Surveillance and time series

Richard White

2018-11-08

Contents

List of Tables

List of Figures

Preface

This is a short presentation given at FHI for the surveillance group.

1 Integrating the R package into the physical system

1.1 Summary

An R package is not enough to run an analysis – something needs to physically call the functions inside the R package. That is, the R package needs to be integrated into the physical system.

Everything related to integrating the R package into the physical system lives in the dashboards repository.

Inside the dashboards repository we have:

```
- dev/
  |-- src/
    |-- sykdomspuls/
       |-- 0_run.sh
       |-- RunProcess.R
       |-- RunTest.R
    |-- normomo/
       |-- 0_run.sh
       |-- RunProcess.R
       |-- RunTest.R
    |-- sykdomspuls_log/
       |-- 0_run.sh
       |-- RunProcess.R
       |-- RunTest.R
    |-- sykdomspuls_pdf/
       |-- 0_run.sh
       |-- RunProcess.R
       |-- RunTest.R
```

1.2 RunProcess.R

1.2.1 Aim

An automated analysis needs to:

1. Know the location of the data/results folders.
2. Check for new data in these folders. If no new data - then quit.
3. Load in the data.
4. Load in the analysis functions.
5. Run the analyses.
6. Save the results.

`RunProcess.R` is responsible for these tasks.

We can think of it as an extremely short and extremely high-level script that implements the analysis scripts.

Depending on the automated analysis `RunProcess.R` can be run every two minutes (constantly checking for new data), or once a week (when we know that data will only be available on a certain day/time).

1.2.2 Bounded context

1. Only one instance of `RunProcess.R` can be run at a time.
2. Data only exists on physical folders on the system.
3. The following folder structure exists on the system (here the name of the automated analysis is `ANALYSIS`):

```
/data_raw/
  |-- ANALYSIS/
/data_clean/
  |-- ANALYSIS/
/data_app/
  |-- ANALYSIS/
/results/
  |-- ANALYSIS/
/src/
  |-- ANALYSIS/
     |-- 0_run.sh
     |-- RunProcess.R
     |-- RunTest.R
```

Point #1 is important because if `RunProcess.R` is run every 2 minutes (constantly checking for new data) but the analyses take 3 hours to run, then we need to ensure that only one instance of `RunProcess.R` can be run at a time.

Point #2 is important because sometimes:

1. Data files need to be downloaded from external SFTP servers (normomo, sykdomspulslog).
2. Results files need to be uploaded to external SFTP servers (sykdomspuls).

If we include code to download/upload the files from SFTP servers inside `RunProcess.R` then it makes it very difficult to test `RunProcess.R` (because we will then need to simulate SFTP servers inside our testing infrastructure). If we know that `RunProcess.R` only accesses files that are available on physical folders in the system, then our testing infrastructure is a lot easier to create and maintain.

1.3 0_run.sh

1.3.1 Aim

The aim of `0_run.sh` is to ensure that:

1. Points 1 and 2 of the bounded context of `RunProcess.R` happen
2. Run `RunProcess.R`

With regards to the bounded context, we ensure that only one instance of `RunProcess.R` is run at a time through the use of `flock`.

(If neccessary) with regards to the bounded context, we use `sshpass`, `sftp`, and `ncftpput` to download/upload files from SFTP servers.

We then run `RunProcess.R` with a standard call:

```
/usr/local/bin/Rscript /src/ANALYSIS/RunProcess.R
```

1.4 RunTest.R

1.4.1 Aim

The aim of `RunTest.R` is to perform integration testing on the automated analysis. This integration testing is performed as part of the Jenkins build pipeline.

2 R packages

2.1 Summary

Each automated analysis has its own R package:

- sykdomspuls
- normomo
- sykdomspulspdf
- sykdomspulslog

Each R package contains all of the code necessary for that automated analysis. Typical examples are:

- Data cleaning
- Signal analysis
- Graph generation
- Report generation

2.2 Requirements

The R packages should be developed using unit testing as implemented in the testthat package.

