Buildautomatisierung mit CMake

Alexander Adam

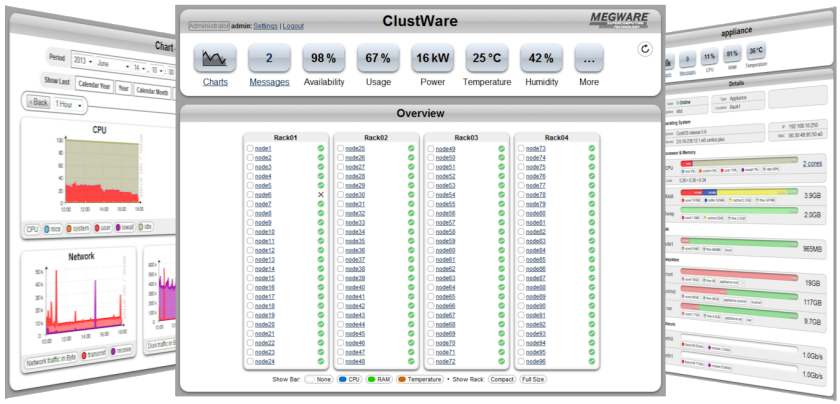
MEGWARE Computer Vertrieb und Service GmbH

20.03.2016

Überblick

Woher kommen wir eigentlich?

- MEGWARE ist ein Chemnitzer Unternehmen mit Hauptgeschäft im Clusterbereich
  $\Rightarrow$ *Hardware (viel)*
- Clustermanagementsoftware ClustWare
  (316 C/C++ Quelldateien)



Beispielprojekt

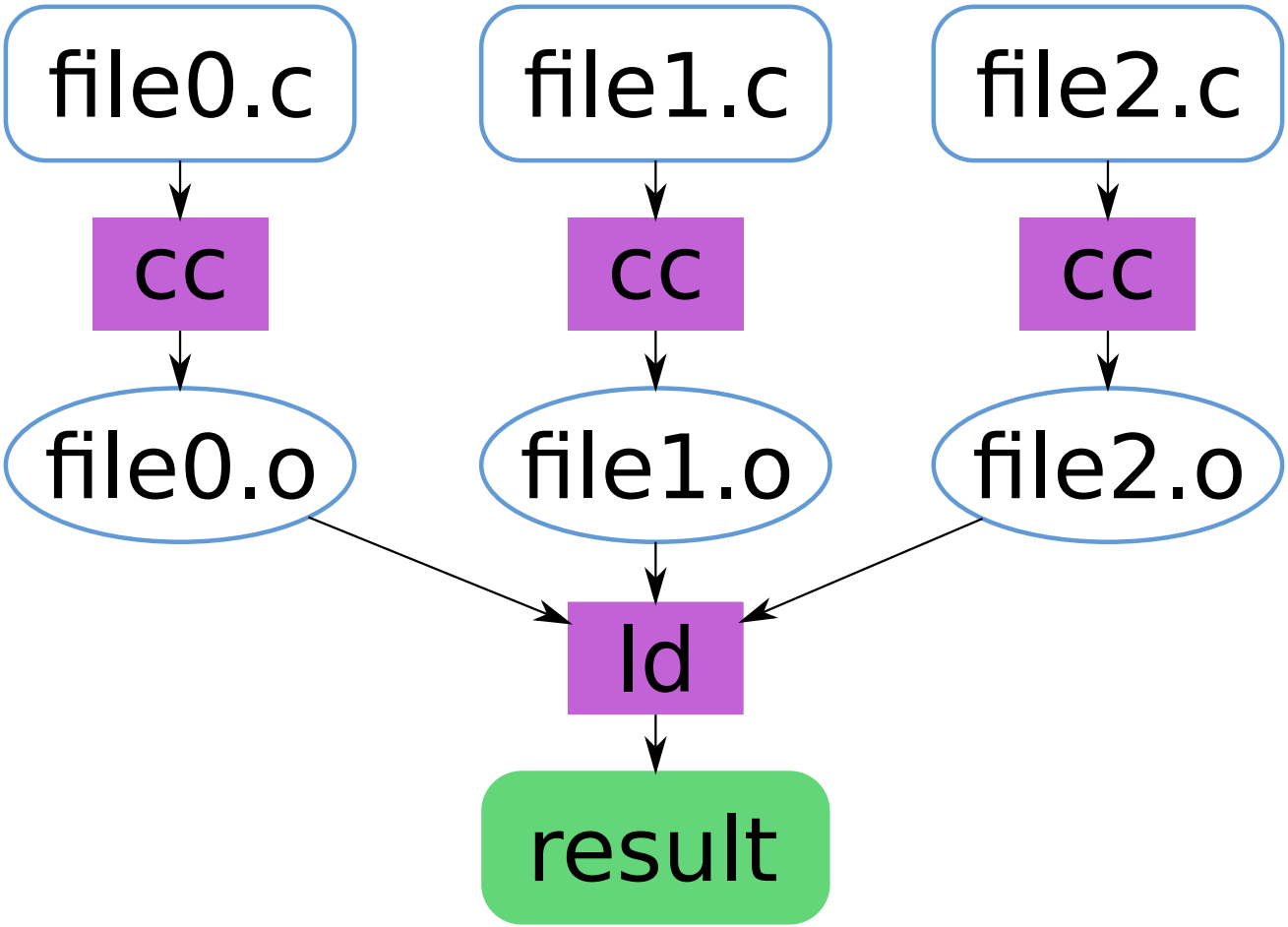
Hauptprogramm

```
#include <stdio.h>

void func1(void);
void func2(void);

int main(void)
{
  printf("-- func1 --\n");
  func1();
  printf("-- func2 --\n");
  func2();
  return 0;
}
```

