Tools for Reproducible Research

Organizing projects; exploratory data analysis

Karl Broman

Biostatistics & Medical Informatics, UW–Madison

`biostat.wisc.edu/~kbroman`
`github.com/kbroman`
`@kwbroman`

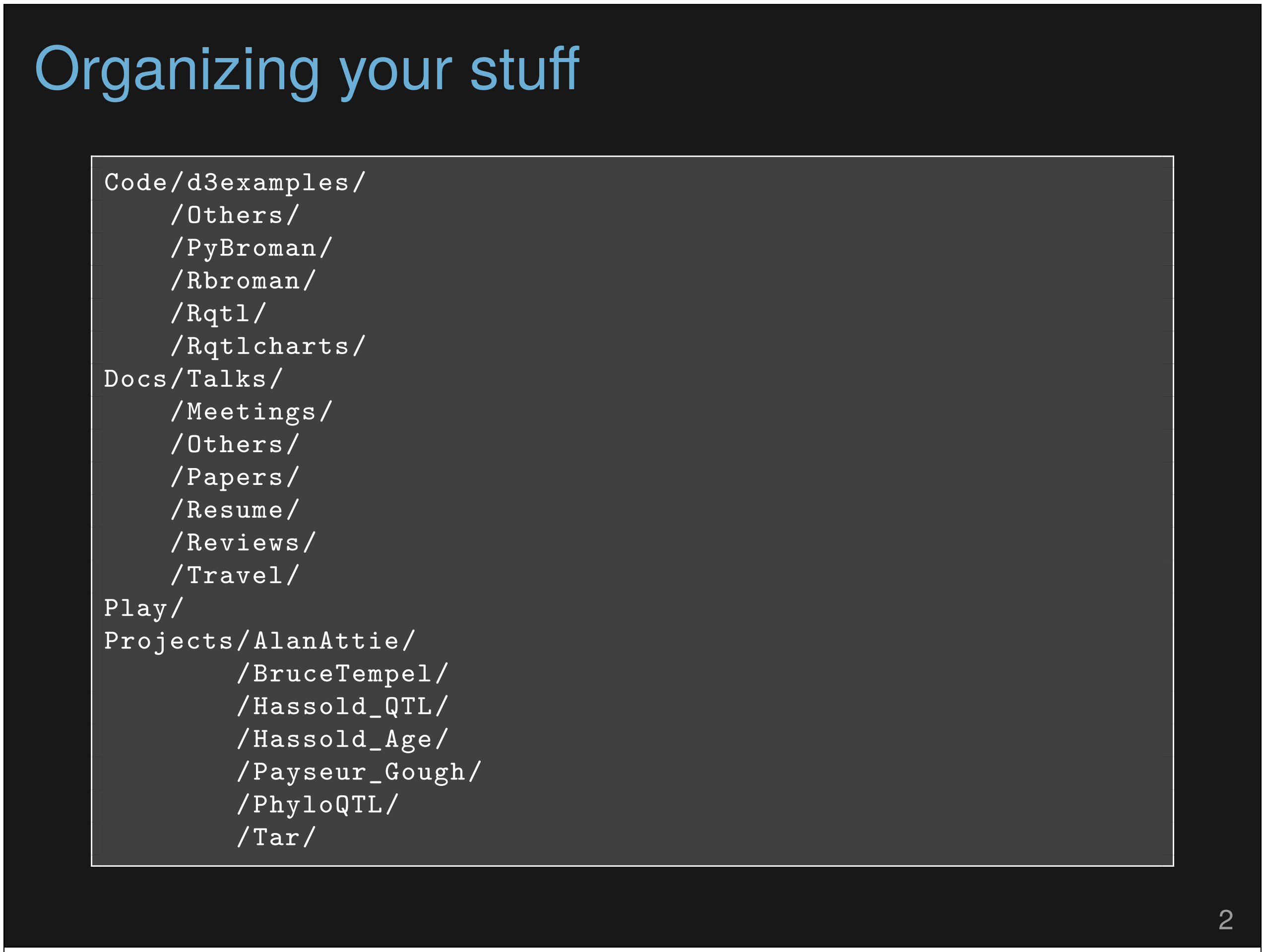
Course web: `bit.ly/tools4rr`

I'm trying to cover two things here: how to organize data analysis projects, so in the end the results will be reproducible and clear, and how to capture the results of exploratory data analysis.

The hardest part, regarding organizing projects, concerns how to coordinate collaborative projects: to keep data, code, and results synchronized among collaborators.

Regarding exploratory data analysis, we want to capture the whole process: what you're trying to do, what you're thinking about, what you're seeing, and what you're concluding and why. And we want to do so without getting in the way of the creative process.

I'll sketch what I try to do, and the difficulties I've had. But I don't have all of the answers.

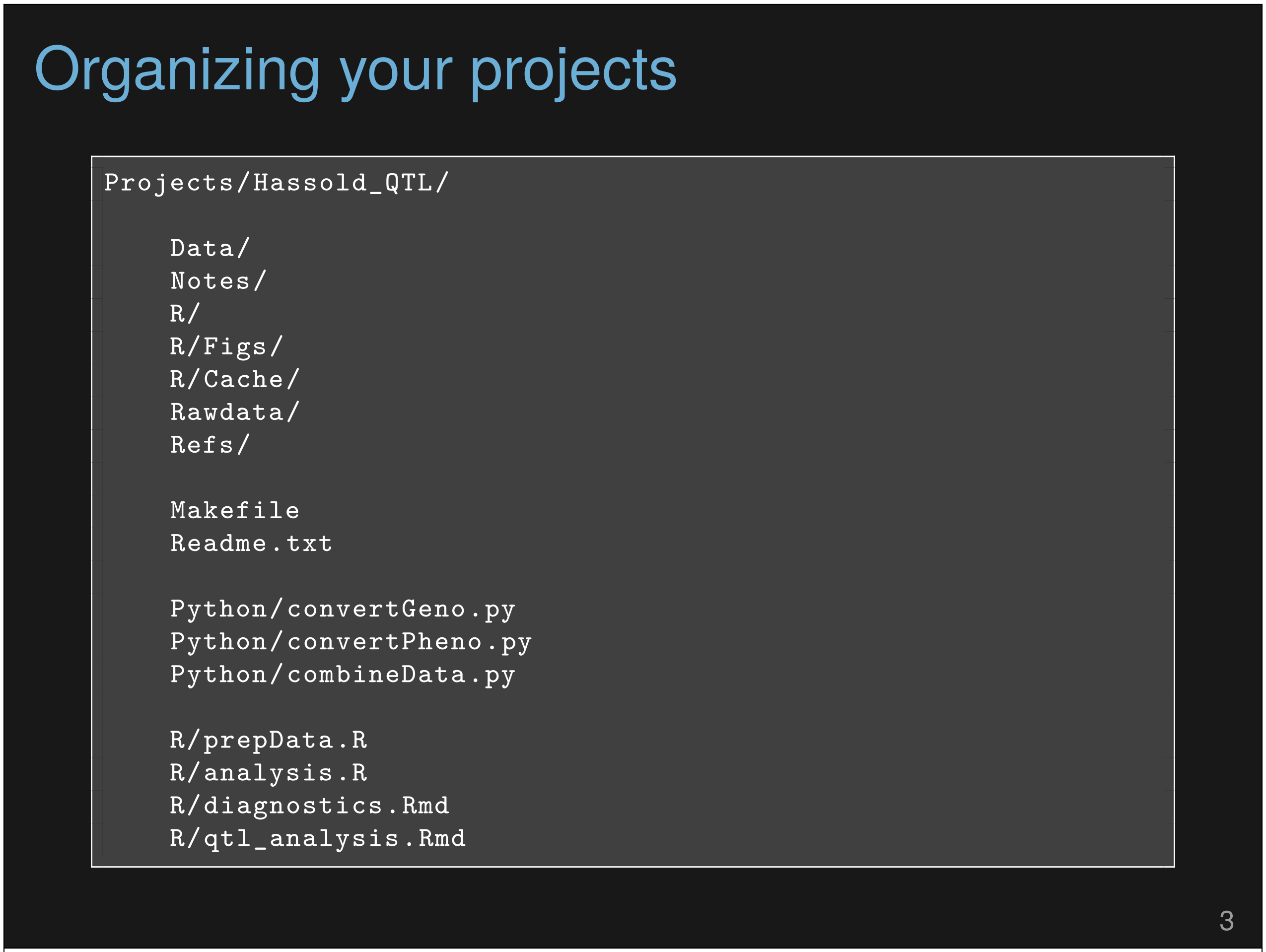


Organizing your stuff

```
Code/d3examples/
    /Others/
    /PyBroman/
    /Rbroman/
    /Rqtl/
    /Rqtlcharts/
Docs/Talks/
    /Meetings/
    /Others/
    /Papers/
    /Resume/
    /Reviews/
    /Travel/
Play/
Projects/AlanAttie/
        /BruceTempel/
        /Hassold_QTL/
        /Hassold_Age/
        /Payseur_Gough/
        /PhyloQTL/
        /Tar/
```

This is basically how I organize my hard drive. You want it to be clear where things are. You shouldn't be searching for stuff.

In my `Projects/` directory, I have a `Tar/` directory with `tar.gz` files of older projects; the same is true for other directories, like `Docs/Papers/` and `Docs/Talks/`.



Organizing your projects

```
Projects/Hassold_QTL/

    Data/
    Notes/
    R/
    R/Figs/
    R/Cache/
    Rawdata/
    Refs/

    Makefile
    Readme.txt

    Python/convertGeno.py
    Python/convertPheno.py
    Python/combineData.py

    R/prepData.R
    R/analysis.R
    R/diagnostics.Rmd
    R/qtl_analysis.Rmd
```

This is how I'd organize a simple project.

