1: Intro to Reproducible Research

`bit.ly/SISBID3`

This is the introductory lecture for the SISBID Workshop on Reproducible Research.

A minimal standard for data analysis and other scientific computations is that they be reproducible: that the code and data are assembled in a way so that another group can re-create all of the results (e.g., the figures in a paper). The importance of such reproducibility is now widely recognized, but it is not so widely practiced as it should be, in large part because many computational scientists (and particularly statisticians) have not fully adopted the required tools for reproducible research.

```
Karl -- this is very interesting,
however you used an old version of
the data (n=143 rather than n=226).

I'm really sorry you did all that
work on the incomplete dataset.

Bruce
```

This is an edited version of an email I got from a collaborator, in response to an analysis report that I had sent him.

I try to always include some brief data summaries at the start of such reports. By doing so, he immediately saw that I had an old version of the data.

Because I'd set things up carefully, I could just substitute in the newer dataset, type "`make`", and get the revised report.

This is a reproducibility success story. But it took me a long time to get to this point.

The results in Table 1 don't seem to correspond to those in Figure 2.

My computational life is not entirely rosy. This is the sort of email that will freak me out.

In what order do I run these scripts?

Sometimes the process of data file manipulation and data cleaning gets spread across a bunch of scripts that need to be executed in a particular order. Will I record this information? Is it obvious what script does what?

Where did we get this data file?

Record the provenance of all data or metadata files.

Why did I omit those samples?

I may decide to omit a few samples. Will I record why I omitted those particular samples?

How did I make that figure?

Sometimes, in the midst of a bout of exploratory data analysis, I'll create some exciting graph and have a heck of a time reproducing it afterwards.

"Your script is now giving an error."

It was working last week. Well, last month, at least.

How easy is it to go back through that script's history to see where
and why it stopped working?

"The attached is similar to the code we used."

From an email in response to my request for code used for a paper.

Reproducible

vs.

Replicable

Computational work is reproducible if one can take the data and code and produce the same set of results. Replicable is more stringent: can someone repeat the experiment and get the same results?

Reproducibility is a minimal standard. That something is reproducible doesn't imply that it is correct. The code may have bugs. The methods may be poorly behaved. There could be experimental artifacts.

(But reproducibility is probably correlated with correctness.)

Note that some scientists say replicable for what I call reproducible, and vice versa.

Levels of quality

- ► Are the tables and figures reproducible from the code and data?

- ► Does the code actually do what you think it does?

- ► In addition to what was done, is it clear why it was done?
    - (e.g., how were parameter settings chosen?)

- ► Can the code be used for other data?

- ► Can you extend the code to do other things?

Reproducibility is not black and white. And the ideal is hard to achieve.

Basic principles

- ▶ Project encapsulated in one directory

- ▶ Organize and document

- ▶ Keep track of the provenance of all data files

- ▶ Everything via code

- ▶ Use a version control system

- ▶ Keep track of versions of dependencies

- ▶ Everything automated

12

Pointing and clicking is not reproducible. Ideally, you press just one button.

Make sure you have all of the data and that you know exactly where it came from.

But what is raw data? How far back should you go? Data that I get from collaborators has usually gone through a considerable amount of pre-processing. Should we have captured that, in order for the work to be considered reproducible?

If your collaborator asks, "In what form would you like the data?" you should respond, "In its current form."

Why do we care?

- ▸ Avoid embarrassment

- ▸ More likely correct

- ▸ Save time, in the long run

- ▸ Greater potential for extensions; higher impact

Doing things properly (writing clear, documented, well-tested code) is time consuming, but it could save you a ton of aggravation down the road. Ultimately, you'll be more efficient, and your work will have greater impact.

Your code and analyses will be easier to debug, maintain, and extend.

Try to avoid

- ▸ Open a file to extract as CSV

- ▸ Open a data file to do even a slight edit

- ▸ Paste results into the text of a manuscript

- ▸ Copy-paste-edit tables

- ▸ Copy-paste-adjust figures

If you do anything "by hand" once, you'll have to do it 100 times.

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.

Basic tools

- ▶ File organization and naming

- ▶ RMarkdown

- ▶ R packages

- ▶ Version control with git/GitHub

- ▶ Automation with Make

These are the basic tools that we will cover in this workshop. RMarkdown allow preparation of beautiful documents without pointing or clicking, and combining code and text.

R's packaging system is among its best features.

Version control isn't strictly necessary for reproducibility, but once you get the hang of it, you'll never go back.

Make is for automation and for documenting dependencies. For reproducibility, the command line is your best friend.

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

(paraphrasing Mark Holder)

The first thing to do is to make your project understandable to others
(or yourself, later, when you try to figure out what it was that you
did.

Segregate all the materials for a project in one directory/folder on
your harddrive.

I prefer to separate raw data from processed data, and I put code in a
separate directory.

Write `ReadMe` files to explain what's what.

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?

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

Painful bits

- ▶ Coming up with good names for things
  - – Concise but informative
  - – Code as verbs; data as nouns
  - – Avoid spaces; avoid symbols except - and _

- ▶ 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

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.

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.

Keep track of versions of things
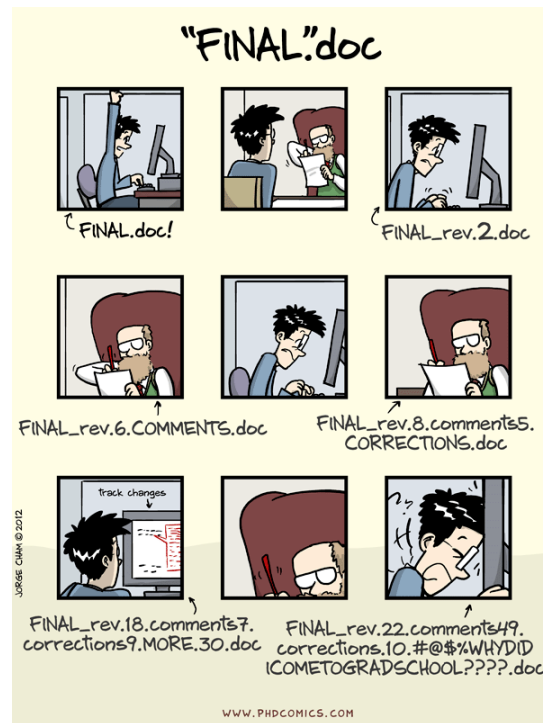
Never include "final" in a file name.

And don't forget...

Backups

You must back up your stuff.

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

The most important tool is the mindset,
when starting, that the end product
will be reproducible.

– Keith Baggerly

So true. Desire for reproducibility is step one.