Cmake

Содержание

CMake

CMake — это кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода.

CMake не занимается непосредственно сборкой, а лишь генерирует файлы управления сборкой из файлов CMakeLists.txt:

- Makefile в системах Unix для сборки с помощью make;
- файлы projects/solutions (.vcxproj/.vcproj/.sln) в Windows для сборки с помощью Visual C++;
- проекты XCode в Mac OS X.

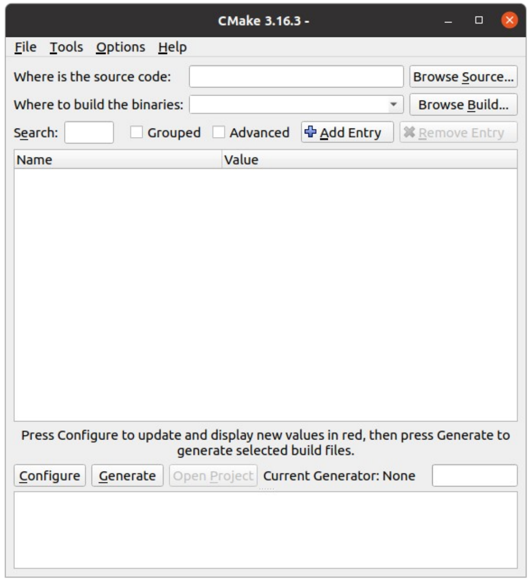
Установка

В Ubuntu вы можете установить командную строку и графическое приложение с помощью:

```
sudo apt install cmake
sudo apt install cmake-gui
```

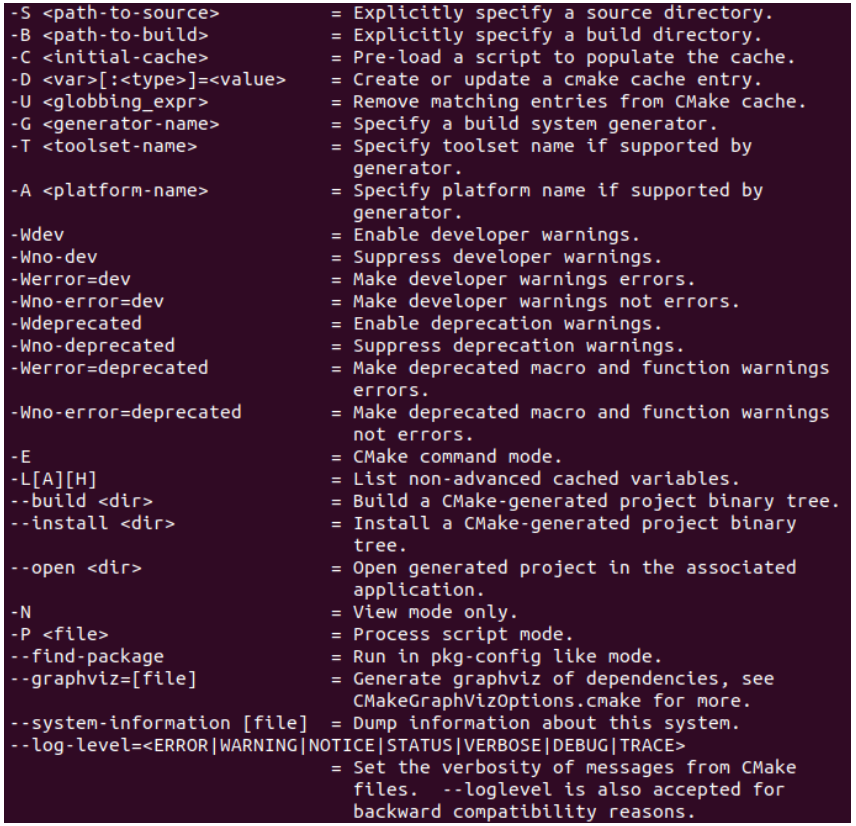
Узнать текущую версию:

```
cmake --version
```

