Hakai Institute R Stats Guide

Brett Johnson

Latest update: 2018-11-22

Contents

Preface **5**

1 Introduction **7**

2 Data Analyst's Toolbox **9**

3 Intro to R **11**

4 General Principles **13**

5 R Workflow Tips **17**

6 Version Control and Collaboration **21**

7 Mapping in R **27**

8 Resources and References **29**

9 Glossary of Terms **31**

```
#Once you build this book, you can serve the book to auto update in the viewer window by removing the h

#bookdown::serve_book(dir = ".", output_dir = "_book", preview = TRUE, in_session = TRUE)
```

Preface

One of Hakai's core themes that cross-cuts each research axis is 'Big-Data and Modelling'. Our ability to gain insight from our long-term ecological data sets is limited only by our capacity to integrate, and digest data. To that end, this guide was developed to serve as a foundation from which to build the internal technical capacity — using a coherent and systematic approach — to turn data into insight, understanding, and ultimately knowledge about the ecosystems we study.

Through this guide you can learn data management skills, R programming basics, project workflow and organization, reproducible research techniques, and collaborative analysis development in an open-science context. These skills are learned through R and R-Studio, granted the skills and principles are relevant to many programming languages or analysis softwares available.

Chapter 1

Introduction

1.1 Filling in the Gaps

Most University courses that teach statistics and data analysis are focused on teaching statistical techniques such as generalized linear models, or logistic regression — which is very necessary and important — but they pay no attention to educating students on how to actually conduct an analysis from start to finish. Questions that often remain un-answered include:

- What is an efficient workflow?
- How do I access and import data?
- How do I clean and manipulate my data into a format to analyze?
- How do I maintain version-control of my analysis?
- How can I re-run my analysis in case I get new data or someone else wants to run my code?
- How can I collaborate on this analysis?
- How can I get my analysis into a format for someone to meaningfully conduct peer-review?
- How do I efficiently produce a professional report or other artifact of my analysis to communicate results?

This guide aims to bridge this gap by taking you through the steps to produce a well-developed analysis that will form a template for conducting your own analyses while working with Hakai. You will be directed through an efficient way to learn the basics of R, and progress to modern methods of communicating results. Learning to develop analyses in R can be a very frustrating and difficult experience. Know that we have all suffered severely in this way, but that this guide will hasten you through the drudgery and frustration. No computer programming experience is required to work through this guide, though an undergraduate course in ecological statistics is recommended as pre-requisite.

1.2 Modern Approach to Data Analysis

There is a vibrant 'open-source' community of people developing methods, packages, and workflows in the R programming world. Consequently, we have some of the most modern, flexible, and high-level methods to develop and communicate statistical analyses.

However, as the R community iterated ideas of how the programming language works at a very base level over the last couple of decades, we are left with a litany of methods and programming syntax. There are typically at least half a dozen ways to write a chunk of code to reach the same desired result in R. This, in part, has given R the reputation as being difficult to learn.

This guide aims to address the 'thousand and one way of doing things' problem in R by focusing on the recent development of packages that form a simple, elegant, and coherent grammar of data analysis. This collection of packages and methods is known as the 'tidyverse', developed in large part by the Chief Scientist of R-Studio, Hadley Wickham, and has been widely adopted as the way forward for academic applications of the R language.

The objectives of this book are to serve as a:

1) Guide to install, set-up, and become familiar with analysis development tools; R, R-Studio, Git and Git Hub.

2) Best-practice guidelines to develop reproducible, accurate, and collaborative analyses in R-Studio.

1.3 Acknowledgements

Much of this document refers you to material that others have worked very hard to make. I simply point to these resources in an order that makes sense and seems systematic to me. Many thanks to the following people for making this possible:

- Dr. Jenny Bryan, Data Science Professor at UBC, RStudio Employee. Twitter, GitHUB.
- Dr. Roger Peng, Professor of Bio-statistics at Johns Hopkins University. Website, GitHUB.
- Dr. Hadley Wickham, Chief Scientist at R-Studio. Website, Twitter, GitHUB.
- Dr. Hillary Parker, Data Scientist at Stitch-fix. Twitter

Chapter 2

Data Analyst's Toolbox

2.1 R and R Studio

R is the statistical programming language that R Studio provides and interface to. I think of R Studio as being the front-end user interface to the coding language R which works in the background. For this all to work you you need to download two things to start; R and R Studio.

1) **Install R**: Go to https://cran.r-project.org/ and follow the link to the download for your operating system.
2) **Install R Studio**: Go to https://www.rstudio.com/products/rstudio/download/ to download R studio.

Now that you have downloaded the tools you need, lets get familiar first with R-Studio and then the R programming language.

