A word from the librarian

Karolos Potamianos

FCC Weekly Software Meeting

20 November 2014

A few words about me

- Position: post-doc at LBNL (USA), working on the ATLAS experiment
- Main role: ATLAS Pixel DAQ coordinator since 2 years
- My background: Applied Physics (Engineering degree from ULB, Belgium) and HEP (PhD in Physics from Purdue University, USA)
- Past research: Higgs (ZH/WH at CDF and H → ZZ → 4l at ATLAS), top quark (single top and top pair at CDF), the Etmiss + jets signature

My motivation

- I have worked with code bases from 3 major experiments (CDF, CMS and ATLAS) and think that some things could be better streamlined, causing less headaches to the (increasingly many) collaborators of HEP experiments
- FCC-SW is a starting effort; I hope I can contribute to making development easier for the collaboration (sound workflows facilitate contributions)

How I see my role

- Colin and Benedikt have already started very nice things, and I would like to help them, and the rest of you, with my contributions

- I have enough programming experience in several programming languages to feel comfortable in recommending coding practices ; I would therefore like to propose to update the current FCC SW coding style, with emphasis on:
  - Code modularity and re-usability (but not just for the sake of it)
  - Common code structure, and relevant, meaningful comments
  - Modern computing practices (esp. regarding variable scope and pointers)
  - Modern C++1x programming

- I don't do this just for the sake of it: I would like to work with a code base that is a **breeze to maintain** and which **has many contributors**, a **fast turn-around** (esp. regarding bug fixes), and an **easy learning curve for newcomers** ; **our goal is to get physics studies (and later results) out!**

The git way

- For FCC-SW we git as our version control system (VCS) – I fully support it!
- But git is not svn (as you've probably figured) ; it's (I believe) easier than SVN (and way more powerful) … once you forget about SVN

- In git, **every user has one copy of the whole repository** (with the complete history of commits – not the case of SVN) and **a working copy**
  - This means that one car work **completely offline** and still have his/her code source under version control (committing is still possible!)
  - But this also means that your commits are local and not shared with anybody until you send them somewhere (push in the git lingo)
  - Work can be shared in many ways, but we recommend using GitHub for centralization purposes (and ease of use) – more on this later
- I don't want to explain git here – there are many tutorials out there and I can summarize the major use cases if there's a request for it – but you can contact me for help ; I will do my best to respond in a timely manner

GitHub

- GitHub is a web-service that allows you to host git repositories and share them with the world
- We have set up one such repository for the FCC-SW, which you can find at https://github.com/HEP-FCC/FCCSW
- The repository above is the official repository, our “common code”
- Users can simply get the code by cloning that repository, selecting a branch or a tag (version of the code) and then run it
- Developers, however, need to first clone the repository in GitHub (under their user account) and then use that to develop on
- Once the feature they’re working on is ready, they should test it and then inform the maintainers (a few people) that their feature is ready to use

- This may sound more difficult than the SVN-way but it has many advantages and, in my own opinion, very little drawbacks

GitHub Workflow

Code user

1. On your computer (or lxplus), clone the FCCSW repository
   `git clone https://github.com/HEP-FCC/FCCSW.git FCCSW`
2. Assuming the master branch is what we want, build the code and run

Code contributors (everybody)

1. Fork the FCCSW repository from https://github.com/HEP-FCC/FCCSW (requires a GitHub account)
2. On your cmoputer, clone the FCCSW repository **from your account**
   `git clone https://github.com/karolos-potamianos/FCCSW.git FCCSW`
3. You can build the code, and run it
4. To contribute, create a branch named after your contribution, and add files you modified (or added)
   ```
   git checkout -b methodToFindDarkMatter
   git add ClassToFindDarkMatter.cxx ClassToFindDarkMatter.h
   # Check your code compiles and doesn't crash for common cases
   git commit -m "Adding a new class to find dark matter"
   ```
5. Your changes are local, you need to push them to your remote repository
   `git push # Some more setup is needed – see TWiki`

GitHub Workflow

6. Once you've pushed them to your GitHub account, others can already test them and comment on the implementation (there's a very nice interface)
7. Once you think this feature you're working on is ready and tested, merge the code form the official GitHub repository to keep in synch
    - » You're recommended do do this often in the development process; the more the better as logical conflicts are easier to address (git takes good care of merging in changes to orthogonal code)
    - » Main advantage: who better than the implementer can best fix conflicts arising from the feature he/she's adding or working on ?
8. Submit a **pull request** in GitHub so that the maintainers can merge it with the official FCCSW repository, and make it easily available
    - » This represent a "validation" of the package but of course does not mean that there are no bugs in the said code
    - » At this point, rapid checks will be done regarding the presence of **compilability**, **code comments** and **code structure**
9. Once the request is approved, your code is part of the FCCSW ☺

GitHub Workflow – Remarks

- **While this modus operandi seems long and tedious, it has many advantages**:
  - You can continue your work autonomously until your feature is ready, with version control (unlike with SVN)
  - Others are not affected by pieces you might break (happens a lot with SVN), although please keep in mind that **every commit should compile!**
  - Your user repository on GitHub is a remote backup, with high availability
  - GitHub is very good regarding code and commit history visualization, commenting and issue reporting, which is why many projects use it

- **Additional advantages related to using git**
  - You can work on multiple topics using branches, and quickly switching form one branch to another – this is particular useful for hotfixes, but also for trying parallel development paths that can be discarded later (you only do a pull request once you have your feature ready, keeping the rest local)
  - By committing often (which is recommended) and doing atomic functional commits (adding all your changes in 1 commit is NOT recommended) git will know about the code changes and merging/reverting/etc. is really easy

General coding remarks

- **Collaborating with other developers (esp. not professional coders, although!) requires some common agreements in order to make everybody's life easier**
  - Nobody wants to spend minutes/hours figuring out that this piece of code simply swaps the bits of a word (OK, real-life cases are more tricky)
  - Nobody wants to move across multiple files to figure out that a given function does/doesn't overload a function defined in the interface
  - Nobody wants to spend a day looking into the implementation of a package in order to add a feature that could have taken 10 minutes to add if the design intentions and implementation choices were clearly expressed
- **Some of these concerns are more advanced, but they have a common source: the lack of proper documentation**
  - Agreed, we are physicists and don't want do spend time writing comments, but the community benefits as a whole! Even yourself, a few years later!
  - Documenting also helps in the design: describe the feature before you implement it, and then check it against your description
- **Some features should be enforced at commit as well:**
  - Compilability of code, code structure and also presence of comments