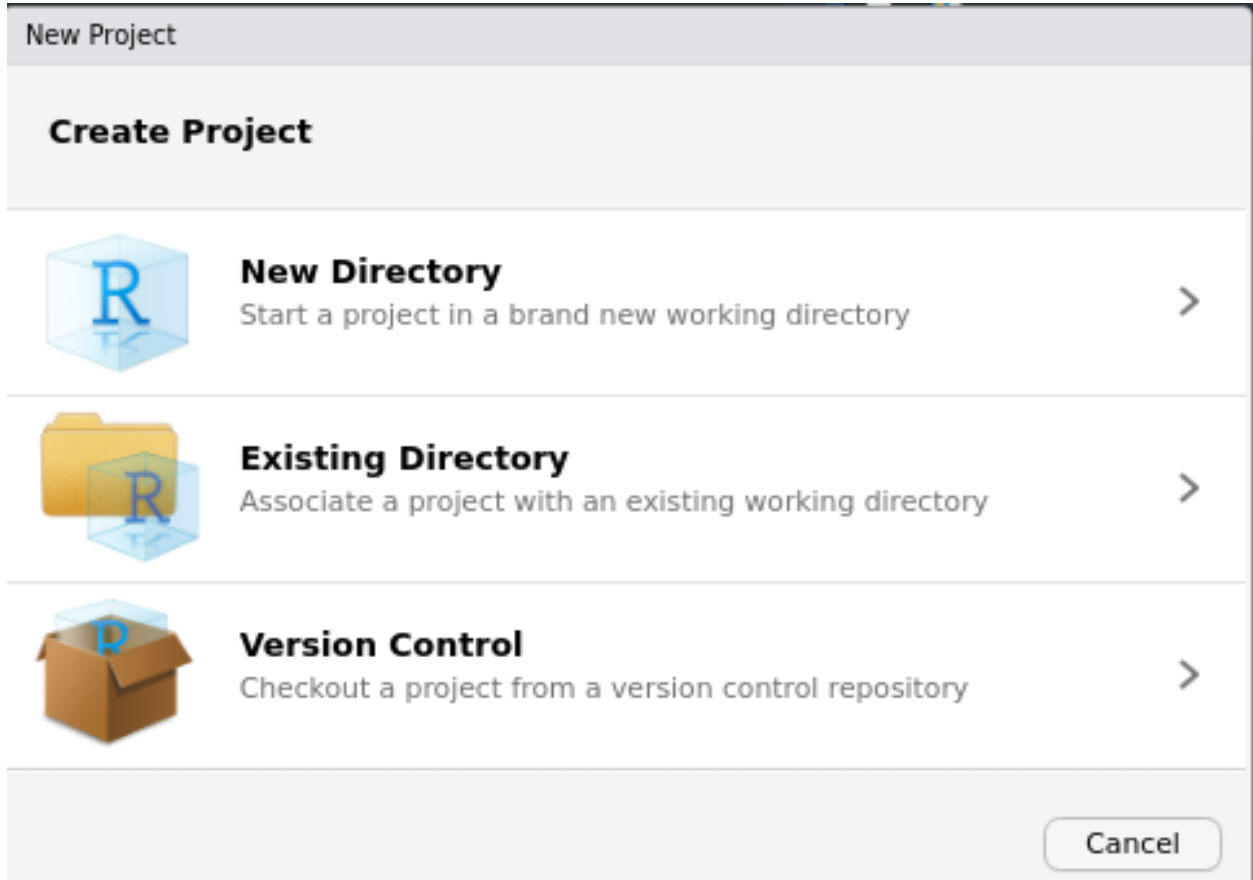


Packages

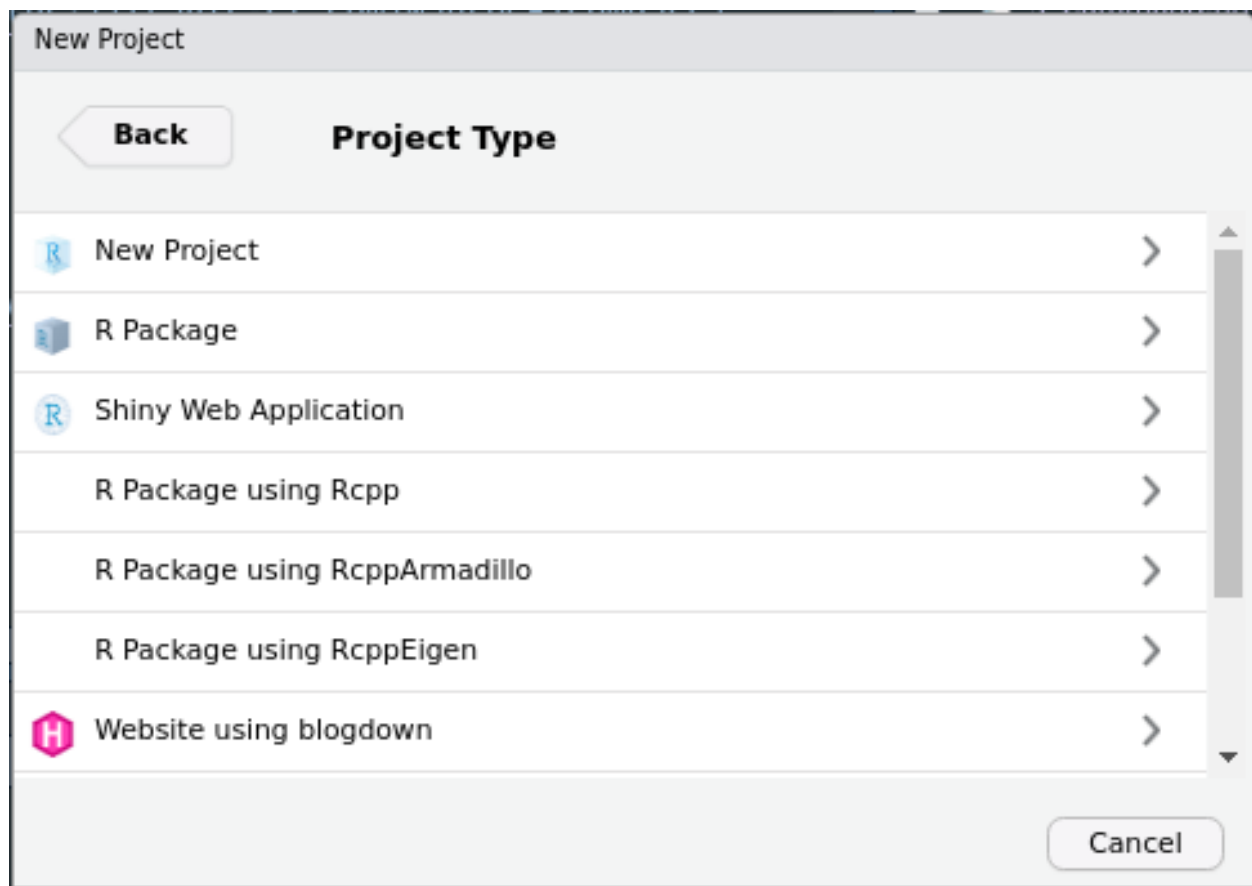
Creating an R Package with RStudio

To Create a package open RStudio, on the menu browse

File > New Project > New Directory




Then, from the following options choose R Package.



At this point, we need to assign the R Package a name and tell RStudio where to place the main folder containing