CMake - eine plattformübergreifende build-Umgebung

Michael Hammer
michael.hammer@tugraz.at

Grazer Linuxtag 2007
http://www.linuxtage.at

20. Mai 2007

Kurze Vorstellung

Profil Michael Hammer

Mein Name ist Michael Hammer (geboren 1981 in Linz)

Mein Werdegang

- Studium: Maschinbau-Mechatronik (TU Graz)
- ab Juni Wiss. Assistent am Institut für Festigkeitslehre
- Numerische Simulation - Finite Elemente in der Festkörpermechanik
- Erstes Linux 1997 SuSE - heute begeisterter Gentoo User

Übersicht

Übersicht

Übersicht

Übersicht

Features
Warum gerade CMake?

Features

- CMake besticht durch einen raschen Einstieg und guter Skalierbarkeit (bis hin zu Projektgrößen wie bei kde4)
- Erzeugt native build Dateien abhängig von der Umgebung (Makefiles, MS Visual, XCode Projects)
- Erlaubt **sowohl** in-tree also auch out-of-tree builds. Letztere ist ein elegante Möglichkeit um die Quellen von den Binaries zu trennen

Features
Warum gerade CMake?

Features

- CMake besticht durch einen raschen Einstieg und guter Skalierbarkeit (bis hin zu Projektgrößen wie bei kde4)
- Erzeugt native build Dateien abhängig von der Umgebung (Makefiles, MS Visual, XCode Projects)
- Erlaubt **sowohl** in-tree also auch out-of-tree builds. Letztere ist ein elegante Möglichkeit um die Quellen von den Binaries zu trennen

Features
Warum gerade CMake?

Features

- CMake besticht durch einen raschen Einstieg und guter Skalierbarkeit (bis hin zu Projektgrößen wie bei kde4)
- Erzeugt native build Dateien abhängig von der Umgebung (Makefiles, MS Visual, XCode Projects)
- Erlaubt **sowohl** in-tree also auch out-of-tree builds. Letztere ist ein elegante Möglichkeit um die Quellen von den Binaries zu trennen

Features

... was ist wenn ich mehr brauche?

Features

- Hat leistungsfähige Kommandos die es leicht machen Header, Libraries oder Executables zu finden
- Bietet auch “advanced” build Fähigkeiten wie bedingte Übersetzung, reguläre Ausdrücke, rekursive Übersetzung mit Variablenvererbung
- Durch die definierten Schnittstellen und die vielen Kommandos ist es leicht CMake zu erweitern ohne dabei die Übersicht zu verlieren

WICHTIG: CMake ist Open Source ;)

Features

... was ist wenn ich mehr brauche?

Features

- Hat leistungsfähige Kommandos die es leicht machen Header, Libraries oder Executables zu finden
- Bietet auch “advanced” build Fähigkeiten wie bedingte Übersetzung, reguläre Ausdrücke, rekursive Übersetzung mit Variablenvererbung
- Durch die definierten Schnittstellen und die vielen Kommandos ist es leicht CMake zu erweitern ohne dabei die Übersicht zu verlieren

WICHTIG: CMake ist Open Source ;)

Features

... was ist wenn ich mehr brauche?

Features

- Hat leistungsfähige Kommandos die es leicht machen Header, Libraries oder Executables zu finden
- Bietet auch “advanced” build Fähigkeiten wie bedingte Übersetzung, reguläre Ausdrücke, rekursive Übersetzung mit Variablenvererbung
- Durch die definierten Schnittstellen und die vielen Kommandos ist es leicht CMake zu erweitern ohne dabei die Übersicht zu verlieren

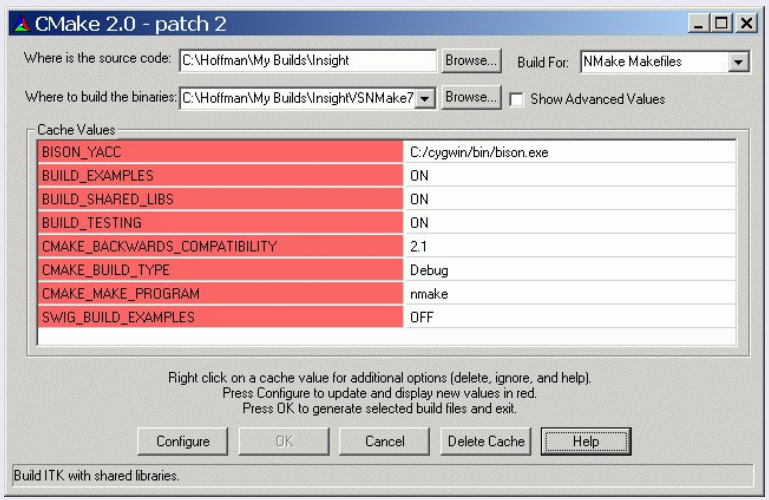
WICHTIG: CMake ist Open Source ;)

GUI

... was wär ein Programm ohne screenshots ...

