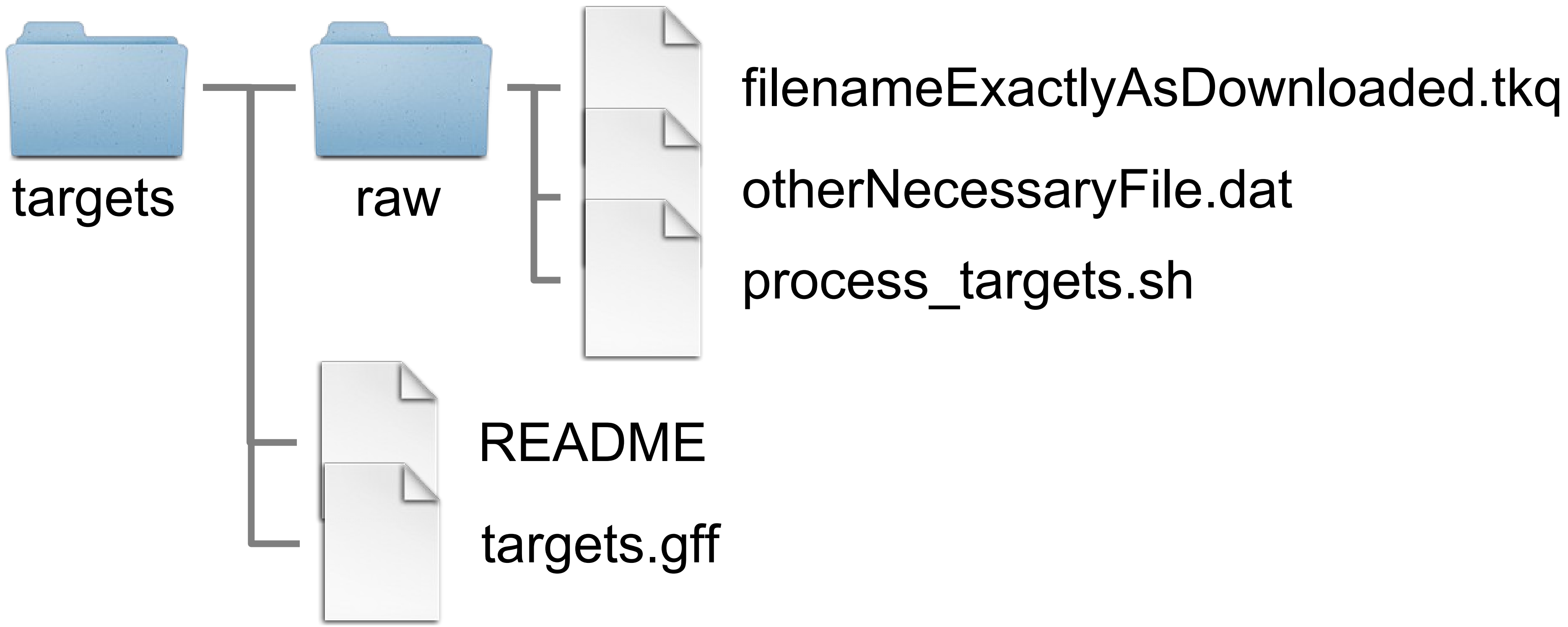
Example:

How exactly did I create targets.gff?

This would be very difficult without additional information.

Why all the fuss?