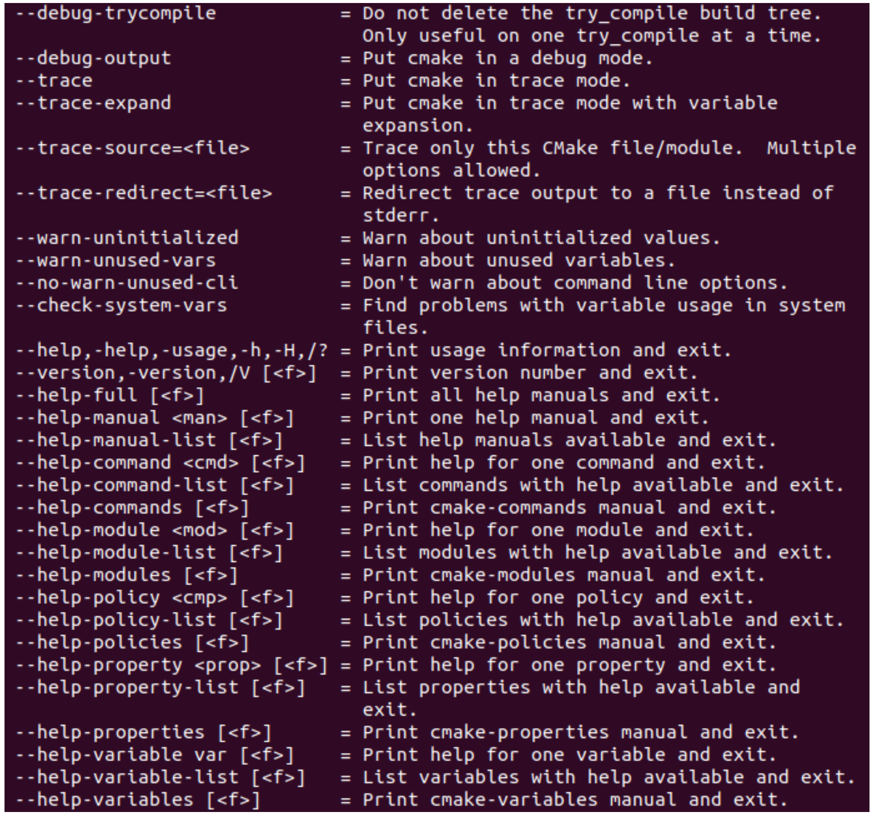


Press Configure to update and display new values in red, then press Generate to generate selected build files.

Опции CMake



```
-S <path-to-source>          = Explicitly specify a source directory.
-B <path-to-build>           = Explicitly specify a build directory.
-C <initial-cache>           = Pre-load a script to populate the cache.
-D <var>[:<type>]=<value>    = Create or update a cmake cache entry.
-U <globbing_expr>           = Remove matching entries from CMake cache.
-G <generator-name>          = Specify a build system generator.
-T <toolset-name>            = Specify toolset name if supported by
                               generator.
-A <platform-name>           = Specify platform name if supported by
                               generator.
-Wdev                        = Enable developer warnings.
-Wno-dev                     = Suppress developer warnings.
-Werror=dev                  = Make developer warnings errors.
-Wno-error=dev               = Make developer warnings not errors.
-Wdeprecated                 = Enable deprecation warnings.
-Wno-deprecated              = Suppress deprecation warnings.
-Werror=deprecated           = Make deprecated macro and function warnings
                               errors.
-Wno-error=deprecated        = Make deprecated macro and function warnings
                               not errors.
-E                           = CMake command mode.
-L[A][H]                     = List non-advanced cached variables.
--build <dir>                = Build a CMake-generated project binary tree.
--install <dir>              = Install a CMake-generated project binary
                               tree.
--open <dir>                 = Open generated project in the associated
                               application.
-N                           = View mode only.
-P <file>                    = Process script mode.
--find-package               = Run in pkg-config like mode.
--graphviz=[file]            = Generate graphviz of dependencies, see
                               CMakeGraphVizOptions.cmake for more.
--system-information [file]  = Dump information about this system.
--log-level=<ERROR|WARNING|NOTICE|STATUS|VERBOSE|DEBUG|TRACE>
                             = Set the verbosity of messages from CMake
                               files.  --loglevel is also accepted for
                               backward compatibility reasons.
```



```
--debug-trycompile           = Do not delete the try_compile build tree.
                               Only useful on one try_compile at a time.
--debug-output               = Put cmake in a debug mode.
--trace                      = Put cmake in trace mode.
--trace-expand               = Put cmake in trace mode with variable
                               expansion.
--trace-source=<file>        = Trace only this CMake file/module.  Multiple
                               options allowed.
--trace-redirect=<file>      = Redirect trace output to a file instead of
                               stderr.
--warn-uninitialized         = Warn about uninitialized values.
--warn-unused-vars           = Warn about unused variables.
--no-warn-unused-cli         = Don't warn about command line options.
--check-system-vars          = Find problems with variable usage in system
                               files.
--help,-help,-usage,-h,-H,/? = Print usage information and exit.
--version,-version,/V [<f>]  = Print version number and exit.
--help-full [<f>]            = Print all help manuals and exit.
--help-manual <man> [<f>]    = Print one help manual and exit.
--help-manual-list [<f>]     = List help manuals available and exit.
--help-command <cmd> [<f>]   = Print help for one command and exit.
--help-command-list [<f>]    = List commands with help available and exit.
--help-commands [<f>]        = Print cmake-commands manual and exit.
--help-module <mod> [<f>]    = Print help for one module and exit.
--help-module-list [<f>]     = List modules with help available and exit.
--help-modules [<f>]         = Print cmake-modules manual and exit.
--help-policy <cmp> [<f>]    = Print help for one policy and exit.
--help-policy-list [<f>]     = List policies with help available and exit.
--help-policies [<f>]        = Print cmake-policies manual and exit.
--help-property <prop> [<f>] = Print help for one property and exit.
--help-property-list [<f>]   = List properties with help available and
                               exit.
--help-properties [<f>]      = Print cmake-properties manual and exit.
--help-variable var [<f>]    = Print help for one variable and exit.
--help-variable-list [<f>]   = List variables with help available and exit.
--help-variables [<f>]       = Print cmake-variables manual and exit.
```