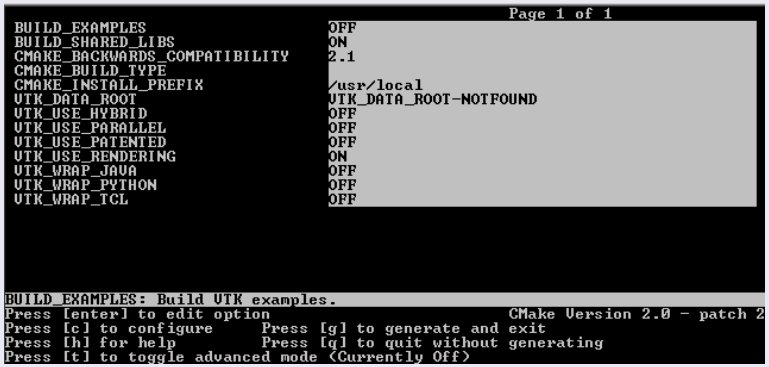
Konfiguration von Grundeinstellungen und CMakeCache Variablen.

Windows



CMakeSetup.exe

Unix Env



ccmake

GUI

... was wär ein Programm ohne screenshots ...

Konfiguration von Grundeinstellungen und CMakeCache Variablen.

Hilfe ich mag keine GUIs!

```
mueli@michael:~/svn/mueli-latex/glt07/cmake$ cmake --help
cmake version 2.4-patch 6
Usage

  cmake [options] <path-to-source>
  cmake [options] <path-to-existing-build>

Command-Line Options
  -C <initial-cache>          = Pre-load a script to populate the cache.
  -D <var>:<type>=<value>     = Create a cmake cache entry.
  -G <generator-name>         = Specify a makefile generator.
  -E                          = CMake command mode.
  -i                          = Run in wizard mode.
  -L[A][H]                    = List non-advanced cached variables.
  -N                          = View mode only.
  -P <file>                   = Process script mode.
  --graphviz=[file]           = Generate graphviz of dependencies.
  --debug-trycompile          = Do not delete the try compile directories..
  --debug-output              = Put cmake in a debug mode.
  --help-command cmd [file]   = Print help for a single command and exit.
  --help-command-list [file]  = List available listfile commands and exit.
  --help-module module [file] = Print help for a single module and exit.
  --help-module-list [file]   = List available modules and exit.
  --copyright [file]          = Print the CMake copyright and exit.
  --help                      = Print usage information and exit.
  --help-full [file]          = Print full help and exit.
  --help-html [file]          = Print full help in HTML format.
  --help-man [file]           = Print a UNIX man page and exit.
  --version [file]            = Show program name/version banner and exit.

Generators

The following generators are available on this platform:
  KDevelop3                   = Generates KDevelop 3 project files.
  Unix Makefiles              = Generates standard UNIX makefiles.
```

Mit dem Aufruf von `cmake` wird der Konfigurationsprozess ausgeführt, die Makefiles und der CmakeCache erstellt.

Und nun Live ...

... einfach zum zuschauen und Fragen stellen!

CMakeCache

Was bitte ist das überhaupt?

Ein zentrales Textfile (`CMakeCache.txt`) in der Wurzel des binary trees welches folgende Dinge beinhaltet.

CMakeCache

- Konfiguriert einige CMake interne Variablen
- Beinhaltet Compiler Flags
- Enthält die Existenz und den Ort von Libraries (gesetzt durch `FIND_LIBRARY`)
- Kann (soll) händisch editiert werden
- Wird **nach** der ersten Erstellung von cmake **nicht** mehr überschrieben!

CMakeCache

Was bitte ist das überhaupt?

Ein zentrales Textfile (`CMakeCache.txt`) in der Wurzel des binary trees welches folgende Dinge beinhaltet.