2.1.1 Step-by-step R/RStudio

1) **R Studio Familiarization**: Watch this R-Studio Tutorial for a nice introduction to the software.
2) **Start Plotting**: Now that you're familiar with the software, lets get your feet wet in R and R-Studio. I recommend you start making some plots by working through the following exercises: Section 3.1 to 3.4 in Hadley Wickham's R for Data Science Book.
3) **Projects Workflow**: Watch R-Studio tutorial: Projects and R Studio
4) **Install Hakai's Application Programming Interface** This API is installed directly in R-Studio and allows you to download data from the Hakai Database. Open up R Studio and in the console window copy and paste the following code and hit enter:

```
install.packages('devtools')
install.packages('tidyverse')
library('devtools')
library('tidyverse')

devtools::install_github("HakaiInstitute/hakai-api-client-r", subdir='hakaiApi')
```

Chapter 3

Intro to R

In this section you will learn the basics of data wrangling in R. Data wrangling is the process of importing, tidying, and transforming your data into a format that you can visualize and model it. Tidy data is a consistent data format that, when followed closely, dramatically reduces the mundane data formatting that is necessary otherwise. The basic concept is that every row is one observation and every column is a variable. For how simple it sounds, creating tidy data sets takes some time to get the hang of initially, but is essential for data wrangling in the tidyverse — the modern set of tools and packages developed in R around the concept of tidy data.

3.1 Primary Resource

The 'bible for a new generation of Data Scientists' is Hadley Wickham and Garrett Grolemund's Book: R For Data Science. This book presents a modern approach to data analysis and leads you to master the tidyverse; a combination of R packages and a well thought out and systematic approach to "import, tidy, transform, visualize, and model data." Using the tidyverse as a foundation for your coding replaces the 'thousand and one ways' of doing things in R into a modern and concise grammar of data analysis development.

3.2 Data Transformation

The most important set of tools for data wrangling that you will use constantly are from the package `dplyr`. They allow you to solve almost all of the data manipulation problems you will encounter. Those who are able to quickly manipulate their data into a format they can visualize and model, will explore their data the most and therefore learn the most. I'd recommend learning these six data wrangling functions off-by-heart:

1) `filter()` to pick observations (rows) based on their values
2) `select()` to pick variables (columns) based on the names
3) `arrange()` to reorder the rows of your data set
4) `mutate()` to create new variables (columns) as a function of existing variables
5) `summarize()` to collapse many values down to a summary
6) `group_by()` to summarize based on the various groups of a variable

To learn how to put this in to practice I recommend reading through the chapter called Data Transformation in R for Data Science and working through all the exercises. This will be a bit of a learning curve but it's time to dive in!

Starting at the Data Transformation chapter assumes you have some very basic knowledge of R already. If that's not the case then I recommend starting at the Workflow: Basics chapter.

3.3 Tidying Your Data

Tidy data is best explained by a quote from Hadley Wickham's paper in the Journal of Statistical Software:

> "Tidy data sets are easy to manipulate, model and visualize, and have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table."

The basic tenets that make a data set tidy are:

1. Each variable must have its own column
2. Each observation must have its own row
3. Each value must have its own cell

All of the data stored in Hakai's database is designed to be tidy, but sometimes you will receive data that is not in a tidy format. The most common problem I run into is that not every row is one observation, it is often many more than one because observations are stored as variables. To get your data into a tidy format you have to spread out the observations into different rows and create new variables that were once values. Luckily, the package `tidyr` has several functions that make this possible.

To get a really strong handle on what tidy data is, and to start practising getting data in to a tidy format I recommend working through the chapter on Tidying Data in the R for Data Science book.

3.4 Visualizing Data

In the last chapter you had a teaser of what is possible in terms of making some nice looking plots when you worked through the first few sections in the Data Visualization chapter in R for Data Science. To see what else is possible and to learn the principles of the layered grammar of graphics, I recommend you work through the remaining sections in the Data Visualization Chapter picking up where you left of at the sub-section on facets

By the end of this you should have a really solid foundation from which to begin wrangling and visualizing your own data. In the next chapter I'll provide some gerenal principles for developing analyses.

3.5 Learning Objectives

Before moving on to the next chapter, work through these chapters mentioned above and complete their respective exercises:

- Workflow: Basics
- Data Transformation
- Tidying Data
- Data Visualization

I highly reccommend working through all the chapters in this book at some point, but these first few chapters are a good start.

Chapter 4

General Principles

New ideas about what makes a good data analysis are emerging. With data being so readily available in vast quantities, analyzing data using out of date methods — such as Microsoft excel — can quickly become overwhelming, not reproducible, error-prone, and difficult to assess for reliability.

