Steps toward reproducible research

Karl Broman

Biostatistics & Medical Informatics, UW–Madison

kbroman.org
github.com/kbroman
@kwbroman
Course web: kbroman.org/AdvData

This lecture is based on slides for a talk I've given a whole bunch of times.

The latest version of the slides is available at the following:

Source: `https://github.com/kbroman/Talk_ReproRes`
These slides, with notes: `https://bit.ly/steps2rr`
Full slides without notes: `https://bit.ly/steps2rr_nonotes`

By "reproducibly," I'm referring to "computational reproducibility," by which I mean that the data and code for a project are packaged together in a way that they can be handed to someone else, who can rerun the code and get the same results—the same figures and tables. This is surprisingly hard to do, and it's even more difficult in the context of a collaboration between two or more data analysts.

This lecture is an overview of reproducible research and the basic principles and tools for ensuring that computational work is reproducible. In subsequent lectures, I'll be drilling into individual tools/topics in more detail.

```
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
```


I'm an applied statistician; my goal is to help people make sense of their data. I have a lot of collaborators, and there's nothing I enjoy more than puzzling over their data. So I write a lot of reports, describing what I've done and what I've learned.

This is an email I got from a collaborator, in response to an analysis report that I had sent him. It's always a bit of a shock to get an email like this: what have I done? Why am I working with the wrong data, and where is the right data?

But what he didn't know is that by this point in my life, I'd adopted a reproducible workflow. Because I'd set things up carefully, I could just substitute in the newer dataset, type a single command ("`make`") to rerun the analyses, and get the revised report.

This is a reproducibility success story. We all make mistakes, but if our projects are reproducible, we can nimbly recover from those mistakes.

There is a second important lesson here: At the start of such reports, I always include a paragraph about our shared goals, along with some brief data summaries. By doing so, he immediately saw that I had an old version of the data. If I hadn't done so, we might never have discovered my error.

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.

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?

Which image goes with which experiment?

For experimental biologists, it can be tricky to keep track of the vast set of images and experiments they perform.

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.

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?

“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 when 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 the data and code are organized in a way that they can be handed to someone else, who can rerun the code and get the same results—the same figures and tables. 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 associated with correctness.)

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

kbroman.org/steps2rr

It was a long, hard process for me to move from my old standard practice to a fully reproducible workflow. In thinking through that process, I wrote down my thoughts on the basic steps to take towards full reproducibility. This forms the basis of what I'll present here.

A little bit reproducible
is better than not reproducible.

A little bit open
is better than not open.

Strive to make each project
a bit better organized than the last.

While it's good to strive for full reproducibility, it can be difficult to achieve. But partially reproducible is better than not-at-all reproducible. Similarly, making data and code partially open is better than nothing.

Don't try to change every aspect of your workflow all at once. Focus on revising one aspect at a time. When you get to the end of a project, you may be dissatisfied with the state of things, but don't give up. Try to make each project a bit better organized and reproducible than the last.

Organize your project

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).

Organize your project

```
RawData/                    Notes/
DerivedData/                Refs/

Python/                     ReadMe.txt
R/                          ToDo.txt
Ruby/                       Makefile

Analysis/
Figures/
```


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

There will be a lot of files. Organize them in a meaningful way.

This is the way I organize a project directory. The key principles are to put everything related to a project in a common directory, but then to separate data from code and separate raw data from processed data.

Write `ReadMe` files to explain what's what. Make sure they stay current.

Chaos

```
AimeeNullSims/     Deuterium/             Ping/
AimeeResults/      ExtractData4Gary/      Ping2/
AnnotationFiles/   FromAimee/             Ping3/
Brian/             GoldStandard/          Ping4/
Chr6_extrageno/    HumanGWAS/             Play/
Chr6_segdis/       Insulin/               Prdm9/
ChrisPlaisier/     Int2_for_Mark/         RBM_PlasmaUrine_2012-03-08/
Code4Aimee/        Islet_2011-05/         Slco1a6/
CompAnnot/         MappingProbes/         StudyLineupMethods/
CondScans/         MultiProbes/           kidney_chr6.R
D2O_2012-02-14/    NewMap/                pck2_sucla2.R
D2O_cellcycle/     Notes/                 penalties.txt
D2Ocorr/           NullSims/              transeQTL4Lude/
Data4Aimee/        NullSims_2009-09-10/
Data4Tram/         PepIns_2012-02-09/
```


This is a folder on my hard drive, for the project that led me to reassess my life.

