Reproducible Research in R

2019-07-24

Contents

Reproducible Research in R

- Level: beginner-intermediate
- Duration: 6 hours
- Student numbers: 25-30

Welcome to the Reproducible Research in R (RRR)[1] workshop. The main aim of this workshop is to set you on the right path of making your research more reproducible and shariable. Reproducible research means that future you and anyone else will be able to pick up your analysis and reproduce the same results, including figures and tables. Reproducible research also implies well-documented research, your code should be well commented and the reasons behind functions and methods should be explained thoroughly throughout the analysis. The communication aspect should not be an afterthought, it should be recorded with your analysis as you are going through it. Rmarkdown is a way of literate programing[2] that keeps code, words and sentences together. The ability to easily collaborate and share your analysis goes hand-in-hand with good record-keeping and reproducibility. We are going to repurpose the git version control tool and leverage the GitHub remote hosting provider for managing and sharing our work. Git + GitHub will provide a very powerful resource for global collaboration and exposure of your work. In this workshop, we are going to version control our work and push it to github, which can then be accessed by your collaborators and supervisors. Git + GitHub should become an integral part of your workflow.

The RRR course given by the Monash Bioinformatics Platform[3] for the Monash Data Fluency[4] initiative. Our teaching style is based on the style of The Carpentries[5].

[1] https://github.com/MonashDataFluency/r-rep-res
[2] http://www.literateprogramming.com/knuthweb.pdf
[3] https://www.monash.edu/researchinfrastructure/bioinformatics
[4] https://monashdatafluency.github.io/
[5] https://carpentries.org/

Learning outcomes

Attendees will learn how to:

- write vanilla markdown, Rmarkdown and bookdown documents
- use `knitr`, `rmarkdown` and `bookdown` R packages to build various document types including PDF, HTML and DOCX
- create reproducible Rmarkdown documents leveraging `.Rproj` and `.RData`
- include in-line citation and full references list in to Rmarkdown document using `.bib` files
- create presentations from Rmarkdown documents that include R code
- work with `git` version control tool
- create reproducible and "backed up" analysis via remote repositories (e.g github)

Workshop description

This workshop is an introduction to writing and communicating research using Rmarkdown. Rmarkdown is an easy way to create documents that include your R code and its output, such figure and tables. Rmarkdown is a single document that can be "knitted" and shared as various document types such as PDF and HTML. Rmarkdown supports scientific writing such as use of citations and figure cross-referencing. Rmarkdown can also be used to create presentations that include your R code and its output. We will also cover bookdown, which is an extension to Rmarkdown that allows the creation of larger documents, such as books with multiple chapters.

In this workshop, we will also cover git version control tool[6] which can help with organising and "checkpointing" Rmarkdown documents, associated R code and data. Git is not a backup system, but it does allow one to retrieve older versions of your work. Git, together with remote repositories like GitHub[7] can provide a centralised location for your research. Together Rmarkdown, git and github can enable reproducible research that is visible and accessible to the greater public, including supervisors and management.

Prerequisite

This is an introductory level workshop, however some prior exposure to R and familiarity with RStudio is assumed. It is highly recommended that you read this article in full[8], if you have to prioritise, read at least these section (1,2,6,10).

[6] https://git-scm.com/book/en/v1/Getting-Started-About-Version-Control
[7] https://github.com
[8] https://peerj.com/preprints/3159/

Additionally, it is highly recommended that you create an account at GitHub[9] and remember your password.

Keywords

- R
- Rmarkdown
- communication
- reproducibility
- git and github

Schedule

10:00-10:30am (30 minutes) Welcome and warm up

```
\(   )/
```

10:30-12:00pm (1.5 hours) Rmarkdown

- Introduction (30 minutes)
- Vanilla markdown (30 minutes)
- Rmarkdown (30 minutes)

12:00-1:00pm (1 hour) lunch

1:00-3:00pm (2 hours) More Rmarkdown

- Git and GitHub markdown (40 minutes)
- More Rmarkdown (40 minutes)
- YAML header (40 minutes)

3:00-3:15pm (15 minutes) Tea break

3:15-4:45pm (1.5 hours) Even more Rmarkdown

- Bibliographies (30 minutes)
- Bookdown (30 minutes)
- Miscellaneous (30 minutes)

4:45-5:00pm (15 minutes) Warm down

[9] https://github.com

Authors and copyright

This course is developed for the Monash Bioinformatics Platform by:

- Paul Harrison[10]
- Adele Barugahare[11]
- Kirill Tsyganov[12]

[10] mailto:paul.harrison@monash.edu
[11] mailto:Adele.Barugahare@monash.edu
[12] mailto:kirill.tsyganov@monash.edu
[13] http://creativecommons.org/licenses/by/4.0/

Part I

Part

In this part of the book we will begin by discussing the Rmarkdown language, where it originated, and the ecosystem that surrounds it. We will look at the core parts of an Rmarkdown document, and review the basics of vanilla markdown as a building block of Rmarkdown. This section will conclude with hands-on experience by writing an Rmarkdown document with embedded R code chunks.

Chapter 1

Introduction to Rmarkdown

Rmarkdown has become much more then just embedding R code into a document. It enables construction of very sophisticated document types from plain text files. Rmarkdown files can become pdf documents and a static website at the same time. It can turn your analysis scattered across several different Rmarkdown documents into a single, multi-paged book with cross-referencing and citations, let's call it a thesis, or a paper, or a course book. With essentially no effort, you can change from "rendering" your Rmarkdown into presentation slides instead of a web-page, ready for a conference in little time. Rmarkdown is a natural evolution of vanilla markdown, backed and extended by the Rmrkdown ecosystem discussed shortly. Rmarkdown[1] isn't the only other flavour of markdown. There are other initiatives such as GitHub Flavored Markdown (GFM)[2] which mainly enhances content appearing on the github site, and CommonMark[3] which tries to unify all of the different flavours of markdown syntax, "but all will converge to pandoc tool." can this be expressed differently?

1.1 Rmarkdown core parts

Rmarkdown documents are broken into three main parts:

- YAML header
- The R chunks
- markdown - plain text

Those are different parts of the document that all work together to form or

[1] https://rmarkdown.rstudio.com/
[2] https://guides.github.com/features/mastering-markdown/
[3] http://commonmark.org/

"render" a final document. Each part can be customised with further options, which will be covered later in the book.

1.1.1 YAML header

The YAML header will always sits at the very top of your Rmarkdown document, and it starts and ends with triple dash symbols, `---`. Note that YAML is indentation and space sensitive, meaning you need to be rather strict about the amount of indentation you use and text strings will need to be quoted.

```
---
title: "Hello world"
author: "Kirill"
date: "17 June 2019"
output: html_document
---
```

1.1.2 The R chunks

These are special parts of the document that hold code that can be executed "inline" of the Rmarkdown document. R chunks are highly customisable via chunk options. We will spending a lot of time in the course working with code chunks and different options types.

````
```{r}
plot(mtcars)
```
````

1.1.3 Everything else is plain old markdown

```
 # Have I been Marked Down?
```

1.2 Challenge: Introduction 1

5 minutes

1. What file extention should we typically use for saving our **Rm**ark**d**own files? answer link[4]

2. What document types can be produced (compiled) from Rmarkdown?

[4] https://superuser.com/questions/249436/file-extension-for-markdown-files

3. Will I have to learn more "languages" to use Rmarkdown (discussion question)?

The short answer is no. Learning and writing Rmarkdown will take you a very long way.

The longer answer is yes. At some point in the future you might want to generate very sophisticated documents and for that you'll most certainly will need at least a tiny amount of html + css knowledge, and maybe some knowledge about LaTeX (I've yet to learn a single thing about LaTeX - so far so good :D).

check out this bit of Rmarkdown[5]

1.3 Rmarkdown ecosystem

Rmarkdown has a relatively complicated ecosystem. It includes several different R packages. Most of those packages "wrap" other existing tools, (tools written by different people), thereby providing an "easy" way to interface with the tools via R language. A large part of the ecosystem exists thanks to pandoc[6] tool.

1.3.1 Pandoc

Pandoc is a stand-alone tool (command line tool) that one can run in the terminal to convert markdown documents to other documents types including html, pdf and MS docs. Since vanilla markdown is pretty simple in what it can produce, pandoc added whole lot of "features", additional marking tags, that one can use to build more elaborate documents from plain text.

1.3.2 YAML

This is stand along language that is used in variety of places, with main advantage to it is that it can be easily ready by humans as well easily parsed by computer. A lot of the time YAML can be used as a configuration file. This is example how it is used with Rmarkdown. We will talk about YAML in more depth in a different section. In brief we will use YAML to set documents appearance and link additional files with the documents, such as bibliographies.

1.3.3 LaTeX

`¯\_()_/¯`

[5] link%20to%20github%20that%20the%20line%20of%20code%20above
[6] https://pandoc.org/

1.3.4 Knitr

As was mentioned before, we are using pandoc[7] to convert markdown to other document types. knitr[8] converts Rmarkdown files into vanilla markdown, which inturn can be converted by pandoc into an html document, for example. knitr[9] can execute R code and assemble the results into markdown.

1.3.5 Rmarkdown

An rmarkdown R package[10] will convert `.Rmd` files into other file format types. Under the hood it will use pandoc[11] to do so. The main function that we are concerned with is `rmarkdown::render()` which will call `knitr::knit()` when required.

1.3.6 Bookdown

A bookdown R package[12] enhances rmarkdown[13] by enabling multi-page documents e.g books and easy cross-referencing.

1.3.7 Others

These are more R packages that enable more things via Rmarkdown.

- xaringan[14]
- blogdown[15]
- thesisdown[16]

1.4 RStudio project

RStudio provides a nice option to create a new project. This in turn will create a new folder with a couple of special features, helping you to stay organised and "containerising" your project.

The following features apply to a project directory

[7] https://pandoc.org/
[8] https://yihui.name/knitr/
[9] https://yihui.name/knitr/
[10] https://github.com/rstudio/rmarkdown
[11] https://pandoc.org/
[12] https://github.com/rstudio/bookdown
[13] https://github.com/rstudio/rmarkdown
[14] https://github.com/yihui/xaringan
[15] https://github.com/rstudio/blogdown
[16] https://github.com/ismayc/thesisdown

- A new R session (process) is started
- The .Rprofile file in the project's main directory (if any) is sourced by R
- The .RData file in the project's main directory is loaded (if project options indicate that it should be loaded).
- The .Rhistory file in the project's main directory is loaded into the RStudio History pane (and used for Console Up/Down arrow command history).
- The current working directory is set to the project directory.
- Previously edited source documents are restored into editor tabs.
- Other RStudio settings (e.g. active tabs, splitter positions, etc.) are restored to where they were the last time the project was closed.

Taken from here[17]

1.5 Setup

We will need to install these packages. Let's install these packages

```
packages <- c("tidyverse",
              "rmarkdown",
              "knitr",
              "bookdown",
              "tinytex",
          "citr")

keep <- packages[!(packages %in% installed.packages()[,"Package"])]

if(length(keep)) {
  install.packages(keep)
}
```

1.6 Challenge: Introduction 2

2 minutes

1. Now is a good time to tweak your RStudio to your needs.

- change font size
- change themes and background color
- rearrange panels

2. Please turn off "Restore .RData into workspace at startup" in "Tools -> Global Options".

[17] https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects

1.7 Useful tips

- don't attempt to compile to `pdf_document` until absolutely necessary. `LaTeX` engine that is used by Rmarkdown to pdf conversion known to have issues with aligning figures and tables. This typically causes figures and tables overflow to next pages and general text misalignment. Get your content written first, intermediate compilation to `html_documents` are totally fine, before worrying about technical issues
- don't save data into `.RData`, this will make your work less reproducible.

