rrr_nb.org

February 10, 2013

Contents

1 About

This is a "notebook" for my work on Really Reproducible Research (RRR). So it's basically a complex scratch pad for working through ideas and assembling materials for the more structured documents that are intended to be shared.

2 Thinking

2.1 Two projects

So there are two sides to this thing. On one side, the "theory." A conceptual framework for discussing reproducibility, and getting a handle on some goals and ideas for making it work (and what it means for it to work). On the other side, an actual system in order to realize the concepts.

I'm not sure about the format. Initially I was thinking of a big book, with Part I being the theory and Part II the implementation, but I'm less sure about that now. I mean, I think the theory is something that could be written up as a pretty traditional article or short book. Maybe. Still struggling a little with the structure, but I could imagine it working fine, once I decide on how I want to organize the chapters/sections.

But I'm starting to re-think Part II. One of the goals needs to be to keep the cost of entry low. And the problem is that people will come to this with different experiences, different habits, and different needs. Some linguists may not do any statistical analysis at all. Some psycholinguists will be interested in starting with the stats stuff, and would benefit if that's all they worked on. Additionally, I'm thinking that for the implementation stuff, I will minimally need three "pieces" for each tool/lesson: (1) motivation (*why* is this a good thing?), (2) "walkthrough" tutorials (*how* do you do it?), and (3) reference material (for people to refer back to, after completing the walkthroughs). Putting all that in one monster PDF may not be the best thing, especially if I want to try a more interactive format for the walkthroughs.

However, I do want the ability to link back and forth. I would like a flow from "why is X important?" to "how do I do X?" and back again. Maybe just a series of linked PDFs would be good.

3 Initial workflow ideas

3.1 Overview summary

Here's a quick summary of all the components:

VCS base Use `git` or some other VCS to track changes, record commits, etc. The main points are to:

- encourage a "literate" workflow, iterating work and (brief) commenting
- enable all the powerful `git` features such as looking back to earlier stages, branching for "safe" development, diffs, collaboration, etc.
- help keep your (virtual) workspaces relatively organized, without explosion of files

Notebook Use `org-mode` to create a Reproducible Research Notebook (RRNB). The main points are:

- Simple text format (easy to diff, portable, etc.) with lots of "fancy" features if you need them.
- Can be "one-stop shopping" for a research project, combining:
  - lit reviews with bibliographic (BibTeX) info
  - links to papers, websites, other docs, etc.

  - notes and ideas about theory, research questions, etc., with links to other parts of the doc as needed/desired
  - code blocks for *any* language, used for data collection, analysis, etc., which can be used actively (i.e., run within the notebook) or "passively" (tangled to an external file to be run).
  - tables, results, raw data, etc. (either links, or in the nifty org-mode table format)
  - complete drafts of papers (either within same doc, or more likely in separate (linked) doc), which can be exported to LaTeX, OpenOffice, etc.
  - TODOs and other functionality for managing a project
- Does not *have* to be the end-all, but minimally useful as a place for notes, with links to other files
- **Scales** in complexity as needed, but does not **require** a complex set-up

Lit review Some processes in the notebook for making a lit review as "reproducible" as possible.

- Reproduce lit searches (when, what terms, what engines?)
- Reproduce lit commentary (take notes on a paper when reading)
- Maintain bibliographical info (create `bibtex` chunks in RRNB, can export at will as needed, never re-type biblio info again!!!). Can use text searching to search in reverse (e.g., find paper called chomsky2001on in PDF form in a project folder or centralized "research library" folder, can do text searches for "chomsky2001on" through all .org files, etc.)

Data collection

- Take notes on data collection
- Keep notes in central notebook, or linked file
- Keep code for collecting data in code block or linked file
- Use table format to keep simple spreadsheet-style notes, like dates of data collection, etc.

Data analysis

Collaboration

Writing papers

3.2 Base

3.2.1 What?

The base of the workflow is "track and document changes." This means using a VCS (version control system/software). I will use git.

The idea here is that at many point along the way in developing any documents or files related to research, changes are tracked, documentation is added, and a history of development is created over time. The VCS can use that history to "rewind the tape," compare versions, and manage collaboration.

3.2.2 Why?

Philosophical Research involves revisions. Lots and lots of changes and revisions. Sometimes those revisions are in the "right" direction, but sometimes, an earlier version of a document (theory/code/stimuli/etc.) or an "alternative" version (if a branching model of development is used) is better. A good VCS enables movement along this history of revisions and changes in order to facilitate the best possible research.

Ethical A good VCS promotes transparency, and may even do a good job of tracking contributions of collaborators, which can help in the distribution of credit (authorship, etc.). But it also improves efficiency when older revisions need to be recovered, and thus can represent signficant savings in time (and money).

Practical See this link and links therein for some other discussion, as well as this classic by Kieran Healy. I'll paraphrase and expand a little here.

- You have access to all older revisions without an explosion of (often confusingly-named) files.
- It's a universal "track changes" mode.
- With a good VCS, you can easily compare ("diff") versions to see exactly what changed and when.
- If you back up your VCS in the cloud or some other method, you have *backed-up* access to all your versions and revisions, which is better than just a single backed-up "final draft."
- You can "try out" some different analyses or development angles, and switch back or combine them later.
- You can facilitate collaboration (assuming you get your collaborators to work with the system, or assuming you can manage it for your collaborators).
- The habit of logging ("committing") changes and documenting them should build more efficient and reproducible work habits.
- If you maintain your VCS, you can return to the project months or years later, and not have to wonder about where your most up-to-date files are, or where that initial version went.

3.2.3 How?

Basic principles

- The system must be "lightweight" – easy to use, and easy to implement, without interrupting ACTUAL WORK. In other words, if it gets in the way, it's not helpful.
- The system needs to be robust against "error." If you have to everything **just right**, it's too burdensome. The point is to help guard against error, not screw up all your files and create more work for you.