Choose good names for things

```
betw_tissue_corr.R      expr_scatterplot_allprobes.R  gve_similarity_alltissues.R
coatcolor_lod.R         expr_scatterplots_dup.R       gve_similarity.R
colors.R                expr_scatterplots_mix.R       gve_supp.R
cover_fig.R             expr_scatterplots_swap.R      insulin_lod.R
eqtl_counts_10.R        expr_swaps.R                  local_eqtl_locations.R
eqtl_counts.R           func.R                        my_plot_map.R
eve_hist.R              genotype_plates.R             my_plot_scanone.R
eve_scheme.R            gve_hist.R                    sex_vs_X.R
eve_similarity.R        gve_new.R                     xchr_fig.R
eve_similarity_supp.R   gve.R                         xist_and_y.R
expr_corr_dup.R         gve_scheme.R
expr_corr_mix.R         gve_similarity_2ndbest.R
```


You'll have a lot of files. In addition to organizing them in subfolders, it's important to choose good names for them.

These names of these files largely explain their contents, but they're also left rather disorganized.

Choose good names for things

```
fig1.png          fig5.png
fig10.png         fig6.png
fig2.png          fig7.png
fig3.png          fig8.png
fig4.png          fig9.png
```


These names are well organized, but you have to remember the order of all of the figures to find the one you want.

And note that, alphabetically, figure 10 ends up between figure 1 and figure 2.

Choose good names for things

- Machine readable
  - No spaces
  - No special characters except _ and -
- Human readable
  - Explain the contents
- Consistent
  - Name similar files in a similar way
- Make use of computer's sorting
  - pad numbers with 0's (e.g., `01`, `02`, ...)
  - start with general grouping, then more specific
  - dates like `2019-05-14`

You want the names to be easily to handle in software, which generally means no spaces or special characters except for underscore and hyphen (which are useful for separating words).

But you want the names to explain the files' contents, so that you don't have to open the files to figure out what they are.

Consistency is important: if you have a bunch of similar files, you should have some system for naming them.

And make use of the computer's sort of files, by padding numbers with 0's (so that 10 appears after 9 rather than before 2) and organizing the files into groups.

Dates should always be written as 'YYYY-MM-DD', so that when sorted they are in order by date.

xkcd.com/1179

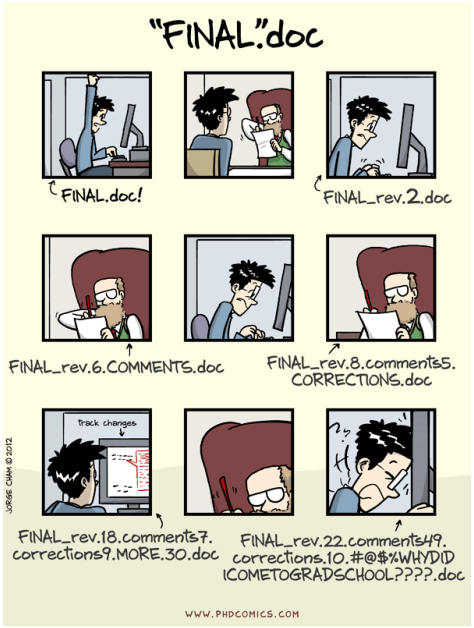
Go with the xkcd format for writing dates, for ease of sorting.

Choose good names for things

```
0_vcf2db.R
1_prep_geno.R
2_prep_pheno_clin.R
2_prep_pheno_otu.R
3_prep_covar.R
4_prep_analysis_pheno_clin.R
4_prep_analysis_pheno_otu.R
5_scans.R
6_grab_peaks.R
7_find_nearby_peaks.R
```


Here's an example to take advantage of the way the computer sorts files: a set of R scripts, which show up in the order they are used.

No “final” in file names



Never include “final” in a file name.

