R, RStudio, R Markdown, Workflow and Github

Day 2

Solveig Bjørkholt

28. June 2022

Plan for today

- What is R and why use it?
- R and RStudio (installation)
- Coding in R
- Calling functions
- LaTex and R Markdown
- Github

What is R and RStudio?

R is a programming language, along with many other programming languages. Perhaps you've heard of some of them? Python, Java, Javascript, PHP, C, C++... We use these programming languages to communicate with the computer. Typically, there are lots of trade-offs with using one language over another, for example whether you want your code to be easily readable by a human, or whether it should make the requests run fast. Also, some programming languages are good for web development (e.g. Javascript and PHP), some work great for background processes (e.g. C and C++), and some are favorites for mathematics and statistical computing (e.g. python and R). We will be focusing on R in these sessions, though we have a long-term hope of integrating some python as well.

Learning R is a journey - and a quite frustrating journey at times, especially if it's the first programming language you learn. So why bother learning it? Why not just use excel or SPSS or another personal drag-and-drop favorite? Well...

- R is free. It's non-proprietary and requires no license. Down with the big corporations, knowledge for all!
- R enhances reproducibility. Anybody can re-run your code to use for their own applications, or to replicate your studies.
- R makes it easier to spot errors. Although errors are pain, we would rather have an error message early in the process than spotting a mistake two days before you think you've finished the project.
- R offers a large and supportive community to help you, both on the web and in real life. R-Meetups, RLadies, RStudio conference, you name it.

We often say that we "code in R", but we "work in RStudio". The reason for this is that R is the machine doing the calculations, while RStudio is the framework we work within. So if programming was a car, then R would be the engine and RStudio would be the bodywork.

Installing R and RStudio (adapted from Louisa Boulaziz)

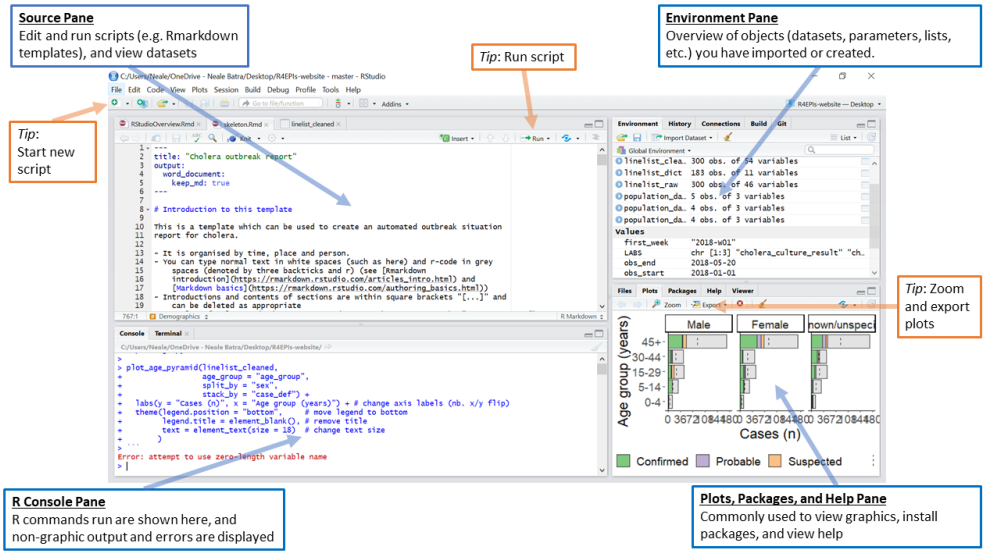
R

1. Open an internet browser and go to www.r-project.org
2. Click the "download R" link in the middle of the page under "Getting Started".
3. Select a CRAN location (a mirror site) and click the corresponding link. Here you should go to the Norway click on the link.
4. Click the "Download R for Windows" or the version for your computer. Link at the top of the page. If you have an old Mac, you need to read the next page carefully.
5. Click on the "install R for the first time" link at the top of the page (Windows). Mac users have to choose the version of R that is suitable for their software system. Old Macbook-users need to make sure that they click on the link corresponding to their software. If you are unsure about your software, click the apple in the left hand corner and go to "about this Mac". There it will say MacOS followed by the name of the software.
6. Click "Download R for Windows" and save the executable file somewhere on your computer. Run the .exe file and follow the installation instructions.
7. Now that R is installed, you need to download and install RStudio.

RStudio

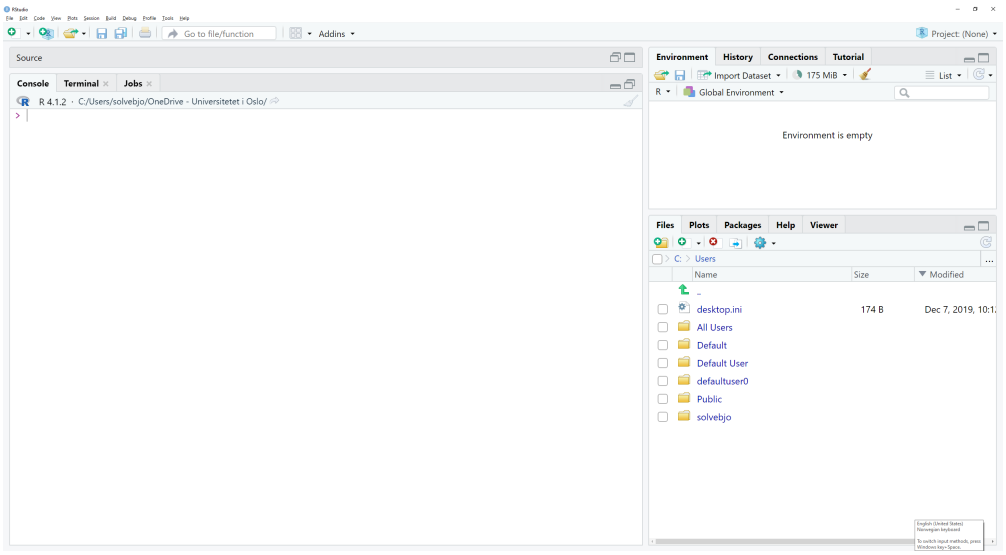
1. Go to www.rstudio.com and click on the "Download RStudio" button.
2. Click on the "Download RStudio Desktop".
3. Click on the version recommended for your system, or the latest Windows version, and save the executable file. Run the .exe file and follow the installation instructions.

