R, RMarkdown, Git, and GitHub for Academic Writing: A Guide

Sergios (Sergey) Charntikov

2017-04-12

Contents

Chapter 1

Preface

The goal for this guide is to provide a comprehensive workflow for devoloping and publishing academic manuscripts with the help of R, Rmarkdown, Git, and GitHub.

1.1 Structure of the manual

This manual is structured in a series of chapters outlining a process of developing a complete submission ready draft of an academic manuscript. In the initial chapters you will learn how to setup your machine to implement proposed workflow, how to setup a project directory to efficiently organize all relevant files, and how to write and a complete manuscript using minimal markup environment. Later you will learn how to harness power of Git to manage incremental development of your manuscript using GitFlow approach (http://jeffkreeftmeijer.com/2010/why-arent-you-using-git-flow/). We will then move to practical solutions of embedding data visualization and data analysis code in the body of your manuscript file to minimize human error and improve overall reproducibility of your study. Finally, we will outline final steps leading to submission of your manuscript including organization of supplemental materials, submission guide, and a complete template to implement the workflow outlined in this manual.

1.2 Software information

R session information when compiling this book is shown below to ensure reproducibility of all the steps outlined in this manual.

```
sessionInfo()
```

```
## R version 3.3.1 (2016-06-21)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 10240)
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
```

```
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] backports_1.0.5 bookdown_0.3    magrittr_1.5    rprojroot_1.2
##  [5] tools_3.3.1     htmltools_0.3.5 rstudioapi_0.6  yaml_2.1.14
##  [9] Rcpp_0.12.10    stringi_1.1.5   rmarkdown_1.4   knitr_1.15.1
## [13] stringr_1.2.0   digest_0.6.12   evaluate_0.10
```

1.3 Acknowledgments

I would like to thank Yihui Xie and Hadley Wickham, of RStudio, for inspiring me to develop this manual and for developing tools that empower scientist to better communicate their scientific findings.

1.4 About the author

Chapter 2

Prerequisites

Here is what you need to implement the workflow outlined in this manual.

Software

- R - https://www.r-project.org/
- RStudio - https://www.rstudio.com/
- Git - https://git-scm.com/downloads
- Optional Git GUI client - https://git-scm.com/downloads
- Optional GitHub Account - https://github.com/
- Optional free private repositories from GitHub for education - https://education.github.com/discount_requests/new
- Optional modern and "hackable" text editor - https://atom.io/

Chapter 3

Rmarkdown workflow

3.1 RMarkdown Template

To write this manuscript we are using Rmarkdown Elsevier template. The `rticles` package provides a suite of custom R Markdown LaTeX formats and templates for various formats, including Elsevier journal template.

Under the hood, LaTeX templates are used to ensure that documents conform precisely to submission standards. At the same time, composition and formatting can be done using lightweight markdown syntax, and R code and its output can be seamlessly included using knitr.

Using `rticles` requires the prerequisites that will be automatically installed with the latest release of RStudio. Please make sure that you are using latest version of RStudio and update all the packages in your package library.

`rticles` - https://github.com/rstudio/rticles

3.1.1 Installation

You can install and use `rticles` from CRAN as follows:

```
install.packages("rticles", type = "source")
```

If you wish to install the development version from GitHub you can do this:

```
devtools::install_github("rstudio/rticles")
```

3.1.2 Using `rticles` from RStudio

- Install latest RStudio
- Install the `rticles` package
  - `install.packages("rticles", type = "source")`
- RStudio::File-New R Markdown-From Template-Elsevier Journal Article

3.1.3 YAML

YAML is the preamble found at the top of the .Rmd file. YAML is used to configure front matter of the article and some other important technical parameters of your document.

Front matter:

```
title:
author:
address:
abstract:
```

Citations and bibliography:

```
bibliography: file_with_your_citations.bib
csl: biomed-central.csl
```

Acceptable bibliography files: mods, bibtex, ris, enl, xml, medline, copac, json.

`csl` defines citation style if other than standard Elsevier format is needed (see https://www.zotero.org/styles for repository of current available styles.).

