R training Achmea Bank week 2

Analysis, documentation and version control

Lars Quaedvlieg

16-12-2019

Introduction

Today:

- Remaining questions from last week
- Version control using Git
- Building a predictive model
- Documentation and structurizing your code

Introduction to Version Control

- For keeping track of your code, instead of saving seperate files named project_v12_final_final_final.R etc. we can use Git in combination with a remote repository (e.g. Github or BitBucket)
- Git is a program (written by Linus Thorvalds) that can be accessed via the terminal. Luckily, R studio (and many other editors) has good Graphical User Interfaces for Git
- With Git you can restore previous versions, collaborate on a project at the same time, review each other's code, seperate development versions from production versions and more

Set up a project

- Create a repository on BitBucket or use an existing one (You need admin rights to be able to create a repository)
- Make a new project in R-Studio in your home directory from version control
- Clone the repository

Track code

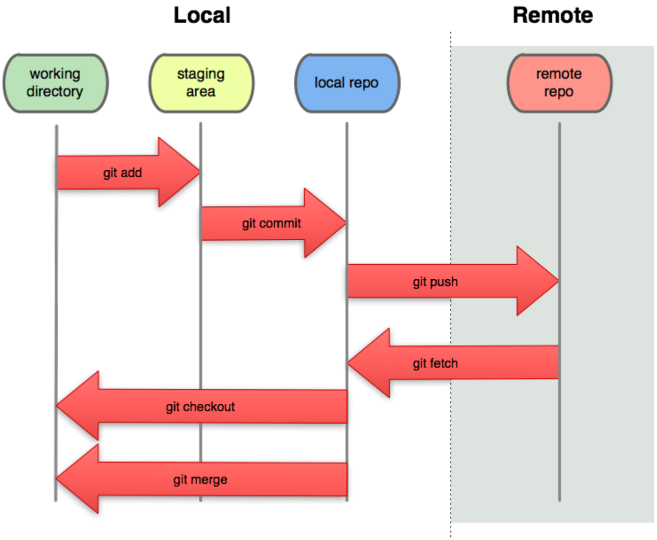

Figure 1: Git Overview

Track code

- When you are happy with your code, use `git add` and `git commit` to 'save' this version of your code in the local repository
- As a rule of thumb, only commit when there are no fatal bugs in your code and your code satisfies the coding standards
- One can regard a commit as an anchor point when climbing a mountain: If you will fall later, you can always return to this point

Track code

- If you want other collaborators to be able to access your code, use `git push`.
- Also do this once every few commits when you are working on a project on your own
- The code will new become available on the remote repository
- Use `git pull` to get code that has been pushed to the remote repository. Rule of thumb: pull before push

Merge conflict

- It can occur that you and someone else have been working on the same part of the code at the same time
- If you pull this code, you will have a so-called merge conflict
- You need to resolve the conflicts manually and subsequently commit and push
- Test the new code for fatal bugs before committing

Branching

- If you want to work on a feature without being disturbed by others, you can create a branch
- The master branch should be the branch that is always working (e.g. in production)
- On a develop branch you can develop new features for your code
- When you are happy with your code, you can merge your branch with a target branch

Branching

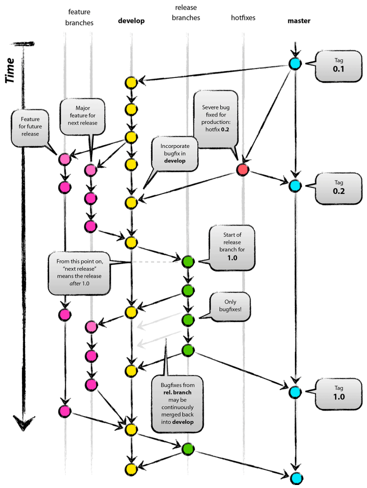
feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Figure 2: Branching Overview

Pull Requests

- If you are working in teams on a project and you want to merge a branch with another branch, you can create a pull request
- Your colleagues can review your code, add comments/suggestions and approve or deny a pull request
- In this way, the code in your project is double checked and the top “layers” of the branching stay clean of bugs
- You can create a pull request in BitBucket (or GitHub)

Pull Requests

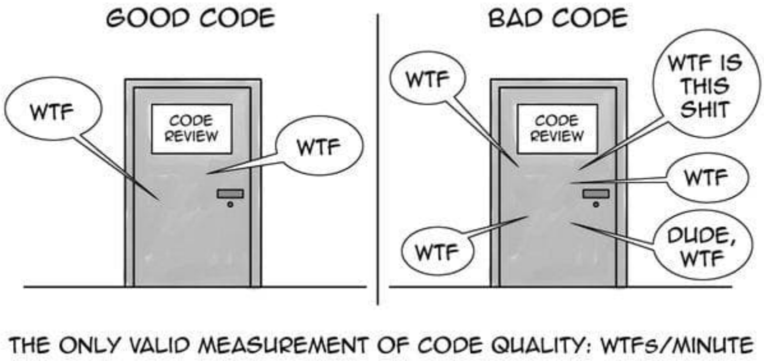

Figure 3: Code Reviews

Ignore

- There are always some files that you do not want to be tracked by Git
- e.g. passwords, (large) datasets, API keys, outputs, plots, sensitive information etc.
- Use the file .gitignore to specify the folders or files that git should never track

Documentation

- Use roxygen skeletons to document your functions
- Add a top header to every script including the author, date, description and usage of the file
- When reviewing code, review comments and documentation as well
- Add a ReadMe to the project
- Be consistent in your code
- Use meaningful variable names

Structurizing

- Keep your code as general as possible
- Functions should be reusable
- Restrict the creation of global variables (variables than can be assessed from anywhere) to only those that are necessary
- Do not hardcode any values. Use parameters instead
- Use files as input, limit the number of manual steps by the end-user to the absolute minimum
- (Visually) describe the dependencies between functions in your documentation

Analysis

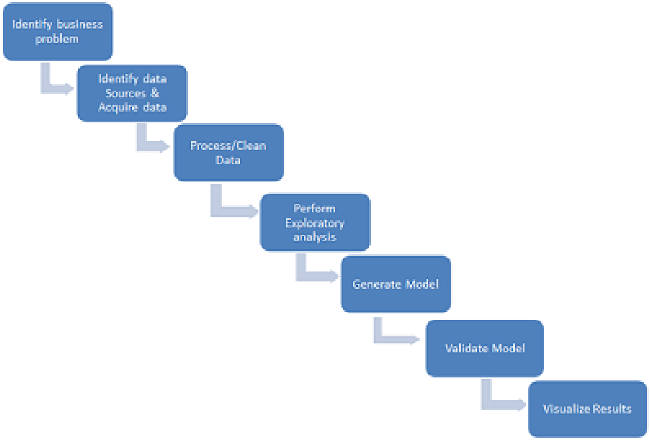
Identify business problem
Identify data Sources & Acquire data
Process/Clean Data
Perform Exploratory analysis
Generate Model
Validate Model
Visualize Results

Figure 4: Data Analysis Overview

Analysis

- Think mathematically on what kind of model you would like to use
- Google what package in R can create such a model
- Build and train the model
- Validate your model on unseen data
- Implement the model

Tutorial

Two parts: - I. Write a function that is as general as possible - II. Work together on a model via Git

Model in Git

- Create teams of 2-3 persons
- If you are working on a private/non-Achmea laptop, make sure your teammates are as well
- Clone the project from BitBucket (Achmea laptop) or GitHub (Private laptop)
- For every team, I have created a branch
- First, work together on this branch
- Later, create a branch from this branch and make a pull request. Add your team mates and me as reviewer.