our package files without our folder structure. Here, I've chosen to put it under the folder `University` and to call it `PerfectRPackage`.

New Project

Back **Create R Package**

 Type: **Package** Package name: **PerfectRPackage**

Create package based on source files:

Add... **Remove**

Create project as subdirectory of:

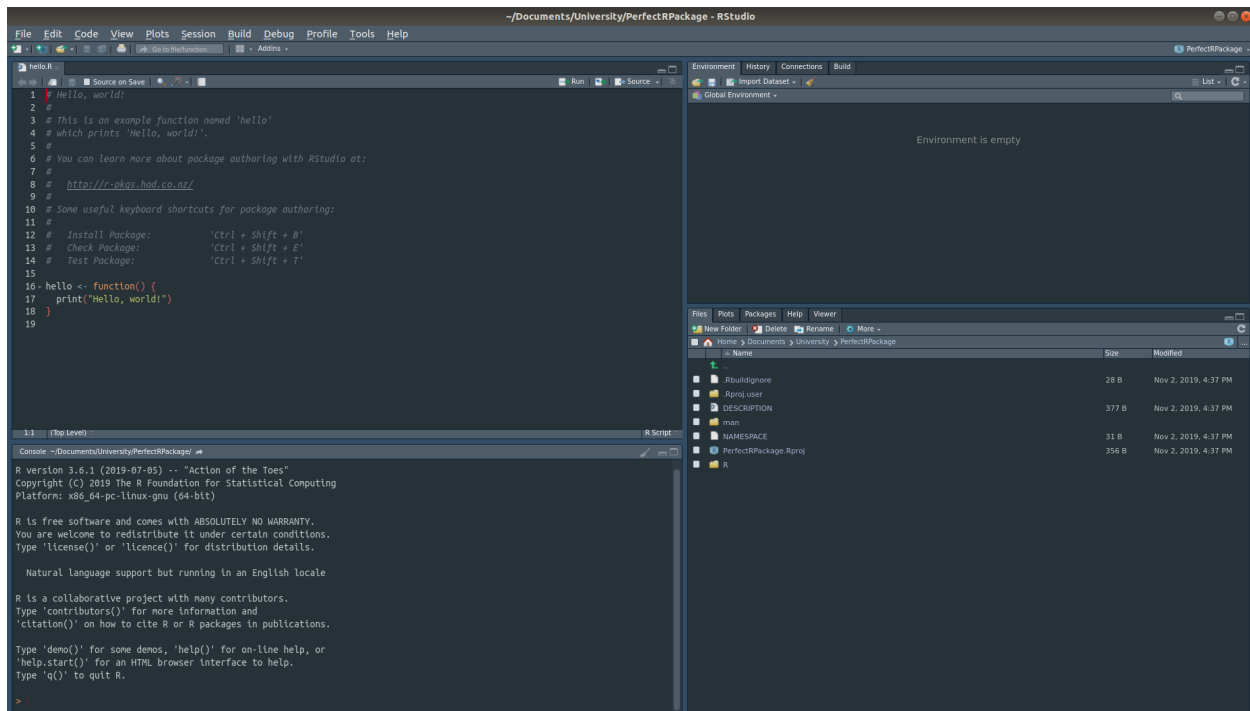
Browse...

