Continuous Integration services – Automate your unit tests

More complicated test scenarios require more advanced approaches: Stubs and Mocks

Some functions cannot be executed for testing purposes, e.g.

- Functions that access different systems, e.g., online authentication
- Persistent manipulations of databases
- Hardware controlling functions, e.g., a robot arm
- Execution of financial transactions, etc.
- Functions' dependency of non-existent code

The solution? Stubs and Mocks, see

https://stackoverflow.com/questions/3459287/whats-the-difference-between-a-mock-stub

Stubs and Mocks

Stubs

- The underlying operation is replaced by a stub for testing
- Stubs can perform primitive operations but usually return only a value
- Example: Test depends on a method that takes 5 minutes to complete, you can replace real implementation with a stub that returns hard-coded values, taking up much less time

Mocks

- In OOP, replacements for full objects are called mocks
- Mocks additionally check if methods were called as expected
- Example: You are testing a user creation class, so you want to check after calling `createUser()`, that `sendConfEmail()` is called

The more tests, the better… Check your code coverage (1/2)

- Code coverage shows which lines of code are tested
- Helps identify non-tested code regions
- Usually measures coverage as ratio, e.g., 60% of all lines, functions, etc.

  Warning: a high coverage does not guarantee thorough testing
- As a recommendation, focus especially on the boundaries (also called edge cases) of parameter ranges (0, `NA`, `Inf`, etc.) to identify unhandled problems

R package covr

- Supports coverage only when testing full packages

  Workaround is to create a dummy package

The more tests, the better… Check your code coverage (2/2)

With covr loaded, you can run `report()` to get a report of your code coverage.

Files | Source

Coverage	File	Lines	Relevant	Covered	Missed	Hits / Line
100.00	R/hello.R	18	1	1	0	2

For more information, visit https://covr.r-lib.org/

Execute all tests & calculate the code coverage automatically on every Push to GitHub

- In software engineering, continuous integration (CI) is the practice of regularly merging all changes to source code to a central repository, building and testing the source code after every change. This allows any bugs that are introduced to be identified quickly.
- Travis CI (travis-ci.org) is a continuous integration service for use with GitHub code repositories. For open-source projects it is free to use. Alternative CI software exists for use with GitHub, and with alternative version control systems and hosting services (for example, Jenkins).

