



Utrecht University

Computational reproducibility

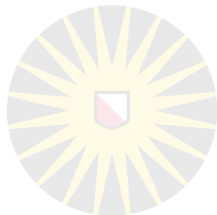
Best practices in writing scientific code

Barbara Vreede

b.m.i.vreede@uu.nl

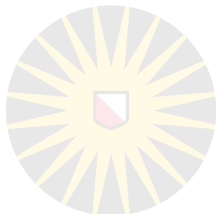
Motivation

What do you want to achieve?



Motivation

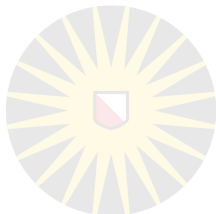
What do you want to achieve?



- My work is accessible.
- My work is reusable.
- My work is trustworthy.

Motivation

What is reproducibility?



- Not the same as replication!
 - We talk about **replication** when others go out and collect new data,
 - and about **reproducibility** when others analyze the same data.
- Requires same data, same code, same environment
- Provided this, it should yield same results.
- Is this enough?

It's all about accessibility!

... right?



Hello!

Welcome to the data repository for "Matthis, Yates, and Hayhoe (2018), Gaze and the visual control of foot placement in natural terrain" published in Current Biology in 2018.

FYI – the repository will also live here for the foreseeable future – Go here to browse the files without needing to download the whole giant zip file -

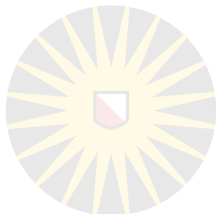
<https://www.dropbox.com/sh/zml8t0mrylu3ds2/AAAdzDivTUsEBrDNTUqgMrWra>

This repository contains the raw and post-processed data from that study, as well as the Matlab code necessary to generate all of the figures from the manuscript from the raw data.

If you are here looking for good, clean, pretty, and efficient code, then prepare yourself for disappointment. This code is a kludgely, ugly, inefficient mess. You remember that scene in *Mad Max : Fury Road* when the guy climbs under the truck with a wrench and tries to fix the engine while they are driving over rough roads and on fire? That's basically the way this code was written. I tried to leave a lot of comments, but still – Working with this code will probably make you hate me, and I am ok with that. It is probably riddled with problems, mistakes, bugs, inefficiencies, vestigial code stubs, etc etc. It is very far from perfect, but I am as confident as I am capable of being that all of the factual claims that were made in the manuscript are accurate.

Motivation

data + code + environment + ...?



- Trust & confidence in the code
- Trust & confidence in the workflow
- *Ability to apply the code!*
- In short: never forget the human who will work with your project.

Motivation

For whom?

- Your colleagues
- Your fellow researchers
- **Yourself** (in 6 months)



Source: [xkcd](#)

Managing your project

"Well begun is half done"



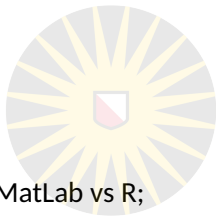
- Contain your project in a single recognizable folder
- Distinguish folder *types*, name them accordingly:
 - **Read-only**: data, metadata
 - **Human-generated**: code, paper, documentation
 - **Project-generated**: clean data, figures, models...
- Initialize a README file, document your project
- Choose a license
- **Publish your project** (more on that later!)

Example



```
•
|-- CITATION
|-- README
|-- LICENSE
|-- requirements.txt
|-- data
|   |-- birds_count_table.csv
|-- doc
|   |-- notebook.md
|   |-- manuscript.md
|   |-- changelog.txt
|-- results
|   |-- summarized_results.csv
|-- src
|   |-- sightings_analysis.py
|   |-- runall.py
```

Source: [Wilson et al., 2017](#)



A note on file formats and naming

- Minimize (!) the use of proprietary software (e.g. MatLab vs R; PhotoShop vs Inkscape)
- Make sure it is analysis-friendly and compatible! (e.g. .csv instead of .xls)
- Use a consistent naming scheme, with e.g. `sepa_rators` or CamelCase
- Write filenames from generic to specific, e.g. `ProjectDSX_cleaning_dataselection.py`
- That especially concerns dates: YEAR - MONTH - DATE! (e.g. 20190923)

‘Pick a license, any license’



- Copyright is implicit; others cannot use your code without your permission.
- Licensing gives that permission, and its boundaries and conditions.
- Choosing a license early on means being aware of your license as the project proceeds (and not creating conflicts).
- There are over 80 OSI-approved licenses (and many, many others) to choose from.

