

# CMake - Buildsystem und mehr

Wolfgang Dautermann

FH JOANNEUM

LIT Augsburg 2019

- 1 CMake - Buildsystem
  - Bauen von Software
  - Zusatzpakete/Libraries finden und verwenden
  - Installieren von Software
- 2 CPack - Paketieren von Software
- 3 CTest - Testen von Software
- 4 CDash - Webbasiertes Dashboard

## Was ist CMake?

### CMake-Eigenschaften

- Higher Level Build Tool  
(vergleichbar mit Autoconf/Automake, Scons, ...)
- Umfasst nicht nur Bau, sondern auch Testen, Paketieren, ...  
(Hauptthema dieses Vortrags)
- Cmake supported die nativen Build-Tools  
(Unix Makefiles, Visual-Studio, KDevelop3, CodeBlocks, Eclipse, ...)

# Wer verwendet CMake?

Bekannte Projekte mit CMake als Buildsystem

- KDE (ab Version 4)
- Mysql
- Scribus
- ...

## Hello-World Beispiel

Wir compilieren & installieren ein einfaches C-Programm

### CMakeLists.txt

```
PROJECT(helloworld)
# Kommentar zum Projekt helloworld
cmake_minimum_required(VERSION 3.5)
add_executable(helloworld helloworld.c)
install(TARGETS helloworld RUNTIME DESTINATION bin)
```

- Variablen case-sensitive
- CMake-Funktionen case-insensitive

## In-Source vs. Out-of-source Build

Wo kommen generierte Dateien (Objectfiles, Executables, ...) hin?

- In-Source: Sourcecode und generierte Dateien (Objectfiles, Executables,...) sind im selben Directory.  
Aufräumen (`make clean / make distclean`) notwendig.
- Out-of-Source: Build-Directory  $\neq$  Sourcecode-Directory
  - Von CMake supported.
  - (sehr!) empfohlen
  - Sourcecodedirectory wird nicht VERSCHMUTZT
  - Alle Dateien werden in einem separaten Build-Directory erzeugt.  
`make clean: rm -rf *` im Build-directory.
  - Verschiedene Builds (Debug, Release, 32Bit, 64Bit,...) gleichzeitig möglich.

## Hello-World Beispiel – Compilieren

Kommandozeile

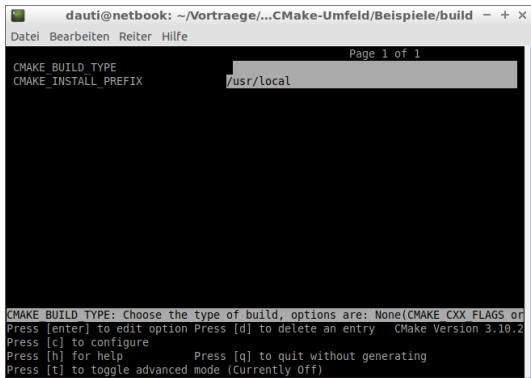
### Aufruf von cmake

```
~/build> cmake ../helloworld
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
[...]
-- Configuring done
-- Generating done
-- Build files have been written to: [...]
~/build> make [VERBOSE=1]
~/build> make install # als root
```

Installationspräfix angeben mit: `-DCMAKE_INSTALL_PREFIX:PATH=/my/path`

# Hello-World Beispiel – Compilieren

## Curses GUI



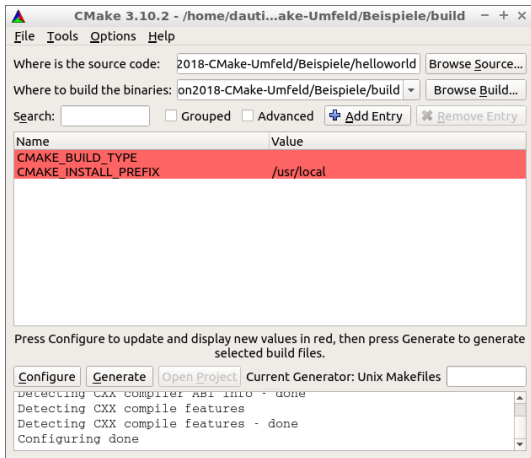
```
dauti@netbook: ~/Vortraege/...CMake-Umfeld/Beispiele/build - + x
Datei Bearbeiten Reiter Hilfe
Page 1 of 1
CMAKE_BUILD_TYPE
CMAKE_INSTALL_PREFIX /usr/local

CMAKE BUILD TYPE: Choose the type of build, options are: None(CMAKE CXX FLAGS or
Press [enter] to edit option Press [d] to delete an entry CMake Version 3.10.2
Press [c] to configure
Press [h] for help Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently off)
```



# Hello-World Beispiel – Compilieren

QT basierte GUI



Live Demo...