Much of the progress in terms of 'developing analyses' has been made in the field of bio-statistics due to the high volume of genomic data that researchers deal with. One of the most concerning examples of what can go wrong with an analysis, is from the field of genomics and cancer treatments. In the 'Duke Scandal', researchers made mistakes in their data analysis, that were extremely difficult to track, and resulted in patients receiving the wrong cancer treatment. This is an extreme example that affected peoples lives directly. I would argue, that the work that we do at Hakai, analyzing ecological data, has much broader implications and should be treated with an equal degree of discretion.

Some important concepts in defining a good data analysis are:

1) Reproducible Research,
2) Open Science Collaboration

These two concepts bring together a very modern way of conducting science that are beneficial in the following ways:

- An additional peer review process becomes possible in the development of your analyses—peers can review how you conducted your analysis
- You can reduce your work in the long run by being able to reproduce your own analyses after you've long forgotten the details of how they were conducted
- If you weave a narrative text into your R code you'll be able to understand what you were thinking at a later time when you revisit it
- You can meaningfully collaborate
- You can show your peers that you have nothing to hide in your analytic methods, immediately increasing the reliablilty of your work
- You can share your analyses in hopes that others will improve the quality of your analysis by offering their insight

4.1 Reproducible Research

A big concern with modern scientific studies is the inability to reproduce study findings. As experimental and analytic methods become increasingly complex, published methods often lack the detail required to be able to reproduce the study. As data analysts, it is our responsibility to include in our analyses sufficient notation to facilitate deep understanding of what was done.

If your study finds something very interesting, people are going to want to know how you came to your conclusion. The gold standard for verifying a study is to independently replicate it. However, before investing the huge amount of resources required to replicate a study independently, a better place to start is to reproduce the study findings using their data and analysis.

In order for your study and your analysis to be reproducible and to be viewed as trustworthy, you need to be able to provide the data, the scripted code you used to clean, summarize, model, and visualize that data, and then a reviewer has to be able to run that same code and see the same results. This level of transparency allows a reviewer to look very closely at how you conducted your analysis. This adds an additional step in the peer review process which has not previously been possible with un-scripted analyses.

A simple example of how easy it is to break the reproducibility chain comes from working with your data in excel. By simply deleting some values that looked to be outliers, without recording anywhere that you did that, or why you did that, you have effectively broken the reproducibility chain. Another person could not receive the raw data and come to the same conclusions as you did — their results would be different because of the missing data.

Some academic journals, such as the Journal of Bio-statistics, are taking proactive steps by adopting a reproducibility policy for article submissions. The journal of Nature has taken a stance on reproducibility and has published several articles in a special called Challenges in Irreproducible Research. If you google reproducible research you will find many initiatives that are attempting to solve this important problem.

Most often, the peer-reviewer or collaborator that you will work with is 'future-you'. Scripting reproducible analyses with embedded narrative allows future-you to understand what past-you was thinking. This, in turn, saves you a lot of time, and allows you to build upon your previous work. The idea of embedding your own narrative, or adding comments to your code, introduces the idea of *literate programming*. Weaving together human-readable narrative text that explains what your computer code is doing and why you decided to do it, greatly increase the quality of your work.

For some comic relief on the follies of working with someone who doesn't use reproducible research methods watch this youtube video

4.2 Open Science Collaboration

4.2.1 Peer Review

Open science collaboration establishes an additional peer-review step in the scientific process. By making your analysis public using a distributed version control system like GitHub you open up the possibility for on-going peer-review of your analysis, as well as the the opportunity for not only your immediate colleagues, but also experts in your field to contribute in a meaningful and formalized way.

4.2.2 Code Review

Before you publish or distribute your analysis, you must have a colleague reveiw your code. You would never publish a written article without sending it to a trusted co-worker or superviser first. Code review is the same thing.

While peer-review permits the scientific merit of your analyses to be assesed, code-review permits a colleague to assess your analyses for coding errors. Developing analyses is difficult and error prone. Using the *distributed version control* system GitHUB, you are able to track changes to your code over time and enable others to suggest edits for your review, leaving you with a history of exactly what was added when and by whom — this is version control. The distributed aspect of 'distributed version control' means that multiple people can access the version control through your *repository* of all the files of your analysis (and their previous versions). You can make your repository (repo) private or public. By acessing your repo, which is hosted on GitHub, others can meaningfully collaborate, conduct peer-review, and code-review.

Using consistent formatting for writing your code, makes it easier for everyone to interpret. I recommended the The tidyverse style guide. Have a read through the guide and try to implement in for your next script. To check that your script follows the tidyverse style guide, a code formatting 'spell-check' package called `lintr()` comes in handy. In the R console simple type:

```
install.packages("lintr")
library(lintr)
lint("/path_to_your_file.R")
```

Lint will take you row by row of your code and tell you how you deviated from the tidyverse style guide.