CMakeCache

- Konfiguriert einige CMake interne Variablen
- Beinhaltet Compiler Flags
- Enthält die Existenz und den Ort von Libraries (gesetzt durch `FIND_LIBRARY`)
- Kann (soll) händisch editiert werden
- Wird **nach** der ersten Erstellung von cmake **nicht** mehr überschrieben!

CMakeCache

Was bitte ist das überhaupt?

Ein zentrales Textfile (`CMakeCache.txt`) in der Wurzel des binary trees welches folgende Dinge beinhaltet.

CMakeCache

- Konfiguriert einige CMake interne Variablen
- Beinhaltet Compiler Flags
- Enthält die Existenz und den Ort von Libraries (gesetzt durch `FIND_LIBRARY`)
- Kann (soll) händisch editiert werden
- Wird **nach** der ersten Erstellung von cmake **nicht** mehr überschrieben!

CMakeCache

Was bitte ist das überhaupt?

Ein zentrales Textfile (`CMakeCache.txt`) in der Wurzel des binary trees welches folgende Dinge beinhaltet.

CMakeCache

- Konfiguriert einige CMake interne Variablen
- Beinhaltet Compiler Flags
- Enthält die Existenz und den Ort von Libraries (gesetzt durch `FIND_LIBRARY`)
- Kann (soll) händisch editiert werden
- Wird **nach** der ersten Erstellung von cmake **nicht** mehr überschrieben!

CMakeCache

Was bitte ist das überhaupt?

Ein zentrales Textfile (`CMakeCache.txt`) in der Wurzel des binary trees welches folgende Dinge beinhaltet.

CMakeCache

- Konfiguriert einige CMake interne Variablen
- Beinhaltet Compiler Flags
- Enthält die Existenz und den Ort von Libraries (gesetzt durch `FIND_LIBRARY`)
- Kann (soll) händisch editiert werden
- Wird **nach** der ersten Erstellung von cmake **nicht** mehr überschrieben!

Und nun Live ...

... einfach zum zuschauen und Fragen stellen!

Das Projekt anlegen - CMakeLists.txt

... oder wie sage ich überhaupt was ich bauen möchte?

Jedes Verzeichnis im source tree erhällt eine `CMakeLists.txt`

CMakeLists.txt

- `PROJECT()` : Definiert vorrangig den Projektnamen
- `SET()` : Setzt Variablen - ${var} zum "Dereferenzieren" von Variablen
- `ADD_EXECUTABLE()` : Fügt executable target hinzu
- `ADD_LIBRARY()` : Fügt library target hinzu - kann sowohl `STATIC` (default) als auch `SHARED` sein

Das Projekt anlegen - CMakeLists.txt

... oder wie sage ich überhaupt was ich bauen möchte?

Jedes Verzeichnis im source tree erhällt eine `CMakeLists.txt`

CMakeLists.txt

- `PROJECT()` : Definiert vorrangig den Projektnamen
- `SET()` : Setzt Variablen - ${var} zum “Dereferenzieren” von Variablen
- `ADD_EXECUTABLE()` : Fügt executable target hinzu
- `ADD_LIBRARY()` : Fügt library target hinzu - kann sowohl `STATIC` (default) als auch `SHARED` sein

Das Projekt anlegen - CMakeLists.txt

... oder wie sage ich überhaupt was ich bauen möchte?

Jedes Verzeichnis im source tree erhällt eine `CMakeLists.txt`

CMakeLists.txt

- `PROJECT()` : Definiert vorrangig den Projektnamen
- `SET()` : Setzt Variablen - ${var} zum “Dereferenzieren” von Variablen
- `ADD_EXECUTABLE()` : Fügt executable target hinzu
- `ADD_LIBRARY()` : Fügt library target hinzu - kann sowohl `STATIC` (default) als auch `SHARED` sein

Das Projekt anlegen - CMakeLists.txt

... oder wie sage ich überhaupt was ich bauen möchte?

Jedes Verzeichnis im source tree erhällt eine `CMakeLists.txt`

CMakeLists.txt

- `PROJECT()` : Definiert vorrangig den Projektnamen
- `SET()` : Setzt Variablen - ${var} zum “Dereferenzieren” von Variablen
- `ADD_EXECUTABLE()` : Fügt executable target hinzu
- `ADD_LIBRARY()` : Fügt library target hinzu - kann sowohl `STATIC` (default) als auch `SHARED` sein

Und nun Live ...

