Marco Schoos

CMake für Einsteiger

Wie Sie Ihren Build-Prozess konfigurieren und optimieren

©2022 Marco Schoos

ISBN Softcover: 978-3-347-61516-8
ISBN Hardcover: 978-3-347-61517-5

Druck und Distribution im Auftrag des Autors:
tredition GmbH, Halenreie 40-44, 22359 Hamburg, Germany

Das Werk, einschließlich seiner Teile, ist urheberrechtlich geschützt. Für die Inhalte ist der Autor verantwortlich. Jede Verwertung ist ohne seine Zustimmung unzulässig. Die Publikation und Verbreitung erfolgen im Auftrag des Autors, zu erreichen unter: tredition GmbH, Abteilung „Impressumservice“, Halenreie 40-44, 22359 Hamburg, Deutschland.

Vorwort

Inzwischen ist es einige Jahre her, doch zu der Zeit als ich begann CMake zu verwenden wollte ich keine Zeit damit „verschwenden“ mich genauer mit CMake zu befassen. Ich wollte nicht zu viel Zeit damit verbringen, meinen Build-Prozess zu konfigurieren, denn das war in meinen Augen nicht so wichtig, sondern wollte schnellstmöglich mein C++-Programm weiterentwickeln. Ich tat das, was viele Leute tun: Ich suchte im Internet nach dem Problem, das ich mit CMake lösen wollte und kopierte die Lösung ohne groß darüber nachzudenken in mein eigenes Projekt. Sicherlich nicht die beste Vorgehensweise, aber zu jenem Zeitpunkt, zumindest kurzfristig gesehen, die schnellste. Später begann ich dann einmal, mich eingehender mit CMake zu beschäftigen. Wie viele von Ihnen aber sicherlich wissen, ist das kein leichtes Unterfangen. Die CMake-Dokumentation ist zwar sehr umfangreich, aber eben nicht gerade für einen Einsteiger geschrieben. Inzwischen komme ich bestens mit ihr zu Recht, denn alle wesentlichen Inhalte werden dort beschrieben, aber für den Einstieg ist sie eben vollkommen ungeeignet. Auch außerhalb der CMake-Dokumentation ist es nicht so einfach mehr über CMake zu erfahren. Klar gibt es hier und ein paar Artikel und Videos, zum Beispiel auf YouTube, aber wirklich einfach gemacht wird einem die Sache nicht. Ich hoffe diesen Umstand zumindest im deutschsprachigen Raum durch die CMake-Videos auf meinem YouTube-Kanal „CodingWithMagga“ etwas verbessert zu haben. Auch in Sachen Literatur sieht es bei CMake eher dürftig aus. Wirklich empfehlen kann ich hier nur das englischsprachige Buch von Craig Scott „Professional CMake: A Practical Guide“[46]. Mein Buch ist meines Wissens nach das Erste, was überhaupt im deutschsprachigen Raum erschienen ist.

Nun halten Sie also das (vermeintlich) erste CMake-Buch in deutscher Sprache in Ihren Händen oder blicken auf einen Bildschirm mit der geöffneten Buchdatei. Dieses Buch deckt einen deutlich größeren Themenbereich ab als meine CMake-Videos und richtet sich dennoch weiterhin an Einsteiger, die ihre ersten Schritte mit CMake machen. Es

