ПОДВОДНЫЕ КАМНИ СМАКЕ

И ГДЕ ОНИ ОБИТАЮТ

Дмитрий Кожевников

ОБО МНЕ

- 7+ лет разработки на C++
  - большой кроссплатформенный CMake-проект
- сейчас - JetBrains, команда CLion
  - (пока что) понимает только CMake проекты

ЧТО ТАКОЕ CMAKE

- система генерации проектов
- система сборки
- инструмент для тестирования (CTest)
- инструмент для создания пакетов и инсталляторов (CPack)

ЗАЧЕМ?

- Windows/Linux/macOS
- clang/gcc/cl.exe
- Make/Ninja/VS solutions/Xcode projects/...
- Пакетные менеджеры
- IDE

ПРИМЕР

(НА САМОМ ДЕЛЕ НЕТ)

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
PROJECT(MyProject)

FIND_PACKAGE(Boost REQUIRED)
INCLUDE_DIRECTORIES(${Boost_INCLUDE_DIR})

ADD_EXECUTABLE(myapp myMain.cpp)

TARGET_LINK_LIBRARIES(myapp ${Boost_LIBRARIES})
```

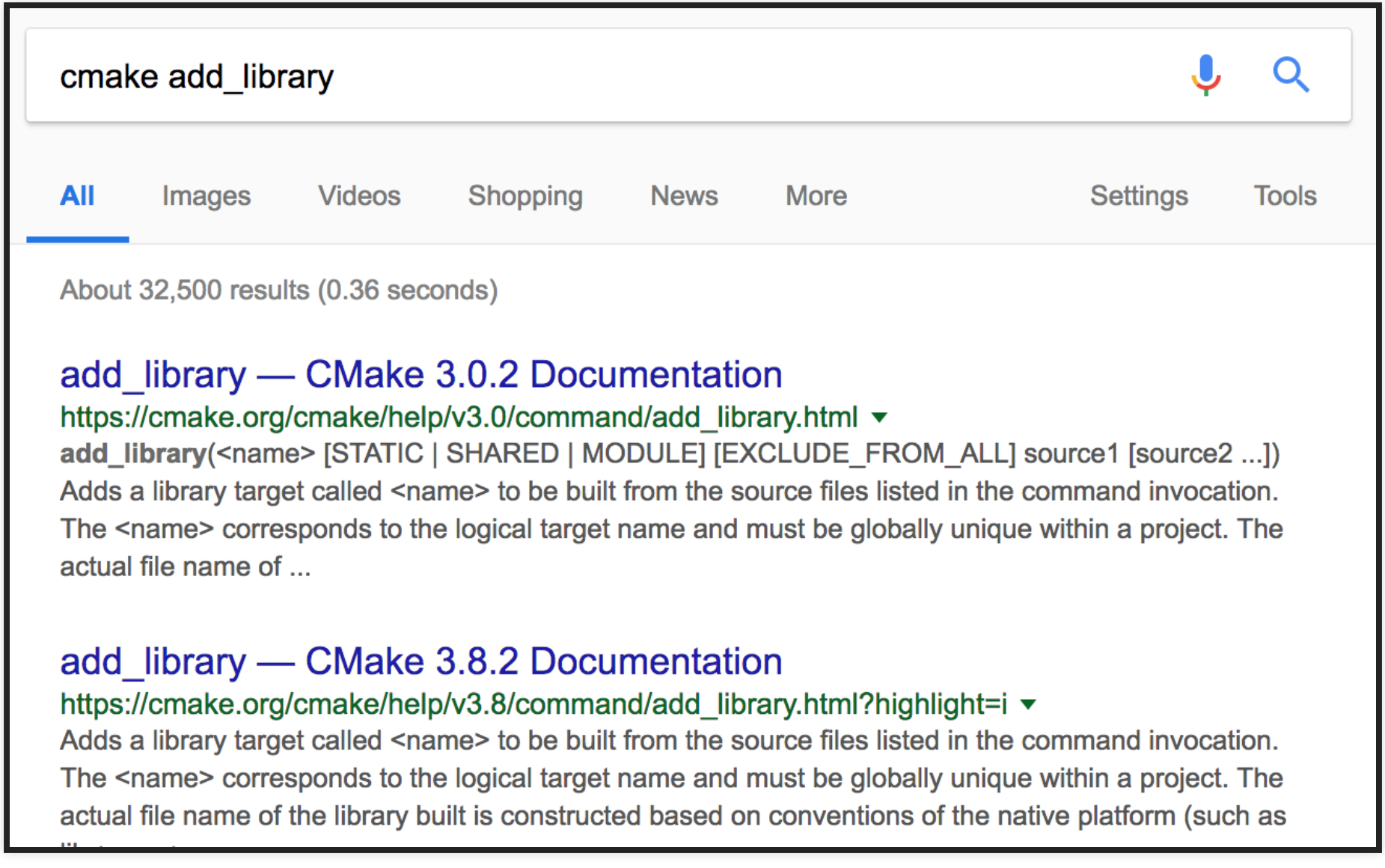
ПРОБЛЕМЫ

- Странный синтаксис
- Глобальное состояние
- Много способов решать одни и те же задачи
- Устаревшая информация/дезинформация

АЛЬТЕРНАТИВЫ?

- Более простые (Make)
- Более привычные (MSVC/Xcode/Autotools)
- Более мощные (Boost.Build, build2)
- Более современные (Meson)

ДЕЗИНФОРМАЦИЯ: GOOGLE



ДЕЗИНФОРМАЦИЯ: CMAKE_MINIMUM_REQUIRED