Chapter 2

Vanilla Markdown

The original (vanilla) version of Markdown invented by John Gruber[1] defines a handful of tags, discussed next. Markdown is a relatively small and simple language for writing plain text documents that are easy-to-write and easy-to-read, but it is greatly enhanced and extended by pandoc tool.

2.1 Vanilla tags

Let's open our first Markdown file.

```
File
    New File
            R Markdown

title = "Learning Markdown"
author = "Me"
```

- select document type **HTML**
- to build (compile) the document by press `knitr` button or `ctrl+alt+k`
- save file by pressing `ctrl^S` and name the files `learning_markdown.Rmd`

Since we are using RStudio and R, it is inevitable that we will be using Rmarkdown flavour, but we can write vanilla markdown because under the hood Rmarkdown will always be converted to vanilla markdown.

From now on we are going to start using `knitr` to compile Rmarkdown into html. Remember from the Rmarkdown ecosystem[2] that `knitr` will convert Rmarkdown to markdown and `rmarkdown` R package will convert - render markdown

[1] https://en.wikipedia.org/wiki/Markdown

[2]

files into html. By pressing that blue button both things will happen automatically and we don't need to think about, but I wanted you to know that.

These are essentially all core (vanilla) markdown tags. Let's practice writing them.

```
# Header1
## Header2
### Header3

Paragraphs are separated
by a blank line.

Two spaces at the end of a line
produces a line break.

Text attributes _italic_,
**bold**, `monospace`.

Horizontal rule:

---

Bullet list:

  * apples
  * oranges
  * pears

Numbered list:

  1. wash
  2. rinse
  3. repeat

A [link](http://example.com).

![Image](Image_icon.png)

> Markdown uses email-style
> characters for blockquoting.
```

2.2 Practice vanilla markdown

Now, it's just a matter of learning some of the markdown syntax. Let's delete all current text from the opened document except the YAML header and type this new text: `Hello world, I'm learning R markdown !` and press the `knit HTML` button.

```
Hello world, I'm learning R markdown !
```

Not much happened. This is because we didn't mark our text in any way. You can put as much text as you want and it will appear as is, unless "specially" marked to look different.

Now add the `#` symbol at the start of the line and press the `knit HTML` button again. We'll be pressing this button a lot! For those who like keyboard short cuts use `ctrl+shift+k` instead.

```
# Hello world, I'm learning R markdown !
```

How about now? A single hash symbol made it a whole lot bigger didn't it? We've marked this whole line to be the header line.

Now make three new lines with the same text, but different numbers of `#` symbols, one, two and three respectively and keep pressing the `Knit HTML` button

```
# Hello world, I'm learning R markdown !
## Hello world, I'm learning R markdown !
### Hello world, I'm learning R markdown !
```

This is how you can specify different headers type using markdown.

Let's now practice making very short document in markdown with a main topic section and two subsections. We will add short sentences in each section. We will start with main section header and a quote. Let's type the following text and `knit` our document.

```
# Learning Markdown

> I'm still learning
```

Now let's add three bullet points summarising what we are going to learn next and then `knit` the document again.

```
# Learning Markdown

> I'm still learning

Here I'll be learning:

- markdown
- Rmarkdown
- git and github
```

Now let's add each one of those bullet items as a subsection to the main "Learning Markdown" section.We are going to use `##` to mark subsections and don't forget to `knit` again.

```
# Learning Markdown

> I'm still learning

Here I'll be learning:

- markdown
- Rmarkdown
- git and github

## Markdown

## Rmarkdown

## Git and GitHub
```

Now let's add a sentence to each section, briefly describing what they are.

```
# Learning Markdown

> I'm still learning

Here I'll be learning:

- markdown
- Rmarkdown
- git and github

## Markdown

Here I'll learng vanilla markdown
```

```
## Rmarkdown

Whereas here I'll be learning Rmarkdown

## Git and GitHub

And this section is scary
```

Let's add a emphasis to some of the words in our document. We are going to add *italic* emphasis to the word "vanilla" and we are going to add **bold** emphasis to the capital letter "R" in the word Rmarkdown. You'll need to `knit` your document still.

```
# Learning Markdown

> I'm still learning

Here I'll be learning:

- markdown
- Rmarkdown
- git and github

## Markdown

Here I'll learng _vanilla_ markdown

## Rmarkdown

Whereas here I'll be learning **R**markdown

## Git and GitHub

And this section is scary
```

2.3 Challenge: Markdown 1

5 minutes

1. How to mark text so that it appears underlined? answer link[3]

[3] https://softwareengineering.stackexchange.com/questions/207727/why-there-is-no-markdown-for-underline

2. Can markdown replace html[4] (discussion question)?

It has replaced html and latex in documentation and communication of results. My feeling is that the data science ecosystem heavily revolves around markdown. But html, pdf and latex, in this context, are simply communication and sharing mediums. On would not want to replace html + css for large website projects.

2.4 Cross-referencing

Let's learn how to add external and internal links to your document, remember the syntax for adding links is `[DESCRIPTION](link-address)`. The external link that we are going to add is going to be this https://rmarkdown.rstudio.com/. Each one of the bullet points above going to become a link to it section. The way you reference internal section is by starting your address with a `#` symbol then simply using all lower case letters for the section name and all spaces need to be converted to a dash symbol `-`. Let's add those things in and re-build our document.

```
# Learning Markdown

> I'm still learning

[External resource](https://rmarkdown.rstudio.com/)

Here I'll be learning:

- [markdown](#markdown)
- [Rmarkdown](#rmarkdown)
- [git and github](#git-and-github)

## Markdown

Here I'll learng _vanilla_ markdown

## Rmarkdown

Whereas here I'll be learning **R**markdown

## Git and GitHub

And this section is scary
```

A bonus exercise is to add logos to each sections. Search internet for:

[4] https://en.wikipedia.org/wiki/HTML

- Markdown logo, and add the image using `![](link_address)` syntax
- RMarkdown logo, and add the image using `![](link_address)` syntax
- Git logo, and add the image using `![](link_address)` syntax
- GitHub logo, and add the image using `![](link_address)` syntax

Note for the external resource that is on internet the address must start with `www` or `https` otherwise address will be interpreted as file path.

Chapter 3

Rmarkdown

The reason that we are learning Rmarkdown[1] is because it gives us a very straightforward way of writing plain text documents with inline R code that will become a very sophisticated document types. The bonus points also come from the fact that Rmarkdown files are easy to version control (git) and see the difference between versions. This approach of interleaving analysis code, commentary and description is very explicit, which has direct implication in reproducibility, shareability and collaboration.

In the Markdown section I've showed you how to start new Rmarkdown document in RStudio, but lets briefly recap how we do that again.

```
File
    New File
           R Markdown

title = "Learning Rmarkdown"
author = "Me"
```

- select document type **HTML**
- to build (compile) the document by press `knitr` button or `ctrl+alt+k`
- save file by pressing `ctrl^S` and name the files `learning_rmarkdown.Rmd`, not an additional **r** there.

There is a hint in the name - **R** as to what programming language will be using for the rest of our course. Hopefully you have some familiarity with an R and it's syntax or you have had some other exposure to other programming languages. I wont be asking you to write your own R code, most of it will be copy and paste from this book. However certain terminology will be assumed going forward.

[1] https://rmarkdown.rstudio.com/

This is mainly things like "variable", which in use in programming languages to store a value, can a string e.g "ten" or a number e.g 10. Typically to get the value out of the variable you'll need to "print" it out. These are bear minimum understanding that you should have, and it'll be enough to get you through this workshop. Let's start by writing some R and embedding it into Rmarkdown documents.

3.1 Embedding R code

RStudio templates `.Rmd` file for us. However lets delete all the text after the yaml header. The following text should remain in the file.

```
---
title: 'Learning Rmarkdown'
author: 'Kirill'
date: '25/07/2019'
output: html_document
---
```

I'm going to explain `knitr::opts_chunk` in later section. An R chunk is a "special" block within the document that will be read and evaluated by `knitr`, ultimately converting everything into plain markdown. But for us it means that we can focus on our analysis and embed R code without having to worry about it. Additionally there are large number of chunk options that helps with different aspects of the document including code decoration and evaluation, results and plots rendering and display.

This is how an R code chunk looks like. If you want to include code into your documents it has to be via R chunks. You can further customise the appearance of your code in the final document with chunk options.

````
```{r}

```
````

The little `r` there specifies the "engine", basically telling Rmarkdown how to evaluate the code inside the chunk. Here we are saying use `R` engine (language) to evaluate the code. The list of languages[2] is rather long, hence why earlier comments about Rmarkdown spanning much greater area then one might think. In this workshop we are only going to focus on R language.

Let's write our first bit of R code inside the Rmarkdown document. First we need to start a new R chunk, which we can be done in these ways:

[2] https://bookdown.org/yihui/rmarkdown/language-engines.html

- simply type it out
- press insert button at the top of the window
- `ctrl+alt+i`

Let's start our new document with the main header section and type the following `# Learning Rmarkdown`. Now let's add simple R code to our chunk, type the following code `a <- "Hello world, I'm learning Rmarkdown !"` and press `knitr` button to build html document. Note that as mentioned above we need to use `print()` statement to get the content of the variable to the scree/final document.

````
```{r}
a <- 'Hello world, I'm learning Rmarkdown !'
print(a)
```
````

Tip: each chunk can be run independently in the console by pressing `ctrl^enter` or little green arrow.

3.2 Chunk options

Each code chunk is highly customisable via chunk options[3]. We are going to learn a few today, but we won't be able to cover all of them, but here is definitive guide from the author of Rmarkdown[4]. However you probably never going to use some of them, but as long as you know what to look for you'll be able to search for then. Note that all chunk options have a default value. Not specifying an options means you are using the default value. These are chunk options that we are going to cover today.

name	value	type	description
child	NULL	code_evaluation	A character vector of filenames. Knitr will knit the files and place them
engine	'R'	code_evaluation	Knitr will evaluate the chunk in the named language, e.g. engine = 'py
eval	TRUE	code_evaluation	If FALSE, knitr will not run the code in the code chunk.
include	TRUE	code_evaluation	If FALSE, knitr will run the chunk but not include the chunk in the fin
fig.align	'default'	plots	How to align graphics in the final document. One of 'left', 'right', or 'ce
fig.cap	NULL	plots	A character string to be used as a figure caption in LaTex.
fig.height	7	plots	The height to use in R for plots created by the chunk (in inches).
fig.width	7	plots	The width to use in R for plots created by the chunk (in inches).
echo	TRUE	results	If FALSE, knitr will not display the code in the code chunk above it's r
results	'markup'	results	If 'hide', knitr will not display the code's results in the final document.
message	TRUE	results	If FALSE, knitr will not display any messages generated by the code.
warning	TRUE	results	If FALSE, knitr will not display any warning messages generated by the

[3] https://bookdown.org/yihui/rmarkdown/r-code.html
[4] https://yihui.name/knitr/options/

General layout of any chunk is

````
```{r chunk_name, options}

```
````

Note a couple of things, there isn't a comma between `r` and `chunk_name`. Not sure why this is. Also note that `chunk_name` is optional, you can skip it, as we have in earlier examples. Naming chunks is good idea to conceptually label the chunk as to what it does, but also we you are going to build more sophisticated documents you'll be able to selectively include chunks by refer to them by the chunk name.

