

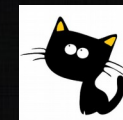
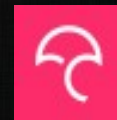
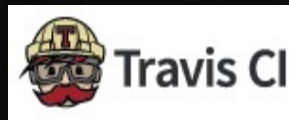
Professional R development

using git, GitHub, Travis CI, testthat, lintr, covr, Codecov and
goodpractice

© 2016 Richel Bilderbeek



<http://www.github.com/richelbilderbeek/PresentationsAboutR>



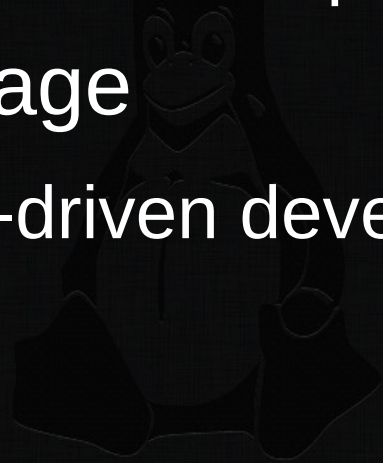
What? Why? Mastery?

- What: follow all good practices by default
- Why: proven to pay off
- Mastery: set up tools to follow all good practices by default



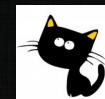
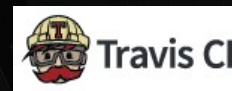
Not about ...

- How to create a package
 - See the talk 'How to develop a package'
- Extending a package
 - See the talk 'Test-driven development'



Setup

- Version control
- Code hosting
- Continuous integration
- Testing framework
- Coding standard
- Code coverage
- Static code analysis



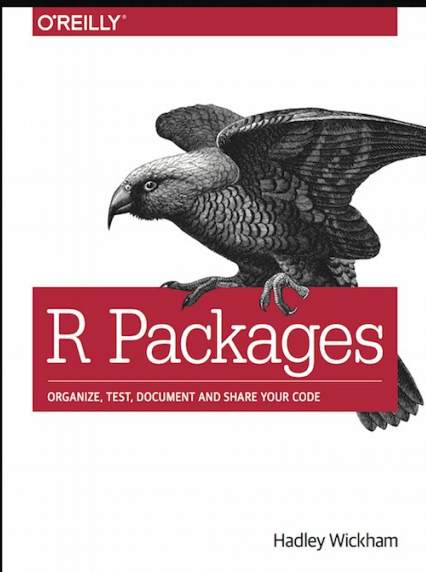
Version control

- Keeps a history of code



Rule 9:
Do source code
management

Keynote Speech at useR!, 2014



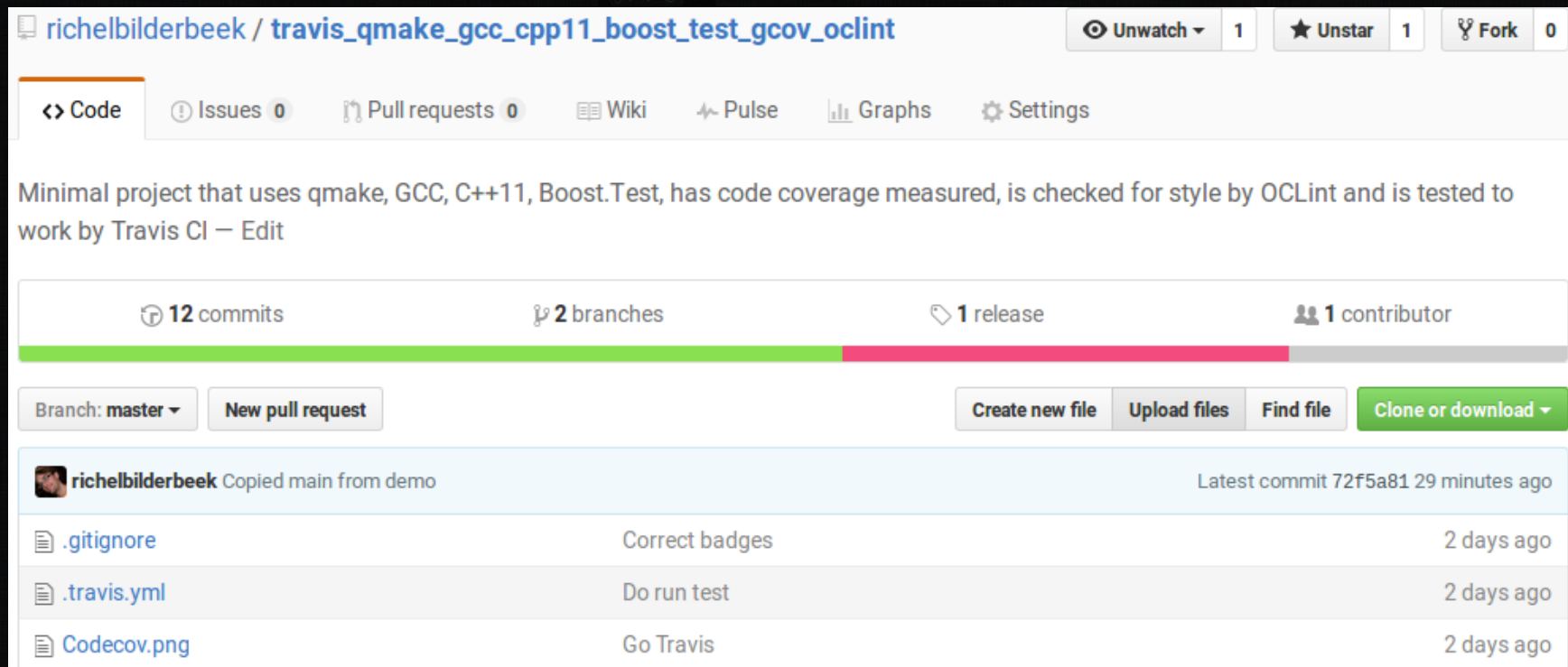
If you are serious
about software development,
you need to learn about git