No “final” in file names

```
Deprecated/                          hypo_prcomp.RData
ReadMe.txt                           islet_int1_final.RData
adipose_int1_final.RData             islet_int2_final.RData
adipose_int2_final.RData             islet_mlratio_final.RData
adipose_mlratio_final.RData          islet_mlratio_nqrank_final.RData
adipose_mlratio_nqrank_final.RData   islet_prcomp.RData
adipose_prcomp.RData                 kidney_int1_final.RData
aligned_geno_with_pmap.RData         kidney_int2_final.RData
batches_final.RData                  kidney_mlratio_final.RData
batches_raw_final.RData              kidney_mlratio_nqrank_final.RData
cpl_final.RData                      kidney_prcomp.RData
d2o_final.RData                      lipomics_final_rev2.RData
gastroc_int1_final.RData             liverTG_final.RData
gastroc_int2_final.RData             liver_int1_final.RData
gastroc_mlratio_final.RData          liver_int2_final.RData
gastroc_mlratio_nqrank_final.RData   liver_mlratio_final.RData
gastroc_prcomp.RData                 liver_mlratio_nqrank_final.RData
hypo_int1_final.RData                liver_prcomp.RData
hypo_int2_final.RData                mirna_final.RData
hypo_mlratio_final.RData             necropsy_final_rev2.RData
hypo_mlratio_final_old.RData         plasmaurine_final_rev.RData
hypo_mlratio_nqrank_final.RData      pmark.RData
hypo_mlratio_nqrank_final_old.RData  rbm_final.RData
hypo_omit.RData
```


This is an actual directory on my computer. If you include `final` in a file name, there's a risk that you'll end up with `final_rev`, `final_rev2`, and `final_old`.

Another problem here is that the files aren't organized very well.

Choose good names for things

```
batches_raw_v1.rds                geneexpr_mlratio_gastroc_v2.rds
batches_v1.rds                    geneexpr_mlratio_hypo_v1.rds
clinical_cpl_v2.rds               geneexpr_mlratio_hypo_v2.rds
clinical_d2o_v2.rds               geneexpr_mlratio_islet_v2.rds
clinical_lipomics_v4.rds          geneexpr_mlratio_kidney_v2.rds
clinical_liverTG_v2.rds           geneexpr_mlratio_liver_v2.rds
clinical_mirna_v2.rds             geneexpr_mlratio_nqrank_adipose_v2.rds
clinical_necropsy_v4.rds          geneexpr_mlratio_nqrank_gastroc_v2.rds
clinical_plasmaurine_v3.rds       geneexpr_mlratio_nqrank_hypo_v1.rds
clinical_rbm_v2.rds               geneexpr_mlratio_nqrank_hypo_v2.rds
Deprecated/                       geneexpr_mlratio_nqrank_islet_v2.rds
geneexpr_int1_adipose_v2.rds      geneexpr_mlratio_nqrank_kidney_v2.rds
geneexpr_int1_gastroc_v2.rds      geneexpr_mlratio_nqrank_liver_v2.rds
geneexpr_int1_hypo_v2.rds         geneexpr_omit_hypo.rds
geneexpr_int1_islet_v2.rds        geneexpr_prcomp_adipose_v2.rds
geneexpr_int1_kidney_v2.rds       geneexpr_prcomp_gastroc_v2.rds
geneexpr_int1_liver_v2.rds        geneexpr_prcomp_hypo_v2.rds
geneexpr_int2_adipose_v2.rds      geneexpr_prcomp_islet_v2.rds
geneexpr_int2_gastroc_v2.rds      geneexpr_prcomp_kidney_v2.rds
geneexpr_int2_hypo_v2.rds         geneexpr_prcomp_liver_v2.rds
geneexpr_int2_islet_v2.rds        geno_aligned_w_pmap.rds
geneexpr_int2_kidney_v2.rds       geno_pmark.rds
geneexpr_int2_liver_v2.rds        ReadMe.txt
geneexpr_mlratio_adipose_v2.rds
```


This is the same set of files, renamed. Using `clinical_` and `geneexpr_` brings similar files together.

A lot of files, but less forbidding.

Document your work

- What is all of this stuff?
- What was your analysis process?

→ `ReadMe` files

An overall `ReadMe` file plus an additional such file in each directory.