... einfach zum zuschauen und Fragen stellen!

Wie behandle ich Libraries?

Besitzen bereits die Fähigkeit der Vererbung an Subverzeichnisse (wieder mehr beim rekursiven Bauen)

CMakeLists.txt

- `INCLUDE_DIRECTORIES()` : Setzen von Pfaden zu inkludierten Headern
- `LINK_DIRECTORIES()` : Verzeichnisse in denen nach libraries gesucht werden soll
- `TARGET_LINK_LIBRARIES()` : Das Einlinken von libraries in ein angegebenes target

Wie behandle ich Libraries?

Besitzen bereits die Fähigkeit der Vererbung an Subverzeichnisse (wieder mehr beim rekursiven Bauen)

CMakeLists.txt

- `INCLUDE_DIRECTORIES()` : Setzen von Pfaden zu inkludierten Headern
- `LINK_DIRECTORIES()` : Verzeichnisse in denen nach libraries gesucht werden soll
- `TARGET_LINK_LIBRARIES()` : Das Einlinken von libraries in ein angegebenes target

Wie behandle ich Libraries?

Besitzen bereits die Fähigkeit der Vererbung an Subverzeichnisse (wieder mehr beim rekursiven Bauen)

CMakeLists.txt

- `INCLUDE_DIRECTORIES()` : Setzen von Pfaden zu inkludierten Headern
- `LINK_DIRECTORIES()` : Verzeichnisse in denen nach libraries gesucht werden soll
- `TARGET_LINK_LIBRARIES()` : Das Einlinken von libraries in ein angegebenes target

Und nun Live ...

... einfach zum zuschauen und Fragen stellen!

Targets

Welche Targets stehen nun zur Verfügung?

Mögliche Targets

- `depend` : Erstellt source dependencies
- `rebuild_cache` : Holt **zusätzliche** cache Einträge
- `edit_cache` : Bringt die GUI ;)
- `dependlocal` : Erzeugt die dependencies des aktuellen directories (dazu später beim rekusiven Bauen)
- `install` : Installiert wie mit `INSTALL_TARGETS` angegeben
- `clean` : Entfernt alle erzeugten Dateien
- `test` : Startet Test (`ADD_TEST`)
- `all`

Und nun Live ...

... einfach zum zuschauen und Fragen stellen!

Rekursives Bauen

... oder warum ist das Dateisystem sonst ein Baum?

Es ist der übliche Weg source code in einem Verzeichnisbaum zu strukturieren.

Rekursives Bauen

- `SUBDIRS()` : Zwingt cmake in das angegebene Subdirectory abzusteigen und dort die CMakeLists.txt zu verarbeiten
- Variablen werden sinnvoll an das Unterverzeichnis vererbt
- Es ist möglich im binary tree nur Teilbäume zu bauen

Rekursives Bauen

... oder warum ist das Dateisystem sonst ein Baum?

Es ist der übliche Weg source code in einem Verzeichnisbaum zu strukturieren.

Rekursives Bauen

- `SUBDIRS()` : Zwingt cmake in das angegebene Subdirectory abzusteigen und dort die CMakeLists.txt zu verarbeiten
- Variablen werden sinnvoll an das Unterverzeichnis vererbt
- Es ist möglich im binary tree nur Teilbäume zu bauen

Rekursives Bauen

... oder warum ist das Dateisystem sonst ein Baum?

Es ist der übliche Weg source code in einem Verzeichnisbaum zu strukturieren.

Rekursives Bauen

- `SUBDIRS()` : Zwingt cmake in das angegebene Subdirectory abzusteigen und dort die CMakeLists.txt zu verarbeiten
- Variablen werden sinnvoll an das Unterverzeichnis vererbt
- Es ist möglich im binary tree nur Teilbäume zu bauen

Und nun Live ...

... einfach zum zuschauen und Fragen stellen!

Kontrollstrukturen

Bringen wir ein wenig Logik in das Ganze ...

Kontrollstrukturen

- `IF() ELSE() ELSIF() ENDIF(expression)` : evaluiert einen gegebene Ausdruck und steigt in den entsprechenden Zweig entspr. TRUE oder FALSE
- `WHILE() ENDWHILE(condition)` : Führt eine Gruppe von Kommandos aus so lange die Condition TRUE ist
- `FOREACH() ENDFOREACH(loop_var)` : über ${loop_var} Zugriff auf aktuelles Element der Liste

Kontrollstrukturen

Bringen wir ein wenig Logik in das Ganze ...