Bibliothek 1 (und analog 2)

```
#include <stdio.h>

void func1(void)
{
  printf(">> Here is "
         "func1 <<\n");
}
```

Ablauf


file0.c
file1.c
file2.c
cc
cc
cc
file0.o
file1.o
file2.o
ld
result

Shellskripte

Beispielskript (Bash)

```
#!/bin/bash

cc -c file0.c
cc -c file1.c
cc -c file2.c

cc file*.o -o result
```

```
$ ./compile_script.sh
```

- mit Aufwand *sehr* flexibel
- Buildanweisungen sind im Code "versteckt"
- für viele Szenarien einsetzbar
- Aufruf direkt ohne weitere externe Abhängigkeiten

Make

Beispiel-Makefile (GNU make)

```
.PHONY: all clean
all: result

result: file0.o file1.o file2.o
	cc $^ -o $@

clean:
	-rm -f file*.o result
```

```
$ make
```

- klare Struktur der Buildabhängigkeiten
- viele vorgefertigte Regeln (bspw. .c → .o)
- einfach für viele Szenarien einsetzbar
- Aufruf direkt ohne weitere externe Abhängigkeiten

Automake / Autoconf (1)

Beispiel-Automake-Skript

```
# what flags you want to
# pass to the cc & ld
AUTOMAKE_OPTIONS = foreign

# this lists the binaries to
# produce, the (non-PHONY,
# binary) targets in the
# previous manual Makefile
bin_PROGRAMS = result
result_SOURCES = file0.c \
                 file1.c \
                 file2.c
```

- sehr mächtig
- schnell sehr unübersichtliches Buildsystem
- (zu) sehr auf GNU-Toolchain zugeschnitten
- Versions-inkompatibilitäten
- Laufzeit (config *und* build)

Automake / Autoconf (2)

Beispiel-Autoconf-Skript

```
AC_PREREQ([2.69])
AC_INIT(clt_proj, 1.0, a@a.ag)
AM_INIT_AUTOMAKE

# Checks for programs.
AC_PROG_CC

# Checks for header files.
AC_CHECK_HEADERS([stdlib.h])

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Aufrufe:

- `$ aclocal`
- `$ automake`
- `$ autoconf`
- `$ ./configure`
- `$ make`

Nochmal mit CMake

Beispiel-CMakeLists.txt

```
project( clt_proj)

cmake_minimum_required( VERSION 3.5)

add_executable( result
        file0.c
        file1.c
        file2.c
)
```

- $ cmake .
- $ make

- legt neues Projekt “clt_proj” an
- legt minimale Version fest
- fügt Programm hinzu
- CMake weiß, wie es mit C-Quellcode umzugehen hat

CMake-Beispielausgabe

