Contents

1 Introduction

This guide is a collection of techinques, styles, suggestions and requirements for collaboratively writing R code for the MSU Spatial Community and Ecology lab.

A programming "style" guide has detailed descriptions of exactly how code is formatted with spacing, indents, function and variable naming, etc. Thee are usually written for professional programmers. This guide is for R users of varying skill levels and has approaches to broad programming issues *like how to manage configuration and data) that we use as a lab to allow for collaboration and longevity. Nothing in here is set in stone and is open for suggestion and additions. There will be some formatting and style requirement. For coding style, we refer to, but don't require the use of the "tidy verse" style guide originally written at google and adopted and amended by Hadley Wickham: (https://style.tidyverse.org/)[https://style.tidyverse.org/]

Certainly when learning or in the middle of developing an idea for program, focusing on style can bog you down, much like writing. Refer to this guide when polishing your code or preparing to share your ideas, and especially when packaging for publication.

As painful as re-formatting your code and project can be, we find that consistency in approach to programming add efficiency to the programming efforts in the lab. This guide is focused on R but some of the ideas should translate to other scripting languages you may use for the lab, such as python, Javsccript (e.g. EarthEngine), or shell scripts (e.g. Slurm scripts for batch computing on the MSU High Performance Computer).

2 Contributing

We need your help. Please contribute or insert comments and your experiences in this guide. This is written in a form of RMarkdown called "bookdown." (there will evenntually be a chapter here about Rmarkdown)

Please contribute if you've never used bookdown. We are delighted if you want to contribute and don't want you to worry about formatting or bookdown tricks. We are interested in your ideas. Simply edit any one of the chapter files (Rmd files that begin with a two digit numeral) as you would edit an Rmarkdown file. If you want to add a new chapter, simply start a new Rmarkdown file, preface with a header (e.g. `# Tracking Woozles`) which will be the chapter title. We can incorporate these changes and edit for bookdown tech as needed.

This project exists on a public github project, so please fork, make changes and create a merge request.

Alternatively if you are unsure or just have a short idea or correction, you may submit a github issue to this project (see Github.com:Creating an issue which requires a github account.

If you are a lab collaborator and have a found a successful technique we want to hear about it. Consider a new chapter describing this and pointers to examples on github.

2.1 Prerequisites

You need a plain text editor that can create Markdown (we use Rstudio). Word Processors insert unreadable characters (curly quotes). When writing markdown ou can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

To render this book, you need the **bookdown** package, which can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this book to HTML... **instructions needed here** The project is set to render the HTML into the 'docs' folder which will make it easy to publish as an HTML website (e.g. github pages).

To compile this book to to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.org/tinytex/.

#Documentation

Documenting your work may be the most important part of this style guide.
There are many ways to get a script to do it's work, but a script is useless if we don't know what it does, and how to use it. There are two main types of in-script documentation : documentation suggested and structure for the lab, and documentation done for R packaging using the program Roxygen2. Roxygen2 is a way to write comments in your code that can automatically be turned into R help documentation. This guide currently does not mention that, but focuses on basics for docs, but these are not exclusive and eventually the stuff here should be made Roxygen2 formatted. Rstudio has features to make it easy to write Roxygen2-style comments.

The following in R code all start with Comments (#) All scripts need a good description in the beginning - what does this script do? What kind of data does it read in and what does it output?

```
# TITLE: name of the script
# AUTHORS (list anyone contributing to the file)
# COLLABORATORS (other people involved in the wider project but not necessarily on the script)
# DATA INPUT (a brief description of the data read in through the script, including what format it's in
# DATA OUTPUT (a brief description of the data output from through the script, including what format it
# PROJECT (what overarching project this is part of: in this case it's Frugivoria)
# DATE (initiation date of script, plus any major update dates)
```

```
NOTE : this guide was written without consideration for the Roxygen documentation system and if you are

Note also that Roxygen2 focus is on creating documentation for using functions, not the provenance of a
```

3 Configuration and R scripts

This is part of the solution for working with configuration in your R scripts. We are assuming that you want to share your scripts one for collaboration but also for reproducibility. One constant problem is scripts need to read files from somewhere, and everyone's computer has different paths, or the paths differ when on the HPC. We often see scripts with the configuration variables at the top of the script, and you have to edit those to your environment, or comment them out. But then those go into the git repository, so collaborators have to change them back. Even for your own scripts, you may want to change a file path depending on if you are running on your laptop or on the HPC for example. It's hard to track down all the places where you have to change these parameters (although global search/replace helps). But it would be better if there was a single, consistent place to put all configuration values shared by scripts in a project Solution

