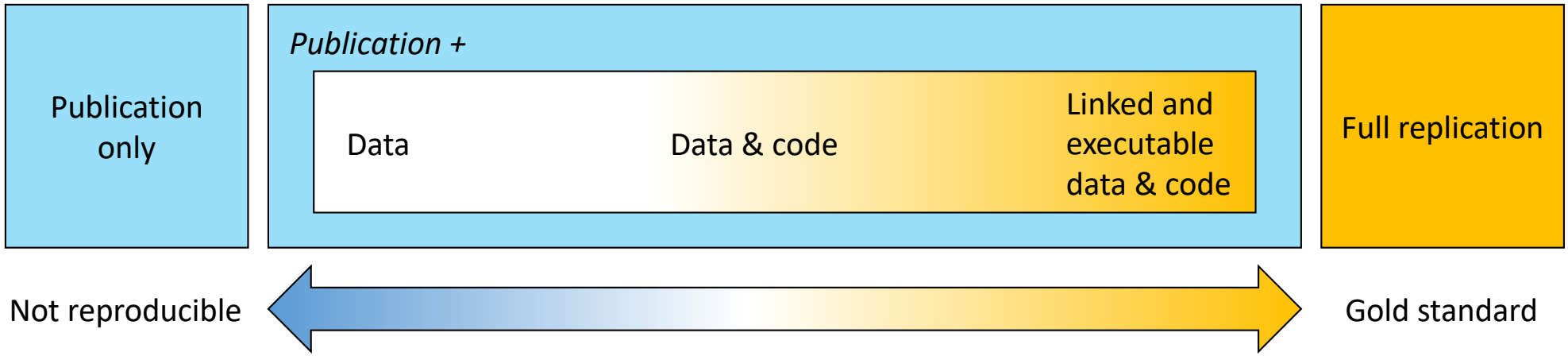
EEB 603 – Reproducible code



BOISE STATE UNIVERSITY

Learning outcomes

- Learn protocol to organize projects for reproducibility.
- Learn basic programming standards to ensure transparency and broad understanding of your data workflow.

Introduction

To make a code reproducible the following steps must be integrated:

1. Establish a reproducible project workflow.
2. Organize/structure project for reproducibility.
3. Ensure basic programming standards.
4. Document and manage dependencies.
5. Produce a reproducible report (with R Markdown).
6. Implement a version control protocol (with Git).
7. Ensure archiving and citation of code.

Introduction

To make a code reproducible the following steps must be integrated:

1. Establish a reproducible project workflow.
2. Organize/structure project for reproducibility.
3. Ensure basic programming standards.
4. Document and manage dependencies.