ist mit Sicherheit kein allumfassendes Werk, welches jeden Aspekt von CMake abdeckt, aber es wird Ihnen die grundlegenden Befehle und Eigenschaften von CMake vermitteln. Sie werden verstehen, wie CMake denkt und funktioniert und werden dabei erkennen, wie Sie gewisse Dinge in CMake umsetzen können. In CMake können Sie, wie in jeder anderen Programmiersprache auch, viele Dinge auf unterschiedliche Weise umsetzen. Dabei gibt es nicht immer unbedingt den einen richtigen Weg, aber häufig gibt es sehr viele „falsche“ Wege, die Sie lieber nicht beschreiten sollten. Nach der Lektüre dieses Buches werden Sie sicher nicht der größte CMake-Experte sein, aber Sie werden viel leichter erkennen können, wo die falschen Wege hinführen. Zudem werden Sie so weit in die Materie eingetaucht sein, dass Sie sich weiterführende CMake-Funktionalitäten, die in diesem Buch nicht behandelt werden, auch problemlos selbst, zum Beispiel durch Studium der CMake-Dokumentation, aneignen können.
Zwar handelt es sich hier natürlich um ein Buch in deutscher Sprache, allerdings werde ich an vielen Stellen englische Begriffe verwenden. Häufig gibt es einfach keine gute deutsche Übersetzung oder die englischen Begriffe sind bereits so fest verankert, dass die Verwendung des deutschen Begriffes nur zu Verwirrung führen würde. Um Ihnen das Leben in dieser Hinsicht zumindest etwas leichter zu machen, finden Sie am Ende des Buches ein Glossar, sowie einen Index. Allgemein wird sich ein gewisses hin und her blättern in diesem Buch nicht vermeiden lassen. Ich würde Ihnen daher empfehlen, sich den Code aus diesem Buch während der Lektüre auf Ihrem PC anzuschauen. Damit vermeiden Sie bereits einiges hin und her springen im Text und können zudem, was eigentlich noch deutlich wichtiger ist, den Code direkt selbst testen und verändern. Probieren Sie verschiedene Dinge aus und schauen Sie, wie CMake sich verhält. So lernen Sie CMake noch viel besser kennen, als es dieses Buch allein zu tun vermag. Den Code müssen Sie jedoch nicht jedes Mal abtippen, ich beschreibe in diesem Buch, wo und wie Sie sich den Code auf Ihren Computer laden können.
Gerne können Sie mir Fragen, Anregungen oder Fehler an marco@codingwithmagga.com schicken. Es würde mich freuen, von Ihnen zu hören. Schauen Sie sich auch gerne einmal auf meiner Webseite www.codingwithmagga.com um.

Inhaltsverzeichnis

- *Kapitel 1* -

Einleitung

Was ist CMake?

Im ersten Kapitel dieses Buches gebe ich erst einmal einen Überblick darüber, was CMake überhaupt ist und warum es in vielen Fällen von Vorteil ist, CMake zu verwenden. Eventuell werden Sie sogar überrascht sein zu sehen, wozu CMake alles in der Lage ist. Im zweiten Abschnitt dieses Kapitels werde ich Ihnen einige zusätzliche Informationen mitgeben, damit Sie dieses Buch optimal nutzen können. Diese nötigen Vorkenntnisse beziehen sich zum einen auf den Aufbau und Struktur dieses Buches, zum anderen geht es um die verwendeten Begrifflichkeiten und Notationen.

1.1. CMake – Ein Build-System-Generator

Um die grundlegende Aufgabe von CMake zu verstehen, ist es notwendig zu verstehen, was ein Build-System eigentlich ist. Ein Build-System ist ein Programm, welches aus Konfigurationsdateien den Compiler eines Programmierprojekts konfiguriert und entsprechend aufruft. Build-Systeme können eigenständige Kommandozeilenanwendungen wie Make, SCons und Ninja sein oder Teil einer integrierten Entwicklungsumgebung (IDE) wie Visual Studio, XCode oder IAR Embedded Workbench. Build-Systeme sind mächtige Werkzeuge und äußerst hilfreich in der Erstellung von Projekten, vor allem wenn diese an Größe und Komplexität zunehmen.
Nun ist CMake im Grunde genommen kein Build-System, sondern ein Build-System-Generator. Das bedeutet, CMake ist in der Lage, die Konfigurationsdateien, die von