```
-- The C compiler identification is GNU 5.3.0
-- The CXX compiler identification is GNU 5.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - 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
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /clt2016/listing0/build
```

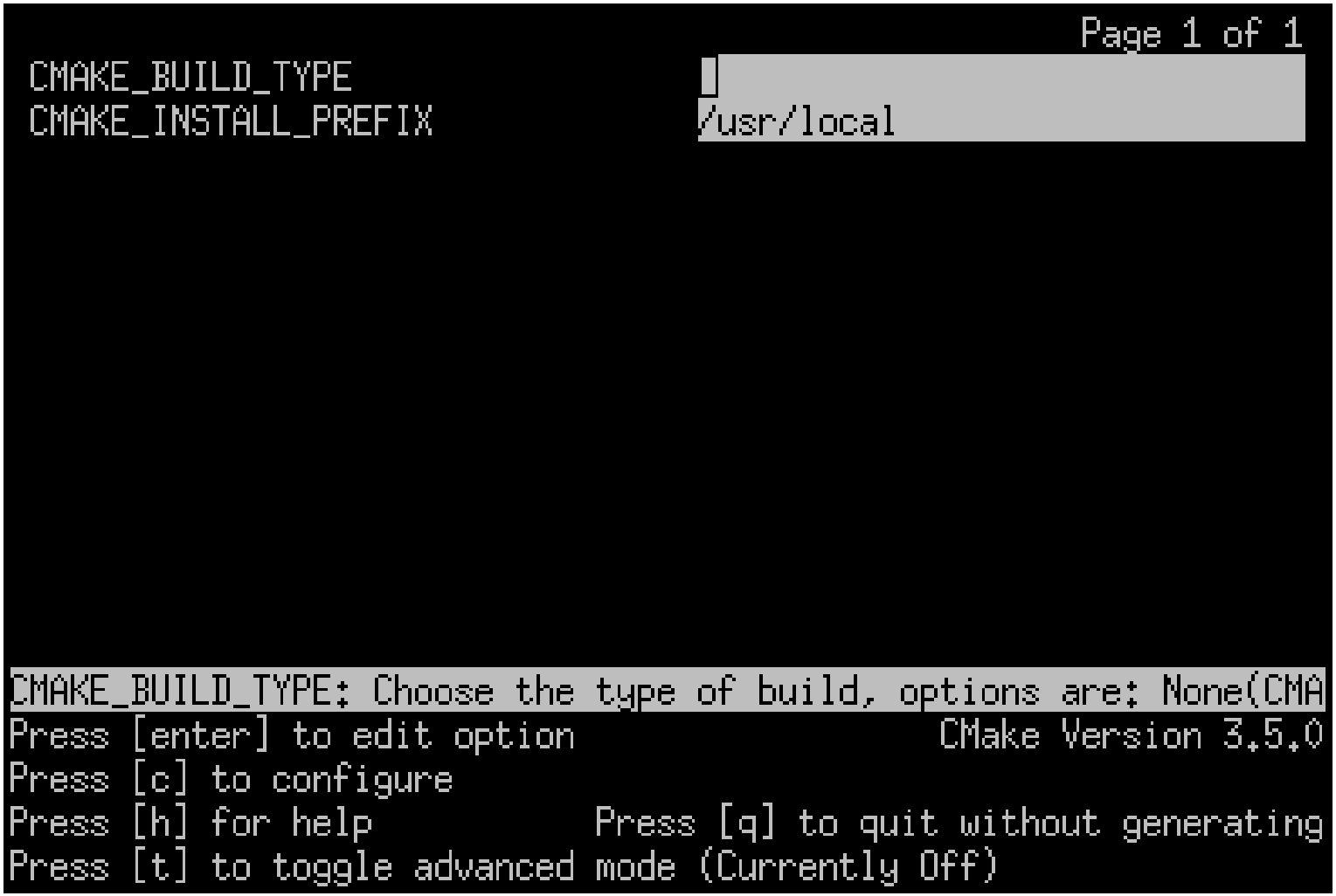
Build-Ausgabe

