Make & CMake

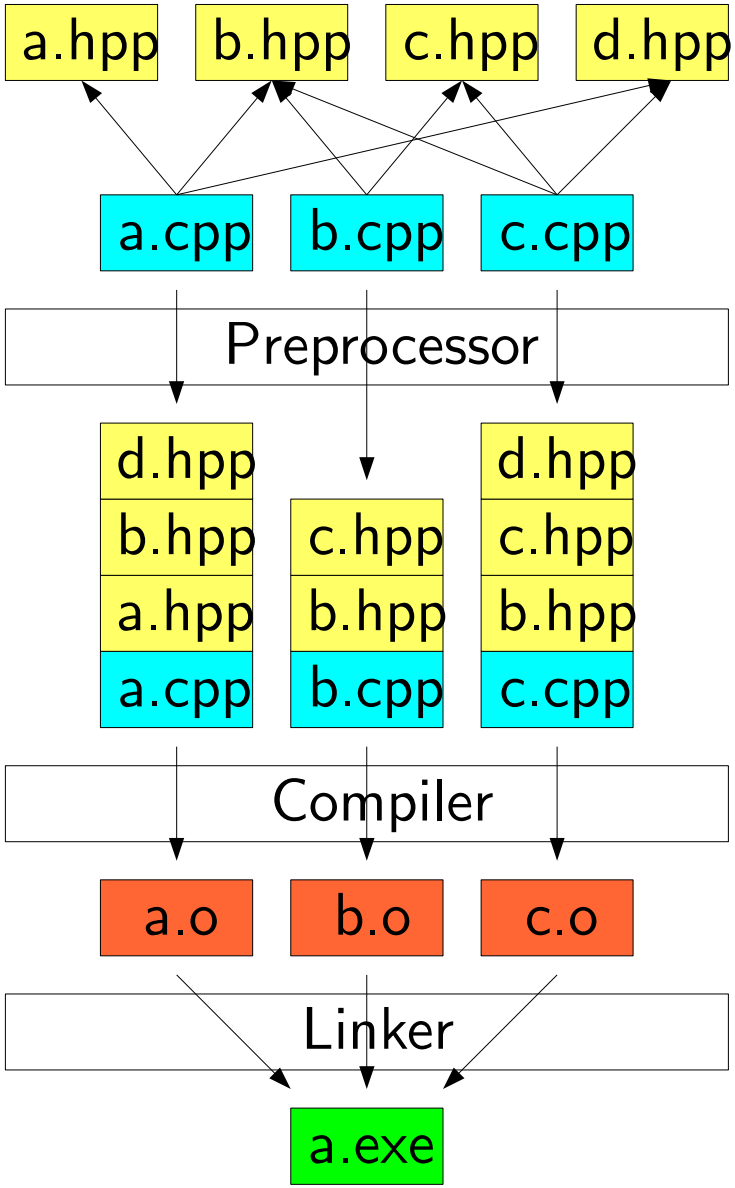
Veit-Hendrik Schlenker

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2016-12-01

Inhaltsverzeichnis

Wie bauen wir Software?




```
cpp hello.c > hello.i
```

```
gcc -c hello.i
```

```
ld -o hello.exe hello.o
```

Wozu brauchen wir Buildsysteme?

- Zeit sparen
- Fehler vermeiden
- Gleiche Ergebnisse erzeugen

- Vorher: Keine Automatisierung/Betriebssystem abhängige Shell-Scripte
- 24.4.1976 Entstehung durch Stuart Feldmann ([mak])
- Heute: Viele Abkömmlinge (dmake,BSD make,GNU Make,nmake)

- Konvertiert eine Quelldatei zur Zieldatei mittels Kommandos
- Entscheidet, ob ein Ziel neu erstellt wird über den Modifikationszeitpunkt
- Arbeitet mit Makefiles
- Syntax ähnlich zur deklarativen Programmierung

```
1 target target target : prerequisite prerequisite
2   (tab) command
3   (tab) command
```

Listing 1: Make Syntax

```
1  edit : main.o kbd.o
2      cc -o edit main.o kbd.o
3
4  main.o : main.c defs.h
5      cc -c main.c
6  kbd.o : kbd.c defs.h command.h
7      cc -c kbd.c
8
9  .PHONY: clean
10 clean :
11     rm edit main.o kbd.o
12
13 foo = c
14 prog.o : prog.$(foo)
15     $(foo)$(foo) -$(foo) prog.$(foo)
```

Listing 2: Make Beispiel

Behandelt wird:

- Wie ersetzt man den Kommandozeilenaufruf mit Makefiles?
- Wie automatisiert man das säubern?
- Wie arbeitet man mit Unterordnern und Variablen?