Separate the raw data from processed data.

Separate code from data.

Include a Readme file and a Makefile.

I tend to reuse file names. Almost every project will have an `R/prepData.R` script.

Of course, each project is under version control (with git)!

`R/analysis.R` usually has exploratory analyses, and then there'll be separate `.Rmd` files with more finalized work.

Organizing a paper

```
Docs/Papers/PhyloQTL/

    Analysis/
    Data/
    Figs/
    Notes/
    R/
    SuppFigs/

    ReadMe.txt
    Makefile
    phyloqtl.tex
    phyloqtl.bib

    Submitted/
    Reviews/
    Revised/
    Final/
    Proofs/
```

This is how I organize the material for a paper.

`R/` contains code for figures; `Analysis/` contains other analysis code; `Data/` contains data; `Figs/` contains the figures; `Notes/` contains notes or references.

Of course, a `Makefile` for compiling the PDF, and perhaps a `ReadMe` file to explain where things are.

And I'll save the submitted version (and text files with bits for web forms at submission), plus reviews, the revised version plus response to reviews, and then the final submitted version and the proofs.

Organizing a talk

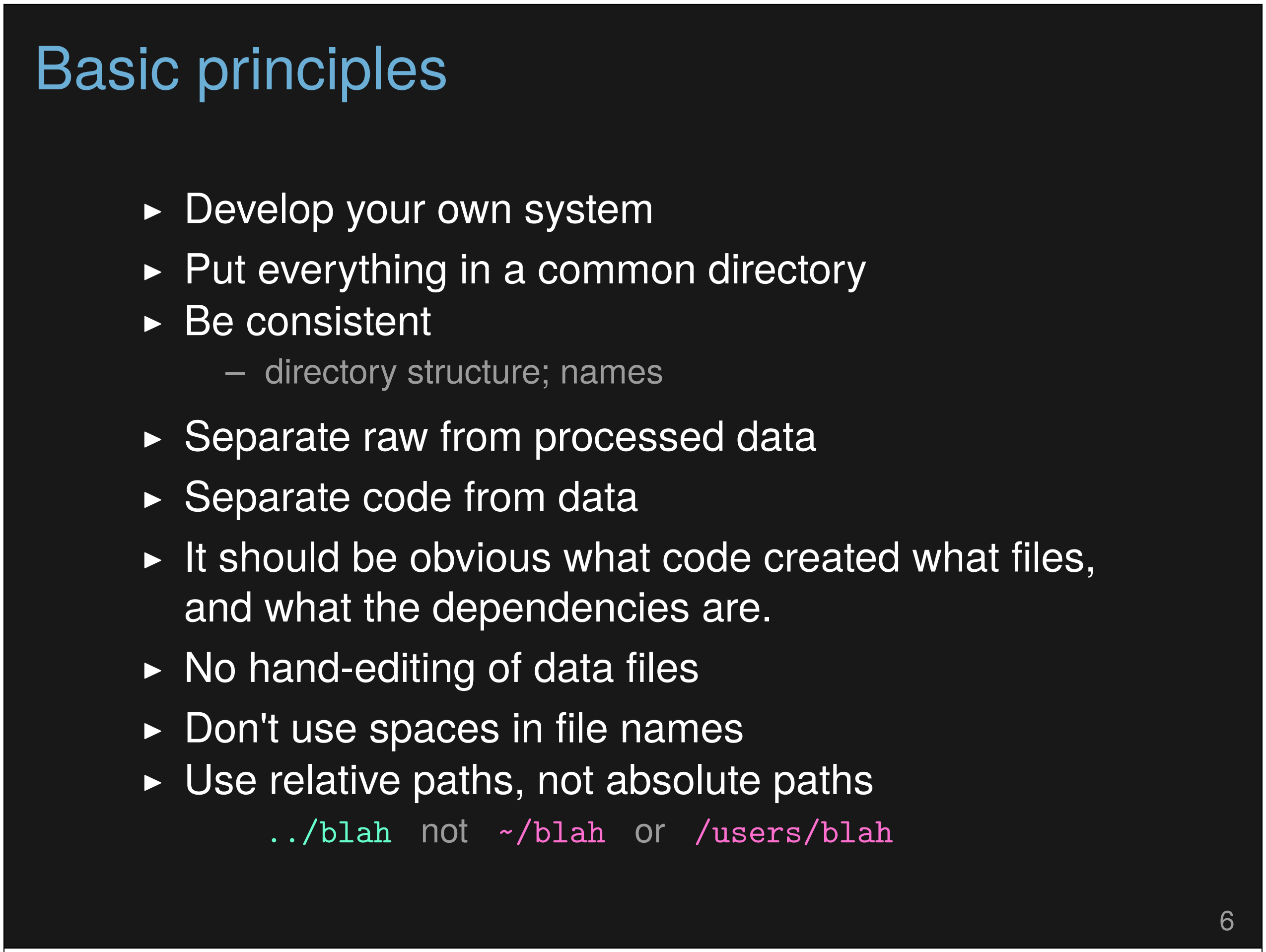
```
Docs/Talks/SampleMixups/

    Figs/
    R/

    ReadMe.txt
    Makefile
    bmi2013.tex

    Old/
```

This is how I organize the material for a talk: much like a paper, but generally a bit simpler.

Again, `R/` contains code for figures and `Figs/` contains the actual figures.

And again, a `Makefile` for compiling the PDF, and perhaps a `ReadMe` file to explain where things are.

And I'll save all old versions in `Old/`



Basic principles

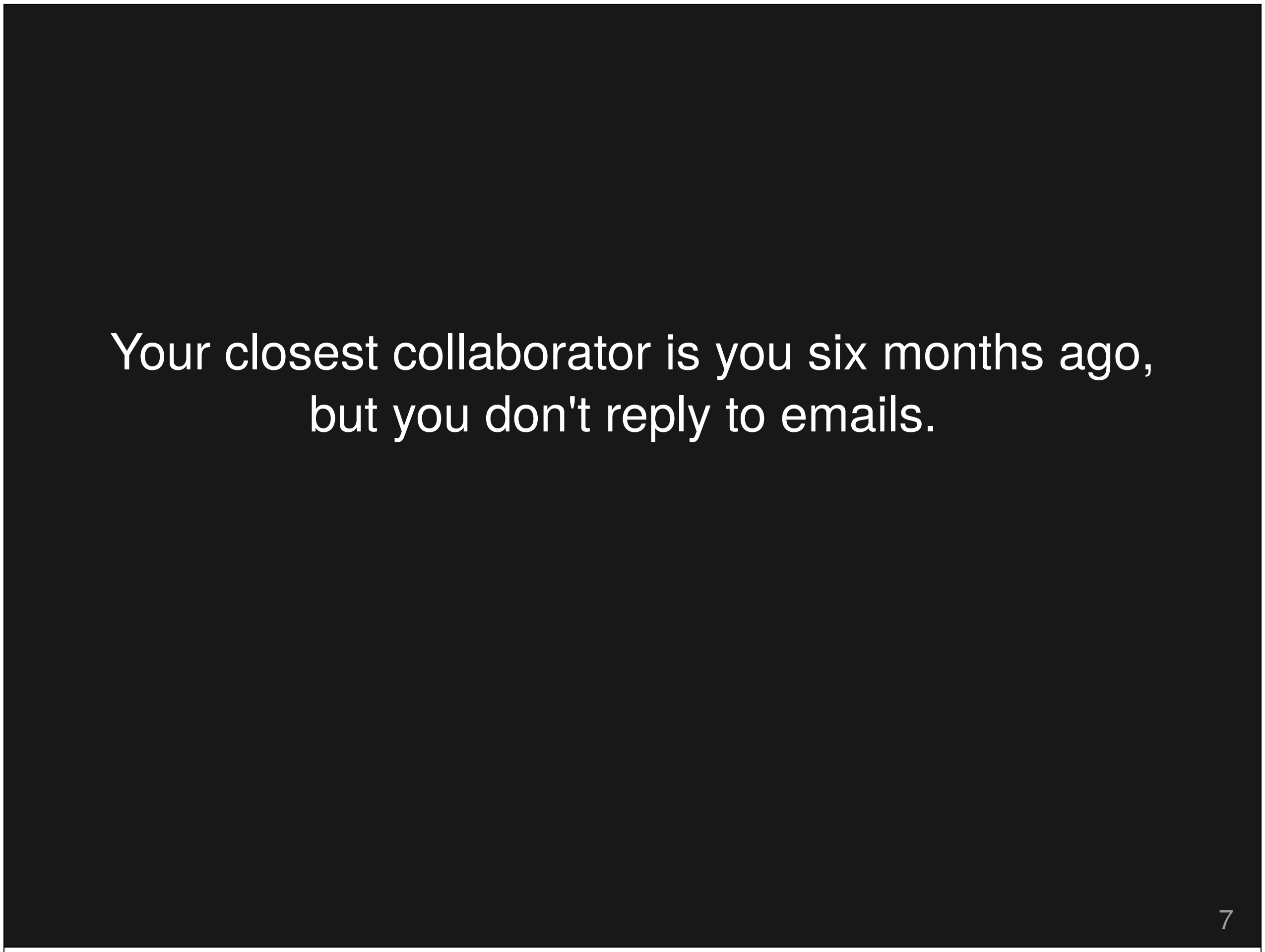
- Develop your own system
- Put everything in a common directory
- Be consistent
  - directory structure; names
- Separate raw from processed data
- Separate code from data
- It should be obvious what code created what files, and what the dependencies are.
- No hand-editing of data files
- Don't use spaces in file names
- Use relative paths, not absolute paths

  `../blah` not `~/blah` or `/users/blah`

I work on many different projects at the same time, and I'll come back to a project 6 months or a year later.

