Building C/C++ Projects with (Modern) CMake

Powerful Cross Platform Build Process Management

Alexander Dahl

http://www.lespocky.de/

2018-10-22

Me

Yet another free software developer . . .

Background

- using Free Software since $\approx$ 2001
- contributing to Free Software since $\approx$ 2003
- diploma in engineering (mechatronics)
- working as Embedded Linux developer
- member of Netz39 Hackerspace

Me

Yet another free software developer . . .

Background

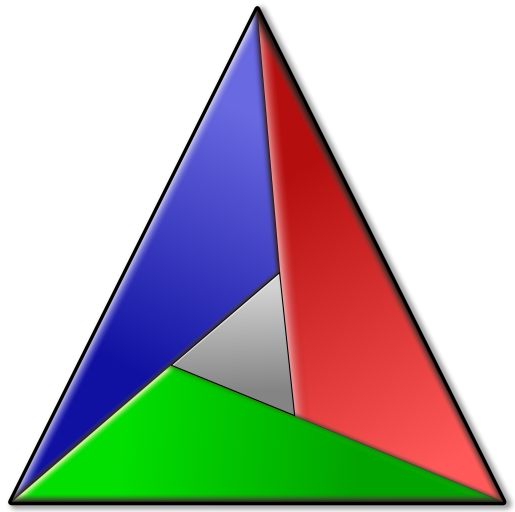
- using Free Software since $\approx$ 2001
- contributing to Free Software since $\approx$ 2003
- diploma in engineering (mechatronics)
- working as Embedded Linux developer
- member of Netz39 Hackerspace

Projects

- fli4l
- buildroot
- ptxdist
- libcgi
- Freifunk

CMake

- Build C/C++ projects
- Compiler independent
- Cross platform
- Together with native build environment
- Simple configuration with `CMakeLists.txt` files
- Out-of-source builds
- Free Software
- And more ...



```
cmake_minimum_required(VERSION 3.1)
project(MyProject
    VERSION 1.0
    DESCRIPTION "Very nice project"
    LANGUAGES CXX
)
```

Getting Started

Installation

- Linux: Use your package manager
- Windows, MacOS: Download from https://cmake.org/
- From Source with your favorite C++ Compiler

Getting Started

Installation

- Linux: Use your package manager
- Windows, MacOS: Download from https://cmake.org/
- From Source with your favorite C++ Compiler

Documentation

- CMake comes well documented
  - https://cmake.org/documentation
  - `man 7 cmake-*`
- on the world wide web
  - look out for "Modern CMake"
  - beware of examples showing old way to do things