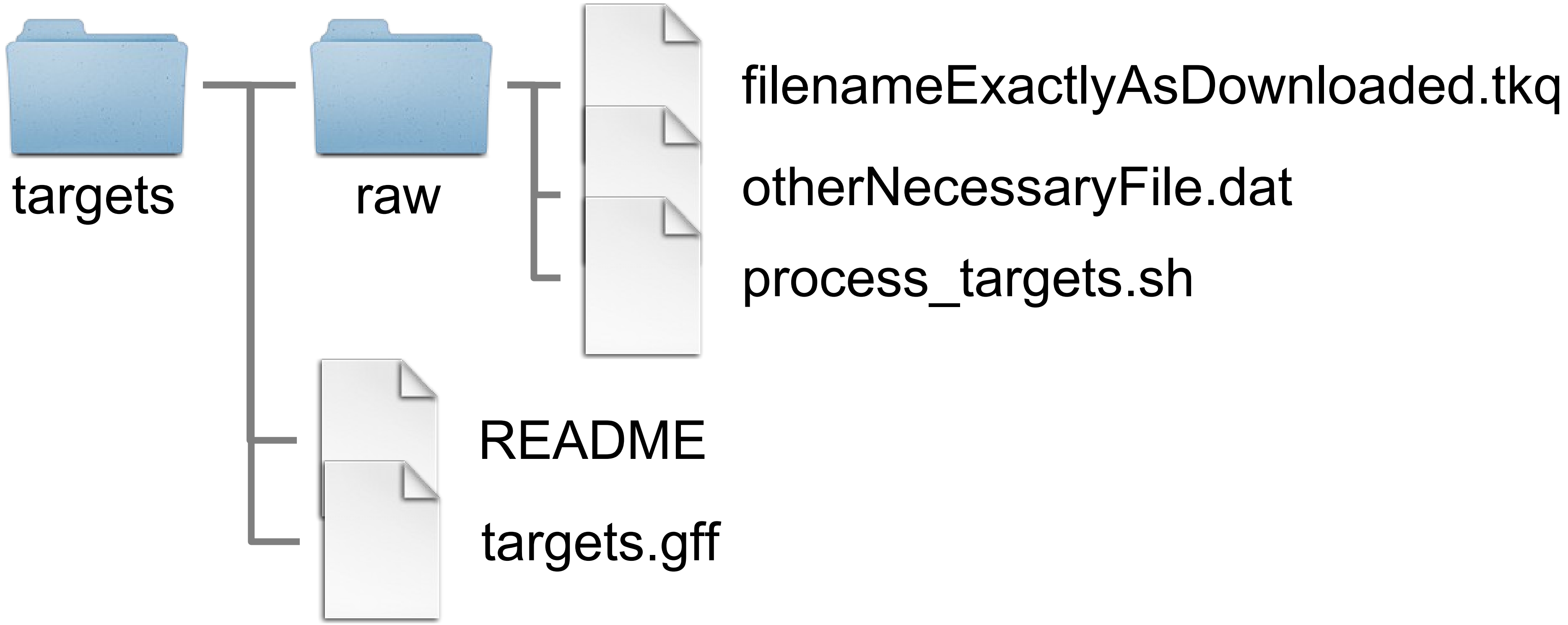
The directory structure:



README - when and where from files were downloaded
process_targets.sh - exact commands used to generate targets.gff from downloaded files

Why all the fuss?

The directory structure:



This made it both 1) possible and 2) easy to know exactly where the data came from and how it was processed.

The right layout for the right project

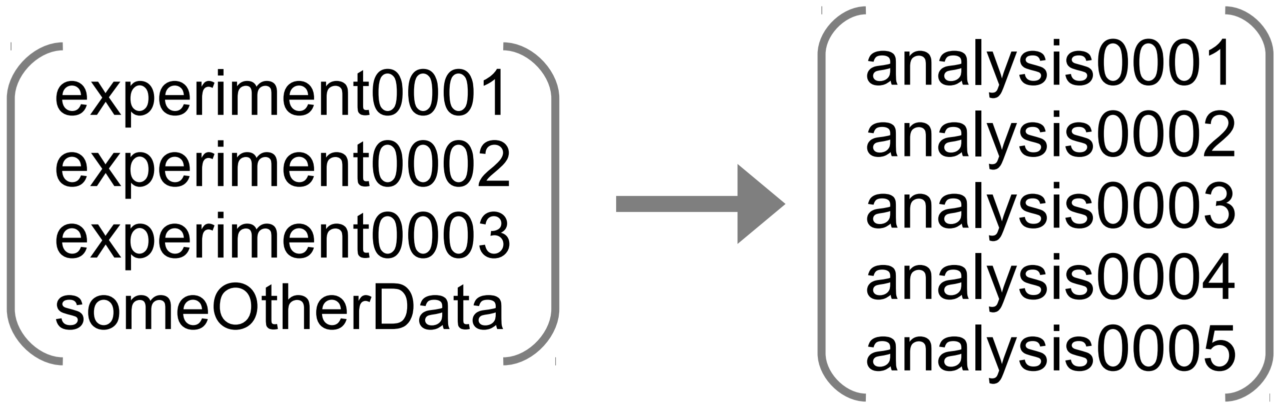
- No hierarchy is perfect for all projects.