☐ Create a git repository

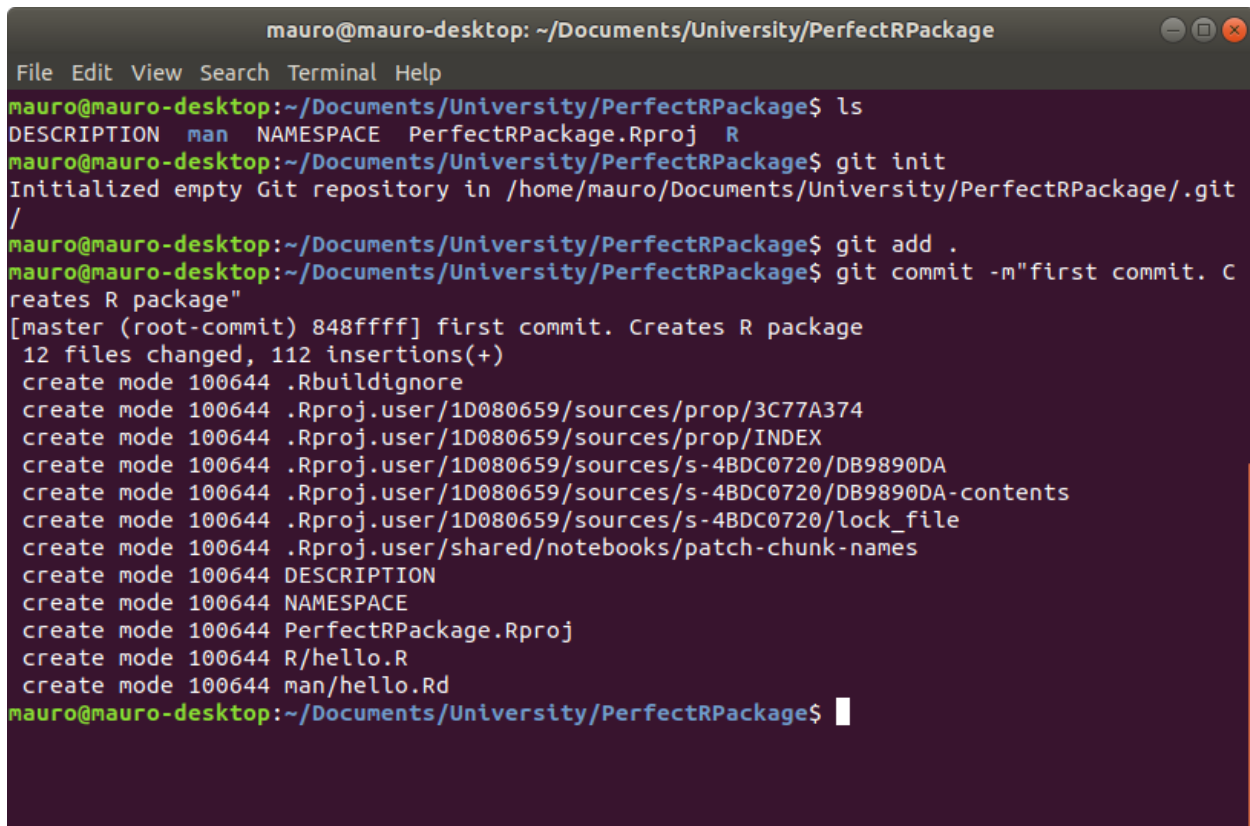
☐ Open in new session **Create Project** **Cancel**

After this, RStudio will open a new session that should look like the following one, with a `hello.R` file and with a lot of new content such as

- `.Rbuildignore`
- `.Rproj.user`
- `DESCRIPTION`
- `man`
- `NAMESPACE`
- `PerfectRPackage.Rproj`
- `R`





Now it's time to create a GitHub repository. Browse inside the package directory, in this case called **PerfectRPackage** on the terminal. Type `git init` to initialize a repository, then follow it with `git add .` to stage all the new files created by RStudio when it created the package. Then, commit these staged changes using `git commit -m"` and put some helpful message in between `"`. It should look something like this





Now we need to create an upstream repository on GitHub. To do that, go on www.github.com, log in and click on the green button **New** to create a new repository


Repositories





 [MauroCE/NeuralNetworksReadingGroup](#)


 [MauroCE/compass](#)

 [MauroCE/StatisticalMethods1](#)

 [MauroCE/StatisticalComputingPortfolio](#)

 [awllee/sc1-2019](#)

 [MauroCE/sc1-2019](#)

 [MauroCE/Dissertation](#)

[Show more](#)

you will be directed to the following page where you can specify the name of the GitHub repository and add other details. I suggest naming it with the same name of the RPackage, in this case **PerfectRPackage**, to add a useful description, keep it Public, and to not initialize it with a README.md file just yet. Finally, click on the green button at the bottom saying **Create Repository**

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner





Repository name *



Great repository names are short and memorable. Need inspiration? How about **shiny-waddle**?