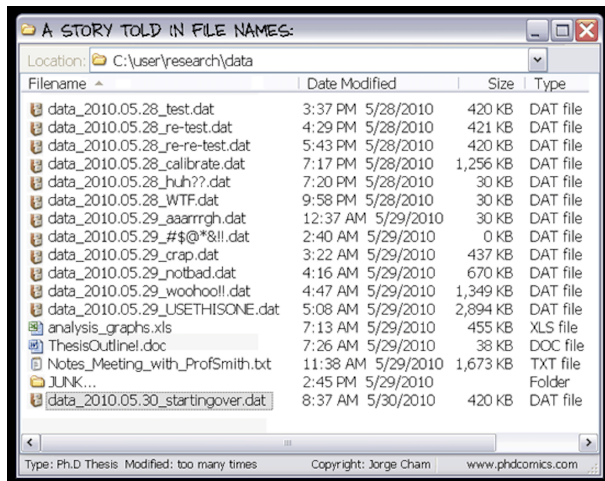
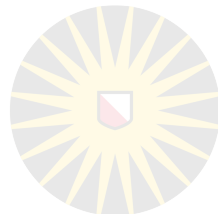
‘Pick a license, any license’



Licenses may specify whether:

- your code can be used commercially (as allows MIT)
- your code is compatible with differently licensed software (as allows MPL)
- derivative works should use a specific license (as says GPL)
- etc...

Does this look familiar?



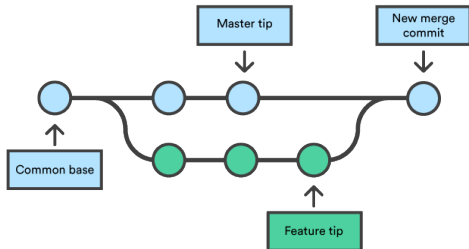
Source: [PhD comics](http://www.phdcomics.com)

Version control

Dedicated software can help manage your project

 **git** facilitates crucial project management:

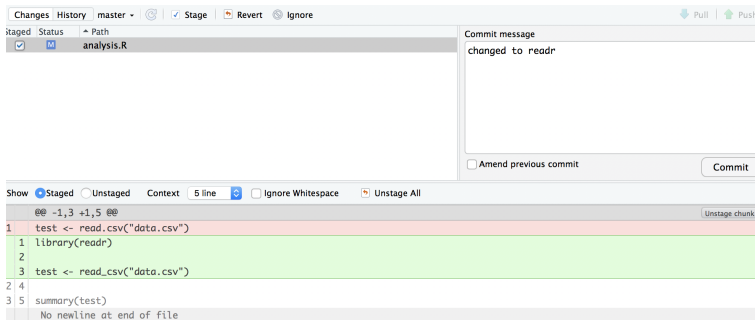
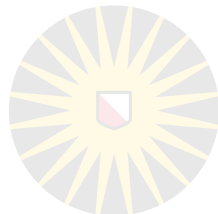
- Backing up different versions of a file
- Experimenting with alternative versions of a file
- Integrating working files and experimental files
- Collaborative coding/writing (more on that later!).



How to git?

Build git into your workflow!

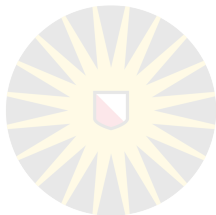
- Via the command line (hard core!)
- Using a **dedicated interface** (e.g. GitKraken)
- From your favorite coding editor (e.g. Rstudio)



Check out Daniël Lakens' detailed **tutorial** on combining Rstudio and Git!

Publishing a project

From git to open



- A git repository lives on your (local) drive
- But you can connect it to a remote repository
- This allows collaboration with co-owners of the project
- Or sharing with (and receiving suggestions from!) anyone in the world.

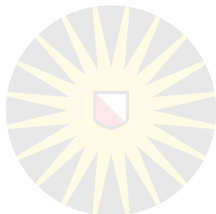


GitLab



GitHub

Public from the get-go?



Yes, this is encouraged!

- Forces you to consider readability throughout
- Allows collaboration and support, facilitates sharing

But it may not be for you:

- Sensitive data?
- Scoop-able code?
- You can still collaborate privately, and make your repository public when the time comes.
- It's all OK! (But think about it!)