Lets start off with these four chunk options:

- `echo` add the content of the chunk into the document i.e the code itself
- `eval` add results of the evaluated code to the document
- `include` add code, results and figures to the document. If `include = FALSE` nothing related to that chunk will be added to the document.
- `results` add results to the document. If `include = TRUE` and `results = "hide"` no results will be added to the document, except for figures.

These allow us fine level control over the final document. Think about who are generating the document for and what type of information you need to share. Sometimes we might want to show the code, but not execute it and other times we might just want to execute it and share the results, e.g plot, without actually showing the code.

Let's add a subsection to our document, type the following `## chunk options intro`. In that section we are going to add a bit more code. We are going to split our `a` variable into three variables. We won't discuss why you would want to do that in your programming practice, in fact this could simply come down to your personal preferences. I'm doing it here mainly to have more code in the code chunk to illustrate a couple of points about chunk options. Add this code to your chunk and press `knitr`

````
```{r}
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```
````

3.2.1 echo and eval

Let's now explore these to chunk options, `eval` and `echo`. Note that the default value for them is `echo = TRUE` and `eval = TRUE`. We should not see any changes to our document.

````
```{r echo = T, eval = T}
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```
````

But if now switch `echo` off by changing `TRUE` to `FLASE`, we should see a change in our final document. Don't forget that you need to re-build your document.

````
```{r echo = F}
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```
````

Okay, we don't see any of our original code. This is the result that we were aiming for. Let's now turn `echo` back on, by adding `echo = T` and turning `eval` off by adding `eval=FALSE`. Once again always re-build your document with `knitr`.

````
```{r eval = F}
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```
````

The effect that we are seeing now is the opposite, i.e we see the results but don't see the code, which is once again what we have anticipated.

3.2.2 include

We going to create another subsection in our document, let's type `## chunk options more`. In this section we are going to learn more chunk option to helps us manipulate our final document look. This option dictates whether the output

of the executed code will be included into the final document. Sometimes you can simply trigger the `eval` flag to achieve a similar result of code not being included, but other times you might want the code to actually be executed but not included. For example when future R chunk relies on the output of this intermediate chunk, but there is no need to include that into the document.

`include` and `results` are best illustrated with a plot example, so we are going to use `mtcars` data set that is already available in RStudio, so you don't have to do anything to get it. You can learn more about the data set by running the following in your R console `?mtcars`, but we are going to skip ahead, since knowing the data in this particular case isn't important. Some of the syntax below can be new and unusual to you. The point of this exercise is to understand code chunk options and not to learn R code. It is totally fine for this example to copy and paste the code. Brief summary about the code `kable` function help with table printing in your final document and `qplot` helps us with plotting the data.

Let's type the following and then build our document.

````
```{r include = T}
knitr::kable(mtcars)
p1 <- ggplot2::qplot(mtcars$mpg, mtcars$drat)
p1
```
````

We expect to see table with data and a plot.

Let's now turn `include` off, by switching `TRUE` to `FLASE` and re-build our document.

````
```{r include = F}
knitr::kable(mtcars)
p1 <- ggplot2::qplot(mtcars$mpg, mtcars$drat)
p1
```
````

We don't expect to see anything at all in our final document, remember that `include` controls code, results and figures output in our final document.

3.2.3 results

I'm now going to introduce `results` options. It is similar to an `echo` option, and some things can be achieved with `echo` alone. In fact it'll take some trial and error before fully appreciating differences between these options, `echo`, `eval`, `include` and `results`. `results` has four options, definitions taken from here[5]:

[5] https://yihui.name/knitr/options/

- `asis`: output as-is, i.e., write raw results from R into the output document
- `hide`: hide results; this option only applies to normal R output (not warnings, messages or errors)
- `markup`: mark up the results using the output hook, e.g. put results in a special LaTeX environment
- `hold`: hold all the output pieces and push them to the end of a chunk

We are only going to look at `asis` (default) and `hide` options mainly due to time constrains, but also because the use case for those options is for more advance/edge use cases.

Let's keep `include = TRUE` by now also add `results = 'hide'`

````
```{r include = T, results='hide'}
knitr::kable(mtcars)
p1 <- ggplot2::qplot(mtcars$mpg, mtcars$drat)
p1
```
````

We some things but not others. We don't see our table but we can see the plot in our final document.

This is going to be rather complicated example. The purpose of it is to illustrate the fine turning that you might want to do in your report. We are going to have three chunk options `include = T`, `results = 'asis'` (default) and `eval = F`. We are also going to add a second code chunk to our document, without any options and we simply going to type `p1` variable in that chunk.

````
```{r include = T, results='asis', eval = F}
knitr::kable(mtcars)
p1 <- ggplot2::qplot(mtcars$mpg, mtcars$drat)
```

```{r}
p1
```
````

In this case we getting our table with data, and two plots, one after another. Typically you wouldn't have those two R chunks next each other, they'll most likely be split apart by some text explaining the data flow.

Remember that `eval` control execution of the code. And let's assume that I don't want to include any of the code in the first chunk in my document. I only want to include final figure i.e `p1` plot. Let's try turning `eval` off, by switching it to `FALSE` and re-build our document

````
```{r include = T, results='asis', eval = F}
knitr::kable(mtcars)
p1 <- ggplot2::qplot(mtcars$mpg, mtcars$drat)
```

```{r}
p1
```
````

We get this error, because that R code hasn't been evaluated and therefore `p1` hasn't been formed.

```
Error in eval(expr, envir, enclos) : object 'p1' not found
Calls: <Anonymous> ... handle -> withCallingHandlers -> withVisible -> eval -> eval
Execution halted
```

If you look at the code closely you will see that it tells you just that **object 'p1' not found** (with additional craft around it)

The way to achieved this is to set `include = F` and `eval = T`. The role of `results = 'hide'` is subtle here but it is a fine level control of showing the code and not showing the code. Later on I'll also show you a way to fold your code in the final document type, using yaml header `code_folding: hide` option. This will hide all of your code in the final document, but will add a toggle to each code chunk to allow user to see the code if interested. The difference between `results` and `echo` is that former will completely remove results from the document, whereas the latter can have additional turning. Once again the reasons to each setting are use by use cases. Are you sharing the code and results or you just sharing results?

3.3 Challenge: Rmarkdown 1

3 minutes

1. Go through all of your code so far and give each chunk a name

   ````
   ```
   {r chunk_name, options}
   ```
   ````

Part II

Part

Chapter 4

Git and GitHub introduction

> When you are rock climbing you want to set your anchors often How often will depend on your experience and desire not to fall Git commit like you are vertically hanging off 70 feet rock

I am going to break to you right at the start that (unfortunately) doing git and GitHub is like rock climbing, but nonetheless it has great benefits for your research and analysis including making it more visible, reproducible and potentially very collaborative.

Git[1] is one of many tools, but it is very popular. Git was designed for **tracking versions** of software development - a.k.a version control tool. While it hasn't been strictly design with scientific research projects in mind we will happily re-purpose git to help us stay on top of our research projects. In the git world everything rotates around a git repository, which is a "special" folder. Inside that folder every file and folder is "tracked" for changes. Git repositories often are synonymous with project folder. In our case the RStudio project folder will be the same as the git folder. In other words we are going to tack all changes in our project. Note that the two, git and RStudio project, are independent of each other and you can use one or another or both.

Below I am trying to illustrate of the differences between do-it-yourself (DIY) version control system, which may be great, and git version control. DIY version control systems are great with two caveats:

- no one else will understand it
- the future you will forget the awesome schema that you have invented

[1] https://git-scm.com/doc

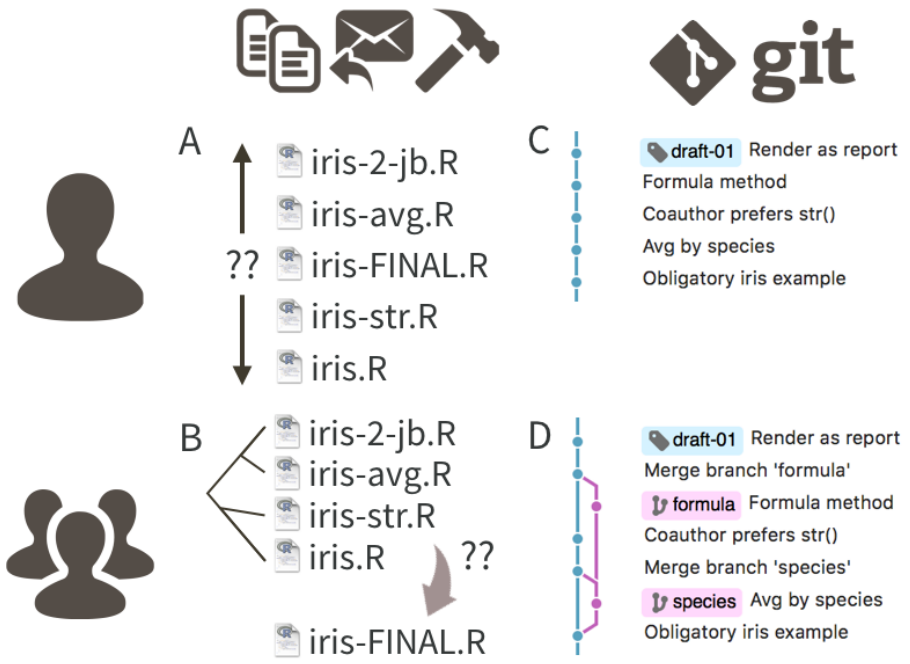


Figure 4.1: This is an example of git version control vs DIY versioning via filesystem

Figure 4.2: https://www.geekboots.com/story/what-is-the-difference-between-bitbucket-github-and-gitlab

From now onwards we are going to use the `git` version control tool. We are also going to use GitHub for storing our files remotely. GitHub isn't the only place that people can use with git for file storage and sharing. Below an illustration of some other common place one can choose to store they git repositories a.k.a projects.

In theory one can also store `git` repositories in google drive or dropbox or other similar places, however neither of those places have been optimised for `git` version control repositories. We will talk shortly about advantages that GitHub brings to the project.

For this workshop we are going to use GitHub, mainly because is very popular and it has a lot of useful features.

4.1 Quick summary

Git and GitHub will help you:

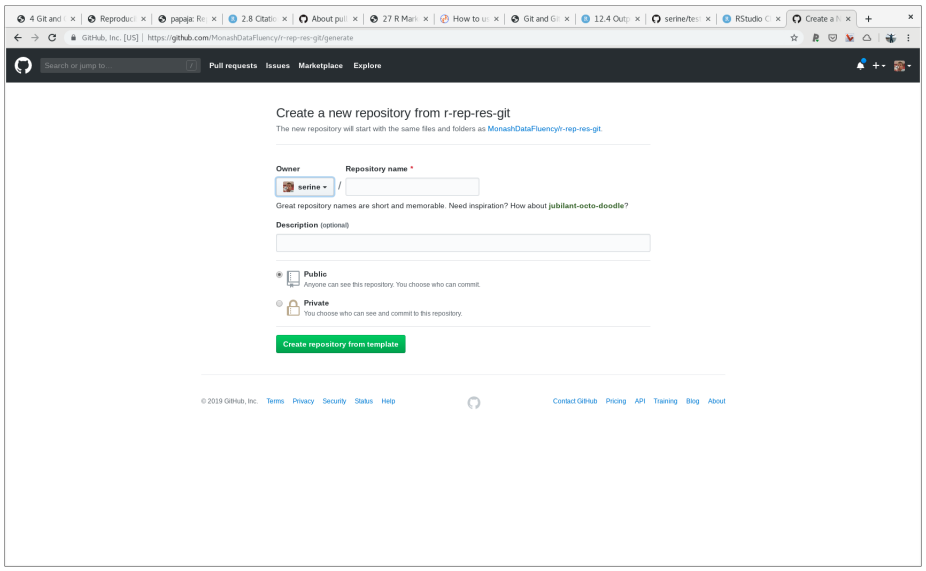
- organise our directory structure
- create "milestones" a.k.a `git commits`
- make apparent which parts of the projects (files) are important
- share your work (e.g GitHub)
- collaborate at the global scale

4.2 Github setup

There are a couple of different ways you can start a project and initiate git repository - git tracking. We are going to start with GitHub first approach. An alternative approach discussed in appendix section D. I hope that everyone had already created GitHub[2] account. We are going to make new github repository from this template[3].