Chapter 13: git and GitHub



Code hosting

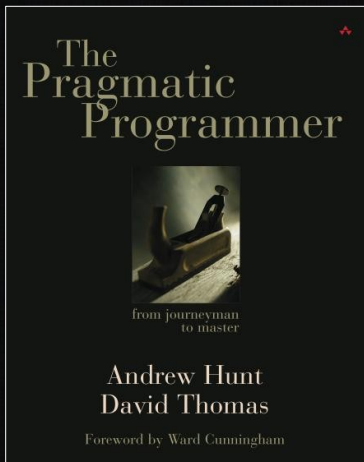
- Host your (version controlled) code
- Using GitHub is good practice [1]



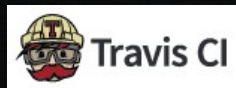
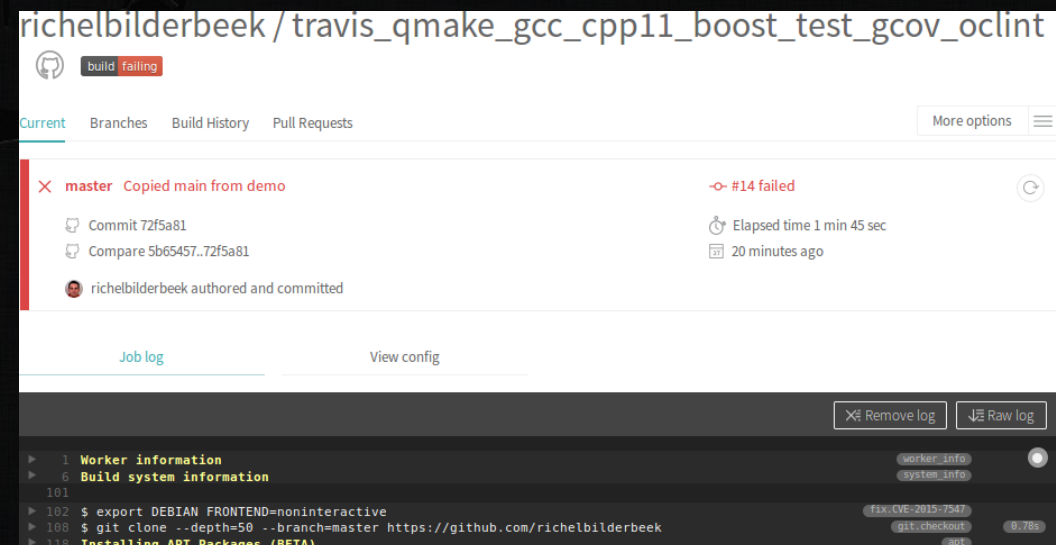
[1] Perez-Riverol, Yasset, et al. "Ten Simple Rules for Taking Advantage of git and GitHub." bioRxiv (2016): 048744.

Continuous integration

- Run scripts upon pushing new code

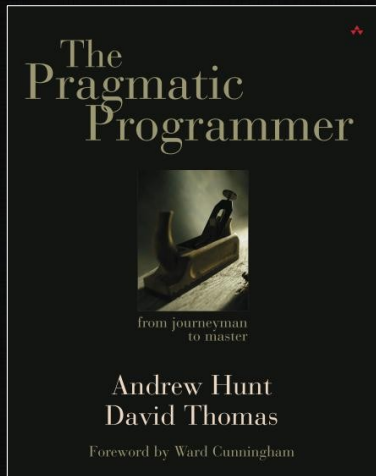


Tip 61: Don't Use Manual Procedures



Testing framework

- To test your code



Tip 62: Test Early.
Test Often. Test Automatically

Testing is a vital part of
package development

Fewer bugs
Better code structure
Easier restarts
Robust code



Chapter 7: Testing



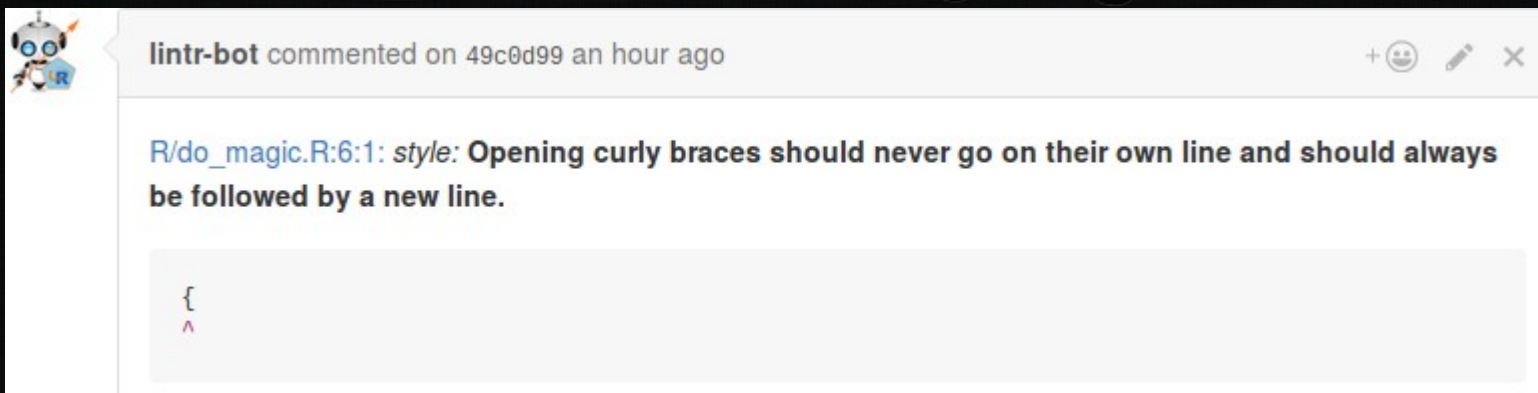
Coding standard

- Use a coding standard [1]



I strongly recommend
that you use a
consistent style

Chapter 3, paragraph 'Code Style'