4.2.3 Collaboration

Software developers have been using this type of development platform for some time. By using these tools, data analysis can progress towards the concept of *developing analyses*, as opposed to an ad-hoc analysis. Ultimately, this approach makes your life easier. If you've experienced the pain of digging through your emails or zip files to find the analysis you or your colleague conducted years ago, only to be left perplexed by which version is the 'master' or 'final' then Git HUB will be a welcome addition to your toolbox.

GitHub can also be used as a project management tool for example by creating to-do lists for yourself and your team. By creating *issues* you can leave yourself a to-do list, or collaborators can identify issues with your analysis. We will cover Git HUB fundamentals in the next chapter.

Chapter 5

R Workflow Tips

5.1 Create a new R-Studio Project

Generally I create a new project for every analysis I undertake, this makes things easy because the working directory remains constant amongst some other advantages.

To start a new project in R-Studio, go to File > New Project.

5.2 File paths and the working directory

Often in an analysis you have to load or save files to a specific folder or file location on your computer. You should not use absolute paths to do this such as `write_csv(file_name, "C:Brett/documents/R projects/Hakai R Analyst/data/file_name.csv")`. The absolute path starts at the root of your computers specific file system, and other people will have different absolute paths on their computer depending on where they saved their files. So if you shared a script with an absolute path, your collaborator won't be able to run the script without headaches of changing the abosulute path. Fortunately, you can use relative paths. Relative paths don't go all the way back to the root of your file system. A relative file path in this example would be `"/Hakai R analyst/data/file_name.csv"`. Using relative paths makes the scripts or programs portable between computers. Defining where a relative path starts from can be done by setting your working directly, or by using the `here()` package.

A recommended method to work with relative file paths is using the `here()` package in conjunction with R-Studio projects. When you create a new R-Studio project, a .Rproj file is automatically created in the new folder that you created for the project. The `here()` package will automatically set your working directory to wherever your .Rproj file is saved. That means you can save a file like this: `write_csv(file_name, here("data", "file_name.csv"))`. Using `here()` means that if you access your collaborators folder where the .Rproj file is and they have been using relative paths using `here()`, the scripts should all just work—no chaning working directories or absolute file paths.

5.2.1 Create folder structure

I use a default folder structure for every analysis based on the files that are produced from every analysis. Using the project directory that you created your new R-Studio project, create these sub-folders within the project folder:

- raw data
- processed data

- scripts
- figures

5.3 Analysis Workflow

Here is a general workflow I typically adhere to, and could be adopted as a starting point from which individual analysts could modify.

5.3.1 Import, pre-process & tidy data

I usually create at least two different scripts in any analysis, which helps me to compartmentalize the different steps of the analysis. I start with a pre-processing script that will read in and format all the different data sets I want to combine and analyze. This is where I make changes to data format that I want to apply to all subsequent uses of that data (so that I don't have to keep doing it in subsequent scripts).

1) **Create pre_process.R script** In your newly created R Studio project, go to File > New File > R Script. Save it in the scripts sub-directory of your project directory.

2) Load packages

3) Import data:

Method 1:From spreadsheet you already have

Most often you're going to want to read in files that are .csv files. These are comma separated value files and can be produced from excel or Google Sheets by saving your excel or Google Sheet file as a .csv file.

The first module of an analysis I produce is a plain .R script that loads in my .csv data file and save it in my R environment as a tibble, a tidy table, using the `new_tbl <- read_csv(here("data", "new_tbl.csv")` format. Before you read in a file, you should load the packages that we will be required for every analysis you conduct using the `library(tidyverse)` function. Note that you should not use the base R function `read.csv` but rather use the tidy-verse function `read_csv`. The base version will inevitably cause frustration due to incorrect variable class assignment for dates.

Method 2: From Google Drive

Using the googlesheets package in R is a pretty powerful tool and allows you to read googlesheets directly into R. This is great if your googlesheet is constantly changing as new data gets entered, and allows easy collaboration on data entry.

To read in a googlesheet:

`install.packages('googlesheets') library(googlesheets)`

your_workbook <- gs_title('Name_of_your_workbook') worksheet1 <- gs_read(your_workbook, ws = "sheet 1")

See this documentation on the googlesheets package for more info.

Method 3:From Hakai database

It is possible to get data from the Hakai database directly into R without having to leave R Studio. This is accomplished by interacting with an application programming interface (API) that was developed for downloading data from Hakai's data portal. Earlier in the Data Analysts Toolbox section we installed Hakai's API. Now we will use it to get Hakai data from the database.

