Revision control and build tools tutorials

Sergei Iskakov

2018 Many Electron Collaboration Summer School Coding Workshop

July 6 2018
Outline

• Build tools
  – Build tools and project structure
  – Build and dependency management

• Revision control systems
  – Centralized repository systems (SVN, CVS, ...)
  – Distributed revision control systems (Git)

• Testing and continuous integration
  – Unit test frameworks
  – Continuous integration
Build tools

Before we start the project

• Programming language based choice
  – Language-specific build tool
  – IDE support

• Codebase structure
  – Intra-project dependencies
  – Build tool dependent project structure
Build tools

- **C/C++/Fortran**
  - Make
  - autoconf
  - CMake
  - Classic Unix build tool. Platform dependent
  - Generates Makefile

- **Java**
  - Ant
  - Maven
  - Replaced Make in Java.
  - Full lifecycle support

- **Python**
  - distutils
  - PyBuilder
  - setuptools
  -
Codebase structure

..

- Makefile.am
- cgetv0.f
- cnaitr.f
- cnapps.f
- cnaup2.f
- cnaudp.f
- cneigh.f
- cneupd.f
- cngets.f
- csortc.f
- cstatn.f
- debug.h
- dgetv0.f
- dnaitr.f
- dnapps.f
- dnaup2.f
- dnaudp.f
- dncorv.f
- dneigh.f
- dneupd.f
- Dtest
Codebase structure

dosomething.cpp
Codebase structure

dosomething.cpp

doeverything.cpp
Codebase structure

- Makefile.am
- cgetv0.f
- cnaitr.f
- cnapps.f
- cnaup2.f
- cnaupd.f
- cneigh.f
- cneupd.f
- cngets.f
- csornc.f
- cstatn.f
- debug.h
- dgetv0.f
- dnaitr.f
- dnnapps.f
- dnaup2.f
- dnaupd.f
- dncconv.f
- dneigh.f
- dneupd.f

- pom.xml
  - src
    - main
      - java
        - com
    - test
      - java
        - com

Maven™
Codebase structure

- Try to think in advance
- Use best practices
- Respect your code users
Codebase structure

- Keep the related files together
- Do not spread class implementation into different files
- Extract most related code into library
- Remove unnecessary dependencies
- Separate tests from main code
- Do not add third-party code into your own codebase
- Refactor code structure
- Respect your code users
Simple build

- Simple compilation example
  - `{CXX} src/blabla.cpp src/blabla2.cpp -l./src -o blabla.exe`
Simple build

• Simple compilation example
  – ${CXX}$ src/blabla.cpp src/blabla2.cpp -l./src -o blabla.exe
  – Add library dependence:
    ${CXX}$ src/blabla.cpp src/blabla2.cpp -l./src -o blabla.exe -L${somelibdir} -ltest
Simple build

- Simple compilation example
  - `CXX` src/blabla.cpp src/blabla2.cpp -I./src -o blabla.exe
  - Add library dependence:

    `CXX` src/blabla.cpp src/blabla2.cpp -I./src -o blabla.exe -L${somelibdir} -ltest

  - Add optimization and additional flags...

    `mpicc` -O3 -g -DNDEBUG -mkl=sequential -qopenmp -march=core-avx2 -g main.cpp.o -o GF2 -L/usr/bin/icpc-15.0-base/compiler/lib -lmpi_usempif08 -lmpi_usempi_ignore_tkr -lmpi_mpifh -lmpi libhdf5_fortran.a libhdf5.a ibhdf5hl_fortran.a libhdf5_hl.a -lz -ldl -lm -limf
Simple build

- Simple compilation example
  - \$\{CXX\} src/blabla.cpp src/blabla2.cpp -I./src -o blabla.exe
  - Add library dependence:
    \$\{CXX\} src/blabla.cpp src/blabla2.cpp -I./src -o blabla.exe -L\$\{somelibdir\} -ltest
  - Add optimization and additional flags...
    mpicc       -O3 -g -DNDEBUG -mkl=sequential -qopenmp -march=core-avx2 -g main.cpp.o -o GF2  -L/usr/bin/
icpc-15.0-base/compiler/lib  -lmpi_usempif08 -lmpi_usempi_ignore_tkr -lmpi_mpifh –lmpi libhdf5_fortran.a
    libhdf5.a ibhdf5hl_fortran.a libhdf5_hl.a -lz -ldl -lm -limf