- sign in to github github.com[4]
- head over to https://github.com/MonashDataFluency/r-rep-res-git
- click "Use this template"

You should see a screen like this



- Repository name "learning_rmarkdown"
- Description "I'm learning Rmarkdown, yay!"
- click "Create repository from template"

Note that a description of the repository is optional, but it is a good idea to write a brief sentence there to message yourself and the public what this project is about.

[2] https://github.com/
[3] https://github.com/MonashDataFluency/r-rep-res-git
[4] https://github.com

Once we have our GitHub repository we need to find a link (URL) and copy a.k.a **clone** (git clone) our repository to our working computer, in our case rstudio.cloud[5]. We want to establish connection between rstudio.cloud[6] and GitHub such that we can with little effort we can copy our work, that is file from rstudio.cloud[7] to GitHub.

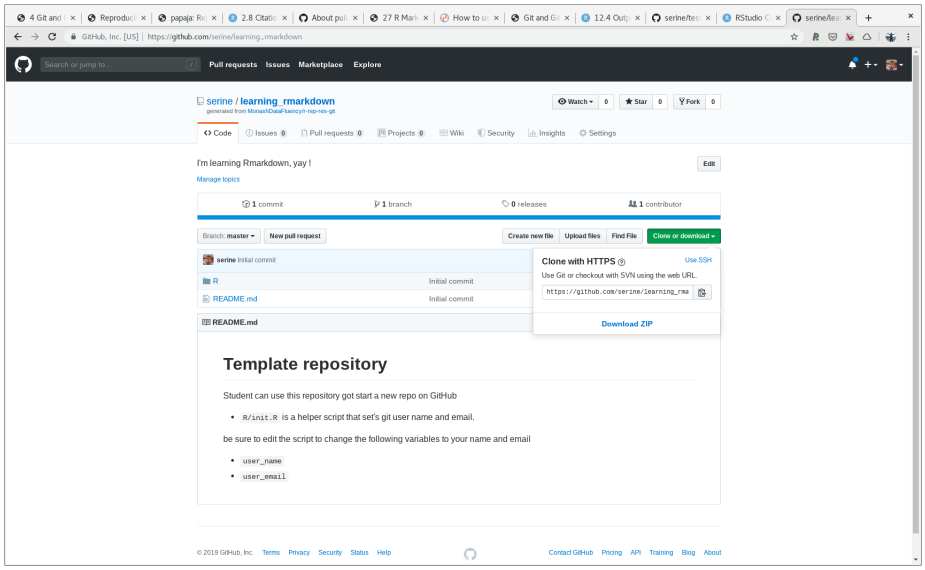
This is how a typical URL looks

```
https://github.com/kirill/learing_rmarkdown.git
```

There are at least three components in that url

- `https://github.com/` the place i.e the name of the website
- `kirill/` username
- `learing_rmarkdown.git` repository name (project name)

You can get this url, but either looking at the address bar of your browser or there is a little drop down menu on the right hand site.



Note that by creating a new github repository we are automatically initialising git for version tracking. Remember that GitHub is just a place where we are storing our git repository.

[5]https://rstudio.cloud
[6]https://rstudio.cloud
[7]https://rstudio.cloud

4.3 RStudio setup

Add description about starting a project from a github at RStudio.

4.4 GitHub things

- PR (pull request)
- gitissues place to talk about issues related to a project
- stars acknowledgement
- watch interested in updates on a projects

collaborators and update dates/commits as a proxy of how active the project is. also do check which files typically being changed. Also mention the fact that it is very explicit when the project was started (initiated) how much work has gone into it (commits history) and roughly time frame and intervals of work

in simple workflow and collaborations git merge will work just fine. git will happily merge two different branches i.e all files in one location with all files in the other location if no two file conflict

4.5 More git

Git is a command line tool however you don't have to learn command line just yet. There are a few git clients available[8] - graphical user interface (GUI) tool / applications that we can use instead of learning command line. We are going to use RStudio which has good git support and therefore Rstudio will be our git client. One rather important note about git clients, most (all) clients will "simply" form a git commands as you would type it out and execute on command line. This means a couple of things:

1. one can use mixture of clients and command line without any issues. For example if one needs more complicated git command one could run it on the command-line.
2. if you need to do a more complicated git kung fu you might only find solution for command line and then it'll be up to you to figure out how to work it into your client

An interesting note about command line git usage noted in Happy Git with R book[9]; One might think that git via cli is "better", however it is more important to get the work done and have it version controlled rather then fight with the

[8] https://happygitwithr.com/git-client.html
[9] https://happygitwithr.com/git-client.html

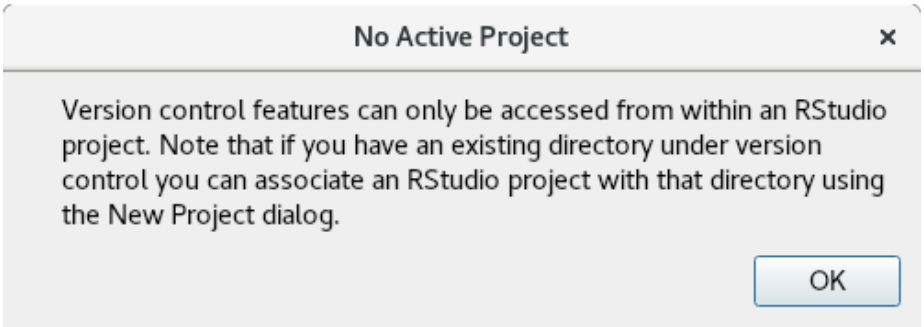
cli. Do take the simplest and quickest path to get your work version controlled. No one will care which client you are using in the end.

4.6 git no nos

- no spaces in file names (this goes beyond git)
- no git repositories inside an existing git repository

4.7 Starting with git

In Rstudio you can only start working with git when you have an existing Rproject directory



4.7.1 Configuring git

You will most certainly forget this step, because you only need to do it once per computer (or new installation of git). Git will remind if you haven't done these steps. These are our very first step in being organise and ready for future collaboration. We need to let git know our name and email address, which will get stored in configuration file.

Unfortunately RStudio doesn't have support for setting up config. It was probably not worth implementing given that you only really do it once. We will have to use terminal (command line) just this once.

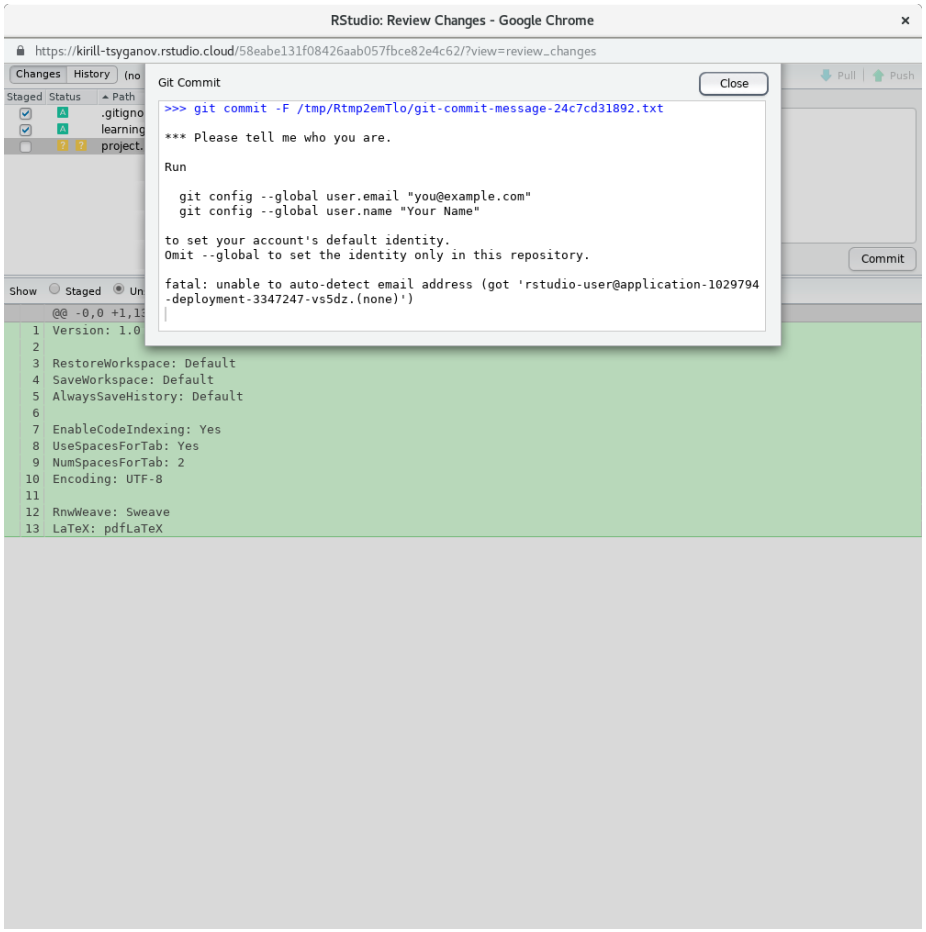
Let's open up a terminal and run a couple of `git` commands

Tools -> Terminal -> New Terminal

```
git config --global user.name kirill
git config --global user.name "kirill.tsyganov@monash.edu"
```

One can then double check that all was set correctly bu running this command.

```
git config --global --list
```



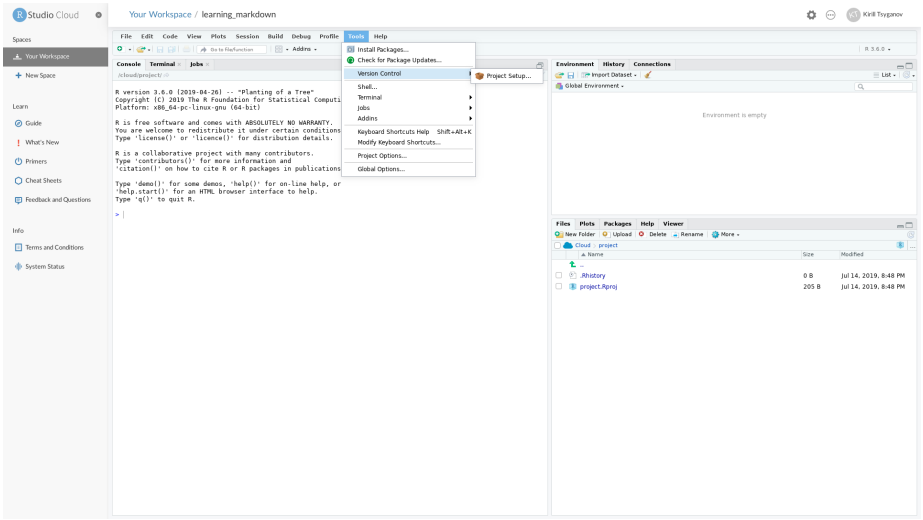
4.7.2 Intiating git repository

In git jargon repository is simply your working folder (folders sometimes also called directories). In our case