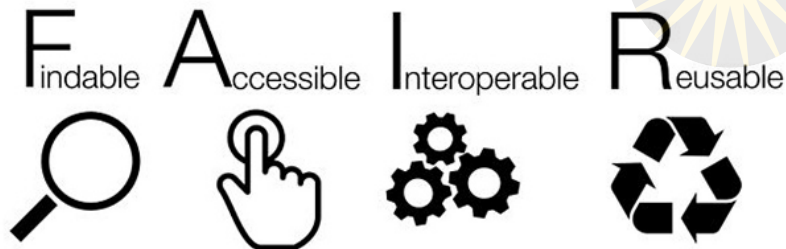
Archiving a project

Finding it a permanent home

- Github/Gitlab are excellent social and constructive platforms...
- ... but they are **not stable repositories**.
- (They're also not great for storing large (data) files...)
- Obtain a DOI for your project by archiving it.
 - Github connects to Zenodo to **archive a release** of your project.
 - Github/gitlab connects to the OSF to store the project as a whole, from where you can register it for a DOI. ([Here](#) is a good tutorial.)



FAIR data

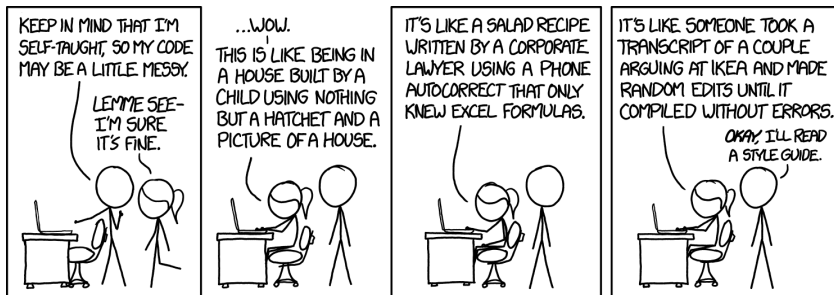


- Findable: descriptive metadata; persistent identifier
- Accessible: consent and risk management, open protocol
- Interoperable: (meta)data standards, documentation
- Reusable: permissively licensed

Best coding practices

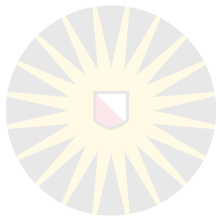
Your code should be...

- Readable
- Robust
- Reusable



Readable code

Good commenting practices

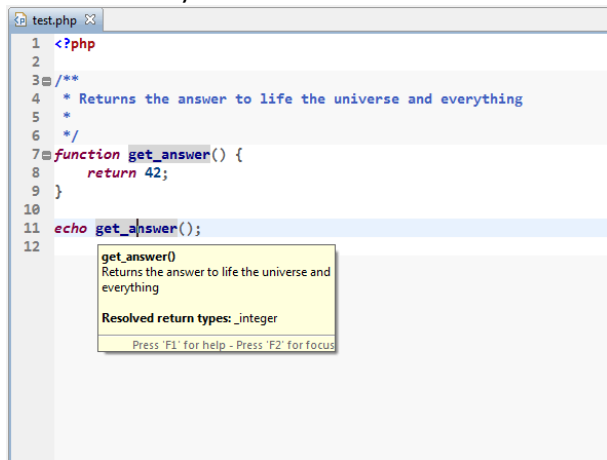


- DO IT.
- ... but do not overdo it: comments should be meaningful
 - Avoid obvious comments
 - Don't repeat yourself
 - Only add 'clarifications' when truly necessary (e.g. "we tried `thisotherfunction()` but it did not yield results")
- Use multi-line commenting style for 'documentation' comments (allows easy collapsing & extracting)
- Use object-specific commenting style to describe functions (allows easy access in IDE)
- Comments should describe the WHY (code describes the HOW)

Readable code

Use object-specific commenting style

This allows easy access in most IDEs:

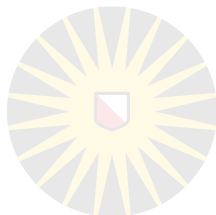


```
test.php X
1 <?php
2
3 /**
4  * Returns the answer to life the universe and everything
5  *
6  */
7 function get_answer() {
8     return 42;
9 }
10
11 echo get_answer();
12
```

get_answer()
Returns the answer to life the universe and everything

Resolved return types: _integer

Press 'F1' for help - Press 'F2' for focus



Readable code

Use multi-line commenting style for 'documentation' comments

Making them collapsible in some IDEs:

```
1  /**  
23  var countBy = createAggregator(function(result, value, key) {  
24    if (hasOwnProperty.call(result, key)) {  
25      ++result[key];  
26    } else {  
27      baseAssignValue(result, key, 1);  
28    }  
29  });  
30
```

```
1  /**  
2    * Creates an object composed of keys generated from the results of running  
3    * each element of `collection` thru `iteratee`. The corresponding value of  
4    * each key is the number of times the key was returned by `iteratee`. The  
5    * iteratee is invoked with one argument: (value).  
6    *  
7    * @static  
8    * @memberOf _  
9    * @since 0.5.0
```

Readable code

Be consistent!