- Use build tools
Make

• Make it easy to build
• Platform dependent
• Need to know where libraries are located

```
OBJ=src/blabra.o src/blabra2.o
INCLUDE="./src"

blabra.exe: ${OBJ}
   ${CXX} ${CXX_FLAGS} ${OBJ} -o blabra.exe

src/%.o : src/%.cpp
   ${CXX} ${CXX_FLAGS} -I${INCLUDE} -c -o $@
```

• It would be good to have Makefiles generated on every platform
Autoconf

- Autoconf is a system for generating a script which will automatically determine the dependency
  - Needs to create project configuration template
  - And Makefile template
Autoconf
Autoconf

• Twice more work to do!
• Not really platform independent
CMake

- Platform-independent
- Easy to write and read!
- Automatically generates useful make targets (make clean)
- Wide repository to track dependencies

```make
cmake_minimum_required(VERSION 2.8)
project(blabla_project CXX)
include_directories(src)
add_executable(blabla.exe src/blabla.cpp src/blabla2.cpp)
```
CMake. Simple build

- Generate Makefile
  - `cmake <codebase root>`

- Run make
  - `make`
  - `make all`
  - `make test`
  - `make world`
CMake. Dependencies

• Add some library dependency

```
cmake_minimum_required(VERSION 2.8)
project(blabla_project CXX)
find_package(MPI)
include_directories(${MPI_CXX_INCLUDE_PATH})
include_directories(src)
add_executable(blabla.exe src/blabla.cpp src/blabla2.cpp)
target_link_libraries(blabla.exe PUBLIC ${MPI_CXX_LIBRARIES})
```

• Resolve library and include path
  
  ```
  c++ -Wl,-search_paths_first -Wl,-headerpad_max_install_names CMakeFiles/blabla.exe.dir/src/blabla.cpp.o CMakeFiles/blabla.exe.dir/src/blabla2.cpp.o -o blabla.exe /usr/local/Cellar/open-mpi/2.0.1/lib/libmpi.dylib
  ```
CMake. Testing

• And tests

CMakeLists.txt

```cpp
enable_testing()
add_subdirectory(test)
add_test(blabla_test test/blabla_test)
add_test(blabla2_test test/blabla2_test)
```

test/CMakeLists.txt

```cpp
add_executable(blabla_test blabla_test.cpp)
add_executable(blabla2_test blabla2_test.cpp)
```

• Creates target "test"

• Make it easy to automate testing
CMake. Custom libraries

• Not all libraries provide a proper `cmake find` script
• Need to create your own.
  – `find_path` for includes
  – `find_library` for libs
  – Try to follow naming conventions
CMake. Summary

• Find all dependencies
• Build all tests
• Compile code in very different environments
• Everything is platform and filesystem independent
Revision control

• Why do we need it?
  – Allows to keep track changes
    • Avoid to create filename mess (e.g. filename.1, filename.2, filename.blablablablalabla ...)
    • Easy to compare different versions
  – Makes it easy to work in a team
    • Conflict resolving
    • Avoid sending files by email
  – Build automation and continuous integration
Which VCS to choose

- **Distributed (git)**
  - Local repository copy
  - More complicated (local + remote)
  - Provided infrastructure (github, bitbucket)
  - Make easy to perform code review

- **Centralized (svn, etc.)**
  - Remote repository
  - Simple and easy
  - Needs additional infrastructure
Git. First steps

- Create repository on github
- Init local repository
  - $> git init .
- Add all files to the stage
  - $> git add .
- Commit to the local repository
  - $> git commit -m "blah-blah-blah"
- Setup remote repository
  - $> git remote add origin remote repository URL
- Publish local repository remotely
  - $> git push -u origin master
Git. First steps

• Get remote repository
  – $> \text{git clone <Repository URL>}$

✓ Clones remote repository locally
✓ Properly configures remote origin

Sample repository: https://github.com/iskakoff/blahblahblah.git
Git. First step

- File changing
  - check project status
    - $> git status
      
      \texttt{modified: src/blabla.cpp}
      
      \texttt{modified: src/blabla.h}
  
  - add changes to stage
    - $> git add <files>
  
  - commit changes
    - $> git commit -m "bla"
  