Usage

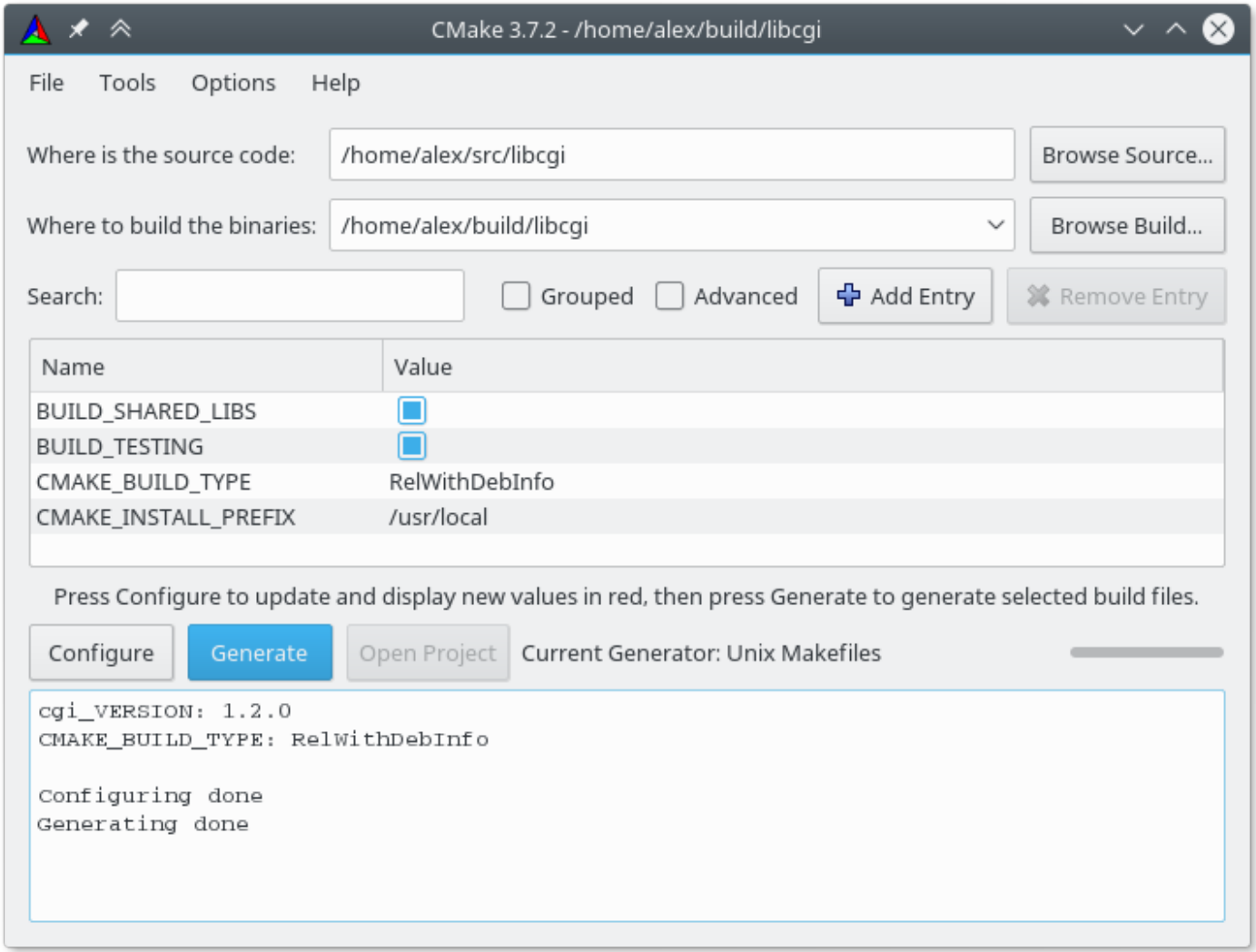
- Use out of tree builds

Command Line

```
~/path/to/your/src $ mkdir build
~/path/to/your/src $ cd build
~/path/to/your/src/build $ cmake ..
~/path/to/your/src/build $ make
```

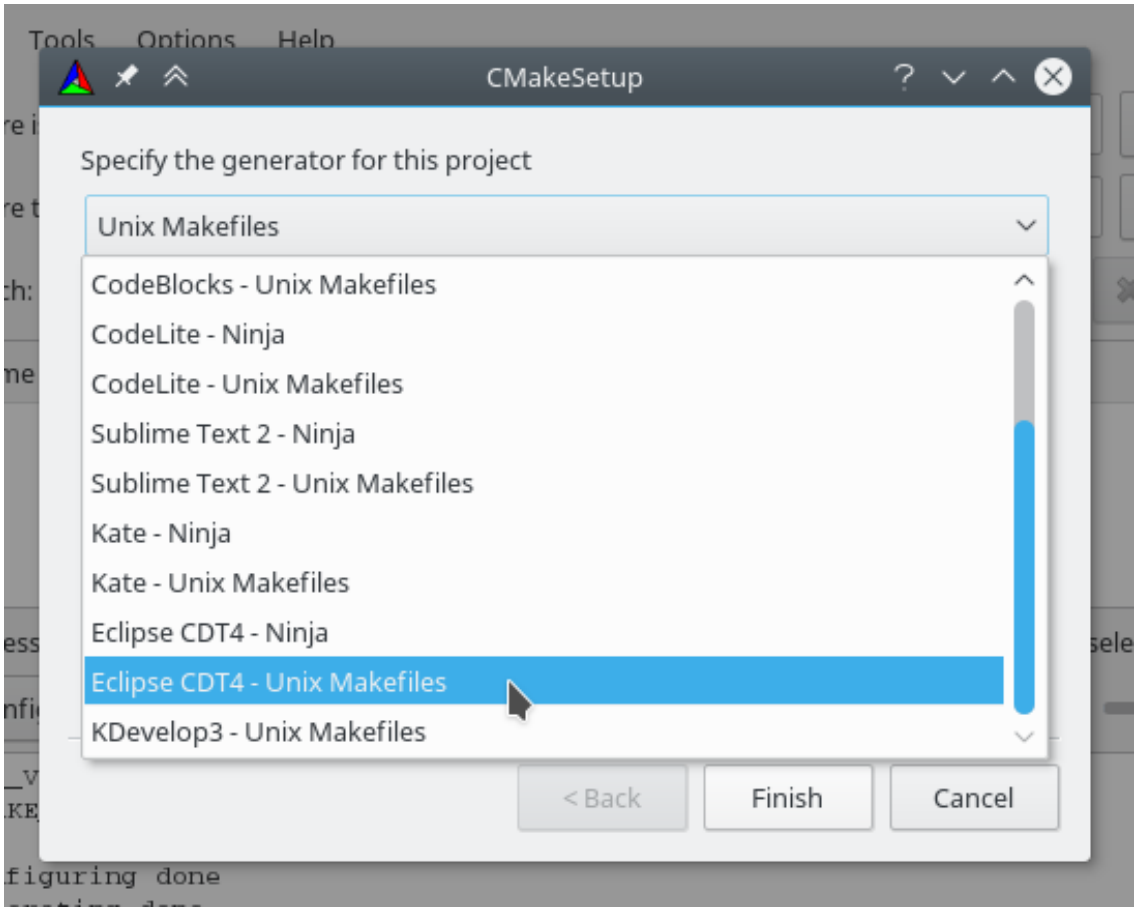
- first call to `cmake` is special, sets generator and compiler
- pre-set options with `-D`
  - you can overwrite options again later
  - useful when building from some external build system (like buildroot, ptxdist, ...)