Стандартные опции сборки

-DCMAKE_BUILD_TYPE= Release, RelWithDebInfo, Debug, …

-DCMAKE_INSTALL_PREFIX= /usr/local, ~/.local

-DBUILD_SHARED_LIBS= ON, OFF

-DBUILD_TESTING=

```
cmake -G "Unix Makefiles" -DFORTE_ARCHITECTURE=Posix -
DFORTE_COM_ETH=ON -DFORTE_COM_FBDK=ON -DFORTE_COM_LOCAL=ON -
DFORTE_TESTS=OFF   -DFORTE_MODULE_CONVERT=ON -
DFORTE_MODULE_IEC61131=ON -DFORTE_MODULE_UTILS=ON -
DFORTE_COM_PAHOMQTT=ON  -DFORTE_COM_PAHOMQTT_LIB=libpaho-
mqtt3a.so -DFORTE_USE_LUATYPES="None" -
DFORTE_RTTI_AND_EXCEPTIONS=ON ../../
```

Hello world

main.cpp

```
#include <iostream>
int main()
{
    std::cout << "Hello World!\n";
    return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.4)
project(hello_world)
add_executable(app main.cpp)
```

Сборка

```
cmake .
cmake --build .   или make
```

“Чистая” сбокра

```
mkdir build
cd build
cmake ..
make
```

Новый стиль “чистой” сборки

```
cmake -S . -B build
cmake --build build
```

Hello World с несколькими исходными файлами

main.cpp

```
#include "foo.h"
int main()
{
    foo();
    return 0;
}
```

foo.cpp

```
#include <iostream>
#include "foo.h"
void foo()
{
    std::cout << "Hello World!\n";
}
```

foo.h

```
void foo();
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.4)
project(hello_world)
include_directories(${PROJECT_SOURCE_DIR})
add_executable(app main.cpp foo.cpp) # be sure
there's exactly one main() function in the source files
```

Hello World - библиотека

CMakeLists.txt

```
project(hello_world)
include_directories(${PROJECT_SOURCE_DIR})
add_library(applib foo.cpp)
add_executable(app main.cpp)
```

Libraries

```
add_library(my_lib lib.cpp)
add_library(my_shared_lib SHARED lib.cpp)      # Builds an shared library
add_library(my_static_lib STATIC lib.cpp)      # Builds an static library
```

include_directories

Структура проекта

```
include\
    myHeader.h
src\
    main.cpp
    CMakeLists.txt
```

CMakeList.txt

```
…
include_directories(${PROJECT_SOURCE_DIR}/include)
…
```

Переменные

Локальные переменые

```
set(MY_VARIABLE "value")
```

Получение значения переменной

```
${MY_VARIABLE}
```

Списки

```
set(MY_LIST "one" "two")
set(MY_LIST "one;two")
```

Переменные кэша

```
set(MY_CACHE_VARIABLE "VALUE" CACHE STRING "" FORCE)
mark_as_advanced(MY_CACHE_VARIABLE)
```

Переменные BOOL

```
option(MY_OPTION "This is settable from the command line" OFF)
```

Свойства

Свойства (Properties)

Установить

```
set_property(TARGET TargetName PROPERTY CXX_STANDARD 11)
set_target_properties(TargetName PROPERTIES CXX_STANDARD 11)
```

