Cmake

Michael Sammer, Christian Kreuzberger

February 13, 2019

Inhaltsverzeichnis

Was ist Cmake?

Cmake ist ein plattformunabhängiges Programmierwerkzeug für die Entwicklung und Erstellung von Software.

Was ist Cmake?

Cmake ist ein plattformunabhängiges Programmierwerkzeug für die Entwicklung und Erstellung von Software.
Cmake wurde im Jahr 2000 entwickelt und wird seit diesem Jahr immer wieder weiterentwickelt.

Was ist Cmake?

Cmake ist ein plattformunabhängiges Programmierwerkzeug für die Entwicklung und Erstellung von Software.
Cmake wurde im Jahr 2000 entwickelt und wird seit diesem Jahr immer wieder weiterentwickelt.
Gründer und Entwickler der Software sind Bill Hoffman, Ken Martin, Brad King, Dave Cole, Alexander Neundorf und Clinton Stimpson.

Was ist Cmake?

Cmake ist ein plattformunabhängiges Programmierwerkzeug für die Entwicklung und Erstellung von Software.
Cmake wurde im Jahr 2000 entwickelt und wird seit diesem Jahr immer wieder weiterentwickelt.
Gründer und Entwickler der Software sind Bill Hoffman, Ken Martin, Brad King, Dave Cole, Alexander Neundorf und Clinton Stimpson.
Das Programmierwerkzeug gibt es für Linux, MacOS und Windows.
Als Programmiersprache wird vermehrt auf C++ gesetzt.

Funktionen

Cmake erzeugt aus Skriptdateien (CMakeLists.txt) Makefiles und Projekte für viele integrierte Entwicklungsumgebungen und Compiler.

Funktionen

Cmake erzeugt aus Skriptdateien (CMakeLists.txt) Makefiles und Projekte für viele integrierte Entwicklungsumgebungen und Compiler.
Beispiele

- Eclipse

Funktionen

Cmake erzeugt aus Skriptdateien (CMakeLists.txt) Makefiles und Projekte für viele integrierte Entwicklungsumgebungen und Compiler.
Beispiele

- Eclipse
- Visual Studio

Funktionen

Cmake erzeugt aus Skriptdateien (CMakeLists.txt) Makefiles und Projekte für viele integrierte Entwicklungsumgebungen und Compiler.
Beispiele

- Eclipse
- Visual Studio
- xCode

Funktionen

Cmake erzeugt aus Skriptdateien (CMakeLists.txt) Makefiles und Projekte für viele integrierte Entwicklungsumgebungen und Compiler.
Beispiele

- Eclipse
- Visual Studio
- xCode
- ...

Funktionen

Cmake erzeugt aus Skriptdateien (CMakeLists.txt) Makefiles und Projekte für viele integrierte Entwicklungsumgebungen und Compiler.
Beispiele

- Eclipse
- Visual Studio
- xCode
- ...

Dabei werden automatisch die Abhängigkeiten für C, C++, Fortran und Java überprüft und parallele Builds unterstützt.

Funktionen

Zusätzlich werden einige Bibliotheken wie SWIG, Boost oder Qt unterstützt. Cmake hat zusätzlich seit Version 2.6 einige Tools zum Testen integriert.

Funktionen

Zusätzlich werden einige Bibliotheken wie SWIG, Boost oder Qt unterstützt. Cmake hat zusätzlich seit Version 2.6 einige Tools zum Testen integriert.

- DART

Funktionen

Zusätzlich werden einige Bibliotheken wie SWIG, Boost oder Qt unterstützt. Cmake hat zusätzlich seit Version 2.6 einige Tools zum Testen integriert.

- DART
- CDash

Funktionen

Zusätzlich werden einige Bibliotheken wie SWIG, Boost oder Qt unterstützt. Cmake hat zusätzlich seit Version 2.6 einige Tools zum Testen integriert.