## CMake Syntax und Features

Variablen (Case-sensitive! X <> x)

### Variablen setzen

```
set (var wert)
set (var a.c b.c c.c) # var="a.c;b.c;c.c" (Liste!)
set (var "hello.c world.c") # var="hello.c world.c"
```

Variablen können beim cmake-Aufruf gesetzt werden:

```
cmake -Dvar=wert ...
```

### File globbing

```
file(GLOB helloworld_sources *.c )
```

## Compilieren: Programme und Libraries

### TARGETS hinzufügen

```
add_executable( <name> sourcefiles )
```

```
add_library( <name> [ (STATIC) | SHARED ] sourcefiles )
```

Den Namen ohne OS-spezifische Pre/Suffixes (<name>.exe, <name>.dll, lib<name>.so, lib<name>.a,...) angeben – wird automatisch ergänzt (und ist dadurch plattformunabhängig!)

## Pakete/Libraries finden

...ich mag nicht alles selber machen

### Pakete finden

```
find_package(<name> [REQUIRED])
```

Folgenden Variablen werden gesetzt:

- `<name>_FOUND` (falls die Suche erfolgreich war)
- `<name>_LIBRARIES1`, `<name>_INCLUDE_DIRS2` (bei Bibliotheken)
- `<name>_EXECUTABLE` (bei Programmen)

(Ev. auch noch weitere: `cmake --help-module Find<name>`)

---

<sup>1</sup>manchmal auch `<name>_LIBRARY` oder `<name>_LIBS`

<sup>2</sup>manchmal auch `<name>_INCLUDES` `<name>_INCLUDE_DIR`

## Pakete/Libraries verwenden

### Include-Pfad ergänzen

```
include_directories(${<name>_INCLUDE_DIRS})
```

### Bibliothek linken

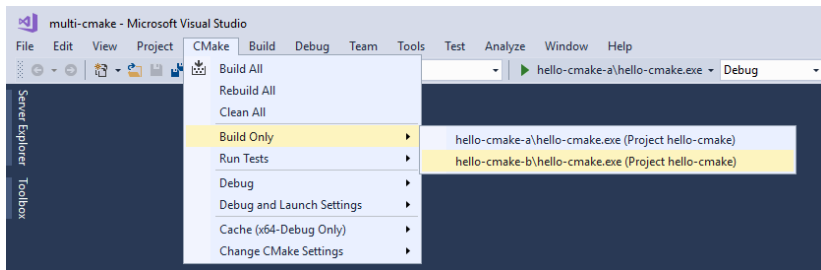
```
target_link_libraries(targetname ${<name>_LIBRARIES})  
link_libraries(${<name>_LIBRARIES})      # Alle targets
```

### Ev. Compilerdefinitionen ergänzen

```
add_definitions(${<name>_DEFINITIONS})
```

## Visual Studio

Credits: <https://blogs.msdn.microsoft.com/vcblog/2016/10/05/cmake-support-in-visual-studio/>



## Installationen

### Targets installieren

```
install(TARGETS myExe mySharedLib myStaticLib
        RUNTIME DESTINATION bin
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib/static)
```

### Files installieren

```
install(FILEs files... DESTINATION <dir>)
install(DIRECTORY dir DESTINATION <dir>)
```

# Cpack - Paketieren von Software

## Sourcecode

Erstellen von Sourcecode-Paketen.

### Welche Pakete sollen erstellt werden?

```
set(CPACK_SOURCE_GENERATOR "TGZ;TBZ2;ZIP;TZ;STGZ")
set(CPACK_SOURCE_IGNORE_FILES "/\\.svn/;/\\.git/;.*~")
set(CPACK_SOURCE_PACKAGE_FILE_NAME "helloworld-1.0")

include(CPack)
```

`make package_source` erstellt die Pakete. (Live Demo)



## Binärpakete: DEB, RPM, ZIP, TAR, ...

Metadaten festlegen (es gibt noch wesentlich mehr...)

```
set(CPACK_GENERATOR "TGZ;TBZ2;ZIP;DEB;RPM")
set(CPACK_PACKAGE_DESCRIPTION_SUMMARY
    "Description of Helloworld")
set(CPACK_PACKAGE_VENDOR "The Helloworld Team")
set(CPACK_PACKAGE_DESCRIPTION_FILE
    "${CMAKE_SOURCE_DIR}/readme.txt")
set(CPACK_RESOURCE_FILE_LICENSE
    "${CMAKE_SOURCE_DIR}/license.txt")
set(CPACK_PACKAGE_VERSION "1.0")
set(CPACK_PACKAGE_CONTACT
    "Helloworld Team <helloworldteam@example.org>")
set(CPACK_PACKAGE_SECTION "games")
INCLUDE(CPack)
```

**make package** erstellt die Pakete. (Live Demo)

## Binärpakete: RPM & SRPM

In CMake/CPack inkludiert ist das Bauen von (binary) RPM.  
Source (& Binary<sup>3</sup>) Pakete können mit UseRPMTTools gebaut werden.  
UseRPMTTools

Metadaten festlegen (es gibt noch wesentlich mehr...)

```
include(UseRPMTTools)
if(RPMTTools_FOUND)
    RPMTTools_ADD_RPM_TARGETS(helloworld)
endif(RPMTTools_FOUND)
```

`make helloworld_rpm / make helloworld_srpm` erstellt die Pakete. (Live Demo)