```
cmake_minimum_required(VERSION 2.8)
```

- Не запрещает использовать фичи из более новых версий
- Включает имитацию особенностей старых версий (CMake policies)

ДВОЙНОЕ РАСКРЫТИЕ ПЕРЕМЕННЫХ

```
cmake_minimum_required(VERSION 2.6)
project(xxx)

set(GNU 1)
# later
if ("${CMAKE_CXX_COMPILER_ID}" MATCHES "GNU")
    message("OK")
else()
    message("FAIL")
endif()
```

Выведет "FAIL" (с предупреждением) в новых версиях CMake

CMAKE_MINIMUM_REQUIRED (НА САМОМ ДЕЛЕ НЕТ)

```
cmake_minimum_required(VERSION 2.6)

foreach(x RANGE 10)
    # introduced in CMake 3.3
    continue()
endforeach()
```

Работает в новых версиях, не работает в старх версиях

ПЕРЕМЕННЫЕ

Переменные окружения:

```
$ VAR=1 cmake ..
```

```
message("VAR=$ENV{VAR}") # prints VAR=1
message("VAR=${VAR}") # prints VAR=
set(ENV{VAR} 2)
message("VAR=$ENV{VAR}") # prints VAR=2
```

Переменные CMake:

```
$ cmake .. -DVAR=1
```

```
message("VAR=${VAR}") # prints VAR=1
message("VAR=$ENV{VAR}") # prints VAR=
set(VAR 2)
```

ПЕРЕМЕННЫЕ CMAKE

```
$ cmake .. -DVAR="Yes"
```

```
set(VAR "No" CACHE TYPE STRING)
message("VAR=${VAR}") # prints VAR="Yes"
```

CMAKECACHE.TXT

- Файл в build-директории
- Двойной назначение:
  - Редактор настроек
    - `vi CMakeCache.txt`
    - `ccache`
    - CMake GUI
  - Кэш долгих операций при конфигурации
    - Общение с компилятором
    - Поиск библиотек

СМЕНА КОМПИЛЯТОРА

Переменные окружения:

```
$ CXX=g++-7 CC=gcc-7 cmake <...>
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.5)

set(CMAKE_C_COMPILER gcc-7)
set(CMAKE_CXX_COMPILER g++-7)

project(cmake_test)

# не здесь!
```

CMakeLists.txt:

```
$ cmake -DCMAKE_CXX_COMPILER=g++-7 ...
```

СМЕНА КОМПИЛЯТОРА: GOTCHA

```
cmake_minimum_required(VERSION 3.9)
project(sample-proj)
message("VAR=${VAR}")
# ...
```