Description (optional)


This is an example RPackage created for the Statistical Computing portfolio.

- ☒  **Public**
Anyone can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

- ☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▼

Add a license: **None** ▼ 

Create repository

Scroll to the bottom, and copy the following commands

...or push an existing repository from the command line

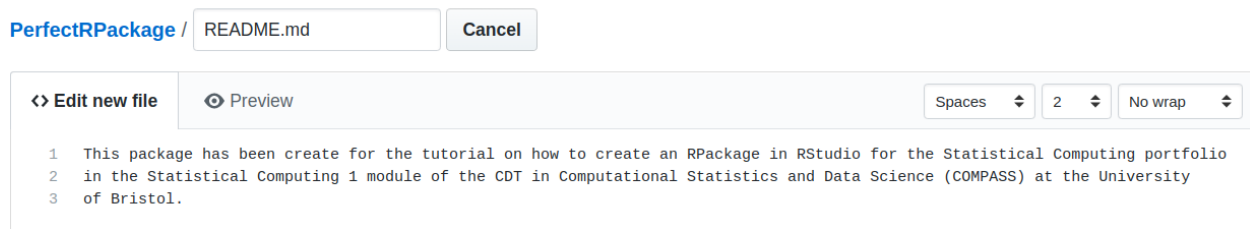
```
git remote add origin https://github.com/MauroCE/PerfectRPackage.git
git push -u origin master
```



Now go back to your terminal, paste the commands and this will connect the repository upstream.

```
mauro@mauro-desktop: ~/Documents/University/PerfectRPackage
File Edit View Search Terminal Help
reates R package"
[master (root-commit) 848ffff] first commit. Creates R package
12 files changed, 112 insertions(+)
create mode 100644 .Rbuildignore
create mode 100644 .Rproj.user/1D080659/sources/prop/3C77A374
create mode 100644 .Rproj.user/1D080659/sources/prop/INDEX
create mode 100644 .Rproj.user/1D080659/sources/s-4BDC0720/DB9890DA
create mode 100644 .Rproj.user/1D080659/sources/s-4BDC0720/DB9890DA-contents
create mode 100644 .Rproj.user/1D080659/sources/s-4BDC0720/lock_file
create mode 100644 .Rproj.user/shared/notebooks/patch-chunk-names
create mode 100644 DESCRIPTION
create mode 100644 NAMESPACE
create mode 100644 PerfectRPackage.Rproj
create mode 100644 R/hello.R
create mode 100644 man/hello.Rd
mauro@mauro-desktop:~/Documents/University/PerfectRPackage$ git remote add origin https://
/github.com/MauroCE/PerfectRPackage.git
mauro@mauro-desktop:~/Documents/University/PerfectRPackage$ git push -u origin master
Counting objects: 21, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (13/13), done.
Writing objects: 100% (21/21), 2.35 KiB | 2.35 MiB/s, done.
Total 21 (delta 0), reused 0 (delta 0)
To https://github.com/MauroCE/PerfectRPackage.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
mauro@mauro-desktop:~/Documents/University/PerfectRPackage$
```

Next, we want to add a README.md file. This will be the “welcome page” of our GitHub repository. To do so, simply go on the PerfectRPackage repository on GitHub and click on **Create New File** button on the right of the screen. This will load the following page. Call your file README.md and write some useful description.



Next, scroll to the bottom of the page, add a comment to describe what you’re doing and commit to the master branch, as show in the following screenshot.



Commit new file

Create README.md

Add an optional extended description...



☒ Commit directly to the `master` branch.

☐ Create a **new branch** for this commit and start a pull request. [Learn more about pull requests.](#)

Commit new file

Cancel

Click **Commit new file**. Next, we need to get these changes on our local version of the repository, so open up the terminal in the package directory, which is `PerfectRPackage` in this case. Type in `git status` to see what's going on in your local repository. If everything is up to date and you don't have any changes pending to be staged, simply type `git pull`. Otherwise, add your changes by typing `git add .` and then commit them with some useful message, using the command `git commit -m"` as shown below.

```
mauro@mauro-desktop: ~/Documents/University/PerfectRPackage
File Edit View Search Terminal Help
mauro@mauro-desktop:~/Documents/University/PerfectRPackage$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   .Rproj.user/1D080659/sources/prop/3C77A374
        modified:   .Rproj.user/1D080659/sources/s-4BDC0720/DB9890DA

no changes added to commit (use "git add" and/or "git commit -a")
mauro@mauro-desktop:~/Documents/University/PerfectRPackage$ git add .
mauro@mauro-desktop:~/Documents/University/PerfectRPackage$ git commit -m"committing new
changes to RPackage"
[master ad53dfb] committing new changes to RPackage
 2 files changed, 2 insertions(+), 2 deletions(-)
mauro@mauro-desktop:~/Documents/University/PerfectRPackage$
```