  - publish remotely
    - $> git push
Git. First steps

• Undo local changes
  – $> git status
  – $> git checkout -- <Filename>
  – $> git status

On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
Git. First steps
Git. First steps

• Get changes from remote
  • $> \text{git pull}$
Git. Conflicts

- During the work your file was changed remotely. pull/push will failed

```
error: Your local changes to the following files would be overwritten by merge:
    src/blabla.h

Please, To https://github.com/iskakoff/blahblahblah.git
    ! [rejected] master -> master (non-fast-forward)
error: failed to push some refs to 'https://github.com/iskakoff/blahblahblah.git'
```

- Conflict need to be resolved
  - `$> git commit -a -m "blah" && git pull`

```
Auto-merging src/blabla.h
CONFLICT (content): Merge conflict in src/blabla.h
Automatic merge failed; fix conflicts and then commit the result.
```
Git. Conflicts

- "<<<<<<<" start of remote changes
- "========" start of local changes
- ">>>>>>>" end of conflicted part
- Resolve conflict, add, commit and push

```cpp
#pragma once

class BlaBla {
    public:
        // construct blah-blah
        BlaBla(int i) : _i(i) {
            // HEAD
            for(int k=1; k<i+1; ++k){
                // do nothing
            }
        // compute some nonsense
        void something();
    private:
        // blah number
        int _i;
};
```
Git. Branches

- Create a branch from master for every new feature.
- Implement and test it on a branch.
- "Merge a branch into master"
Git. Branches

• Create branch
  – $> \text{git checkout -b <branch name>}

• Setup remote branch
  – $> \text{git remote add <remote branch name>}

• Publish your branch remotely
  – $> \text{git push}
Git. Advanced steps

• Does "Master->Branch->Master" pattern is good?
Git. Advanced steps

• Does "Master->Branch->Master" pattern is good?
  – No!
Git. Advanced steps

• Does "Master->Branch->Master" pattern is good?
  – No!
  – Many developer can create a mess in the code
Git. Advanced steps

• Does "Master->Branch->Master" pattern is good?
  – No!
  – Many developer can create a mess in the code

• Pull-request should be used
Git. Advanced steps

• Does "Master->Branch->Master" pattern is good?
  – No!
  – Many developer can create a mess in the code

• Pull-request should be used
  – Changes can be approved by other developer
Git. Advanced steps

• Does "Master->Branch->Master" pattern is good?
  – No!
  – Many developer can create a mess in the code

• Pull-request should be used
  – Changes can be approved by other developer
  – Easy way to force code-review
Git. Pull-request Workflow

1. Fork/Branch and Make Changes
2. Open Pull Request
3. CI Server
4. Peer Review
5. Approve Pull Request
6. Update Pull Request
7. Close and Merge Pull Request
Git. Advanced steps

• Continuous integration (CI)
  – All machines configuration is different (different library versions, different compilers)
  – What works for you does not suppose work on any different computer
  – Use CI services
    • travis
    • jenkins
    • codeship
Continuous integration

- Build and test your code for different architecture
- Can be easily integrated with version control system
- Each push or pull-request will be tested
- Helps to make code more stable
Github+CI. Travis CI

• Simple integration
  – Sign in to Travis CI with your GitHub account
  – Accept the GitHub access permissions
  – Write and push travis configuration file into your repository

• Simply can test different OS/compilers/library versions
Github+Travis

- Specify language
- OS-version
- Compiler version
- Pre-install libraries
- Build and test

```bash
# Specify c language to get right CXX compiler
language: c ffortran

# Ubuntu 14.04 Trusty support
sudo: required
dist: trusty

compiler:
- gcc

# specify additional libraries to install
before_install:

# create task matrix for different configuration
matrix:
  include:
    - compiler: gcc
      addons:
        apt:
          sources:
            - ubuntu-toolchain-r-test
          packages:
            - g++-4.9
            - gfortran-4.9
          env: CXX=g++-4.9 FC=gfortran-4.9
      script:
        - mkdir build
        - cd build
        - cmake -DTesting=ON ..
        - make
        - make test
```
Github+Travis

- Publish commit
- Check for CI report
- Fix tests failure if necessary
- Publish commit

Error notifications

Fix tests failure
Git. IDE integration

• Visualize revision history
• Do not need to remember all commands
• Easy to resolve conflict
Thank you