```
$ cmake .. -DCMAKE_CXX_COMPILER=g++ -DVAR=1
VAR=1

$ cmake .. -DCMAKE_CXX_COMPILER=clang++ -DVAR=1
You have changed variables that require your cache to be delet
 Configure will be re-run and you may have to reset some varia
 The following variables have changed:
 CMAKE_C_COMPILER=g++
VAR=
```

Нужно перезапустить еще раз!

КРОСС-КОМПИЛЯЦИЯ

Вручную:

```
$ cmake \
    -DCMAKE_CXX_COMPILER=/path/to/g++ \
    -DCMAKE_SYSTEM_NAME=Linux \
    -DCMAKE_FIND_ROOT_PATH=... \
    ...
```

TOOLCHAIN FILE

```
set(CMAKE_CXX_COMPILER "/path/to/g++")
set(CMAKE_SYSTEM_NAME "Linux")
set(CMAKE_FIND_ROOT_PATH "...")
# ...
```

```
$ cmake -DCMAKE_TOOLCHAIN_FILE=/path/to/toolchain.cmake
```

Преимущества:

- Можно написать один раз и использовать в разных проектах
- Может быть написан автором toolchain-а

КАК ИСКАТЬ БИБЛИОТЕКИ

```
find_package(Xxx)
```

По умолчанию:

- Ищет `FindXxx.cmake`
  - Может поставляться с CMake (Boost, SDL, OpenSSL, ...)
  - Может находиться в `CMAKE_MODULE_PATH`
- Ищет `XxxConfig.cmake`
  - Обычно поставляется и устанавливается с библиотекой (Qt, ...)
- Исполняет найденный скрипт

FIND_PACKAGE: GOTCHA

- В CMakeCache.txt сохраняются результаты поиска библиотек
- Успешный поиск не повторяется заново
- При смене версии/путей результат может устареть
- Решение - удалить соответствующие переменные (или CMakeCache.txt целиком)

BOOST: GOTCHA

- Boost - не CMake-проект
- FindBoost.cmake - поддерживается community
- Перестает работать с каждым новым релизом Boost
- Решение: скопировать FindBoost.cmake из cmake master

ПРИМЕР

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
PROJECT(MyProject)

FIND_PACKAGE(Boost REQUIRED)
INCLUDE_DIRECTORIES(${Boost_INCLUDE_DIR})

ADD_EXECUTABLE(myapp myMain.cpp)

TARGET_LINK_LIBRARIES(myapp ${Boost_LIBRARIES})
```

- Как узнать имена переменных?
- Как понять, какие из них использовать?
- Что будет при опечатке?
- Как узнать все требования к использованию?

СОВРЕМЕННЫЙ CMAKE

- Не использовать глобальное состояние
- Не менять флаги компилятора напрямую
- Targets описывают требования к своей сборке через "target properties"
- "Target properties" транзитивно распространяются на зависимости

CMAKE PROPERTIES

Global:

```
include_directories(...)
set_property(GLOBAL INCLUDE_DIRECTORIES ...)
```

Target:

```
target_include_directories(foo ...)
set_property(TARGET foo INCLUDE_DIRECTORIES ...) # or
set_property(TARGET foo INTERFACE_INCLUDE_DIRECTORIES ...)
```

- PRIVATE: для собственной сборки
- INTERFACE: для сборки зависимостей
- PUBLIC: PRIVATE + INTERFACE

ПРИМЕР

```
cmake_minimum_required(VERSION 3.9)
project(MyProject)

find_package(Boost REQUIRED COMPONENTS filesystem)
add_executable(myapp myMain.cpp)
target_link_libraries(myapp PRIVATE Boost::filesystem)
```

TARGET_LINK_LIBRARIES

Флаг линковщика:

```
target_link_libraries(myapp -lfoo)
```

```
/usr/bin/c++  ... -lfoo
```

Имя библиотеки:

```
target_link_libraries(myapp foo)
```

```
/usr/bin/c++  ... -lfoo
```

Target:

```
target_link_libraries(myapp target)
```

```
/usr/bin/c++  ... <???>
```

MODERN CMAKE!

```
target_link_libraries(myapp PRIVATE Boost::filesystem)
```

"::" запрещает нежелательное поведение

НЕ МЕШАТЬ!

