cmake

March 23, 2017

Basics

```
cmake_minimum_required(VERSION 3.1)
project(myproject)

add_executable("${PROJECT_NAME}"
    main.cpp
    mylib.h mylib.cpp
)
```

Basics

```
cmake_minimum_required(VERSION 3.1)
project(myproject)

add_executable("${PROJECT_NAME}"
    main.cpp
    mylib.h mylib.cpp
)
```

- Язык исходников определяется по расширению

Basics

```
cmake_minimum_required(VERSION 3.1)
project(myproject)

add_executable("${PROJECT_NAME}"
    main.cpp
    mylib.h mylib.cpp
)
```

- Язык исходников определяется по расширению
- `*.h` файлы не компилируются, но добавляются в проект

Local libraries

```
cmake_minimum_required(VERSION 3.1)
project(myproject)

add_library(mylib
    mylib.h mylib.cpp
)

add_executable("${PROJECT_NAME}" main.cpp)
target_link_libraries("${PROJECT_NAME}" mylib)
```

Local libraries

```
cmake_minimum_required(VERSION 3.1)
project(myproject)

add_library(mylib
    mylib.h mylib.cpp
)

add_executable("${PROJECT_NAME}" main.cpp)
target_link_libraries("${PROJECT_NAME}" mylib)
```

- `mylib` — имя библиотеки

Local libraries

```
cmake_minimum_required(VERSION 3.1)
project(myproject)

add_library(mylib
    mylib.h mylib.cpp
)

add_executable("${PROJECT_NAME}" main.cpp)
target_link_libraries("${PROJECT_NAME}" mylib)
```

- `mylib` — имя библиотеки
- Имя файла с библиотекой платформозависимо

External libraries

```
find_package(GTest REQUIRED)

add_executable(test_mylib test_mylib.cpp)
target_link_libraries(test_mylib mylib ${GTEST_LIBRARY})
```

External libraries

```
find_package(GTest REQUIRED)

add_executable(test_mylib test_mylib.cpp)
target_link_libraries(test_mylib mylib ${GTEST_LIBRARY})
```

- `find_package(Library)` ищет среди модулей cmake файл `FindLibrary.cmake`

External libraries

```
find_package(GTest REQUIRED)

add_executable(test_mylib test_mylib.cpp)
target_link_libraries(test_mylib mylib ${GTEST_LIBRARY})
```

- `find_package(Library)` ищет среди модулей cmake файл `FindLibrary.cmake`
- Откуда берутся find-скрипты:
  - поставляются вместе с cmake
  - поставляются вместе с библиотекой
  - пишутся руками

External libraries

```
find_package(Boost REQUIRED COMPONENTS filesystem system)

target_link_libraries(mylib
    ${Boost_FILESYSTEM_LIBRARY}
    ${Boost_SYSTEM_LIBRARY}
)
```

- Поведение и аргументы `find_package(Library)` не очень специфицированы

Include directories

```
#include <boost/variant.hpp>
```

- Где ищется файл `boost/variant.hpp`?

Include directories

```
#include <boost/variant.hpp>
```

- Где ищется файл `boost/variant.hpp`?

  ```
  include_directories(${Boost_INCLUDE_DIRS})
  ```

Include directories

```
#include <boost/variant.hpp>
```

- Где ищется файл `boost/variant.hpp`?

  ```
  include_directories(${Boost_INCLUDE_DIRS})
  ```

  - Общие пути поиска для всех target-ов

Include directories

```
add_library(mylib mylib.h mylib.cpp)
target_include_directores(mylib PUBLIC
    ${CMAKE_CURRENT_SOURCE_DIR}
)
```

- Пути поиска только для данного target-а

Header-only libraries

```
add_library(mylib INTERFACE)

target_sources(mylib INTERFACE
    ${CMAKE_CURRENT_SOURCE_DIR}/mylib.h
)

target_include_directories(mylib INTERFACE
    ${CMAKE_CURRENT_SOURCE_DIR}
    ${Boost_INCLUDE_DIRS}
)
```

Link dependencies vs. link interface

- `libtest.so` зависит от `libstatic.a` и `libdynamic.so`

Link dependencies vs. link interface

- `libtest.so` зависит от `libstatic.a` и `libdynamic.so`
- `libstatic.a` будет включен в `libtest.so`