Furthermore, the R package should operate (and be able to be tested) independently from the real datasets on the system. This is because the real datasets cannot be shared publically or uploaded to github. To circumvent this issue, each package will need to develop functions that can generate fake data. GenFakeDataRaw is one example from sykdomspuls.

We also require that unit tests are created to test the formatting/structure of results. ValidateAnalysisResults is one example from sykdomspuls, where the names of the data.table are checked against reference values to ensure that the structure of the results are not accidentally changed.

2.3 Deployment via travis-ci and drat

Unit testing is then automatically run using travis-ci. If the R package passes all tests, then we use drat to deploy a built version of the package to Folkehelseinstituttet's R repository: https://folkehelseinstituttet.github.io/drat/.

3 Umbrella Infrastructure

- Three computers

3.1 Executive summary

The dashboards project is a project at FHI concerned with running automated analyses on data.

In principle, the dashboards project is split up into three parts:

1. The overarching infrastructure (i.e. Docker containers, continuous integration, chron jobs, etc.)
2. The R package for each automated analysis
3. The executable for each automated analysis

3.2 What is an automated analysis?

An automated analysis is any analysis that:

1. Will be repeated multiple times in the future
2. Always has an input dataset with consistent file structure

3. Always has the same expected output (e.g. tables, graphs, reports)

3.3 Why not have one project for each automated analysis?

Automated analyses have a lot of code and infrastructure in common.

Automated analyses:

1. Need their code to be tested via unit testing to ensure the results are correct
2. Need their code to be tested via integration testing to ensure everything runs
3. Need to be run at certain times
4. Need to be able to send emails notifying people that the analyses have finished running
5. Need to make their results accessible to the relevant people

By combining them all in one umbrella project we can force everyone to use the same infrastructure, so we:

1. Only need to solve a problem once
2. Only need to maintain one system
3. Can easily work on multiple projects, as we all speak the same language

4 Contributing

4.1 Development guidelines

We try to follow the GitHub flow for development.

1. Fork [this repo][repo] and clone it to your computer. To learn more about this process, see this guide.

2. Add the Folkehelseinstituttet repository as your upstream:

   ```
   git remote add upstream https://github.com/folkehelseinstituttet/ORIGINAL_REPOSITOR
   ```

3. If you have forked and cloned the project before and it has been a while since you worked on it, merge changes from the original repo to your clone by using:

   ```
   git fetch upstream
   git merge upstream/master
   ```

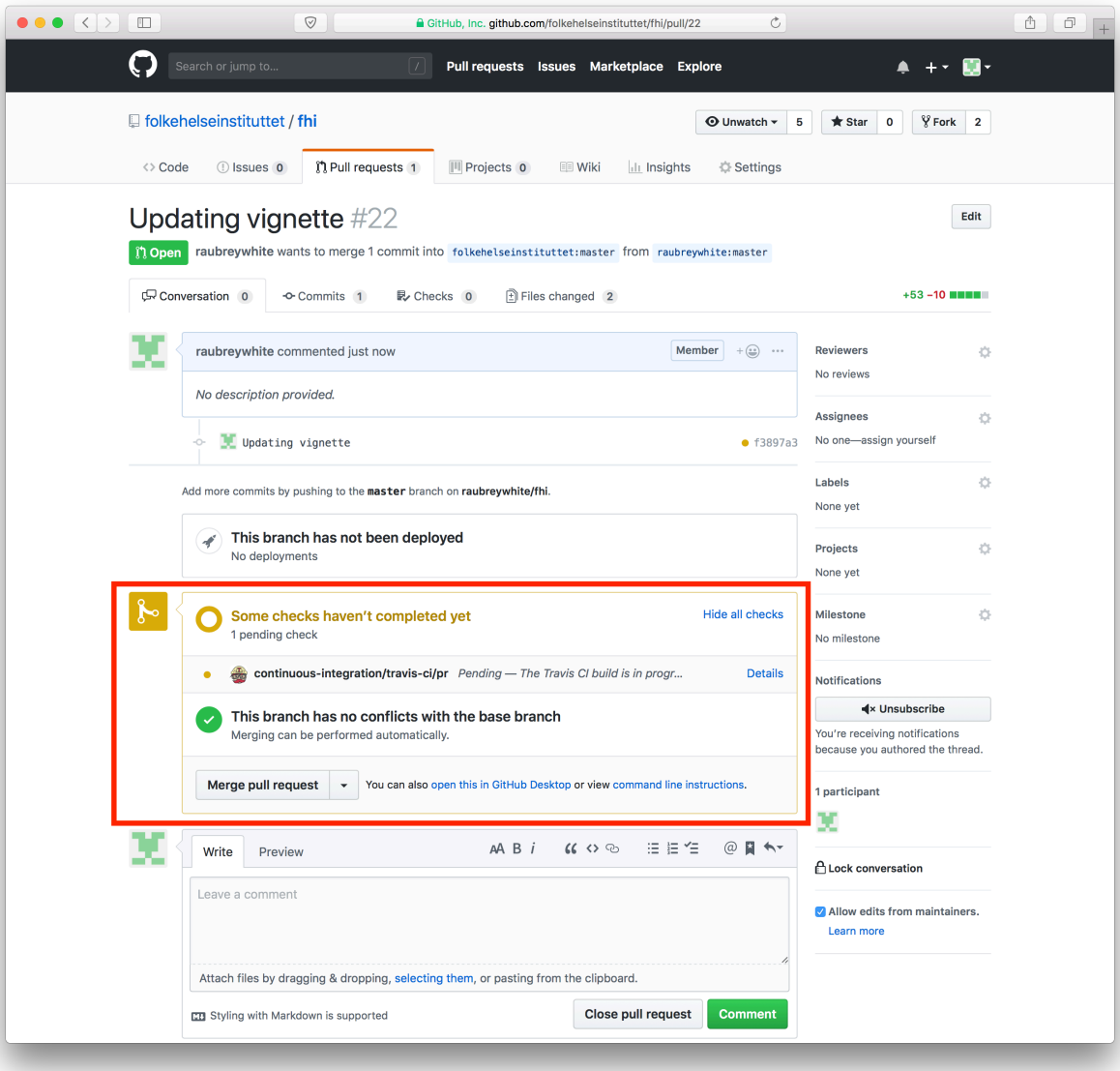
4. Open the RStudio project file (`.Rproj`).

