How-to Software Guide (CWI CI Group)

Jan-Willem Buurlage, Allard Hendriksen

November 21, 2018

Contents

1 TODO Introduction

2 How-to

2.1 Python

2.1.1 TODO Auto-format your code

1. PEP8
2. yapf

2.1.2 Call MATLAB from Python

Author: *Jan-Willem Buurlage*

To install the MATLAB engine wrapper for Python for a Conda environment on your CWI workstation, run the following commands:

```
conda create -n matlab python=3.6
conda activate matlab
cd /opt/sw/matlab-YYYYx/extern/engines/python
python setup.py build --build-base="/tmp" install
```

Now, you can call MATLAB functions from Python:

```
import matlab.engine

# start the MATLAB program (which takes a long time)
eng = matlab.engine.start_matlab()
# alternative: start asynchronously
future = matlab.engine.start_matlab(async=True)
# ... other Python code
eng = future.result()
```

You can call your own MATLAB scripts (`.m` files in the same folder), such as the following `triarea.m`:

```
function a = triarea(b,h)
a = 0.5*(b.* h);
```

from Python using:

```
x = eng.triarea(1.5, 3.5)
```

For conversions between arrays and MATLAB, see the MATLAB Python API docs.

See also:

- Installation documentation from MathWorks
- Calling user scripts from Python

2.2 C++

2.2.1 TODO Auto-format your code

1. `clang-format`

2.2.2 TODO Use CMake to build your software

1. C++ Weekly, Intro to CMake
2. CMakePrimer (LLVM)
3. CppCon 2017: Mathieu Ropert "Using Modern CMake Patterns to Enforce a Good Modular Design"
4. C++Now 2017: Daniel Pfeifer "Effective CMake"
5. Dependency management CMake/Git Example:

```
find_package(ZeroMQ QUIET)

if (ZeroMQ_FOUND)
    add_library(zmq INTERFACE)
    target_include_directories(zmq INTERFACE ${ZeroMQ_INCLUDE_DIR})
    target_link_libraries(zmq INTERFACE ${ZeroMQ_LIBRARY})
else()
    message("'zmq' not installed on the system, building from source...")

    execute_process(COMMAND git submodule update --init --remote -- ext/libzmq
WORKING_DIRECTORY ${CMAKE_SOURCE_DIR})

    set(ZMQ_BUILD_TESTS OFF CACHE BOOL "disable tests" FORCE)
    set(WITH_PERF_TOOL OFF CACHE BOOL "disable perf-tools" FORCE)
    add_subdirectory(${CMAKE_SOURCE_DIR}/ext/libzmq)
    set(ZMQ_INCLUDE_DIR ${CMAKE_SOURCE_DIR}/ext/libzmq/include)
```

```
        # ZeroMQ names their target libzmq, which is inconsistent => create a ghost de
        add_library(zmq INTERFACE)
        target_link_libraries(zmq INTERFACE libzmq)
    endif()
```

6. `https://foonathan.net/blog/2018/10/17/cmake-warnings.html`

2.2.3 TODO Use a good set of compile commands

1. Sensible compile flags
   (a) `-Wall`
   (b) `-Werror`
   (c) `-Wfatal`
   (d) ...

2.2.4 TODO Manage dynamic dependencies

Three places that a binary looks for shared dependencies

1. `LD_LIBRARY_PATH`
2. `rpath` encoded in binary
3. system default paths

Danger of (1) is that it overrides the specific dependencies of all binaries run.

For shared systems, or non-root users, (3) can be a problem.

For 2 you proceed as follows:

- set `LD_RUN_PATH` to something hardcoded
- use `-R` in gcc

To check the `RPATH` in a binary on Linux, use `readelf -d <binary>`.

To list all dynamic dependencies, use `ldd <binary>`

See also: `https://www.eyrie.org/~eagle/notes/rpath.html`.