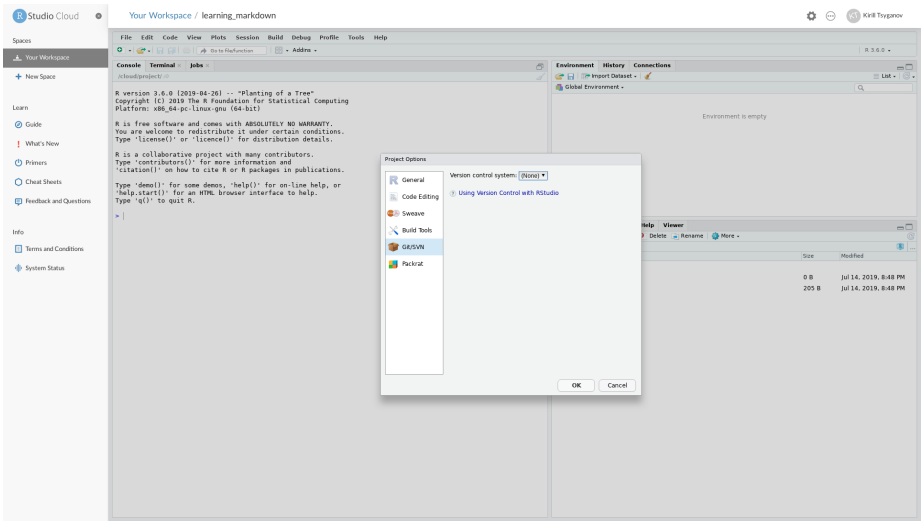
```
Rproject folder == Rproject directory == git repository == git repo
```

Let's initate git repository

Tools -> Version Control -> Project Setup -> Git/SVN



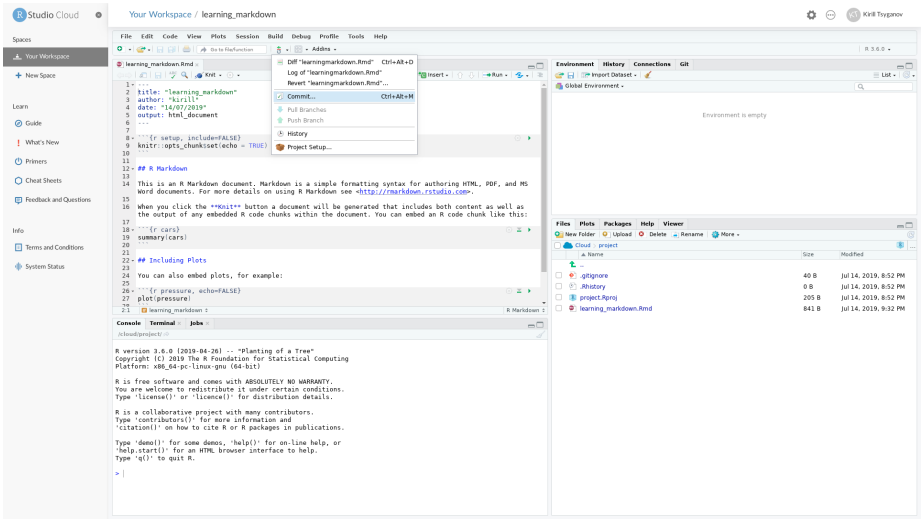
And select from the drop down options "Version control system" Git



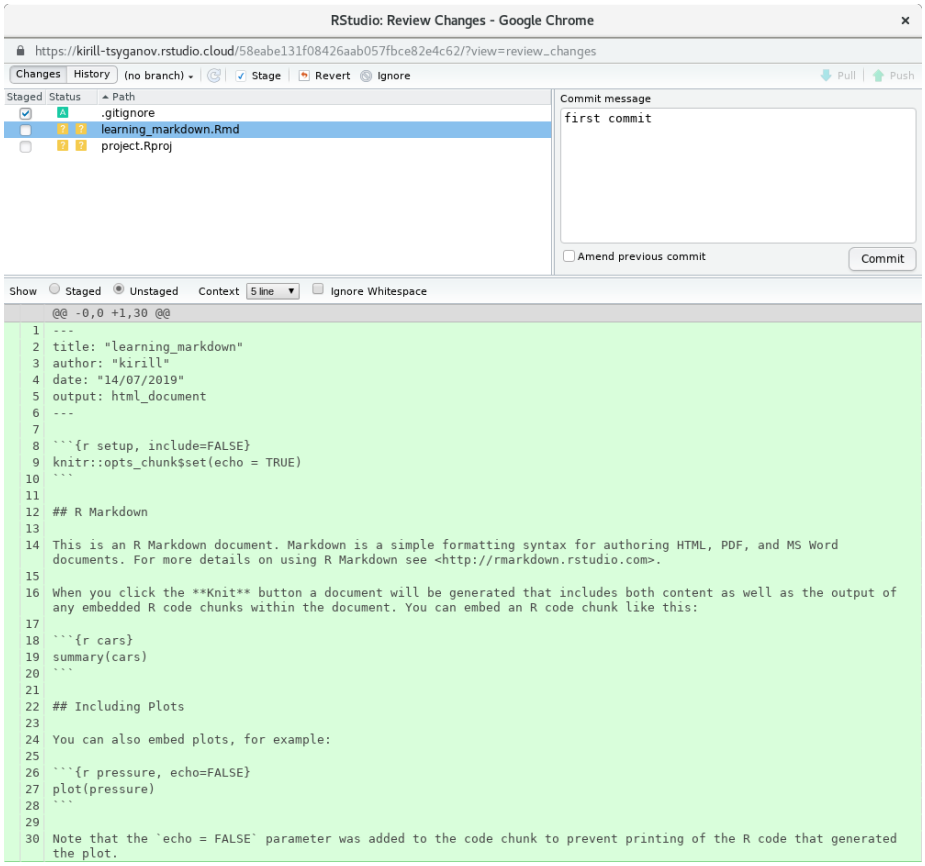
The command line equivalent is navigating to your project directory and running `git init`

4.7.3 First commit

Let's make our first commit, use drop down menu as indicated on the image below to select `commit` option



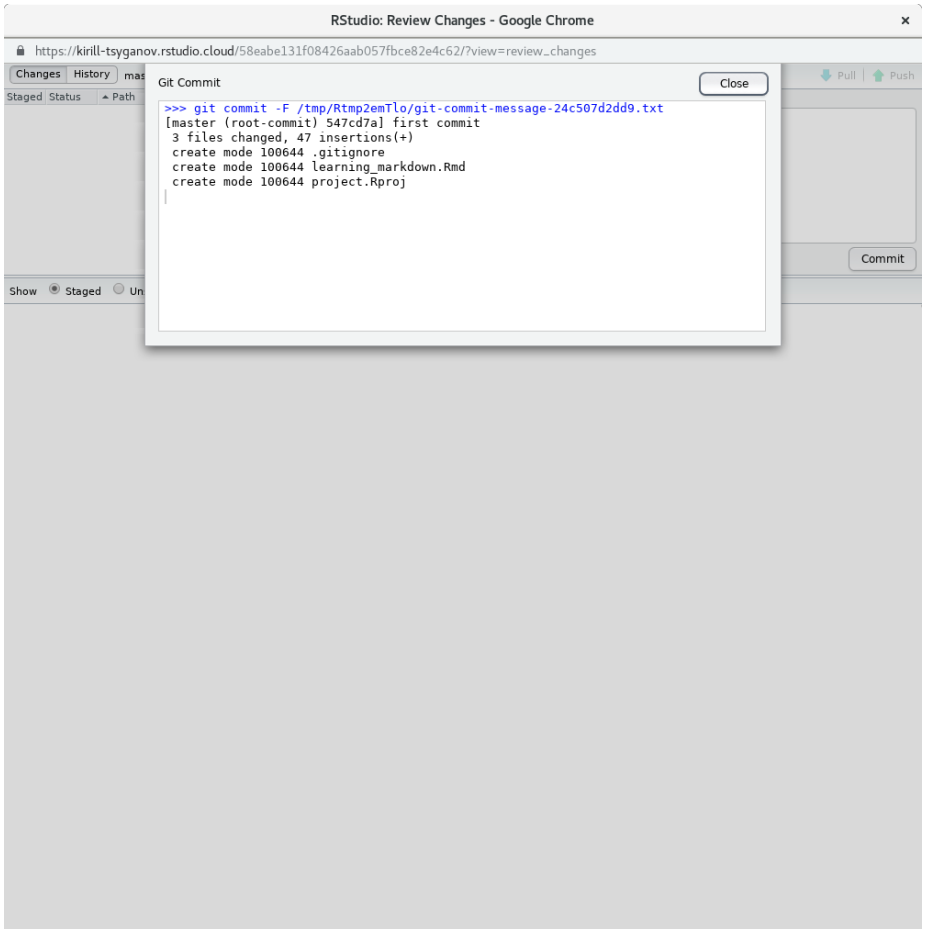
You should see a new window poped up



Then we are going to add three files

- `.gitignore`
- `project.Rproj`
- `learning_markdown.Rmd`

Write a commit message and press commit. And this is how happy git commit looks like



The commit message is rather important. Remember that commit message is:

- a message to a future you
- a message to your supervisor
- a message to all other external people

Those commit messages are means of communications e.g

- "fixed figure 1 legend"
- "added new paragraph to chapter 1"
- "I bloody hate this project delete everything, starting from scratch"

Good thing is, as long as you "tracking" your deletes you can always go back to them and check what you have deleted and revert some of those changes back when needed. However in this workshop we won't be covering much of that.

Also note that commit message don't have to long, and can be as short as one work - "update2", but at the same time well written commit message will help you and other.

http://r-pkgs.had.co.nz/git.html#commit-best-practices

4.8 Which files to commit?

This section will be extended in the future release, but I highly recommend reading this article, specifically section 10: Which files to commit from here[10]

[10] https://peerj.com/preprints/3159/

Chapter 5

More Rmarkdown

We are going to inrease the difficulty a little bit and we are going to start working towards our final documnet[1].

Typically you will have some data set that you are trying to analyse and later present. There are likely to be some other prior steps before you get your tabular data. Those prior steps should also be documented. In this workshop we are going to start with a tabular data set straight away. We are going use Domestic Airlines - On Time Performance[2] data set from data.gov.au[3]

5.1 Setup

First thing first is we need to download data. `read_csv()` function from `readr`[4] package can "read" directly from url, but we are going to "cache" a file first and then we are going to reference our local copy. This will shorten our final html building time. We should also check licence on the data set, especially if you are going to publish some of your analysis. This data is *Creative Commons Attribution 3.0 Australia* licence, there is no problem in downloading and using the data.

Let's open new Rmarkdown file and delete everything from it except the yaml header.