Kontrollstrukturen

- `IF() ELSE() ELSIF() ENDIF(expression)` : evaluiert einen gegebene Ausdruck und steigt in den entsprechenden Zweig entspr. TRUE oder FALSE
- `WHILE() ENDWHILE(condition)` : Führt eine Gruppe von Kommandos aus so lange die Condition TRUE ist
- `FOREACH() ENDFOREACH(loop_var)` : über ${loop_var} Zugriff auf aktuelles Element der Liste

Kontrollstrukturen

Bringen wir ein wenig Logik in das Ganze ...

Kontrollstrukturen

- `IF() ELSE() ELSIF() ENDIF(expression)` : evaluiert einen gegebene Ausdruck und steigt in den entsprechenden Zweig entspr. TRUE oder FALSE
- `WHILE() ENDWHILE(condition)` : Führt eine Gruppe von Kommandos aus so lange die Condition TRUE ist
- `FOREACH() ENDFOREACH(loop_var)` : über ${loop_var} Zugriff auf aktuelles Element der Liste

Conditional Compiling

Warum immer das Selbe tun?

Zeige ich hier anhand eines DEBUG switches.

DEBUG Modus

- `#ifdef DEBUG {source} #endif` : Kappsle code Teile mit Preprocessoranweisungen
- `IF(${proj}_DEBUG) ENDIF(${proj}_DEBUG)` : Frage mit Hilfe von Kontrollstrukturen Variablen ab
- `SET_SOURCE_FILE_PROPERTIES` : als Beispiel um für einzelne source files Flags zu konfigurieren

Conditional Compiling

Warum immer das Selbe tun?

Zeige ich hier anhand eines DEBUG switches.

DEBUG Modus

- `#ifdef DEBUG {source} #endif` : Kappsle code Teile mit Preprocessoranweisungen
- `IF(${proj}_DEBUG) ENDIF(${proj}_DEBUG)` : Frage mit Hilfe von Kontrollstrukturen Variablen ab
- `SET_SOURCE_FILE_PROPERTIES` : als Beispiel um für einzelne source files Flags zu konfigurieren

Conditional Compiling

Warum immer das Selbe tun?

Zeige ich hier anhand eines DEBUG switches.

DEBUG Modus

- `#ifdef DEBUG {source} #endif` : Kappsle code Teile mit Preprocessoranweisungen
- `IF(${proj}_DEBUG) ENDIF(${proj}_DEBUG)` : Frage mit Hilfe von Kontrollstrukturen Variablen ab
- `SET_SOURCE_FILE_PROPERTIES` : als Beispiel um für einzelne source files Flags zu konfigurieren

Und nun Live ...

... einfach zum zuschauen und Fragen stellen!

Macros

Ein Weg wiederkehrende Aufgaben zu automatisieren.

Erstellen von Executables

```
MACRO(CREATE_EXECUTABLE NAME
  SOURCES LIBRARIES)
  ADD_EXECUTABLE(${NAME} ${SOURCES})
  TARGET_LINK_LIBRARIES(${NAME}
    ${LIBRARIES})
ENDMACRO(CREATE_EXECUTABLE)
ADD_LIBRARY(MyLibrary lib.cpp)
CREATE_EXECUTABLE(Hello hello.cpp MyLibrary)
```

Find Module

Wenn wir auf der Suche nach libs sind ...

Kommandos

- `FIND_PATH()` : zum Lokalisieren von Headern - sucht den Pfad einer Datei
- `FIND_LIBRARY()` : Sucht nach libs in den Standardverzeichnissen

Variablen (landen im CMakeCache)

- `${lib}_LIBRARIES` : Pfad zur library
- `${lib}_INCLUDE_DIR` : Pfad zu den Headern der library
- `${lib}_FOUND` : TRUE oder FALSE - je nachdem ob die lib gefunden wurde

Find Module

Wenn wir auf der Suche nach libs sind ...

Kommandos

- `FIND_PATH()` : zum Lokalisieren von Headern - sucht den Pfad einer Datei
- `FIND_LIBRARY()` : Sucht nach libs in den Standardverzeichnissen

Variablen (landen im CMakeCache)

- `${lib}_LIBRARIES` : Pfad zur library
- `${lib}_INCLUDE_DIR` : Pfad zu den Headern der library
- `${lib}_FOUND` : TRUE oder FALSE - je nachdem ob die lib gefunden wurde

Find Module

Wenn wir auf der Suche nach libs sind ...