Build-Systemen benötigt werden, zu generieren. Dabei unterstützt CMake eine große Menge an Build-Systemen auf allen möglichen Betriebssystemen. Dies ist wohl auch der größte Vorteil, den man aus der Verwendung von CMake ziehen kann. Jeder Entwickelnde kann selbst entscheiden, unter welchem Betriebssystem er welches Build-System und welchen Compiler verwenden möchte. CMake übernimmt die Aufgabe, die entsprechenden Konfigurationsdateien zu erzeugen. So ist es möglich, dass jede Person auf ihre bevorzugte Art und Weise arbeiten kann. Zudem ist es für die Qualität einer Software sicherlich von Vorteil, wenn die Software durch mehrere unterschiedliche Compiler erstellt werden kann. So werden eventuell Probleme gefunden, die durch einen bestimmten Compiler übersehen werden oder spezifische Implementierungen entdeckt, die nur von einem bestimmten Compiler verstanden werden.

CMake stellt für diese Aufgabe mehrere sogenannte Generatoren bereit. Man wählt den benötigten Generator für das eigene Build-System aus oder benutzt den standardmäßig verwendeten CMake-Generator[1] und CMake erstellt die entsprechenden Konfigurationsdateien. Es muss genau einer der CMake-Generatoren für ein CMake-Build ausgewählt werden, um festzulegen, welches Build-System verwendet werden soll. Auf jedem Betriebssystem stehen unterschiedliche CMake-Generatoren zur Verfügung. CMake kann eine entsprechende Liste der unterstützten Generatoren anzeigen, dies stelle ich genauer in Kapitel 2.2.2 vor.

Grundlage zur Erstellung der Konfigurationsdateien sind die in der CMake-Scriptsprache erstellten *CMake Lists.txt*-Dateien. Es existieren auch noch weitere von CMake verwendete Dateien, auf die ich jedoch erst an der entsprechenden Stelle im Buch zu sprechen komme. In diesem Buch wird es vorrangig um die Erstellung der *CMake Lists.txt*-Dateien gehen. Ich werde Ihnen zeigen, wie Sie ausführbare Dateien und Bibliotheken erstellen, wie Sie diese untereinander verlinken und wie Sie externen Code einbinden können und noch einiges mehr.

CMake kann aber nicht nur zur Generierung von Konfigurationsdateien für Build-Systeme verwendet werden, sondern stellt noch weitere sehr interessante Funktionalitäten bereit. Mit dem CMake-Programm CTest ist es möglich erstellte Tests auszuführen und die Ergebnisse dieser Tests auszugeben. Mehr dazu in Kapitel 10. Das CMake-Programm CPack kann CMake-Generatoren konfigurieren, die zur Erstellung von Installationsdateien oder Source Packages verwendet werden können. In Kapitel 9 gehe

[1] Unter Ubuntu ist das meist das Kommandozeilentool Make.

ich auf das Thema Packages in CMake ein, das Programm CPack selbst behandle ich in diesem Buch jedoch nicht.

1.2. Zusätzliche Informationen – Was Sie vor dem Lesen wissen sollten

Im Folgenden gebe ich Ihnen einen kurzen Überblick über die in diesem Buch verwendeten Konventionen und Notationen, damit Sie sich schnellstmöglich zurechtfinden. Bei der Lektüre dieses Buches werden Sie immer wieder auf graue Kästen, von mir im Folgenden als Listings bezeichnet, stoßen, in denen sich Code, aber auch Programmausgaben befinden können. Jedes dieser Listings ist entsprechend nummeriert und Sie finden hinten im Buch eine Auflistung aller Listings, geordnet nach ihrer jeweiligen Kategorie. Insgesamt gibt es vier unterschiedliche Kategorien von Listings, die jeweils Ihren eigenen Zweck erfüllen: Code, Codeschnipsel, C++-Code und Terminal. Ich gebe im Folgenden nun zu jedem einzelnen ein Beispiel und gehe auf die jeweiligen Besonderheiten dieser einzelnen Listing-Kategorien ein:

Code

Code 1.1: Beispiel für ein Code-Listing

```
1  cmake_minimum_required(VERSION 3.7)
2
3  project(
4    code_3-1
5    LANGUAGES CXX
6    )
7
8  add_executable(
9    hello_world
10   main.cpp
11   )
```

Code-Listings beinhalten stets ein voll funktionsfähiges CMake-Script, meist in Form einer *CMakeLists.txt*-Datei. Sie finden alle Code-Listings nach Kapiteln sortiert in einem GitHub-Repository, das Sie sich ganz einfach auf Ihren Computer herunterladen und damit arbeiten können. Genauere Informationen dazu finden Sie später in Kapitel 2.1.3. Alle CMake-Befehle innerhalb eines Listings sind fett gedruckt, wie zum Beispiel **cmake_minimum_required**() oder **project**(). CMake-Befehle werden immer kleingeschrieben. Zusätzlich sind alle Keywords, wie zum Beispiel **VERSION** oder **LANGUAGES**, ebenfalls fett gedruckt. Keywords, oder übersetzt Schlüsselwörter, sind feststehende Begriffe in CMake-Befehlen, die als Argumente mit übergeben werden. Keywords werden immer vollständig in Großbuchstaben geschrieben. Alle CMake-Befehle, sowie alle Keywords, die in diesem Buch vorkommen, sind in einem Index hinten im Buch verzeichnet. Später in Kapitel 4.1 lernen Sie noch CMake-Variablen und in Kapitel 4.2 sogenannte Properties kennen, die alle ebenfalls in diesem Index verzeichnet sind.

Codeschnipsel

Codeschnipsel 1.1: Beispiel für ein Codeschnipsel-Listing

```
1  project(
2    <ProjectName>
3    [VERSION <major>[.<minor>[.<patch>[.<tweak>]]]]
4    [DESCRIPTION <ProjektBeschreibung>]
5    [HOMEPAGE_URL <URL>]
6    [LANGUAGES <Programmiersprache1>...]
7    )
```

Codeschnipsel sind kleine Ausschnitte von CMake-Code, die für sich allein genommen nicht lauffähig sind. Meist nutze ich Codeschnipsel um einen CMake-Befehl[2] oder eine Funktionsweise in CMake zu erläutern. Wenn Sie in einem Listing oder im Fließtext einen Begriff in spitzen Klammern geschrieben sehen, wie zum Beispiel <ProjectName>, so stellt dies lediglich einen Platzhalter dar. Häufig können Sie an dessen Stelle eigene

[2] Im Prinzip sind CMake-Befehle nichts anderes als Funktionen. Ich verwende jedoch hier den Begriff Befehl, um von CMake bereitgestellte Befehle von selbst geschriebenen Funktionen (siehe Kapitel 4.5) und C++-Funktionen eindeutig abzugrenzen.

Namen wählen, wie etwa einen Namen für das Projekt anstelle des Platzhalters <Project-Name>. In manchen Fällen gibt es Restriktionen für Platzhalter, beispielsweise müssen die Platzhalter <major>, <minor>, <patch> und <tweak> im obigen Codeschnipsel Zahlen sein. Sollte es bestimmte Restriktionen geben, so werde ich dies im Text entsprechend anmerken. Steht ein Teil des Codes in eckigen Klammern, wie zum Beispiel [**HOMEPAGE_URL** <Url>], so ist dieses Argument optional und muss in dem entsprechen CMake-Befehl nicht verwendet werden. Das Ganze gilt auch bei mehreren eckige Klammern, wie bei [**VERSION** <major>[.<minor>[.<patch>[.<tweak>]]]]. Das Keyword **VERSION** muss nicht verwendet werden. Wenn es jedoch verwendet wird, so muss auch die Versionsangabe <major> folgen, die anderen jedoch nicht. Diese können jeweils einzeln folgen, sodass die folgenden Versionsangaben alle zulässig sind:

- <major>
- <major>.<minor>
- <major>.<minor>.<patch>
- <major>.<minor>.<patch>.<tweak>

Die Verwendung dreier Punkte ... deutet an, dass weitere Argumente der gleichen Art möglich sind. So können auf das Keyword **LANGUAGES** mehrere Programmiersprachen <Programmiersprache1>, <Programmiersprache2>, <Programmiersprache3> usw. folgen. Alle weiteren Argumente nach <Programmiersprache1> werden durch die drei Punkte ... symbolisiert. Weiterhin gibt es die Möglichkeit, dass Keywords durch das „oder“ Symbol | getrennt sind, wie im folgenden Beispiel:

Codeschnipsel 1.2: Weitere Beispiel für ein Codeschnipsel-Listing

```
1 add_library(
2   <BibliotheksName>
3   [STATIC|SHARED|MODULE]
4   [EXCLUDE_FROM_ALL]
5   [<SourceDatei1> <SourceDatei2> ...]
6   )
```

Im Befehl **add_library**() kann nur eines der Keywords **STATIC**, **SHARED** oder **MODULE** verwendet werden, was durch das Symbol | verdeutlicht wird. Es darf auch keins

dieser Keywords verwendet werden, wie es durch die eckigen Klammern verdeutlicht wird.

C++-Code

C++-Code 1.1: Beispiel für ein C++-Code-Listing

```
1 #include <iostream>
2
3 int main()
4 {
5   std::cout << "Hello World!" << std::endl;
6   return 0;
7 }
```

Die Beispiele, die ich mit den gezeigten CMake-Scripten kompiliere, sind stets in der Programmiersprache C++ geschrieben. CMake unterstützt aber auch weitere Programmiersprachen, worauf ich in Kapitel 3.1 noch einmal genauer eingehen werde. Da ich ganz allgemein die Verwendung von CMake in diesem Buch aufzeigen möchte, versuche ich so selten wie möglich auf den zugehörigen C++-Code einzugehen. Manchmal ist dies zum besseren Verständnis jedoch notwendig, sodass ich in diesem Falle auf diese C++-Code-Listings zurückgreife. Den Code aus allen C++-Code-Listings finden Sie auch auf GitHub. Sie sollten beachten, dass der verwendete C++-Code nur als Beispiel dient und somit so geschrieben ist, dass er maximal verständlich ist. Der Fokus in diesem Buch liegt natürlich auf CMake respektive den CMake-Scripten.

Terminal

Terminal 1.1: Beispiel für ein Terminal Listing

```
 1 $ cmake -G Ninja ..
 2 -- The CXX compiler identification is GNU 9.3.0
...  ...
11 -- Build files have been written to:
      /<PATH_TO_PROJECT>/build
12
```

```
13 $ cmake --build .
14 Scanning dependencies of target hello_world
15 [ 50%] Building CXX object
       CMakeFiles/hello_world.dir/main.cpp.o
16 [100%] Linking CXX executable hello_world
17 [100%] Built target hello_world
```

Ich führe alle Aufrufe von CMake, Kompilierungen und Ausführungen in einem Terminal auf Ubuntu 20.04 aus und verwende für die Ausgabe diese Terminal-Listings. Jedoch wurden (fast) alle CMake-Scripte auch auf den Betriebssystemen Windows und macOS getestet. Mehr Informationen dazu in Kapitel 2.1.3. Häufig kürze ich längere Ausgaben mit drei Punkten ab ..., oder verwende die bereits oben erwähnten Platzhalter, wie <PATH_TO_PROJECT> in Zeile 11.