- Thinking hard at the **beginning** can save pain and suffering later on

- For example…

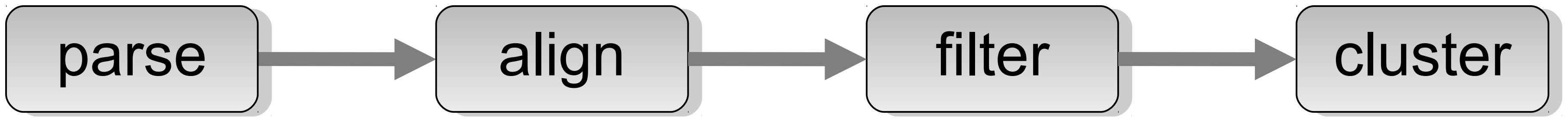
The right layout for the right project

What we've talked about so far is great for projects with a strong separation between data and analysis:



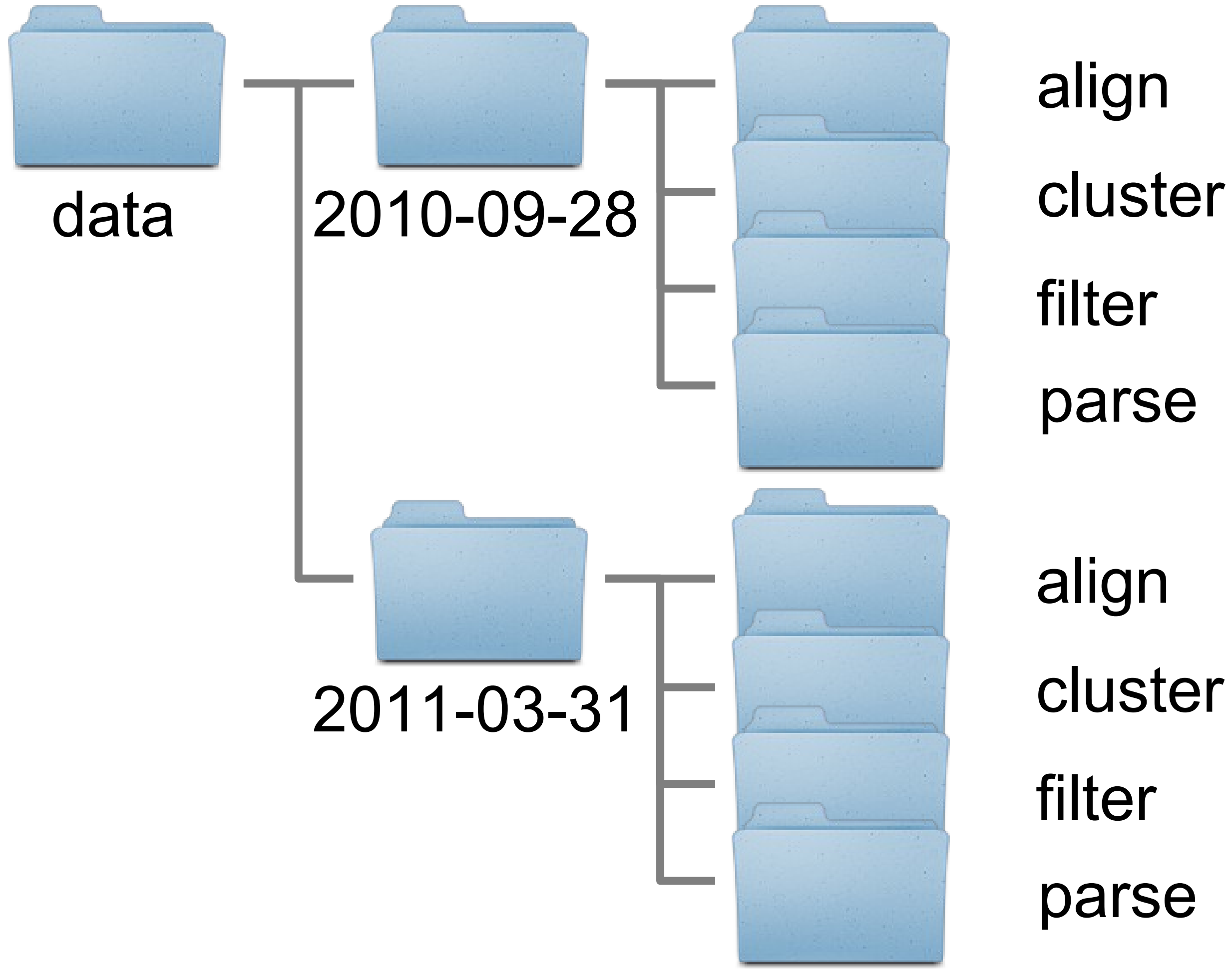
The right layout for the right project

But it doesn't work as well for pipelines:



The right layout for the right project

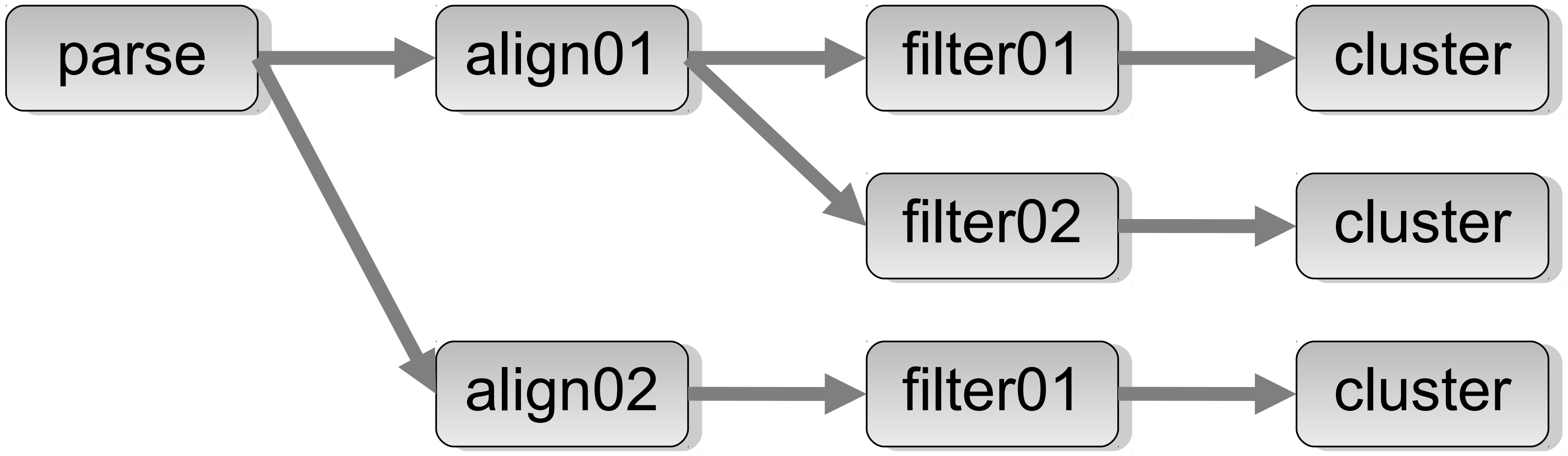
One representation I have seen is:



The right layout for the right project

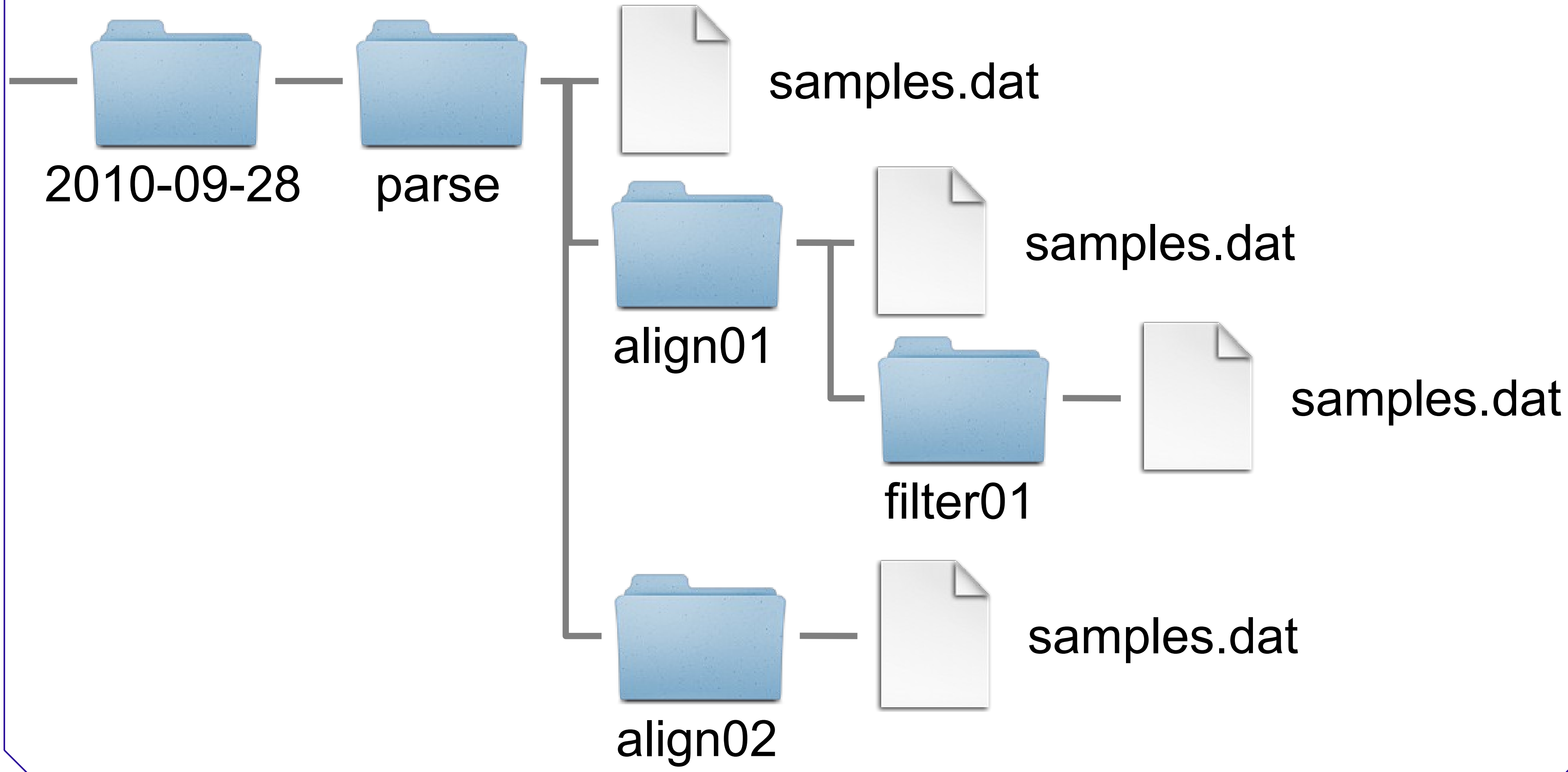
But:

1)I doesn't make the pipeline ordering clear
2)It doesn't scale well with alternatives:



The right layout for the right project

A better approach captures the dependency structure of the pipeline:



Summary

- Think hard at the beginning
- Version control metadata, not data files
- An intelligent structure not only makes your data easier to archive, track, and manage, but also reduces the chance that the paths in the pipeline get crossed and the data out the end isn't what you think it is.

created by

Orion Buske

May 2011