How RStudio works

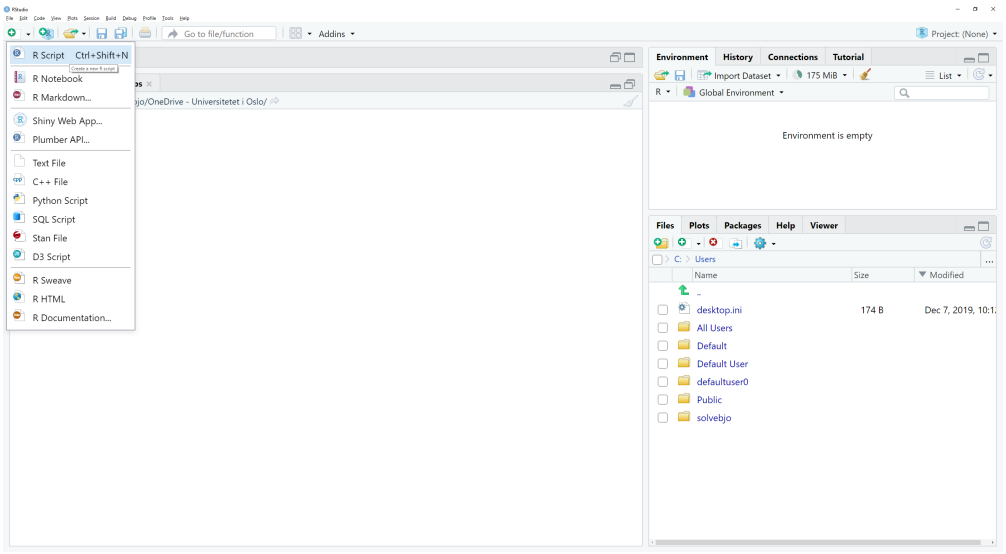


Writing and running code

When you have R and RStudio installed, open RStudio. Remember that we code in R, but work in RStudio, so this is the program you typically open. Your screen should look something like this:



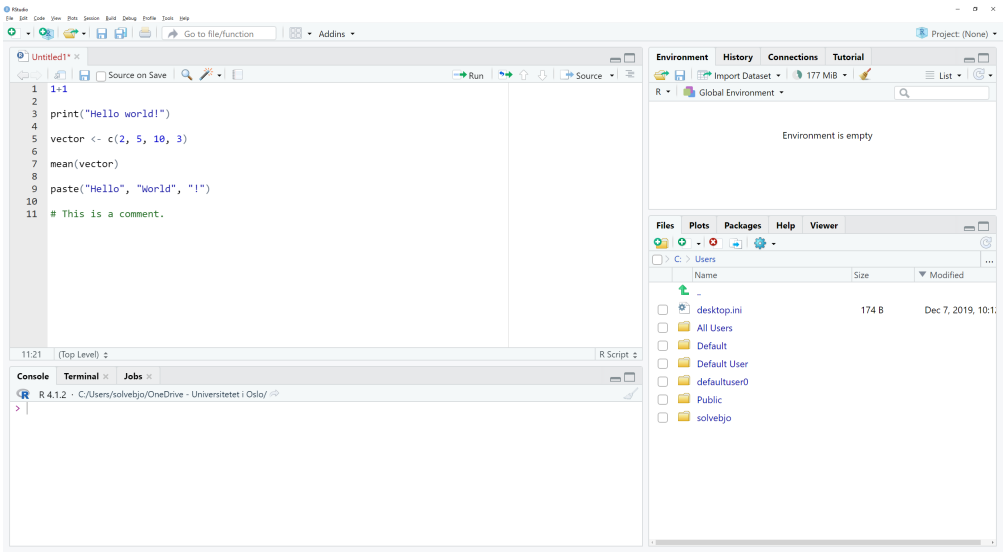
We want to write our code in a script. A script is kind of like a word document, just for code. Scripts can be saved, stored and shared with others, which is preferable to the alternative - to write your code, execute it and then lose it once you close your programs and go home for the day. To start a script in RStudio, go to "File" and choose "New file", "R Script". Alternatively, click on the document with a green plus-sign in the top left hand corner and choose "R Script" - as shown below.



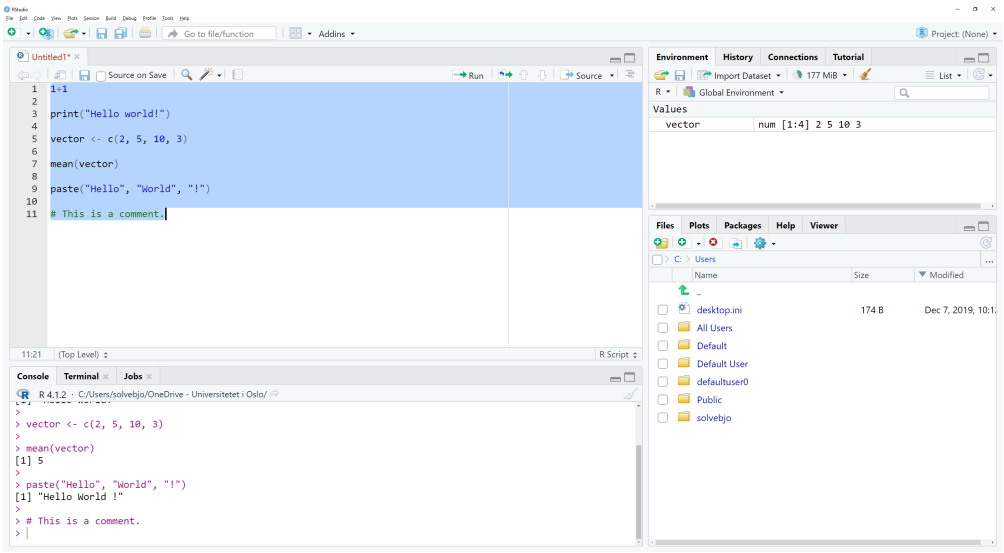
The script emerges on the top left hand corner of your RStudio interface. Try typing the following into your script:

```
1 + 1

print("Hello world!")

vector <- c(2, 5, 10, 3)

mean(vector)
```

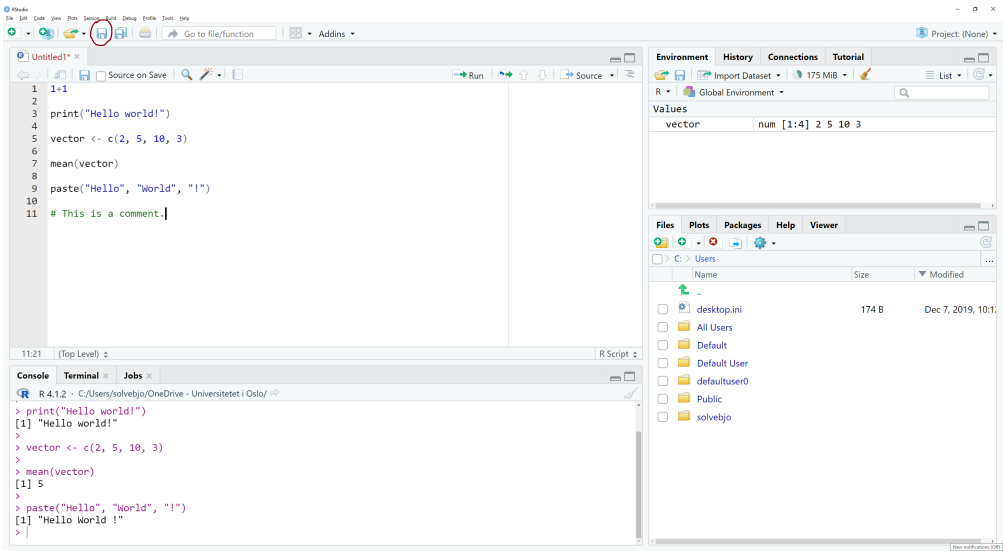
```r
paste("Hello", "World", "!")

# This is a comment.
```



Then mark the code and click CNTR+ENTER (i.e. run the code!). The output from your code appears in the bottom left hand corner of your RStudio interface. This is called the "Console". It's actually where R works, so you can view it as the engine of the car. All output from your code will appear here.



