Linux Automation Konferenz 2005

Praxis Cross-Plattform Entwicklung

Echte Linux Applikationen
unter Windows erstellen.

- HOWTOs -

Referent : Friedhelm Wolf, Homag AG
(friedhelm.wolf@homag.de)

Inhaltsverzeichnis

1 Beschreibung der Installation von Cygwin...3
2 Erstellen eines Cross-Compilers..4
2.1 Prinzip..4
2.2 Step Guide zur Cross-Compiler Erstellung (mit Abkürzung)...........5
2.2.1 Vorbereitungen...5
2.2.2 Target Linux Runtime zusammensammeln :................................5
2.2.3 Erstellen der binutils :...6
2.2.4 Erstellen der gcc :...6
2.2.5 Erstellen des gdb:...6
2.2.6 Auf dem Weg zur Automatisierung..7
3 Benutzung von Crosstool..8
4 GNU Toolchain...9
4.1 Beschreibung..9
4.3 Die GNU Compiler Collection..9
4.4 Makefiles und GNU Make..9
4.5 GNU autotools..11
4.6 Aufruf GNU Toolchain..16

1 Beschreibung der Installation von Cygwin

Siehe auch http://www.cygwin.com

1. Download des cygwin Setup Programms unter http://www.cygwin.com/setup.exe
2. Starten von Setup.exe > Weiter
3. **Choose Installation Type :** Bei Erstinstallation sollte „Install from Internet“ ausgewählt werden. > Weiter
4. **Choose Installation Directory :** Zielverzeichnis für cygwin angeben. Zum Beispiel: “C:\cygwin“ Es wird geraten, nicht direkt nach „C:\“ zu installieren, da dies zu Problemen führen kann.
5. Default Text Style auf Unix setzen, wenn nur Linefeed als Zeilenendkennung verwendet werden soll. > Weiter
6. **Select Local Package Directory :** Hier werden alle hernutergeladenen Packete für spätere Nachinstallationen gespeichert werden. Zum Beispiel : „C:\cygwin-packages“ > Weiter
7. **Select Connection Type :** Falls ein Proxy benötigt, muss die Verbindung jetzt angegeben werden. > Weiter
8. **Choose Download Site(s) :** Mirror Seite angeben. Gute Erfahrungen wurden mit ftp://ftp-stud.fht-esslingen.de gemacht. > Weiter
9. **Select Packages :** Nach dem automatischen Herunterladen der Packet-Informationen können die zu installierenden Packete ausgewählt werden. Für eine Erstinstallation kann hier „Curr“ (oben rechts) als Installationsvoreinstellung ausgewählt werden, wodurch gängige Packete ausgewählt werden. Die sinnvollste „View“ Einstellung ist die alphabetische Auflistung („View Full“). Es sei erwähnt, dass der Umgang mit diesem Packet-Verwaltungsprogramm dem Verfasser einiges an Nerven gekostet hat, weil es entweder bei Nachinstallationen alles neu installiert hat (dies kann mit Auswahl von „Keep“ anstatt „Curr“ verhindert werden.) oder alte Packetrepositorys auf dem Rechner waren, die inkonsistent zu neu installierten Packeten war. Folgende Packete sollten auf jeden Fall mit installiert werden (indem man auf die Spalte „New“ des jeweiligen Eintrags klickt) : `autoconf, automake, libtool, make, gcc, gdb` und `wget` (für crosstool). > Weiter
10. Die benötigten Packete werden jetzt hernutergeladen und danach entpackt und installiert. Danach ist Cygwin lauffähig.
11. Über den Aufruf der Datei `[C:\cygwin\]cygwin.bat` kann eine Shell gestartet werden. Da die Cygwin Programme jedoch ausnahmslos Windows-Executables sind, kann man sie durch setzen der entsprechenden Pfade auch aus der Windows Eingabeaufforderung aufrufen.
12. Cygwin Shell starten und "`mkpasswd -l -d > /etc/passwd`"

    und "`mkgroup -l -d > /etc/group`" aufrufen.
13. Erweitern der `PATH` Variable um folgende Einträge ist sinnvoll : `C:\cygwin\bin;C:\cygwin\usr\local\bin`, außerdem sollten später hier die Pfade zu den benötigten Cross-Compilern angegeben werden.