- DART
- CDash
- CTest

Funktionen

Zusätzlich werden einige Bibliotheken wie SWIG, Boost oder Qt unterstützt. Cmake hat zusätzlich seit Version 2.6 einige Tools zum Testen integriert.

- DART
- CDash
- CTest
- CPack

Funktionen

Zusätzlich werden einige Bibliotheken wie SWIG, Boost oder Qt unterstützt. Cmake hat zusätzlich seit Version 2.6 einige Tools zum Testen integriert.

- DART
- CDash
- CTest
- CPack

Mit CPack ist es möglich einige bekannte Installationspakete wie Windows Installer (MSI) oder ZIP zu erstellen.

CmakeLists.txt

Die CMakeLists.txt ist die wichtigste Datei, die man benötigt um mit Cmake arbeiten zu können.
Sie muss im Rootordner des Projektes vorhanden sein, sowie in allen Unterordnern, die das Projekt umfasst.
Auf den nächsten Folien lernen wir einige wichtige Befehle kennen die in der CMakeLists.txt stehen können und zum Teil auch müssen sowie Befehle die zum ausführen von Cmake unerlässlich sind.

wichtige Befehle für Einsteiger!

wichtige Befehle für Einsteiger!

- PROJECT(PROJEKTNAME)

wichtige Befehle für Einsteiger!

- PROJECT(PROJEKTNAME)
- cmake_minimum_required (VERSION x.x.x)

wichtige Befehle für Einsteiger!

- PROJECT(PROJEKTNAME)
- cmake_minimum_required (VERSION x.x.x)
- set()

wichtige Befehle für Einsteiger!

- PROJECT(PROJEKTNAME)
- cmake_minimum_required (VERSION x.x.x)
- set()
- add_executable

wichtige Befehle für Einsteiger!

- PROJECT(PROJEKTNAME)
- cmake_minimum_required (VERSION x.x.x)
- set()
- add_executable
- cmake

wichtige Befehle für Einsteiger!

- PROJECT(PROJEKTNAME)
- cmake_minimum_required (VERSION x.x.x)
- set()
- add_executable
- cmake
- make

wichtige Befehle für Einsteiger!

- PROJECT(*PROJEKTNAME*)

Dieser Befehl ist einer der ersten Befehle in dem CMakeLists.txt. Durch diesen Befehl wird einzig und alleine der Projektname gesetzt.

wichtige Befehle für Einsteiger!

- cmake.minimum.required (VERSION x.x.x)

Mit diesem Befehl setzt man zu Beginn der CMakeLists.txt fest, welche Version von Cmake man verwendet.

wichtige Befehle für Einsteiger!

- set()

set() ist der Befehl mit dem man Variablen in der Datei instanziert. Dabei können verschiedene Werte oder auch Listen zugewiesen werden. Beim Beispiel wird man das später noch genauer sehen.

wichtige Befehle für Einsteiger!

- add.executable

Der Befehl add.executable macht ein Programm ausführbar.
Im Beispiel werden wir dann sehen wie die Datei hello.c zu einem ausführbaren Programm mit dem Namen HelloWorld wird.

wichtige Befehle für Einsteiger!

- cmake

Der Befehl cmake wird direkt in der Shell ausgeführt.
Mit diesem Befehl wird der Inhalt von CMakeLists.txt ausgeführt und alle Befehle in dieser Datei ausgeführt.

wichtige Befehle für Einsteiger!

- make

Der Befehl make ist die Kurzform von cmake.
Er macht im Grunde genau das gleiche wie cmake, doch er ist um einen Buchstaben kürzer.