Next type

```
git fetch origin --prune
```

followed by

```
git rebase --preserve-merges origin/master
```


This will allow you to finally push your changes using `git push`.

```
mauro@mauro-desktop:~/Documents/University/PerfectRPackage$ git rebase --preserve-merges origin/master
Successfully rebased and updated refs/heads/master.
mauro@mauro-desktop:~/Documents/University/PerfectRPackage$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
mauro@mauro-desktop:~/Documents/University/PerfectRPackage$ git push
Counting objects: 9, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 764 bytes | 382.00 KiB/s, done.
Total 9 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/MauroCE/PerfectRPackage.git
  3425682..8886856 master -> master
mauro@mauro-desktop:~/Documents/University/PerfectRPackage$
```

Testing, Documentation, Licensing and README.md

First of all, we need to install some packages. Open RStudio, go to the console and type

```
install.packages(c("devtools", "roxygen2", "testthat", "knitr"))
```

When they're finished installing, simply load the `devtools` package by typing

```
library(devtools)
```

Now we want to add an author to the package. So type in something like this

```
mauro <- person(given="Mauro", family="Camara Escudero", email="maurocamaraescudero@gmail.com")
```

and assign it to the `author` variable by typing

```
devtools.desc.author = mauro
```

Now it's time to add a LICENCE to our Package. The easiest way to add an MIT license is to type

```
use_mit_license("Mauro Camara Escudero")
```

Now, we can add some documentation with Roxygen. To do this, simply open the `hello.R` script and click anywhere on the body of the `hello` function. Then, in the menu click

```
Code > Insert roxygen skeleton
```

To add a documentation skeleton. Remember to save the R script file. Now it's time to actually create the documentation. Since the file `NAMESPACE` was created by RStudio and not by Roxygen, we can delete such file for now. Then, go to the console and type

```
document()
```

This will re-create the `NAMESPACE` file and add a new folder called `man`, containing markdown documents with the documentation. The next thing to do is to install the package. Go to the console and type

```
install()
```

Next, we want to create some tests. Type

```
use_testthat()
```

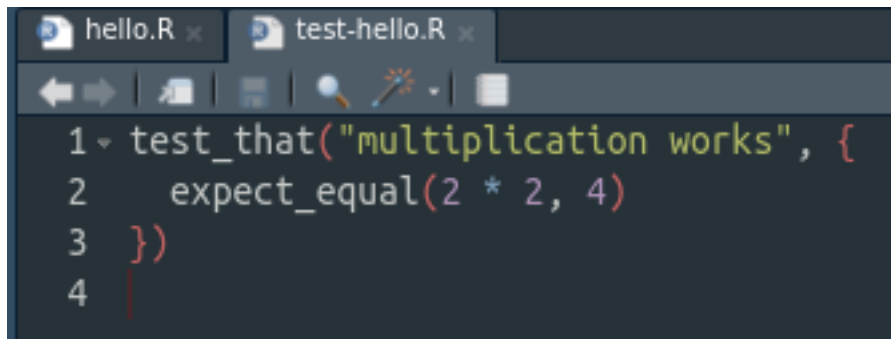
and you should get an output that looks like this

```
> use_testthat()
✓ Adding 'testthat' to Suggests field in DESCRIPTION
✓ Creating 'tests/testthat/'
✓ Writing 'tests/testthat.R'
● Call 'use_test()' to initialize a basic test file and open it for editing.
```

To create a new test file for the function `hello` go to the console and type

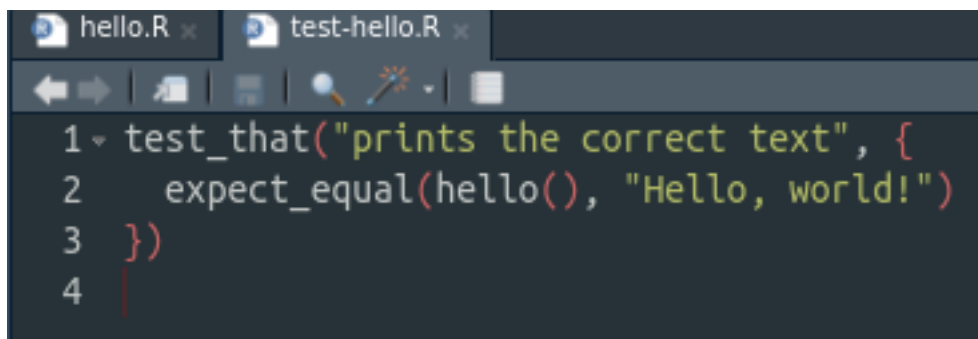
```
use_test("hello")
```

This will create a new file called `test-hello.R`, which will look like this



```
1 test_that("multiplication works", {
2   expect_equal(2 * 2, 4)
3 })
4
```

Modify this function to do a sensible test. For instance, give a sensible name and check that the function `hello` actually prints `Hello, World!`. It should look like this:



```
1 test_that("prints the correct text", {
2   expect_equal(hello(), "Hello, world!")
3 })
4
```

Now we can actually test our function. To do so, go to the console, load the `testthat` package by typing

```
library(testthat)
```

Then load the `PerfectRPackage` in the console by typing

```
load_all()
```

and test it using

```
test()
```

Hopefully, the test will be fine and you should obtain a summary of the testing that looks like this:

```
> test()
Loading PerfectRPackage
Testing PerfectRPackage
✓ | OK F W S | Context
: | 0      | hello[1] "Hello, world!"
✓ | 1      | hello

== Results ==
OK:      1
Failed:  0
Warnings: 0
Skipped: 0
```

At this point you could push all the new changes to the upstream repository. Notice that if you did not create a `README.md` file, you can create it now together with its RMarkdown version by typing the following in the console

```
use_readme_rmd()
```

If you do this, remember to re-knit the `README` document before committing the changes.