Cross-referencing

```
output:
   bookdown::pdf_book:
    base_format: rticles::elsevier_article
```

3.2 Writing in markdown

3.2.1 Overview

Writing in markdown is fairly easy even without prior knowledge of other markup languages. Markdown is a lightweight markup language design so it can be converted to many other formats including LaTex - a format commonly used for typesetting academic typography. Writing in markdown is much more pleasing than writing in LaTex because of the minimal markup. Because of the minimal markup, documents written in markdown can be easily read and edited without code distractions. Furthermore, because markdown documents are written in plain text they can be edited with a plain text editor which are freely available on all major platforms. There are many specially designed editors with additional features that may be more attractive to different groups of developers or writers working with markdown documents. Editors that work great for writing academic style documents include RStudio, Atom, and Sublime Text.

3.2.2 Markdown basics

Emphasis

```
*italic*   **bold**

_italic_   __bold__
```

Headers

```
# Header 1

## Header 2

### Header 3
```

Lists

```
* Item 1
* Item 2
```

```
    + Item 2a (note: indent by 4 spaces before +)
    + Item 2b
```

Images

Images on the web or local files in the same directory:

```
![](http://example.com/logo.png)
```

```
![optional caption text](figures/img.png)
```

More on Markdown basics can found on the excellent RStudio website: http://rmarkdown.rstudio.com/authoring_basics.html

3.3 LaTex

Any LaTex code can be included in the markdown document. LaTex code included in the `.Rmd` file will be processed ("knitted") by RStudio's `knitr` package and then passed to `pandoc` document converter. In this process, `knitr` takes plain text document with markup and code, executes embedded commands, and then compiles results into a separate document which is then passed to `pandoc` for final conversion. In the workflow relevant to writing academic manuscript using raw `.Rmd` file, `.Rmd` file is first converted to `.tex` and then the resulting `.tex` file is converted to .pdf.

3.4 Tables

3.4.1 Plain markdown table

Adding tables to a manuscript is easy with plain markdown. See an example below of how you can

```
Header 1 | Header 2
-------------- | -------------
Cell 1 | Cell 2
Cell 3 | Cell 4
```

When creating a table in markdown there is no need to perfectly align all separators.

The code above will generate this table:

Header 1	Header 2
Cell 1	Cell 2
Cell 3	Cell 4

3.4.2 LaTex tables

LaTex tables can be embedded into `.Rmd` markdown file. LaTex tables can be generated using popular R packages like `stargazer` or `xtable`. LaTex tables can be also generated by numerous free online conversion or table generation services (just google it).

3.4.3 R code generated tables

To reference a table a table needs to be included in the r chunk with the chunk label.

```
```{r cars-table}
library("knitr")

kable(
  head(mtcars[, 1:8], 10), booktabs = TRUE,
  caption = 'A table of the first 10 rows of the mtcars data.'
)
```
```

In the example above the chunk label is `cars-table` so to reference this table use prefix `tab:` plus the chunk label - `tab:cars-table`. Here is the referencing in action (see Table `\@ref(tab:cars-table)`).

Using this approach tables will be generated at the end of the manuscript accompanied with an automatic generation of the "List of Tables" (hyperlinks in all locations included).

3.5 Figures

Adding figures to your manuscript can be done by inserting pre-generated images or by generating figures right from the embedded R code. It is highly advisable to adapt the latter approach of generating figures from embedded r code to enhance reproducibility of your findings and to minimize common versioning errors (unintended inclusion of earlier or inaccurate version of the figure).

3.5.1 Inserting an image from a file

```
![caption for my image](path/to/image.jpg)
```

It is possible to include an image from any accessible directory. However, it is advisable to have a /figures or /images folder in your project directory with all the figures or images used for the project.

Elsevier template will automatically generate figure caption. If you are using other template to typeset your manuscript then you may need to add `fig_caption: yes` YAML option to render a caption.