GUI


CMake 3.7.2 - /home/alex/build/libcgi
File
Tools
Options
Help
Where is the source code:
/home/alex/src/libcgi
Browse Source...
Where to build the binaries:
/home/alex/build/libcgi
Browse Build...
Search:
Grouped
Advanced
Add Entry
Remove Entry
Name
Value
BUILD_SHARED_LIBS
BUILD_TESTING
CMAKE_BUILD_TYPE
RelWithDebInfo
CMAKE_INSTALL_PREFIX
/usr/local
Press Configure to update and display new values in red, then press Generate to generate selected build files.
Configure
Generate
Open Project
Current Generator: Unix Makefiles
cgi_VERSION: 1.2.0
CMAKE_BUILD_TYPE: RelWithDebInfo
Configuring done
Generating done

Generators

- Makefiles
  - Borland
  - MSYS
  - MinGW
  - NMake
  - Unix
  - ...
- Ninja
- Visual Studio
- Xcode
- ...

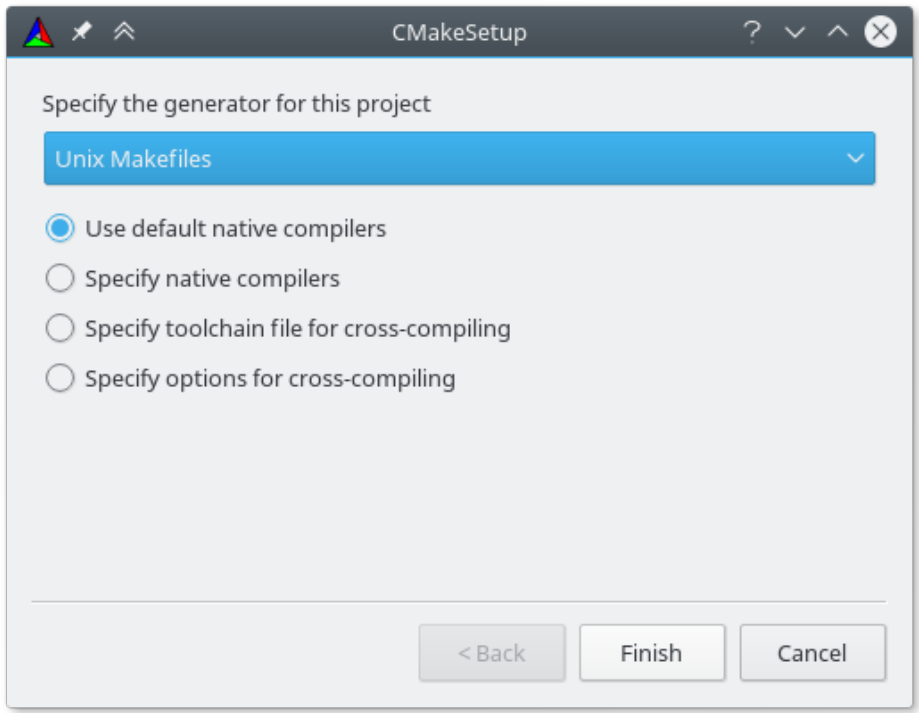


Help

cmake-generators(7)

Compilers

- AppleClang
- Clang
- GNU (GCC)
- MSVC (Visual Studio)
- SunPro
- Intel
- XL (IBM)
- ...



