Paketbau mit Cmake

Zusammenfassung eines Kurzvortrag am Linuxwochenende 2010 (Wien)

Wolfgang Dautermann

2010-09-25
FH JOANNEUM

Inhaltsverzeichnis

Zusammenfassung

Dieser Kurzvortrag für das Linuxwochende 2010 in Wien entstand sehr spontan als es schon andere Kurzvorträge zu anderen Package-Methoden gab. Er ist keinesfalls vollständig, sondern soll nur kurz die Möglichkeiten des Paketbaus mit `cmake` zeigen.

Freie Software muss einerseits entwickelt werden und damit diese auch komfortabel verwendet werden kann müssen andererseits auch *Pakete* für alle möglichen Linux-Distributionen, diverse andere Unix-Varianten und sonstige Betriebssysteme gebaut werden.

Dies wird meist von unterschiedlichen Teams gemacht – das Entwicklerteam ist von den Package-Maintainern unterschiedlich und diese müssen die Arbeit für jedes neue Softwarerelease neu machen.

Gehts auch einfacher? Wie kann der/die Softwareentwickler/in den Package-Maintainern die Arbeit erleichtern?

1 Cmake

Cmake ist ein bekanntes Open-Source-Build-System, vergleichbar mit Autoconf/Automake, Scons und anderen bekannten Werkzeugen. Grosse freie Softwareprojekte wechseln vom altbekannten GNU Autoconf/Automake zu Cmake, am bekanntesten war der Wechsel des KDE-Projekts.

Ein Cmake-Projekt (eine `CMakeLists.txt`-Datei) um ein einfaches C-Programm zu übersetzen ist sehr übersichtlich:

```
PROJECT(helloworld)
cmake_minimum_required(VERSION 2.8)

ADD_EXECUTABLE(helloworld helloworld.c)
# bei mehr Sourcefiles:
```

```
# ADD_EXECUTABLE(helloworld h1.c h2.c ...)

# Installation des Binaries
INSTALL(TARGETS helloworld RUNTIME DESTINATION bin)
# Installation der Documentation (README, LICENSE)
INSTALL(FILES LICENSE README DESTINATION share/doc/helloworld)
```

Damit kann man ein Projekt compilieren (`make`) und auch installieren (`make install`). Cmake unterstützt Out-of-Source-builds, die auch empfohlen werden (anstatt einem `make clean`, `make distclean` o.ä. kann man einfach alle Dateien im Build-Tree löschen. Cmake unterstützt umfangreiche Features wie das Testen ob und wo Libraries, Programme, Compiler, Interpreter, ... vorhanden sind – u.a. das ganze KDE Projekt verwendet Cmake als Buildsystem!

Dieses Minmalbeispiel geht von folgender Dateistruktur aus:

```
~/src/
~/src/CMakeLists.txt
~/src/helloworld.c
~/src/LICENSE
~/src/README
~/build/
```

Gebaut wird die Software im `~/build/`-Verzeichnis, der Source ist im `~/src/`-Verzeichnis:

```
user@host:~/build> cmake ../src
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dauti/build
user@host:~/build> make
Scanning dependencies of target helloworld
[100%] Building C object CMakeFiles/helloworld.dir/helloworld.c.o
Linking C executable helloworld
[100%] Built target helloworld
```

Und damit endet meist der Support durch das Projektteam, die Pakete für die diversen Linux-Distributionen (RPM, DEB, ...) müssen von den Package-Maintainern selbst gemacht werden.

2 Paketbau mit Cmake

`cmake` beinhaltet ein eingebautes Packageing-Tool – cpack.