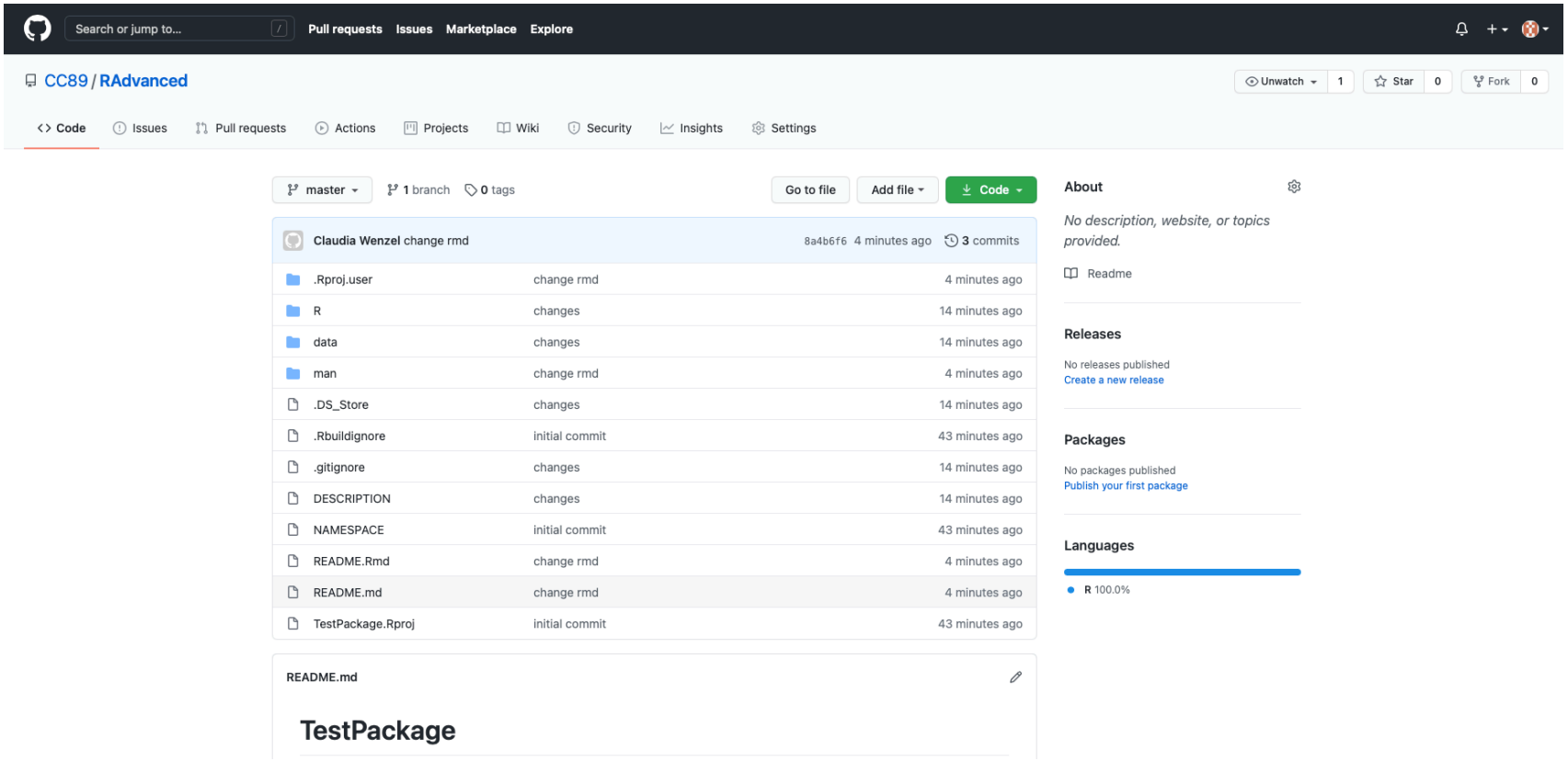
https://warwick.ac.uk/fac/sci/wdsi/events/wrug/resources/unittesting.pdf

Steps to enable automatic test execution & code coverage calculation for a R package

1. Upload your package to GitHub.
2. Create an account at travis-ci.com.
3. Activate the GitHub repository containing your package.
4. Add a .travis.yml file to the root directory of your package – the use_travis function in the usethis package will do this. The use_covr function will help you to set up the automatic calculation of the code coverage.
5. Commit this new file to your Git repository.