I don't want to spend much time figuring out where things are and how things were created: have a `Makefile`, and keep notes. But notes are not necessarily correct while a `Makefile` would be.

Plan for the whole deal to ultimately be open to others: will you be proud of the work, or embarrassed by the mess?



Your closest collaborator is you six months ago,
but you don't reply to emails.

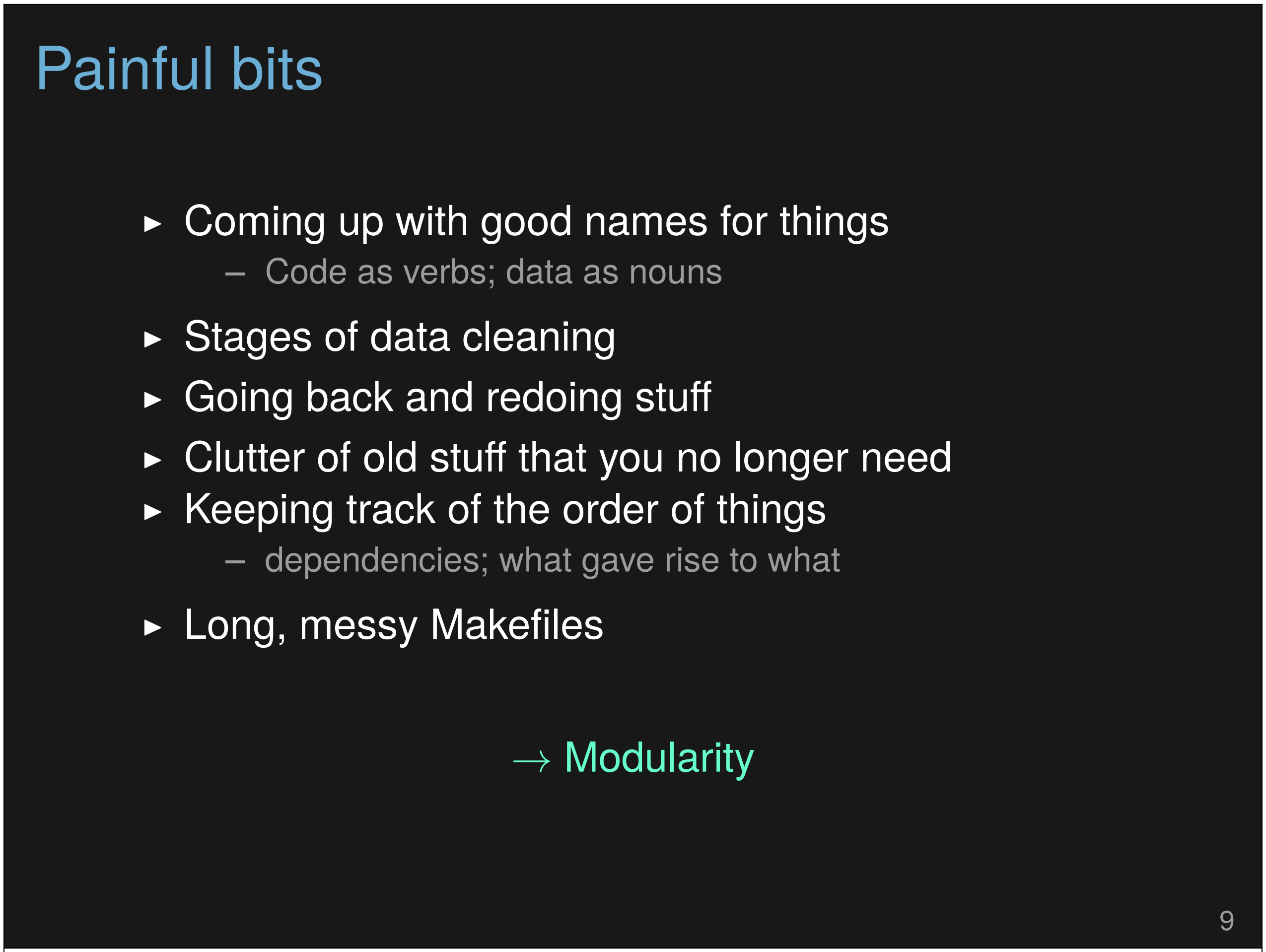
I heard this from Paul Wilson, UW-Madison.

Organization takes time.

There’s no getting around the fact that doing things properly takes longer, in the short term.

If you have a good system and good habits, it won’t seem like it takes so long.

But definitely, it’s a large up-front investment in order to potentially save a lot of time and aggravation later.



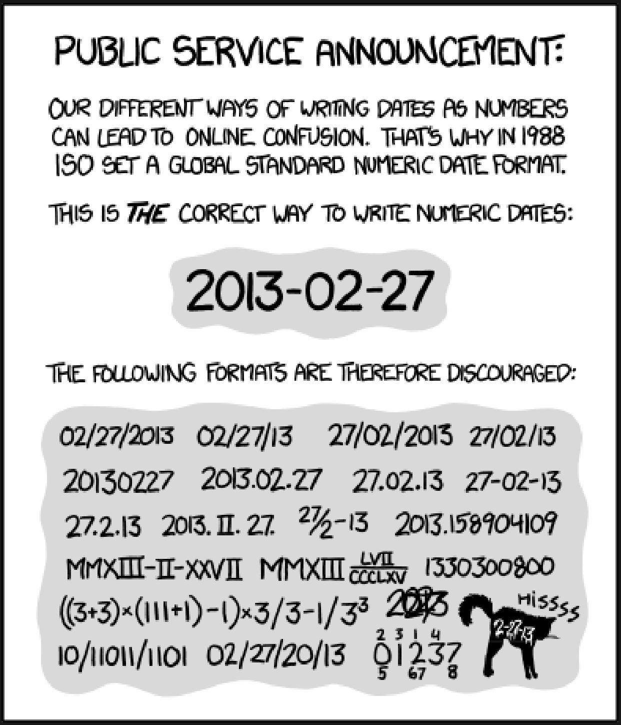
Painful bits

- Coming up with good names for things
  - Code as verbs; data as nouns
- Stages of data cleaning
- Going back and redoing stuff
- Clutter of old stuff that you no longer need
- Keeping track of the order of things
  - dependencies; what gave rise to what
- Long, messy Makefiles

→ Modularity

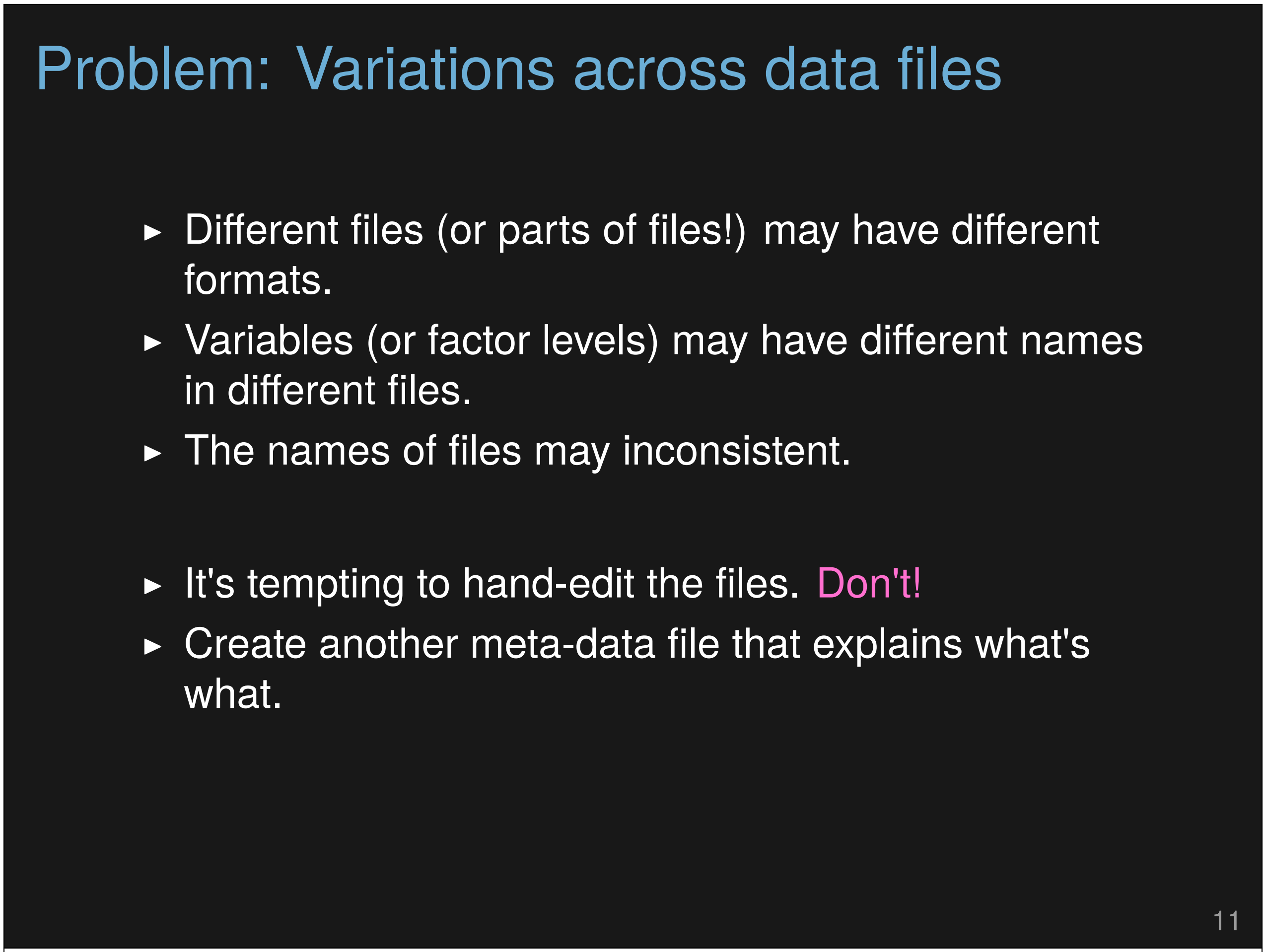
I don't have many solutions to these problems. Version control helps. And try to break things down into different stages, in case one aspect needs to be revised. Maybe use different subdirectories for the different stages of data cleaning.