To save the script, hit the blue disc-symbol as shown in the picture below.



Objects

Notice that something has appeared in the upper right hand corner of your RStudio interface. This is an object. It has emerged because of this line in our script:

```
vector <- c(2, 5, 10, 3)
```

This line, because of the arrow (`<-`), assigns values to an object and stores it in the short-term memory of R. The box with the object is the short-term memory, also known as "Environment". We frequently assign values to objects in R, so it's good to familiarize yourself with this now. Think: Whenever you want to use an object later, make an object of it.

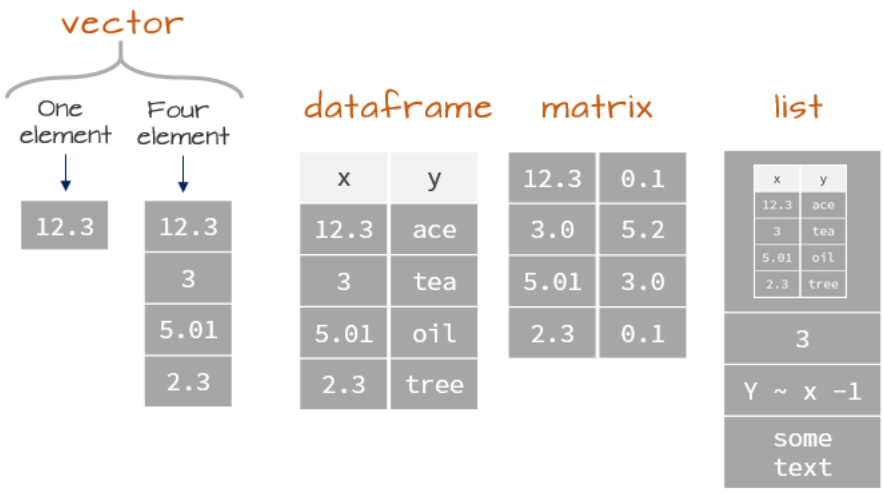
In general, to make an object, choose whichever name you like, add an arrow towards the name of the object, and then add the operation you want done. If you want to combine several elements, e.g. several numbers or several words, you need to put a c in front of your parentheses. `c` stands for "combine". For example:

```
numberone <- 1

numbers <- c(1, 2, 3, 4, 5)

meanofnumbers <- mean(c(1, 2, 3, 4, 5))

names <- c("Billy", "Joe", "Vera")
```

Why did we call the object "vector" in the first place? Because it *is* a vector. You can think of vectors as a sequence of information, for example a bunch of numbers or a bunch of names. `numberone`, `meanofnumbers`, `numbers` and `names` are all vectors. There are other types of objects in R. The ones we will learn are: vector, dataframe, matrix and list. The difference between them is this:

- Vectors have the same data type and amount to *one* variable.
- Dataframes can have different data types and amount to *several* variables.
- Matrices have the same data type and amount to *several* variables.
- Lists have several types of data types and amount to *one* or *several* variables.[1]

[1]Some illustrations of data objects in R also operate with arrays, but we will not focus on that here.



To know what an object contains, try writing the name of the object and run it[2].

Data types

What does "data type" mean, then? Well, R has several data types and we will focus on four[3]. These are `numeric`, `integer`, `character`, `factor` and `logical`, and they are also called *classes*. They differ from each other in the types of data they store, be it strings, numbers or logical values.

Data type	Properties
`numeric`	contains decimal and whole numbers, also called "double"
`integer`	contains only whole numbers
`character`	contains character strings ("words")
`factor`	contains categories (either ranked or unranked)
`logical`	contains only two values, TRUE and FALSE

Numbers and integers can be one number, sequences of numbers, or mathematical expressions. For example:

```
100

1 + 100 + 20

1.4353  # commas are written as dots (.)

1.5 + 2.9 - 8.3

2 * 8  # multiply is written as star (*)

9/3  # divide is written as slash (/)

c(3, 100, 203.4, 15000)
```

Strings are sequences of characters, often words. We need to surround them in quotation marks. For example:

[2]To "run a code" means to mark it and hit CNTR+ENTER
[3]Some people also add "complex" as a data type, but this is outside our scope.

```
"A"

c("A", "B")  # If 'A' and 'B' were two categories, this object could also be a factor.

"Hello"

"We are learning R!"

c("Hello.", "We are learning R!")
```

Logical vectors take two values; TRUE and FALSE.

```
TRUE

FALSE

c(TRUE, FALSE, FALSE, FALSE, TRUE)
```

To check the data type, use the function `class()`.

```
class(100)
```

```
## [1] "numeric"
```

```
class("A")
```

```
## [1] "character"
```

```
class(FALSE)
```

```
## [1] "logical"
```

Logical operators

At last, we can use different operators to check whether to objects are equal to each other. We wil not delve into this now, but I leave this here because it might be useful later.

Operator	Meaning
==	equals
<	less than
>	bigger than
<=	less than or equals
>=	bigger than or equals
!=	does not equal
!x	does not equal x
|	or
&	and

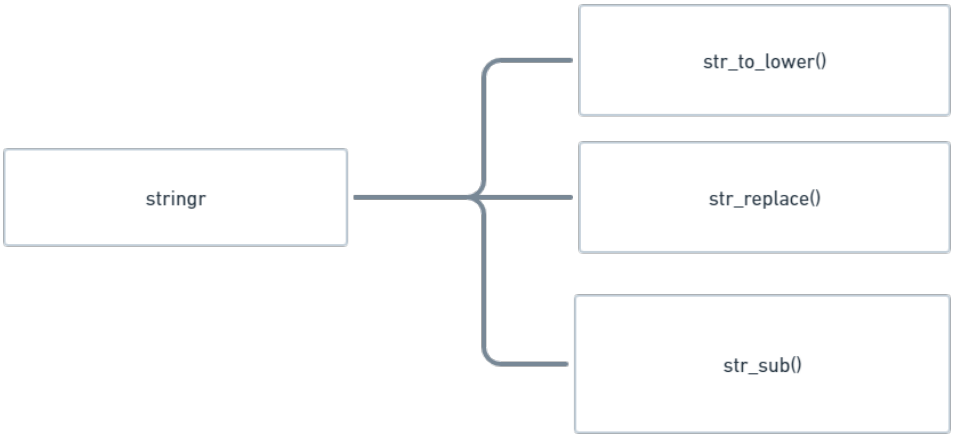
Calling functions

As you can see, we can use R as a calculator if we want to. Writing `1+1` gives `2` in the Console. But the real power of R comes from running functions. A function is a command that tells R what to do with something. We've already seen a few functions so far, for example `print()`, `mean()` and `class()`. `print()` tells R to give us the output of an object, `mean()` tells R to calculate the average of an object, and `class()` tells R to give us the data type of an object.

Learning functions is a bit like learning vocabulary in a language, so don't fret if you find yourself grasping for the name of a function. It's the same as when you somewhat know a language and can't really recall a word. The solution? Ask someone or look it up on the internet.

