List of things to discuss and work on

Date: 2017-06-22
Present: Radovan, Thor, Jyry, Bjørn

3-6 month questionnaire

- Finalize typeform and email (email text sent to technical writer, should have feedback before thursday)
- Send to course participants
- https://kthw.typeform.com/to/AiZoPc

Lessons

- Idea: divide into self-study and workshop versions
- Go through Richard Darst's material (http://rkd.zgib.net/scicomp/) and see how we can use it
- Brain-storming: New lesson on reproducible research
- Idea: invite R expert to next workshop. What specifications should we provide?
    - Do we want a statistics expert or an R programming expert?
- A language-independent hands-on introduction to testing

CodeRefinery website

- New layout for workshop page? One experimental option in: https://github.com/wikfeldt/coderefinery.org/tree/new-workshop-layout
- Use our logo

Core content analysis for modules

Git

- Content that is essential
    - Commit
    - Commit message
    - Log
    - Branch
    - Merging
    - Staging area
    - Merge conflict and resolving it
    - Remote repository
- Content that is be useful that a
    - Fast-forward merges vs merges with merge commit (or does this come later?)
- Content that is helpful but not essential to continuing
    - Git aliases
    - Git stash
    - The structure of the .git directory

Git collaborative

- Content that is essential
    - Local repository
    - Remote repository
    - Connection between local and remote branch
    - Fork
    - Motivation for the forking model
    - Pull Request
- Content that is helpful but not essential
    - Git hooks

Jyry: I would like to add elementary code review to this exercise by having Travis test that the tweets are less than 140 characters long. This would also neatly introduce Travis at the concept level.

Testing

-

IDEs

What is the goal of this session? Focus on what is easy in an IDE and difficult without it.

- Essential content
    - Using an IDE can save developer time
    - IDEs are good for writing tests alongside the code
    - You can do most everyday operations things inside an IDE, but not all the complex ones
    - Step-by-step debugging
    - Refactoring support
- Often useful but not always essential content
    - Jupyter support
    - Virtual environment support (too python-specific)
- Can we do the examples in a couple of JetBrains IDEs?
    - Java
    - JavaScript
    - **C/C++**
    - PHP
    - Ruby
    - .Net
    - Databases
    - Go
    - iOS

Distributing

- Motivation: nobody wants to build your software from scratch if they can avoid it
- Examples of packaging in a couple of languages
    - Python
    - R
    - C source distribution?

Licensing

- Core content
    - What is copyright and why does it extend to code?

    - Where does the copyright go to?
        - Finland and Norway, everything to the employer
        - In Sweden there needs to be a separate clause
    - Why all software that is intended to be shared should have a license
    - Main licenses
        - BSD
        - Apache
        - GPL/LGPL (and why not to)
        - MIT
    - Concept of derivative work

Documenting

- Core content
    -

What should we change in the lessons?

These impressions were typed up one day after the workshop, still fresh in our memories.

General:

- visually separate what you can read later and what you definitely need to read, since we “rush” people through they risk otherwise scrolling over
    - CSS classes
- There was no time for branch design
- Add a document where we give people an overview of the accounts they create during the workshop and how they can again get rid of them after the workshop
- We should talk about modular code development
- Tell more clearly what the minimum is and then build things up, e.g. in documentation the minimum is document how to run something.
- Make an introduction of the contents of the course to be given in the beginning of day 1

Overview:

- Always need: version control, licensing
- Once you start writing non-trivial code or have a team: collab, testing, complexity
- Once you have users: documentation, distribution
- Need for more speed or interfacing to legacy code: MMA
- Need to teach or showcase results, exploratory work: Jupyter
- Reproducible platforms: Docker
- Once you want to have several tools under one roof: IDE

Git intro:

- staging area discussion was confusing and did not go well; visualize, clarify, simplify, use whiteboard and post-its to demonstrate staging, take snapshots with cell-phone

Jupyter:

- start slower
- first start by opening a new notebook and play around with it, type-along style
- we need to better motivate why notebooks can be useful [Radovan: I do not know how to make it better]
- Could we motivate it as a special case of documenting code and
- Don't spend as much time on nbdime-diff (somehow it did not work as some participants expected and then they spent 10-15 mins debugging the diffing)
- Keep the group work, perhaps more groups with other use cases

Git collab:

- Add travis check on tweets.
- Mention the value of issues, let people fix an issue with their tweets
- Ask people whether they first want an exercise and then explanation or the other way
- Consider starting with a simple exercise
- Make screenshots or clarify on how to set up Travis and Coveralls
- mention concept of protected branches

Archaeology:

- In the bisect exercise show less digits and the digits then need to be obviously wrong or obviously right.

IDEs:

- make a bit less about pycharm (meaning that our motivation is not so much to promote pycharm but rather promote IDE features and we use pycharm as the platform to demo it)
- simplify the hands-on
- De-emphasize keyboard
- Skip the type-along creation, have a small ready-made project
- Git integration was nice to show but threw windows people off the session since pycharm then could not find the right git.exe
- Demonstrate
    - Refactoring
    - Debugging
    - Automatic styling
    - (running unit tests)

Testing:

- we currently have too much stuff

- move the theory part to a guide
- introduce testing in 5 mins with 2 examples, pseudocode, no jargon
- consider starting with pseudo-code and discussions
- explain how to test functions with side-effects in pseudocode
- mention mocks or demo with pseudocode
- let people refactor a stateful function so that it becomes easy to test
- add 2 more languages so that people can choose
- make adding the untested function more explicit as a separate exercise step
- example of accepting or rejecting an untested function, demonstrate quality controls (green checkmarks)
- make it clear that Travis is not python-specific

CMake:

- Make example worked.
- Move the language features and stuff into a document and out of the demonstration.
- Consider moving entire module to a separate workshop.
- Replacing make with snakemake, and/or look at desktop/work pipelines instead?
- Still fails on Windows.

MMA:

- Examples still too complex.
- Remove dependencies, installing dependencies introduced trouble and additional complexity
- Could the libraries be tested in Travis? We should test as much as possible in Travis, then we at least catch Python 2/3 issues and Linux/Mac.
-

Documentation:

- Simplify exercise with the feature chapter, people were struggling setting up the toc part in the index.rst - ideally they should not need to struggle with that because that was not the main message.
- Consider letting people test out md in one of the online MD editors.
- Consider having exercise with source code and documentation under doc/

Layout for 2-3 day workshops:

Day 1:

- Git intro
- Doc

- Jupyter

Day 2:

- Git collab
- Modular development
- Testing
- IDE
- Licensing

Separate 1-day topics:

- CMake
- MMA
- Archaeology, Branch design
- Packaging
- Complexity
- Docker
- Data management?