A point that was raised in the discussion: Have periodic "versions" for a project, perhaps labeled by date. Move all the good stuff over and retire the stuff that is no longer useful or necessary.


PUBLIC SERVICE ANNOUNCEMENT:
OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT.
THIS IS THE CORRECT WAY TO WRITE NUMERIC DATES:
2013-02-27
THE FOLLOWING FORMATS ARE THEREFORE DISCOURAGED:
02/27/2013 02/27/13 27/02/2013 27/02/13
20130227 2013.02.27 27.02.13 27-02-13
27.2.13 2013. II. 27. 27⁄2-13 2013.158904109
MMXIII-II-XXVII MMXIII LVII/CCCLXV 1330300800
((3+3)×(111+1)-1)×3/3-1/3³ 2013
HISSSSS
2-27-13
10/11011/1101 02/27/20/13 01237

xkcd.com/1179

Go with the xkcd format for writing dates, for ease of sorting.



Problem: Variations across data files

- Different files (or parts of files!) may have different formats.
- Variables (or factor levels) may have different names in different files.
- The names of files may inconsistent.

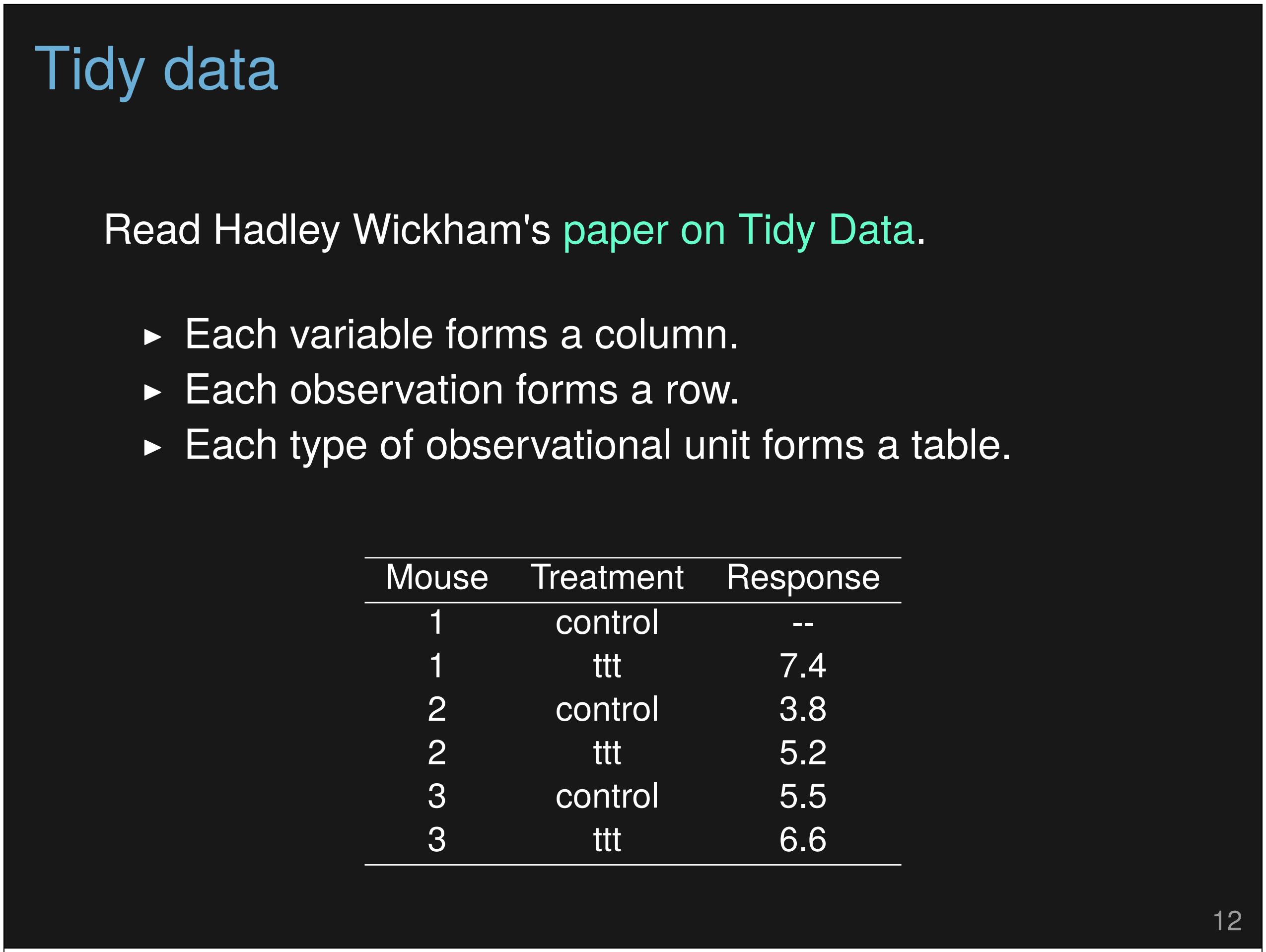
- It's tempting to hand-edit the files. Don't!
- Create another meta-data file that explains what's what.

Scientists aren't trained in how to organize data.

Multiple people in a lab might have his/her own system, or an individual's system may change over time (or from the top to the bottom of a file!)

Create a separate file with meta-data: "These are the files. In this file, the variable is called blah while in that file it's blather."

The meta-data file should be structured as data (e.g., as a comma- or tab-delimited file) for easy parsing.



Tidy data

Read Hadley Wickham's paper on Tidy Data.

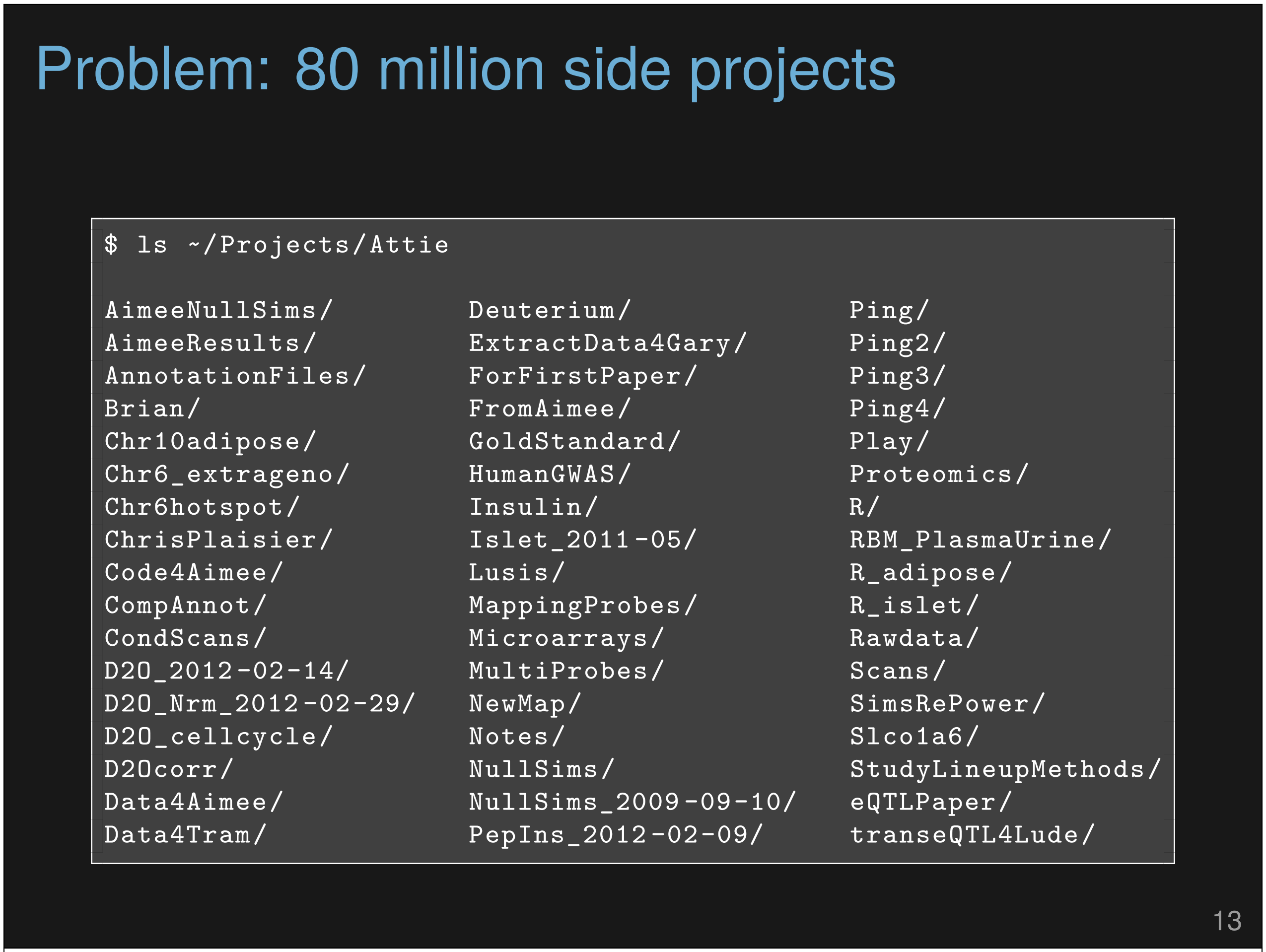
- Each variable forms a column.
- Each observation forms a row.
- Each type of observational unit forms a table.

