Dashboards project

Richard White

2018-11-09

Contents

List of Tables

List of Figures

Preface

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

If you are just joining this project, please see the quickstart section.

1 Introduction

1.1 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 two parts:

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

1.2 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)

1.3 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 and coding principles, 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

1.4 Important repositories

1.4.1 Infrastructure

https://github.com/raubreywhite/dashboards_control/ (private)

This contains the Dockerfiles, cronfiles, all bash scripts, etc.

http://github.com/raubreywhite/docker/

This contains the base analysis Dockerfile

https://rocker-project.org

https://folkehelseinstituttet.github.io/fhi/

This is an R package that contains helper functions.

1.4.2 Automated analyses R packages

https://folkehelseinstituttet.github.io/dashboards_sykdomspuls/

https://folkehelseinstituttet.github.io/dashboards_normomo/

https://folkehelseinstituttet.github.io/dashboards_sykdomspuls_pdf/

https://folkehelseinstituttet.github.io/dashboards_noispiah/

https://folkehelseinstituttet.github.io/dashboards_sykdomspuls_log/

2 Umbrella Infrastructure

2.1 Physical Hardware and Subscriptions

- One Github organization (http://github.com/folkehelseinstituttet/)
- One Github team (https://github.com/orgs/folkehelseinstituttet/teams/dashboards)
- One drat repository (https://folkehelseinstituttet.github.io/drat/)
- One travis-ci.org account (http://travis-ci.org/folkehelseinstituttet)
- One travis-ci.com account (http://travis-ci.com/folkehelseinstituttet)
- One Docker hub account (http://hub.docker.com/u/raw996/)

- At least three computers:
  1. Production linux computer `smhb`
  2. Integration testing linux computer `linux`
  3. Development linux computers (1 per person)

2.1.1 Requirements - smhb

- Git
- Docker Engine - Community (https://www.docker.com/products/docker-engine)

2.1.2 Requirements - linux

- Git
- Docker Engine - Community (https://www.docker.com/products/docker-engine)
- Jenkins installed via a Docker container (http://jenkins.io)

2.1.3 Requirements - dev

- Git
- Docker Engine - Community (https://www.docker.com/products/docker-engine)

2.2 Configuration, Scripts, etc.

Most of the bash scripts, Docker files, passwords, etc. are all hosted at the private Github repository raubreywhite/dashboards_control.

```
- $DASHBOARDS_FOLDER/dashboards_control/
  |-- bin/
    |-- dev_down.sh
    |-- dev_up.sh
    |-- docker_build.sh
    |-- docker_login.sh
    |-- docker_push_test_to_prod.sh
    |-- prod_down.sh
    |-- prod_up.sh
    |-- prod_update.sh
    |-- public.sh
    |-- test_noispiah.sh
    |-- test_normomo.sh
    |-- test_sykdomspuls_log.sh
    |-- test_sykdomspuls_pdf.sh
    |-- test_sykdomspuls.sh
```

```
|-- infrastructure/
  |-- dashboards_nginx/
  |-- dashboards_r/
    |-- add_autofs.sh
    |-- add_cron.sh
    |-- auto.mounts
    |-- crontab
    |-- Dockerfile
    |-- emails_sykdomspuls_alert_test.xlsx
    |-- emails_sykdomspuls_alert.xlsx
    |-- emails.xlsx
    |-- httr-oauth_2017_09_17
    |-- mail.json
    |-- repo-key
    |-- secret.sh
  |-- dashboards_shiny/
  |-- docker-compose-api.yml
  |-- docker-compose-dev.yml
  |-- docker-compose-prod.yml
  |-- docker-compose-test.yml
```

