Buildsysteme

Seminar „Effiziente Programmierung“

Yannic Köster

2017-11-09

Gliederung (Agenda)

Was ist ein Buildsystem?

- Begriff nicht genau definiert, doch generell:
  - Dient dem Automatisieren vom Bau von Software
  - Kompiliert, linkt und verpackt Code
- Unterscheiden:
  - Tools, welche Buildprozess durchfürhen
  - Tools, welche Scripts für diese gernerieren

Wieso nutzt man Buildsysteme?

- Konsistente Buildumgebung
- Zuverlässige Ausführung üblichen Schritte
- Manuelle Prozesse automatisieren
- Überprüfung von Abhängigkeiten
- Nicht veränderten Code nicht neu kompilieren

Buildsystem

- Im Rahmen des Vortrags behandelte Systeme
  - Make
  - CMake
  - Meson
- Weitere Systeme
  - Ninja
  - Gradle
  - Autotools
  - Waf
  - Premake

Make

- Führt Buildprozess durch
- Benötigt Makefiles für Anweisungen
- Nicht identisch auf verschiedenen Plattformen → Makefile anpassen
- Lizenz: GNU GPL v3+

Make - Ablauf

- Programm erstellen
- Makefile erstellen
- Make ausführen

Make - Makefile Beispiel

```
1  CC = g++
2  CFLAGS  = -g -Wall
3  LDFLAGS = -lboost_date_time
4  default: Demo
5  Demo:  main.o Klasse2.o
6      $(CC) $(CFLAGS) $(LDFLAGS) -o Showprogramm main.o
       ↪ Klasse2.o
7
8  main.o:  main.cpp header.h
9      $(CC) $(CFLAGS) -c main.cpp
10
11 Klasse2.o:  Klasse2.cpp header.h
12     $(CC) $(CFLAGS) -c Klasse2.cpp
13 clean:
14     $(RM) Demo *.o *~
```

Make - Makefile

- (Compiler Variable definieren)

```
1 [Var_Compiler] = [Compiler]
```

- (Compiler Flags Variable definieren)

```
1 [Var_Flags] = -g -Wall
```

- Ziel angeben

```
1 default: demo
```

Make - Makefile

- Programm definieren

```
1 Demo:  main.o Klasse2.o
2  $(Compiler) $(Flags) -o [Output] [Klassenobjekte]
```

- Einzelende Klassen Objekte definieren

```
1 [Klassenobjekt].o:  [Abhängigkeiten]
2   $(Compiler) $(Flags) -c [Quellcode Datein]
```

Make - Abhängigkeiten

- (Abhängigkeiten Variable deklarieren)

```
1 [Var_Dependency] = [Dependency]
```

- Abhängigkeiten der Programmdefinition hinzufügen

```
1 Demo:  main.o Klasse2.o
2  $(Compiler) $(Flags) $(Dependency)
3  -o [Output] [Klassenobjekte]
```

Make - Projektbeispiele

- Linux Kernel
- Kleine Projekte

CMake

- Erstellt Buildfile , wird von Ninja, Make,etc. genutzt um Programm zu bauen
- Bietet verschiedene Backends an
- Lizenz: BSD License 2.0

CMake - Ablauf

- Programm erstellen
- CMakeLists.txt Datei erstellen
- Buildverzeichnis Inhalt durch CMake erstellen

```
1 $ cmake [Quellcode-Verzeichnis/CMakeLists.txt]
```

```
1 $ make
```

CMake - CMakeLists.txt Beispiel

```
1  Project(new)
2  cmake_minimum_required(VERSION 3.0)
3
4  set(src main.cpp Klasse2.cpp)
5
6  find_package(
7  Boost REQUIRED COMPONENTS date_time)
8  if(Boost_FOUND)
9    include_directories(${Boost_INCLUDE_DIRS})
10   add_executable(Showprogramm ${src})
11   target_link_libraries(
12   demo ${Boost_LIBRARIES})
13 endif()
```