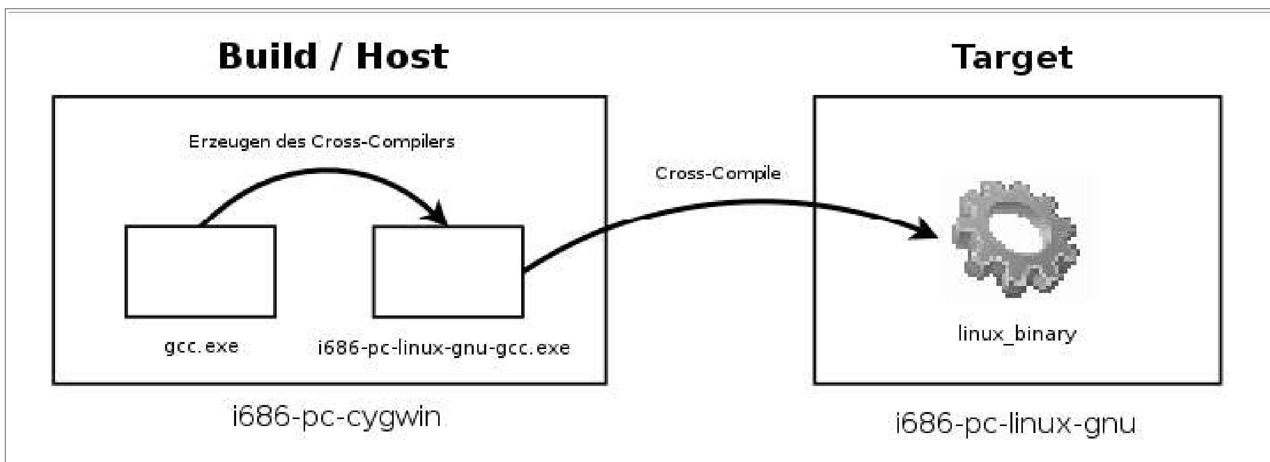
2 Erstellen eines Cross-Compilers

2.1 Prinzip

Ein Cross Compiler ist ein Compiler, der Binaries für eine andere Rechnerplattform erzeugt. Unter einer Plattform versteht man ein Rechnersystem mit einer bestimmten Prozessorarchitektur, einem bestimmten Betriebssystem und zugehörigen Dateiformaten. An der Erzeugung eines Cross Compilers können bis zu drei verschiedene Plattformen beteiligt sein. Diese sind folgendermaßen benannt: Das **Build-System** ist die Plattform, auf dem der Compiler durch Kompilierung erzeugt wird. Das **Host-System** ist die Plattform, auf dem der erzeugte Compiler ausgeführt wird. Das Target-System ist die Plattform, auf dem Executables, die der erzeugte Compiler erzeugt, ausgeführt werden. Im Umfeld der Entwicklung von Cross Compilern sind sogenannte „Configuration Triplets" zur exakten Identifizierung einer Plattform gängig. Diese setzen sich folgendermaßen zusammen:

```
<CPU>-<Hersteller>-<Betriebssystem>
```

Beispiele hiefür sind i686-pc-cygwin (IBM PC mit Intel x86er CPU und GNU Linux als Betriebssystem), wobei pc eine häufig vorkommende Herstellerangabe ist, oder sparc-sun-solaris2.7 (SPARC CPU der Firma Sun46 mit dem Betriebsystem Sun Solaris.) Es kann jede Kombination der Systeme vorkommen, oft sind das Build- und das Host-System identisch. Als Candian Cross werden Builds bezeichnet, bei denen drei unterschiedliche Systeme einbezogen sind. Bei einem „normalen" Compiler sind alle drei Systeme identisch. Analog zu Cross Compilern gibt es Cross Assembler, Cross Linker und Cross Debugger. Im Rahmen der Portierung des Werkzeugmanager Projekts wird ein Cygwin Host, Cygwin Build, Linux Target Cross Compiler benötigt:



Die Möglichkeit einen Cross Compiler zu erstellen, ist schon im configure Script der einzelnen relevanten Programme der GNU Compiler Collection enthalten. Dieses nimmt Parameter zur Angabe der drei Systeme entgegen und erzeugt das entsprechende Executable.

Der Aufruf für obiges Beispiel würde folgendermaßen lauten:

./configure --build=i686-pc-cygwin --host=i686-pc-cygwin --target=i686-pc-linux-gnu

2.2 Step Guide zur Cross-Compiler Erstellung (mit Abkürzung)

2.2.1 Vorbereitungen

Vorraussetzung ist eine fertig installiertes Cygwin (siehe Kapitel 1).

Zur Vereinfachung der Cross-Compilierung, sollten folgende Umgebungsvariablen in Cygwin angelegt werden. Dazu öffnet man die Datei ~/.bashrc und trägt folgende Befehle ein:

```
export HOST=i686-pc-cygwin
export BUILD=i686-pc-cygwin
export TARGET=i686-pc-linux-gnu
export PREFIX=/usr/local/linux
export SRC_ROOT=/usr/src
```

Mit dem Befehl `mkdir -p $PREFIX/$TARGET` wird das benötigte Installations-Verzeichnis angelegt.

2.2.2 Target Linux Runtime zusammensammeln :

Die nächsten Schritte müssen auf der Linux Plattform ausgeführt werden, für die der CrossCompiler gedacht ist:

1. Anlegen eines Verzeichnisses (zum Beispiel /linux-runtime/), in das alle benötigten Dateien kopiert werden. Dieses wird ab jetzt mit `RT` abgekürzt.
2. Kopieren des kompletten `/usr/include` Verzeichnisses in das Verzeichnis `RT/include`. *Tipp*: Am schnellsten mit `(cd /usr; tar cf - include) | tar xf -`
3. Auffinden aller Symbolic Links mit `find . -lname "*" -printf "%p -> %l\n"` und löschen aller Links, die aus dem include Verzeichnis heraus verweisen (im Testfall: ./X11 und das gesamte ./GL Verzeichnis)
4. Mit `ln -eichns include sys-include` im RT Verzeichnis einen Link auf das include Verzis mit dem Namen sys-include anlegen, der für das Kompilieren von GCC später benötigt wird.
5. Folgende Dateien aus dem /lib Verzeichnis ins RT/lib Verzeichnis kopieren: `libm-2.3.2.so libm.so.6 libdl-2.3.2.so libdl.so.2 libc-2.3.2.so libc.so.6   ld-2.3.2.so ld-linux.so.2 libpthread-0.10.so libpthread.so.0`
6. Folgende Dateien aus dem /usr/lib Verzeichnis ins RT/lib Verzeichnis kopieren: libm.a libc_nonshared.a libc.a libpthread_nonshared.a libpthread.a gcrt1.o crt1.o crti.o crtn.o Mcrt1.o libc.so libpthread.so libdl.a libdl.so

7. Anpassen der Datei */lib/clib.so*, indem man den Eintrag `/lib/libc.so.6` zu `/usr/local/linux/i686-pc-linux-gnu/lib/libc.so.6` und den Eintrag `/usr/lib/libc nonshared.a` zu `/usr/local/linux/i686-pc-linux-gnu/lib/libc_nonshared.a` ändert. Ebenso in der Datei *libpthread.so*. Dadurch wird der Linker angewiesen, auch unter Cygwin die Linux Bibliotheken zu benutzen.
8. Das Komplette RT Verzeichniss muss jetzt (am besten gepackt mit "`tar -zcvf linux-targetruntime.tar.gz .`") auf den Cygwin Rechner ins Verzeichnis

`$PREFIX/$TARGET` kopiert werden.