In variable and function names:

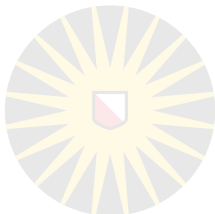
- Use a naming scheme (e.g. with CamelCase or `sepa_rators`)
- Use consistent temporary names (e.g. `'i'`, `'j'` for counters, `'ret'` for return variables...)
- Use specific styles for Functions vs `variable_names` (check a style guide!)

In indentation:

- Don't mix tabs and spaces.
- Choose a style and stick to it!

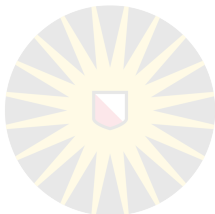
Readable code

Some final guidelines...



- Use descriptive variable names
- Keep code lines brief (max 80 characters!)
- Group code and use whitespace strategically.
- No
 - Endless
 - » Nesting
 - Please!
- Read your code, clean up comments and dead code!
- Advanced but recommended: **refactoring your code.**

Robust code



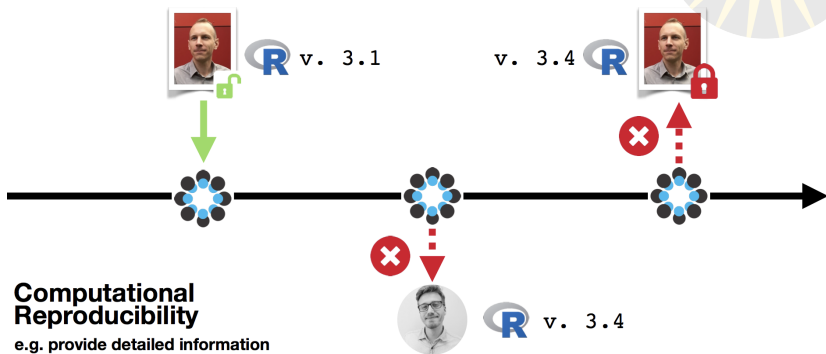
- Separate code and data: don't hard code variables.
 - *and if you must...*
 - Do it once, e.g. `weeklength = 7`
 - Use a config file!
- Don't repeat yourself: write repeating code into a function.
- Advanced but recommended: write unit tests.

Reusable code



- DON'T REPEAT YOURSELF (feel the irony here?)
- Separate code and data (also bears repeating!)
- Consider a config file (more irony...)
- Separate functions into generic and specific elements:
 - Generic part will be reused more often
 - Specific elements are unique applicable in specific situations
- Do One Thing (and do it well)
 - One function for one purpose
 - One class for one purpose
 - One script for one purpose (no fishing to recycle it!)

Context can make or break reproducibility



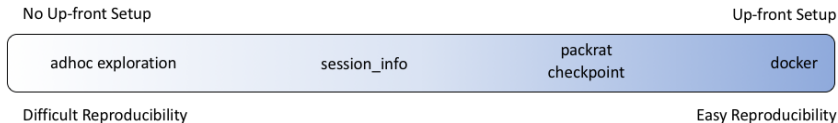
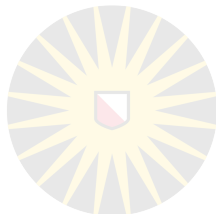
Source: [Timo Roettger on OSF](#).

Ensuring reproducibility



- Basic level: documentation
 - What dependencies/software are you using? Which versions?
 - What operating system are you working on?
- Intermediate level: dependency management
 - **Packrat** (in R): stores a snapshot of local R libraries
 - **Checkpoint** (in R): calling R packages by date
- Advanced level: containers
 - **Docker**: Linux-based project container
 - **Singularity**: packaging workflows and softwares, allowing communication outside of the container
 - **CodeOcean**: user-friendly, collaborative environment; interactive capsule.

The reproducibility trade-off



What is important to you? [Discuss!]



Take home messages

Project management:

- Start with a single project folder, including a README
- Distinguish read-only, human-written, project-written folders
- License from the start
- Consider open source software and interoperable filetypes

Make sure your code is:

- Readable
- Robust
- Reusable

Make your project reproducible:

- Write/extract project documentation
- Document dependencies
- Manage dependencies with specific packages
- (advanced) Place your project in an (interactive) capsule