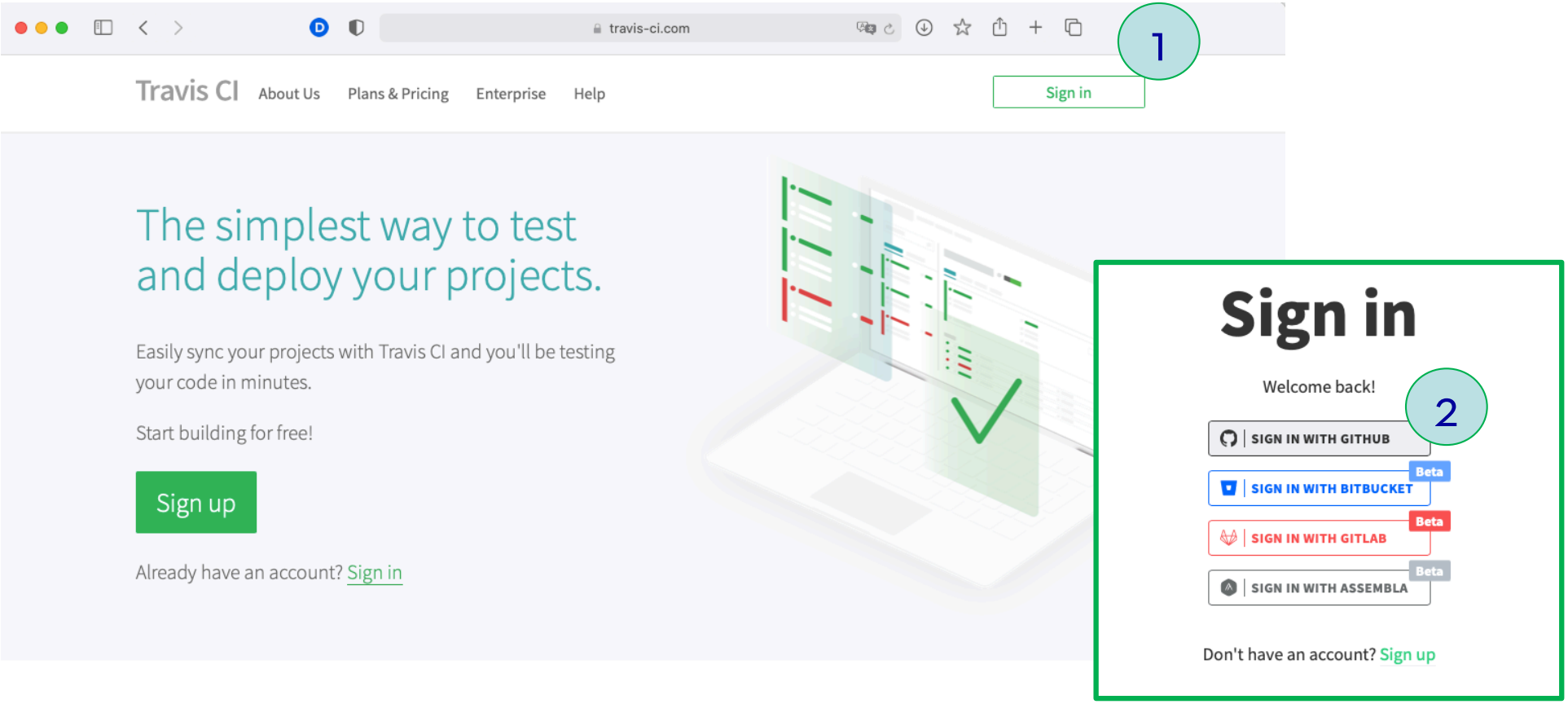
Now, whenever any commits are pushed to your online GitHub repository, Travis will automatically build and check your package. Travis will notify you if the check results in any errors, including those generated by failed unit tests.

Step 1: Upload package to Github



Step 2: Create account for Travis CI

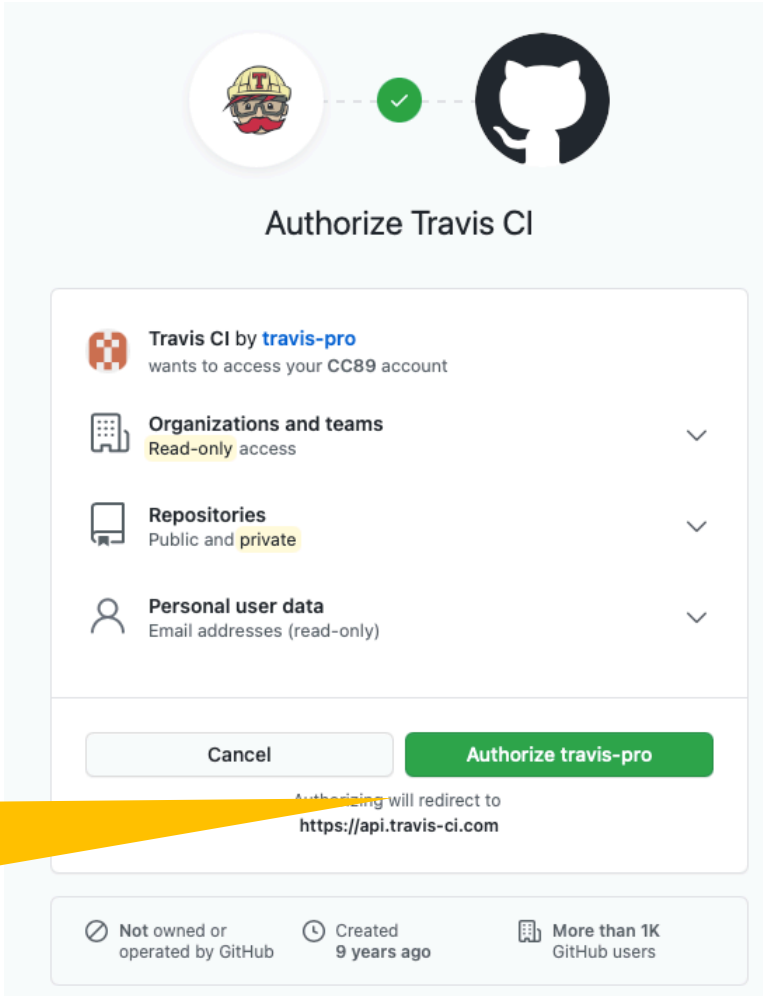
Visit: https://travis-ci.com/ and click "Sign in with GitHub".