Packages

Most functions we get from packages. Packages are bundles of functions that we import to R, and they are made by the wide community of people out there who work on R-content. For example, `stringr` is a package that gives us a lot of functions we can use to work with strings (i.e. character things). Some of these functions are `str_to_lower()`, which make all characters into lower case, `str_repalce`, which takes part of a string and replaces it with something else, and `sub_str()`, which takes out a part of a string.



To get a package into R, we first have to install it (from the world wide web), then import it (telling R that we want to use it in this script).

To install a package, use the function `install.packages()` and put the name of the package in quotation marks. You only need to do this *once* (unless you uninstall and reinstall R, or want to update your packages to newer versions). So no need to populate your script with `install.packages()`.

```
install.packages("string")
```

To import a package, use the function `library()`. Here, you do not need quotation marks.

```
library(stringr)
```

```
## Warning: package 'stringr' was built under R version 4.1.3
```

Once this is in order, we can start using functions from the package `stringr`. A full overview of all the functions in a package is available on the internet through a document that all package-makers have to make for their package. In the case of `stringr`, a quick internet search leads us to this document.

Let's try some of the functions!

```
string <- "This is a string, meaning a sequence of characters, typically words. It is also a vector, si

print(string)
```

```
## [1] "This is a string, meaning a sequence of characters, typically words. It is also a vector, since
```

```
str_to_lower(string)  # Sets all characters to lower case.
```

```
## [1] "this is a string, meaning a sequence of characters, typically words. it is also a vector, since
```

```
str_replace(string, "This is a string, meaning a", "A string is a")  # Replaces some parts of a string
```

```
## [1] "A string is a sequence of characters, typically words. It is also a vector, since it contains i
```

```
str_sub(string, 1, 16)  # Picks out characters from place 1 to place 16.
```

```
## [1] "This is a string"
```

Your own functions

You can make your own functions too.

```
make_lower_and_pick_start <- function(x) {

    x <- str_to_lower(x)
    x <- str_sub(x, 1, 16)

    return(x)

}

make_lower_and_pick_start(string)
```

```
## [1] "this is a string"
```

Installing LaTex

Now that we know how to work with packages, let's tackle the fact that we're going to write in RMarkdown. The great thing about R Markdown is that we can produce nice reports, for example in PDF. However, we need LaTex to make R Markdown reports in PDF, so we need to install this as well.

LaTex is a document preparation system, much like word, except it is more code-heavy and allows you to create beautiful documents. TinyTex is a custom LaTeX distribution, so this is what we'll install. See https://yihui.org/tinytex/ for more informaton.

To install TinyTex from R, we need to install the package `tinytex` and use the function `install_tinytex()`.

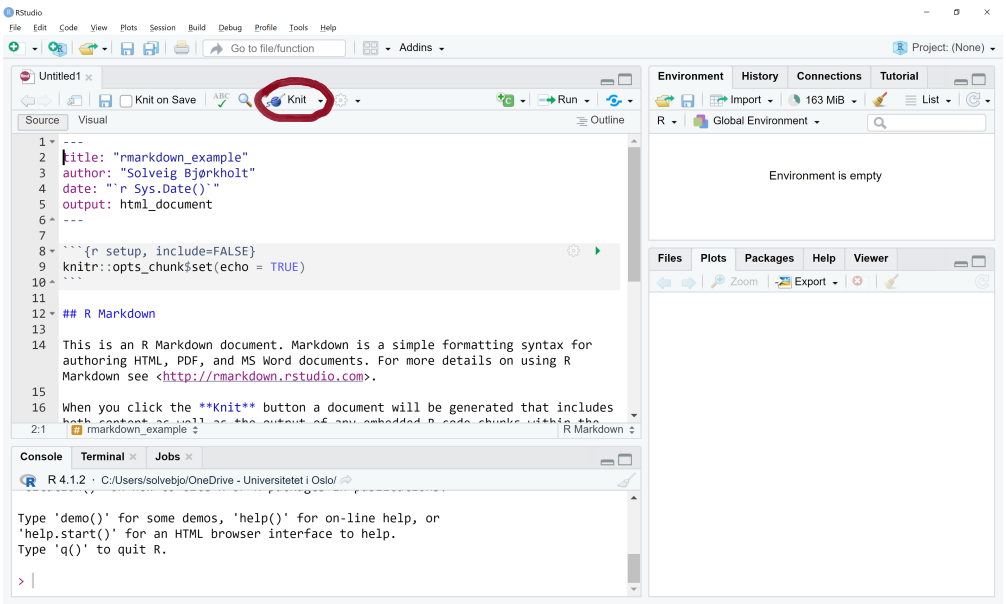
```
install.packages("tinytex")
install_tinytex()
# to uninstall TinyTeX, run tinytex::uninstall_tinytex()
```

Introduction to R Markdown

R Markdown is a type of script that contains both text and code, and that can be turned into both documents and web pages. It's very handy in that regard. To open an R Markdown file in RStudio, click "File", "New file" and "R Markdown". Or, alternatively, click the document sign on the top left corner of your RStudio interface and choose "R Markdown". You'll get a box asking you for the title of your document. Call it whatever you like, then hit OK.

For those curious: R Markdown is a way to integrate writing in Markdown through R. Markdown is a markup language which, guess what, is also a type of computer language! In other words, we use it to speak with the computer, telling the computer the different properties of our text, be it a title, **bold**, *italics* or `code`. However, in contrast to for example R, python and Java, markup languages are much easier to read by humans. Other examples of markup languages are HTML and XML. This is not super important to know in order to write in R Markdown, but it's nice with a bit of context now and then.

Now, your RStudio interface should look something like the picture below. Notice the "Knit" button at the top, and try clicking it. It will prompt a request to save your file on your computer, and once you've done that, the R Markdown will convert your code into a nice HTML report[4] Navigate to where you saved your file, click on the name of the file with the ".html" ending, and open it in an internet browser (such as Firefox or Chrome).



To modify the report, here are a few things to know:

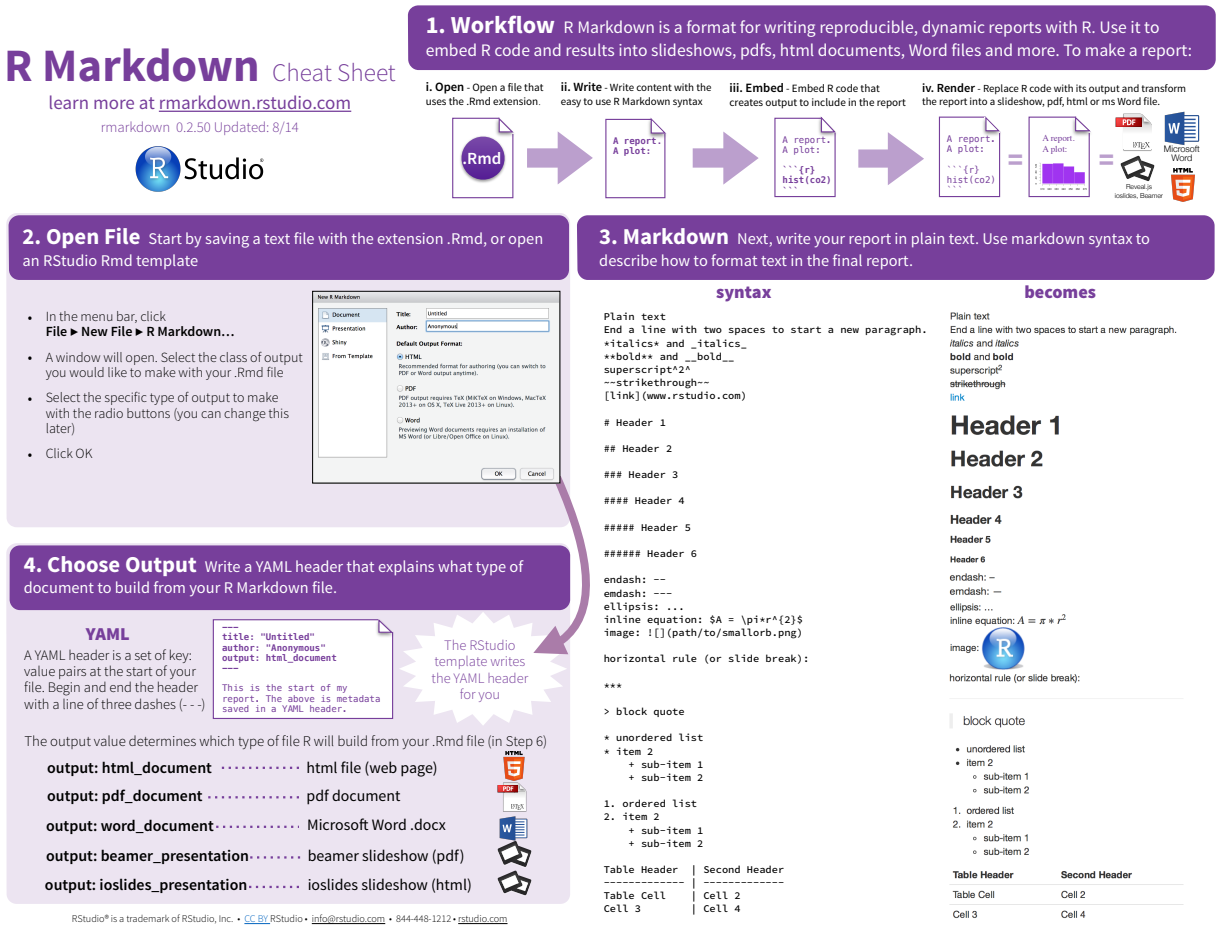
- In the beginning of the document, the `knitr::opts_chuck$set(echo = FALSE)` tells R to use the `opts_chuck` function from the `knitr` package, and to modify the object set. In here, you can modify what you want the general settings of the document to be. `echo = FALSE` means that you do not want code to show typically, just the output of it. Other options are for example `include` (whether to typically include chunks) and `warnings` (whether to typically show warning messages from code).

[4]HTML is the language of the web. Most webpages are written in HTML.

To add a chuck of code, add the hyphens, curly parentheses and the r, and end with three curly parentheses as well. In the middle, write your code.

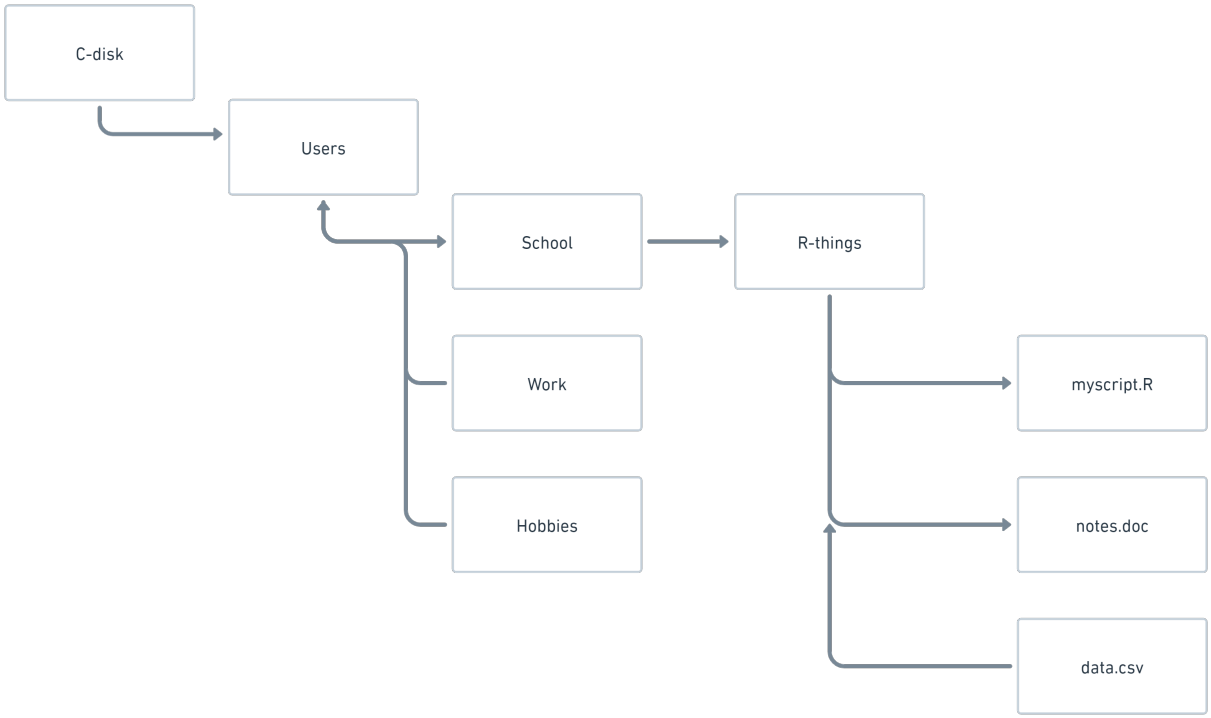
```{r}
```

For more settings, consult the image below, also available here, and search around on the internet whenever you wonder about something. The best way to learn is to just start and look up things as you need them.



Paths and folders

Your computer stores everything you work with in folders (also called "directories"). If you open File Explorer on Windows or Finder App on Mac, you'll see the archive that your computer is storing for you. The archive has an arrangement which makes it easy to search through your files – a tree-like structure. You have a folder, and within that folder you can have another folder, and within that folder, another folder, until there is a file. This tree-structure, from root to branch, is often called a "path". For example, the path to your "myscript.R" could look something like the structure below:



Here, C-disk is the root and the file is myscript.R. The path would be: C-disk/Users/School/R-things/myscript.R

This is important, because as with any archive, you need to know where you save things. It's also very important because when we import things into R, for example a dataset, you need to tell the computer where to find the file with that dataset. We do that by specifying the path to the file. For example, if we would want to read data.csv into R, the code would be:

```
dataset <- read_csv("C-disk/Users/School/R-things/data.csv")
```

Notice that in the code above, `read_csv` is the function that reads a dataset into R, the argument is the path to the file, and we assign the dataset to an object called "dataset" by the `dataset <-` part.

In RStudio, the window in the right hand corner mirrors the folders on your computer under the tab "Files". You can make adjustments here just as you can in your folder structure.

Github

We're going to work in Github. This is a version control space, meaning that it's a place where you can share your code with others and keep track of who changes what, when, how, and whether it was a change that we would really want to keep. It enables you to share your code with the world, and can be an excellent way to build your CV. Moreover, if you want to get up and serious with your project, you can integrate Github with many other platforms such as Amazon and Google Cloud, creating fluent pipelines for you work. So, there are at least three main benefits to Github:

- It makes collaboration easier.
- It gives your work exposure and publicity.
- It can be an important component in integration with other platforms.