2.2.1 $DASHBOARDS_FOLDER/dashboards_control/bin/dev_down.sh

Dev script to shut down docker-compose

2.2.2 $DASHBOARDS_FOLDER/dashboards_control/bin/dev_up.sh

Dev script to start docker-compose

2.2.3 $DASHBOARDS_FOLDER/dashboards_control/bin/docker_build.sh

Builds all relevant Docker containers

2.2.4 $DASHBOARDS_FOLDER/dashboards_control/bin/docker_login.sh

Logs in to docker-hub

2.2.5 $DASHBOARDS_FOLDER/dashboards_control/bin/docker_push_test_to_prod.s

Retags 'test' Docker containers to 'production' and pushs them to docker-hub

2.2.6 $DASHBOARDS_FOLDER/dashboards_control/bin/prod_down.sh

Only to be used on the production computer `smhb`. This bash script stops the docker-compose.

Note: This script is not used.

2.2.7 $DASHBOARDS_FOLDER/dashboards_control/bin/prod_up.sh

Only to be used on the production computer `smhb`. This bash script starts up the docker-compose.

Note: This script is not used.

2.2.8 $DASHBOARDS_FOLDER/dashboards_control/bin/prod_update.sh

Only to be used on the production computer `smhb`. This bash script:

1. Removes unused Docker container/images (important so we don't run out of space)
2. Runs some scripts required for network access
3. Pulls the latest version of raubreywhite/dashboards_control
4. Stops the docker-compose-prod.yml
5. Pulls the latest production images for:
    - raw996/dashboards_r:production
    - raw996/dashboards_nginx:production
    - raw996/dashboards_shiny:production
6. Starts the docker-compose-prod.yml

2.2.9 $DASHBOARDS_FOLDER/dashboards_control/bin/public.sh

Lists a bunch of environmental variables

2.2.10 $DASHBOARDS_FOLDER/dashboards_control/bin/test_noispiah.sh

Runs `/r/noispiah/src/RunTest.R` inside raw996/dashboards_r:test

Note: This script is generally only run by `Jenkins` on `linux` as the integration testing for noispiah.

2.2.11 $DASHBOARDS_FOLDER/dashboards_control/bin/test_normomo.sh

Runs `/r/normomo/src/RunTest.R` inside raw996/dashboards_r:test

Note: This script is generally only run by `Jenkins` on `linux` as the integration testing for normomo.

2.2.12 $DASHBOARDS_FOLDER/dashboards_control/bin/test_sykdomspuls_log.sh

Runs `/r/sykdomspulslog/src/RunTest.R` inside raw996/dashboards_r:test

Note: This script is generally only run by `Jenkins` on `linux` as the integration testing for sykdomspuls_log.

2.2.13 $DASHBOARDS_FOLDER/dashboards_control/bin/test_sykdomspuls_pdf.sh

Runs `/r/sykdomspulspdf/src/RunTest.R` inside raw996/dashboards_r:test

Note: This script is generally only run by `Jenkins` on `linux` as the integration testing for sykdomspuls_pdf.

2.2.14 $DASHBOARDS_FOLDER/dashboards_control/bin/test_sykdomspuls.sh

Runs `/r/sykdomspuls/src/RunTest.R` inside raw996/dashboards_r:test

Note: This script is generally only run by `Jenkins` on `linux` as the integration testing for sykdomspuls.

2.2.15 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/add

See autofs.

2.2.16 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/add

See cron.

2.2.17 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/aut

See autofs.

2.2.18 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/cro

See cron.

2.2.19 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/Do

See here.

2.2.20 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/ema

A list of email addresses used in sykdomspuls.

2.2.21 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/ema

A list of email addresses used in sykdomspuls.

2.2.22 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/ema

A list of email addresses used in many projects.

2.2.23 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/htt oauth_2017_09_17

The authorization file for the dashboardsfhi@gmail.com. This probably needs to be refreshed every year??

2.2.24 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/mai

Related to httr-oauth.

2.2.25 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/rep key

The repo-key for downloading the private raubreywhite/dashboards_control repository.

2.2.26 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/dashboards_r/sec

Passwords.