2.2.3 Erstellen der binutils :

Die binutils sind Programme, die der Compiler benötigt, um lauffähige Programme zu erstellen. Elemente sind z.B. der Linker und der Assembler. Folgende Schritte sind notwendig:

1. Zum Herunterladen der Quellcodes von binutils unter http://www.gnu.org./directory/binutils.html den Link Source Tarball anklicken. Das Archiv wird nach `$SRC_ROOT` entpackt. (Im Test wurden die Binutils in der Version 2.14 benutzt).
2. Anlegen des Build Verzeichnisses mit dem Befehl `mkdir -p $SRC_ROOT/build/binutils`
3. Wechseln in dieses Verzeichnis
4. Erstellen der binutils mit dem Befehlsaufruf `$SRC_ROOT/binutils-2.14/configure --with-included-gettext --target=$TARGET --host=$HOST --build=$BUILD --prefix=$PREFIX -v`
5. Als nächstes werden die Binutils compiliert mit dem Aufruf `make`. Der Erstellungsvorgang dauert einige Zeit.
6. Installation der Binutils mit `make install`
7. Um die Binutils auch benutzen zu können muss folgendes Verzeichnis mit folgendem Befehl, der in die Datei /.bashrc eingetragen wird, zur PATH Variable hinzugefügt werden: `export PATH=$PATH:$PREFIX/bin`

2.2.4 Erstellen der gcc :

Als nächstes kann die GNU Compiler Collection als Cross Compiler erstellt werden. Folgende Schritte sind dafür notwendig:

1. Herunterladen des Quellcodes von http://www.gnu.org/directory/gcc.html. (Im Test Version 3.3.3)
2. Entpacken des Archives nach `$SRC ROOT`.
3. Anlegen des Build Verzeichnisses mit dem Befehl `mkdir -p $SRC_ROOT/build/gcc`
4. Wechseln in dieses Verzeichnis
5. Erstellen der gcc mit dem Befehlsaufruf `$SRC_ROOT/gcc-3.3.3/configure --with-included-gettext --enable-languages=c,c++ --host=$HOST --target=$TARGET --build=$BUILD --prefix=$PREFIX -v`
6. Als nächstes wird der GCC mit dem Aufruf `make` kompiliert.
7. Installation des GCC mit `make install`

2.2.5 Erstellen des gdb:

Nun kann der GNU Debugger als Cross Debugger erstellt werden. Folgende Schritte sind dafür notwendig:

1. Herunterladen des Quellcodes von http://www.gnu.org/directory/gdb.html. (Im Test Version 6.1.1)
2. Entpacken des Archives nach `$SRC_ROOT`.
3. Anlegen des Build Verzeichnisses mit dem Befehl `mkdir -p $SRC_ROOT/build/gdb`

4. Wechseln in dieses Verzeichnis
5. Erstellen der gcc mit dem Befehlsaufruf `$SRC_ROOT/gcc-6.1.1/configure --with-included-gettext --host=$HOST --target=$TARGET --build=$BUILD --prefix=$PREFIX -v`
6. Als nächstes wird der GDB compiliert mit dem Aufruf `make`
7. Installation des GDB mit `make install`

Die Cross-Toolchain kann nun ausgeführt werden. Der Eintrag `/usr/local/linux/bin` sollte zur `PATH` Variable hinzugefügt werden.

2.2.6 Auf dem Weg zur Automatisierung

Es wurde schnell deutlich, dass es sinnvoll ist, diesen Vorgang zu scripten. Vorraussetzungen für das folgende Script sind vorhandene Quellcode Archive der benötigten Anwendungen, sowie eine tar-Datei mit der Linux Runtime (beschrieben unter 2.2.2). Die Variablen am Anfang können zum Anpassen der Verzeichnisse und Versionen verwendet werden:

```
# /bin/bash
export TARGET=i686-pc-linux-gnu
export HOST=i686-pc-cygwin
export BUILD=i686-pc-cygwin
export PREFIX=/usr/local/linux
export SRC_ROOT=/usr/src
export TARS=/usr/local/tars
export SRC_BINUTILS=binutils-2.14
export SRC_GCC=gcc-3.3.3
export SRC_GDB=gdb-6.1.1
mkdir -p $PREFIX/$TARGET
cd $PREFIX/$TARGET
tar -zxvf $TARS/linux-runtime.tar.gz
cd $SRC_ROOT
tar -zxvf $TARS/$SRC_BINUTILS.tar.gz
tar -zxvf $TARS/$SRC_GCC.tar.gz
tar -zxvf $TARS/$SRC_GDB.tar.gz
mkdir -p build/binutils
mkdir -p build/gcc
mkdir -p build/gdb
cd $SRC_ROOT/build/binutils
$SRC_ROOT/$SRC_BINUTILS/configure --with-included-gettext --host=$HOST --
build=$BUILD --target=$TARGET --prefix=$PREFIX -v
make
make install
export PATH=$PATH:$PREFIX/bin
cd $SRC_ROOT/build/gcc
$SRC_ROOT/$SRC_GCC/configure --with-included-gettext --enable-
languages=c,c++ --host=$HOST --build=$BUILD --target=$TARGET --
prefix=$PREFIX -v
make
make install
cd $SRC_ROOT/build/gdb
$SRC_ROOT/$SRC_GDB/configure --with-included-gettext --host=$HOST --
build=$BUILD --target=$TARGET --prefix=$PREFIX -v
make
make install
```