Well-named files and directories makes everything easier.

Also, keep the documentation current. There's nothing worse than documentation that is out of date and doesn't match the contents.

Organizing data in spreadsheets

	A	B	C	D	E	F	G
1	1MIN						
2			Normal			Mutant	
3	B6	146.6	138.6	155.6	166	179.3	186.9
4	BTBR	245.7	240	243.1	177.8	171.6	188.1
5							
6	5MIN						
7			Normal			Mutant	
8	B6	333.6	353.6	408.8	450.6	474.4	423.8
9	BTBR	514.4	610.6	597.9	412.1	447.4	446.5

How you organize your data within files can have a big impact on how easy they are to work with.

You can probably figure out what the numbers mean here, particularly if I tell you that there were triplicate measurementss under two treatments (1 min or 5 min) of cells that were either normal or mutant and cam from mouse strains B6 or BTBR.

But it's hard to tell a computer program about the data structure here.

Organizing data in spreadsheets

	A	B	C	D
1	ttt_min	strain	mutation	response
2	1	B6	normal	146.6
3	1	B6	normal	138.6
4	1	B6	normal	155.6
5	1	B6	mutant	166
6	1	B6	mutant	179.3
7	1	B6	mutant	186.9
8	1	BTBR	normal	245.7
9	1	BTBR	normal	240
10	1	BTBR	normal	243.1
11	1	BTBR	mutant	177.8
12	1	BTBR	mutant	171.6
13	1	BTBR	mutant	188.1
14	5	B6	normal	333.6
15	5	B6	normal	353.6

This is the first few rows of a reorganized version of the data, as a rectangle where the rows are individual measurements and the columns are variables.

This is maybe less pretty, but it's much easier to work with.

Organizing data in spreadsheets

- Make it a rectangle
- Individual measurements as rows; variables as columns
- Single header row
- One item per cell
- No empty cells
- No calculations in the raw data
- No highlighting or coloring as data

Broman and Woo (2018) Am Stat 72:2-10
doi.org/gdz6cm

Here are some key principles for organizing data in spreadsheets: make a rectangle with a single header row.

Never do calculations in your raw data file. If you're doing analyses or making charts in Excel, do so in a copy of the data file. Every time you open the raw data file, there's a risk that you'll mess things up.

“What the heck is ‘`FAD_NAD SI 8.3_3.3G`’?”

Sometimes the columns in your data files have meaning only to you.

If the data analyst can’t connect to the measurements, they’re just columns of numbers.

Metadata

- Create a data dictionary
  - Explain each column
  - Include different versions of the variable names (compact vs descriptive)
  - Units
  - Allowable values
- The metadata are data
  - Make it a rectangle

Clear metadata is critical for others to be able to understand your data. In particular, make a data dictionary that describes the variables. In addition to a description of each column, I like to have short and longer versions of the names for use in data visualizations, as the column names themselves can be cryptic.

These metadata are data, and so rather than make a Word documention describing the data, I personally would prefer to have another data file with the metadata.

Data dictionary

	A	B	C	D
1	name	plot_name	group	description
2	mouse	Mouse	demographic	Animal identifier
3	sex	Sex	demographic	Male (M) or Female (F)
4	sac_date	Date of sac	demographic	Date mouse was sacrificed
5	partial_inflation	Partial inflation	clinical	Indicates if mouse showed partial pancreatic inflation
6	coat_color	Coat color	demographic	Coat color, by visual inspection
7	crumblers	Crumblers	clinical	Indicates if mouse stored food in their bedding
8	diet_days	Days on diet	clinical	Number of days on high−fat diet

Here's an example data dictionary. You might also include units and informationa about possible valid values.

Everything with a script

If you do something once,
you'll do it 1000 times.

The most basic principle for reproducible research is: do everything via code.

Downloading data from the web, converting an Excel file to CSV, renaming columns/variables, omitting bad samples or data points...do all of this with scripts.

You may be tempted to open up a data file and hand-edit. But if you get a revised version of that file, you'll need to do it again. And it'll be harder to figure out what it was that you did.

Some things are more cumbersome via code, but in the long run you'll save time.