2.2.27 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/docker-compose-api.yml

Docker-compose file used for testing the API.

2.2.28 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/docker-compose-dev.yml

Docker-compose file used for development. See here.

2.2.29 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/docker-compose-prod.yml

Docker-compose file used for the production computer. See here.

2.2.30 $DASHBOARDS_FOLDER/dashboards_control/infrastructure/docker-compose-test.yml

Docker-compose file used for integration testing of `Jenkins` on `linux`. See here.

2.3 Analysis Docker Image

2.3.1 Images

Our analysis Docker images are based off the rocker images. More specifically, the rocker/verse:3.5.0 image.

This Docker image is then expanded upon by a separate Dockerfile raw996/dhadley. This Docker image is automatically rebuilt by `Jenkins` on `linux` whenever the repository is updated. The resultant Docker image is pushed to raw996/dhadley:3.5.0. This image is a general-purpose analysis image, with no sensitive information in it.

This Docker image is then expanded upon by a separate Dockerfile raw996/dashboards_r. This Docker image is automatically rebuilt by `Jenkins` on `linux` whenever the repository is updated. The resultant Docker image is locally tagged as `raw996/dashboards_r:test` and then a number of integration tests are performed on it. If the integration tests are passed, then the Docker image is retagged and pushed to raw996/dashboards_r:production. This image is private as it contains passwords and email addresses.

2.3.2 Automated Analysis R Packages

3 travis

3.0.1 File Structure

Inside `raw996/dashboards_r` we have the following file structure:

```
/data_raw/
  |-- normomo/
  |-- noispiah/
  |-- sykdomspuls/
  |-- sykdomspuls_pdf/
  |-- sykdomspuls_log/
/data_clean/
  |-- normomo/
  |-- noispiah/
  |-- sykdomspuls/
  |-- sykdomspuls_pdf/
  |-- sykdomspuls_log/
/data_app/
  |-- normomo/
  |-- noispiah/
  |-- sykdomspuls/
  |-- sykdomspuls_pdf/
  |-- sykdomspuls_log/
/results/
  |-- normomo/
  |-- noispiah/
  |-- sykdomspuls/
  |-- sykdomspuls_pdf/
  |-- sykdomspuls_log/
/usr/local/lib/R/site-library/  <soft linked to /r>
  |-- <OTHER R PACKAGES INSTALLED HERE>/
  |-- fhi/
  |-- normomo/
  |-- noispiah/
  |-- sykdomspuls/
  |-- sykdomspuls_pdf/
  |-- sykdomspuls_log/
  |-- <OTHER R PACKAGES INSTALLED HERE>/
```

Note that we have a soft link between `/r` and `/usr/local/lib/R/site-library/`.

3.0.2 cron

We use cron to schedule the analyses. The schedule is specified in crontab.

The cronjobs are only activated when the environmental variable `ADD=cron` is defined. Cronjobs are then activated through add_cron.sh.

In principle, cronjobs should only be activated on `smhb`.

3.0.3 autofs

We use autofs to connect to the F network. The network locations, username, and password are specified in auto.mounts.

Autofs is only activated when the environmental variable `ADD_AUTOFS=yes` is defined. Autofs is then activated through add_autofs.sh.

In principle, autofs should only be activated on `smhb`.

3.1 Reverse Proxy Docker Image

We use nginx as a reverse proxy to make rstudio server available to the developers.

The relevant Dockerfile is here and is pushed to raw996/dashboards_nginx:production after integration testing is passed.