Implementation

- Use git
  - ONLY work on projects that are git repos. If you are working on something that's not utterly "disposable," make a git repo (`git init`) and track changes.
  - Any time you **stop** working, you should commit! This is maybe the hardest, since sometimes you get interrupted. So this is an important habit. Ideally, committing changes at least once at the end of the day is best. BUT even if you only commit changes once a week, it's a big improvement on not using a VCS! So I think in practice, you need to commit changes on a regular-enough schedule that corresponds to "significant additions," which may be defined differently person to person. Committing after every sentence in a draft is obviously overkill. Use `git log` and `git status` to check on what has uncommitted changes, and when changes were last committed.

  - Don't proliferate the repo unnecessarily. I mean, trust the VCS, and use `git branch`, instead of cloning the repo to some other place on your machine and working on it there. This is defeating the point!
  - Make sure you know where the up-to-date repo is. If you can use GitHub (either because you don't mind your work being public, or because you are paying for a private repo), then pushing changes to GitHub is a great way to know exactly where your most up-to-date repo is! Otherwise, you can use `git log` or `merge` to get things back together.

- When you can't use git:
  - Act as if you are!
  - When you would normally commit changes, just create a little commit message.
  - When you can get access to git again (hopefully once a day, or nearly so), add the new files/changes, and use the commit message to update the repo.
  - This requires a little more vigilance in creating the messages, though, because it's easier to lose track of changes. For example, if you forget to comment that you changed file X, you might forget to add it to the git repo until much later. But if you minimally keep a good record of which files you change/add, it's not that big a deal.

Okay, so taking these points into consideration, here are my working rules:

1. When starting work:
   (a) Start git anytime you do ANY work, and if it's not a repo already, use `git init`
   (b) Before making new changes, run `git status` and consider committing any uncommitted changes.
   (c) If you **can't** use git, start a "COMMIT" file to log changes as you go, keeping track of ALL files that you add/change.
2. Work! (forget about git while you work)
3. When you hit a stopping point, and some significant progress has been made, `git add .` and `git commit`, or just `git commit -a`. This will pop up an editor to write the commit message. Or use `git commit -m` plus a message in quote to add a short message.
4. If you are in the middle of something AND you know you will come back to it soon, AND you are not worried about losing what you did, you can put off the commit until later.
5. Make a rule about where the repo should "live." If not on GitHub, and you use multiple machines, make a rule about which machine should always have the most up-to-date repo. Collaboration is different. "Self-collaboration" can be confusing. DON'T make copies of the repo on your own machine (do branching instead).
6. Use `.gitignore` file to simplify commits. Avoid "manual ignore" by leaving a bunch of files untracked but still visible to git.

That's pretty simple, really.

3.2.4 Upshot

This is the "base" because:

- it involves minimal "new stuff". You work the same as you always do, with whatever programs you want.
- you're just building an additional habit of logging that work, and committing changes to a VCS.
- this very simple use of git is easy to learn, and git is easy to install, and cross-plaform.
- it will insidiously introduce you to the value of simple text, when you get to the point of wanting to diff commits.
- you can start doing this in an hour or less.

3.3 Notebook

3.3.1 What?

The next big step is to dive into the idea of keeping a "reproducible research notebook" (RRNB). This is the core of the system I'm imagining, because it's a flexible way to start folding more and more activities into a reproducible format.

3.3.2 Why?

Philosophical This is all about recording the scientific process, and making it easier and easier for other people to reproduce and replicate your work. These are big things! The notebook is what keeps it all in one place.

Ethical Transparency, transparency, transparency!

Practical The idea is one-stop shopping for all of your work. You put your ideas into it. You put your literature and comments/notes on papers into it. You put your code into it. You might even put your data into it, but you can at least put links to your data into it. You can also manage TODOs (using org-mode) and create a full working environment within the notebook. How did you run that analysis from 3 papers ago? Right there in the notebook (or follow a link). Where did you see the idea for this analysis? Link to a website/paper/whatever. Could you share some papers where someone can read about X? Tangle out the BibTeX info, or export a whole section with all your notes as well. And it's all simple text!

3.3.3 How?

Emacs and Org-mode!

3.3.4 Upshot

4 My working system

4.1 Start working

1. Find (or create) task in .org file (life, or some other agenda file)
2. Clock in (`C-c C-x C-i`)

3. Start `git` bash, `cd` to project directory
4. Run `git status` to make sure there are no currently loose threads

4.2 Thinking

1. Use RRNB as a place to make notes, sketch out ideas, etc.
2. Use `git` and `org` links to keep the place clean. If you need to scrap something and clean-up, either:
   (a) use commits to make sure you can "rewind" to the earlier version, or
   (b) archive the scrapped stuff to another place, and add a link if you don't want to forget about it
   (c) **stop keeping infinite iterations of ideas around like a packrat!!!**

4.3 Lit review

4.4 Analysis

4.5 Writing

4.6 Interruptions

1. Use `org-capture` (in my `.emacs`, set as `C-c r`) to jump to an "interruption" task in `org-mode`
2. This automatically handles clock-out and clock-in
3. When done, `C-c C-c` files interruption in `notes.org`, and automatically clocks you back in to previous task!
4. It also returns you to your *complete window arrangement* of whatever you were doing before! This is ridiculously awesome, and yet another reason to do everything in Emacs ;-)

4.7 Wrap-up

1. Save all changes
2. Use `git status` to check on any uncommitted changes
3. Use `git add` and `git commit` (or, if file is already being tracked, `git commit -a` as shortcut) to add and commit changes
4. Clock out (`C-c C-x C-o`)