Mouse	Treatment	Response
1	control	--
1	ttt	7.4
2	control	3.8
2	ttt	5.2
3	control	5.5
3	ttt	6.6

Read the paper!

When you convert data into a better form, convert it into the tidy form.

Also, consider Hadley's tools, like dplyr



Problem: 80 million side projects

```
$ ls ~/Projects/Attie

AimeeNullSims/        Deuterium/             Ping/
AimeeResults/         ExtractData4Gary/      Ping2/
AnnotationFiles/      ForFirstPaper/         Ping3/
Brian/                FromAimee/             Ping4/
Chr10adipose/         GoldStandard/          Play/
Chr6_extrageno/       HumanGWAS/             Proteomics/
Chr6hotspot/          Insulin/               R/
ChrisPlaisier/        Islet_2011-05/         RBM_PlasmaUrine/
Code4Aimee/           Lusis/                 R_adipose/
CompAnnot/            MappingProbes/         R_islet/
CondScans/            Microarrays/           Rawdata/
D2O_2012-02-14/       MultiProbes/           Scans/
D2O_Nrm_2012-02-29/   NewMap/                SimsRePower/
D2O_cellcycle/        Notes/                 Slco1a6/
D2Ocorr/              NullSims/              StudyLineupMethods/
Data4Aimee/           NullSims_2009-09-10/   eQTLPaper/
Data4Tram/            PepIns_2012-02-09/     transeQTL4Lude/
```

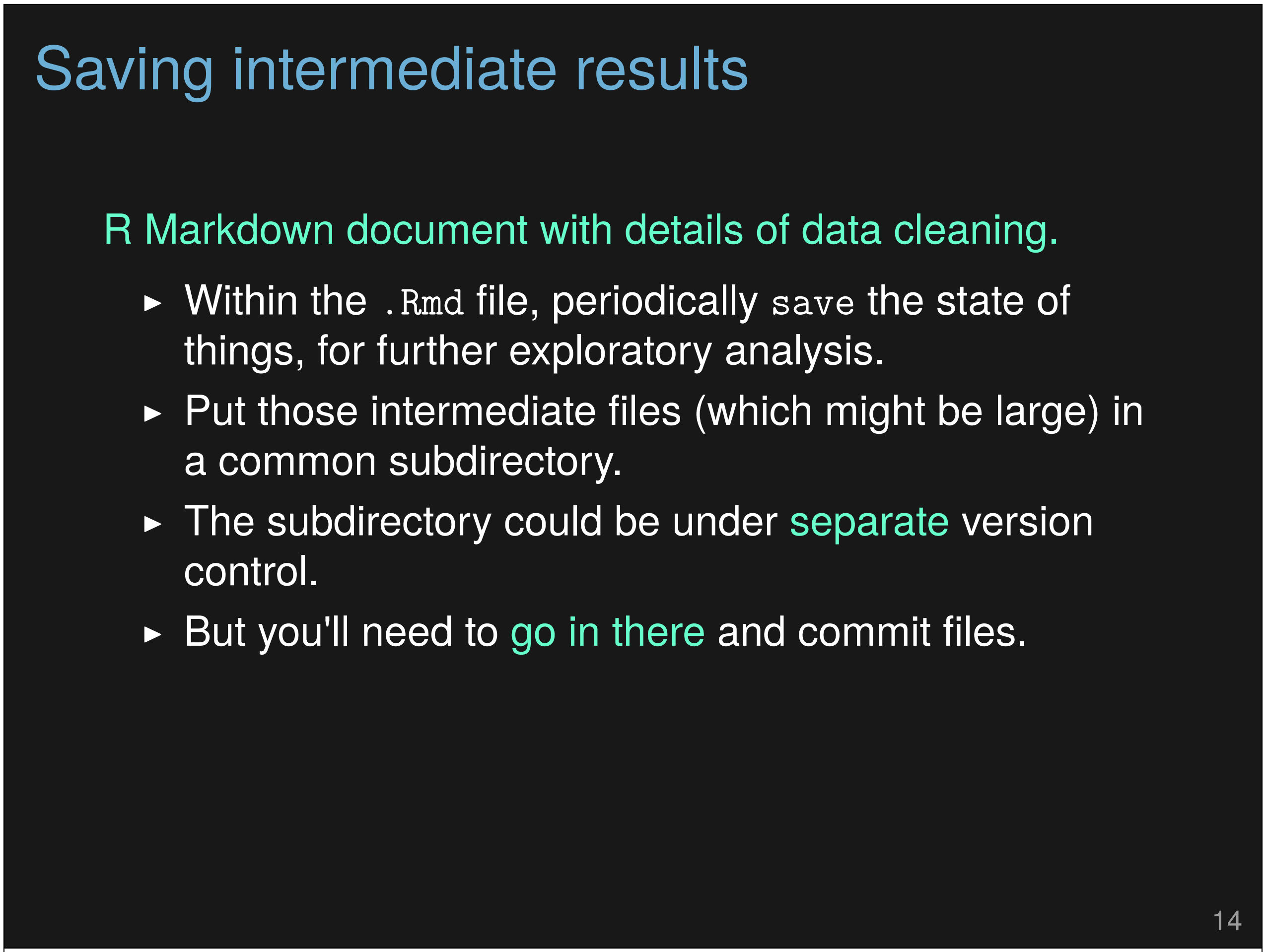
This is a project-gone-wrong.

A key problem in research is that you don't really know what you're doing when you get started. It seems best to separate out each side-project as a separate directory, but it can be a nightmare to find things later.

If each of these subdirectories was nicely organized and had a `ReadMe` file, you could `grep` your way through them.

I sort of like the idea of separate directories for the different aspects of mucking about. And second versions are always better. Maybe we should plan to muck about separately and then bring a more refined analysis back into a common directory?

A point raised in the discussion: Put defunct side projects into an `Old/` subdirectory, and put active but not yet clearly interesting ones into `New/` or `Play/`. This will help to avoid the clutter.



Saving intermediate results

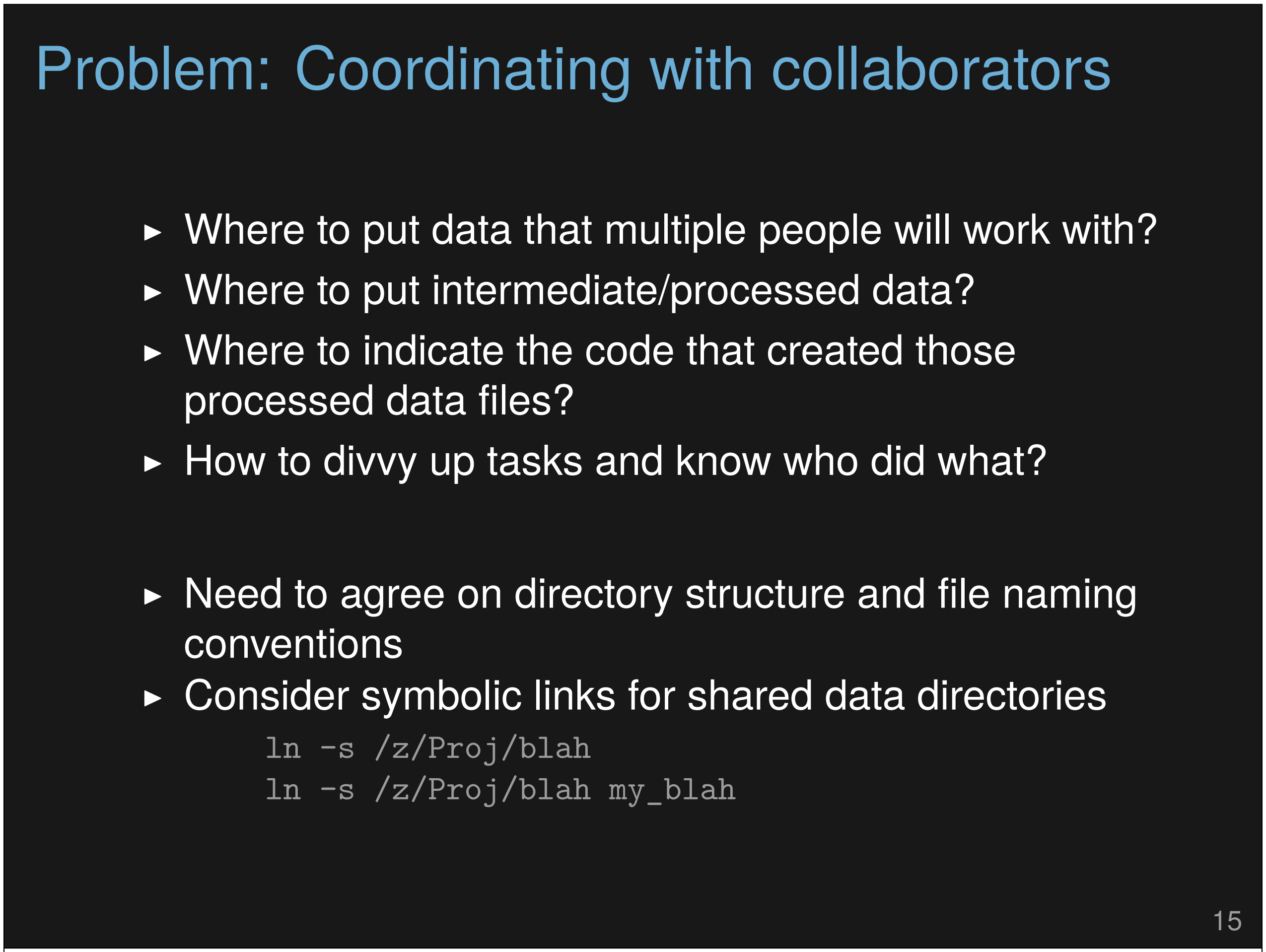
R Markdown document with details of data cleaning.