Travis for Continuous Integration

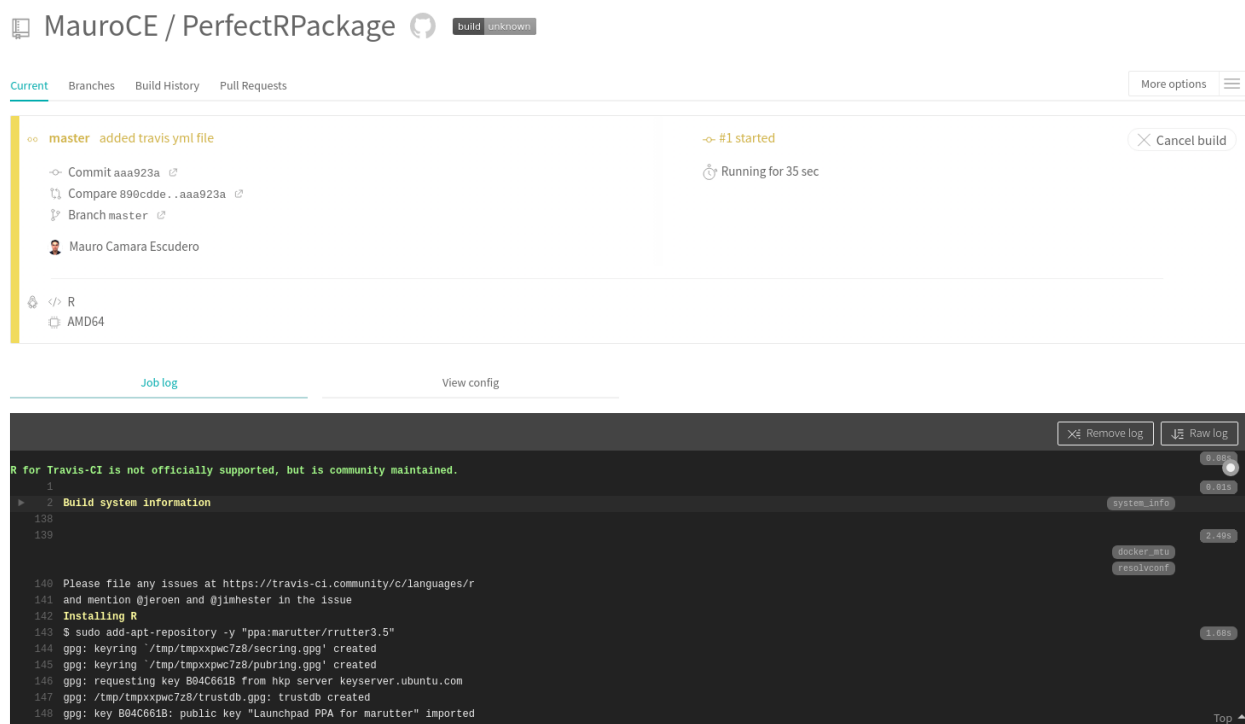
To do continuous deployment, go on <https://travis-ci.com/> and login with your GitHub account, granting access to the repositories for which you want continuous development. Next, create a new text file called `.travis.yml` and paste the following content

```
language: r
dist: xenial
cache: packages
branches:
  only:
    - master

r_github_packages:
  - r-lib/covr

after_success:
  - Rscript -e 'covr::codecov()'
```

commit the changes. From the **next** commit, travis will start building and testing your package. Go on the travis website and you should be able to see a screen looking like this



MauroCE / PerfectRPackage build unknown

Current Branches Build History Pull Requests More options

master added travis.yml file

Commit aaa923a

Compare 890cde...aaa923a

Branch master

Mauro Camara Escudero

R

AMD64

#1 started

Running for 35 sec

Cancel build

Job log View config

R for Travis-CI is not officially supported, but is community maintained.

1

2 Build system information

138

139

140 Please file any issues at https://travis-ci.community/c/languages/r

141 and mention @jeroen and @jimhester in the issue

142 Installing R

143 \$ sudo add-apt-repository -y "ppa:marutter/rrutter3.5"

144 gpg: keyring '/tmp/tmpxxpw7z8/secring.gpg' created

145 gpg: keyring '/tmp/tmpxxpw7z8/pubring.gpg' created

146 gpg: requesting key B04C6618 from hkp server keyserver.ubuntu.com

147 gpg: /tmp/tmpxxpw7z8/trustdb.gpg: trustdb created

148 gpg: key B04C6618: public key "Launchpad PPA for marutter" imported

149 Total number of packages: 1

Remove log Raw log

0.00s

0.01s

2.49s

1.68s

Top

Adding travis badge to GitHub readme

If you want to add a badge to your README.md file, simply go on the travis website shown above and click on the badge saying **build unknown**. Select **Markdown**