Damit kann man einerseits distributionsübergreifende Pakete im `tar.gz`, `tar.bz2`, ... Format (sowohl vom Sourcecode als auch fertig compilierten Projekt machen – und zusätzlich auch automatisch RPM und DEB Pakete

bauen[1]. Um das zu erreichen müsste das ursprüngliche `CMakeLists.txt` um wenige Zeilen erweitert werden:

```
PROJECT(helloworld)
cmake_minimum_required(VERSION 2.8)

ADD_EXECUTABLE(helloworld helloworld.c)
# Installation des Binaries
INSTALL(TARGETS helloworld RUNTIME DESTINATION bin)
# Installation der Documentation (README, LICENSE)
INSTALL(FILES LICENSE README DESTINATION share/doc/helloworld)

IF (UNIX)
  SET(CPACK_CMAKE_GENERATOR "Unix Makefiles")
  SET(CPACK_SOURCE_GENERATOR "TGZ;TBZ2")
  SET(CPACK_GENERATOR "TGZ;TBZ2;DEB;RPM")
  SET(CPACK_PACKAGE_DESCRIPTION_SUMMARY "Description of Helloworld")
  SET(CPACK_PACKAGE_VENDOR "The Helloworld Team")
  SET(CPACK_PACKAGE_DESCRIPTION_FILE "${CMAKE_CURRENT_SOURCE_DIR}/README")
  SET(CPACK_RESOURCE_FILE_LICENSE "${CMAKE_CURRENT_SOURCE_DIR}/LICENSE")
  SET(CPACK_PACKAGE_VERSION_MAJOR "0")
  SET(CPACK_PACKAGE_VERSION_MINOR "1")
  SET(CPACK_PACKAGE_VERSION "${CPACK_PACKAGE_VERSION_MAJOR}.${CPACK_PACKAGE_VERSION_MINOR}")
  SET(CPACK_SOURCE_PACKAGE_FILE_NAME "${CMAKE_PROJECT_NAME}-${CPACK_PACKAGE_VERSION}")
  SET(CPACK_PACKAGE_CONTACT "Helloworld Team <team@helloworld.org>")
  SET(CPACK_PACKAGE_SECTION "games")
  INCLUDE(CPack)
ENDIF (UNIX)
```

Das beinhaltet nur ein paar allgemeine Informationen über das Paket, das dann in DEB/RPM-Paketen als Metainformation aufscheint. Und dann können beispielsweise mit `make package` tar.gz/tar.bz2/RPM/DEB Pakete automatisch erzeugt werden.

```
user@host:~/build> make package
[100%] Built target helloworld
Run CPack packaging tool...
CPack: Create package using TGZ
CPack: Install projects
CPack: - Run preinstall target for: helloworld
CPack: - Install project: helloworld
CPack: Compress package
CPack: Finalize package
CPack: Package /home/user/build/helloworld-0.1-Linux.tar.gz generated.
CPack: Create package using TBZ2
CPack: Install projects
CPack: - Run preinstall target for: helloworld
CPack: - Install project: helloworld
CPack: Compress package
CPack: Finalize package
CPack: Package /home/user/build/helloworld-0.1-Linux.tar.bz2 generated.
CPack: Create package using DEB
CPack: Install projects
CPack: - Run preinstall target for: helloworld
```

[1] Auch Pakete für Windows (NSIS) und MacOS sind möglich

```
CPack: - Install project: helloworld
CPack: Compress package
CPack: Finalize package
CPack: Package /home/user/build/helloworld-0.1-Linux.deb generated.
CPack: Create package using RPM
CPack: Install projects
CPack: - Run preinstall target for: helloworld
CPack: - Install project: helloworld
CPack: Compress package
CPackRPM: Will use GENERATED spec file:
     /home/user/build/_CPack_Packages/Linux/RPM/SPECS/helloworld.spec
CPack: Finalize package
CPack: Package /home/user/build/helloworld-0.1-Linux.rpm generated.
```

Cmake/Cpack unterstützt dabei umfangreiche Features wie (natürlich) Dependencies, Pre/Postinstallskripte, usw. siehe die Links im Anhang.

Literatur

[1] Cmake Website: `http://www.cmake.org`

[2] Packaging with Cpack: `http://www.vtk.org/Wiki/CMake:Packaging_With_CPack`

[3] Cpack generators: `http://www.vtk.org/Wiki/CMake:CPackPackageGenerators`

[4] Cmake Wiki: `http://www.vtk.org/Wiki/CMake`