https://github.com/dwyl/learn-travis

Step 2: Create account for Travis CI

You will be redirected to GitHub where you need to click "Authorize travis-pro"

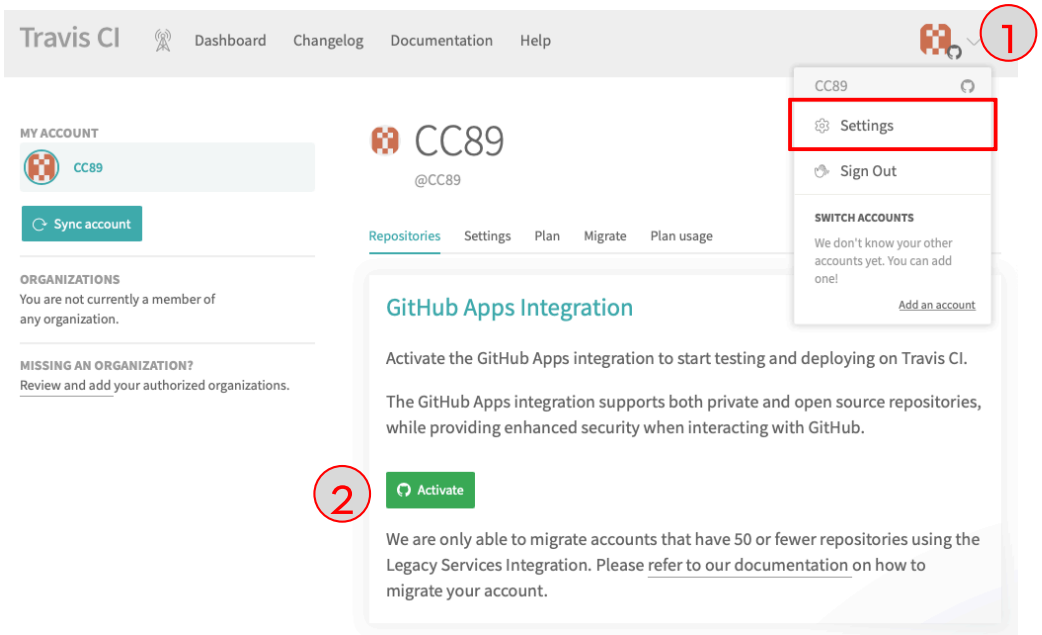


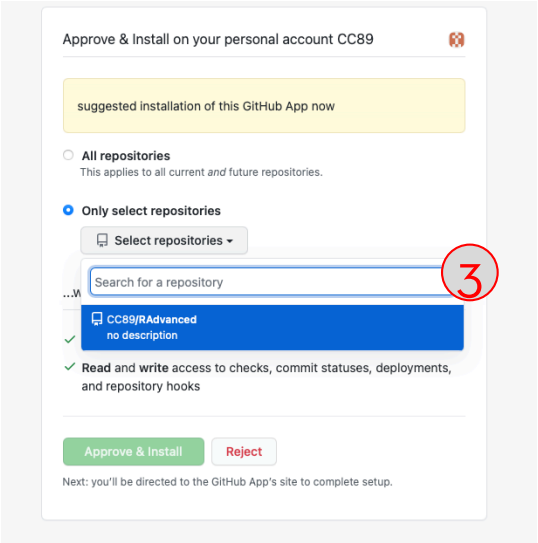
Note: If you ever want to stop Travis from accessing your GitHub account, simply visit: https://github.com/settings/applications and click on Revoke.