3.2 Docker Compose

Docker compose is used to integrate these Docker images into the local filesystem. We have multiple docker-compose files for different reasons:

- For production on `smhb`
- For testing on `linux`
- For development on a dev computer

3.3 Production Computer - smhb

3.3.1 crontab

```
# m h  dom mon dow   command
5 1 * * * /home/raw996/git/dashboards_control/bin/prod_update.sh
@reboot /home/raw996/git/dashboards_control/bin/prod_update.sh
```

3.3.2 File Structure

```
/home/raw996/git/dashboards_control/
```

3.3.3 Explanation

`smhb` is designed to be extremely simple. It has two jobs:

1. Updating raw996/dashboards_r:production, raw996/dashboards_shiny:production, and raw996/dashboards_nginx:production
2. Making sure that docker-compose-prod.yml is running

We purposefully minimize all integration with the base machine, because this enables us to be deploy-environment independent and to move our deployment anywhere with minimal hassle.

3.4 Integration Testing Computer - linux

3.4.1 File Structure

```
/home/raw996/
  |-- docker-compose-jenkins/
    |-- docker-compose.yml
  |-- data/
    |-- data_app/
      |-- normomo/
      |-- sykdomspuls/
    |-- data_clean/
      |-- normomo/
      |-- sykdomspuls/
    |-- data_raw/
      |-- normomo/
        |-- FHIDOD2_20170425.txt
      |-- sykdomspuls/
        |-- partially_formatted_2017_08_01.txt
    |-- data_results/
      |-- normomo/
      |-- sykdomspuls/
  |-- jenkins/
```

3.4.2 Explanation

`linux` has three jobs:

1. Building [raw996/dhadley](https://github.com/raubreywhite/docker/blob/master/dhadley/Dockerfile
2. Integration testing for the automated analyses
3. Pushing raw996/dashboards_r:production, raw996/dashboards_shiny:production, and raw996/dashboards_nginx:production to Docker hub after successful integration testing

Integration testing happens by Jenkins running:

- test_noispiah.sh
- test_normomo.sh
- test_sykdomspuls.sh
- test_sykdomspulslog.sh
- test_sykdomspulspdf.sh

3.5 Unit Testing - travis-ci

Travis-ci is used for unit testing of packages. 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/.

4 R packages

4.1 Generic

4.1.1 Overview

Each automated analysis has its own R package:

- sykdomspuls
- normomo
- noispiah
- 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

4.1.2 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.

4.1.3 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/.

4.1.4 Integration with the local file system

We assume that the local file system follows this file structure, and this is provided via Docker-compose from the umbrella infrastructure.

Referencing the `data_raw`, `data_clean`, `data_app`, and `results` folders are done through the the fhi package.

4.1.5 inst/src/RunProcess.R

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).

4.1.6 inst/src/RunTest.R

This file is the brains of the integrated testing. This file will be run by the $DASHBOARDS_FOLDER/dashboards_control/bin/test_*.sh files.

4.1.7 inst/bin/0_run.sh

This file is used to:

1. Do any pre-analysis steps (e.g. download new data from an SFTP server)
2. Run inst/src/RunProcess.R

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

3. Do any post-analysis steps (e.g. upload results to an SFTP server)

This file will be run by cron.

5 Contributing/Quickstart

5.1 Quickstart

1. Create a project folder for your code (let us say `~/git/dashboards`). This folder will be accessible from inside your Docker container as `/dashboards/`
2. Create a project folder for your data (let us say `~/data/`).
3. Clone the following repos inside `~/git/dashboards`:
   - https://github.com/raubreywhite/dashboards_control/ (private)
4. Fork the following repos, and then clone the forks to your computer inside `~/git/dashboards`:
   - https://github.com/folkehelseinstituttet/dashboards/
   - https://github.com/folkehelseinstituttet/dashboards_sykdomspuls/
   - https://github.com/folkehelseinstituttet/dashboards_normomo/
   - https://github.com/folkehelseinstituttet/dashboards_sykdomspuls_pdf/
   - https://github.com/folkehelseinstituttet/dashboards_noispiah/
   - https://github.com/folkehelseinstituttet/dashboards_sykdomspuls_log/
5. For each repo, add the folkehelseinstitutet repository as your upstream:

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

6. In your `~/.profile` add the following three lines:

```
export DASHBOARDS_DATA=~/data/
export DASHBOARDS_FOLDER=~/git/dashboards/
export PATH=$PATH:$DASHBOARDS_FOLDER/dashboards_control/bin/
```

7. Build your Dockerfiles:

```
docker_build.sh
```

8. A number of folders will have been automatically created (see below). Please put in your development datafiles into the appropriate `data_raw/` folder:

```
- $DASHBOARDS_DATA/
  |-- data_app/
    |-- sykdomspuls/
    |-- normomo/
    |-- sykdomspuls_log/
    |-- sykdomspuls_pdf/
  |-- data_clean/
    |-- sykdomspuls/
    |-- normomo/
    |-- sykdomspuls_log/
    |-- sykdomspuls_pdf/
  |-- data_raw/
    |-- sykdomspuls/
    |-- normomo/
    |-- sykdomspuls_log/
    |-- sykdomspuls_pdf/
  |-- results/
    |-- sykdomspuls/
    |-- normomo/
    |-- sykdomspuls_log/
    |-- sykdomspuls_pdf/
```

9 Run your docker-compose:

```
dev_up.sh
```

10. Open a browser and go to http://localhost:8788/
11. Login using `username=rstudio` and `password=rstudio1`
12. Using the project menu, open the project corresponding to the automated analysis you are interested in. This should be located at `/dashboards/`.

5.2 New Automated Analysis

This will walk you through the creation of a new automated analysis called "yyyy".

1. Create a new repository on folkehelseinstituttet with the name `dashboards_yyyy`.
2. Fork this repository to your Github account
3. Clone the forked repository to `~/git/dashboards/dashboards_xxxx`
4. Download dashboards_template and copy all of its contents into `~/git/dashboards/dashboards_yy`
5. Edit the following files:
   - `~/git/dashboards/dashboards_yyyy/DESCRIPTION` (`Package: xxxx` -> `Package: yyyy`)
   - `~/git/dashboards/dashboards_yyyy/tests/testthat.R` (`library(xxxx)->library(yyyy)`)
   - `~/git/dashboards/dashboards_yyyy/.travis.yml` (xxxx->yyyy)
   - `~/git/dashboards/dashboards_yyyy/inst/bin/0_run.sh` (xxxx->yyyy)

- `~/git/dashboards/dashboards_yyyy/inst/src/RunProcess.sh` (xxxx->yyyy)
- `~/git/dashboards/dashboards_yyyy/inst/src/RunTest.sh` (xxxx->yyyy)

6. Ask Richard to log into travis-ci.com and add the `GITHUB_PAT` environmental variable to activate continuous integration for `yyyy`
7. Submit pull requests and ensure that:
   - Everything works
   - The package is successfully build and included in https://github.com/folkehelseinstituttet/drat/tree/gh-pages/src/contrib
8. Add your package under the secton "## DRAT PACKAGES FROM FHI" at https://github.com/raubreywhite/dashboards_control/blob/master/infrastructure/dashboards_r/Dockerfile
9. Create the 4 folders `data_raw`, `data_clean`, `results`, `data_app` at https://github.com/raubreywhite/dashboards_control/blob/master/infrastructure/dashboards_r/Dockerfile
10. Add your package's 0_run.sh to https://github.com/raubreywhite/dashboards_control/blob/master/infrastructure/dashboards_r/crontab
11. Add your package's test_yyyy.sh to `Jenkins` on `Linux`
12. Hope it works!