Link dependencies vs. link interface

- `libtest.so` зависит от `libstatic.a` и `libdynamic.so`
- `libstatic.a` будет включен в `libtest.so`
- `libdynamic.so` будет линковаться динамически в рантайме

Link dependencies vs. link interface

- `libtest.so` зависит от `libstatic.a` и `libdynamic.so`
- `libstatic.a` будет включен в `libtest.so`
- `libdynamic.so` будет линковаться динамически в рантайме
- При линковке с `libtest.so` надо линковаться и с `libdynamic.so`

Link dependencies vs. link interface

- `libtest.so` зависит от `libstatic.a` и `libdynamic.so`
- `libstatic.a` будет включен в `libtest.so`
- `libdynamic.so` будет линковаться динамически в рантайме
- При линковке с `libtest.so` надо линковаться и с `libdynamic.so`
  - `libdynamic.so` — link interface для `libtest.so`

Link dependencies vs. link interface

```
target_link_libraries(test
    PRIVATE static
    PUBLIC dynamic
)
```

Link dependencies vs. link interface

```
target_link_libraries(test
    PRIVATE static
    PUBLIC dynamic
)
```

- PRIVATE и PUBLIC библиотеки используются при вызове линкера
- PUBLIC и INTERFACE добавляются в link interface

Link dependencies vs. link interface

```
target_link_libraries(test
    PRIVATE static
    PUBLIC dynamic
)
```

- PRIVATE и PUBLIC библиотеки используются при вызове линкера
- PUBLIC и INTERFACE добавляются в link interface

- Аналогичное для target_include_directories и target_sources

Project hierarchy

```
.
`-- lib1
|   `-- CMakeLists.txt
|   `-- mylib.cpp
`-- lib2
|   `-- CMakeLists.txt
|   `-- mylib.cpp
`-- main.cpp
`-- CMakeLists.txt
```

```
add_subdirectory(lib1)
add_subdirectory(lib2)
add_executable(main main.cpp)
target_link_libraries(main lib1 lib2)
```

Compiler flags

```
set(CMAKE_CXX_FLAGS
    "${CMAKE_CXX_FLAGS}
    -Wall -Wextra -Werror
    -std=c++1y"
)
```

Compiler flags

```
set(CMAKE_CXX_FLAGS
    "${CMAKE_CXX_FLAGS}
    -Wall -Wextra -Werror
    -std=c++1y"
)
```

- Разные компиляторы имеют разные опции

Compiler flags

```
set(CMAKE_CXX_FLAGS
    "${CMAKE_CXX_FLAGS}
    -Wall -Wextra -Werror
    -std=c++1y"
)
```

- Разные компиляторы имеют разные опции
- `-std=c++1y` или `-std=c++14`?

Compiler flags

```
set(CMAKE_CXX_FLAGS
    "${CMAKE_CXX_FLAGS} -Wall -Wextra -Werror")
set(CMAKE_CXX_STANDARD 14)
```

Definitions

```
#ifdef USE_MY_STRUCT
struct mystruct {};
#endif
```

- Как через cmake передать USE_MY_STRUCT?

Definitions

```
#ifdef USE_MY_STRUCT
struct mystruct {};
#endif
```

- Как через cmake передать USE_MY_STRUCT?

```
add_definitions(-DUSE_MY_STRUCT)
```

Definitions

```
#ifdef USE_MY_STRUCT
struct mystruct {};
#endif
```

- Как через cmake передать USE_MY_STRUCT?

```
add_definitions(-DUSE_MY_STRUCT)

add_definitions(-DMAX_SIZE=4096)
```

Definitions

```
#ifdef USE_MY_STRUCT
struct mystruct {};
#endif
```

- Как через cmake передать USE_MY_STRUCT?

```
add_definitions(-DUSE_MY_STRUCT)

add_definitions(-DMAX_SIZE=4096)
```

- Как добавить definition только для одного target-а?