To work in Github, we need to first create a user account. Then, your team needs to create a repository that can be shared among you. A repository is a bit like a folder on Github.

Create a user account for Github

1. Open https://github.com in a web browser, and then select Sign up.
2. Enter your email address.
3. Create a password for your new GitHub account, and Enter a username, too. Next, choose whether you want to receive updates and announcements via email, and then select Continue.
4. Verify your account by solving a puzzle. Select the Start Puzzle button to do so, and then follow the prompts.
5. After you verify your account, select the Create account button.
6. Next, GitHub sends a launch code to your email address. Type that launch code in the Enter code dialog, and then press Enter.
7. GitHub asks you some questions to help tailor your experience. Choose the answers that apply to you in the following dialogs:

- How many team members will be working with you?
- What specific features are you interested in using?

8. On the "Where teams collaborate and ship" screen, you can choose whether you want to use the Free account or the Team account. To choose the Free account, select the Skip personalization button.
9. GitHub opens a personalized page in your browser.

Create a repository

1. Click on your profile button in the right hand corner of your Github screen and choose "Your repositories".
2. Click the green button "New".
3. Type a short, memorable name for your repository.
4. Optionally, add a description of your repository.
5. Choose a repository visibility. Here, you can pick "Public".
6. Under "Initialize this repository with", choose "Add a README file".
7. Click "Create repository".

Now, to add collaborators:

1. Collect the Github usernames for the rest of the team.
2. Navigate to the main page of the repository.
3. Under your repository name, click Settings.
4. In the "Access" section of the sidebar, click "Collaborators".
5. Click "Add people".

Once the other collaborators have accepted the invitation to collaborate, everyone should be able to contribute.

Now, even though we have a shared Github repository, we still work and code in peace on our own computers. But whenever we make a change in the code and think it's time to share it with the rest of the team (for example because we need feedback or because we believe a piece of code to be done), we can choose to (1) add the change to the Github repository, (2) commit to the change and (3) push the change. This is the general flow of working with Github.

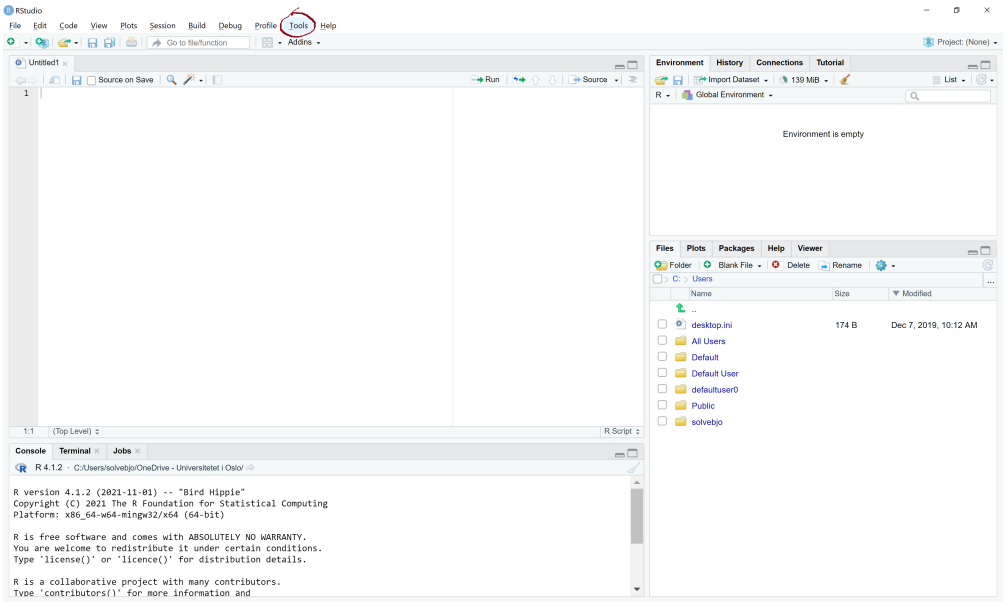
How do we add, commit and push a change from our local computer to the Github repository? By cloning the repository to our computer. This is possible to do in the terminal, but we will here show you how to do it in RStudio.