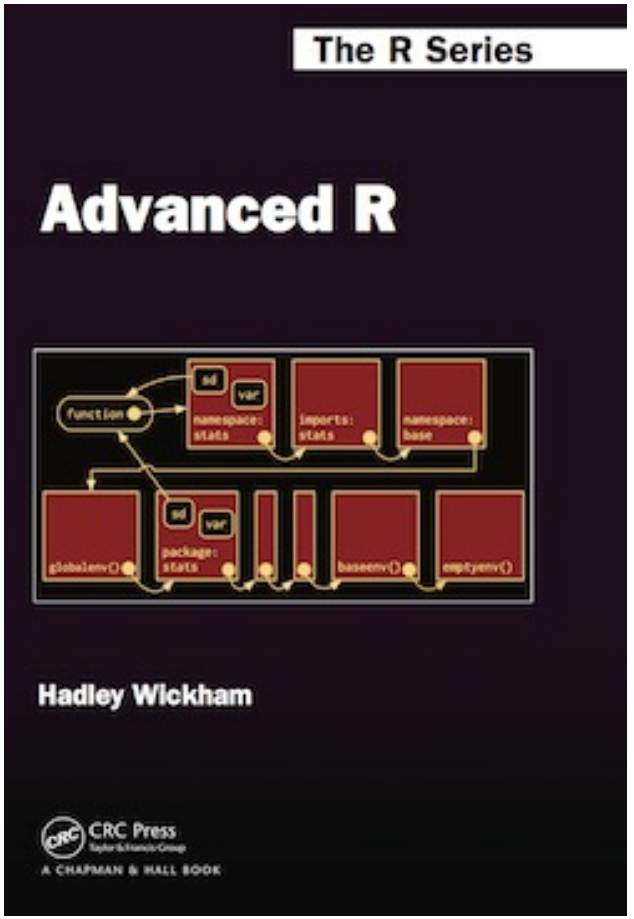
Chapter 5: Reproducible code
TODAY

5. Produce a reproducible report (with R Markdown). Chapter 1
6. Implement a version control protocol (with Git). Chapter 12: Bioinfo. tutorial
7. Ensure archiving and citation of code. Chapter 5: Data management

Before starting

- **Choose a project folder structure.**
- **Choose a file naming system.**
- Choose a coding style.
- Install and set up a version control software (Git) and connect to online account.