R has a built in feature to read a file called .Renviron exactly for this purpose. Using files that start with "." (so-called dot-files) for configuration is common in Linux, where R was born. Note There are a couple of libraries to make this a bit more slick, but this document describes only features that are part of Base R. We can add description of how to incorporate those below

Good summary of .Renviron : www.dartistics.com/renviron.html

Not a bad summary of Environment variables : https://en.wikipedia.org/wiki/Environment_variable

More detail on Environment vars in Mac/Linux : https://dev.to/gajesh/environment-variables-in-linux-and-mac-os-4j30

To see how R reads dot-files .Renviron and .Rprofile, read the R help ?Startup or see this copy on the web (I don't know how current the copy is) : https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/Startup

Note that there is also a package called "startup" on CRAN which is an enhancement and not part of base R.

.Renviron is for setting values for script configuration. It uses the Operating System Environment to store these (e.g. not the same as an R environment), e.g. "environment variables" Any program can read or set an environment variable. One commonly used variable is $HOME which is your home directory.

3.1 Quick Points on Configuration

when R starts it looks for two files

.Renviron <- set of environment variables listed in here. Written as shell script

.Rprofile <- options for how you like to use R. written as R code

If these files are in the local directory, it reads those and stops.

If not it looks for them in your home directory (global settings). It only reads one version (e.g. it doesn't read local then also read global settings)

3.2 Using Renviron

```
#  .Renviron file, each person makes their own copy of .Renviron  Note you can use existing operating sy
API_KEY=abc123
OUTPUT_PATH="$HOME/Documents/NEONOrganism/DataPull"
```

inside the R code, which does not have to be edited by each person

4 this would have to go at the top of each function that needed this configuration

You'd have to set these at the top of each script.

Alternatively, I would try to avoid having a handful of scripts, to which you'd have to copy/paste those same statements at the top. The software engineering principle here is "Don't Repeat Yourself" or… keep your code DRY. To do so, wrap code into functions as much as possible, and pass those as configuration to the function. That advantage is that is make those functions testable and portable

The main script to run then would

Workflow to use lab code that needs configuration in Renviron:

1. clone the repository
2. in the README.md look to see which configuration settings are needed and/or looked for a file Renviron_example.txt

3. create a new .Renviron file in the top level folder of the repository with the necessary configuration
4. open a new Rstudio project in that folder (or start R) which would then read the .Renviron settings
5. run the code
6. repeat for different environment (e.g. laptop vs HPCC)

workflow when using configuration in your code with Renviron:

1. put configuration in .Renviron and add Sys.getenv() to set those vars in R
2. test
3. add those configuration vars in Renivron_example.txt with comments to describe possible values
4. add a list of the configuration need in the README.md
5. make sure .gitignore has .Renviron listed in it to keep it out of github
6. optional but polite thing to do for other users of your code: wrap the Sys.getenv() with an if statement or exception handling to
    1) check that the var is set in .Renviron
    2) set a reasonable default if it's not
    3) test that the value the user supplied (or default) actually work
    4) throw exception if they don't and print message which value needs to be set and where to look for guidance (see README)

4.1 File Paths

File paths for data files are a special case when working with configuration and there are different ways to handle them. One is to use the same directory in all of you

4.2 Complimentary Packages

These other packages/features work to make configuration easier, or could be part of a configuration strategy.

startup: https://cran.r-project.org/web/packages/startup/vignettes/startup-intro.html Package from Henrik Bengtsson to make it easier to work with .Renviron and Rprofile. Don't know how well is plays with config

config: https://cran.r-project.org/web/packages/config/vignettes/introduction.html
Package from Rstudio to make it easy to have different Environments so you can easily switch out configuration, for example the paths/data connections you'd use for testing would be different for local and from HPCC. This is especially important when working with "production" or shared databases as you want to develop/test code on a local copy before running it on the central database or one used by an active web application

dotenv: older package that borrows from python. Does not use Renviron but a "dot-env" or ".env" file just like Renviron. I think using Renviron makes more sense but mention this as it comes up when searching. A package like this is required in Python because there is nothing built-in to handle config.

options: https://stat.ethz.ch/R-manual/R-devel/library/base/html/options.html part of base R and a way to set options globally so you don't have to send them as parameter to every function. This is complimentary to Renviron: set config values in Renviron, use the options() function to set global options var in R based on those env variables.

Profile or Customization : the .Rprofile should be used to customize your R experience, not set configuration for a particular set of scripts. See https://www.statmethods.net/interface/customizing.html