- Within the `.Rmd` file, periodically `save` the state of things, for further exploratory analysis.
- Put those intermediate files (which might be large) in a common subdirectory.
- The subdirectory could be under separate version control.
- But you'll need to go in there and commit files.

I want a reproducible analysis document, but I want to be able to grab objects from the middle of the process for further exploration. So I'll include code chunks to save the state of things, say in a `Cache` or `RData` subdirectory.

Subdirectories can be their own git repositories: Include that subdirectory in the `.gitignore` file, and then use `git init` within the subdirectory.

A point raised in the discussion: how to synchronize a project between computers? If we don't put the intermediate files in the main repository, we can't rely on GitHub. (For a simple manuscript or talk, it's okay to reconstruct things on another computer, but for big analyses, you wouldn't want to.) I use ChronoSync to synchronize my Mac desktop and laptop. Maybe Dropbox or Google Drive would be useful for this. You'd still want to use git and and GitHub, but you could supplement them by having the repository sit in your Dropbox folder.



Problem: Coordinating with collaborators

- Where to put data that multiple people will work with?
- Where to put intermediate/processed data?
- Where to indicate the code that created those processed data files?
- How to divvy up tasks and know who did what?

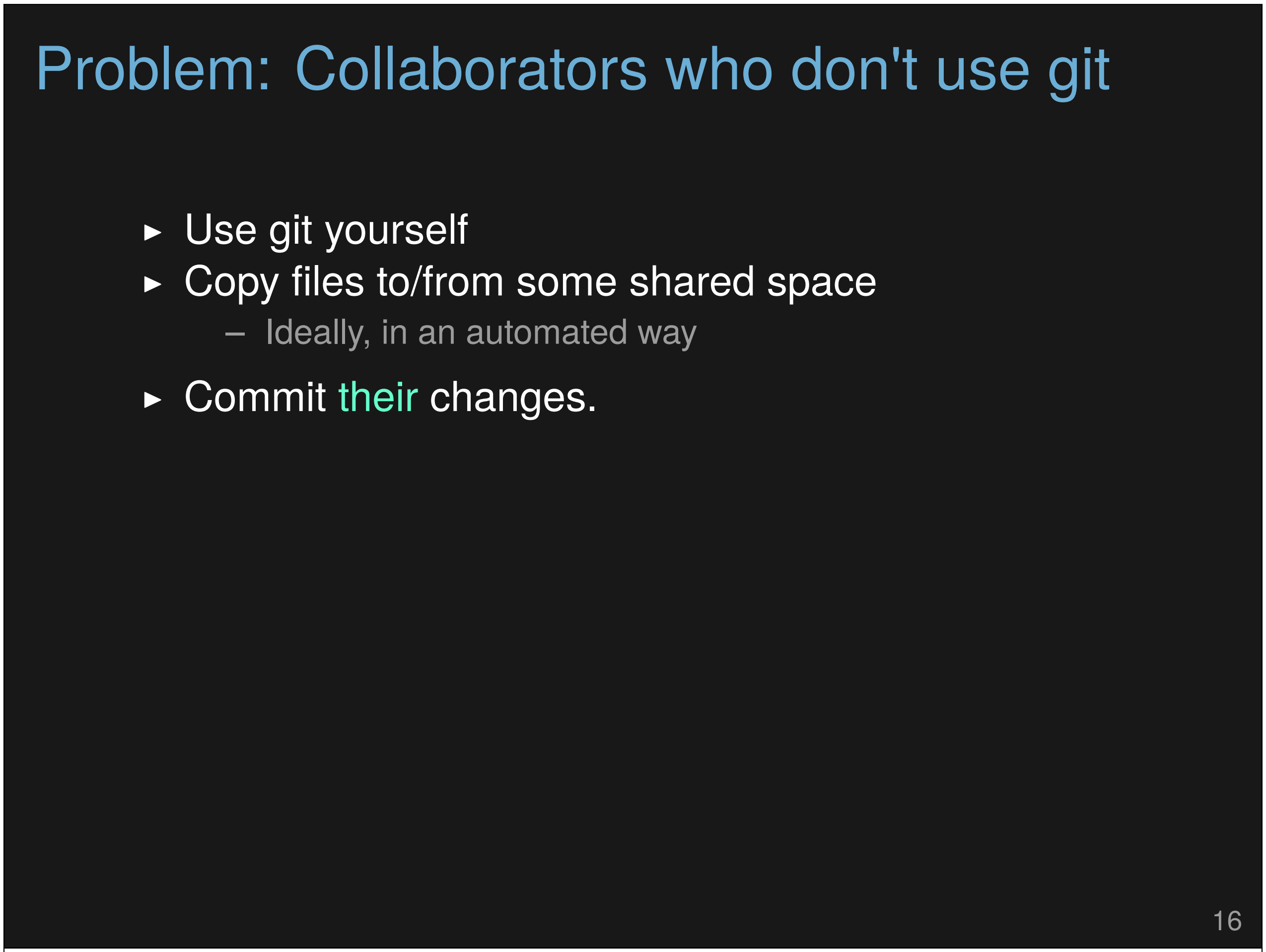
- Need to agree on directory structure and file naming conventions
- Consider symbolic links for shared data directories

```
ln -s /z/Proj/blah
ln -s /z/Proj/blah my_blah
```

Ideally, everything synchronized with git/GitHub.

The keys: planning and regular communication

Symbolic links are also called “soft links.” It’s just like a file shortcut in Windows.

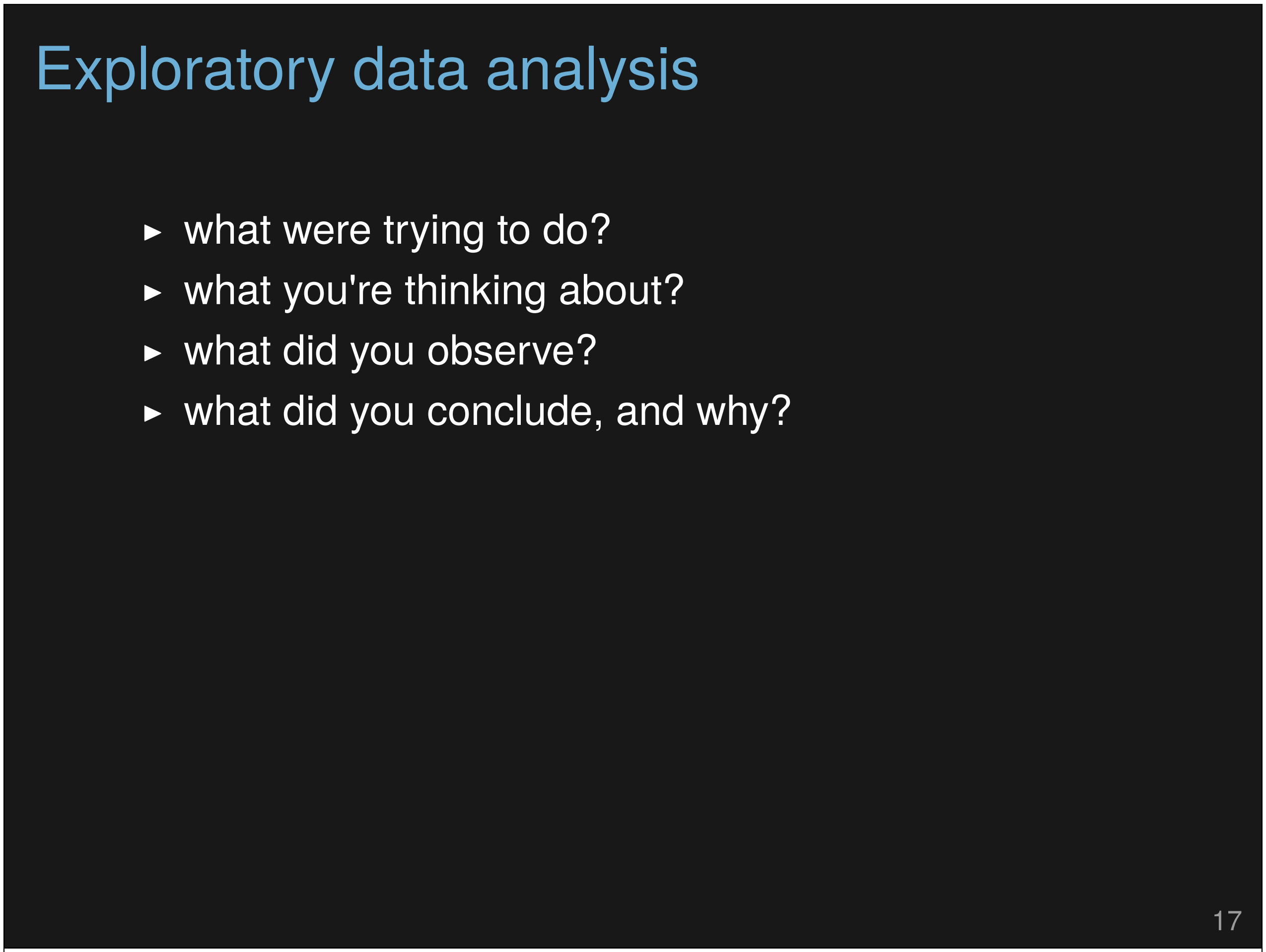


Problem: Collaborators who don't use git

- Use git yourself
- Copy files to/from some shared space
  - Ideally, in an automated way
- Commit *their* changes.