Beispiele

- Hello World

Beispiele

- Hello World
- Libraries

Hello World

Im diesem Beispiel haben wir ein kleines Programm in C geschrieben.

```
#include <stdio.h>
int main()
{
   printf("Hallo Welt!n");
   return 0;
}
```

Hello World

Die CMakeLists.txt muss wie folgt aussehen.

```
cmake_minimum_required (VERSION 2.8.11)

project (HELLO)



# Der Befehl add_executable() macht aus der Quelldatei main.c ein ausführbares Programm
hallowelt.

add_executable(hallowelt main.c)
```

Hello World

nächste Schritte

- Shell öffnen

Hello World

nächste Schritte

- Shell öffnen
- in den Projektordner wechseln

Hello World

nächste Schritte

- Shell öffnen
- in den Projektordner wechseln
- den Befehl cmake . ausführen (Der Punkt heißt, dass cmake im aktuellen Ordner ausgeführt wird)

Hello World

nächste Schritte

- Shell öffnen
- in den Projektordner wechseln
- den Befehl cmake . ausführen (Der Punkt heißt, dass cmake im aktuellen Ordner ausgeführt wird)
- Enter drücken

Hello World

Cmake liest im Anschluss die CMakeLists.txt, sucht einen Compiler und erzeugt die Makefiles.

```
01 ~/src/hallo/ $ cmake .
02 -- Check for working C compiler: /usr/bin/gcc
03 -- Check for working C compiler: /usr/bin/gcc -- works
04 -- Check size of void*
05 -- Check size of void* - done
06 -- Check for working CXX compiler: /usr/bin/c++
07 -- Check for working CXX compiler: /usr/bin/c++ -- works
08 -- Configuring done
09 -- Generating done
10 -- Build files have been written to: ~/src/tests/hallo
```

Hello World

Im Anschluss an das Auslesen, Suchen und Erzeugen wird das Projekt übersetzt und ist fertig und ausführbar.

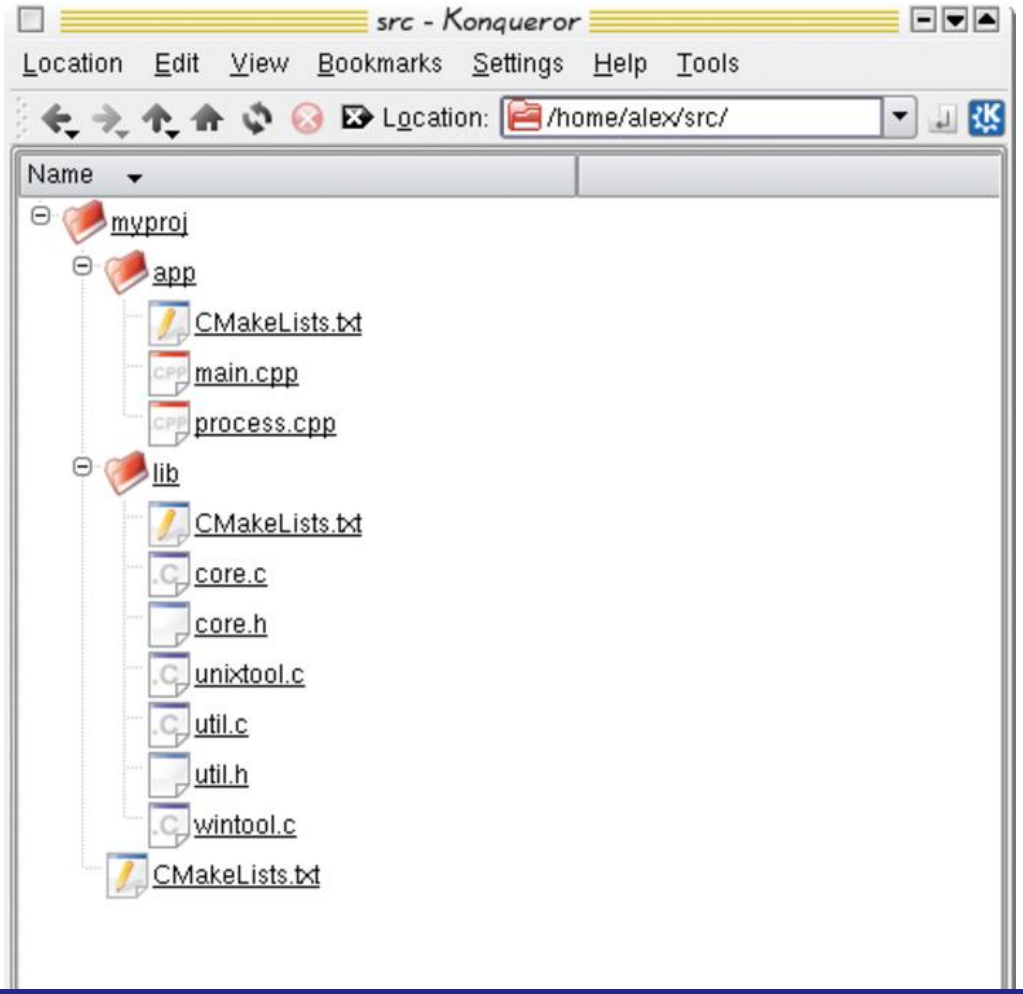
```
01 ~/src/hallo $ make
02 Scanning dependencies of target hallowelt
03 [100%] Building C object CMakeFiles/hallo.dir/main.o
04 Linking C executable hallowelt
05 [100%] Built target hallowelt
06 ~/src/hallo $ ./hallowelt
07 Hallo Welt!
```