Kommandos

- `FIND_PATH()` : zum Lokalisieren von Headern - sucht den Pfad einer Datei
- `FIND_LIBRARY()` : Sucht nach libs in den Standardverzeichnissen

Variablen (landen im CMakeCache)

- `${lib}_LIBRARIES` : Pfad zur library
- `${lib}_INCLUDE_DIR` : Pfad zu den Headern der library
- `${lib}_FOUND` : TRUE oder FALSE - je nachdem ob die lib gefunden wurde

Find Module

Wenn wir auf der Suche nach libs sind ...

Kommandos

- `FIND_PATH()` : zum Lokalisieren von Headern - sucht den Pfad einer Datei
- `FIND_LIBRARY()` : Sucht nach libs in den Standardverzeichnissen

Variablen (landen im CMakeCache)

- `${lib}_LIBRARIES` : Pfad zur library
- `${lib}_INCLUDE_DIR` : Pfad zu den Headern der library
- `${lib}_FOUND` : TRUE oder FALSE - je nachdem ob die lib gefunden wurde

Find Module

Wenn wir auf der Suche nach libs sind ...

Kommandos

- `FIND_PATH()` : zum Lokalisieren von Headern - sucht den Pfad einer Datei
- `FIND_LIBRARY()` : Sucht nach libs in den Standardverzeichnissen

Variablen (landen im CMakeCache)

- `${lib}_LIBRARIES` : Pfad zur library
- `${lib}_INCLUDE_DIR` : Pfad zu den Headern der library
- `${lib}_FOUND` : TRUE oder FALSE - je nachdem ob die lib gefunden wurde

Und nun Live ...

... einfach zum zuschauen und Fragen stellen!

Weitere Module

Was sonst noch so alles configuriert gehört ...

Neben der Detektion von einzelnen libraries sind aber auch noch andere Aufgaben teil des Konfigurationsprozesses:

Konfiguration

- `Check<>.cmake` : Checked gezielte Aufgaben, z.b. ob ein Compiler bestimmte Flags versteht
- `Test<>.cmake` : Testmodule die z.B. big endian, little endian detektieren
- `Use<>.cmake` : Setzt Pfade zu libraries und includes eines ganzen Paketes - z.b. wxWidets

Learning by doing - watch out `/usr/share/cmake/Modules`.

Weitere Module

Was sonst noch so alles configuriert gehört ...

Neben der Detektion von einzelnen libraries sind aber auch noch andere Aufgaben teil des Konfigurationsprozesses:

Konfiguration

- `Check<>.cmake` : Checked gezielte Aufgaben, z.b. ob ein Compiler bestimmte Flags versteht
- `Test<>.cmake` : Testmodule die z.B. big endian, little endian detektieren
- `Use<>.cmake` : Setzt Pfade zu libraries und includes eines ganzen Paketes - z.b. wxWidets

Learning by doing - watch out `/usr/share/cmake/Modules`.

Weitere Module

Was sonst noch so alles configuriert gehört ...

Neben der Detektion von einzelnen libraries sind aber auch noch andere Aufgaben teil des Konfigurationsprozesses:

Konfiguration

- `Check<>.cmake` : Checked gezielte Aufgaben, z.b. ob ein Compiler bestimmte Flags versteht
- `Test<>.cmake` : Testmodule die z.B. big endian, little endian detektieren
- `Use<>.cmake` : Setzt Pfade zu libraries und includes eines ganzen Paketes - z.b. wxWidets

Learning by doing - watch out `/usr/share/cmake/Modules`.

Ein wirklich großes Projekt

... oder warum es sich lohnt sich mit cmake zu beschäftigen

http://websvn.kde.org

Wie geht es weiter?

Weiterführend

- `man cmake`
- `http://www.cmake.org/HTML/Documentation.html`
- `http://www.cmake.org/Wiki/CMake` und `http://www.cmake.org/Wiki/CMake_FAQ`
- abhängig von der Distribution - `/usr/share/cmake/Modules`
- source trees von bestehenden Projekten - `http://websvn.kde.org`

So Long, and Thanks for All the Fish...

Vielen Dank für Ihre Aufmerksamkeit!

```
while(1);
```

(nun können Sie mich mit Fragen löchern :))