In manchen Fällen sind Zeilen zu lang, um Sie in einem Listing in diesem Buch vollständig abdrucken zu können. Achten Sie daher auf die Zeilennummern auf der linken Seite. Wenn diese dort fehlt, so gehört die entsprechende Zeile zur vorherigen Zeile dazu. Zum Beispiel ist dies im obigen Terminal 1.1 in den Zeilen 11 und 15 der Fall.
Wie Sie sicher bereits bemerkt haben, sind Begriffe aus diesen Listings im Fließtext immer grau hinterlegt. Zusätzlich sind alle Dateien und Ordner wie *CMake Lists.txt*, *build/* oder *main.cpp* ebenfalls grau hinterlegt und zusätzlich zur besseren Unterscheidung kursiv gedruckt. An manchen Stellen verwende ich ein Sternsymbol * als Platzhalter, zum Beispiel *.exe oder *_FOUND. Der Platzhalter steht dann entweder für einen beliebigen Namen oder einen von mehreren Namen, der sich aus dem entsprechenden Kontext ergibt.[3]
Ich werde in diesem Buch immer wieder auf die CMake-Dokumentation und andere Webseiten und Bücher für weiterführende Themen verweisen. Dazu verwende ich die folgende Notation „[x]“, wobei das „x“ für eine Zahl steht. Anhand dieser Zahl können Sie den entsprechenden Link im Literaturverzeichnis nachsehen. Sie finden alle Links zusätzlich noch einmal auf meiner Internetseite www.codingwithmagga.com und auf der GitHub Seite des Codes (siehe Kapitel 2.1.3). So sparen Sie sich das abtippen. Im

[3] Gibt es etwa die CMake-Variablen BOOST_FOUND und OPENCV_FOUND, so können beide an der Stelle von *_FOUND stehen.

Falle der CMake-Dokumentation verweisen alle Links auf die aktuellste CMake-Version. Möchten Sie sich die Dokumentation zu einer bestimmten CMake-Version ansehen, findet sich auf der Webseite ein Dropdown Menü in der linken oberen Ecke zur Auswahl der gewünschten Version.
Das Buch ist so aufgebaut, dass Sie bis einschließlich Kapitel 5 der Kapitelstruktur des Buches folgen sollten. Anschließend sollten Sie ein solides Grundwissen haben, um die nächsten Kapitel in beliebiger Reihenfolge anzugehen. Sollten einmal Themen aus einem anderen Kapitel verwendet werden, so wird dies entsprechend deutlich gemacht. Am Ende der meisten Kapitel respektive Unterkapitel finden Sie einen Merkkasten wie den auf der nächsten Seite, der die wichtigsten Informationen noch einmal zusammenfasst. Sind Sie bereits mit den Themen eines Kapitels im Groben vertraut, so mag es unter Umständen auch ausreichend sein, wenn Sie sich nur die Merkkästen der jeweiligen Kapitel ansehen. Auch um später noch einmal eine kurze Auffrischung der jeweiligen Kapitel zu erhalten, kann das Lesen dieser Merkkästen hilfreich sein.

Merkkasten

- CMake ist ein Build-System-Generator, der sehr viele Betriebssysteme, Build-Systeme und Compiler unterstützt.
- Zur Generierung der Konfigurationsdateien stellt CMake sogenannte Generatoren zur Verfügung.
- Grundlage zur Erstellung der Konfigurationsdateien sind die in der CMake-Scriptsprache erstellten *CMake Lists.txt*-Dateien.
- Es gibt die vier folgenden Kategorien von Listings in diesem Buch: Code, Codeschnipsel, C++-Code und Terminal. Jede Listing-Kategorie deckt einen anderen Bereich des hier im Buch vorgestellten Codes respektive Programmausgaben dar.
- CMake-Befehle werden immer klein und Keywords immer großgeschrieben. Beide Wortgruppen werden zur besseren Darstellung fett gedruckt.
- Wörter in spitzen Klammern (<ProjectName>) sind Platzhalter, Wörter in eckigen Klammern sind optional und drei Punkte ... geben an, dass noch weitere Argumente der gleichen Art folgen können.
- Bis einschließlich Kapitel 5 sollten Sie der Kapitelstruktur dieses Buches folgen, anschließend können Sie sich relativ frei durch dieses Buch durcharbeiten.