Libraries

In diesemBeispiel befindet sich wenn alles fertig ist alles in einem Ordner.
Es besteht auch die Möglichkeit mehr Strutkur in die verschiedenen Projekte zu bringen.
Will man mehr Struktur haben, kann man das durch Libraries erreichen.
Damit befindet sich im Rootordner nur noch die CMakeLists.txt und die anderen Dateien in verschiedenen Unterordnern.

Libraries

Ein Beispiel für ein Projekt mit Libraries



Libraries

Die CmakeLists.txt im Rootordner.
In diesem Fall im Ordner myproj

```
cmake_minimum_required (VERSION 2.8.11)

project (HELLO)



add_subdirectory(lib)

add_subdirectory(app)
```

Libraries

Die CmakeLists.txt im Ordner lib.

```
cmake_minimum_required (VERSION 2.8.11)

project (HELLO)

set(libSrcs core.c util.c)

set(libSrcs ${libSrcs} unixtool.c)

add_library(util SHARED ${libSrcs})

install(TARGETS util DESTINATION lib)
```

Libraries

Die CmakeLists.txt im Ordner app.

```
cmake_minimum_required (VERSION 2.8.11)

project (HELLO)


set(fooappSrcs main.cpp process.cpp)

add_executable(fooapp ${fooappSrcs})

target_link_libraries(fooapp util)

install(TARGETS fooapp DESTINATION bin)
```

generierte Dateien

- CMakeCache.txt

generierte Dateien

- CMakeCache.txt
- Makefile

generierte Dateien

- CMakeCache.txt
- Makefile
- cmake_install_cmake

CMakeCache.txt

- Beinhaltet Daten vom Konfigurationslauf
- Eine Art Konfigurationsfile

Makefile

Die Datei Makefile ist das fertige Ergebnis von Cmake.
Diese Datei wird normalerweise manuell angelegt und Cmake macht diese Datei automatisch.

Zusammenfassung

- komplexes Tool
- für große Projekte gut geeignet
- kostenlos
- BSD-Lizenz

Quellen

- www.cmake.org
- http://www.linux-magazin.de/ausgaben/2007/02/mal-ausspannen/
- https://wolfgang.dautermann.at/vortraege/CLT2011-Cmake.pdf
- https://www.youtube.com/watch?v=hcTvIgdvOYY
- https://ebookcentral.proquest.com/lib/unisalzburg-ebooks/reader.action?docID=4822818&ppg=1

Danke!