Life would be easier if all of our analysis collaborators adopted git. Teach them how?!

When I'm working with a collaborator on a paper, I may get comments from them as a marked-up PDF. I'll save that in the repository and will incorporate and commit the changes in the source files, on my own.

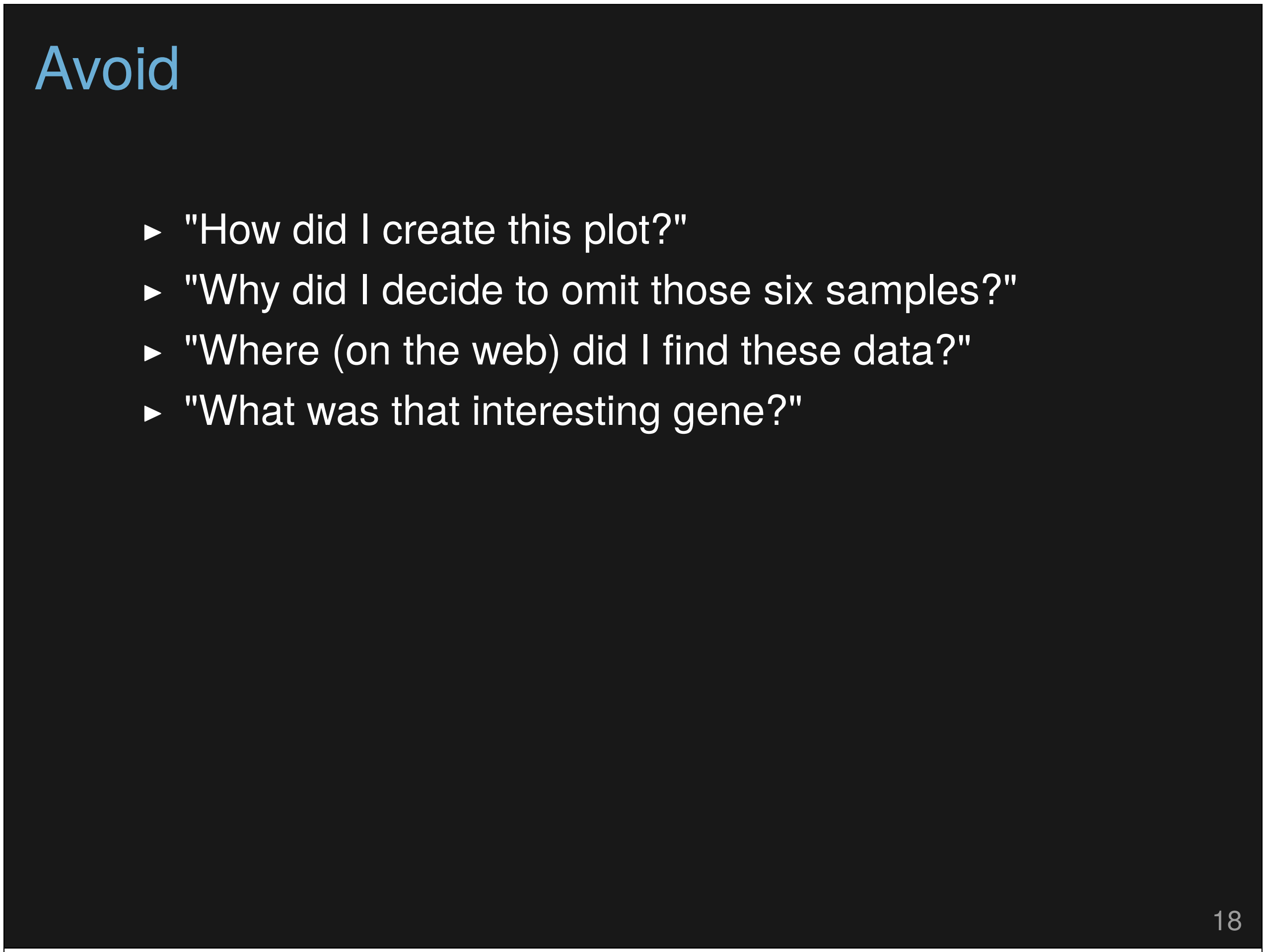


Exploratory data analysis

- what were trying to do?
- what you're thinking about?
- what did you observe?
- what did you conclude, and why?

We want to be able to capture the full outcome of exploratory data analysis.

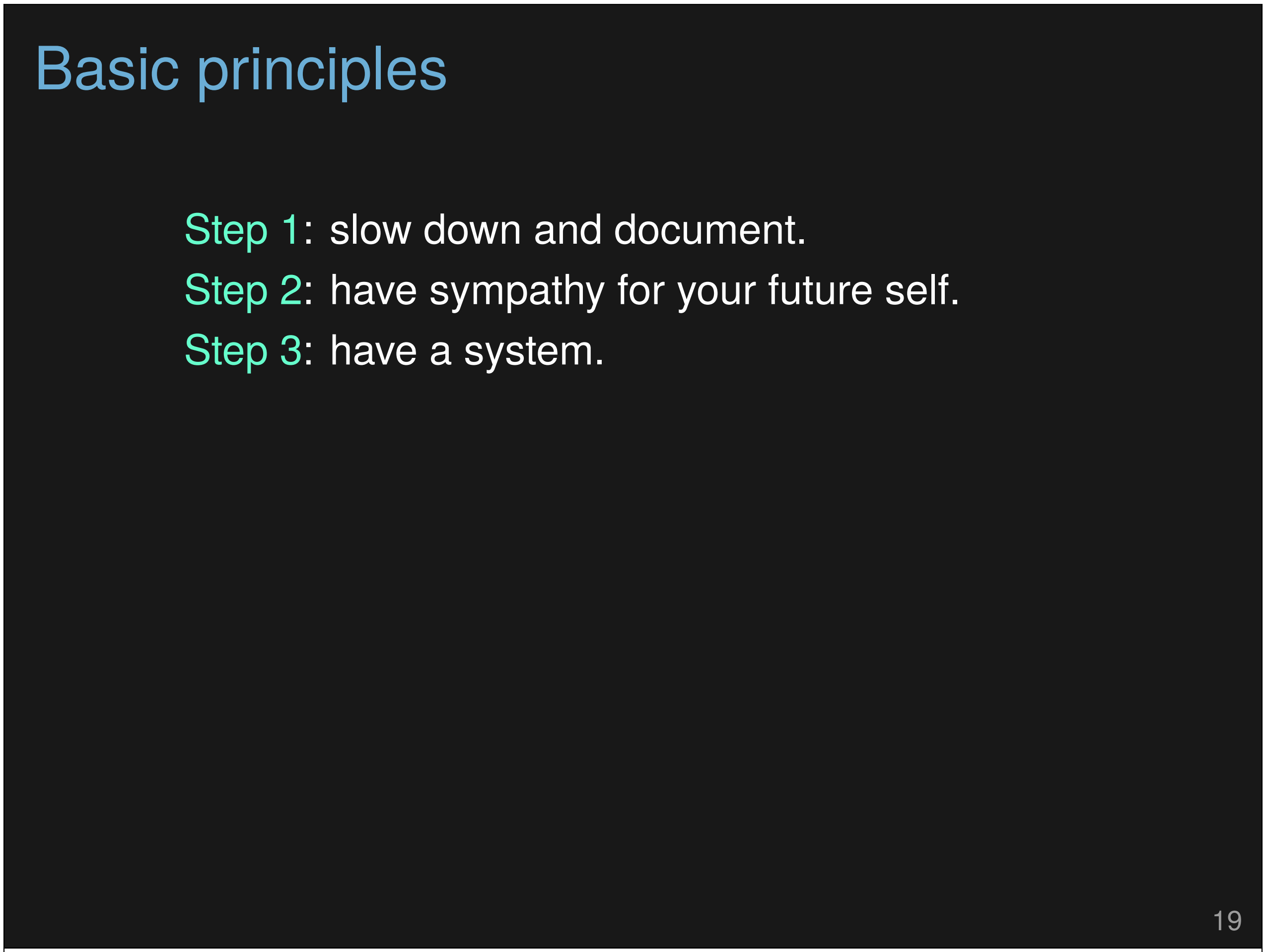
But we don't want to inhibit the creative flow. How to capture this stuff?



Avoid

- "How did I create this plot?"
- "Why did I decide to omit those six samples?"
- "Where (on the web) did I find these data?"
- "What was that interesting gene?"

I’ve said all of these things to myself.