---

<sup>3</sup>Auf Debian/Ubuntu geht das Binary RPM nicht, weil CMake (als RPM-Paket!) installiert sein müsste

# Crosscompiling für Windows

...und Erstellen eines Installers unter Linux

## Compiler und BS festlegen (oft externe Toolchain)

```
set(CMAKE_SYSTEM_NAME Windows)

set(HOST i686-w64-mingw32)
set(CMAKE_C_COMPILER    ${HOST}-gcc)
set(CMAKE_CXX_COMPILER  ${HOST}-g++)
set(CMAKE_RC_COMPILER   ${HOST}-windres)

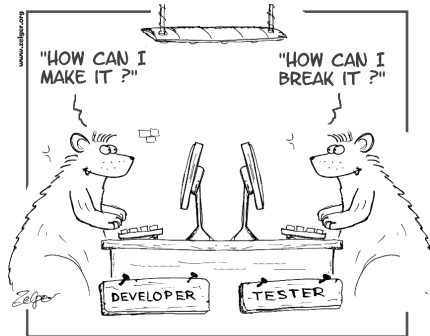
set(CPACK_SOURCE_GENERATOR "ZIP")
set(CPACK_GENERATOR "NSIS")
```

(Live Demo)

# Programmer vs. Tester

Credits:

<https://simply-the-test.blogspot.co.at/2010/04/different-goals.html>



They weren't so much different, but they had different goals

## CTest – Testen von Software

ermöglicht automatisierte Tests

### enable\_testing() und add\_test()

```
enable_testing()
add_test(Testname1
        ${CMAKE_BINARY_DIR}/<programm> [argumente])
add_test(Testname2
        ${CMAKE_SOURCE_DIR}/<script> [argumente])
```

Test ist gültig bei Exitcode == 0.

### Starten der Tests

```
make test
ctest [--verbose]
```

## CTest – Testen von Software

ermöglicht automatisierte Tests

### Regular Expressions / Timeout

```
set_tests_properties(Testname PROPERTIES  
                    PASS_REGULAR_EXPRESSION "Okay")  
set_tests_properties(Testname PROPERTIES  
                    FAIL_REGULAR_EXPRESSION "Failed")  
set_tests_properties(Testname PROPERTIES TIMEOUT "120")
```

## CDash

open-source web-based server for continuous integration



CMake										
Dashboard		Calendar	Previous	Current	Project					
No file changed as of <b>Saturday, October 29 2016 - 21:00 EDT</b>										
2 hours ago: 171 tests not run on OpenBSD 6.0-sparc64-gcc-4.9										
2 hours ago: 53 tests failed on OpenBSD 6.0-sparc64-gcc-4.9										
2 hours ago: 12 warnings introduced on OpenBSD 6.0-sparc64-gcc-4.9										
2 hours ago: 1 error introduced on OpenBSD 6.0-sparc64-gcc-4.9										
3 hours ago: 1 test failed on master-Win64Cygwin-gnu										
<a href="#">See full text</a>										
Style										
Site	Build Name	Update		Configure		Build		Test		
dashmaccini64.kitware	KWStyle	Files	Error	Warn	Error	Warn	Build Time			
		0	0	0	0	0	17 hours ago			
Scan Build										
Site	Build Name	Update		Configure		Build		Test		
elysium-linux.kitware	Linux-clang-scanbuild	Files	Error	Warn	Error	Warn	Build Time			
		0	0	0	0	0	15 hours ago			
Nightly Expected										
Site	Build Name	Update		Configure		Build		Test		
dash2win64.kitware	Jom-VS8	Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	Build Time
dash2win64.kitware	vs10-32-ninja	0	0	0	0	0	0	2	417	8 hours ago
hyfbioh.kitware	Linux-ninja-gcov	0	0	0	0	0	0	2	416	10 hours ago
pioneer-of-fec.de	Linux-Gentoo-HPPA32-4.6	0	0	0	0	0	0	1	407	11 hours ago
dash2win64.kitware	Win32-cygwin	0	0	0	0	0	50	1	435	16 hours ago
		0	0	0	0	0	0	1	426	10 hours ago

## CDash

Eintragen von CTest-Testergebnissen auf einen CDash-Server

### Öffentliche oder selbst gehostete Server

- Öffentlich (<http://my.cdash.org>)  
Free und gegen Bezahlung
- selbst gehostet: Software ist Open Source

### Erstellen einer CTestConfig.cmake (Download aus CDash)

```
set(CTEST_DROP_METHOD "http")  
set(CTEST_DROP_SITE "cdash.example.org")  
set(CTEST_DROP_LOCATION "/submit.php?project=name")  
set(CTEST_DROP_SITE_CDASH TRUE)
```



## Links und weiterführende Infos

- <http://www.cmake.org>
- <http://www.cmake.org/Wiki/CMake>

### inkludierte Hilfe

```
man cmake
cmake --help
      --help-full
      --help-command cmd
      --help-module module
      [...]
```

Fragen? Feedback?

Vielen Dank für Ihre Aufmerksamkeit

Wolfgang Dautermann

wolfgang.dautermann [AT] fh-joaanneum.at