CMake - CMakeLists.txt

- Projektdeklarierung

```
1 Project([Projektname] [Sprache])
```

- Deklarieren von mindest-CMake Version

```
1 cmake_minimum_required(VERSION X.Y)
```

- Ausführbare Datei erstellen

```
1 add_executable([Datei] [Quelldateien])
```

CMake - Abhängigkeiten

- Package suchen

```
1 find_package([Paketname])
```

- Package in Build einbinden

```
1 include_directories (${[Paket]_INCLUDE_DIRS})
```

- Libaries mit ausführbarer Datei linken

```
1 target_link_libraries(
2 [Ausführbare Dateiame]
3 ${[Paket]_LIBRARIES})
```

CMake - Projektbeispiele

- Netflix
- Blender
- MySQL

Meson

- Benötigt min. Python 3.4
- Unterstützte Backends: Ninja, Visual Studio, Xcode
- Erstellt build.ninja, wird von Ninja genutzt um Progrmm zu bauen
- Lizenz: Apache 2.0 License

Meson - Ablauf

- Programm erstellen
- meson.build Datei erstellen
- Buildverzeichnis Inhalt durch Meson erstellen:

```
1 $ meson [Build Verzeichnis]
```

- innerhalb des Buildverzeichnises

```
1 $ ninja
```

Meson - meson.build Beispiel

```
1 project('Showprogramm', 'cpp')
2 src = ['main.cpp','Klasse2.cpp']
3
4 boost_dep = dependency(
5                     'boost',
6                      modules : ['date_time'])
7 exe = executable('Showprogramm', src ,
8                   dependencies : boost_dep)
```

Meson - meson.build

- Besitzt mindestens Projektdeklarierung

```
1 project('[Projektname]', '[Programmiersprachen]')
```

- ausführbare Datei erstellen

```
1 executable('[Datei]', '[genutzte Dateien]')
```

Meson - Abhängigkeiten

- Abhängigkeiten deklarieren

```
1 [Var_Dependency] = dependency('[Name]')
```

- diese Abhängigkeiten nun bei executable einfügen

```
1 executable(
2 '[Dateiname]',
3 '[genutzte Dateien]',
4 dependencies: [Var_Dependency]
5 )
```

Meson - Projektbeispiele

- GNOME
- Xorg (neben Autotools)
- systemd

Zusammenfassung

- Make
  - Nutzt Makefiles um Programm zu bauen
  - Makefile muss manuell geschrieben werden
  - Bei vielen Dateien / Änderungen sehr aufwendig
  - Crossplattform-unfreundlich
- CMake
  - Nutzt CMakeLists um Makefile zu erstellen
  - Integriert Make, Ninja, etc. um Programm zu bauen
- Meson
  - Nutzt Meson.build um build.ninja zu erstellen
  - Integriert Ninja, um Programm zu bauen

Fazit

- Make allein nutzen umständlich
- CMake und Meson relativ ähnlich
- Nutzung abhängig davon, welche Syntax bevorzugt wird
- Build-Zeit recht ähnlich

Literatur 1

- Allgemein
  - Boost Beispielfunktion
    `http://www.boost.org/doc/libs/1_42_0/doc/html/date_time/date_time_io.html`
  - Testvergleich von CMake und Meson von eang.it
    `https://eang.it/is-meson-really-faster-than-cmake/`
- Make
  - Ubuntuusers Artikel zu Makefiles
    `https://wiki.ubuntuusers.de/Makefile/`
  - Makefile Tutorial auf swarthmore.edu
    `https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html`
  - Linux Kernel make
    `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/?h=v4.14-rc7`

Literatur 2

- CMake
  - Githubseite von CMake
    `https://github.com/Kitware/CMake`
  - CMake Dokumentation
    `https://cmake.org/documentation/`
  - CMake Boost einbindung
    `https://cmake.org/cmake/help/v3.0/module/FindBoost.html`
- Meson
  - Meson Dokumentation
    `http://mesonbuild.com/`
  - Githubseite von Meson
    `https://github.com/mesonbuild/meson`