Project organization

David Benkeser, PhD MPH

Emory University

Department of Biostatistics and Bioinformatics

INFO550

bit.ly/info550

Additional reading

File organization and naming are powerful weapons against chaos.

Jenny Bryan

Basic principles

- Put everything in one version-controlled directory.
- Develop your own system.
- Be consistent, but look for ways to improve.
  - naming conventions, file structure, `make` structure
- Raw data are sacred. Keep them separate from everything else.
- Separate code and data.
- Use `make` files and/or READMEs to document dependencies.
- No spaces in file names.
- Use meaningful file names.
- Use YYYY-MM-DD date formatting.
- No absolute paths.
- Use a package management system.

You mostly collaborate with yourself, and me-from-two-months-ago never responds to email.

Karen Cranston

What to organize?

It is probably useful to have a system for organizing:

- data analysis projects;
- first-author papers;
- talks.

The systems should adhere to the same general principles, but different requirements may necessitate different structures.

Think about organization of a project from the outset!

Collaborative projects

Collaborative projects present a greater challenge.

- Not everyone is comfortable with LaTeX or git or ...

I don't have a great solution for this.

- Google drive/Word online helps to a certain extent, but you lose in other areas (reference management, math typesetting)
- Overleaf has gotten much better for LaTeX

Some advice:

- Address organization from the outset.
- Ideally, bring people on board to your (version controlled, reproducible) system.
- Keep open lines of communication (especially if using GitHub)

Example data analysis project

```
analysis/
    raw_data/
    data/
    R/
      R/00_clean_data.R
      R/01_fit_models.R
      R/02_make_figures.R
      R/03_summarize_results.R
      R/04_report.Rmd
    figs/
    sandbox/
      sandbox/exploratory.R
    ref_papers/
    Makefile
    README.md
    renv
```

Example paper organization

```
paper/
    analysis/
      analysis/README.md
      analysis/00_clean_data.R
      analysis/01_fit_models.R
      analysis/02_make_figures.R
      analysis/sandbox
    sim/
      sim/README.md
      sim/helper_functions.R
      sim/sim_script.R
      sim/run_sim_script.sh
      sim/sandbox
    figs/
    notes/
    ref_papers/
    submitted/
    revision/
    final/
    README.md
    Makefile
    my_paper.tex
    my_refs.bib
```

Example class organization

See the class GitHub repository!

Organizing data

Raw data are sacred... but may be a mess.

- You'll be surprised (and disheartened) by how many color-coded excel sheets you'll get in your life.

Tempting to edit raw data by hand. **Don't!**

- Everything scripted!

Use meta-data files to describe raw and cleaned data.

- structure as data (e.g., `.csv` so easy to read)

Organizing data

Hadley Wickham defined the notion of tidy data.

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

ptid	day	age	drug	out
1	1	28	0	0
1	2	28	0	1
2	1	65	0	0
2	2	65	1	1
3	1	34	0	0
3	2	34	-	1

Exploring data

One of the first things we'll often do is open the data and start poking around.

- Could be informal, "getting to know you."
- Could be more formal, "see if anything looks interesting."

This is often done in an ad-hoc way:

- entering commands directly into R;
- making and saving plots "by hand";
- etc...

Slow down and document.

- Your future self will thank you!

Exploring data

Other helpful ideas for formalizing exploratory data analysis:

- `.Rhistory` files
  - all the commands used in an R session
- Informal `.Rmd` documents.
  - easy way to organize code/comments into readable format
- `save` intermediate objects and workspaces
  - and document what they contain!
- `knitr::spin`
  - writing `.R` scripts with rendered-able comments

The `here` package

No absolute paths.

- Absolute paths are the enemy of project reproducibility.

For `R` projects, the `here` package provides a simple way to use relative file paths.

- Read Jenny Bryan and James Hester's chapter on project-oriented work-flows.

The use of `here` is dead-simple and best illustrated by example.

The `here` package

Consider this simple project structure.

```
my_project/
    data/
      my_data.csv
    output/
    R/
      R/my_analysis.R
    Rmd/
      Rmd/my_report.Rmd
```

Here, the folder `my_project` is the **root directory**.

