How-to Software Guide (CWI CI Group)

Jan-Willem Buurlage, Allard Hendriksen

November 1, 2018

Contents

Part I

TODO Introduction

Part II

How-to

Chapter 1

Python

1.1 TODO Auto-format your code

1. PEP8
2. yapf

Chapter 2

C++

2.1 TODO Auto-format your code

1. `clang-format`

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 gho
        add_library(zmq INTERFACE)
        target_link_libraries(zmq INTERFACE libzmq)
    endif()
```

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

2.3 TODO Use a good set of compile commands

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

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`.

2.5 Create Python bindings using pybind11

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.

Chapter 3

General

3.1 TODO Write good documentation

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

3.2 TODO Write good commit messages

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

3.3 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.

3.4 TODO Set up your Git branches

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

3.5 TODO Use module systems

3.6 TODO Set up travis CI

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