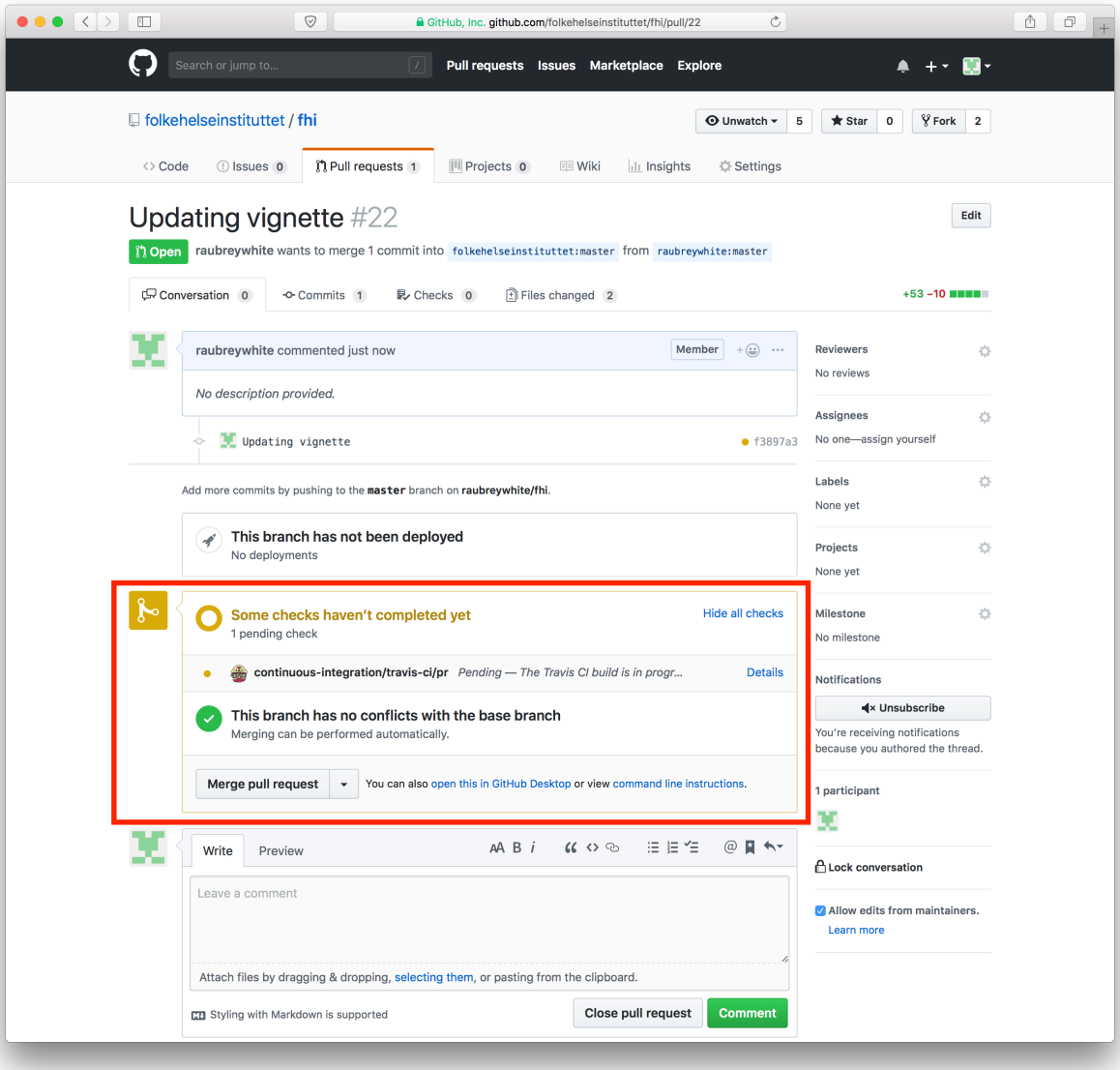
5.3 Development guidelines

We try to follow the GitHub flow for development.