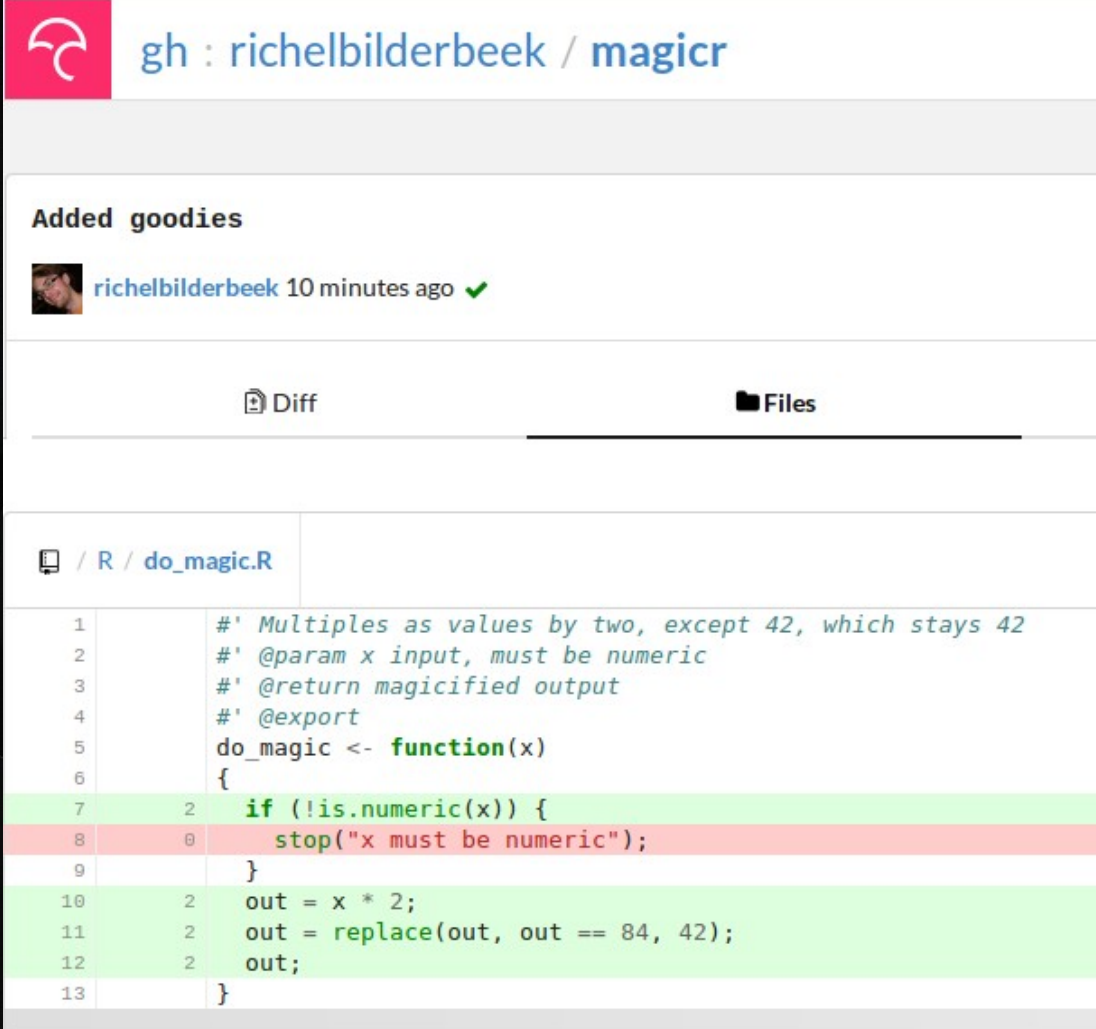


Johnson, Paul E. "R Style. An Rchaeological Commentary." (2016).



Code coverage

- Measures code that is actually used
- Correlates with quality [1]



The screenshot shows a GitHub repository page for 'richelbilderbeek / magicr'. It displays a commit titled 'Added goodies' by 'richelbilderbeek' from 10 minutes ago. Below the commit, there are tabs for 'Diff' and 'Files'. The 'Diff' tab is selected, showing a diff for the file 'R / do_magic.R'. The diff shows changes to the 'do_magic' function, with line numbers 1 through 13. The code is as follows:

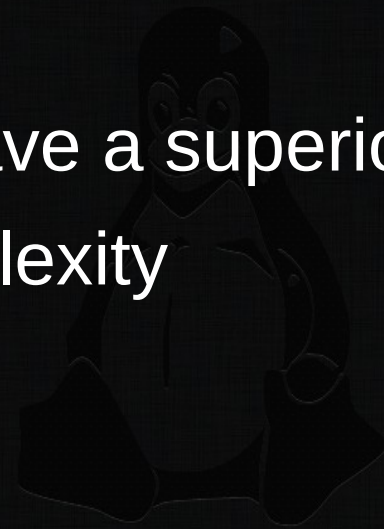
```
1      #' Multiplies as values by two, except 42, which stays 42
2      #' @param x input, must be numeric
3      #' @return magicified output
4      #' @export
5      do_magic <- function(x)
6      {
7      2    if (!is.numeric(x)) {
8      0    stop("x must be numeric");
9      }
10     2    out = x * 2;
11     2    out = replace(out, out == 84, 42);
12     2    out;
13     }
```

[1] Del Frate, Fabio, et al. "On the correlation between code coverage and software reliability." Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on. IEEE, 1995.