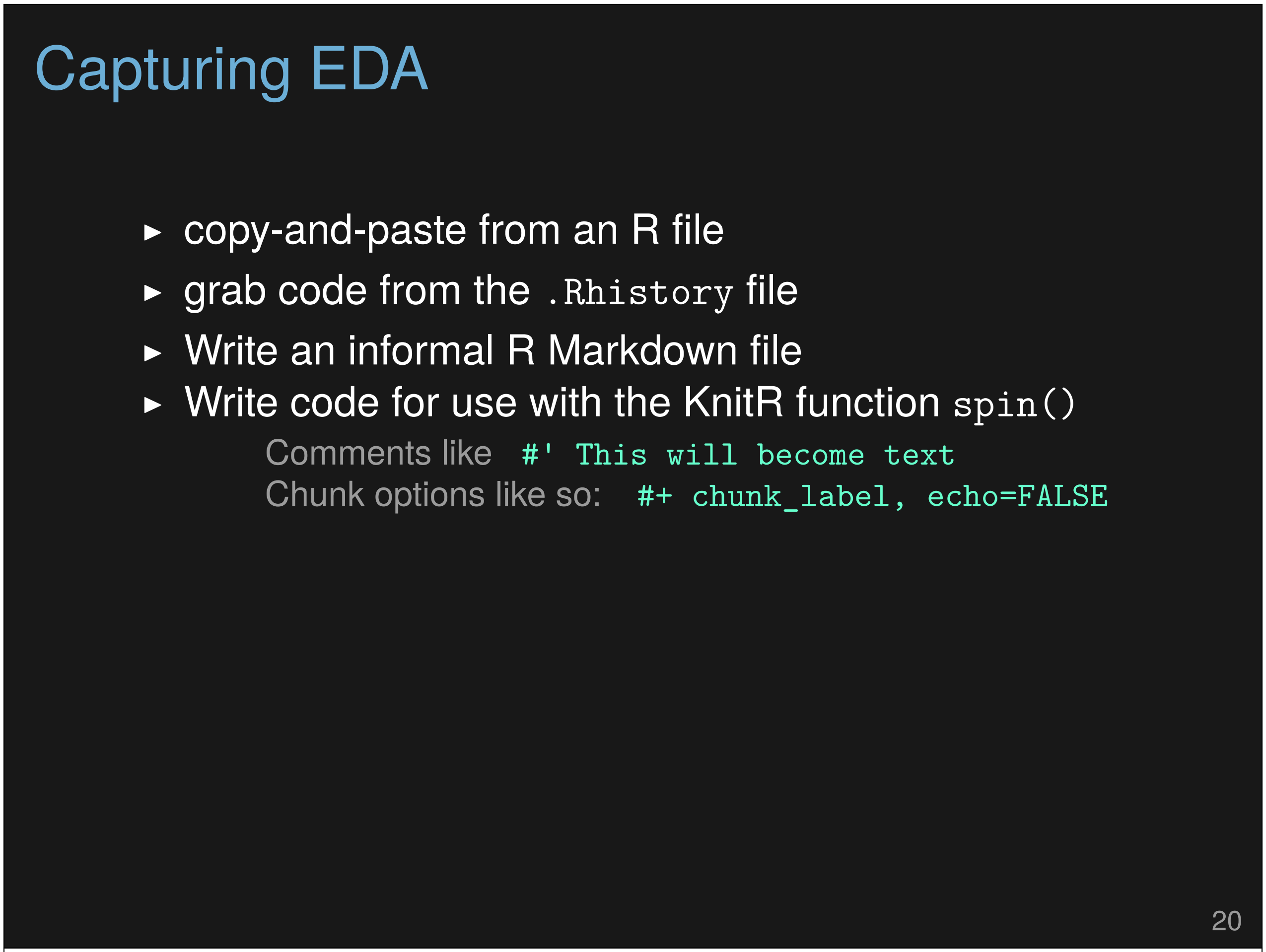
Basic principles

Step 1: slow down and document.
Step 2: have sympathy for your future self.
Step 3: have a system.

I can't emphasize these things enough.

If you're not thinking about keeping track of things, you won't keep track of things.

One thing I like to do: write a set of comments describing my basic plan, and then fill in the code afterwards. It forces you to think things through, and then you'll have at least a rough sense of what you were doing, even if you don't take the time to write further comments.



Capturing EDA

- copy-and-paste from an R file
- grab code from the `.Rhistory` file
- Write an informal R Markdown file
- Write code for use with the KnitR function `spin()`

  Comments like `#' This will become text`
  Chunk options like so: `#+ chunk_label, echo=FALSE`

There are a number of techniques you can use to capture the EDA process.

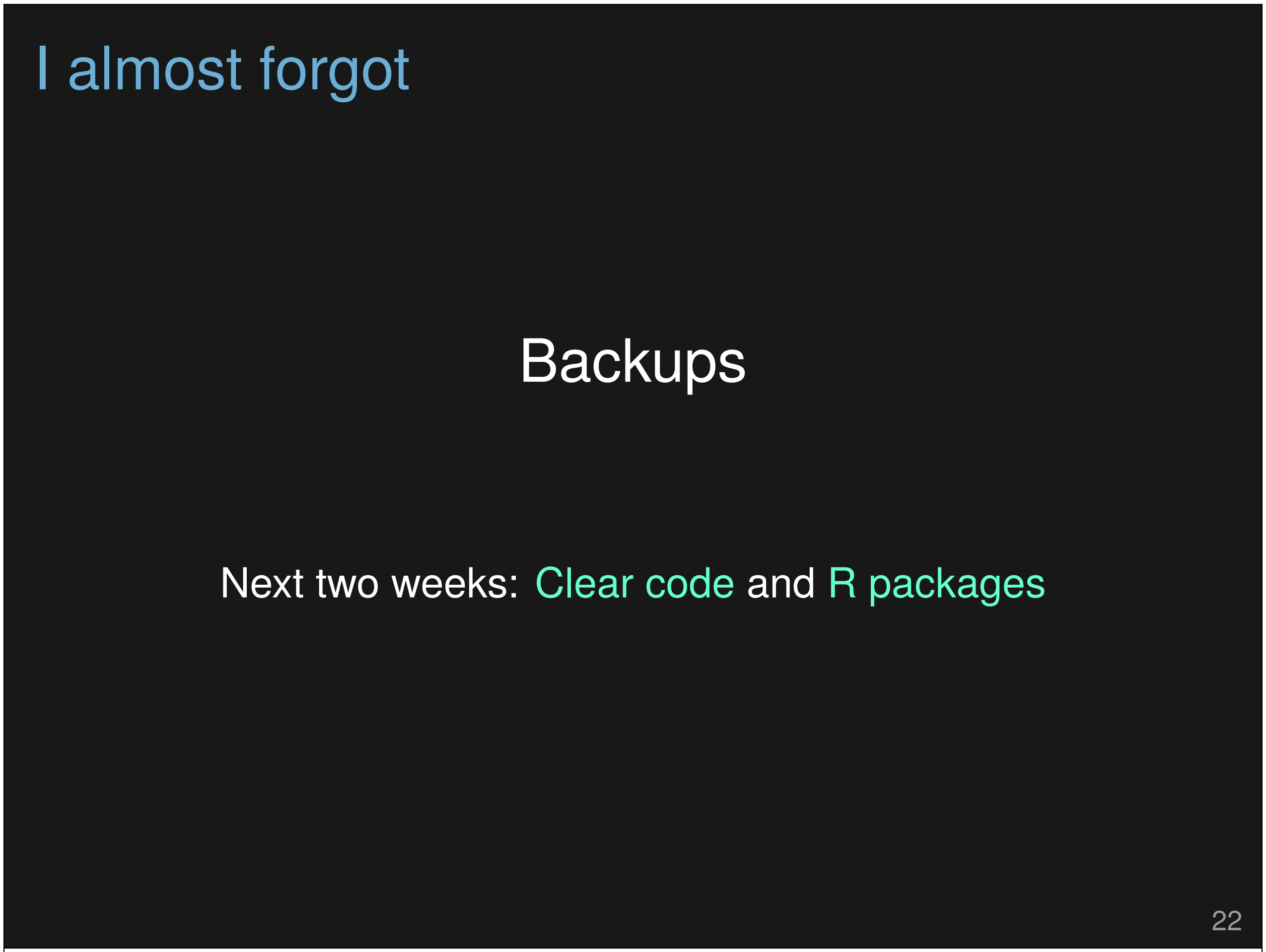
You don't need to save all of the figures, but you do need to save the code and write down your motivation, observations, and conclusions.

I usually start out with a plain R file and then move to more formal R Markdown or AsciiDoc reports.

A file to spin()

```
#' This is a simple example of an R file for use with spin().

#' We'll start by setting the seed for the RNG.
set.seed(53079239)

#' We'll first simulate some data with x ~ N(mu=10, sig=5) and
#' y = 2x + e, where e ~ N(mu=0, sig=2)
x <- rnorm(100, 10, 5)
y <- 2*x + rnorm(100, 0, 2)

#' Here's a scatterplot of the data.
plot(x, y, pch=21, bg="slateblue", las=1)
```

Here's an example R file for use with `spin()`.



I almost forgot

Backups

Next two weeks: Clear code and R packages

You must back up your stuff.

You can generally rely on your department server, and you should also make use of GitHub. But if you have other stuff on a laptop or at home, you want to be sure to back that up to. Hard drives are cheap.

On a Mac, I use the built-in Time Machine, but I also use SuperDuper! to create a bootable clone.

Also important in all of this is writing clear, modular code. With R, it's best to pull out reuseable code as an R package. We'll talk about these two topics over the next two weeks.