Help

cmake-compile-features(7)

Syntax

First Command

```
# Copyright 2018 Jon Doe
cmake_minimum_required(VERSION 3.1)
```

Second Command

```
project(foo
    VERSION 1.0
    DESCRIPTION "Very nice project"
    LANGUAGES CXX
)
```

- One or multiple files called `CMakeLists.txt`
- Commands are documented
- Syntax straight forward
  - Whitespace does not matter
  - Comments start with '#' and go to end of line
  - Variable expansion with `${VARNAME}`

Example Command

```
message(STATUS "foo_VERSION: ${foo_VERSION}")
```

Variables

- Set variables with `set()`
- Use `ALL_CAPS` for variable names
- Access variables with `${MY_VARIABLE}`
- Other commands may modify variables
- Beware of spaces, if not quoted, that can create lists
- Always enclose paths in spaces `"${MY_PATH}"`
- Scope is function or directory, not parent (by default)
- Variables can be cached across multiple runs
- Special global CMake variables exist

Help

cmake-language(7), cmake-variables(7)

Cache and Globals

Cache

- Cached variables can be listed and manipulated in GUI
- Command '`option()`' is syntactic sugar for cached boolean variables
- Cache is a text file in build directory

Cache and Globals

Cache

- Cached variables can be listed and manipulated in GUI
- Command '`option()`' is syntactic sugar for cached boolean variables
- Cache is a text file in build directory

Globals

- `CMAKE_BUILD_TYPE`
  - Debug, Release, RelWithDebInfo, ...
- `CMAKE_INSTALL_PREFIX`
  - defaults to `/usr/local`, like `prefix`
- `BUILD_SHARED_LIBS`

Targets

Modern CMake

Think in targets!

Targets

Modern CMake

Think in targets!

Simple Executable

```
add_executable(one
    two.cpp
    three.h
)
target_link_libraries(one
    two::two
)
```

- Target names
  - one
  - two::two

More Targets

Simple Library

```
add_library(two STATIC
    two.cpp
    three.h
)
add_library(two::two ALIAS two)
```

- Omit `STATIC` and let `BUILD_SHARED_LIBS` decide
- Use `INTERFACE` for header only libraries

Help

cmake-commands(7)

Add Build Information

Properties

```
set_target_properties(two PROPERTIES
    SOVERSION   ${PROJECT_VERSION_MAJOR}
    VERSION     ${PROJECT_VERSION}
    C_STANDARD  99
)
```

- See cmake-properties(7)

Add Build Information

Properties

```
set_target_properties(two PROPERTIES
    SOVERSION   ${PROJECT_VERSION_MAJOR}
    VERSION     ${PROJECT_VERSION}
    C_STANDARD  99
)
```

- See cmake-properties(7)

Include Directories

```
target_include_directories(two
    PUBLIC
        $<BUILD_INTERFACE:${PROJECT_SOURCE_DIR}/include>
        $<INSTALL_INTERFACE:${CMAKE_INSTALL_INCLUDEDIR}>
)
```

Glue It Together

The Most Powerful CMake Command

```
target_link_libraries(one
    PUBLIC
        two::two
    PRIVATE
        ${THREE_LIBRARIES}
)
```

Glue It Together

The Most Powerful CMake Command

```
target_link_libraries(one
    PUBLIC
        two::two
    PRIVATE
        ${THREE_LIBRARIES}
)
```

Link Items

- A library target name
- A full path to a library file
- A plain library name
- A link flag
- A generator expression

Chain Your Targets

- Use target names as link items
  - Use namespaces, to avoid accidentally wrong interpretation of link items
  - Use `PUBLIC`, `PRIVATE`, `INTERFACE` to not propagate non necessary dependencies
- Possible are all kinds of targets
  (even targets, which don't lead to actually calling a linker)
- CMake generates dependency tree
- Build is done in the right order automatically
- Use imported targets for external dependencies
- Export your own library targets (relocatable packages)

Antipatterns

Avoid functions with global scope!

- e.g. `link_directories()`, `include_libraries()`

Antipatterns

Avoid functions with global scope!

- e.g. `link_directories()`, `include_libraries()`

Do not set compiler flags in `CMakeLists.txt`

- Breaks portability between different compilers

Antipatterns

Avoid functions with global scope!