Reproducible reports

Gough project diagnostics

Karl Broman, 3 March 2014

Combine genotypes and phenotypes

I've combined the initial genotypes (using the re-clustered genotypes for plates 14-16) with the well-behaved portion of the re-run genotypes. I'm focusing on 36813 markers that are informative (though, as we'll see, there are still a lot of badly behaved and basically non-informative markers that need to be removed). I've combined data on replicate samples, to give one set of genotype calls for each sample.

There are 1497 genotyped mice and 1464 phenotyped mice. All of the mice in the phenotype data have genotypes, but there are 33 genotyped mice with no phenotypes, including 3 Gough mice and 30 F2 progeny.

I love R Markdown for making reproducible reports that document the full details of my analysis. R Markdown mixes Markdown (for light-weight markup of text) and R code chunks; when processed with knitr, the R code is executed and results inserted into the final document.

With these informal reports, I seek to fully capture the entirety of my data explorations and decisions.

Python people should look at Jupyter notebooks.

Automate the process (GNU Make)

```
R/analysis.html: R/analysis.Rmd Data/cleandata.csv
    cd R;R -e "rmarkdown::render('analysis.Rmd')"

Data/cleandata.csv: R/prepData.R RawData/rawdata.csv
    cd R;R CMD BATCH prepData.R

RawData/rawdata.csv: Python/xls2csv.py RawData/rawdata.xls
    Python/xls2csv.py RawData/rawdata.xls > RawData/rawdata.csv
```


GNU Make is an old (and rather quirky) tool for automating the process of building computer programs. But it's useful much more broadly, and I find it valuable for automating the full process of data file manipulation, data cleaning, and analysis.

In addition to automating a complex process, it also documents the process, including the dependencies among data files and scripts.

Fancier example

```
FIG_DIR = Figs

mypaper.pdf: mypaper.tex ${FIG_DIR}/fig1.pdf ${FIG_DIR}/fig2.pdf
    pdflatex mypaper

# One line for both figures
${FIG_DIR}/%.pdf: R/%.R
    cd R;R CMD BATCH $(<F)

# Use "make clean" to remove the PDFs
clean:
    rm *.pdf Figs/*.pdf
```


As I said, you can get really fancy with GNU Make.

Use variables for directory names or compiler flags. (This example is not a good one.)

Use pattern rules and automatic variables to avoid repeating yourself. With `%`, we have one line covering both `fig1.pdf` and `fig2.pdf`. The `$(<F)` is the file part of the first dependency.

Look at the manual for make and the many online tutorials, such as the one from Software Carpentry.

How do you use make?

- If you name your make file `Makefile`, then just go into the directory containing that file and type `make`
- If you name your make file `something.else`, then type
  `make -f something.else`
- Actually, the commands above will build the first target listed in the make file. So I'll often include something like the following.
  `all: target1 target2 target3`

  Then typing `make all` (or just `make`, if `all` is listed first in the file) will build all of those things.
- To be build a specific target, type `make target`. For example, `make Figs/fig1.pdf`

I can't believe that I forgot to explain this the first time I gave this lecture.

Write modular code

- Modular code is easier to understand, maintain, and reuse.
- Turn repeated code into functions
- Combine useful functions into a package or module

Another important step towards reproducibility is to revise your code to make it more clear.

The single most important step towards clear code is to pull out complex or repeated code as a separate function. This makes your code easier to read and maintain.

Next, combine those functions together into a package or module. It's surprisingly easy to create an R package (see `https://kbroman.org/pkg_primer`) and it's even easier to make a Python module.

When writing functions, try to write them in a somewhat-general way and then pull them out of the project as separate package or module, so that you (and/or others) may reuse them for other purposes.

Keeping track of versions

- Google drive / Dropbox / Box
- Version numbers in file names
- Formal version control (e.g., git/GitHub)
  - Browse changes
  - Try new things without fear of breaking what works
  - Jump to the state of the project at any time point
  - Merge simultaneous changes from multiple people

We all struggle to keep track of versions of things.