Получить

```
get_property(ResultVariable TARGET TargetName PROPERTY CXX_STANDARD)
```

Программирование в CMake

Оператор if

```
if(variable)
    # If variable is `ON`, `YES`, `TRUE`, `Y`, or non zero number
else()
    # If variable is `0`, `OFF`, `NO`, `FALSE`, `N`, `IGNORE`, `NOTFOUND`, `""`, or ends in `-
    NOTFOUND`
endif()
    # If variable does not expand to one of the above, CMake will expand it then try again
```

Программирование в CMake

Цикл foreach

```
set(MYLIST "a;b;c")
foreach(LETTER ${MYLIST})
    message("${LETTER}")
endforeach()


(1) foreach(LETTER a b c) [...]
(2) foreach(LETTER a;b;c) [...]
(3) set(MYLIST "a;b;c")
    foreach(LETTER ${MYLIST}) [...]
```

Программирование в CMake

Фунции

```
# Определение функции "print_numbers":
function(print_numbers NUM1 NUM2 NUM3)
    message(${NUM1} " " ${NUM2} " " ${NUM3})
endfunction()

# Определение макроса "print_words":
macro(print_words WORD1 WORD2 WORD3)
    message(${WORD1} " "  ${WORD2} " " ${WORD3})
endmacro()

# Вызов функции "print_numbers", которая напечатает "12 89 225":
print_numbers(12 89 225)

# Вызов макроса "print_words", который напечатает "Hey Hello Goodbye":
print_words(Hey Hello Goodbye)
```

Программирование в CMake

```
function(custom_function)
   # Вызвать механизм обработки аргументов для текущей функции:
    cmake_parse_arguments(CUSTOM_FUNCTION "LOW;HIGH" "NUMBER" "COLORS" ${ARGV})

    # Напечатает "'LOW' = [TRUE]":
    message("'LOW' = [${CUSTOM_FUNCTION_LOW}]")
    #Напечатает "'HIGH' = [FALSE]":
    message("'HIGH' = [${CUSTOM_FUNCTION_HIGH}]")
    # Напечатает "'NUMBER' = [30]":
    message("'NUMBER' = [${CUSTOM_FUNCTION_NUMBER}]")
    # Напечатает "'COLORS' = [red;green;blue]":
    message("'COLORS' = [${CUSTOM_FUNCTION_COLORS}]")
endfunction()

# Вызвать функцию "custom_function" с произвольными аргументами:
custom_function(LOW NUMBER 30 COLORS red green blue)
```

Отладка

Печать переменных

```
message(STATUS "MY_VARIABLE=${MY_VARIABLE}")
```

Встроенный модуль CMakePrintHelpers

```
include(CMakePrintHelpers)
cmake_print_variables(MY_VARIABLE)
cmake_print_properties(
    TARGETS my_target
    PROPERTIES POSITION_INDEPENDENT_CODE
)
```

Отладка

Трасировка

```
cmake -S . -B build --trace-source=CMakeLists.txt
--trace-expand
```

Сборка в режиме отладки

```
-DCMAKE_BUILD_TYPE=Debug
```

Как только вы сделаете сборку в режиме отладки, вы можете запустить на ней отладчик, такой как **gdb** или **lldb**

Поиск пакетов

Команда find_package находит и загружает настройки внешнего проекта.

Пример поиск GSL и последущая линковка

```
# Загрузить настройки пакета библиотеки "GSL":
Find_package(GSL 2.5 REQUIRED)
# Скомпоновать исполняемый файл с библиотекой "GSL":
target_link_libraries(MyExecutable GSL::gsl)
# Уведомить компилятор о каталоге заголовков "GSL":
target_include_directories(MyExecutable ${GSL_INCLUDE_DIRS})
```

Поиск пакетов

CUDA

```
find_package(CUDA 7.0 REQUIRED)
message(STATUS "Found CUDA ${CUDA_VERSION_STRING} at $
{CUDA_TOOLKIT_ROOT_DIR}")
```

OpenMP

```
find_package(CUDA)
if(OpenMP_CXX_FOUND)
    target_link_libraries(MyTarget PUBLIC OpenMP::OpenMP_CXX)
endif()
```

Список литературы

Learning cmake

https://en.wikipedia.org/wiki/CMake

https://cliutils.gitlab.io/modern-cmake/chapters/basics/functions.html

https://habr.com/ru/post/431428/

https://habr.com/ru/post/432096/