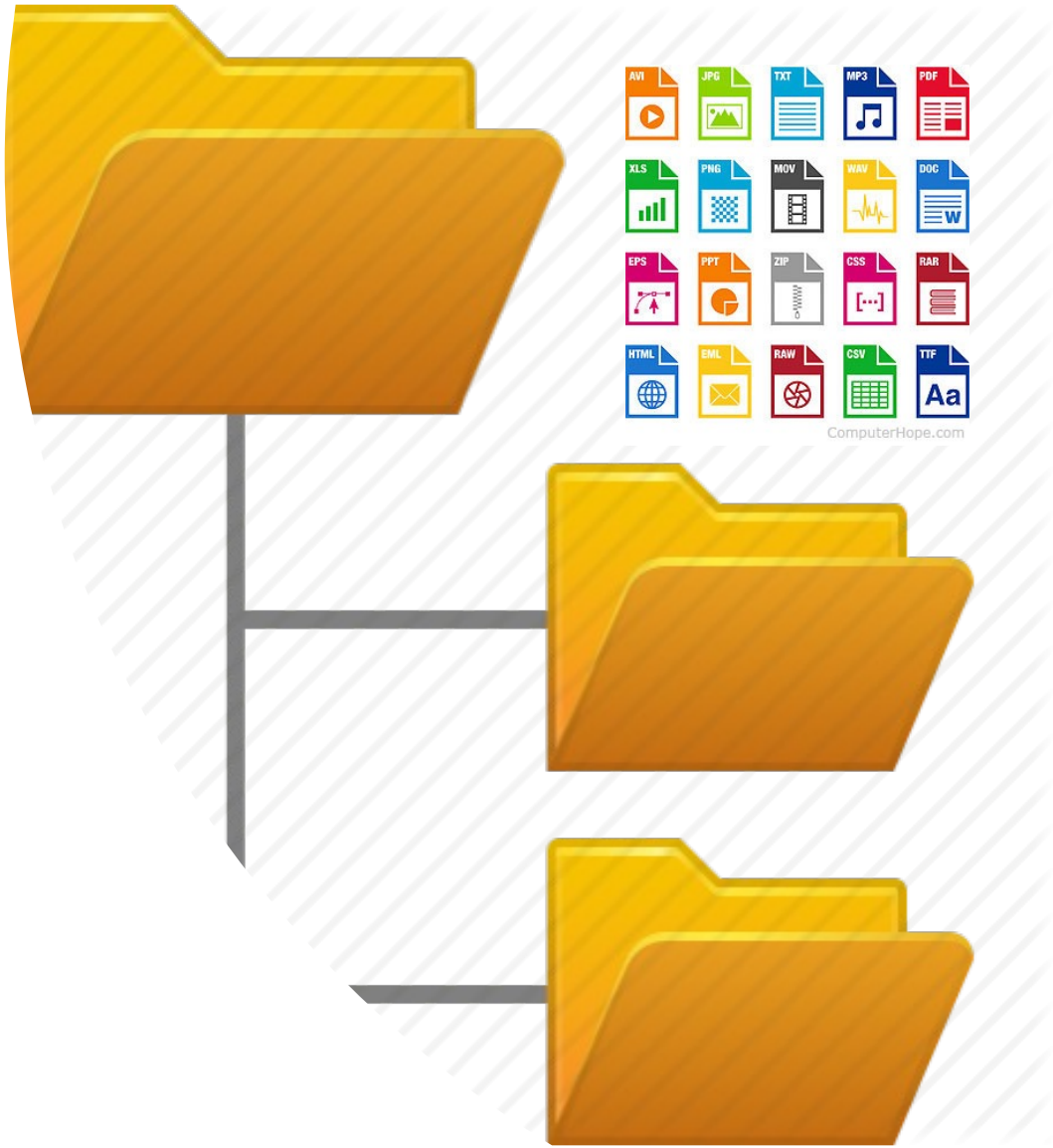
Chapter 5: Data management



https://adv-r.hadley.nz

Guidelines to ensure best processing of data

- **File formats:** Data should be written in non-proprietary formats, also known as open standard formats (e.g. .csv, .txt, .jpeg).
- **File names and folders:** To keep track of data and know how to find them, digital files and folders should be structured and well organized. Use a folder hierarchy that fits the structure of the project and ensure that it is used consistently.
- **File names should be:**
  - Unique,
  - Descriptive,
  - Succinct,
  - Naturally ordered and consistent,
  - Describing the project, file contents, location, date, researcher's initials and version.



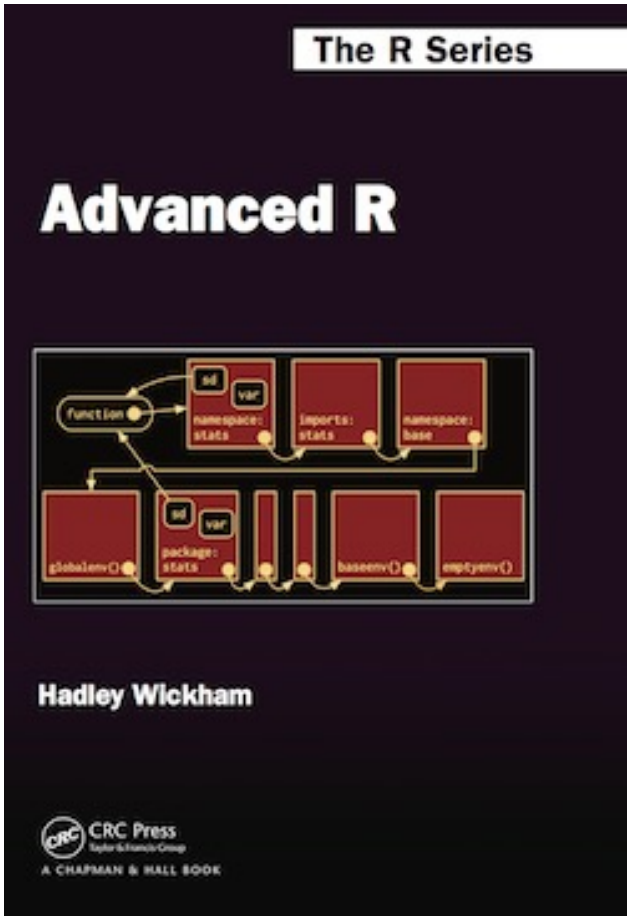
Guidelines to ensure best processing of data

- **File names should not include spaces** – these can cause problems with scripting and metadata.
- **Quality assurance:** Checking that data have been edited, cleaned, verified and validated to create a reliable masterfile, which will become the basis for further analyses
- **Assurance checks may include:**
  - Identifying estimated values, missing values or double entries.
  - Performing statistical analyses to check for questionable or impossible values and outliers (which may just be typos from data entry).
  - Checking the format of the data for consistency across the dataset.
  - Checking the data against similar data to identify potential problems.

Before starting

- Choose a project folder structure.
- Choose a file naming system.
- **Choose a coding style**.
- Install and set up a version control software (Git) and connect to online account.