Below is a quickstart example of how you can download some chlorophyll data. Run the code below one line at a time. When you run the 'client <- …' line a web URL will be displayed in the console. Copy and paste that URL into your browser. This should take to you a webpage that displays another web URL, this is your authentication token that permits you access to the database. Copy and paste the URL into the console in R where it tells you to do so.

```
devtools::install_github("HakaiInstitute/hakai-api-client-r", subdir='hakaiApi')

library('hakaiApi')

# Get the api request client, run this line independently before the rest of the code
client <- hakaiApi::Client$new() # Follow stdout prompts to get an API token

# Make a data request for chlorophyll data
endpoint <- sprintf("%s/%s", client$api_root, "eims/views/output/chlorophyll?limit=50")
data <- client$get(endpoint)

# Print out the data
print(data)
```

By running this code you should see chlorophyll data in your environment. The above code can be modified to select different datasets other than chlorophyll and filter based on different logical parameters you set. This is accomplished by editing the text after the ? in `"eims/views/output/chlorophyll?limit=50"`.

The formula you set after the question mark is known as query string filtering. To learn how to filter your data read this.

To read generally about the API and how to use it for your first time go here.

Here is a more advanced querystring that you could use as a template:

```
chl_endpoint <- sprintf("%s/%s", client$api_root, 'eims/views/output/chlorophyll?date>=2016-11-01&date<20

chl <- client$get(chl_endpoint)
```

Make sure to add &limit=-1 at the end of your query string so that not only the first 20 results are downloaded, but rather everything matching your query string is downloaded.

The page documenting the API usage can be found here

Once I have my data set in my work environment, I'll make sure that all the variable classes are what I want them to be. Luckily, when `read_csv` reads in your file, it prints a summary of the name and type of each variable in your data frame.

I always check:

- that my Date column has the class 'Date'
- that categorical variables are stored as factors
- anything that is not a categorical variable is not stored as a factor but rather is numeric, character, or integer for example.

One of the most common problems of why you get errors in R is from R assigning incorrect variable classes. Often you want your variable to be of class numeric, or character, but it is assigned as a factor. This is extremely common and can be remedied by coercing your variable class, or using tools that are part of the tidyverse

Once your data is imported into R and you are happy with the variable classes, it's time to start wrangling your data into a format you can visualize and model.

5.3.2 Exploratory data analysis

This is where you start playing with the data. You will likely try many different summaries and visualizations while you seek a deeper understanding of your data. Read the Exploratory Data Analysis chapter in R for

Data Science.

I typically create an exploratory data analysis (EDA) script specifically for this, and save all my figures to an exploratory_figures sub-directory using the function `ggsave(here("figures", "figure_1.png"))`. I use R Markdown files (.Rmd) for my EDA. This allows me to weave narrative explanations of what exploration I am doing in each separate code chunk. Writing this narrative about why you chose to explore a certain analysis or summary, and why it did or didn't work is key in documenting your analysis so you can come back to it, make sense of what you've done, and pick up where you left off.

EDA allows me to try many different summaries or visualizations without having to worry about making the output perfect or pretty. This process will give you insight to your data as well as generate new questions and provoke new analyses. One excellent tool that can help you explore your data interactively is using shiny apps. There's too much detail to focus on how to produce shiny apps here but suffice to say that creating interactivity with your data set is an excellent way to explore many different relationships between your data.

Exploratory analysis can probably go on forever depending on the complexity of your data. You will probably think of 1000 different ways to look at your data, but after some time you need to refocus on your original question and think carefully about what analyses and visualizations are best to communicate. Because you've been storing all of your EDA in and EDA script you've got a record of every path you went down while exploring your data. You can then cherry pick the analyses, plots, and summaries you want to include in your final scripts and simply copy and past them into a new .Rmd script.

5.3.3 Predict and Model

The type of statistical modelling you are going to do is dependent on the question you are asking, and is outside the scope of this guide. For some general guidelines on predicting and modeling read the section Model in R for Data Science.

5.3.4 Communicate

What your final data product is going to be (summary report, figures for your manuscript, an interactive dashboard for a website), will dictate what your final scripts will be. As a baseline I recommend .Rmd as the final format because this gives you a lot of flexibility in terms of polished data products.

This is an area where R-Studio really shines. There are so many templates for communicating your analysis avaliable. Check out some of the videos about RMarkdown and read the section in R for Data Science called Communicate to learn about the variety of ways you can communicate your analysis.

Chapter 6

Version Control and Collaboration

Version control is an additional level of saving your files. The old school method of version control is to have multiple versions of the same file on your computer with different dates or initials to identify the version you want to work on (eg. fishy_analysis_V9_2017_05_11_BJ_edits.R, etc…) — this is what version control using Git and GitHub is aims to simplify.

There are two methods that are recommended for version controlling your files: Google Drive and Git/GitHUB. Each has their own advantages and disadvantages and can be used under different circumstances.