Плохо экспортировать properties, влияющие на сборку того, кто тебя использует

- C++ standard
- Warning level
- Компилятор

IMPORTED TARGET

Создать target с нужными свойствами

```
find_package(Foo) # old-style Find module

add_library(foo SHARED IMPORTED)
set_target_properties(foo PROPERTIES
    IMPORTED_LOCATION ${FOO_LIBRARIES}
    INTERFACE_INCLUDE_DIRECTORIES ${FOO_INCLUDE_DIR}
    INTERFACE_COMPILE_DEFINITIONS "ENABLE_FOO")
```

EXPORTED TARGET

- Автоматизация создания imported targets
- На этапе инсталляции генерирует код с предыдущего слайда
- CMake-файл распространяется с библиотекой

ОТЛАДКА

RTFM

ОТЛАДОЧНАЯ ПЕЧАТЬ: ПЕРЕМЕННЫЕ

```
message(...)
```

Вывести некоторые переменные:

```
include(CMakePrintHelpers)
cmake_print_variables(CMAKE_C_COMPILER CMAKE_CXX_COMPILER)
```

Вывести все переменные:

```
get_cmake_property(varNames VARIABLES)
include(CMakePrintHelpers)
cmake_print_variables(${varNames})
```

ОТЛАДОЧНАЯ ПЕЧАТЬ

Полезные переменные:

```
include(CMakePrintSystemInformation)
```

Переменные окружения:

```
execute_process(COMMAND "${CMAKE_COMMAND}" "-E" "environment")
```

Вывести все target properties:

https://stackoverflow.com/a/34292622

ОТЛАДОЧНЫЕ ФЛАГИ

Вывести переменные из CMakeCache.txt:

```
$ cmake -LAH
```

Больше сообщений:

```
$ cmake --debug-output
```

Stacktraces при ошибках:

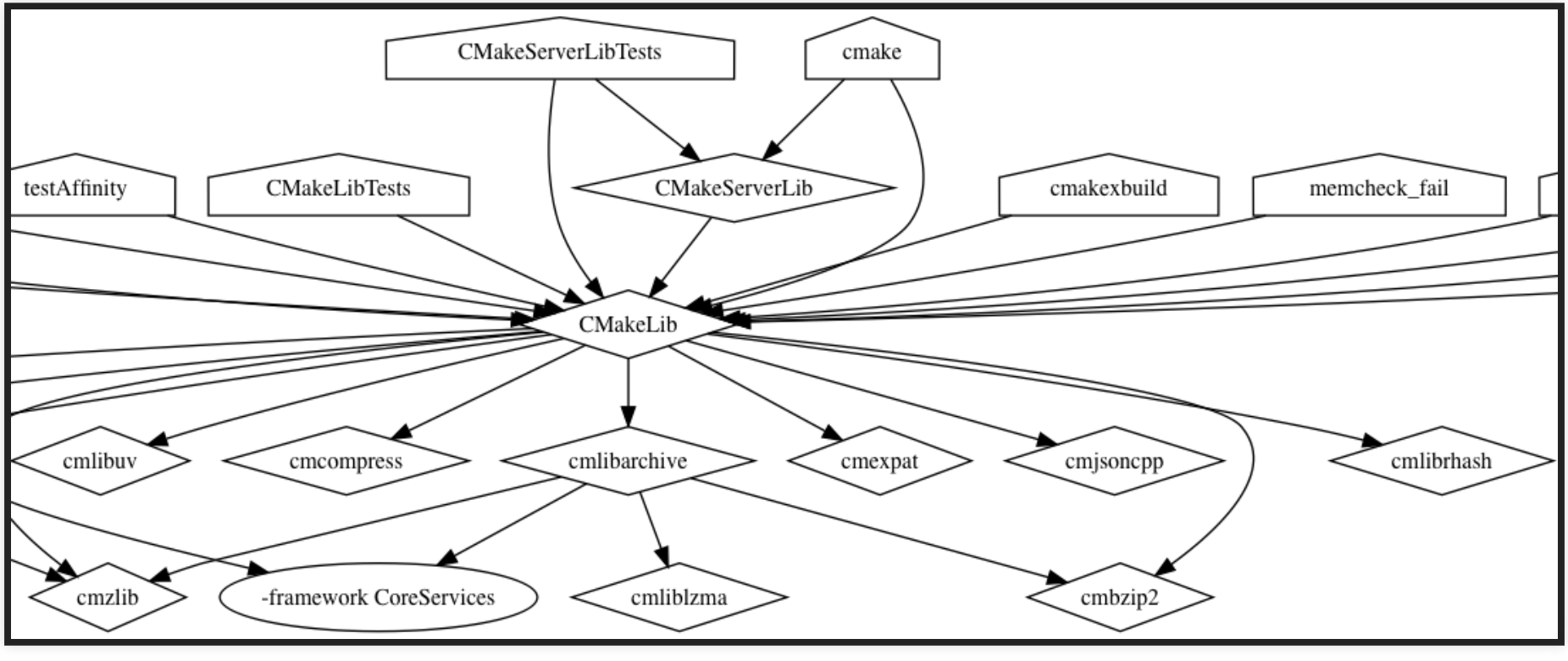
```
$ cmake --trace
```

Полезные предупреждения:

```
$ cmake --warn-unitialized --warn-unused-vars
```

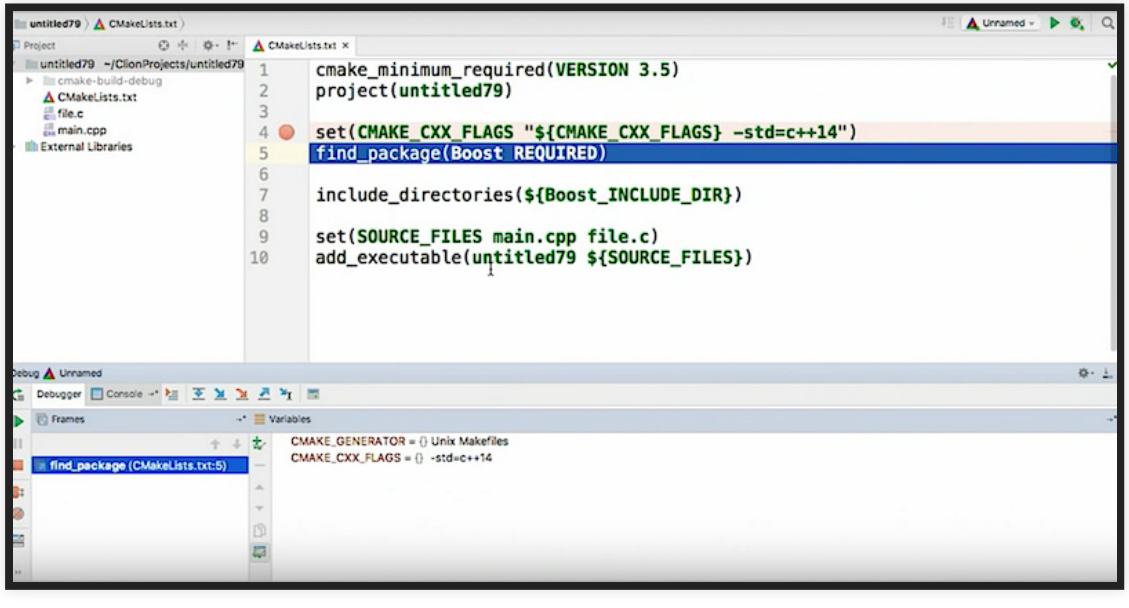
ГРАФ ЗАВИСИМОСТЕЙ

```
$ cmake --graphviz=[file] ...
```



ИНТЕРАКТИВНЫЙ ОТЛАДЧИК

- Justin Berger's fork
- В процессе разработки
- https://gitlab.kitware.com/jdavidberger



ИНФОРМАЦИЯ

- Daniel Pfeifer "Effective CMake" (C++Now 2017)
- Stephen Kelly "Embracing Modern CMake"
- Mathieu Ropert "Modern CMake for modular design" (CppCon 2017)
- Pablo Arias "It's Time To Do CMake Right"
- CppLang Slack #cmake

https://github.com/fastasturtle/cmake-talk/

СПАСИБО ЗА ВНИМАНИЕ!