Writing code for a wet-lab audience

Lyuba V. Bozhilova

Department of Clinical Neurosciences and MRC Mitochondrial Biology Unit, University of Cambridge

An Open Science Workshop, 17 October 2022

R's biggest challenge is that most R users are not programmers.

Hadley Wickham, *Advanced R* (2019)

Why share code (with "non-coders")?

For my sake, but also for the benefit of others:

- so they can better understand the work, and
- so they can use it and build on it.

In my case, "others" often have limited experience and little training and support for writing code.

They can follow a software tutorial, but won't know what a pull request is.

What the past year has ~~taught~~ reminded me

Code is like maths: sometimes nothing is obvious and everything is scary.

We can choose to meet people where they are:

- Conventions are not universal.
- Shiny stuff can be intimidating.

Good code is good for everyone:

- Modular code improves understanding.
- Good coding style is not just for pros.

Nothing is ever obvious

Things that routinely give people a headache:

- OS incompatibility,
- not installing required libraries,
- not updating anything, ever,
- not running code in the right directory,
- not knowing how to read in data.

This stuff is for the README, or for a small set-up script. And always provide (and describe!) correctly formatted input data.

Docs and errors spook people

I used to find Stack Overflow confusing and unpleasant. Many of my colleagues still do.

Reading documentation and parsing error messages is a learnt skill, and your audience might not be there.

The right person to test your code should know a lot less about it:

- someone from a different field,
- a student… or a patient supervisor!

Plain scripts, not (just) notebooks

Most people do not know what to do with a .ipynb or .qmd file extension.

You should always assume they will not read the free-form text either.

If you are going to share a notebook, share a heavily commented plain script version of your code as well.

Inline documentation

If you have something to say, say it in the comments.

- Give an overview of the script at the start.
- Split scripts into sections.

```
#----- Parse data
#----- Calculate ABC
#----- Plot DEF
```

- Explain important steps within sections.

```
# To test GHI, we perform JKL.
```

- Copy relevant expected output, e.g. from `summary()` calls or tests.

Modular code

Modular code is not just more adaptable, it is also clearer.

- Make the code itself modular.
- Split code
    - across files,
    - across sections within a file, and
    - in subsections or paragraphs within sections.
- Save and reload intermediate states, within reason.

(Not) following a style guide

I like the tidyverse style guide, but this does not mean I follow it religiously.

- Number scripts in order of execution, e.g. `00-plot-setup.R`, `01-data-preproc.R`, ..., `05-figures.R`.
- Give output the same prefix as the script it comes from, e.g. `01-summary-stats.Rdata`, `05-fig2.jpg`.
- Include data types in variable names (e.g. `oocytes_df`).

To `sum(up)`

Code, like maths, can freak people out, and we should be aware of that when writing for a less experienced audience.

We can adapt how we present code and documentation to formats they are more comfortable with (e.g. scripts with inline comments).

A lot of good code practice helps everyone, regardless of experience (e.g. writing modular code with consistent style).

Thanks!

- Patrick Chinnery's Mitochondrial Genomics lab, especially
  - Tom Barton-Owen
  - Fei Gao
  - Malwina Prater
  - Shafiqur Rahman
- Laura Kremer (Karolinska Institutet)
- The Data Championship Programme

Bioinformatics @ Patrick Chinnery's lab
lb925@cam.ac.uk
GitHub & Twitter: @lbozhilova