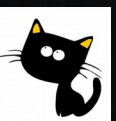
Test for good practices

- Checks code beyond the interpreter and lintr
 - Coding style
 - Functions that have a superior alternative
 - Cyclomatic complexity



It is good practice to

* omit trailing semicolons from code lines. They are not needed and most R coding standards forbid them



This presentation

- Introduce the package 'magick'
- (Put it on GitHub)
- Activate Travis CI
- Activate other tools



do_magic.R

```
#' Multiples as values by two,
#'   except 42, which stays 42
#' @param x input, must be numeric
#' @return magicified output
#' @export
do_magic <- function(x)
{
  if (!is.numeric(x)) {
    stop("x must be numeric");
  }
  out = x * 2;
  out = replace(out, out == 84, 42);
  out;
}
```

test-do_magic.R

```
context("do_magic")

test_that("do_magic: use", {
  expect_equal(do_magic(42), 42)
  expect_equal(do_magic(1), 2)
})
```


Output

```
> devtools::check()  
Updating prde documentation  
Loading prde  
[...]  
* checking tests ...  
  Running 'testthat.R'  
OK  
* DONE  
  
Status: OK  
  
R CMD check results  
0 errors | 0 warnings | 0 notes
```



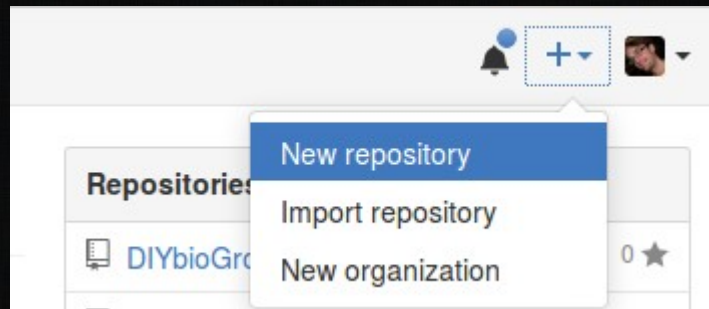
Next steps

- Create a GitHub repository
- Clone it to the local computer
- Put your code in that folder
- Upload the code



Create a GitHub repository


- Create a new repository



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner **Repository name**

 **richelbilderbeek** / ✓

Great repository names are short and memorable. Need inspiration? How about **fluffy-broccoli**.

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **R** | Add a license: **GNU General Public License v3.0** ⓘ

Create repository

Clone it to the local computer

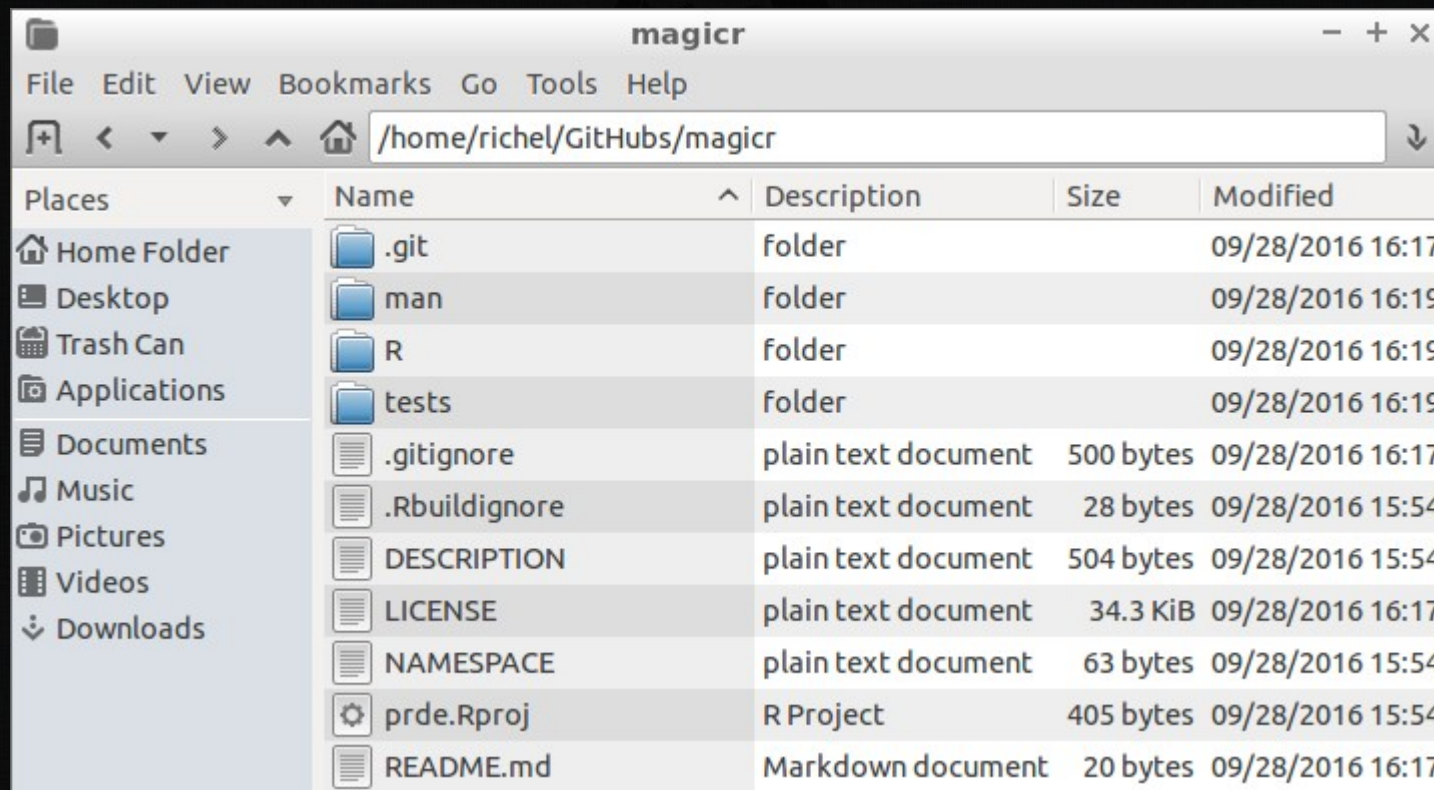
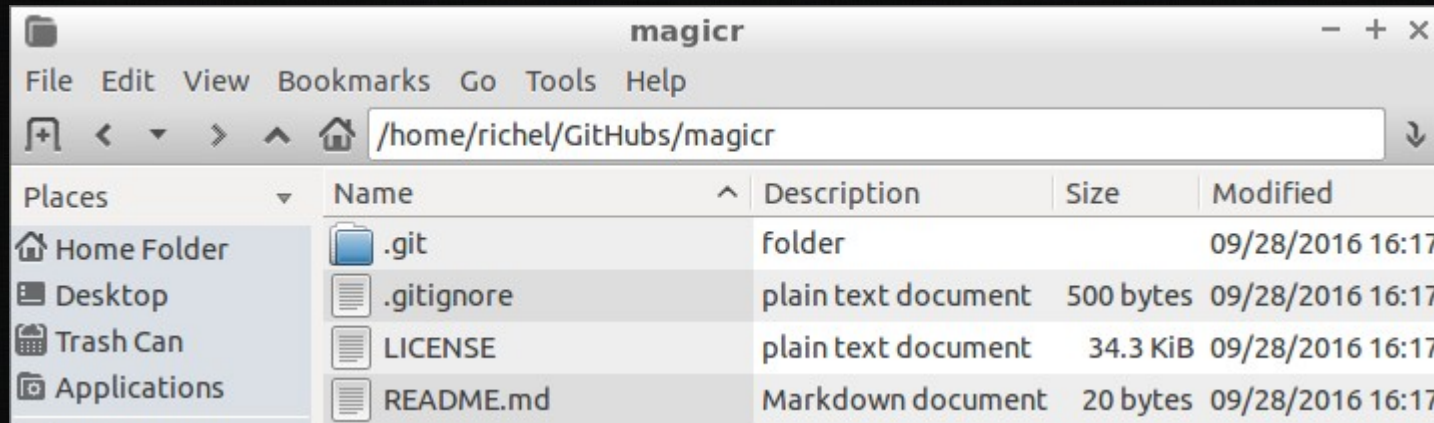
- `git clone https://github.com/[username]/[repository name]`