Definitions

```
#ifdef USE_MY_STRUCT
struct mystruct {};
#endif
```

- Как через cmake передать USE_MY_STRUCT?

```
add_definitions(-DUSE_MY_STRUCT)

add_definitions(-DMAX_SIZE=4096)
```

- Как добавить definition только для одного target-а?

```
target_compile_definitions(mylib PRIVATE -DUSE_MY_STRUCT)
```

Functions

```
function(my_function ARG)
    include_directories(${ARG})
endfunction(my_function)
```

Functions

```
function(my_function ARG)
    include_directories(${ARG})
endfunction(my_function)

my_function(${CMAKE_CURRENT_SOURCE_DIR})
```

Examine CMake cache

- Как сменить тип сборки (`Debug`, `Release`, etc.)?

Examine CMake cache

- Как сменить тип сборки (`Debug`, `Release`, etc.)?

```
$ ccmake .
```

- Переменная `CMAKE_BUILD_TYPE` отвечает за тип сборки

Examine CMake cache

- Как сменить тип сборки (`Debug`, `Release`, etc.)?

```
$ ccmake .
```

- Переменная `CMAKE_BUILD_TYPE` отвечает за тип сборки
  - компиляторы могут различаться настолько, что даже тип сборки полностью нельзя абстрагировать

Testing

```
enable_testing()
add_test(test_name binary_name)
```

Testing

```
enable_testing()
add_test(test_name binary_name)

$ ctest --verbose
```

Custom commands

```
generated.cpp: generated.config
    generator -o generated.cpp generated.config
```

- Аналог в cmake?

Custom commands

```
generated.cpp: generated.config
    generator -o generated.cpp generated.config
```

- Аналог в cmake?

```
add_custom_command(
    OUTPUT ${CMAKE_CURRENT_SOURCE_DIR}/generated.cpp
    COMMAND generator -o generated.cpp generated.config
    DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
)
```

Custom commands

```
generated.cpp: generated.config
    generator -o generated.cpp generated.config
```

- Аналог в cmake?

```
add_custom_command(
    OUTPUT ${CMAKE_CURRENT_SOURCE_DIR}/generated.cpp
    COMMAND generator -o generated.cpp generated.config
    DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
)
```

- Можно использовать как обычные исходники

```
add_library(generated_lib generated.cpp)
target_link_libraries(generated_lib
    PUBLIC generator_runtime)
```

Custom commands

```
generated.cpp: generated.config
    generator -o generated.cpp generated.config
```

- Аналог в cmake?

```
add_custom_command(
    OUTPUT ${CMAKE_CURRENT_SOURCE_DIR}/generated.cpp
    COMMAND generator -o generated.cpp generated.config
    DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
)
```

- Можно использовать как обычные исходники

```
add_library(generated_lib generated.cpp)
target_link_libraries(generated_lib
    PUBLIC generator_runtime)
```

- Лучше не указывать OUTPUT файл в нескольких независимых target-ах одновременно

Custom target

```
add_custom_target(generated.cpp
    ALL
    COMMAND generator -o generated.cpp generated.config
    WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
    DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
    SOURCES ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
)
```

Custom target

```
add_custom_target(generated.cpp
    ALL
    COMMAND generator -o generated.cpp generated.config
    WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
    DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
    SOURCES ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
)
```

- Считаются всегда устаревшими

Custom target

```
add_custom_target(generated.cpp
    ALL
    COMMAND generator -o generated.cpp generated.config
    WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
    DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
    SOURCES ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
)
```

- Считаются всегда устаревшими
- Можно добавить зависимости с помощью `add_dependencies`

Custom target

```
add_custom_target(generated.cpp
    ALL
    COMMAND generator -o generated.cpp generated.config
    WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
    DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
    SOURCES ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
)
```

- Считаются всегда устаревшими
- Можно добавить зависимости с помощью `add_dependencies`
- Можно не включать в default target `ALL`

Custom target

```
add_custom_target(generated.cpp
    ALL
    COMMAND generator -o generated.cpp generated.config
    WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
    DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
    SOURCES ${CMAKE_CURRENT_SOURCE_DIR}/generated.config
)
```

- Считаются всегда устаревшими
- Можно добавить зависимости с помощью `add_dependencies`
- Можно не включать в default target `ALL`

```
set_target_properties(generated.cpp
    PROPERTIES EXCLUDE_FROM_ALL 1
)
```

Custom CMake modules

```
set(CMAKE_MODULE_PATH
    "${CMAKE_CURRENT_SOURCE_DIR}/CMake/Modules"
    ${CMAKE_MODULE_PATH})
```

More

- Generator expressions
- CPack
- CMake variables
- ???