3.5.2 R code generated figures

It is advisable to generate figures directly from a code embedded in the manuscript. One of the options to do that would be to include R code for each figure in each separate `figure` environment (chunk) at the very end of your manuscript. This way each figure `chunk` can have reference label and a figure captions (e.g., `{r nice-fig, fig.cap='Here is a nice figure!'}`). Chunk parameters also can be used fine tune apperance of the figure in the manuscritp (e.g., `out.width='80%', fig.asp=.75, fig.align='center'`).

For example:

```
```{r nice-fig, fig.cap='Here is a nice figure!', out.width='80%',
fig.asp=.75, fig.align='center'}
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```
```

Figure can be referenced by its code chunk label with the `fig:` prefix, e.g., see Figure `\@ref(fig:nice-fig)`.

The following guideline is from https://bookdown.org/yihui/bookdown/

> If you want to cross-reference figures or tables generated from a code chunk, please make sure the chunk label only contains alphanumeric characters (a-z, A-Z, 0-9), slashes (/), or dashes (-).

3.5.3 Crossreferensing tips

To cross reference figures or tables generated within a code chunk labels within the chunk should only contain alphanumeric characters (a-z, A-Z, 0-9), slashes (/), or dashes (-). For example `{r nice-fig` and not `{r nice.fig` or `{r nice&fig`.

3.5.4 Setting aspect ratio of plots

The aspect ratio of a figure is the ratio of its longer side to its shorter side - the with to height ratio. The golden ratio (1.618) is the ratio that has been appearing in ancient artwork as early as 2,400 BCE and fascinated many ancient mathematicians and artists alike. Edward Tufte suggested that "[g]raphics should tend toward the horizontal" and proposed the use of Golden ratio to create visually appealing figures (Tufte, E. R. 2001). Aspect ratio in R chunk can be set by using `fig.asp` command. For example, to set a Golden ratio for a figure (1/1.618=0.618) you would type `fig.asp=0.618`

> The chunk option `fig.asp` can be used to set the aspect ratio of plots, i.e., the ratio of figure height/width. If the figure width is 6 inches (`fig.width = 6`) and `fig.asp = 0.7`, the figure height will be automatically calculated from `fig.width * fig.asp = 6 * 0.7 = 4.2`.
>
> The actual size of a plot is determined by the chunk options fig.width and fig.height (the size of the plot generated from a graphical device), and we can specify the output size of plots via the chunk options out.width and out.height. The possible value of these two options depends on the output format of the document. For example, `out.width = '30%'` is a valid value for HTML output, but not for LaTeX/PDF output. However, knitr will automatically convert a percentage value for out.width of the form x% to (x / 100) `\linewidth`, e.g., `out.width = '70%'` will be treated as `.7\linewidth` when the output format is LaTeX. This makes it possible to specify a relative width of a plot in a consistent manner.

Chapter 4

GitFlow for Manuscript Preparation

4.1 Why use Git for manuscript preparation?