[1] examples/single_page_report.html
[2] https://data.gov.au/data/dataset/domestic-airline-on-time-performance
[3] https://data.gov.au
[4] https://readr.tidyverse.org/

5.1.1 Setting global chunk options

As you have learned already you can manipulate each R chunk with options, but you can also set global settings for each chunk. Let's set `echo = TRUE` and `message = FALSE` globally. This means every R chunk will be echoed i.e shown in the final document and no messages will appear anywhere in the document.

```
options(encoding="utf-8")

knitr::opts_chunk$set(echo = TRUE,
                      message = FALSE)
```

5.1.2 Loading libraries

We are going to do our analysis with the help of `tidyverse`[5] library, let's load it in.

```
library(tidyverse)
```

5.1.3 Downloading the data

We are doing conditional download, so that we don't need re-download every time we build a document.

```
fn_data <- "domestic_airline_performance.csv"
fn_notes <- "domestic_airline_performance_notes.txt"

if(!file.exists(fn_data)) {
  url_data <- "https://data.gov.au/data/dataset/29128ebd-dbaa-4ff5-8b86-d9f30de56452/re
  url_notes <- "https://data.gov.au/data/dataset/29128ebd-dbaa-4ff5-8b86-d9f30de56452/r

  download.file(url_data, fn_data)
  download.file(url_notes, fn_notes)
}

df <- read_csv(fn_data, quote = "")
df
```

```
## # A tibble: 80,083 x 14
##    Route Departing_Port Arriving_Port Airline Month Sectors_Schedul~
##    <chr> <chr>          <chr>         <chr>   <dbl>            <dbl>
```

[5] https://www.tidyverse.org/

```
##  1 Adel~ Adelaide       Brisbane      All Ai~ 37987              155
##  2 Adel~ Adelaide       Canberra      All Ai~ 37987               75
##  3 Adel~ Adelaide       Gold Coast    All Ai~ 37987               40
##  4 Adel~ Adelaide       Melbourne     All Ai~ 37987              550
##  5 Adel~ Adelaide       Perth         All Ai~ 37987              191
##  6 Adel~ Adelaide       Sydney        All Ai~ 37987              486
##  7 Albu~ Albury         Sydney        All Ai~ 37987              168
##  8 Alic~ Alice Springs  Sydney        All Ai~ 37987               63
##  9 All ~ All Ports      All Ports     All Ai~ 37987            31913
## 10 Bris~ Brisbane       Adelaide      All Ai~ 37987              155
## # ... with 80,073 more rows, and 8 more variables: Sectors_Flown <dbl>,
## #   Cancellations <dbl>, Departures_On_Time <dbl>, Arrivals_On_Time <dbl>,
## #   Departures_Delayed <dbl>, Arrivals_Delayed <dbl>, Year <dbl>,
## #   Month_Num <dbl>
```

5.2 Exploring the data

Let's explore our data set. It is always helpful to get more information about the data set, good to start with getting dimensions of the `data.frame`, i.e number of rows and columns. Let's include the following bit of come into our Rmarkdown document. Note that instead of executing and showing the code we are stroing results in a variable `d`, and for that we are using `echo = FALSE` chunk option.

````
```{r echo = F}
d <- df %>% dim
```
````

Later in the text we will be able to access variable as you would in R

```
`r d[1]`
```

Lets add the following sentence to our Rmarkdown document and then `knit` to see the results.

```
total number of observation `r d[1]` and total number of variables `r d[2]`
```

Now we are going to find out names of all airlines presented in the data set and number of observation for each airline. In this case we are not interested in showing actual code, so we are going to hide it with `echo = FALSE` once again.

````
```{r echo = F}
df %>%
```
````

```
  group_by(Airline) %>%
  summarise(n = n()) %>%
  arrange(-n)
```

I hope you have noticed "All Airlines" name in the `Airline` column. I am not fairly with such airlines. This is likely some summary field that we don't know about. In the real life you should consult people who has generated the data set, but in our case we are simply going to filter those observations out. Once again we are hiding the results from the final document, by setting `eval = TRUE` to make sure that the code runs since we will need `df2` later in the document.

```
```{r echo = F, eval = T}
df2 <- df %>% filter(Airline != 'All Airlines')
```
```

5.3 Including external files

- `child` chunk option allows us to include external files into our document

You have probably noticed that we have downloaded two files, the data set and the notes about the dat set. I didn't want to spend time going over those notes, but let's include them into our Rmarkdown document for future references.

Let's add the following text to out document

```
### Additional information about the data set

```{r child = 'domestic_airline_performance_notes.txt'}

```
***
```

5.4 Challenge: More Rmarkdown 1

5 minutes

1. Can you summarise routes in similar way as we did with airlines? use `group_by(Route)`

2. Can spot an odd route in you summary? If you can filter is our from `df2`. `filter(Route != "All Ports-All Ports")`

5.5 Visualising the data

Our document looks pretty good so far, let's add some visualisation to our document, with that will be covering these chunk options the following chunk options, all to do with figures manipulation.

- `fig.align` - left, right, center or default (left)
- `fig.height` - height specified in inches
- `fig.width` - width specified in inches
- `fig.cap` - string of text in quotes

Firstly lets make sure we have our data properly filtered. We are going to filter out `All Ports-All ports` routes, since this is similar to `All airlines` field and likely to be some summary field that we are not interested in. We also going to only look at two airlines, Jetstar and Qantas, due to time constrains. Here we don't need to set any particular chunk options because I think it would be informative to show our filtering and we actually need that code to be run.

````
```{r}
df2 <- df %>% filter(Airline == 'Jetstar' | Airline == 'Qantas')
                     Route != 'All Ports-All Ports')
```
````

Here we are summarising our data so that we have an idea of how many times a particular location had be used by our airlines per year. Let's include that bit code into our Rmarkdown document and once again let's hide our code from the document by setting `echo = FALSE`

````
```{r echo = FALSE}
df2 %>%
  group_by(Airline, Year, Departing_Port) %>%
  summarise(n = n()) %>%
  ungroup %>%
    ggplot(aes(Departing_Port, n, color = Airline)) +
    geom_boxplot() +
    theme(axis.text.x=element_text(angle=45, hjust=1))
```
````

Let's experiment with setting different dimensions to our figure. Lets set both height and width to 4. Remember that units for height and width are in inches.

````
```{r echo = FALSE, fig.height = 4, fig.width = 4}
df2 %>%
````
````

```
  group_by(Airline, Year, Departing_Port) %>%
  summarise(n = n()) %>%
  ungroup %>%
    ggplot(aes(Departing_Port, n, color = Airline)) +
    geom_boxplot() +
    theme(axis.text.x=element_text(angle=45, hjust=1))
```

Now let's try to align our figure to the center by setting `fig.align = 'center'` and re-build our document.

Let's experiment with setting wight to 15 and height to 9. At some point though we are going to start hitting the physical limit of the html page.

5.6 Challenge: More Rmarkdown 2

5 minutes

1. Can you align figure to the right?? `fig.align = 'right'`

2. Align figure to the center and add figure legend using `fit.cap` chunk option `fig.align = 'center', fig.cap = 'Figure 1: blah'`

3. Can you add some emphasis to figure legend, e.g make important parts bold or underlined? Remember that figure legend is just a string of text and any text can be marked

5.7 More plots

Let's attempt to see what is the distribution of cancellation in any given year

Note the `warning` message that comes up in the text, let's assume we understand it and let's just turn it off by setting `warning = F`

```
```{r, fig.width = 14, fig.height = 9}
df2 %>%
  filter(Airline == 'Jetstar' | Airline == 'Qantas') %>%
  select(Airline, Departing_Port, Cancellations, Year) %>%
    ggplot(aes(Departing_Port, Cancellations, color = factor(Year))) +
      geom_boxplot() +
      facet_wrap(~Airline) +
      theme(axis.text.x=element_text(angle= 45, hjust=1))
```
```

5.8 More chunk options

I'm going to share a couple of more code chunks, these are mainly cosmetic, some of you may never use them.

- `prompt=FALSE` i.e mimic *consosle*
- `comment=` remove hash symbol at the front of the output

For this example I'm going to use simple `for` loop. We are going to use this variable `sentence <- c("Let", "the", "computer", "do", "the", "work")`

````
```{r}
sentence <- c('Let', 'the', 'computer', 'do', 'the', 'work')

for(word in sentence){
  print(word)
}
```
````

```
sentence <- c("Let", "the", "computer", "do", "the", "work")

for(word in sentence){
  print(word)
}
```

```
## [1] "Let"
## [1] "the"
## [1] "computer"
## [1] "do"
## [1] "the"
## [1] "work"
```

Let's add `prompt=TRUE`

```
> sentence <- c("Let", "the", "computer", "do", "the", "work")
>
> for(word in sentence){
+   print(word)
+ }
```

```
## [1] "Let"
## [1] "the"
## [1] "computer"
## [1] "do"
## [1] "the"
## [1] "work"
```

Chapter 6

YAML introduction

As it was mentioned earlier in the book yaml is a stand along language, often it is used as a configuration file, which is true in the case of Rmarkdown. We are going to use it in conjunction with Rmarkdown documents by embeding yaml blob into Rmarkdown files, just like we did with R chunks. YAML blob, or in the context of Rmarkdown YAML header, must be at the very top of your Rmarkdown file. The yaml header enable high degree of customisation for our final documents. We can also use yaml header to "attach" or reference other external files such as bibliographies and styling in the form of css[1].

For the purpose of configuring Rmarkdown documents you need to know three variable types:

- `scalar` stand along value e.g 3 or "car"
- `list` a collection of items e.g ["learing", "to", "use" "yaml"]
- `map` a different collection type that can hold simple types. e.g rmd_files: ["00-index.Rmd", "01-introduction.Rmd"]

6.1 YAML header

YAML is relativelly similar to another file format JSON[2], if you are familiar with one you should have little problem wrapping your head around the other. Fundamentally both file types trying to provide structured relationship between items via key = value pairing. Keys sometimes interchangeably used with tags. Remember that yaml language, unlike json, is very sensetive to spaces and indentations. Below are some examples of valid and invalid yaml syntax.

[1] https://en.wikipedia.org/wiki/Cascading_Style_Sheets
[2] https://en.wikipedia.org/wiki/JSON

6.1.1 Example 1

This is a valid yaml header. In this exapmle we have a key `title` with scalar value `Learning Rmarkdown`

```
---
title: "Learning Rmarkdown"
---
```

6.1.2 Example 2

This is also valid YAML header. Here we have `rmd_files` key that has as it's value a list of items. In yaml there is two ways one can specify a list.

```
---
rmd_files:
  - 00-index.Rmd
  - 01-introduction.Rmd
---
```

And

This is an alternative way to specify a list.

```
---
rmd_files: ["00-index.Rmd", "01-introduction.Rmd"}
---
```

6.1.3 Example 3

This on the other hand isn't valid YAML header

```
---
rmd_files:
  00-index.Rmd
  01-introduction.Rmd
---
```

6.2 Challenge: YAML 1

5 minutes

1. Is this a valid YAML header?

```
----
title: "Hello world!"
author: Me!
---
```

Yes, everything looks good to me

2. What about this YAML header?

```
----
title: "Hello world!"
author: Me!
bookdown::gitbook:
  config:
    toc:
      collapse: subsection
---
```

Yes, this is highly neted, but still a completely valid yaml header

3. How do you get a list of all possible keys and values (discussion question) ? Read the docs (will need to expand this answer)

6.3 General yaml header

This handful of tags are general between different document types. Most other key, values are aimed at the specific document type. In the next few sections we will look at yamls keys specific for Rmarkdown, bookdown and pdf final files configuration.

```
---
title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'
---
```

6.4 Challenge: YAML 2

5 minutes

3. Is this a valid yaml?

```
```
---
date: `r format(Sys.time(), '%d %B, %Y')`

```
```

It is certainly valid by Rmarkdown standards, but it may not be in other systems

6.5 Rmarkdown yaml header

Here we are starting to see another new key `output` with a value `html_document`

```
---
title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'
output: html_document
---
```

These are some of the possible values that `output` can take

- `html_notebook` Interactive R Notebooks
- `html_document` HTML document w/ Bootstrap CSS
- `pdf_document` PDF document (via LaTeX template)
- `word_document` Microsoft Word document (docx)
- `odt_document` OpenDocument Text document
- `rtf_document` Rich Text Format document
- `md_document` Markdown document (various flavors)

Turns out that each one of those values is a also a key that can take other values.

```
---
title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'
output:
  html_document:
    toc: true
    toc_depth: 3
    number_sections: yes
---
```

Note that indentation is very important here as it reflects the relationship between key value pairs.