Status Image

BRANCH

master

FORMAT

Markdown

RESULT

[!Build Status](https://travis-ci.com/MauroCE/PerfectRPackage.svg?branch=master)
(https://travis-ci.com/MauroCE/PerfectRPackage)

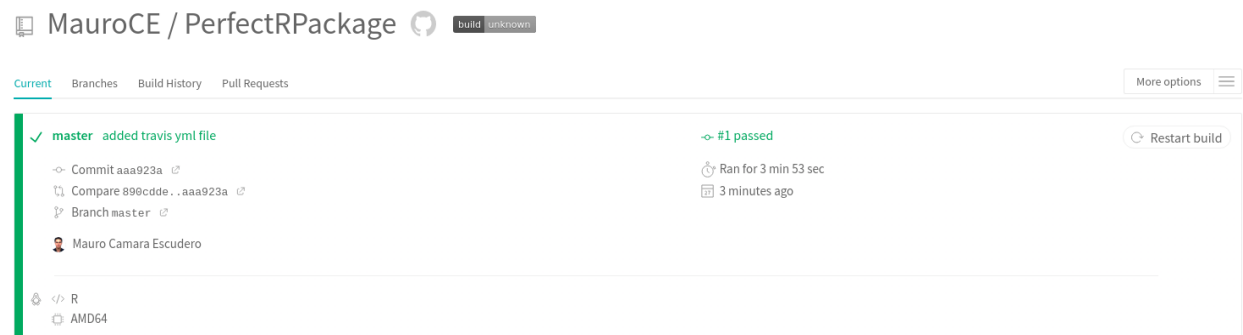
Copy the text in the last cell, go into your `README.rmd` file and type it under the comments saying where to add the badge, like shown in the picture below

```
# PerfectRPackage

<!-- badges: start -->
<!-- badges: end -->

[![Build Status](https://travis-ci.com/MauroCE/PerfectRPackage.svg?branch=master)](https://travis-ci.com/MauroCE/PerfectRPackage)
```

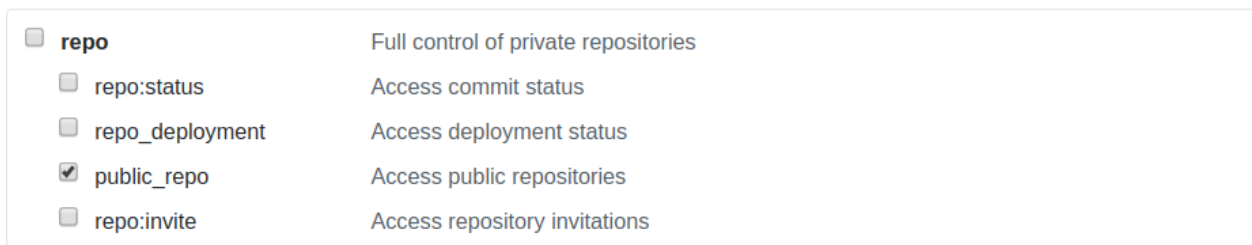
Save the RMarkdown file. If everything went well, in the travis website you should now see a successful build.



Finally, knit the `README.rmd` and then push your changes to the upstream directory, and you should then be able to see the badge on your `README.md` file on GitHub.

Hosting a Website with Travis on GitHub pages

Go to GitHub and click on your profile image on the top right corner to toggle the menu. Click on **Settings** and then click to the bottom option **Developer Settings**. Here, click on **Personal access tokens** and click on **Generate new token**. Make sure that `public_repo` is selected.




Copy the token and go to the travis website. In the successful build, click on **More options** on the right of the screen and choose **Settings**.

Scroll down to **Environment Variables**. Here, choose a name that will be used as a variable holding your personal access token. For instance `GITHUB_PAT` and paste your Personal Access Token from github in the **value** box. Then click **Add**. In the following screenshot I show you where you should be adding this information, without actually adding it for security reasons

Environment Variables

Customize your build using environment variables. For secure tips on generating private keys [read our documentation](#)

 If your secret variable has special characters like `&`, escape them by adding `\` in front of each special character. For example, `ma&w!doc` would be entered as `ma&w\!doc`.

NAME	VALUE	BRANCH	
<input type="text" value="Name"/>	<input type="text" value="Value"/>	<input type="text" value="All branches"/>	<div><input type="checkbox"/> <small>DISPLAY VALUE IN BUILD LOG</small></div> <div><input type="button" value="Add"/></div>