Chapter 5:
Data management



https://adv-r.hadley.nz

Coding style

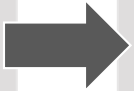
- **The foundation of writing readable code is to choose a logical and readable coding style, and to stick to it.**
- **Some key elements to consider when developing a coding style are:**
  - Using meaningful file names, and numbering these if they are in a sequence.
  - Concise and descriptive object names. Variable names should usually be nouns and function names verbs.
  - Using names of existing variables or functions should be avoided.

Coding style

- **The foundation of writing readable code is to choose a logical and readable coding style, and to stick to it.**
- **Some key elements to consider when developing a coding style are:**
  - Spacing should be used to improve visual effect: use spaces around operators (=, +, -, <-, etc.), and after commas (much like in a sentence).
  - Indentation should be with two spaces, not tabs, and definitely not a mixture of tabs and spaces.
  - Assignment (in R). Use <-, not =, for assignment.

Before starting

- Choose a project folder structure.
- Choose a file naming system.
- Choose a coding style.
- Install and set up a version control software (Git) and connect to online account.



First steps

- **Create the project folder and subfolders.**
- **Add a README file describing the project.**
- Create a version control repository for the project and connect it to online remote repository.
- Add a LICENSE file.
- Create a new reproducible report for the project.

The simplest and most effective way of documenting your workflow – its inputs and outputs – is through good file system organization, and informative, consistent naming of materials associated with your analysis.