```
$ make
Scanning dependencies of target result
[ 25%] Building C object CMakeFiles/result.dir/file0.c.o
[ 50%] Building C object CMakeFiles/result.dir/file1.c.o
[ 75%] Building C object CMakeFiles/result.dir/file2.c.o
[100%] Linking C executable result
[100%] Built target result
```

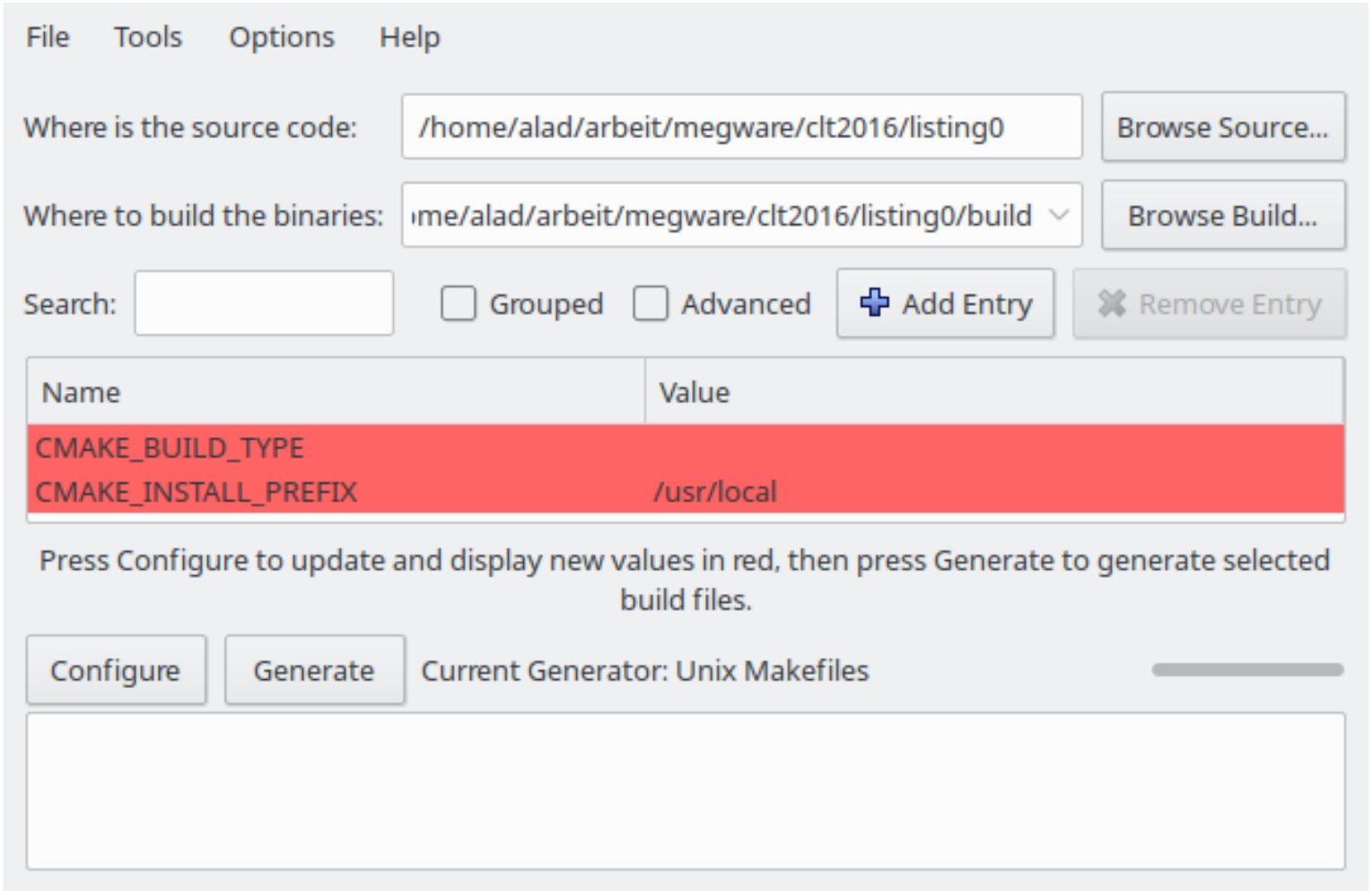
Bis jetzt ...

- Systemcompiler wird herausgesucht und genutzt
- “Out of Source”-Builds (cmake legt die Buildumgebung im aktuellen Arbeitsverzeichnis an)
- Integration in diverse IDEs (QtCreator, XCode, VisualC++, CodeBlocks, KDevelop, Eclipse) und Buildsysteme (`make` und `ninja`)
- textbasierte GUI für Anpassungen am Build (`$ make edit_cache`)
- kleine Hilfe, was alles gebaut werden kann (`$ make help`)

Der CMake-Cache

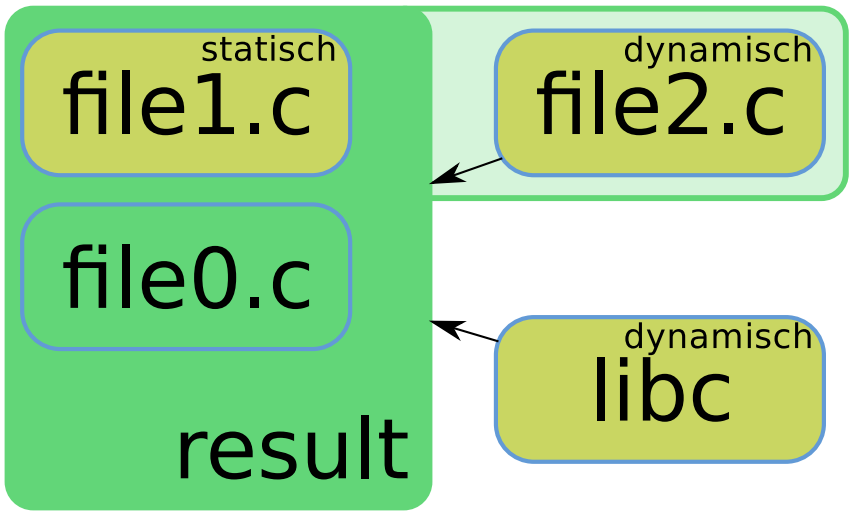


Der CMake-Cache



Was war das gleich?

- Sammlung von Funktionen
- statisch oder dynamisch gelinkt
- projektextern (Abhängigkeiten) oder -intern



Eigene Bibliotheken

- Definition mittels `add_library()`
- Nutzung mittels `target_link_library()`

Beispiel - Nutzung

