Tools : getting started with Cmake and git

Cmake

Recall the sequence of commands we performed in the complex number example:

```
g++ -c cmpl.cpp
g++ -c cmpl2.cpp
ar cr libcmpl.a cmpl.o cmpl2.o
g++ -o cmpl_main main.cpp -L. -lcmpl
```

For even larger projects this quickly becomes very cumbersome

we would like to have an easy and automated way of doing this.

old days: write a makefile by hand; drawback : is platform, compiler and often hardware dependent, and hard to use/maintain for very large software packages

Cmake provides an open source and *cross-platform* answer to this, see www.cmake.org

We make a file CMakeLists.txt that looks as follows:

```
$ more CMakeLists.txt
cmake_minimum_required (VERSION 2.6)
project (Tutorial)
add_executable(Tutorial cmpl.cpp cmpl2.cpp main.cpp)
```

Cmake

It is in general a terrible idea to build in the same directory as the source files. Let us assume all source files are in subdirectory src and we make sure there is a subdirectory build:

```
th-ea-lswtb02:Ver1 Lode.Pollet$ tree
.
├── build
└── src
    ├── CMakeLists.txt
    ├── cmpl.cpp
    ├── cmpl.h
    ├── cmpl2.cpp
    └── main.cpp
```

Now we go to the build directory and invoke cmake ../src :

```
th-ea-lswtb02:build Lode.Pollet$ cmake ../src/
-- The C compiler identification is AppleClang 7.3.0.7030029
-- The CXX compiler identification is AppleClang 7.3.0.7030029
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/cc
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/Lode.Pollet/Lectures/Programming2016/Programs/Lec_Cmake/Ver1/build
```

A number of files and directories have been created. The most important one is Makefile

Invoking ccmake . shows a GUI where more options can be seen and changed.

Cmake

Invoking make creates an executable Tutorial

In order to make a library we proceed as follows in the CMakeLists.txt file:

```
$ more CMakeLists.txt
cmake_minimum_required (VERSION 2.6)
project (Tutorial)
add_library(ComplexFunctions cmpl.cpp cmpl2.cpp)
add_executable(Tutorial main.cpp)
target_link_libraries (Tutorial ComplexFunctions)
```

Proceeding as before, this produces an executable Tutorial and a library libComplexFunctions.a in the build directory

Cmake

We can choose to make the library optional as follows (not really useful for this example)

```
cmake_minimum_required (VERSION 2.6)
project (Tutorial)
# should we use our own complex functions?
option (USE_MYCMPL
        "Use Tutorial provided complex implementation" ON)

if (USE_MYCMPL)
  add_library(ComplexFunctions cmpl.cpp cmpl2.cpp)
  set (EXTRA_LIBS ${EXTRA_LIBS} ComplexFunctions)
endif (USE_MYCMPL)

# add the executable
add_executable (Tutorial main.cpp)
target_link_libraries (Tutorial  ${EXTRA_LIBS})
```

Note the use of the EXTRA_LIBS
when invoking ccmake . we see that a new option has become available: USE_MYCMPL which can be switched on or off.

Cmake

As a final step, we would like to add the executable in a specified bin directory, the header file in a include directory, and the library in a lib directory as specified below:

```
.
├── bin
├── build
├── include
├── lib
└── src
    ├── CMakeLists.txt
    ├── cmpl.cpp
    ├── cmpl.h
    ├── cmpl2.cpp
    └── main.cpp
```

we modify the CMakeLists file as follows:

```
cmake_minimum_required (VERSION 2.6)
project (Tutorial)
# should we use our own complex functions?
option (USE_MYCMPL
        "Use Tutorial provided complex implementation" ON)

if (USE_MYCMPL)
  add_library(ComplexFunctions STATIC cmpl.cpp cmpl2.cpp)
  set (EXTRA_LIBS ${EXTRA_LIBS} ComplexFunctions)
endif (USE_MYCMPL)


# add the executable
add_executable (Tutorial main.cpp)
target_link_libraries (Tutorial  ${EXTRA_LIBS})

install (TARGETS Tutorial DESTINATION bin)
install (FILES cmpl.h DESTINATION include)
install (TARGETS ComplexFunctions DESTINATION lib)
```

Cmake

The directories bin, include, and lib are with respect to the cmake_install root.

to make sure the cmake_install root is set correctly, we can invoke cmake as:

```
th-ea-lswtb02:build Lode.Pollet$ cmake -DCMAKE_INSTALL_PREFIX=/Users/Lode.Pollet/Lectures/Programming2016/Programs/Lec_Cmake/Ver4/ ../src/
```

with the GUI ccmake . we can modify the options if needed. After configuring&generating we proceed by:

```
th-ea-lswtb02:build Lode.Pollet$ make
[ 20%] Building CXX object CMakeFiles/ComplexFunctions.dir/cmpl.cpp.o
[ 40%] Building CXX object CMakeFiles/ComplexFunctions.dir/cmpl2.cpp.o
[ 60%] Linking CXX static library libComplexFunctions.a
[ 60%] Built target ComplexFunctions
[ 80%] Building CXX object CMakeFiles/Tutorial.dir/main.cpp.o
[100%] Linking CXX executable Tutorial
[100%] Built target Tutorial
th-ea-lswtb02:build Lode.Pollet$ make install
[ 60%] Built target ComplexFunctions
[100%] Built target Tutorial
Install the project...
-- Install configuration: ""
-- Installing: /Users/Lode.Pollet/Lectures/Programming2016/Programs/Lec_Cmake/Ver4/bin/Tutorial
-- Up-to-date: /Users/Lode.Pollet/Lectures/Programming2016/Programs/Lec_Cmake/Ver4/include/cmpl.h
-- Installing: /Users/Lode.Pollet/Lectures/Programming2016/Programs/Lec_Cmake/Ver4/lib/libComplexFunctions.a
```

The executable is now in the bin directory, the library in the lib directory etc.

Switching between debug and release versions can be set by the CMAKE_BUILD_TYPE flag. To clean all object files, libraries and executables, invoke make clean

To see the explicit commands, invoke make VERBOSE=1