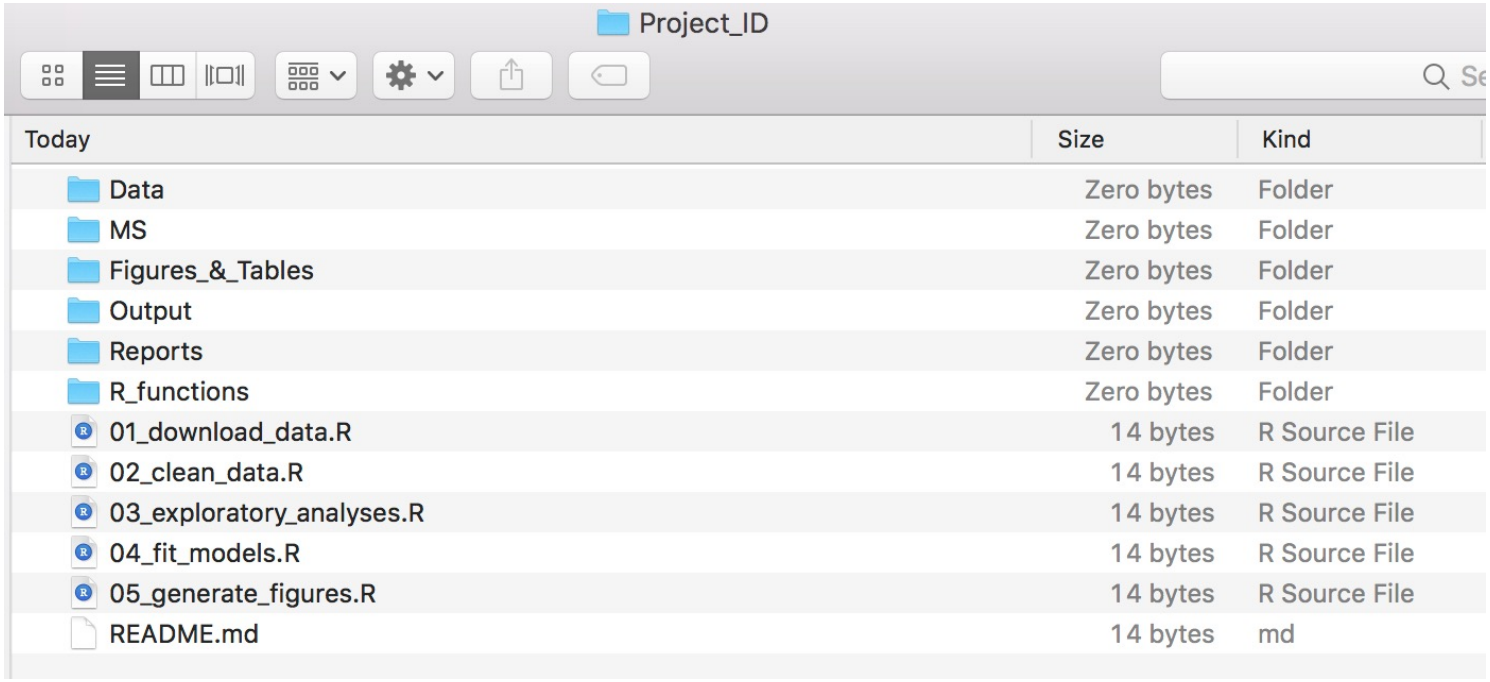
Before starting

- Choose a project folder structure.
- Choose a file naming system.
- Choose a coding style.
- Install and set up a version control software (Git) and connect to online account.

First steps

- **Create the project folder and subfolders.**
- **Add a README file describing the project.**
- Create a version control repository for the project and connect it to online remote repository.
- Add a LICENSE file.
- Create a new reproducible report for the project.

Example file structure of a simple analysis project



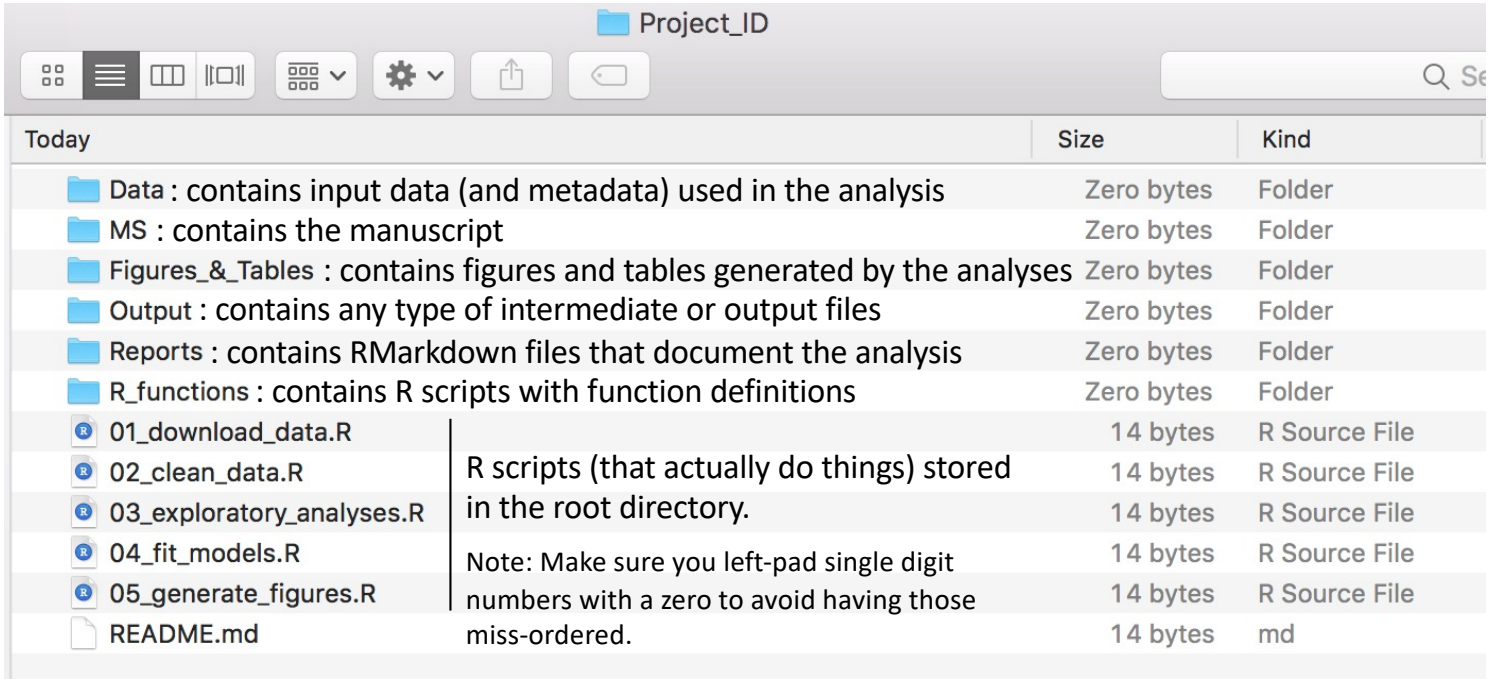
Before starting

- Choose a project folder structure.
- Choose a file naming system.
- Choose a coding style.
- Install and set up a version control software (Git) and connect to online account.