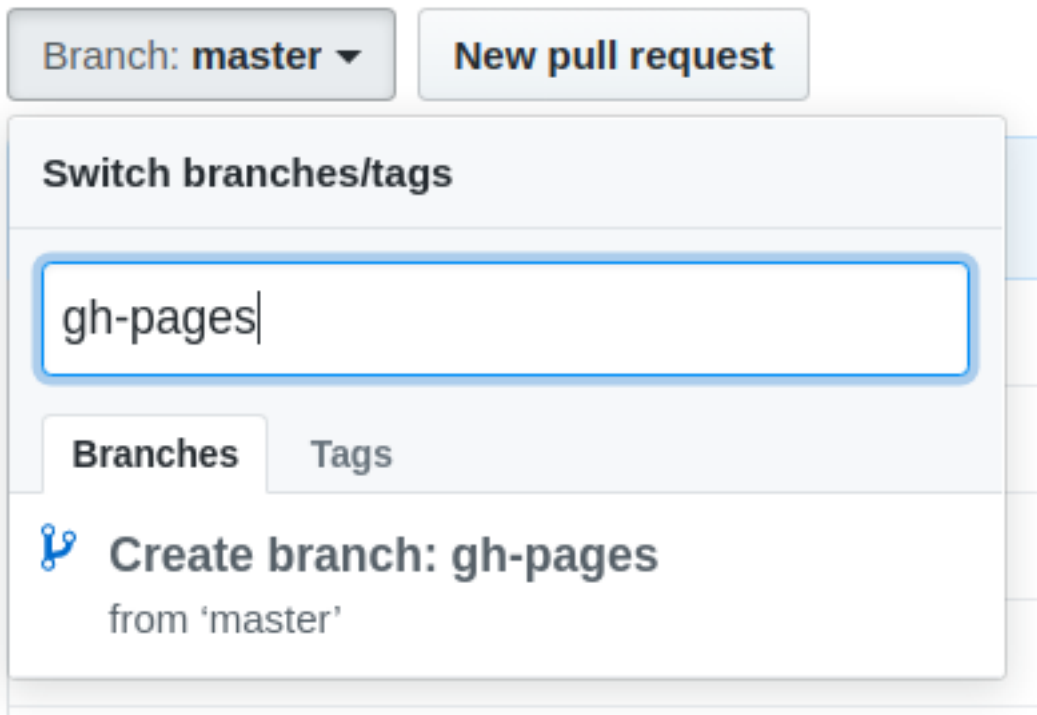
Now go into the `.travis.yml` file and change the content to

```
language: r
dist: xenial
cache: packages
branches:
  only:
    - master
r_packages:
  - rmarkdown

script:
  - Rscript -e 'rmarkdown::render("website.Rmd", output_dir="public")'

deploy:
  provider: pages
  skip_cleanup: true
  github_token: $GITHUB_PAT
  on:
    branch: master
  local_dir: public
  target_branch: gh-pages
```

Make sure that the setting `github_token` is set to the same name that you've given to the github token on travis, in this case `GITHUB_PAT`. Notice how we've set `target_branch: gh-pages`. Go on GitHub and create a new branch called `gh-pages`. To do so, click `Branch:master` type `gh-pages` and click at the bottom where it says `Create branch: gh-pages`.



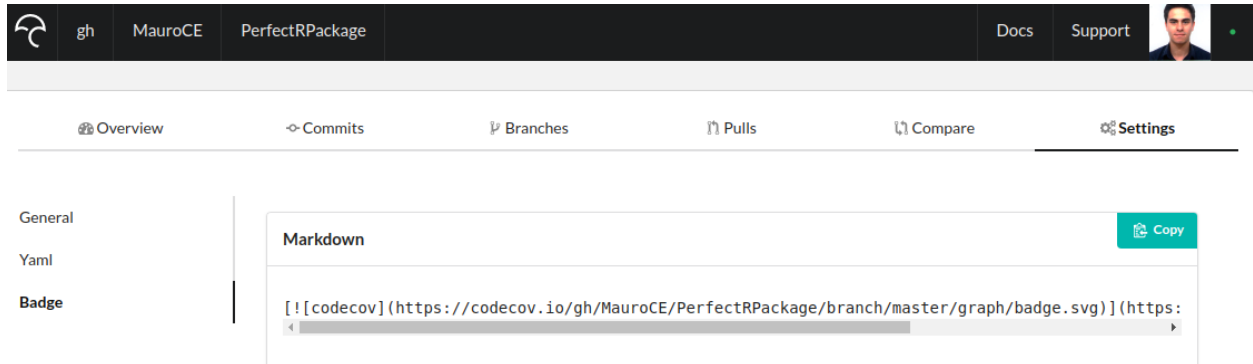
Finally, notice how we've told travis to build the website for `website.Rmd` RMarkdown file. This doesn't exist yet, so we'll create it now. Go to RStudio and create such file in the main folder `PerfectRPackage`.

```
hello.R x README.Rmd x test-hello.R x .travis.yml x website.Rmd x
1 ---
2 title: "website"
3 author: "Mauro Camara Escudero"
4 date: "11/2/2019"
5 output: html_document
6 ---
7
8 ```{r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 ## R Markdown
13
14 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents.
15 For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
16
17 When you click the Knit button a document will be generated that includes both content as well as the output of
18 any embedded R code chunks within the document. You can embed an R code chunk like this:
19
20 ```{r cars}
21 summary(cars)
22 ```
23
24 ## Including Plots
25
26 You can also embed plots, for example:
```

Finally, add, commit and push the changes so that travis can build the website. When travis build will be successful, your website will be available at <https://mauroce.github.io/PerfectRPackage/website.html>.

Adding Codecov Badge

To add a codecov badge, go to www.codecov.io and login with github. In the **PerfectRPackage** repository click on **Settings** and on **Badge** on the left hand side of the screen. Copy the code under Markdown, as shown below



and paste it under the travis build badge on the **README.Rmd** file. Knit it, add the changes, commit them and push them. If the build is successful, you should see a codecov badge on your **README.md** file on GitHub.