6.1 Google Drive

6.1.1 Version Control

Googlesheets, Google Docs, Google Slides, has for me, replaced Excel, Word, and Powerpoint. The ability to collaborate in real time on the same file as someone else has great advantages and solved many of the issues with version controlling edits to a source file. The version controlling is taken care of for you and you can look into the version history to see exactly what and when changes were made and by whom. Essentially everyone works off the master copy (or using git jargon — commits to the master branch). The version history of any Google file (Sheet, Doc, Slides) can found on the browser version of Google Drive by opening the document or sheet and selecting `file` > `version history`. If you'd like to make a specific version more memorable before or after some significant change, you can 'name the version'. The actual name of the file does not change, however the version will receive a name in the version history. This allows you to better keep track of important changes to a file.

Keeping data files and R scripts in Google Drive for version controlling isn't quite as slick as it is for google products (sheets etc.). It does, however, certainly have some advantages compared to emailing colleuages different versions of files back and forth, and renaming them each time.

Everytime you save a file that is located in a Google Drive folder on your computer, it is backed up in the cloud and the version of that file is saved. Managing the version history of files other than Google Sheets, Docs, Slides etc, is a little different. Only the most recent versions of your file will be saved unless you right-click the file in the web version of Google Drive and check the box that says 'Keep Forever'.

Google Drive is a simplified and intuitive version control system that can work well under some scenarios. Google Drive falls short, however, when you and a colleauge are working in a collaborartive development of an .R file. Maybe you are both working on the same .R file to build a complicated analysis, and your part depends on their part so you have to do it in the same script. If there is a chance you could be working on the same script at the same time, then Google Drive will fail you. If you both opened the .R file from

Google Drive at the same time and one person finished making their edits after the other, the first persons edits would be erased as the second person uploads their version. This is one of the major problems that Git and GitHUB solves.

6.2 Git and GitHUB

Similar to the relationship between R and R-Studio, Git is the *local* version control system running in the background on your computer while GitHub is the remote user interface for saving, tracking, and sharing updated versions of your files that are hosted on a remote server. Git and GitHub are integral for tracking the evolution of a set of files, and the development of your analysis.

The additional 'save' that comes from version control using git is known as a *commit*. You save your files like you normally would, but every once in a while you commit your files as an official version to be remembered. A commit can be thought of as a bullet point in the to do list of your analysis, and each commit you make must be accompanied by a message. For example; 'read in data and tidy it up', or 'remove observations from non-standard sampling event, and re-fit GLM'. Git tracks the commits you make in R-Studio locally on your own computer. When you are ready for a series of commits to be made public, you *push* your commits to your *remote* repository at GitHub.

If you're interested in learning why version control is important, I encourage you to skim Chapter 1; 'Why Git? Why GitHub' from Jenny Bryan's Book. I recommend referring back to this book whenever you have a question about using git and GitHub with R-Studio. You can also watch this video for an introduction to Git and GitHub.

6.2.1 Step-by-step Git/GitHub

1) **Setup a GitHub Account**: First, you're going to want to sign up for a GitHub account
2) **Install and set-up Git**: Installing Git locally and getting Git to communicate to your remote GitHub website account, and then getting them to talk with R Studio takes a number of steps to complete…This next part can be painful to work through, but it is 100 % necessary and well worth it in the long-run. Install Git and GitHub and get them talking to R-Studio using Jenny Bryan's Guide, follwow along wth chapter 7, 8, 10, and 13.
3) **Put this all in context**: To learn how to use Git, GitHub and R-Studio I recommend watching the R-Studio Essentials tutorial; GitHub and R-Studio.

If you are working directly for Hakai, you should contact the Ecological Information Management department: eims@hakai.org, to request that you be brought on to the Hakai GitHub team so that your work is stored in the Hakai repositories rather than on your personal account.

6.2.2 Start an R-Studio project under version control

6.2.2.1 Set up a repo on GitHUB

If you are working on a specific program, there's a good chance your program already has a central GitHUB repo set up for all your programs analyses. If you've been asked to conduct some data summaries or analyses you should talk to your PI about storing the code you write in your program's repo. If your program doesn't have a repo, you're going to want to set one up. Make sure you talk to your PI and someone from Hakai EIMS about this.

Whether you're setting up a repo on your own GitHub account or the Hakai one, your going to want to first create the repo, and then create your R-Studio project by following these instructions. If the repo you want to use already exists, then just go straight to the next section: 'Start a new R-Studio project.'

1) Go to https://github.com and login.
2) Create a new repository on your own account, or on the Hakai account depending on the scenario.
3) Call it something-descriptive-but-concise.
4) Make it private to start (talk to your PI or IT about making it Public) and click yes to initialize with a README.
5) Click the big green button "Create repository."
6) Copy the HTTPS clone URL to your clipboard via the green "Clone or Download" button. You'll need this to link the repo with R-Studio.