A screenshot of a terminal window with a title bar that reads 'richel@arduinorood: ~'. The terminal shows the execution of the command 'git clone https://github.com/richelbilderbeek/magicro'. The output indicates the repository is being cloned into a directory named 'magicro', showing progress for counting and compressing objects, and finally checking connectivity.

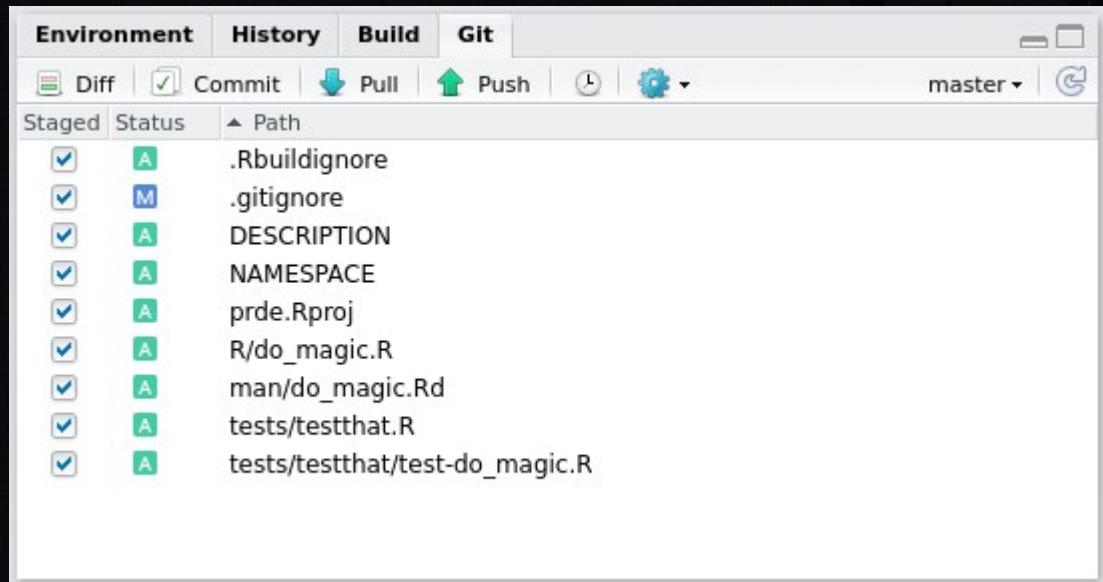
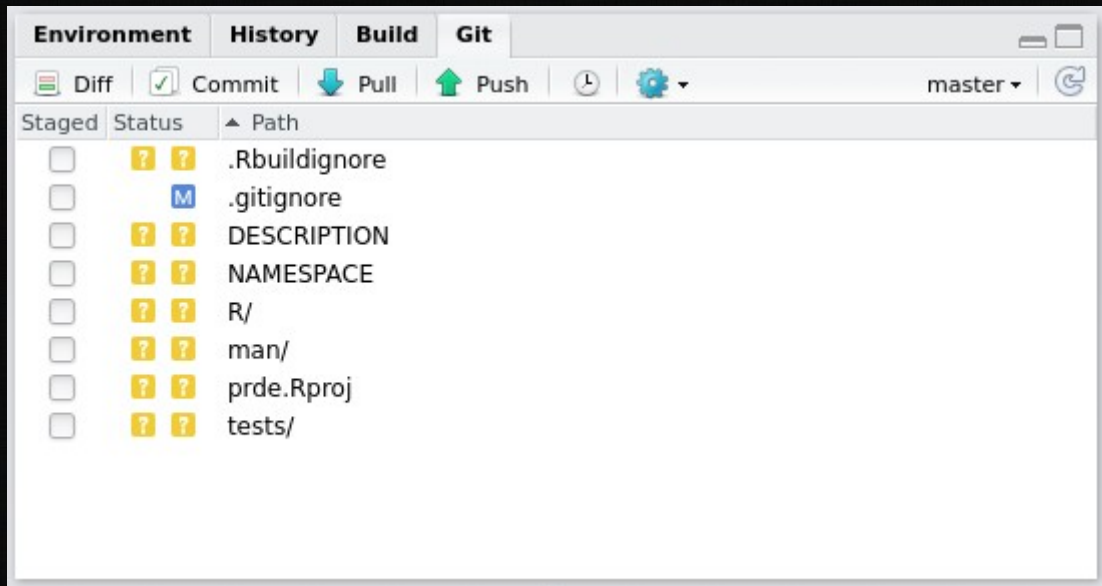
```
richel@arduinorood:~$ git clone https://github.com/richelbilderbeek/magicro
Cloning into 'magicro'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
Checking connectivity... done.
```

- (Wouldn't it be nice if this step could also be done from R studio?)