Git is a free and open source version control system. Git is designed to efficiently handle small and large project whether your are working on PC, Mac, or Linux. Git features make is a versatile companion for any size of the writing project but it especially stands out in its ability to handle complex writing projects like an original research manuscript for example. Git allows and encourages of creation of multiple local or remote branches that can be used for independent development of project features. When branches mature, or when features are fully developed, they can be seamlessly merged into the main branch called `master`. Thus, new branches can be conceptualized as a *sandbox* for new feature where user can freely explore a hypothetical scenario of developing this new feature without modifying content of a `master` branch (context of a main project or a file). This ability to easily create new branches and seamlessly switch between branches facilitates more efficient workflow for writing complex documents like a research manuscript. Although this method requires additional time investment to master this process, it is likely that many researchers will find this investment beneficial in the long term.

4.2 General overview of project structure over time

Research manuscripts are an example of

4.3 General overview of team workflow

Content coming soon.

4.4 General outline

- `master` branch will be used ONLY for submission quality files
- `rough_draft` branch will be used ONLY for final draft quality work
- we will use local branches to develop content for `rough_draft` stage and merge to `rough_draft` through pull requests only
- we will merge to `master` branch only when we have `rough_draft` ready for submission
- we will exclude `*.pdf` files from git tracking by adding them to .gitignore‘ file
- we will exclude `images/*` directory from git tracking by adding this directory to `.gitignore` file

- `.gitignore` file should be in the main directory with the following exceptions:
  - .Rproj.user
  - .Rhistory
  - .RData
  - .Ruserdata
  - Data-Visualization-Analysis/figures/*
  - *.pdf
  - .Rproj

4.5 Practical appliation to a manuscript development

The folowing visual representation of a GitFlow was heavily adapted from: http://nvie.com/posts/a-successful-git-branching-model/

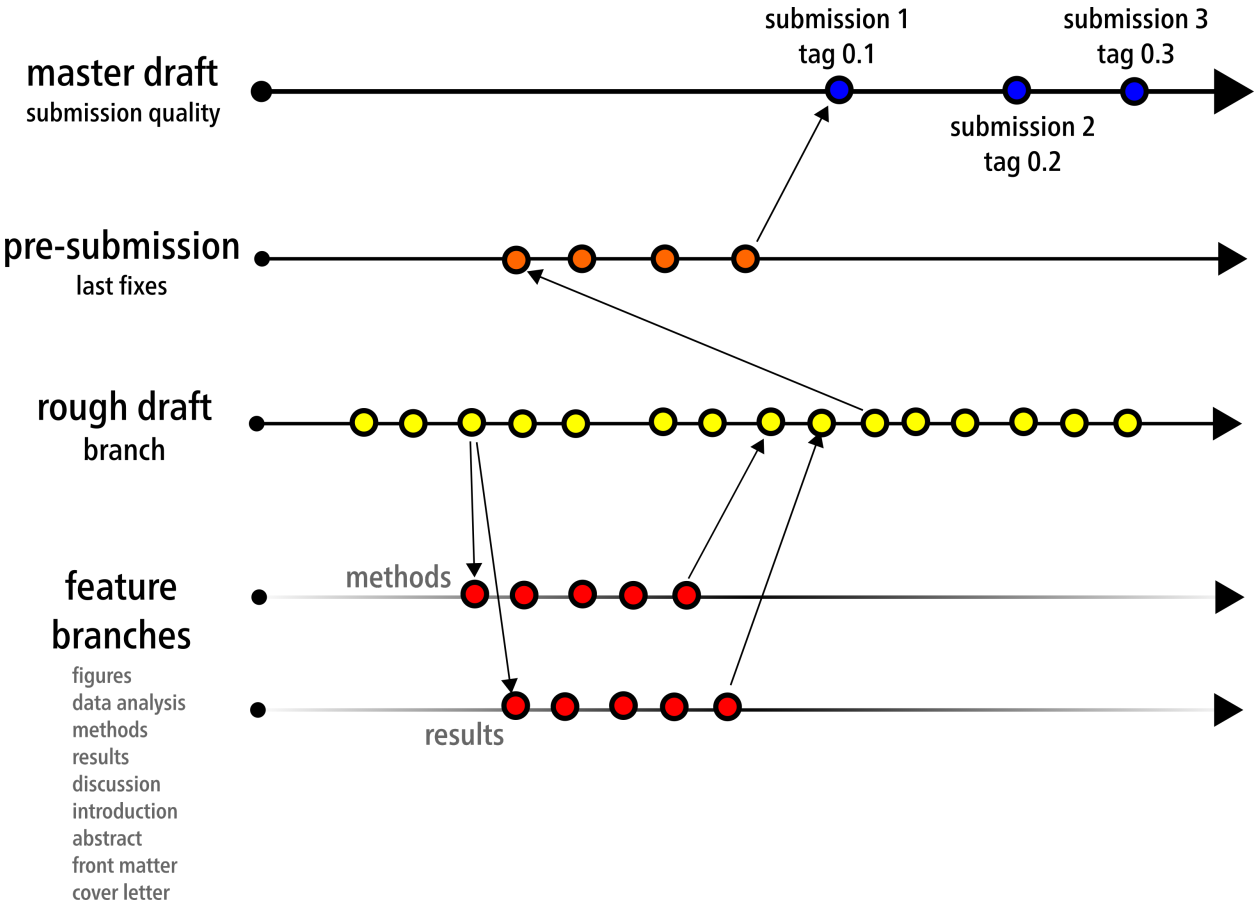

Figure 4.1: GitFlow for development of a Research Manuscript

4.6 Main branches

The central repo holds two branches with an infinite lifetime:

- `master` (publication ready state)
- `rough-draft` (latest delivered development state)

When `rough-draft` branch is reached stable state suitable for submission to a journal, all of the changes are merged into `master` branch and tagged with a manuscript release number.

4.7 Supporting branches

Supporting branches have limited life time and will be removed eventually.

Two different types of branches we may use:

- Feature branches
- Pre-submission branches

4.7.1 Feature branches

Feature branches are used to develop a specific part of the manuscript. These are the branches where the most work is done by the members of the team. These branches are created locally on the machines where these features are being develop and they are merged through Pull requests only into rough-draft branch. These branches are short lived and will follow the following convention:

May branch off from:

- `rough-draft`