```
target_link_libraries( result
        lib1
        lib2
)
```

Beispiel - statisch

```
add_library( lib1 STATIC
        file1.c
)
```

Beispiel - dynamisch

```
add_library( lib2 SHARED
        file2.c
)
```

Ausflug Variablensichtbarkeiten

- Sichtbarkeit nach Definition auch in Unterverzeichnissen
- Unterverzeichnisse sind eine neue Sichtbarkeitsdomäne, mittels `PARENT_SCOPE` nach "oben" propagierbar
- Cache:
  - über mehrere CMake-Läufe persistenter globaler Variablenbereich
  - Ablagebereich für Bibliothekssuchergebnisse und Nutzereinstellungen
  - Veränderbar mittels: `$ make edit_cache`
  - mittels `-D<var>=<wert>` auf Kommandozeile setzbar

Fremde Bibliotheken

- umfangreiche Bibliothekssuchmodule – `find_package()` (bspw. Git, GTK, LATEX, MPI, OpenGL, Qt, ZLIB)
- eigene Suchmodule – `find_library()`, `find_path()`
- Zugriff auf pkg-config – `find_package( PkgConfig)` `pkg_check_modules (FOO glib-2.0>=2.10 gtk+-2.0)`

Wir suchen Bibliotheken – CMakeLists.txt

Modulpfad anpassen

```
set( CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}
     CACHE STRING "Zusätzlicher CMake-Modulpfad"
)
```

Nutzung des Suchmodules