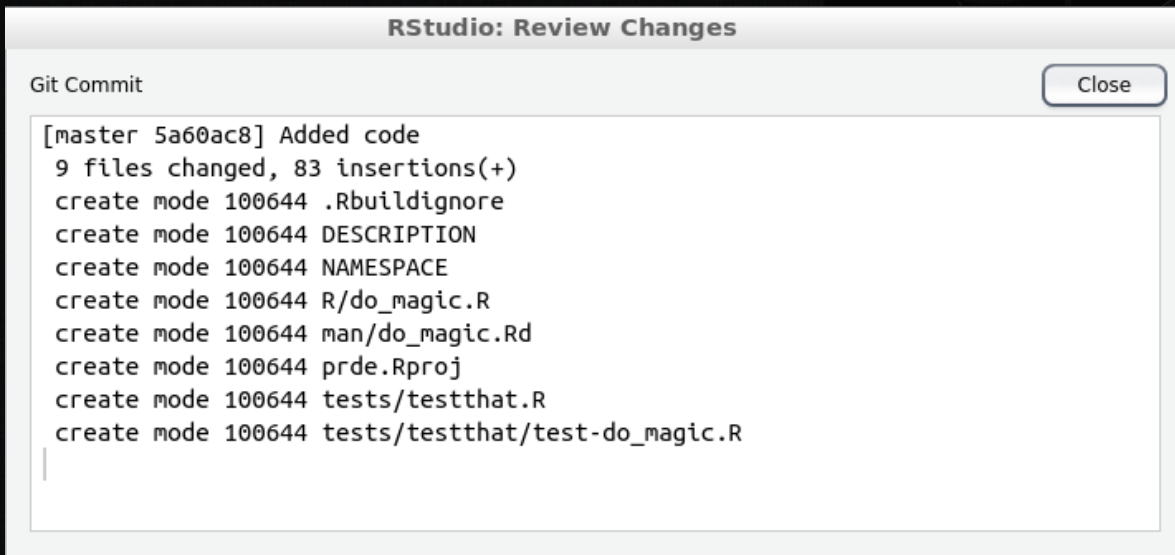
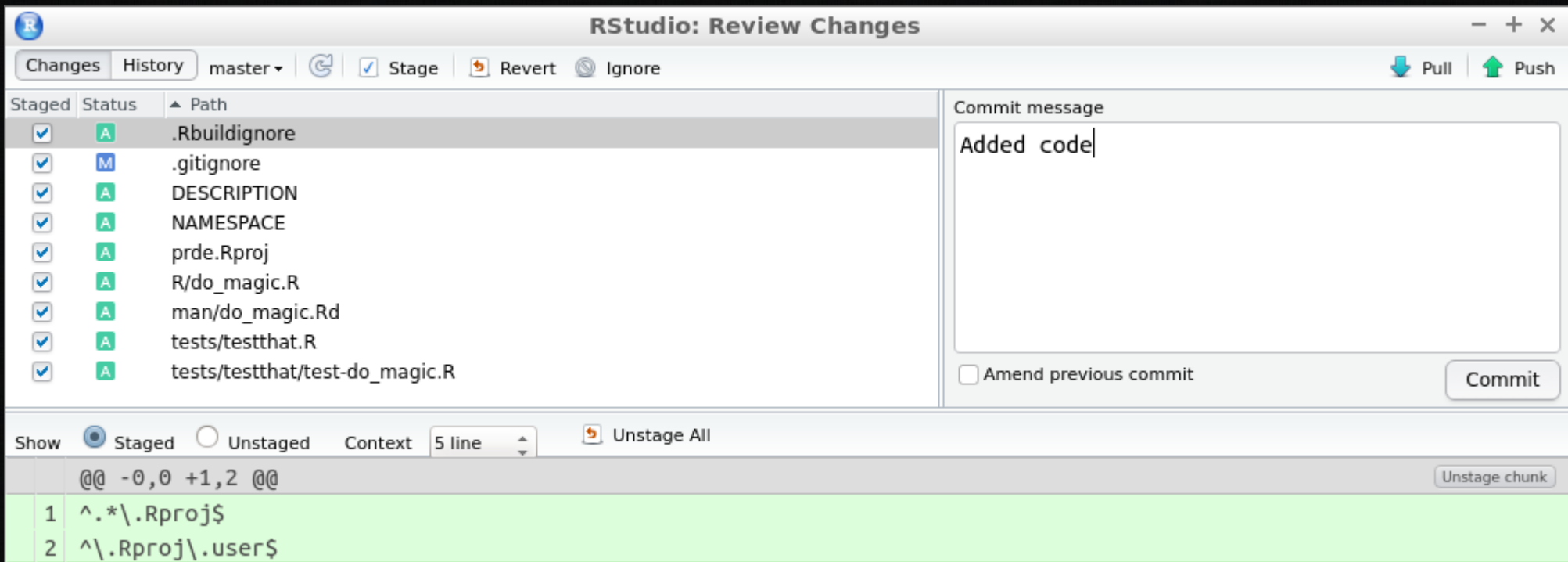
Put your code in that folder