3 Benutzung von Crosstool

Diese ersten Ansätze zur Automatisierung einer CrossCompiler-Erstellung führten zur Endeckung von Dan Kegels "CrossTool" einer Sammlung von Shell-Scripten und Konfigurationsdateien für viele Zielsysteme (im Internet unter http://www.kegel.com/crosstool). Die gesammelten Erfahrungen sind noch nicht sehr umfangreich. Es wurde versucht einen PowerPC CrossCompiler für Cygwin zu Erstellen. Dazu wurde, wie folgt vorgegangen:

1. Downloaden von Crosstool unter http://www.kegel.com/crosstool
2. Entpacken ins `/usr/local` Verzeichnis
3. folgende Quellcode Packete ins `/usr/src` Verzeichnis entpacken:
   - binutils-2.14
   - gcc-3.3.3
   - gdb-6.1.1
   - glibc-2.3.1
   - linux-2.4.29
4. Die linuxthreads-Sourcen (glibc-linuxthreads-2.3.1) ins `TARBALLS_DIR` (z.B. `/usr/local/tars`) legen
5. Erstellen des Verzeichnisses `/opt/crosstool` und ändern der Berechtigung auf den Benutzer
6. Schreiben einer eigenen .dat Datei: `homag-ppc.dat`

```
TARBALLS_DIR=/usr/local/tars
SRC_DIR=/usr/src
RESULT_TOP=/opt/crosstool

BINUTILS_DIR=binutils-2.14
GCC_DIR=gcc-3.3.3
GLIBC_DIR=glibc-2.3.4
LINUX_DIR=linux-2.4.29
GLIBCTHREADS_FILENAME=glibc-linuxthreads-2.3.4

TARGET=powerpc-750-linux-gnu
TARGET_CFLAGS="-O"
GCC_LANGUAGES="c,c++"
GCC_EXTRA_CONFIG="--with-cpu=750 --enable-cxx-flags=-mcpu=750"
```

7. Schreiben eines eigenen Scriptes: homag-ppc.sh

```
#!/bin/sh
set -ex
eval `cat homag-ppc.dat` sh all.sh --notest --nounpack
```

8. Aufruf des Scripts aus `/usr/local/crosstool...`
9. Die Toolchain wird in `/opt/crosstool/powerpc-750-linux-gnu` erstellt.
10. Einbinden des Verzeichnisses in die $PATH Variable

4 GNU Toolchain

4.1 Beschreibung

Die GNU Toolchain ist ein umfassender Begriff für die Entwicklungswerkzeuge des GNU Projekts. Sie hängen sehr eng zusammen und werden sowohl in der Anwendungs- als auch in der Betriebssystementwicklung eingesetzt. Der Linux Kernel, das freie Betriebssystem BSD und viele Anwendungen auf Embedded Systemen werden mit ihrer Hilfe erstellt. Einzelne Programme der Toolchain finden auch auf Solaris und Windows Betriebssystemen Verwendung.

Die wichtigsten Programme sind

- Die GNU Compiler Collection (GCC) - eine Sammlung von Compilern für mehrere Sprachen
- Die GNU Binutils - die viele Programme als Grundlage des Compilers, wie Linker und Assembler beinhalten.
- Der GNU Debugger (GDB) - ein kommandozeilenbasierter, interaktiver Debugger.
- Das GNU Build System (autotools) - Enthält die Programme autoconf, automake, libtool, und GNU Make und stellt damit Werkzeuge zum automatischen Erstellen von komplexen Applikationen bereit.

4.3 Die GNU Compiler Collection

Als GCC wird sowohl das FrontEnd des GNU Projektes für mehrere Compiler unterschiedlicher Programmiersprachen, als auch darin enthaltene C/C++ Compiler bezeichnet. Im folgenden ist immer vom C Compiler die Rede. Auf vielen Plattformen und Betriebssystemen (ausgenommen Microsofts Windows Betriebssystem) ist dieser Compiler inzwischen ein Standardwerkzeug. Er unterstützt neben C und C++ noch mehrere andere Programmiersprachen, enthält einen Präprozessor, Assembler, Linker und lässt sich sehr genau konfigurieren. Er ist der Compiler, der wohl am meisten auf verschiedene Hardwareplattformen portiert worden ist.

Eine detailierte Beschreibung zur Handhabung des GCC, die von Richard M. Stallman, dem Hauptentwickler geschrieben wurde, gibt es online als Buch : http://gcc.gnu.org/onlinedocs/gcc/ .