First steps

- Create the project folder and subfolders.
- Add a README file describing the project.
- Create a version control repository for the project and connect it to online remote repository.
- Add a LICENSE file.
- Create a new reproducible report for the project.

Example file structure of a simple analysis project



Portable code: Absolute vs. Relative paths

- An absolute path is one that gives the full address to a folder or file. A relative path gives the location of the file from the current working directory.
- For example based on species_data.csv stored in the Data folder
  - Absolute path: **C:/Project_ID/**Data/species_data.csv
  - Relative path: Data/species_data.csv
- **Using relative path and running from the project folder makes code portable.**
- In RStudio do: *Session -> Set Working Directory -> To Source File Location*

Before starting

- Choose a project folder structure.
- Choose a file naming system.
- Choose a coding style.
- Install and set up a version control software (Git) and connect to online account.

First steps

- Create the project folder and subfolders.
- Add a README file describing the project.
- Create a version control repository for the project and connect it to online remote repository.
- Add a LICENSE file.
- Create a new reproducible report for the project.

Write reproducible code

Write pseudocode

Write code
(functions & associated scripts)

Program defensively

Comment (#)

Test

Document code
(manage dependencies & reproducible report)

Writing clear, reproducible code has (at least) three main benefits:

1. It makes **returning to the code much easier** a few months down the line.
2. Results of your analysis **are more easily scrutinized by the readers of your paper**, meaning it is easier to show their validity.
3. Having clean and reproducible code available can **encourage greater uptake of new methods that you have developed**.

Commenting code

- How often have you revisited an old script six months down the line and not been able to figure out what you had been doing?
- A comment is a line of code that is visible, but does not get run with the rest of the script.
- In R and Python this is signified by beginning the line with a #.

  E.g. # Load data -----
- **Comments should explain the why**, not the what (we know that by reading the code).

Writing functions

- A function is useful when you need to repeat the same task many times!
- A function is a self-contained block of code that performs a single action.
- A function takes in a set of arguments, applies the action, and returns an object of any data type.
- A function should not rely on data from outside of the function, and should not manipulate data outside of the function.