Push it to GitHub

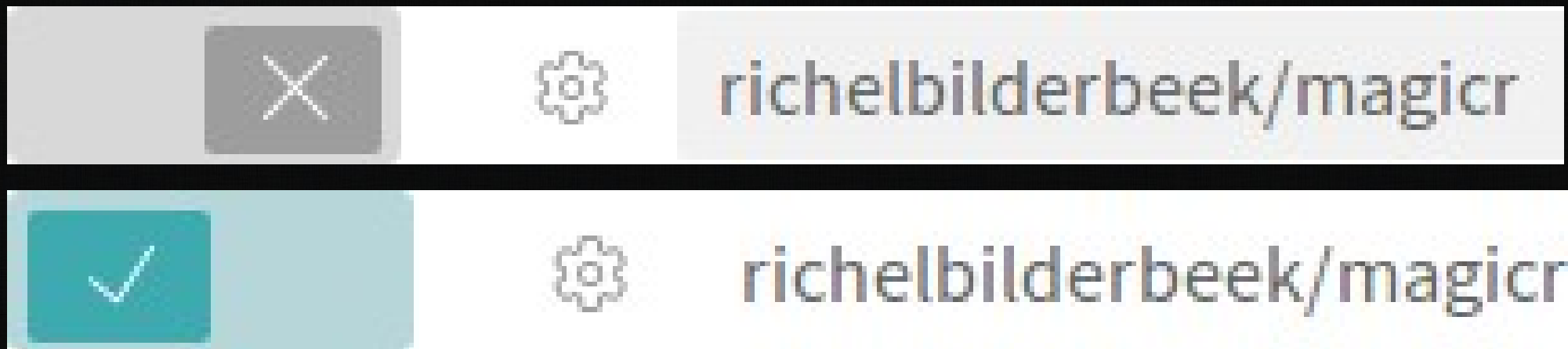
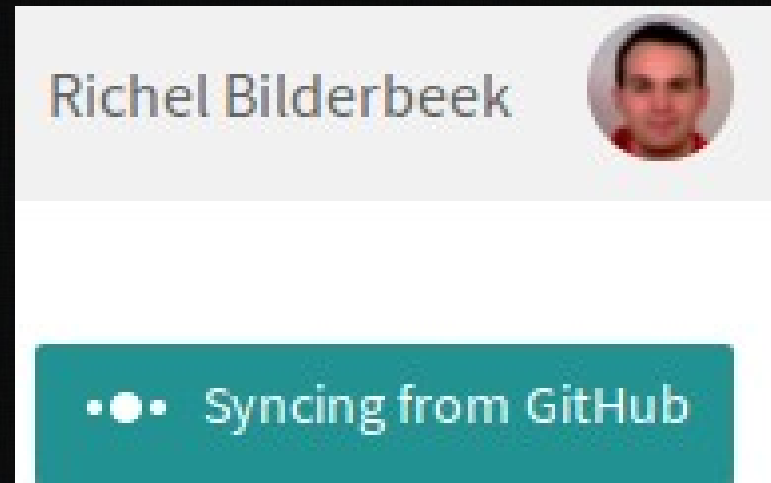


Push to GitHub



- Then push!

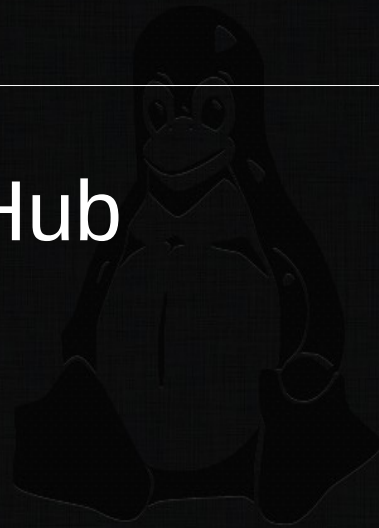
Activate Travis CI



Add file .travis.yml to root of package

```
language: r  
cache: packages
```

- Then push to GitHub
- Then wait ...



Success

richelbilderbeek / prde

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Professional R Development Example — Edit

3 commits 1 branch 0 releases 1 contributor GPL-3.0

Branch: master New pull request Create new file Upload files Find file Clone or download

richelbilderbeek Build badges 1 Latest commit 11052f1 5 hours ago

R	Go Travis	5 hours ago
man	Go Travis	5 hours ago
tests	Go Travis	5 hours ago
.Rbuildignore	Go Travis	5 hours ago
.gitignore	Initial commit	6 hours ago
.travis.yml	Go Travis	5 hours ago
Codecov.png	Build badges	5 hours ago
DESCRIPTION	Go Travis	5 hours ago
LICENSE	Initial commit	6 hours ago
NAMESPACE	Go Travis	5 hours ago
README.md	Build badges	5 hours ago
TravisCI.png	Build badges	5 hours ago

richelbilderbeek / magicr build passing

Current Branches Build History Pull Requests More options

✓ master Go Travis #1 passed Restart build

Commit 54bdc7c Elapsed time 5 min 42 sec
Compare 5a60ac8..54bdc7c about a minute ago

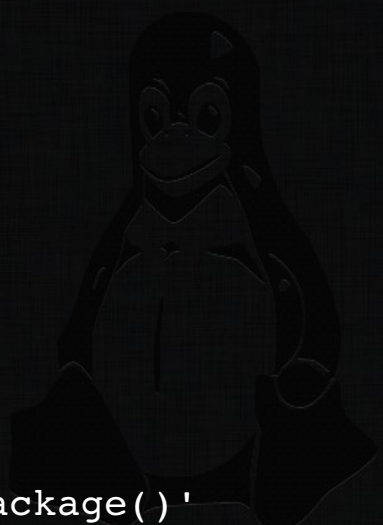
richelbilderbeek authored and committed

Activate other tools

```
language: r
cache: packages


r_github_packages:
  - jimhester/lintr
  - jimhester/covr
  - MangoTheCat/goodpractice


after_success:
  - Rscript -e 'lintr::lint_package()'
  - Rscript -e 'library(covr); codecov()'
  - Rscript -e 'library(goodpractice); gp()'
```




- Then push to GitHub
- Then wait ...