https://github.com/dwyl/learn-travis

Step 3: Activate Travis CI for the GitHub repository with the R package

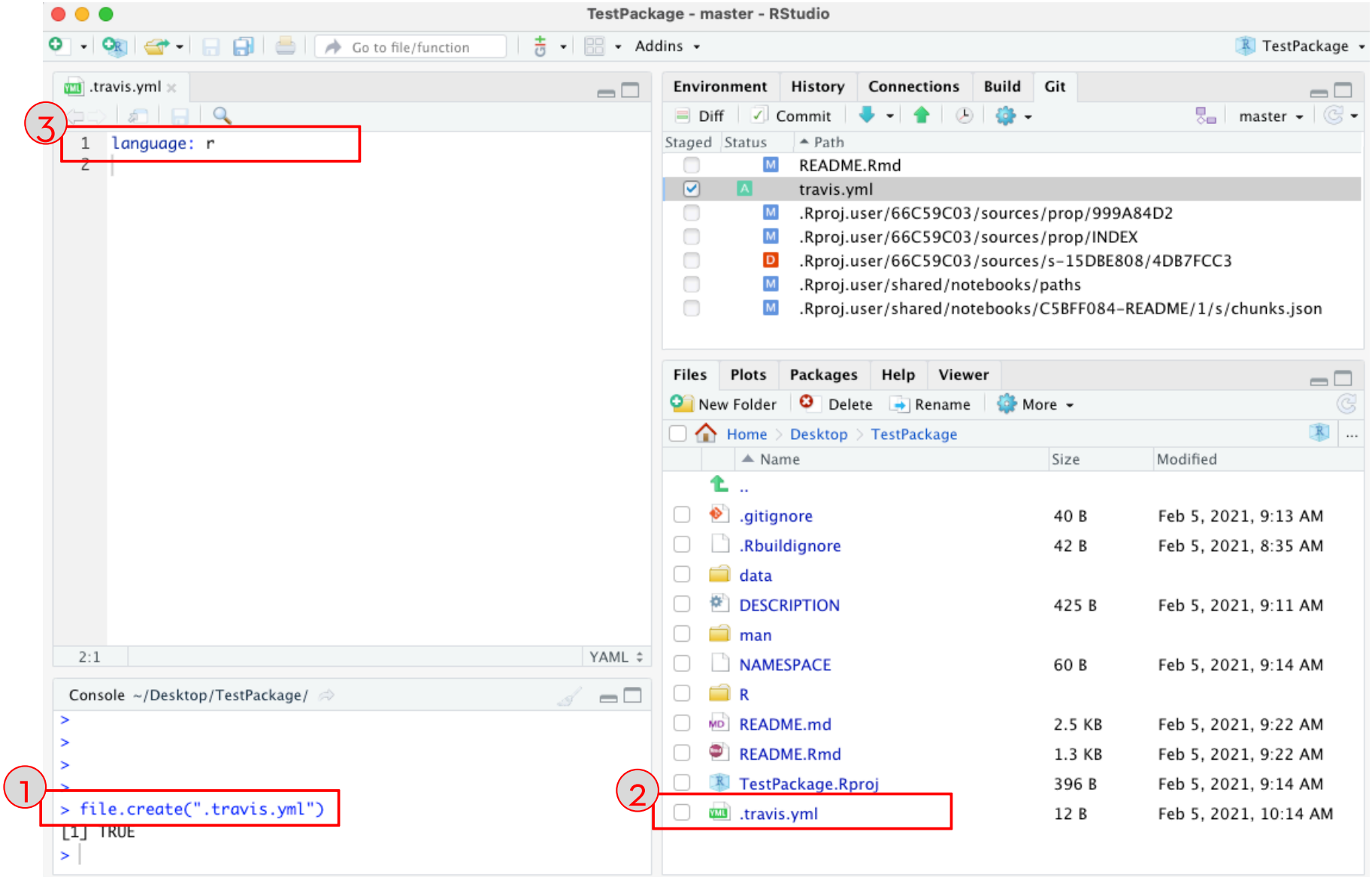
Once you have allowed access you will be taken back to Travis where you will need to enable a specific Git Repository. You can also do this in your Travis Profile: https://travis-ci.com/profile