6.6 Rmarkdown rendering

As we have discussed in the introduction Rmarkdown is an abmigious word. It could mean an rmarkdown R package[3] that converts or renders markdown into various document types. We can also be refering to an Rmarkdown document that we are writing. And we also could also use it in a more generic sence refering to the ecosystem.

Here I'm going to specifically talk about Rmarkdown R package and the `render` function i.e `rmarkdown::render()`. `render` can take an option for output format. You can either specify the output format by directly giving it a function i.e

```
rmarkdown::render(output_format = "html_document")
```

Or pass that same information via yaml header. As you might have noted above, `html_document` itself has a bunch of options. There would be a way to pass them manually in R console, but it is much nice and more reproducible to pass them via yaml header.

The two way to figure out which options are available for `html_document` is to either google (an obvious one) or take a look at the help page for that function i.e

```
?rmarkdown::html_document
```

6.7 Challenge: YAML 3

5 minutes

1. Can you find in the help page which values `df_print` from `html_document` can take? run `?rmarkdown::html_document` in Rstudio console to open help page and search for `df_print`

2. Set `df_print` to a value that will give you paginated HTML table. `paged`

6.8 More Rmarkdown tags

Let's add more options to our yaml file and `knit` it and see what has changed.

[3] https://github.com/rstudio/rmarkdown

```
---
title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'
output:
  html_document:
    df_print: paged
    toc: true
    toc_float: true
    toc_collapsed: true
    toc_depth: 3
    number_sections: false
    code_folding: "hide"
    theme: sandstone
    highlight: espresso
---
```

As I have eluted earlier if you simply change `output: html_document` to `output: pdf_document` you are going to get pdf document instead. Also remember that all of those options that we gave to `html_document` are specific to that document type only, although some tags might have the same name in other document types e.g `toc` and `toc_depth` keys also exist in `pdf_document`. We are not going to spend time on building a pdf document. It is a more finicky to build because it relying on LaTeX engine and LaTeX can be very finicky. This is more relevant for more complex documents. This is why we recommend to firstly build html document and only once you have finished you analysis and start building your pdf documents.

The more interesting, in my view, document type is `ioslides_presentation`

6.9 Presentation slides

As I've mentioned in previous section, `output` has many options, one of which is `ioslides_presentation`. Let's comment `html_document` with all it's options out for now, add `output: ioslides_presentation` and re-compile out document

```
---
title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'
output: ioslides_presentation
#output:
#  html_document:
```

```
#    df_print: paged
#    toc: true
#    toc_float: true
#    toc_collapsed: true
#    toc_depth: 3
#    number_sections: false
#    code_folding: "hide"
#    theme: sandstone
#    highlight: espresso
---
```

As you might have guessed `##` symbol in the case of `ioslides_presentation` means the beginning of the slide. While amount of work is minimal to convert between `html_document` and `ioslides_presentation` you will obviously need to reduce amount of text.

Part III

Part

In this section we are going to cover two types of referencing; cross-referencing tables, figures and other text and sections through out the document and citation i.e referencing exterinal resources. Cross referencing internal section is relativelly straight forward we just need to point to the resource using “speack” key, which will talk about shortly, whereas for referencing an external resource we need an additional bibliographies files that will hold citation information.

Chapter 7

Bibliographies

We have already talked about yaml files format being a plain text and structured file format. I've also mentioned json in passing, as being abother plain text and structured file format. One of the main purposed for having structure is so that computer can understand (parse) the information. bibtex[1] is yet another plain text, but structured file format, often times referred to as bibliographical database file. Bibtex originated in 1985, here is an interview with the author, Oren Patashnik, of bibtex if you interested to know how it came to be[2]. Whereas both yaml and json originated start of 2000's, 15 years later. The reason I am mentioning the dates is because in theory any one of those other formats, yaml or json, could have been easily re-purposed for citations. For example Citation File Format (CFF)[3] is in fact, exactly that, yaml based, citation format. There are many different citation file formats (10-20). Rmarkdown once again leverages an external tool pandoc-citepro[4] to generate and embed citation. pandoc-citepro[5] can work with several different file formats including bibtex[6], RIS[7] and EndNote[8]. Hopefully this gives you a bit of an overview what is possible and available regarding citations. In this workshop we are going to use bibtex[9] file format only. Note that very common file extension for bibtex[10] is `.bib` and this is the one we are going to use. Let's have a look at a typical `.bib` file content, the one shown below has two citation entries.

```
@Manual{R-base,
```

[1] http://www.bibtex.org/
[2] https://tug.org/interviews/patashnik.html
[3] https://citation-file-format.github.io/
[4] http://hackage.haskell.org/package/pandoc-citeproc
[5] http://hackage.haskell.org/package/pandoc-citeproc
[6] http://www.bibtex.org/
[7] https://en.wikipedia.org/wiki/RIS_(file_format)
[8] https://en.wikipedia.org/wiki/EndNote
[9] http://www.bibtex.org/
[10] http://www.bibtex.org/

```
  title = {R: A Language and Environment for Statistical
    Computing},
  author = {{R Core Team}},
  organization = {R Foundation for Statistical Computing},
  address = {Vienna, Austria},
  year = {2017},
  url = {https://www.R-project.org/},
}

@book{xie_allaire_grolemund_2019,
      place={Boca Raton},
      title={R Markdown: the definitive guide},
      publisher={CRC Press, Taylor & Francis Group},
      author={Xie, Yihui and Allaire, J. J. and Grolemund, Garrett},
      year={2019}
}
```

There are three core parts to any citation in the `.bib` file:

- `type` of the citation. Type begins with an `@` sign
- `key` used in our document to include citation. First item inside curly brackets
- `field` provide information about the thing that you are citing. Fields are key, value pairs

One thing to note about `.bib` file it is case-insensitive i.e book, Book and BOOK have the same meaning and effect.

7.1 Challenge: Bibliographies 1

5 minutes

1. Can you identify types of references shown in above? There are two citation with two different types `manual` and `book`

2. Can you identify keys for the references shown above? There are two citation with two unique key each, `R-base` and `xie_allaire_grolemund_2019`

3. Can you identify at least two different fields in any of the citations? Everything that has an equals sign is a fields.

7.2 Where do you get bibtex file?

There are two ways to get `.bib` files:

- manually curate one: remember that this is a plain text file format, so you can just type it out in a text editor and save as `.bib` file
- export out of your citation management or the publication website

I tried a couple of website, pubmed[11] and nature.com[12], neither of those appear to allow you to export `.bib`. Pubmed exports `.nbib`, which is a different file format if you look inside. Nature exports `.ris`, again not the one that we want. Google scholar[13] has a button **import into Bibtext** that will give the correct citation.

As I've mentioned above all of those file formats are actually supported[14] the only problem is figuring out which "key" you can use to include your citation into the document. And there isn't much resource out there to help you with other bibliography file formats. For now I would say the best and easiest option is to stick with `.bib` format.

Let's practice including citation into our Rmarkdown document. This is the paper that we are going to cite *Excuse me, do you have a moment to talk about version control?*. Go to this URL https://peerj.com/preprints/3159/, look for "Download" button on the right hand site, from the drop down select "BibTex". This will download citation file. On my computer the file name was `peerj-preprints-3159.bib`.

If you are curious to take a pick at the file content on your local computer, you can open such file with a text editor. Do not attempt to open with any other programs, such ash MS word or citation management tools. Most likely your computer will want to do just that, do not succumb! Mac people can use "TextMate" and windows people can use "Notepad" text editors. But this isn't necessary.

One you have `.bib` file, let's upload it into `rstudio.cloud`. Navigate to your "Files" panel, there you should see "Upload" button. Use that menu to upload your `.bib` file onto `rstudio.cloud`. You should see `.bib` file in your file browser if you are successful. If you double click on that file it should open up in a different panel, text editor panel, and you should see the content of that file as shown below.

```
@article{10.7287/peerj.preprints.3159v2,
 title = {Excuse me, do you have a moment to talk about version control?},
 author = {Bryan, Jennifer},
 year = 2017,
 month = aug,
 keywords = {Git, GitHub, workflow, data science, R Markdown, reproducibility, R language},
```

[11] https://www.ncbi.nlm.nih.gov/pubmed/
[12] https://www.nature.com
[13] https://scholar.google.com.au/
[14] https://github.com/jgm/pandoc-citeproc/blob/master/man/pandoc-citeproc.1.md

```
 abstract = {
        Data analysis, statistical research, and teaching statistics have at least one
      },
 volume = 5,
 pages = {e3159v2},
 journal = {PeerJ Preprints},
 issn = {2167-9843},
 url = {https://doi.org/10.7287/peerj.preprints.3159v2},
 doi = {10.7287/peerj.preprints.3159v2}
}
```

I hope every can identify "key" for this citation - `10.7287/peerj.preprints.3159v2`. To me this is very annoying key to use since it is not very memorable and hard to type. Let's edit this file in place and change that key to `Bryan2017`. Remember that key has to be unique per `.bib` file, but they can be any text. Make sure you save your changes (`ctrl^S`) and you can close this file now.

Let's include a reference to this `.bib` file in our Rmarkdown document via yaml header, using `bibliography` key.

```
---
bibliography: "peerj-preprints-3159.bib"
---
```

Now let's use our article key to cite in the text `[@Bryan2017]` and re-compile our html document.

7.3 Challenge: Bibliographies 2

5 minutes

1. Go to google scholar[15] and search for that same article *Excuse me, do you have a moment to talk about version control?*, find and export `.bib` citation and include it into our Rmarkdown document. Relatively hard task since I haven't explained how to include multiple `.bib` files into yaml header.

2. If you are manually curating `.bib` file, how do you know which types and fields are allowed? This resource can be useful

7.4 More about `.bib` and Rmarkdown

As was mentioned above one can append multiple references into a single `.bib` file or you can provide a list of `.bib` file

[15] https://scholar.google.com.au/

```
---
bibliography: ["peerj-preprints-3159.bib", "another_reference.bib"]
---
```

OR

```
---
bibliography:
  - "peerj-preprints-3159.bib"
  - "another_reference.bib"
---
```

If you want to reference multiple articles in the text each key should be separated by a semicolon (;)

```
[@Bryan2017; @bryan2018excuse]
```

You can include any text inside square brackets

```
[see @Bryan2017 p 12; also this ref @bryan2018excuse]
```

If you would like to suppress author's name you can use minus sign

```
[-@Bryan2017; -@bryan2018excuse]
```

You can also enable hyperlinking of the citation to the corresponding entry in the references as follows

```
---
link-citations: true
---
```

7.5 Citing R packages

R provides convenient `citation()` function that one can use to cite R packages e.g

```
citation("ggplot2")
```

7.6 Challenge: Bibliographies 3

2 minutes

1. Can you get `.bib` citation for `bookdown` package? You don't get `.bib` text content right away, but if you are careful at reading the message you will understand that you need to do this

```
citation("bookdown") %>% print(bibtex = TRUE)
```

Chapter 8

Bookdown

All `.Rmd` files located in the same directory will be compiled into the book in the (alphabetical?) order.