5. Make your changes:

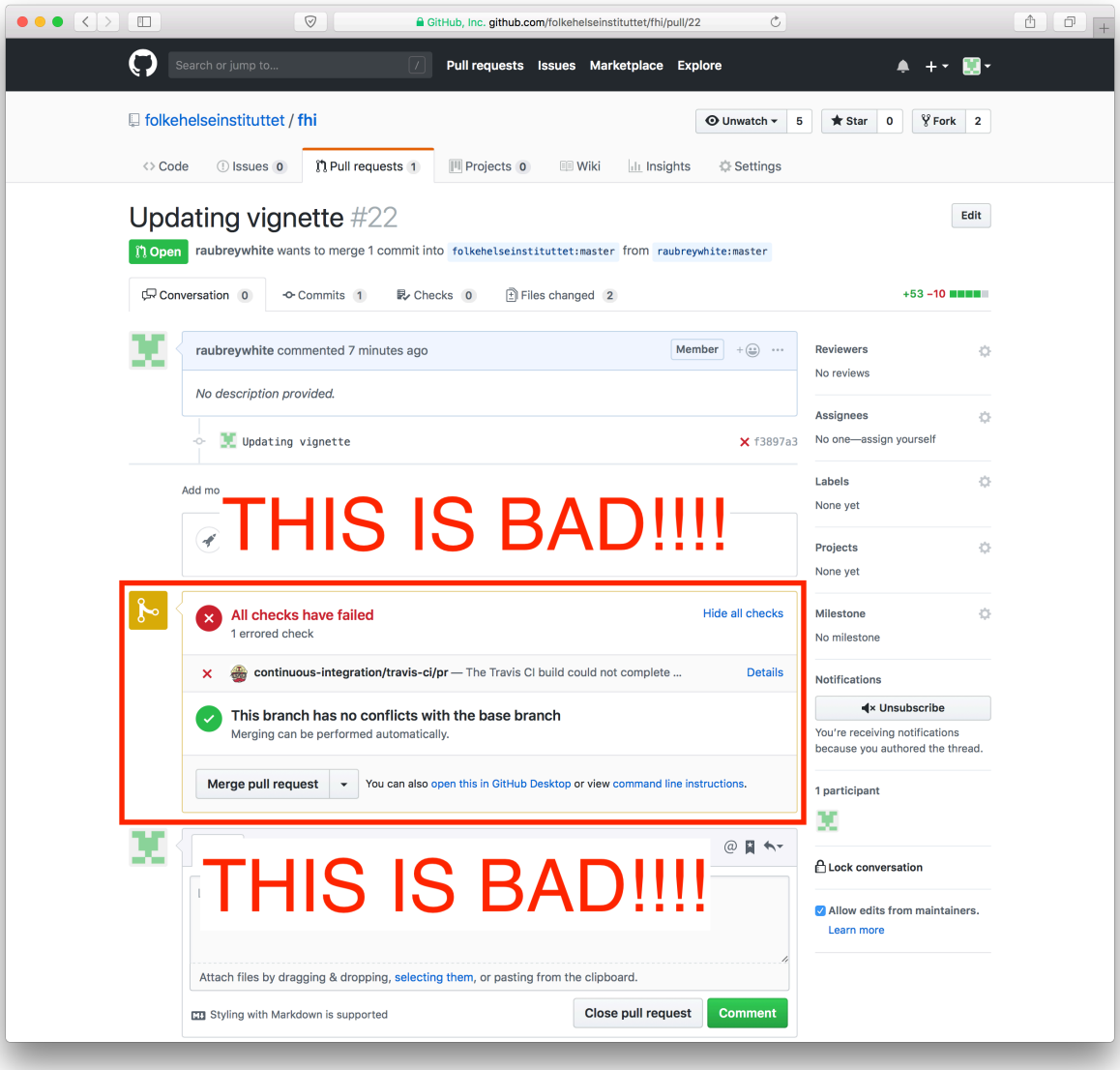
   - Write your code.
   - Test your code (bonus points for adding unit tests).
   - Document your code (see function documentation above).
   - Do an `R CMD check` using `devtools::check()` and aim for 0 errors and warnings.

- Commit your changes locally
- Merge changes from the original repo (again)
- Do an `R CMD check` using `devtools::check()` and aim for 0 errors and warnings.

6. Commit and push your changes.
7. Submit a pull request.
8. If you are reviewing the pull request, wait until the travis-ci unit tests have finished



9. Please make sure that the unit tests `PASS` before merging in!!



4.2 Code style

- Function names start with capital letters
- Variable names start with small letters
- Environments should be in ALL CAPS
- Reference Hadley's style code
- <- is preferred over = for assignment
- Indentation is with two spaces, not two or a tab. There should be no tabs in code files.

- if () {} else {} constructions should always use full curly braces even when usage seems unnecessary from a clarity perspective.
- TODO statements should be opened as GitHub issues with links to specific code files and code lines, rather than written inline.
- Follow Hadley's suggestion for aligning long functions with many arguments:

```
long_function_name <- function(a = "a long argument",
                               b = "another argument",
                               c = "another long argument") {
  # As usual code is indented by two spaces.
}
```

- Never use print() to send text to the console. Instead use message(), warning(), and error() as appropriate.
- Use environment variables, not options(), to store global arguments that are used by many or all functions.