4.4 Makefiles und GNU Make

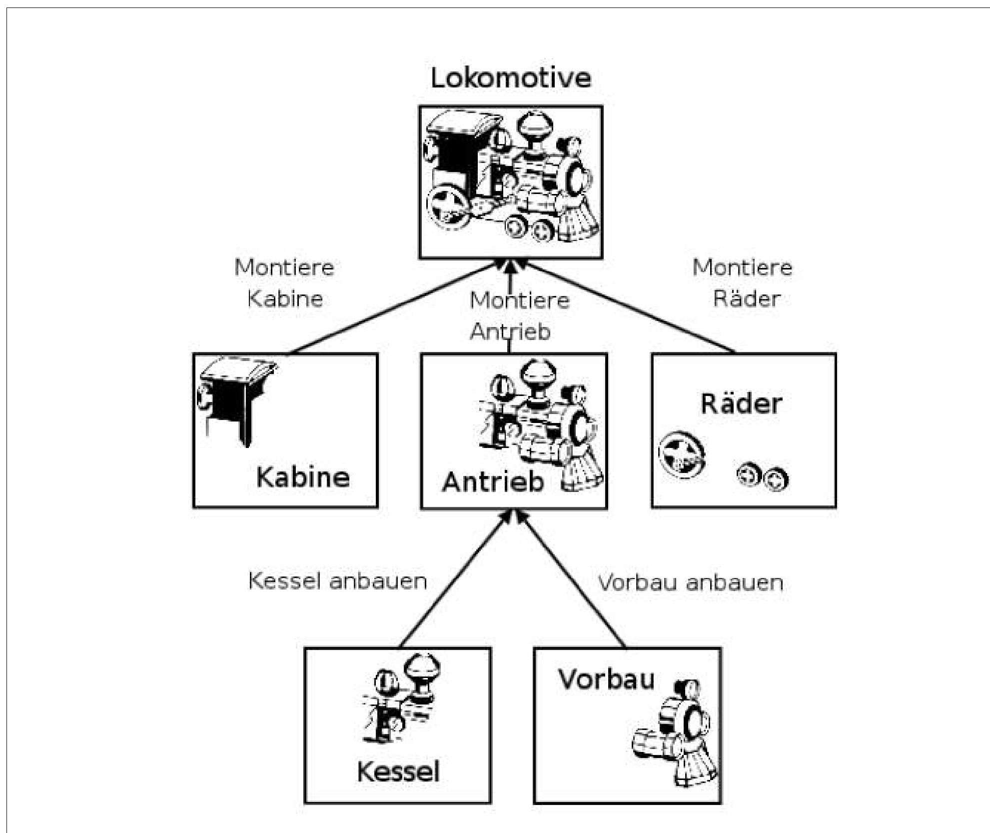
Das Make Programm dient zum automatisierten Verwalten und Erstellen von Dateien und kann dabei komplexe Abhängigkeiten berücksichtigen. Als Eingabe benötigt das Programm eine Textdatei, die im Normalfall den Namen "Makefile" trägt. Das Eingabeformat stellt an sich eine Programmiersprache dar und besteht aus folgenden Komponenten:

Das Hauptelement eines des Makefiles sind seine Regeln (engl. "**rules**"). Sie geben an, von der Existenz welcher Dateien oder Vorraussetzungen (den Prerequisites) zu erzeugende Dateien (die Targets) abhängig sind und welche Befehle (engl. "**commands**") ausgeführt werden sollen, um diese zu erstellen.

Die Syntax einer Regel sieht folgendermaßen aus:

```
<Liste der Targets> : <Liste der Prerequisites>
<Tab-Zeichen><Commands>
```

Am Beispiel der Konstruktion einer Lokomotive soll die Syntax verdeutlicht werden:



In die Sprache des Makefiles abgebildet ergeben sich folgende Regeln:

```
Lokomotive : Kabine Antrieb Räder
	Montiere Antrieb
	Montiere Räder
	Montiere Kabine

Antrieb : Kessel Vorbau
	Kessel anbauen
	Vorbau anbauen

Kabine :
	Baue Kabine
Räder :
	Baue Räder
Kessel :
	Montiere Kamin
	Montiere Ofen
Vorbau :
	Baue Grundgerüst
	Montiere Lampen
```

Natürlich handelt es sich bei den *targets* und *prerequisties* meistens um Dateien und bei den *commands* meistens um Programmaufrufe, wie z.B. den GNU Compiler. Eine typische, einfache Regel eines Makefiles

zum Erzeugen einer Applikation mit dem Namen "hallo" wäre:

```
hallo : hallo.c hallo.h
gcc -o hallo hallo.c
```

Variablen können mit

```
<Variablennname>=<Text>
```

deklariert und mit

```
$<Variablenname>
```

wieder aufgerufen werden.

Es gibt auch andere Zuweisungsoperatoren, wie z.B. :=, += und ?= .Innerhalb einer Regel können spezielle Variablen, die "**automatic variables**" , verwendet werden, von denen die wichtigsten hier aufgelistet sind:

- % = Im Namen des Targets angegeben, dient dieses Zeichen als Indikator, dass hier jede mögliche Zeichenkombination stehen kann. Der Teil, der beim konkreten Aufruf dieser Regel das %-Zeichen ersetzt wird Stamm genannt.
- $@ = Targetname der Regel
- $* = Stamm des passenden Targets
- $^ = Liste aller Prerequisites
- $< = Name des ersten Prerequisites

Auch Funktionen stehen zur Verfügung, wobei die Syntax folgendermaßen aussieht:

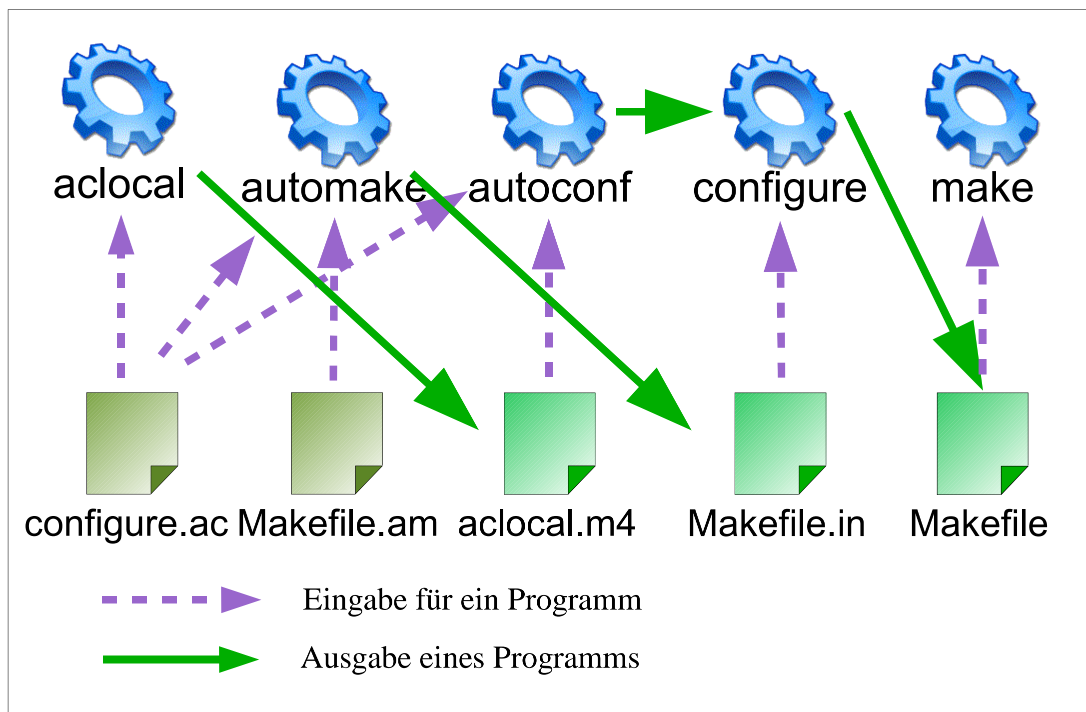
```
$(<Funktionsname> <Parameter1>,<Parameter2>,...)
```

Eine ausführliche Beschreibung von GNU Make findet sich Internet und in Richard Stallmanns Buch "GNU Make". Eine Online-Dokumentation befindet sich unter: http://www.gnu.org/software/make/manual/html_mono/make.html

4.5 GNU autotools

Das automatische Build-System von GNU besteht aus den beiden Anwendungen *automake* und *autoconf*, sowie einigen zusätzlichen Scripten und Programmen, wie aclocal und libtool. Libtool ist zum Beispiel für die portable Erstellung von Objektdateien, Bibliotheken (statisch und dynamisch) und Executables verwendet wird. Diese Programme dienen dazu den Erstellungsprozess durch Automation zu vereinfachen und durch automatische Tests portabel zu machen.

Das Zusammenwirken der einzelnen Tools ist sehr komplex und nicht einfach zu durchschauen, da das System nach und nach gewachsen ist und das Augenmerk immer auf Portabilität und nicht auf Einfachheit gerichtet war. Das folgende Diagramm zeigt grob auf, wie die einzelenen Programme zusammenarbeiten.



Das Programm *aclocal* erzeugt mithilfe der Datei *configure.ac* die Datei *aclocal.m4*. Darin sind Makros enthalten, die die Erstellung des configure Scripts steuern. Das Programm *autoconf* erstellt aus diesen Makros eine Scriptdatei mit dem Namen *configure*. Es ist auch möglich, eigene Makros zu definieren. Das *configure* Script führt Tests aus, die auf benötigte Komponenten und Funktionen des Betriebssystems prüfen. Bei Erfolg wird ein weiteres Script namens *config.status* erstellt (nicht im Diagramm), mit dem ein *Makefile* erzeugt wird. Das *configure* Script führt das *config.status* Script gleich anschließend aus. Neben den Informationen über das Betriebssystem,die aus den Tests gewonnen werden, werden auch Informationen über die Abhängigkeiten der Quelldateien und den Erstellungsablauf benötigt. Dazu dient das Programm *automake*, das Informationen über benötigte Dateien und deren Abhängigkeiten aus der Datei *Makefile.am* entnimmt. Als Ausgabe erzeugt *automake* die Datei *Makefile.in*, die vom *configure* Script genutzt wird.

Die Syntax der ***autoconf*** Eingabedatei (meist *configure.ac* oder *configure.in*) wird ausführlich im autoconf Manual erklärt, das auch online zugänglich ist:
http://www.gnu.org/software/autoconf/manual/autoconf-2.57/html_mono/autoconf.html

Wichtige Makros sind:

- `AC_INIT, AM_INIT_AUTOMAKE` : Initialisierung und Koniguration für autoconf und automake
- `ARG_ENABLE(...)` : Angabe von Optionen, die dem configure Script mitgegeben werden können und Definition ihrer Funktion.
- `AC_CHECK_PROG(...), AC_PROG_<...>` : Test, ob ein benötigtes Programm auf dem System vorhanden ist.
- `AC_CHECK_LIB(...)` : Test, ob eine benötigte Bibliothek auf dem System, und bestimmte Einsprungspunkte in der Bibliothek vorhanden sind.
- `AC_CHECK_HEADER(...), AC_HEADER_<...>` : Test, ob eine benötigte Header-Datei auf dem System vorhanden ist.

- `AC_OUTPUT(...)`: Definition der Ausgabedateien.