- files that start with an underscore are skipped
- if there is an `index.Rmd` it will always be treated as a first file
- those settings can be overwritten via `_bookdown.yml`
- `_bookdown.yml` must co-exist with `.Rmd` files in the "book" directory

Bookdown has extended markdown even further for math https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html#tab:theorem-envs

8.1 work in progress

Besides these html_document() options,

gitbook() has three other arguments:

- `split_by` argument specifies how you want to split the HTML output into multiple pages
  - `rmd`: use the base filenames of the input Rmd files to create the HTML filenames, e.g., generate chapter3.html for chapter3.Rmd.
  - `none`: do not split the HTML file (the book will be a single HTML file).
  - `chapter`: split the file by the first-level headers.
  - `section`: split the file by the second-level headers.

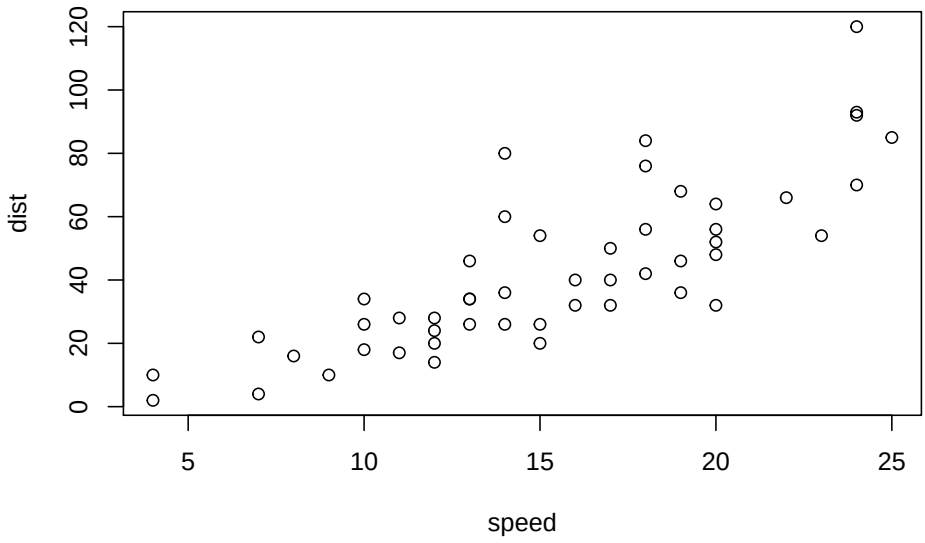


Figure 8.1: A scatterplot of the data `cars` using **base** R graphics.

  - `chapter+number` and `section+number`: similar to chapter and section, but the files will be numbered.

- `split_bib`
- `config`
- `collapse`
  - `section`
  - `subsection`
    scroll_highlight: yes before: null

8.2 Cross-references

A normal paragraph.

```
plot(cars)  # a scatterplot
```

8.3 check

need to address difference between `html_document2` vs `gitbook`

https://bookdown.org/yihui/rmarkdown/bookdown-output.html#bookdown-output

Chapter 9

Miscellaneous

In general think about fast turn around time and easi of editing.

this will remove numbering from that header

```
# Preface {-}
```

9.1 YAML

WE are using yaml language define certain parameters that meant to do to different tools

some are designated for pandoc other for bookdown package other general rmarkdown/knitr settings

9.2 LaTeX

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\begin{matrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{matrix}$$

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \end{bmatrix}$$

9.3 Tabbed sections

```
## Quarterly Results {.tabset}

### By Product

(tab content)

### By Region

(tab content)

## Quarterly Results {.tabset .tabset-fade .tabset-pills}
```

9.4 Figure options via yaml

> This sounds interesting

ok, I've tested out and fig_height and width via yaml do the same thing as when passed through chunk options. I guess yaml allows global defition, although one can set chunk options globally too..

also need to cover `out.width = "70%"`

pretty good resource about image resizing https://sebastiansauer.github.io/figure_sizing_knitr/

9.5 tables Rmarkdown

can't really describe at this stage where this is come from. it appears that it has links with pagedown and paged.js library

- `paged`

max.print The number of rows to print. rows.print The number of rows to display. cols.print The number of columns to display. cols.min.print The minimum number of columns to display. pages.print The number of pages to display under page navigation. paged.print When set to FALSE turns off paged tables. rownames.print When set to FALSE turns off row names.

References

Appendix A

Appendix

A.1 Long list of chunk options

name	value	type	description
child	NULL	code_evaluation	A character vector of filenames. Knitr will knit the f
code	NULL	code_evaluation	Set to R code. Knitr will replace the code in the chu
engine	'R'	code_evaluation	Knitr will evaluate the chunk in the named language
echo	TRUE	results	If FALSE, knitr will not display the code in the code
eval	TRUE	code_evaluation	If FALSE, knitr will not run the code in the code ch
include	TRUE	code_evaluation	If FALSE, knitr will run the chunk but not include t
purl	TRUE	code_evaluation	If FALSE, knitr will not include the chunk when run
collapse	FALSE	results	If TRUE, knitr will collapse all the source and outpu
results	'markup'	results	If 'hide', knitr will not display the code's results in t
error	TRUE	results	If FALSE, knitr will not display any error messages
message	TRUE	results	If FALSE, knitr will not display any messages genera
warning	TRUE	results	If FALSE, knitr will not display any warning messag
comment	'##'	code_decoration	A character string. Knitr will append the string to t
highlight	TRUE	code_decoration	If TRUE, knitr will highlight the source code in the
prompt	FALSE	code_decoration	If TRUE, knitr will add > to the start of each line o
strip.white	TRUE	code_decoration	If TRUE, knitr will remove white spaces that appear
tidy	FALSE	code_decoration	If TRUE, knitr will tidy code chunks for display wit
opts.label	NULL	chunks	The label of options set in knitr:: opts_template() t
R.options	NULL	chunks	Local R options to use with the chunk. Options are
ref.label	NULL	chunks	A character vector of labels of the chunks from whic
autodep	FALSE	cache	If TRUE, knitr will attempt to figure out dependenc
cache	FALSE	cache	If TRUE, knitr will cache the results to reuse in futu
cache.comments	NULL	cache	If FALSE, knitr will not rerun the chunk if only a co
cache.lazy	TRUE	cache	If TRUE, knitr will use lazyload() to load objects in
cache.path	'cache/'	cache	cache.vars NULL A character vector of object names
dependson	NULL	cache	A character vector of chunk labels to s
dev	'png'	plots	The R function name that will be used as a graphica
dev.args	NULL	plots	Arguments to be passed to the device, e.g. dev.args=
dpi	72	plots	A number for knitr to use as the dots per inch (dpi)
external	TRUE	plots	If TRUE, knitr will externalize tikz graphics to save
fig.align	'default'	plots	How to align graphics in the final document. One of
fig.cap	NULL	plots	A character string to be used as a figure caption in l
fig.env	'figure'	plots	The Latex environment for figures.
fig.ext	NULL	plots	The file extension for figure output, e.g. fig.ext='png
fig.height	7	plots	The height to use in R for plots created by the chun

A.2 Citation

A.2.1 Yaml header

You can also include citation into yaml header e.g

```
references:
- id: fenner2012a
  title: One-click science marketing
  author:
  - family: Fenner
    given: Martin
  container-title: Nature Materials
  volume: 11
  URL: 'http://dx.doi.org/10.1038/nmat3283'
  DOI: 10.1038/nmat3283
  issue: 4
  publisher: Nature Publishing Group
  page: 261-263
  type: article-journal
  issued:
    year: 2012
    month: 3
```

While this is handy for one or two citation, but this isn't practical for a study that has more then a few citations.

A.2.2 Changing citation style

Apparently if you go here[1] and download individual `.csl` files, specific for your citation style you then should be able to reference that file in yaml header

```
---
csl: "harvard-anglia-ruskin-university.csl"
---
```

OR

```
---
citation-style: "harvard-anglia-ruskin-university.csl"
---
```

[1] https://github.com/citation-style-language/styles

This is explained here[2]

However I wasn't able to change my style. I suspect there are some subtleties between bookdown and rmarkdown specifically `html_docment` and `html_docment2` outputs

Here[3] meant to be the solution by using

```
---
pandoc_args: [--csl=harvard-anglia-ruskin-university.csl]
---
```

But that also didn't work for me.

Also note that `biblio-style` "only applied to LaTeX output. For other output formats, you need to use the csl option in YAML or –csl in pandoc_args:"[4]

```
---
biblio-style: apalike
---
```

A.2.3 BibTex

This is a good resource[5] for manual bibtex curation.

A.3 Git and GitHub

A brief note about README files. It is regarded as a "silent" way of communication, where you can tell all necessary information another person need to know about your project. For a software tool you would put information about how to build that particular tool and dependencies. In our case we will add information how to build final html report. We will do this a bit later in the workshop.

A.4 Difference between Markdown and HTML

A.4.1 This is markdown

```
# Learning Markdown
```

[2] https://r4ds.had.co.nz/r-markdown.html#yaml-header
[3] https://stackoverflow.com/questions/48965247/use-csl-file-for-pdf-output-in-bookdown
[4] https://github.com/rstudio/bookdown/issues/354
[5] http://bib-it.sourceforge.net/help/fieldsAndEntryTypes.php

```
> I'm still learning

[External resource](https://rmarkdown.rstudio.com/)

Here I'll be learning:

- [markdown](#markdown)
- [Rmarkdown](#rmarkdown)
- [git and github](#git-and-github)

## Markdown

Here I'll learng _vanilla_ markdown

## Rmarkdown

Whereas here I'll be learning **R**markdown

## Git and GitHub

And this section is scary
```

A.4.2 This is simplified HTML

In actual fact there is more html tags required at the top of the document to turn it into fully functioning web-page

```
<div id="learning-markdown" class="section level1">
<h1>Learning Markdown</h1>
<blockquote>
<p>I'm still learning</p>
</blockquote>
<p><a href="https://rmarkdown.rstudio.com/">External resource</a></p>
<p>Here I'll be learning:</p>
<ul>
<li><a href="#markdown">markdown</a></li>
<li><a href="#rmarkdown">Rmarkdown</a></li>
<li><a href="#git-and-github">git and github</a></li>
</ul>
<div id="markdown" class="section level2">
<h2>Markdown</h2>
<p>Here I'll be learng <em>vanilla</em> markdown</p>
</div>
<div id="rmarkdown" class="section level2">
```

```
<h2>Rmarkdown</h2>
<p>Whereas here I'll be learning <strong>R</strong>markdown</p>
</div>
<div id="git-and-github" class="section level2">
<h2>Git and GitHub</h2>
<p>And this section is scary</p>
</div>
</div>
```

A.4.3 This is simplified LaTeX

Similar to HTML, there is more crafted goes at the top of the LaTeX document in order to turn into fully functioning document.

```
\begin{document}
\maketitle

\hypertarget{learning-markdown}{%
\section{Learning Markdown}\label{learning-markdown}}

\begin{quote}
I'm still learning
\end{quote}

\href{https://rmarkdown.rstudio.com/}{External resource}

Here I'll be learning:

\begin{itemize}
\tightlist
\item
  \protect\hyperlink{markdown}{markdown}
\item
  \protect\hyperlink{rmarkdown}{Rmarkdown}
\item
  \protect\hyperlink{git-and-github}{git and github}
\end{itemize}

\hypertarget{markdown}{%
\subsection{Markdown}\label{markdown}}

Here I'll be learng \emph{vanilla} markdown

\hypertarget{rmarkdown}{%
\subsection{Rmarkdown}\label{rmarkdown}}
```

```
Whereas here I'll be learning \textbf{R}markdown

\hypertarget{git-and-github}{%
\subsection{Git and GitHub}\label{git-and-github}}

And this section is scary


\end{document}
```