- Where `.git` lives
- All file paths should be **relative** to `my_project`!

The `here` package

Each `R` script or `Rmd` report, should contain a call to `here::i_am('path/to/this/file')` at the top.

- `path/to/this/file` should be replaced with the path **relative** to the project's **root directory**.
- `here::i_am` means use function `i_am` from `here` package.

For example, the file `R/my_analysis.R` might look like this.

```
# include at top of script
here::i_am('R/my_analysis.R')

# now add all your great R code...
```

The `here` package

`Rmd/my_report.Rmd` should include an `R` chunk that calls `i_am`.

````
---
output: html_document
---

```{r}
here::i_am('Rmd/my_report.Rmd')
```

<!-- Now the rest of your Rmd code -->
````

The `here` package

The call to `i_am` establishes the **root directory**.

- Subsequent file paths can be made using the `here` function.

For example, `my_analysis.R` might look like this.

```
# include at top of script
here::i_am('R/my_analysis.R')

# load data
my_data <- read.csv(here::here('data', 'my_data.csv'))

# do some analysis to get results

# save results
save(my_results, file = here::here('output', 'my_results.RData'))
```

The `renv` package

Use a package management system.

To increase reproducibility of a project, we must keep track of what packages are used.

- Want to avoid chasing down 100 errors like this:

```
Error in library(ggplot2) : there is no package called
'ggplot2'
```

The `renv` package is useful to this end.

The `renv` package

Download the `example_project` folder.

```
example_project/
    Makefile
    figs/
    R/
      R/barchart.R
    Rmd/
      Rmd/report.Rmd
```

If you scan the code, you will see we need **three R packages**.

- `here`, `wesanderson`, `knitr`, and `rmarkdown`

The `renv` package

Open an `R` session from the `example_project` folder.

- Install the `renv` package if needed.

Initialize the project by running the following command.

```
renv::init()
```

You will see a lot of output. What just happened??

`renv.lock` file

You should now see **lockfile**, `example_project/renv.lock`.

```
{
  "R": {
    "Version": "4.0.3",
    "Repositories": [
      {
        "Name": "CRAN",
        "URL": "https://cloud.r-project.org"
      }
    ]
  },
  "Packages": {
    "base64enc": {
      "Package": "base64enc",
      "Version": "0.1-3",
      "Source": "Repository",
      "Repository": "CRAN",
      "Hash": "543776ae6848fde2f48ff3816d0628bc"
    },
  ...
```

`renv.lock` file

This **lockfile** records all of the information about packages needed by your project.

- Version of package, where was it installed from, etc...

How does it know?

- `renv` scans all files in your project directory.
- Looks for `library`, `require`, or `package::function`.

`.Rprofile`

We also see `example_project/.Rprofile`, containing one line.

```
source("renv/activate.R")
```

We have not talked about `.Rprofile`'s because they are generally antithetical to reproducible research.

When `R` starts, it searchers for `.Rprofile` and runs what it finds.

- Use this to change various options, always load packages, etc...

In this case, whenever we start `R` in `example_project`, we **activate** our `R` environment.

- Telling `R` where packages for this project are saved, etc...
- Details are not too important.

`renv` folder

You will also see a folder `example_project/renv/`.

- This folder contains your **project library**.

`renv` tries to be clever about installing packages.

- Already have a package installed elsewhere? `renv` will link to it.
- Otherwise, package is installed in `renv/library`.

Note that `renv/.gitignore` ensures packages not put under version control.

Activating `renv`

`renv` will automatically be active in any `R` session that is run from the `example_project` directory.

- Recall the presence and function of `.Rprofile`.

To activate in an `R` session run from elsewhere: `renv::activate('path/to/renv')`.

For using `renv` with R Markdown projects, this is important!

- Either need to `activate` in a code chunk.
- Or use `knitr::opts_knit$set(root.dir = here::here())` (assuming the project root contains your `renv` project library).