```
find_package( DL)
```

Nutzung der Suchergebnisse

```
target_link_libraries( result ${DL_LIBRARY})
```

Suchmodul

FindDL.cmake

```
include(       FindPackageHandleStandardArgs)

find_library( DL_LIBRARY      dl)
find_path(    DL_INCLUDE_DIR dlfcn.h)

find_package_handle_standard_args(
                DL
                DEFAULT_MSG
                DL_LIBRARY
                DL_INCLUDE_DIR
)
```

Kling gut, aber wozu?

- Anpassung der Software auf das Build-/Zielsystem (bspw. Installationspfade, Bibliotheksversionen)
- Zu- und Abschalten diverser Features (graphische Nutzeroberfläche, Datenbankunterstützung)
- automatisierte Integration von Informationen (Versionsstände, Datum, ...)

Build-/Zielsystem (1)

Beispiel

```
if (DL_FOUND)
        option( USE_DL "Nutze libdl" TRUE)
else( DL_FOUND)
        set( USE_DL FALSE CACHE BOOL "Nutze libdl")
endif( DL_FOUND)
```

- Einfache An/Aus-Optionen mittels `option()`
- meist <XYZ>_FOUND definiert
- `if()... else()... endif()`

Build-/Zielsystem (1)

Beispiel

```
if( USE_DL)
        target_link_libraries( result ${DL_LIBRARY})
else( USE_DL)
        message( STATUS "Building without libdl")
endif( USE_DL)
```

- Einfache An/Aus-Optionen mittels `option()`
- meist <XYZ>_FOUND definiert
- `if()`... `else()`... `endif()`

Build-/Zielsystem (1)

config.h.in

```
#cmakedefine USE_DL
#cmakedefine USE_GIBTS_NICHT
const char dl_use = "@USE_DL@";
```

⇓ `configure_file( config.h.in config.h)`

config.h

```
#define USE_DL
/* #undef USE_GIBTS_NICHT */
const char dl_use = "ON";
```

Installation

Beispiel

```
install( TARGETS result lib1 lib2
         RUNTIME DESTINATION bin/
         LIBRARY DESTINATION lib/
         ARCHIVE DESTINATION lib/static/
)
```

- `$ make install`
- CMAKE_INSTALL_PREFIX wird vor relative Pfade gesetzt (analog zu `configure --prefix=<pfad>`)

Pakete - Grundlagen

- CPack ist für das Paketbauen verantwortlich
- gehört zum CMake-Programmpaket
- Aktivierung des Paketfeatures:

```
include( CPack)
```

- Erstellen eines Paketes (gepacktes Archiv) mittels:

```
$ make package
```

Distributionspakete - Debianarchive

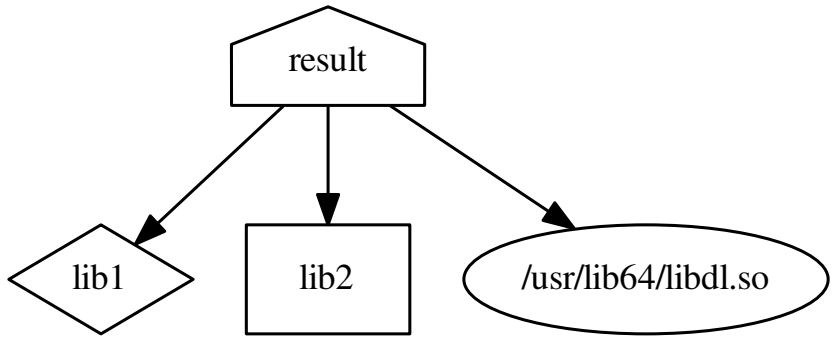
Beispiel

```
set( CPACK_GENERATOR
    "STGZ"
    "DEB"
    CACHE STRING "Packaging types for binary packages"
)
set( CPACK_PACKAGE_CONTACT "CLT Speaker")
```

- Abbildung von Laufzeitabhängigkeiten
- Skripte vor und nach der Installation
- Updatemöglichkeit

Sträflich vernachlässigt

- CTest – automatisiertes Abspielen von Testfällen
  `enable_testing()`
  `add_test()`
- Portabilität: “`cmake -E ...`” für verschiedene, oft genutzte Operationen
- Abhängigkeiten mit `--graphviz=<output>` visualisierbar



Weitere Infos

- gute Referenzdokumentation unter `http://www.cmake.org/`
- Kitware CMake-Buch: gedrucktes Tutorial mit anschließend ausgedruckter Referenz


THAT'S ALL
FOLKS !