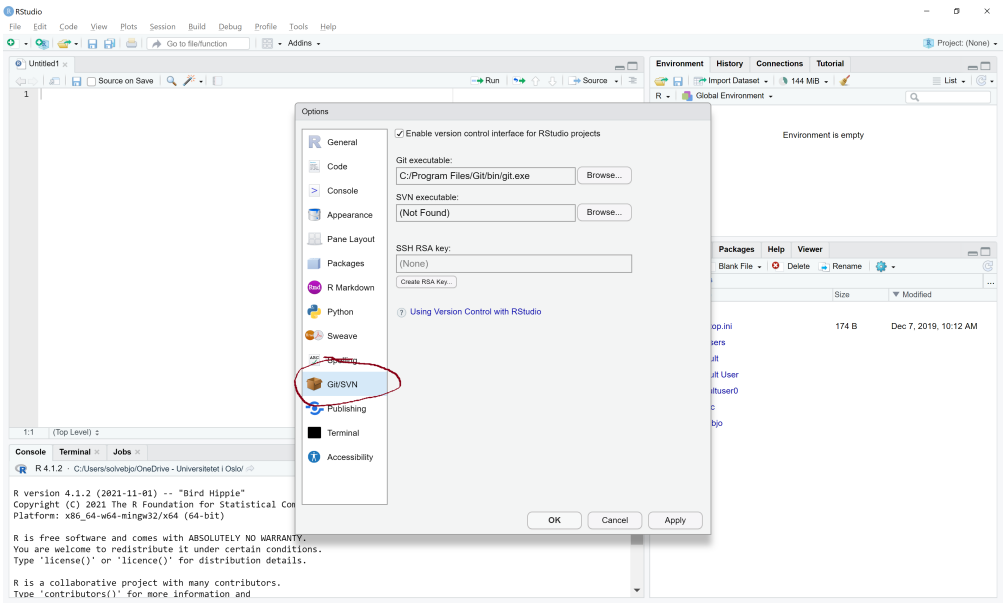
Github and RStudio

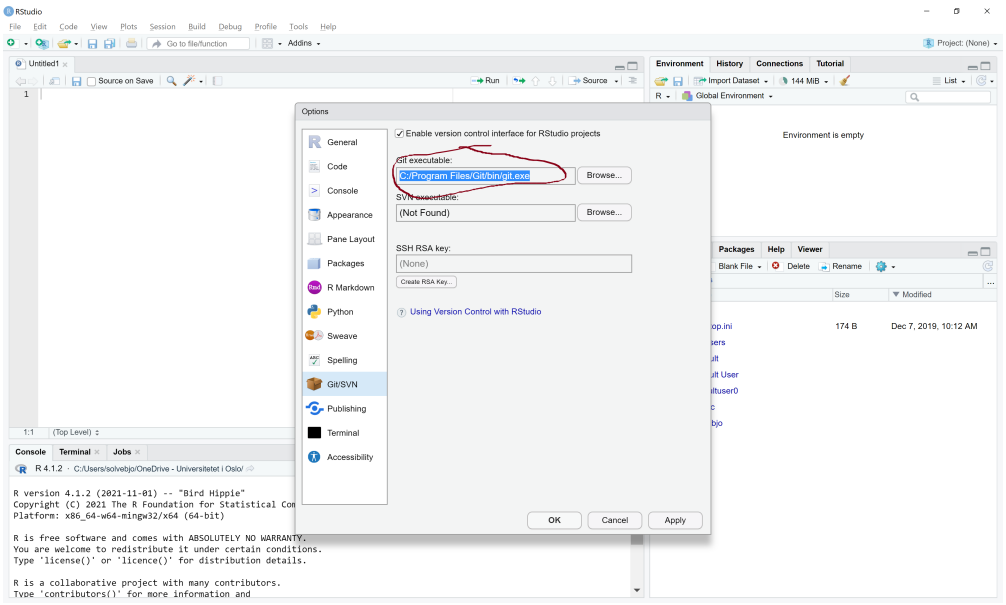
Github can work with many platforms, including RStudio! To get there, however, the first thing we need to do is to install git, the system that allows us to work with Github from our computer. So Github is the version control **space**, and git is the version control **system**. Git can be downloaded from here: http://git-scm.com/downloads

Once you've installed git, you'll need to activate it in RStudio:

1. Find the Tools menu in the top of the RStudio interface and go to Global Options.
2. Click Git/SVN.
3. Click Enable version control interface for RStudio projects.
4. Under "Git executable", if there is no path there, enter the path for your the git file that you downloaded earlier.

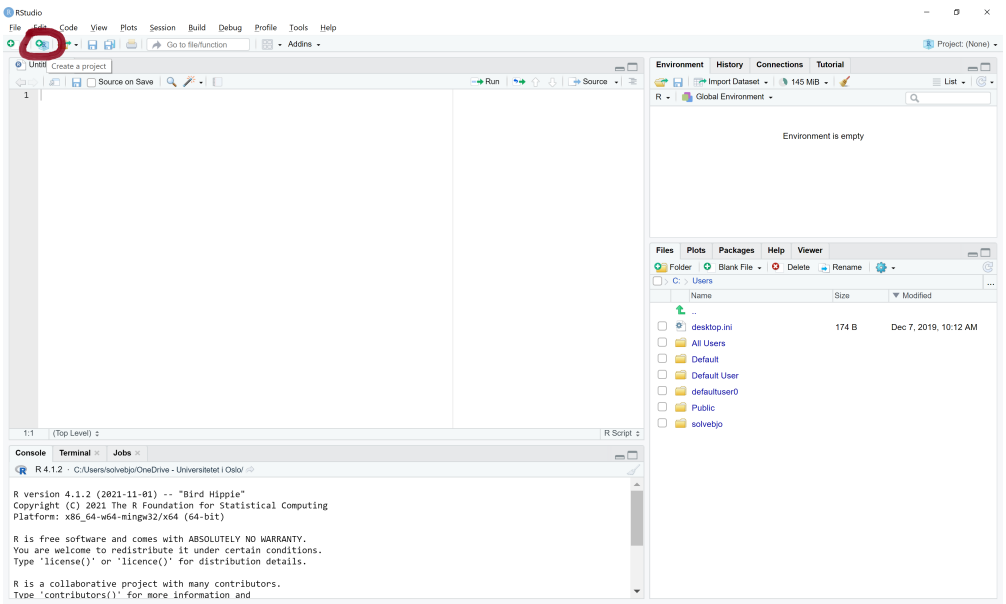




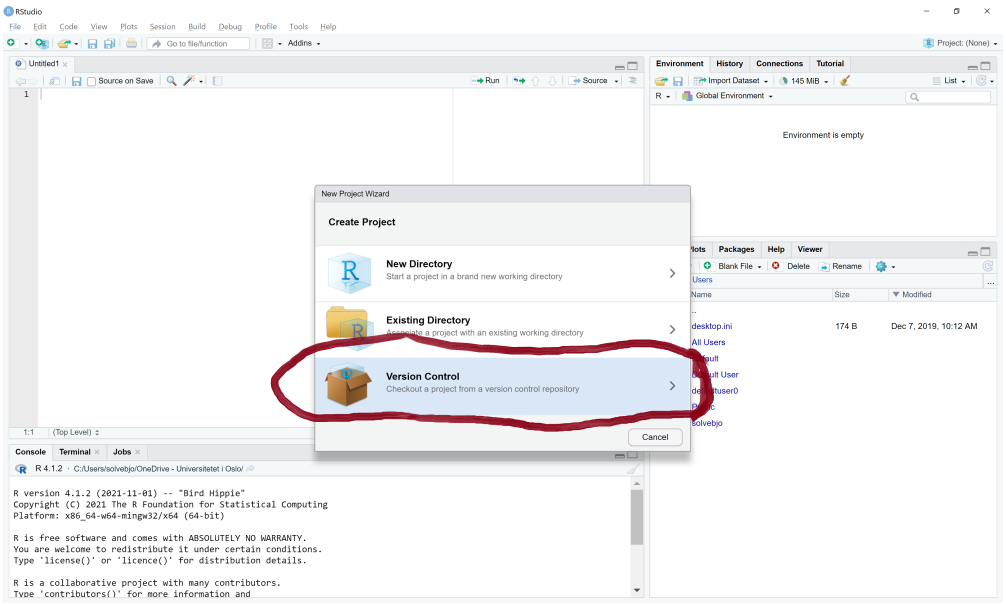


When this is up and working, we need to use our newly installed git to do the actual cloning the Github repository to our own computer. To do this, you need to create a new project in RStudio that is tied to the repository. This is how to do that:

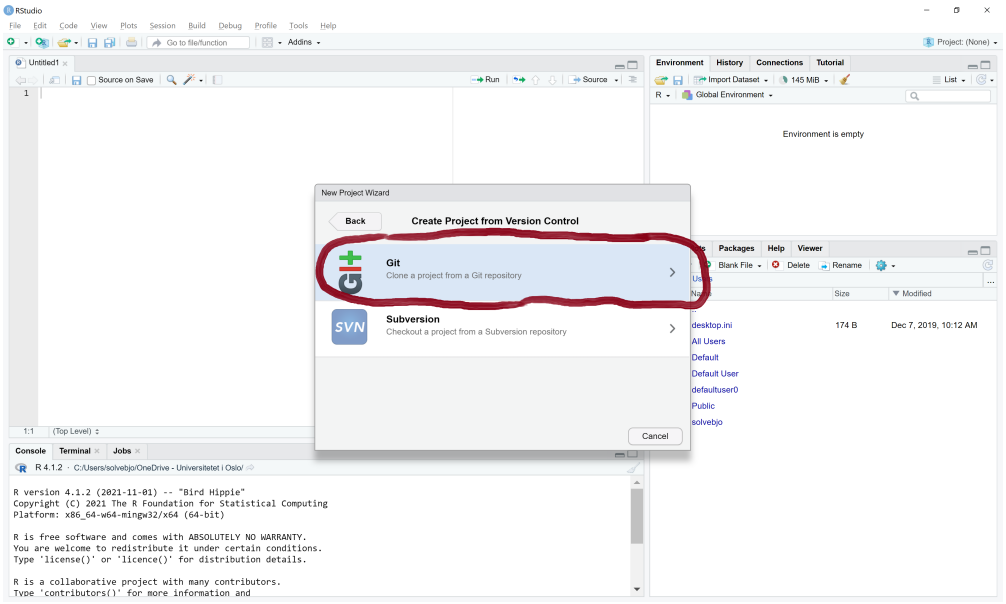
1. Under "File" in the upper left hand corner of your RStudio, choose "New project". Alteratively, click on the icon of an R in a box with a green plus sign.