2.2.5 Create Python bindings using `pybind11`

Author: *Jan-Willem Buurlage*

Adding Python bindings to C++ code is straightforward with pybind11. A good setup is as follows. (All relative to the root folder of the C++ project, which I call `your_project` here)

1. Add pybind11 as a git submodule

   ```
   git submodule add https://github.com/pybind/pybind11.git ext/pybind11
   ```

2. Set up the Python bindings Make a directory `python`, containing at least three files:

   (a) `python/src/module.cpp` This contains the actual bindings, an example is like this:

   ```
   #include <pybind11/pybind11.h>
   namespace py = pybind11;

   #include "your_project/your_project.hpp"

   using namespace your_project;

   PYBIND11_MODULE(py_your_project, m) {
       m.doc() = "bindings for your_project";

       py::class_<your_project::object>(m, "object");
   }
   ```

   (b) `python/your_project/__init__.py` The entry point for the Python specific code of your project. Also reexports symbols from the generated bindings.

   ```
   from py_your_project import *
   ```

   (c) `python/CMakeLists.txt` You can build the bindings using CMake.

   ```
   set(BINDING_NAME "py_your_project")
   set(BINDING_SOURCES "src/module.cpp")

   set(CMAKE_LIBRARY_OUTPUT_DIRECTORY "${CMAKE_CURRENT_SOURCE_DIR}")

   pybind11_add_module(${BINDING_NAME} ${BINDING_SOURCES})
   ```

```
target_link_libraries(${BINDING_NAME} PRIVATE your_project)
```

3. Add it as a subdirectory In the main `CMakeLists.txt` of your project, add the Python folder:

   ```
   ...
   add_subdirectory("ext/pybind11")
   add_subdirectory("python")
   ```

   Now, the python bindings will be built alongside your project.

2.3 General

2.3.1 Contribute to someone else's project

Author: *Henri der Sarkissian*

Having a common software repository helps us to share and use code from other members, but also contribute to them. This implies obtaining the source code (and not the compiled binaries or the conda package) from Github and commit the changes. Obviously, commiting changes comports a risk of introducing bug or unwanted features for the software. You should therefore commit your changes in a separate branch and open a pull request. After inspection, your changes will eventually be accepted and incorporated into the main branch. Let us now describe this procedure step by step.

First, checkout the source code and cd to this directory.

```
git git@github.com:cicwi/RECAST3D.git
cd ~/projects/recast3d/
```

Change something:

```
touch some_file_to_add
```

Stage file:

```
git add some_file_to_add
```

```
git checkout -b reverse_polarity
```

```
git commit -m "Reverse the polarity of neutron flow"
```

```
git push origin reverse_neutron_flow
```

Visit the project page on GitHub, and in the tab *Pull requests* click on *New pull request*. For more information about pull requests, see `https://help.github.com/articles/about-pull-requests/`.

2.3.2 Record an animated GIF of your screen

Author: *Jan-Willem Buurlage*

For recording animated GIFs of a region of your screen, you can use Peek `https://github.com/phw/peek`. The easiest way to get it to run on a CWI workstation is by using an AppImage of a recent release, which you can get from the releases page `https://github.com/phw/peek/releases`.

After downloading, make the file executable (either using your file manager, or by calling `chmod +x [peek_release.AppImage]` from a terminal). Now you can run Peek, which has a straightforward interface: resize the Peek window and click on record, after three seconds it will start recording the region of your screen that is visible.

2.3.3 TODO Write good documentation

- `http://stevelosh.com/blog/2013/09/teach-dont-tell/`

2.3.4 TODO Write good commit messages

- `http://chris.beams.io/posts/git-commit/`

2.3.5 TODO Write a good readme

This github repo contains a useful model of maturity levels for a project's README.md file. It defines both the current level of maturity of a README and gives pointers on how to improve.

2.3.6 TODO Set up your Git branches

- **Branching model**: `http://nvie.com/posts/a-successful-git-branching-model/`

2.3.7 TODO Use module systems

2.3.8 TODO Set up travis CI

1. C++17
2. travis.yml / Makefile