In der Homag AG kommt zur Konfigurierung eines größeren C-Projektes zur Werkzeugverwaltung auf einer Homag-CNC Maschine folgende *configure.ac* zum Einsatz:

```
# autoconf / automake initialisation
# set PACKAGE defines and init autoconf
AC_INIT(wzm,4.33)
# check for host/build/target aliases/triplets
AC_CANONICAL_SYSTEM
# check if sources are available
AC_CONFIG_SRCDIR(../../src/wzm_main.c)
# calling macros for automake initialisation
AM_INIT_AUTOMAKE
# require autoconf version 2.59 or higher
AC_PREREQ(2.59)
# _GNU_SOURCE determines that libc headers provide standard GNU system
# interfaces
AC_DEFINE(_GNU_SOURCE)

#checks for programs
AC_PROG_CPP(i686-pc-linux-gnu-gcc)
AC_PROG_CC(i686-pc-linux-gnu-gcc)
AC_PROG_CXX(i686-pc-linux-gnu-g++)
AC_PROG_LD(i686-pc-linux-gnu-ld)
AC_PROG_LN_S
AC_PROG_LIBTOOL
AC_PROG_MAKE_SET

#checks for libraries
save_LDFLAGS=$LDFLAGS
LDFLAGS="-L/usr/local/linux/i686-pc-linux-gnu/lib \
      -L../../../../../ACE_wrappers_vob/ACE_wrappers/build/linux/lib \
      -
L../../../../../ACE_wrappers_vob/HomagCorbaUtils/release/i686_linux32 \
      -L../../../../../Corba_vob/DIAG_IDL/release/i686_linux32"

AC_CHECK_LIB([stdc++],[main],[],
      [AC_MSG_ERROR([libstdc++.so needed but not found.])])
AC_CHECK_LIB([rt],[shm_open],[],
      [AC_MSG_ERROR([librt.so needed but not found.])])
AC_CHECK_LIB([m],[cos],[],
      [AC_MSG_ERROR([libm.so needed but not found.])])

AC_CHECK_LIB([ACE],[main],[],
      [AC_MSG_ERROR([libACE.so needed but not found.])])
AC_CHECK_LIB([ACEXML],[main],[],
      [AC_MSG_ERROR([libACEXML.so needed but not found.])])
AC_CHECK_LIB([ACEXML_Parser],[main],[],
      [AC_MSG_ERROR([libACEXML_Parser.so needed but not found.])])
AC_CHECK_LIB([TAO],[main],[],
      [AC_MSG_ERROR([libTAO.so needed but not found.])])
AC_CHECK_LIB([TAO_ETCL],[main],[],
      [AC_MSG_ERROR([libTAO_ETCL.so needed but not found.])])
AC_CHECK_LIB([TAO_DynamicAny],[main],[],
      [AC_MSG_ERROR([libTAO_DynamicAny.so needed but not found.])])
AC_CHECK_LIB([TAO_PortableServer],[main],[],
      [AC_MSG_ERROR([libTAO_PortableServer.so needed but not found.])])
AC_CHECK_LIB([TAO_Svc_Utils],[main],[],
      [AC_MSG_ERROR([libTAO_Svc_Utils.so needed but not found.])])
AC_CHECK_LIB([TAO_IORTable],[main],[],
      [AC_MSG_ERROR([libTAO_IORTable.so needed but not found.])])
AC_CHECK_LIB([TAO_Messaging],[main],[],
      [AC_MSG_ERROR([libTAO_Messaging.so needed but not found.])])
```

```
AC_CHECK_LIB([TAO_CosNaming],[main],[],
      [AC_MSG_ERROR([libTAO_CosNaming.so needed but not found.])])
AC_CHECK_LIB([TAO_CosNotification],[main],[],
      [AC_MSG_ERROR([libTAO_CosNotification.so needed but not found.])])
AC_CHECK_LIB([HomagCorbaUtils],[main],[],
      [AC_MSG_ERROR([libHomagCorbaUtils.so needed but not found.])])
AC_CHECK_LIB([DIAG_IDL],[main],[],
      [AC_MSG_ERROR([libDIAG_IDL.so needed but not found.])])

# Checks for typedefs, structures, and compiler characteristics.
AC_HEADER_STAT
AC_HEADER_STDBOOL
AC_C_CONST
AC_TYPE_UID_T
AC_TYPE_PID_T
AC_TYPE_SIZE_T
AC_HEADER_TIME
AC_STRUCT_TM
AC_C_VOLATILE

#checks for compiler characteristics
AC_PROG_CPP
AC_C_CONST
AC_C_INLINE
AC_PROG_GCC_TRADITIONAL
AC_PROG_CXXCPP

# Checks for library functions.
AC_FUNC_CLOSEDIR_VOID
AC_FUNC_ERROR_AT_LINE
AC_FUNC_FORK
AC_PROG_GCC_TRADITIONAL
AC_FUNC_LSTAT
AC_FUNC_LSTAT_FOLLOWS_SLASHED_SYMLINK
AC_FUNC_MALLOC
AC_FUNC_MEMCMP
AC_FUNC_MKTIME
AC_FUNC_MMAP
AC_FUNC_REALLOC
AC_FUNC_SELECT_ARGTYPES
AC_FUNC_SETVBUF_REVERSED
AC_TYPE_SIGNAL
AC_FUNC_STAT
AC_FUNC_STRFTIME
AC_FUNC_STRTOD
AC_FUNC_VPRINTF
AC_CHECK_FUNCS([atexit bzero floor ftime ftruncate gethostbyname
gethostname gettimeofday inet_ntoa localtime_r memmove memset mkdir munmap
putenv select setlocale socket strchr strdup strerror strrchr strstr
strtol strtoul tzset uname])

AC_CHECK_LIB(pthread,pthread_create,[g_cv_libpthread=yes],
[g_cv_libpthread=no])
if test $g_cv_libpthread=yes; then
      AC_DEFINE(LIBPTHREAD_EXISTS)
      PTHREAD_LIBS="-lpthread"
else
      AC_MSG_WARN([Pthread library is not supported])
fi

LDFLAGS="$save_LDFLAGS"
AC_OUTPUT(Makefile)
```

Anmerkung: Durch Angabe des Cross-Compilers wird auf dessen Vorhandensein überprüft und dieser auch als Standardcompiler ausgewählt.