6.2.2.2 Start a new R-Studio project

To link the gitHUB repo to your Google drive folder and a new project in R-Studio:

1) Open R-Studio and go to File > New Project > Version Control > Git. In the "repository URL" paste the URL of the Git Hub repository you just made. Under project directory name, type in the name of whatever this analysis is, for example "soft sediment spp diversity" — this will be the name of the analysis' directory. Under 'Create project as a sub-directory of' navigate to your programs Google Drive folder for analyses. Click "Create Project".
2) Go to Tools in the menu bar, and navigate to Project Options. Click Packrat, and then check the box 'Use Packrat with this project'. This may take a few minutes to initialize, but is important to make your analyses reproducible in the long term.

This should download the README.md file that we created on GitHub in the previous step. Look in R Studio's file browser pane for the README.md file."

6.2.3 GitKraken

When you first start, using the git feature directly in R-Studio is more than adequate. If however, you'd like more features, then I recommend the following software and workflows.

GitKraken is a graphical user interface that allows you to visualize your repositories branches and history. The Hakai IT team uses this software when developing tools, and it works quite well for backing up and monitoring the development of an analysis as well. Check out GitKraken to see if it's something you'd like to use.

6.2.3.1 Git Flow

Git Flow is a workflow that can be implemented in GitKraken by going to GitKraken > Preferences > GitFlow and then initialize gitflow. The workflow is generally good whether working individually or collaboratively. Some will argue that there's no such thing as working individually, especially when you consider that you're constantly collaborating with past-you, and future-you.

Some terms related to version control and Git Flow:

- **Branching** - This is a core concept in Git Hub and version control in general – The master branch must always remain stable and working. When you initialize git flow it creates a development branch. Essentially, you are creating an environment where you can try out new ideas without directly manipulating the stable 'master' version of your analysis. When you are happy with the changes you *push* the development branch to the master branch.

- **Commits** - Commits represent significant changes to your branch. I think of them as mini-milestones that add up to your complete analysis. Perhaps you're mini-milestone is to create a linear model of something. You can work locally and make save files on your computer while working on the linear model like you normally would. When you finish your linear model, this is a good time to make a commit to your branch. You must name each commit, and this should be descriptive. In this case 'fitted linear model' would be ideal.

To learn more about git flow check out this article

6.2.4 Fork and Pull

If someone from outside the Hakai Institute GitHUB account wants to review your analysis and suggest changes, make additions, or generally collaborate with you, you can use a different workflow known as the fork and pull model.

- **Fork** - This is the process of copying someones repository and creating a fork in the development branch where you can work the forked copy all you want without affecting the original code. When you're happy with the changes you made to the repo, you can make a pull request.
- **Pull Requests** - I like to think of pull requests as an opportunity to review and assess external changes to the repo before you may decide to merge the fork back into the main branch. Something that helped clarify what a pull request is for me, was thinking about the difference between a push (write) request, and a pull (read) request. In the context of a shared repository, a pull request gives everyone that the repo is shared with a change to *read* your request to consider the sum of your mini-milestones. Pull requests should amount to the completion of a component of your project. See here for more info.
- **Merge** - Once a Pull Request has been sufficiently reviewed, discussed, and tested (deployed), the new code can be merged with the master branch, typically by the team owner, or maintainer.
- **Issues** - You can create an issue to raise a bug report, suggest some functionality, or discuss the objectives of a certain project.

6.3 Best Practices

6.3.1 Citing analyses

- Citing analyses can be done as text files that get included with data packages that can be distributed with manuscript.
- Hakai has the ability to zip scripts and data files from a gitHUB repo and assign a DOI to it with a metadata entry. This isn't mutually exclusive to maintaining a GitHUB repo and pointing collaborators to archived versions of a repo.

6.3.2 Private v. public repos

- We don't really want anonymous use of our analyses, so it's best to use private repos and add external collaborators if we want them to review or contribute to our analysis.

6.3.3 Granularity of repos

- Best practice is to not make repos too small.

- In cases where you want to cite an analysis, need finer control of access permissions, or when a tool requires a separate repo (eg. creating an R package) you'll have to create a separate repo.
- There are ways to take folders out of one repo to create an independent repo which retains commit history. Therefore, if you are uncertain about whether you need to split into a different repo, it's best to lump folders and split later if need be.
- Managing repos with gitHUB 'teams' is a good workflow.

6.3.4 Code Licensing

- It's a good idea to license with MIT license to permit commercial use.
- This license should be included in every script or data package, but we will still control who has access to code and data by requiring them to request access to data packages or scripts.

