ADF&G Division of Sport Fisheries
Introduction to Git

Adam Reimer

2023-06-26

Table of contents

Preface

The author is aware that each analyst has a unique level of expertise and each analysis has unique requirements. As the analyst it is up to you to use professional judgment to decide which techniques are useful for the analysis you are conducting. That said, we offer the following general advice.

- Best practice documents tend to describe appropriate practices for the largest, most complex analyses you are likely to face. If you project is simple, the overhead associated with some of these techniques may not be justified. Use your professional judgement to discern how to follow the spirit of reproducible research while modifing this guidance to specific situations you encounter. The author understands that and applies similar discretion his work.
- This document represent the collective experience of XX years of work at ADF&G. Our vision is that this guidance will be followed closely if you have minimal prior experience. Younger employees introducing and adopting modern research practices is one of the key ways a large agency can keep current over time. We recognize seasoned employees have reproducible workflows they are happy with and may be hesitant to learn new techniques. We would encourage those employees to try some of these techniques on new projects and to migrate existing projects at opportune times and/or prior to transiting a project to an different analyst.

Division of Sport Fish Github Transition

Biometricans with ADF&G Sport fish will be required to store their analysis on GitHub starting in 2024. This means that all operations plans and reports published in 2024 and later will be required to reference a GitHub site and a specific commit where the published analysis is stored. This decision was reached by a consensus of Biometrican IIIs and will streamline final biometric reviews and transitioning projects between biometrics staff. This document was produced to help provide guidance for satisfying this requirement.

1 Introduction

There is a a lot of git stuff out there and most of it is not intended for scientists (one of the best science focused git tutorials for R is HappygitwithR). In this document our goal to amalgamate the most common Git commands and strategies into a workflow that is helpful for ADF&G fisheries analysis.

1.1 What does Git do?

Git offers a way to track changes in your analysis without requiring the analyst to create different versions of the same file. To use git an analyst **init**ializes their working directory (hopefully an R project). Files involved in the analysis (data, scripts, functions, model code) are **add**ed so that git knows to track changes associated with each file. When the analyst makes a **commit**, a snapshot of all tracked files at that specific point in time are recorded along with a message describing the commit and an automatically assigned a unique identifier. The analyst can also **tag** important commits. Because you can **checkout** prior commits this system allows for traditional file versioning with a structured system while ensuring the all commits are documented and the most important commits are easily identifiable.

The collection of all the commits, messages, tags and identifiers associated with a projects is called a **repository**. When a repository is created on your computer or private/company network it is **local**. An analyst can **push** a local repository to a **remote** repository (stored on the cloud). Alternatively, the analyst can **pull** a remote repository to their computer or private/company network to either create or update a local repository. Because multiple local repositories can push and pull to the same remote repository Git allows collaboration between analysts while maintaining the documentation and unique identifiers associated with each commit. Github is the most popular hosting service facilitating these collaborative features of Git.

1.2 How do I interact with Git?

One problem with widespread adoption of Git at ADF&G is that there is no accepted standard for how to interact with Git. The options are a GUI (graphical user interface) or a command line interface (terminal). Herein, we will focus on how to use Git while interacting through the terminal. We make this choice for 3 reasons. First, because the terminal is a command line

interface it is easy to demonstrate the exact steps that were taken to achieve each outcome. This presentation is more concrete than a parade of screenshots. A second issue is that there are a lot of GUIs out there. I will include an appendix which shows how to do the most common Git functions in RStudio. While not a particularly good Git GUI the RStudio interface is sufficient for most tasks and readily available for most readers of this book. If you have a GUI you highly recommend we welcome your input. Please fork the repository associated with this book, add an appendix with instructions for your prefered GUI and submit a pull request… instructions on how to do so are included in the chapter on collaboration. A final reason to use command line in the context of this book is that the terminal commands are commonly described in Stack Overflow if you Google how to accomplish a task in Git. Rest assured, I don't use the terminal to interact with Git the majority of the time[1] and I don't recommend you do either. But I do think you will end up there eventually, that the terminal has pedagogical advantages and that if you can use the terminal the GUI's are easier to understand.

In order to use the terminal effectively it helps to make one change to the Rstudio defaults by executing the following point and click commands: *Tools>Terminal>Terminal Options…>(change initial directory to 'project directory')*. This change will ensure your terminal opens in the correct directory and save some unnecessary terminal commands.

1.3 Conventions used in this book

Throughout this document you will find code blocks which show the command line prompt, the command given, and the result received for each action demonstrated. Code blocks are identifiable by a grey bar along the left hand margin. In the terminal, each command line prompt ($) is proceeded by the username, shell type, and directory location. In the terminal session below we can see that the username was amreimer@DFGSXQDSF206801, the shell type was MING64 and the directory location was S:/RTS/Reimer/Research_Best_Practices/git_practice.

```
amreimer@DFGSXQDSF206801 MINGW64 /s/RTS/Reimer/Research_Best_Practices/git_practice
$
```

When point and click sequences are referenced in this text button names will be shown in *italic*, button names in series will be separated by and the greater than symbol (>) and any actions required will be described using (*italics enclosed in parenthesis*). For an example see the last paragraph of the preceding section.

When git commands are referenced in the text they will be displayed as an `inline code block`. R objects and Git branches are also referenced as `inline code blocks`.

[1] I generally use the RStudio GUI first, the terminal when the RStudio GUI is not sufficient and Git Kraken when I have a lot of branches to visualize/review.

2 The Basic Git Workflow

Basic Git use includes how to create a repository, track changes in the files within your repository, view your repository history, and view line by line changes in modified files. The title for each section in this chapter will be a basic Git command with the text, code blocks, and figures in each section describing the use and result for each command.

2.0.1 git init

If you have an empty folder that you would like make a git repository `git init` is the appropriate command. In the example below I have an empty folder named "git_practice" on S drive under S:/RTS/Reimer/Research_best_practices. The terminal session below shows 3 commands and the output received after each command: 1) verify the directory is not a git repository with `git status`, 2) make it a git repository with `git init`, 3) verify the directory is now a git repository with `git status`.

```
amreimer@DFGSXQDSF223076 MINGW64 /s/RTS/Reimer/Research_Best_Practices/git_practice
$ git status
fatal: not a git repository (or any of the parent directories): .git

amreimer@DFGSXQDSF223076 MINGW64 /s/RTS/Reimer/Research_Best_Practices/git_practice
$ git init
Initialized empty Git repository in //dfg.alaska.local/DSF/Anchorage/RTS/Reimer/Research_B

amreimer@DFGSXQDSF223076 MINGW64 /s/RTS/Reimer/Research_Best_Practices/git_practice ({main
$ git status
On branch {main}

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        git_practice.Rproj
```