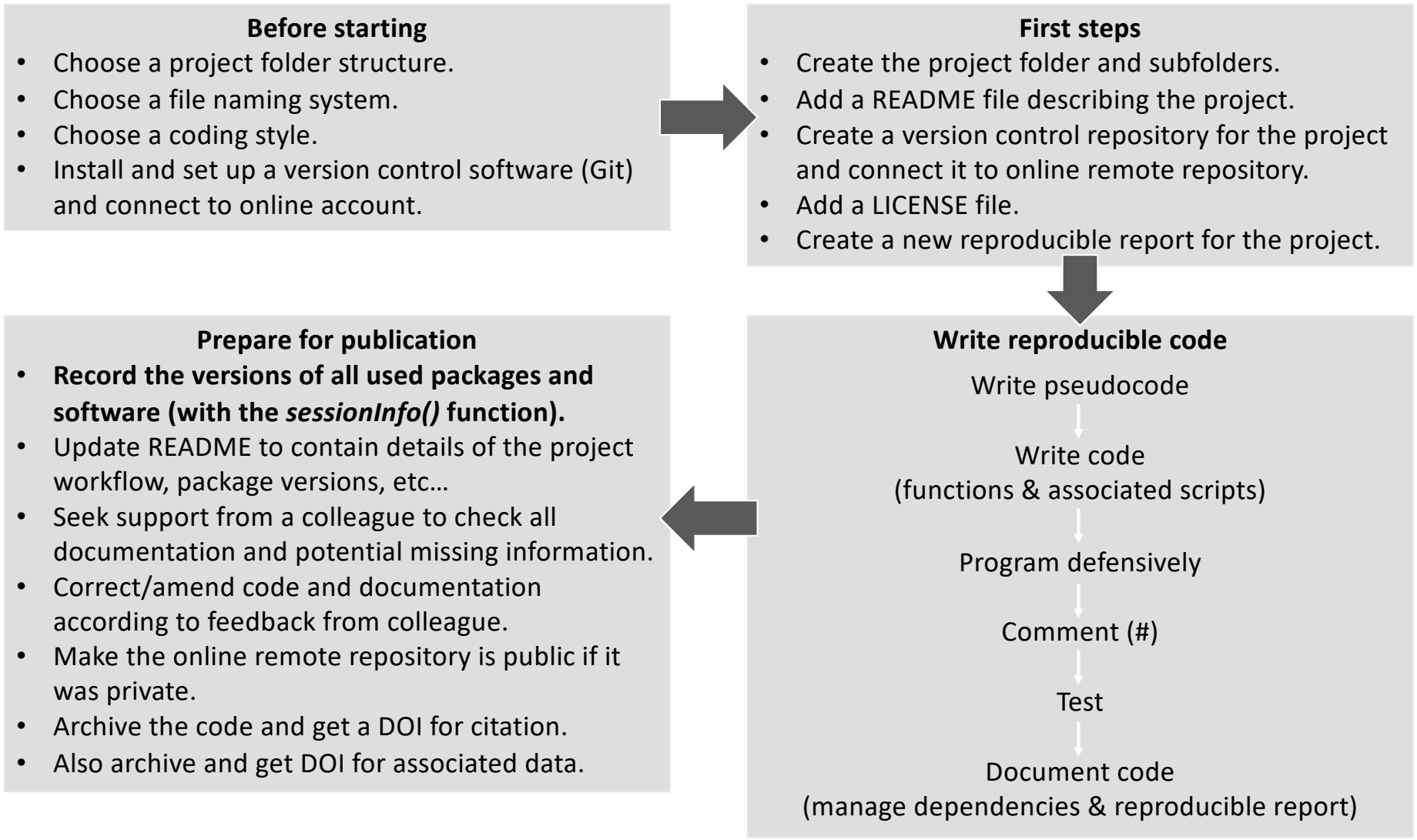
Writing functions

- How does a function look like in R?

```
Name <- function(argument(s)){
    some code using argument(s)
    return
}
```

Defensive programming: Allow debugging

- Defensive programming is a technique to ensure that **code fails with well-defined errors**, i.e. where you know it should not work.
- The key is to **'fail fast'** and ensure that the **code throws an error (meaningful to you)** as soon as something unexpected happens.
- This creates a little more work for the programmer, but it makes debugging code a lot easier at a later date.


Before starting
• Choose a project folder structure.
• Choose a file naming system.
• Choose a coding style.
• Install and set up a version control software (Git) and connect to online account.
First steps
• Create the project folder and subfolders.
• Add a README file describing the project.
• Create a version control repository for the project and connect it to online remote repository.
• Add a LICENSE file.
• Create a new reproducible report for the project.
Write reproducible code
Write pseudocode
Write code
(functions & associated scripts)
Program defensively
Comment (#)
Test
Document code
(manage dependencies & reproducible report)
Prepare for publication
• Record the versions of all used packages and software (with the sessionInfo() function).
• Update README to contain details of the project workflow, package versions, etc…
• Seek support from a colleague to check all documentation and potential missing information.
• Correct/amend code and documentation according to feedback from colleague.
• Make the online remote repository is public if it was private.
• Archive the code and get a DOI for citation.
• Also archive and get DOI for associated data.

Reporting R packages & versions

R version and packages that I used to create this chapter

```
sessionInfo()
```

```
## R version 3.4.1 (2017-06-30)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.4
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.4.1  backports_1.1.2 magrittr_1.5    rprojroot_1.3-2
##  [5] tools_3.4.1     htmltools_0.3.6 yaml_2.1.16     Rcpp_0.12.15
##  [9] stringi_1.1.6   rmarkdown_1.10  knitr_1.20      stringr_1.3.0
## [13] digest_0.6.15   evaluate_0.10.1
```