Must merge back into:

- `rough-draft`

Branch naming convention:

- anything except master, rough draft, pre-submission

4.7.1.1 Creating a feature branch

When starting work on a new feature, branch off from the `rough-draft` branch.

$ git checkout -b myfeature rough-draft

Switched to a new branch “myfeature”‘

4.7.1.2 Incorporating a finished feature on develop

Finished features may be merged into the develop branch to definitely add them to the upcoming release:

$ git checkout develop

Switched to branch ‘develop’

$ git merge –no-ff myfeature

Updating ea1b82a..05e9557

(Summary of changes)

$ git branch -d myfeature

Deleted branch myfeature (was 05e9557).

$ git push origin develop

The `--no-ff` flag causes the merge to always create a new commit object, even if the merge could be performed with a fast-forward. This avoids losing information about the historical existence of a feature branch and groups together all commits that together added the feature.

In the latter case, it is impossible to see from the Git history which of the commit objects together have implemented a feature—you would have to manually read all the log messages. Reverting a whole feature (i.e. a group of commits), is a true headache in the latter situation, whereas it is easily done if the –no-ff flag was used.

Yes, it will create a few more (empty) commit objects, but the gain is much bigger than the cost.

4.7.2 Pre-submission branch

May branch off from:

`rough-draft`

Must merge back into:

`master`

Branch naming convention:

`pre-submission*`

This branch is to bring a complete rough draft to a specific publication standard. Various journals will have variuos requirements. This branch will branch of a complete rough-draft and will merge only into master. A submission will be residing in the master branch.

4.7.2.1 Creating pre-submission branch

$ git checkout -b `presubmission` rough-draft

Switched to a new branch “presubmission”

$ ./bump-version.sh 0.1

Files modified successfully, manuscript version bumped to 0.1.

$ git commit -a -m “Bumped manuscript version number to 0.1”

[presubmission 41e61bb] Bumped version number to 0.1

1 files changed, 1 insertions(+), 1 deletions(-)

Don’t forget to bump the version number after branching off!

Then, bring the manuscript to a submission ready state in one or more separate commits.

$ git commit -m “Manuscript is brought to submission ready state for sumbmission in …”

[presubmission abbe5d6] Manuscript is brought to submission ready state for sumbmission in …

5 files changed, 32 insertions(+), 17 deletions(-)

4.7.2.1.1 Finishing presubmission branch

When finished, the `presubmission` branch needs to be merged into `master`.

First, update master and tag the release.

$ git checkout master

```
Switched to branch 'master'
```

$ git merge –no-ff presubmission

```
Merge made by recursive.
(Summary of changes)
```

$ git tag -a 0.1

Finally, remove the temporary `presubmission` branch:

$ git branch -d presubmission

```
Deleted branch presubmission (was abbe5d6).
```

Chapter 5

Data Visualization

Chapter 6

Data Analysis

Chapter 7

Project Organization

Chapter 8

Manuscript Template

Chapter 9

Supplemental Materials

Chapter 10

Manuscript Submission

Bibliography