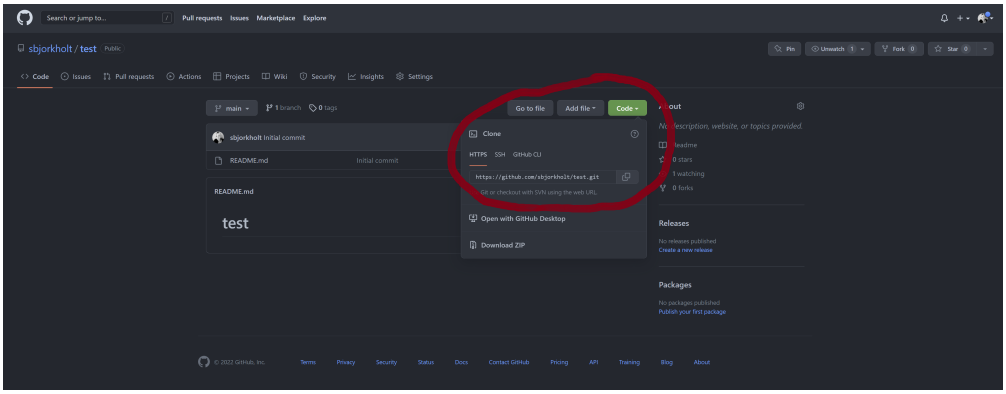
2. Choose "Version Control".

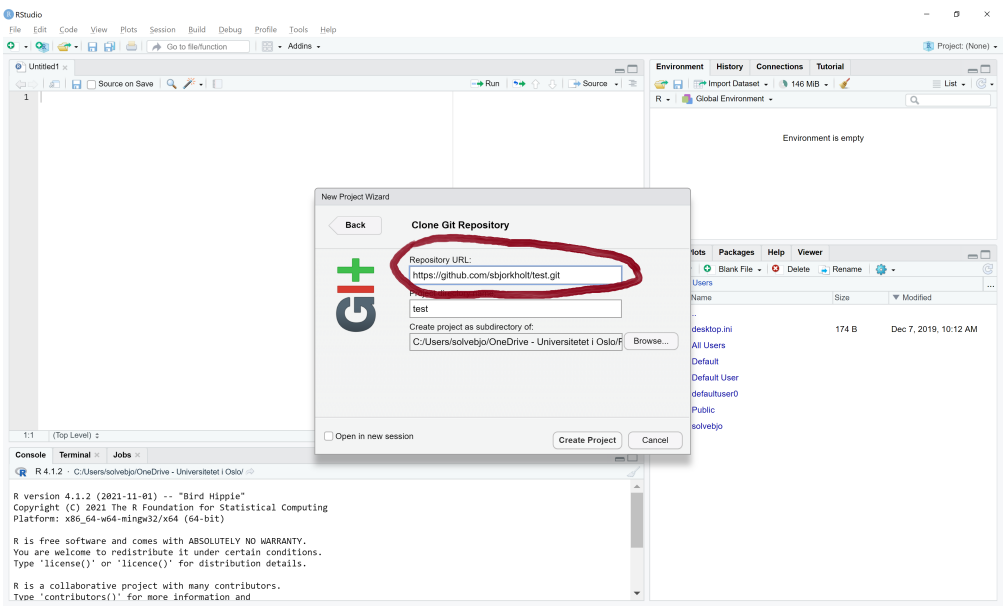


3. Choose “Git”.



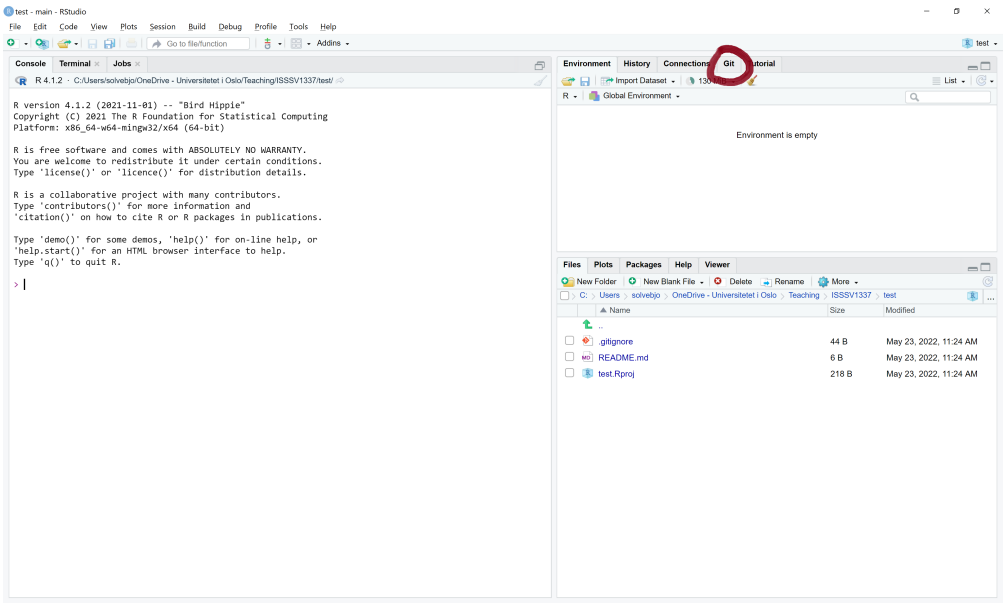
4. Here, you are asked for the Repository URL. To find that, go back to the main page in your Repository in Github, and click on the green “Code” button. Copy the URL-path shown there. Then, paste it into the space for your Repository URL in RStudio.





5. Use the “Browse” button to choose where on your computer you want to place your folder. It should be a place you can easily find again. And it should not be under “Downloads”. Recall that a Repository in Github is kind of like a folder, so when you clone the repository like this, you will create a folder on your computer that matches the folder on Github.

And now, you’ve successfully cloned the Github repository to your own computer. In the upper right hand corner of your Rstudio, there is now a tab called “Git”. Moreover, in the bottom right hand corner, under “Files”, you find the folder on your computer which matches the Github repository.



To get the latest updates from your repository, click "Pull". Make it a habit to click "Pull" regularly. It will spare you the pain of so-called "merge conflicts". If you want to add your code to the common repository:

1. Click "Pull".
2. Click "Commit".
3. Add a description of what you've done and why.
4. Click "Push".

See the video on this page for a demonstration.

Some of you, if not all, will almost certainly get trouble with Github during this course. We will deal with them as they arise. Since this is a learning space which increases the risk of errors, we also recommend keeping a backup of your repository. This can seem cumbersome, but it's a part of the learning process. In the end, working with Github will prove vastly superiour to many alternatives, such as sending code over mail.