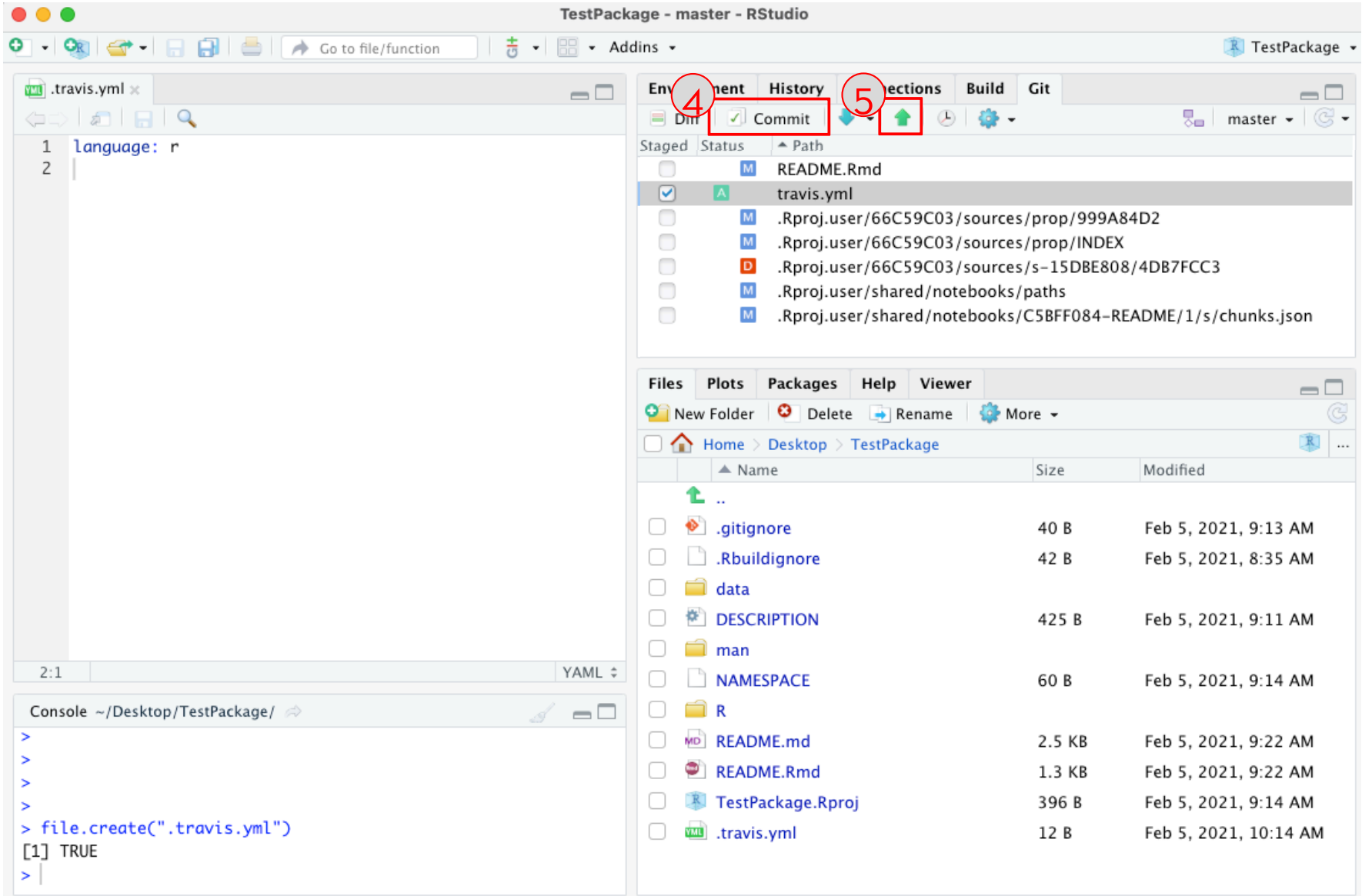


https://github.com/dwyl/learn-travis

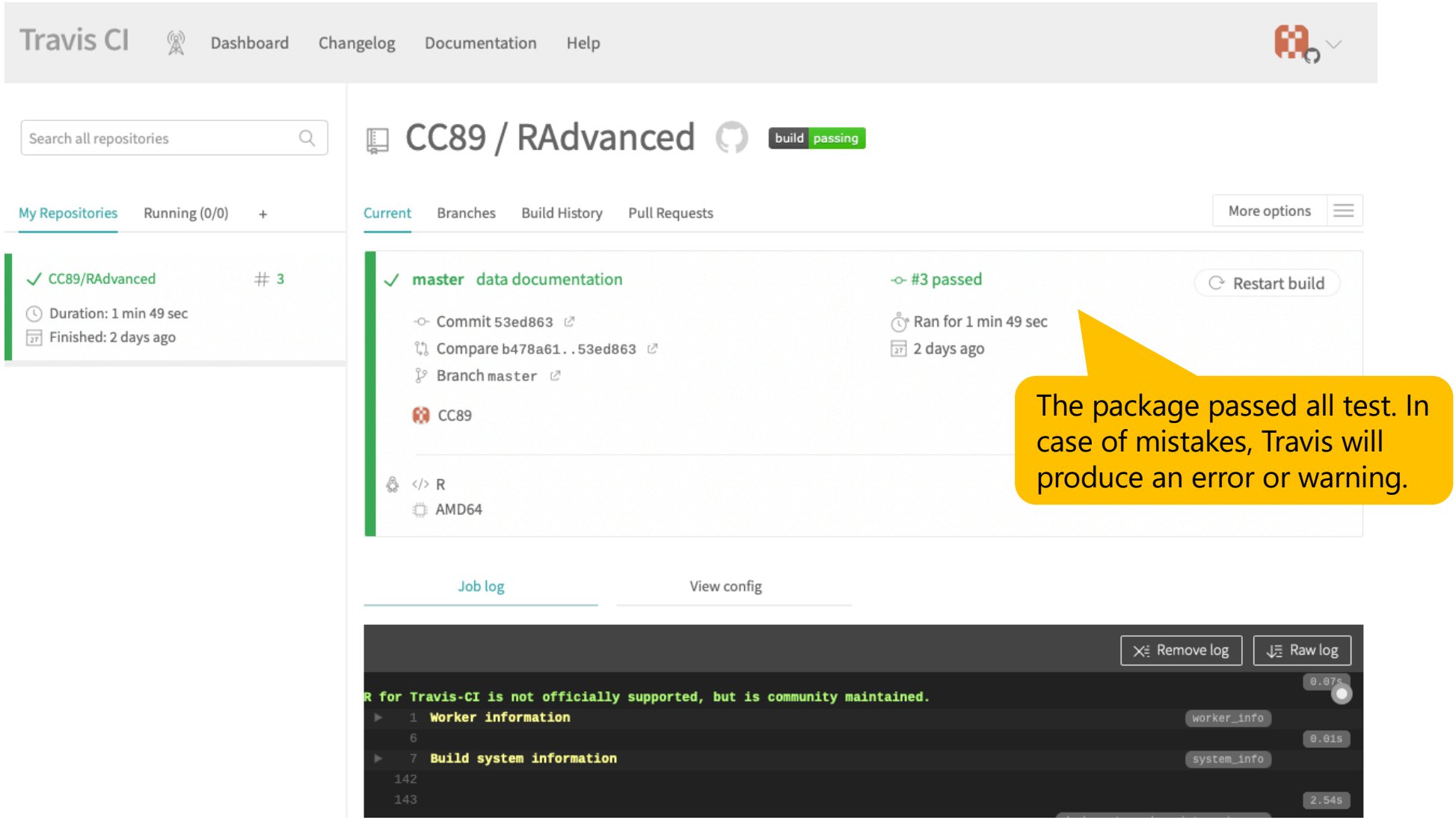
Step 4: Add a travis.yml file to the root directory of your package



Step 5: Commit this new file to your Git repository.



How to see if Travis CI did actually run the tests?



Now it’s your turn!