- Kapitel 2 -

Grundlagen

Bevor es richtig los geht

In diesem Kapitel gehe ich zunächst auf ein paar Grundlagen ein, bevor ich auf die Syntax der CMake-Scriptsprache zu sprechen komme. Zunächst zeige ich Ihnen, wie Sie Ihr System einrichten sollten, um die in diesem Buch dargestellten Beispiele auch auf Ihrem Computer ausführen zu können. Dabei gehe ich auf verschiedene Betriebssysteme (Windows, Mac, Linux) ein. Außerdem erkläre ich Ihnen, wie Sie Zugriff auf die Codebeispiele aus diesem Buch erhalten. Anschließend widme ich mich dem Ablauf des CMake-Build-Prozesses. Ich zeige Ihnen wie aus etwas C++-Code und einem CMake-Script eine ausführbare Datei erzeugt werden kann und wie Sie diese ausführen können.

2.1. Einrichtung des Systems – Installation der wichtigsten Komponenten

Falls Sie bereits CMake in einer relativ aktuellen Version (mind. 3.5)[1] und einen entsprechenden C++-Compiler auf Ihrem Rechner installiert haben, können Sie die Abschnitte 2.1.1 und 2.1.2 überspringen. Für diejenigen, die Ihren Rechner gerade erst

[1] Noch besser wäre mindestens Version 3.7, die ich als Mindestvoraussetzung für die meisten Beispielprogramme nutze. Einige Codebeispiele benötigen auch eine höhere CMake-Version bis maximal 3.17.

einrichten, gehe ich kurz darauf ein, wie Sie die benötigte Software für dieses Buch installieren.

2.1.1. CMake

Es gibt viele verschiedene Möglichkeiten, CMake auf Ihrem Rechner zu installieren. Sie sollten darauf achten, dass Sie eine CMake-Version installieren, die nach Ihrer aktuellen Compiler Version erschienen ist. So ist sichergestellt, dass CMake alle Befehle Ihres Compilers kennt. Generell gilt bei der CMake-Version sowieso: je aktueller, desto besser.[2] Das heißt, falls Sie gerade erst mit CMake starten und noch keinen Compiler installiert haben oder nicht wissen, welche Compiler Version Sie installiert haben, installieren Sie einfach die aktuellste Version von CMake.[3] Auch wenn sich CMake bereits auf Ihrem Rechner installiert haben, sollten Sie eventuell über ein Upgrade nachdenken. Auf keinen Fall sollten Sie eine CMake-Version unter 3.5 verwenden, da es zuvor noch einige Probleme, insbesondere für Mac Nutzer, gab. Ihre aktuelle CMake-Version können Sie in einem Terminal mit dem Befehl **cmake** --version überprüfen.

Terminal 2.1: Überprüfung der CMake-Version

```
1 $ cmake --version
2 CMake–Version 3.19.1
3
4 CMake suite maintained and supported by Kitware
      (kitware.com/cmake).
```

Unter Windows können Sie CMake über den Paketmanager Chocolatey mit dem Befehl **choco install** cmake installieren.

Terminal 2.2: CMake-Installation mittels Chocolatey

```
1 $ choco install cmake
```

Eine weitere Möglichkeit CMake auf Windows zu installieren, ist der Weg über die offizielle Webseite https://cmake.org/download/ [38]. Dort gibt es auch CMake-

[2] CMake ist abwärtskompatibel, d. h. Sie können auch mit neueren Versionen älteren Code ausführen. Kleinere Ausnahmen gibt es aber natürlich immer, diese kommen jedoch wirklich selten vor und sind in der Regel kein Grund an einer veralteten CMake-Version festzuhalten.

[3] Beim Schreiben dieses Buches ist derzeit CMake-Version 3.22.2 aktuell.