Chapter 7

Mapping in R

7.1 Leaflet

It is possible to use the Javascript based widget known as Leaflet from within R. This is great for making some simple maps in an interactive R Markdown document, as well as in Shiny apps. You can use basemaps that Hakai hosts from the ArcGIS Online Server (AGOL), or you can use the basic maps that are provided. You are able to add points, visualize heat maps, legends, cloropleths, and other simple GIS tasks.

To do this from R you need to:

```r
# The package that allows you add leaflet map widgets to your interactive web documents:
install.packages('leaflet')

# The package that allows you to use map tiles from Hakai's AGOL
install.packages('leaflet.esri')
```

Follow the instructions on R-Studio's Leaflet package webpage to create your map. The included basemaps can be viewed here

To use the basemaps on Hakai's AGOL server use the `addEsriTiledMapLayer()` function from the `esri.leaflet` package where the documentation can be read here

Hakai ArcGIS Online maps can be viewed here. You can first view any of these basemaps in AGOL to decide which you'd like to use. Once you've decided, you simply copy the URL and paste it into the `addEsriTileMapLayer()` function

Example:

```r
library(leaflet)
library(leaflet.esri)
# library(magrittr)
#  leaflet() %>%
#         # Set the default lat and long and zoom level
#         setView(lng = -126, lat = 50.3601, zoom = 7) %>%
#         # Add default tile
#         addTiles() %>%
#         # Add tile from Hakai AGOL
#         addEsriTiledMapLayer(url = "https://ags.hakai.org:6443/arcgis/rest/services/AGOL_basemaps/Mar
```

Chapter 8

Resources and References

The one and only introductory R book to read from front to back:

- R For Data Science

Workflows for Scientific Computing

- Good Enough Practices for Scientific Computing
- Best Practices for Scientific Computing

Functional Programming (Loops and stuff)

- Purr Tutorial
- Advanced R

Git and GitHub

- Happy Git with R by Jenny Bryan

Plots

- Cookbook for R

Statistical Modeling Examples

- UCLA Institute for Digital Research and Education
- Generalized Linear Models Tutorial by Sean Anderson

Statistics

- Undergraduate Biologist's Guide with R Examples

Tidyverse

- Tidyverse Website
- Tidyverse style guide

Chapter 9

Glossary of Terms

Branching: From GitHUB: “A branch is a parallel version of a repository. It is contained within the repository, but does not affect the primary or master branch allowing you to work freely without disrupting the”live” version. When you’ve made the changes you want to make, you can merge your branch back into the master branch to publish your changes.”

Commit: From GitHub: “A commit, or”revision“, is an individual change to a file (or set of files). It’s like when you save a file, except with Git, every time you save it creates a unique ID (a.k.a. the”SHA” or “hash”) that allows you to keep record of what changes were made when and by who. Commits usually contain a commit message which is a brief description of what changes were made.”

Developing Analyses: A term coined by Hillary Parker that describes the process by which a data anlaysis is ensured to be reproducible, accurate and collaborative.

Distributed Version Control: A system of tracking the changes in a set of files that allows devlopers to work on locally stored copies of the same set of files, before sharing the changes with a remote copy of the set of files.

Fork:From GitHub: “A fork is a personal copy of another user’s repository that lives on your account. Forks allow you to freely make changes to a project without affecting the original. Forks remain attached to the original, allowing you to submit a pull request to the original’s author to update with your changes. You can also keep your fork up to date by pulling in updates from the original.”

Issues: From GitHub: “Issues are suggested improvements, tasks or questions related to the repository. Issues can be created by anyone (for public repositories), and are moderated by repository collaborators. Each issue contains its own discussion forum, can be labeled and assigned to a user.”

Local: The set of your repositories files that are stored on your own computer.

Literate Programming: A programming paradigm invented by Donals Knuth that puts the emphasis on human readability of code and flow of code structure that is logical to the human.

Merge: From GitHub: “Merging takes the changes from one branch (in the same repository or from a fork), and applies them into another. This often happens as a pull request (which can be thought of as a request to merge), or via the command line. A merge can be done automatically via a pull request via the GitHub web interface if there are no conflicting changes, or can always be done via the command line. For more information, see”Merging a pull request.”

Remote: From GitHub: “This is the version of something that is hosted on a server, most likely GitHub. It can be connected to local clones so that changes can be synced.”

Pull Request: From GitHub: “Pull requests are proposed changes to a repository submitted by a user and accepted or rejected by a repository’s collaborators. Like issues, pull requests each have their own discussion forum.”

Push: From GitHub: “Pushing refers to sending your committed changes to a remote repository, such as a repository hosted on GitHub. For instance, if you change something locally, you’d want to then push those changes so that others may access them.”

Working Directory: The folder in which your R session is reading and writing files from and to.