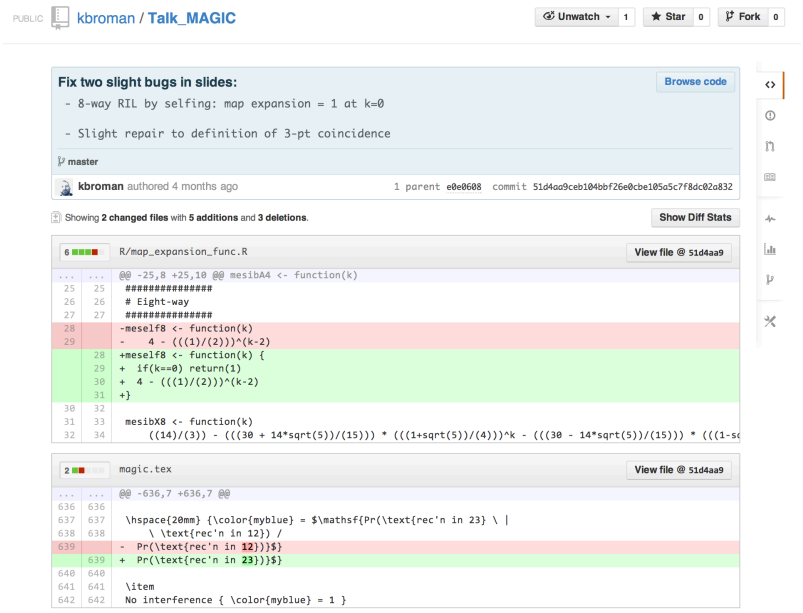
Shared drives (like google drive, dropbox, and box) often keep track of past versions, but usually there's a time limit (like 30 days or a year).

You can make copies of file with a version number appended to the name. You might zip up a directory and include the date in the zipped file.

Formal version control has a number of advantages, including easy of browsing the history or jumping to a particular time point. The ability to merge simultaneous changes from multiple users is a key advantage.

git can be hard to learn; it's designed for pretty hard-core programmers. But there are growing learning resources, and the long-term payoff is considerable. For collaborative projects, the payoff is immediate.

Version control (git/GitHub)



git has a steep learning curve, but ultimately I think you'll find it really helpful.

The big selling point is in collaboration: merging changes from collaborators, and keep your work synchronized.

Longer term, there's great value in having the entire history of changes to your project. If something stops working, you can go back to any point in that history to see when it stopped working and why.

With git, you can also work on new features or analyses without fear of breaking the parts that are currently working well.

Backups

- Multiple places, including off-site
- Automatic

I can't emphasize enough the importance of backups. And you must have a copy off-site. And if it's not automatic, it won't happen.

License your software

Pick a license, any license

– Jeff Atwood

If you don't pick a license for your software, no one else can use it.

So if you want to distribute your code so that others can reproduce your analyses, you need to pick a license, any license.

I choose between the MIT license and the GPL.

Don't use the Creative Commons licenses for code. But feel free to use them for other things.

Share your stuff

- Code
  - GitHub / BitBucket
  - Zenodo (archival, with DOIs)
- Data
  - Domain-specific repository (e.g., dbGAP)
  - General repository (e.g., github, figshare, zenodo, datadryad)
  - Institutional repository

A reproducible workflow is valuable even if you don't intend to share your work with others.

But if do want to share, it's best to place things at a third-party site. Ideally one that can be trusted as an archive and that provides DOIs.

Place code at GitHub (or the similar site, BitBucket). The only problem is that it can't necessarily be trusted to still be there 5 years from now. There's an easy way to have "releases" archived at zenodo.org automatically, with a DOI. So I recommend that.

For data, it's probably best to use a domain-specific repository, if there is an appropriate one. Otherwise, general repositories github, figshare, zenodo, or datadryad. Again, github is not ideal because it's not archival and doesn't give DOIs.

Summary

1. Organize your project
2. Choose good names for things
3. Document what's what
4. Organize data as a rectangle
5. Metadata is data
6. Everything with a script
7. Even better: reproducible reports
8. Automate the process (GNU Make)
9. Write modular code (functions and packages)
10. Use version control (git/GitHub)
11. License your software
12. Share your data and code

Summaries are always good.

Again, don't try to change everything at once. Reproducibility can be surprisingly hard and requires a daily commitment. And here I'm just thinking about a project with a single data analyst. A collaboration with multiple analysts is yet harder.