Collaborating with `renv`

A typical `renv` collaborative workflow on GitHub:

- **User A** initializes the lockfile using `renv::init()`.
- **User A** commits `renv.lock`, `.Rprofile`, and `renv/activate.R` and pushes to GitHub.
- **User B** pulls from GitHub, opens `R`, and uses `renv::restore()` to synchronize their local project directory.
- **User B** adds new packages to code, uses `renv::snapshot()` to record changes to `renv.lock`
- **User B** commits `renv.lock` and pushes to GitHub.
- **User A** pulls from GitHub, opens `R`, and uses `renv::restore()` to synchronize their local project directory.
- ...

Breakout exercise

With a partner, choose **User A** and **User B**.

User A

- initialize the lockfile for `example_project`
  - open `R` in `example_project` and run `renv::init()`
- use `git init` to initialize version control of `example_project`
- create a GitHub repository for `example_project`
- add the GitHub repository as a remote to your local repository
  - `git remote add origin https://github.com/usera/example_project.git`
  - `git remote add origin git@github.com:usera/example_project.git`
- commit all files locally and push to repository

Breakout exercise

User B

- create a fork of **User A**'s repository
- `git clone` the fork to your local machine
- `cd` into `example_project` and open `R`
- run `renv::restore()` to synchronize package library
- confirm that you can build the report
  - run `make report` from `example_project` directory
  - open `Rmd/report.html` to confirm correct build

Breakout exercise

User B now wants to change the colors of the graph

- open `R` in `example_project` directory
- run `renv::remove('wesanderson')` to remove the `wesanderson` package from the lockfile
- replace lines 4-5 of `barchart.R` with the following and save

```
library(RColorBrewer)
colors <- brewer.pal(3, "Dark2")
```

- open `R` in `example_project` and run `renv::status()`
- if prompted, run `install.packages("RColorBrewer")`
- run `renv::snapshot()` to add `RColorBrewer` to the lockfile
- commit changes and push to your fork
- submit a PR to User A's repository

Breakout exercise

User A

- add a remote linking to user B's repository
  - `git remote add userb https://github.com/userb/example_project.git`
- `fetch` User B's `master` branch
  - `git fetch userb master`
- `checkout` User B's `master` branch
  - `git checkout remotes/userb/master`

Breakout exercise

User A

- `cd` into `example_project` and open `R`
- run `renv::restore()` to synchronize your package library
- confirm that you can build the report
  - run `make report` from `example_project` directory
  - open `Rmd/report.html` to confirm correct build

Breakout exercise

User A

- if the report builds correctly, create a local branch named `userb` from User B's `master` branch
  - `git checkout -b userb`
- `merge` User B's `master` into your `master`
  - `git checkout master`
  - `git merge userb`
- close the PR by pushing to GitHub
  - `git push origin master`

Breakout exercise

User A has changed their mind about the colors!

- open `R` in `example_project` directory
- run `renv::remove('RColorBrewer')` to remove `RColorBrewer` from the lockfile
- replace lines 4-5 of `barchart.R` with the following and save

```
colors <- c("red", "blue", "green")
```

- open `R` and run `renv::status()` and `renv::snapshot()`
- confirm that you can build the report
  - run `make report` from `example_project` directory
  - open `Rmd/report.html` to confirm correct build
- if the report builds correctly, `commit` and `push`

Breakout exercise

User B

- add a remote linking to user A's repository
  - `git remote add usera https://github.com/usera/example_project.git`
- `fetch` User A's `master` branch
  - `git fetch usera master`
- `checkout` User A's `master` branch
  - `git checkout remotes/usera/master`

Breakout exercise

User B

- `cd` into `example_project` and open `R`
- run `renv::status()` and, if needed, `renv::restore()` to synchronize your package library
- confirm that you can build the report
  - run `make report` from `example_project` directory
  - open `Rmd/report.html` to confirm correct build

Breakout exercise

User B

- if the report builds correctly, create a local branch named `usera` from User A's `master` branch
  - `git checkout -b usera`
- `merge` User A's `master` into your `master`
  - `git checkout master`
  - `git merge usera`
- update your fork of the repository by pushing to GitHub
  - `git push origin master`