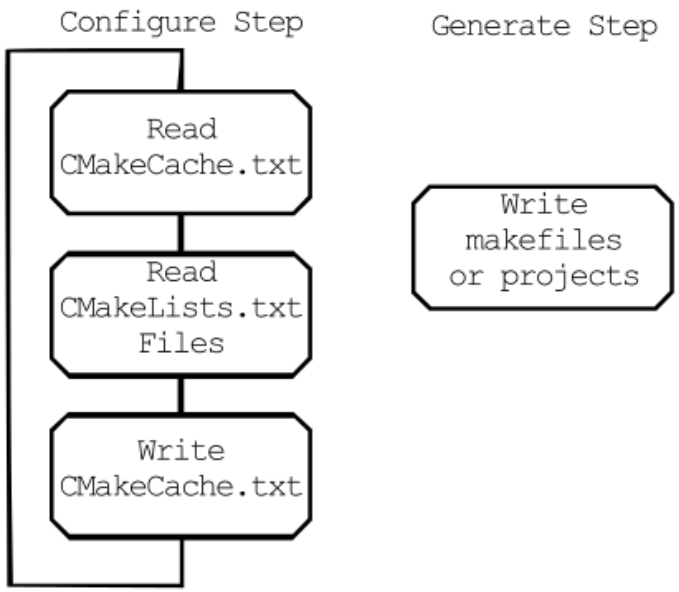
Die verwendeten Präsentationsdateien sind unter [Git] verfügbar

- Generiert für Make, Visual Studio, Xcode, Eclipse
- Bietet daher die Möglichkeit, Änderungen im Build selber (andere Bibliotheken, neue Dateien etc.) schnell auf alle Plattformen zu bringen
- Toolchains und Librarys werden automatisch gefunden und konfiguriert

- Vorher: Verschiedene Buildscripte pro Plattform
- pcmaker (1999) Konvertierte Makefiles zu NMake Dateien für Windows [aos]
- CMake (2000) entstand mit Teilen vom pcmaker und dem Gedanken, einige Funktionalitäten vom Unix configure Tool anzunehmen [cma]

Der CMake Buildprozess in drei Phasen:

1. Configure-Phase: Es wird eine “Zusammenfassung“ der nötigen Schritte und Variablen erzeugt und gespeichert
2. Generate-Phase: Aus der Zusammenfassung werden die Buildanweisungen für das gewünschte System (Makefile, Solution File etc.) erzeugt
3. Die Buildanweisungen können jetzt in der jeweiligen IDE oder Terminal ausgeführt werden.



1. Umgebungsvariablen werden in der CMakeCache gespeichert
2. Anweisungen in der CMakeLists werden ausgeführt für eine In-Memory Repräsentation des Builds.
3. Die CMakeCache wird neu geschrieben mit den gewonnen Informationen

```
1 cmake_minimum_required (VERSION 2.6)
2 project (HelloWorld helloworld.cpp)
3 SET(PROGRAM_SRCS HelloWorld.c)
4 include_directories( \
5     "${PROJECT_SOURCE_DIR}/MathFunctions")
6 add_subdirectory (MathFunctions)
7
8 add_executable(HelloWorld ${PROGRAM_SRCS})
9 target_link_libraries (Tutorial MathFunctions)
```

Listing 3: CMake Beispiel

```
1 add_library(MathFunctions mysqrt.cpp)
```

Listing 4: MathFunctions

Behandelt wird:

- Wie kommt man vom Makefile zum CMake File?
- Was muss man beachten?
- Wie funktioniert CMake unter anderen Systemen?

Die verwendeten Präsentationsdateien sind unter [Git] verfügbar

Zusammenfassung:

- Buildsysteme wie Make nehmen uns wiederkehrende Arbeit ab
- Sie verhindern Fehler
- Sie sparen Zeit
- Und im Falle von CMake ermöglichen sie auch eine Vereinheitlichung der Plattformen

Daher sind wir dank moderner Buildsysteme in der Lage, mehr Zeit mit der Hauptaufgabe der Softwareentwicklung zu verbringen:
Die Entwicklung von Software.

[aos] Aosa cmake.
http://www.aosabook.org/en/cmake.html.

[cma] Cmake history. https://cmake.org/overview/.

[Git] Beispieldateien. https:
//gitlab.com/vhschlenker/semvortrag-dateien.

[mak] Make first occurrence.
http://minnie.tuhs.org/cgi-bin/utree.pl?file=
V7/usr/src/cmd/make/ident.c.