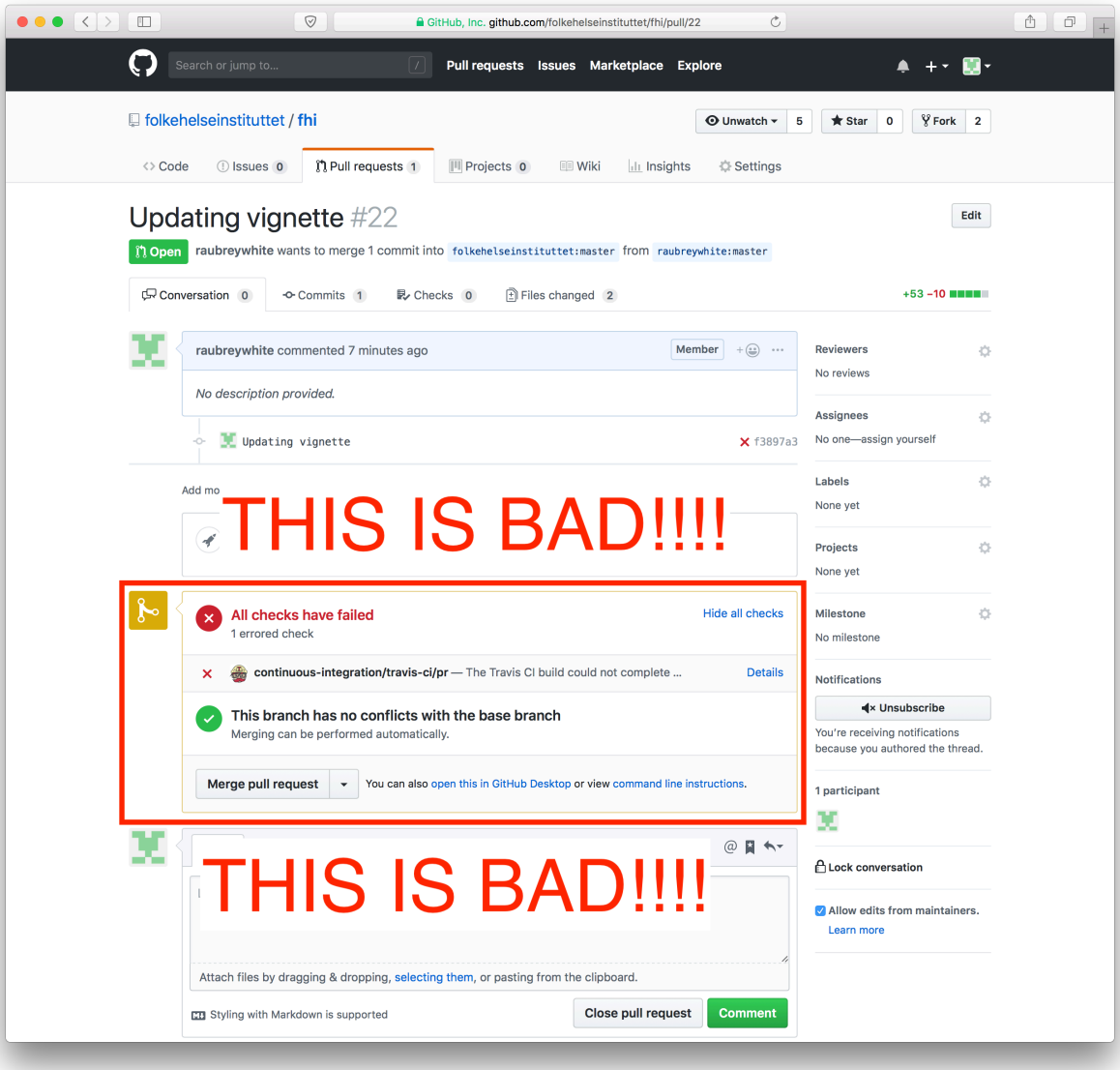
1. If you have forked and cloned the project before and it has been a while since you worked on it, merge changes from the original repo to your clone by using:

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

2. Open the RStudio project file (`.Rproj`).
3. 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.
4. Commit and push your changes.
5. Submit a pull request.
6. If you are reviewing the pull request, wait until the travis-ci unit tests have finished



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



5.4 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.