- e. g. `link_directories()`, `include_libraries()`

Do not set compiler flags in `CMakeLists.txt`

- Breaks portability between different compilers

Alternatives

- Require language standards (e. g. C99, C++11, . . .)
  - Meta compiler features (CMake 3.8+)
  - Compiler features (CMake 3.1+)
  - Per target properties (`CXX_STANDARD`, . . .)
- Manually override compiler flags from “outside” (e. g. CMake Cache)
- Make use of `CMAKE_BUILD_TYPE`

Best Practices

- Define everything per target
- Treat your `CMakeLists.txt` as code
  - Readable
  - Well documented
  - Put `CMakeLists.txt` in version control
- Think in targets
- Export your targets
- Make alias targets (aka namespaces for target names)
  - Make use of `add_subdirectory()` and `find_package()` consistent
  - Ensure `target_link_libraries()` uses a target
- `lower_case` function names
- `UPPER_CASE` variable names

Imported Targets

- CMake looks for CMake Packages with exported targets first
- Modern `find_package()` modules provide IMPORTED targets
  - e. g. `OpenSSL::SSL`, see cmake-modules(7)

```
find_package(OpenSSL)
target_link_libraries(two
    PUBLIC
        OpenSSL::Crypto
)
```

- Use `find_package(PkgConfig)` and `pkg_check_modules()` with option `IMPORTED_TARGET`
- Create IMPORTED targets with `add_library()` for your own `FindFoo.cmake` modules

Export Library Targets

```
include(GNUInstallDirs)                    # cmake 2.8.5

install(TARGETS two
    EXPORT two-targets
    LIBRARY DESTINATION "${CMAKE_INSTALL_LIBDIR}"
    ARCHIVE DESTINATION "${CMAKE_INSTALL_LIBDIR}"
)

install(EXPORT two-targets
    NAMESPACE two::
    DESTINATION "${CMAKE_INSTALL_LIBDIR}/cmake/two"
)
```

Help

cmake-packages(7)

Create (Relocatable) CMake Packages

```
include(CMakePackageConfigHelpers)  # cmake 2.8.8

configure_package_config_file(two-config.cmake.in
    "${CMAKE_CURRENT_BINARY_DIR}/two-config.cmake"
    INSTALL_DESTINATION "${CMAKE_INSTALL_LIBDIR}/cmake/two"
    PATH_VARS CMAKE_INSTALL_INCLUDEDIR
    NO_CHECK_REQUIRED_COMPONENTS_MACRO
)
write_basic_package_version_file(
    "${CMAKE_CURRENT_BINARY_DIR}/two-config-version.cmake"
    COMPATIBILITY SameMajorVersion
)
install(FILES
    "${CMAKE_CURRENT_BINARY_DIR}/two-config.cmake"
    "${CMAKE_CURRENT_BINARY_DIR}/two-config-version.cmake"
    DESTINATION "${CMAKE_INSTALL_LIBDIR}/cmake/two"
)
```

CMake Package Config File

```
@PACKAGE_INIT@

get_filename_component(two_CMAKE_DIR
    "${CMAKE_CURRENT_LIST_FILE}" PATH
)
include(CMakeFindDependencyMacro)

find_dependency(OpenSSL REQUIRED)

if(NOT TARGET two::two)
    include("${two_CMAKE_DIR}/two-targets.cmake")
endif()
```

Help

cmake-packages(7)

Generate Things at Build Time

`configure_file()`

- Create `version.h` from `version.h.in`
- Substitute placeholders with contents of CMake variables
- Set Preprocessor switches from CMake options with `#cmakedefine`
- `configure_package_config_file()` uses the same mechanism

Generate Things at Build Time

`configure_file()`

- Create `version.h` from `version.h.in`
- Substitute placeholders with contents of CMake variables
- Set Preprocessor switches from CMake options with `#cmakedefine`
- `configure_package_config_file()` uses the same mechanism

Custom Targets and Custom Commands

- `add_custom_command()` to call external tools generating things
- `add_custom_target()` for always to built targets
- Functions, macros, and programming directives

Related Tools

- CTest
  - Test driver for your unit tests
  - `enable_testing()` and `add_test()`
- CPack
  - Tarballs
  - Debian-Packages
  - Installers (may need external Tools)
  - ...
- CDash
  - Web-based software testing server
  - CTest can report to CDash
  - Runs valgrind, coverage tools, ...

The Last Slide

Contact Me

E-Mail post@lespocky.de

WWW lespocky.de or blog.antiblau.de

Twitter @LeSpocky

Slides

- `hg clone https://bitbucket.org/lespocky/talks`

License