Results: code coverage

 gh : richelbilderbeek / magicr

Docs 

Added goodies

 richelbilderbeek 10 minutes ago ✓

master at 80.00%

49c0d99 master

Diff Files Builds Graphs

/ R / do_magic.R

5 4 0 1 80.00%

```
1  #' Multiplies as values by two, except 42, which stays 42
2  #' @param x input, must be numeric
3  #' @return magicified output
4  #' @export
5  do_magic <- function(x)
6  {
7    2  if (!is.numeric(x)) {
8      0  stop("x must be numeric");
9    }
10   2  out = x * 2;
11   2  out = replace(out, out == 84, 42);
12   2  out;
13 }
```


Results: lintr

```
▼ 1784 $ Rscript -e 'lintr::lint_package()' after success
1785 R/do_magic.R:6:1: style: Opening curly braces should never go on their own line and should always be followed by a new line.
1786 {
1787 ^
1788 R/do_magic.R:10:7: style: Use <-, not =, for assignment.
1789   out = x * 2;
1790   ^
1791 R/do_magic.R:11:7: style: Use <-, not =, for assignment.
1792   out = replace(out, out == 84, 42);
1793   ^
1794
```



lintr-bot commented on 49c0d99 an hour ago



R/do_magic.R:6:1: style: Opening curly braces should never go on their own line and should always be followed by a new line.

```
{
^
```

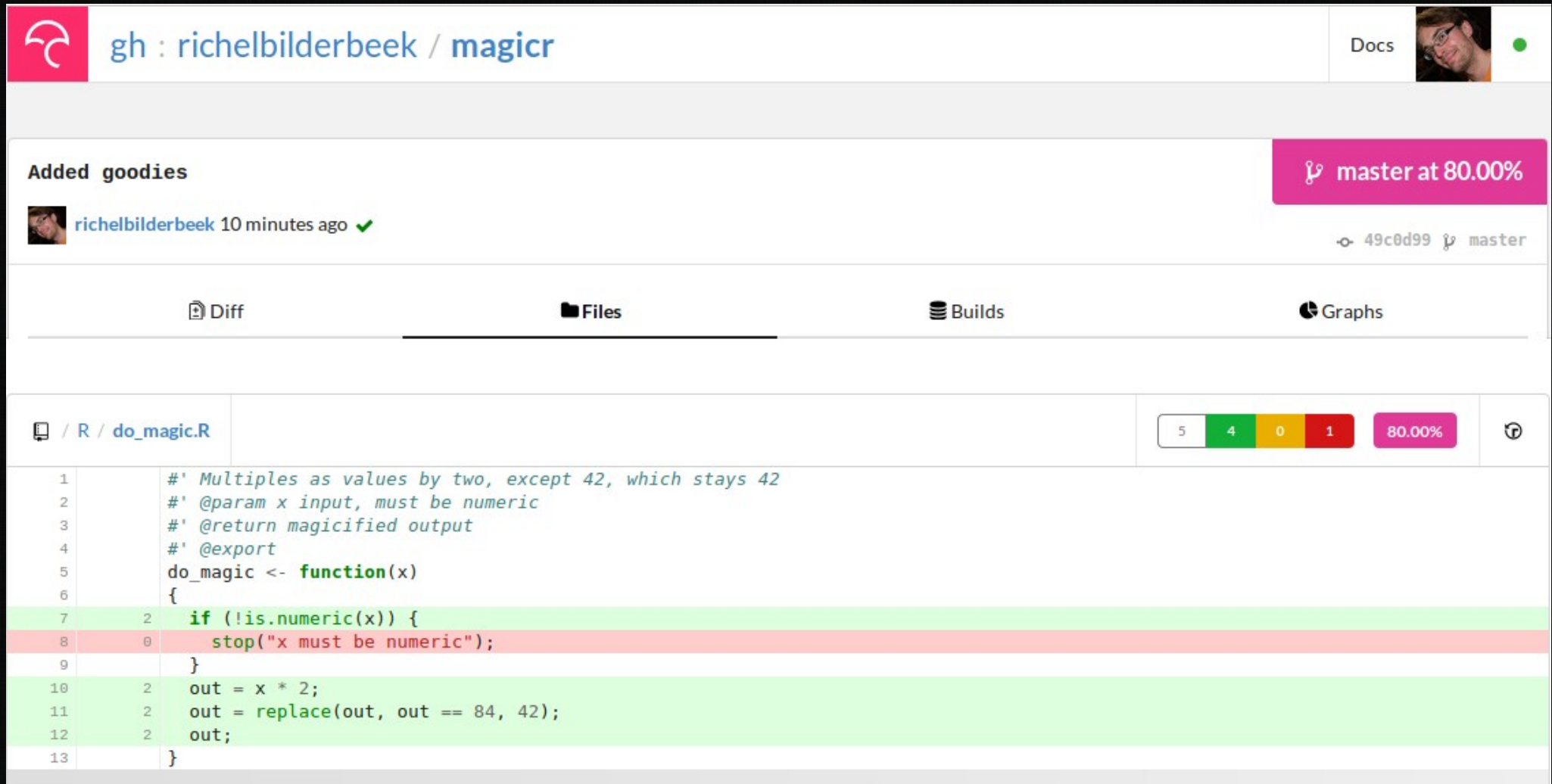
R/do_magic.R:10:7: style: Use <-, not =, for assignment.

```
out = x * 2;
    ^
```

R/do_magic.R:11:7: style: Use <-, not =, for assignment.

```
out = replace(out, out == 84, 42);
    ^
```

Results: code coverage





Results: goodpractice

It is good practice to

- * omit trailing semicolons from code lines. They are not needed and most R coding standards forbid them



My experiences

- Due to this setup, I have
 - Responded to bugs faster 
 - Found and removed dead code 
 - Learned a professional R coding standard
 - Reduced complexity of working code
 - Learned new things
 - Allow collaborators to do the same

Conclusion

- It is doable to follow all good practices in three Good Steps:
 - Follow a package structure
 - Put that package on GitHub
 - Add Travis CI and other tools
- I think this setup is useful for
 - R newbies
 - R pros