Das ***automake*** Tool stellt die Möglichkeit zur Verfügung, das Erstellen von Programmen zu automatisieren, dabei wird der Entwickler in folgenden Hauptbereichen unterstützt:

- Erstellen von Programmen mit Angabe der benötigten Quelldateien, Programmaufrufen und Abhängigkeiten im Ablauf der Erstellung der Programmteile (siehe auch GNU Make).
- Testen der Programme mit Testprogrammen
- Löschen von nicht mehr benötigten Dateien, die beim Erstellen erzeugt werden
- Installation und Deinstallation des Programms
- Veröffentlichung des Programms

In Eingabedateien für *automake* kann sowohl GNU Make Syntax, als auch bestimmte Variablen genutzt werden, die dann von automake in normalen *Makefile* Code umgewandelt wird.

Eine einfache Datei *Makefile.am* könnte folgendermaßen aussehen:

```
1: bin_PROGRAMS = true false
2: false_SOURCES =
3: false_LDADD = false.o
4:
5: true.o: true.c
6: $(COMPILE) -DEXIT_CODE=0 -c true.c
7:
8: false.o: true.c
9: $(COMPILE) -DEXIT_CODE=1 -o false.o -c true.c
```

In Zeile 1 werden die zu erstellenden Programme angegeben. Über die `SOURCES` Variable kann für jedes Programm angegeben werden, welche Quellcodedateien benötigt werden. In diesem Fall kann *automake* aus den Regeln von Zeile 5 bis 9 selbst die benötigten Dateien ermitteln. Mit der `LDADD` Variable können Bibliotheken angegeben werden, die beim Linken des Programms hinzugefügt werden sollen.

Für eine detailierte Beschreibung von automake sei auf die offizielle Dokumentation im Internet verwiesen: http://www.gnu.org/software/automake/manual/html_mono/automake.html

Als Beispiel aus dem Produktivbereich ist hier das Makefile.am des Werkzeugverwaltungsprogramms der Homag AG aufgeführt:

```
## grouped source files
source = \
  ../../src/corba/OrbRunner.cpp \
  ../../src/corba/SignalHandler.cpp \
  ../../src/d_kflib.c \
  ../../src/wzm_db.c \
  ../../src/wzm_diag_recorder.cpp \
  ../../src/wzm_isg.c \
  ../../src/wzm_main.c \
  ../../src/wzm_sps.c \
  ../../src/wzm_tool_life.cpp

hmg_util = \
  ../../../cnc_kern/awd/os9_nc/hmg_util/get_h1.c \
  ../../../cnc_kern/awd/os9_nc/hmg_util/pr_vers.c

awd_util = \
  ../../../cnc_kern/std/awd_util/awd_util.c

util = \
  ../../../cnc_kern/std/fblock/util/isg_lib.c \
```

```
    ../../../cnc_kern/std/fblock/util/nc_util.c

os_spez = \
    ../../../cnc_kern/awd/os9_nc/os_spez/link_linux.c \
    ../../../cnc_kern/awd/os9_nc/os_spez/os_linux.c \
    ../../../cnc_kern/awd/os9_nc/os_spez/term20_c.c \
    ../../../cnc_kern/awd/os9_nc/os_spez/univ_w32.c \
    ../../../cnc_kern/std/isg_util/os_spez/shm_linux.c \
    ../../../cnc_kern/std/isg_util/os_spez/sleep.c

## executable to produce
bin_PROGRAMS = nc85_wzm_corba

## compiler flags for gcc
nc85_wzm_corba_CPPFLAGS = \
    -I$(srcdir)/../../include \
    -I$(srcdir)/../../../cnc_kern/std/fblock/include \
    -I$(srcdir)/../../../cnc_kern/awd/os9_nc/include \
    -I$(srcdir)/../../../../woodwop_vob \
    -I$(srcdir)/../../../../WoodWOP_vob/ng40/include \
    -I$(srcdir)/../../../../woodwop_vob/ng40/ext_include \
    -I$(srcdir)/../../../../woodwop_vob/framwork/includes \
    -I$(srcdir)/../../../../woodwop_vob/data/inc \
    -I$(srcdir)/../../../../ACE_wrappers_vob/HomagCorbaUtils/interface \
    -I$(srcdir)/../../../../Corba_vob/DIAG_IDL/interface \
    -I$(srcdir)/../../../../ACE_wrappers_vob/ACE_wrappers \
    -I$(srcdir)/../../../../ACE_wrappers_vob/ACE_wrappers/TAO \
    -I$(srcdir)/../../../../ACE_wrappers_vob/ACE_wrappers/TAO/orbsvcs \
    -DCPU=I80486 \
    -DGCM_SIMU=0 \
    -DNT_DRIVER=0 \
    -DRTDRIVER=0 \
    -DANSI_DEKL=1 \
    -DHAVE_CORBA \
    -DPC85=1 \
    -DNC84=1 \
    -DENT=1 \
    -DFM_NEU_STD=1 \
    -DCOMPILER_GNU_CC \
    -DHMG_LINUX

## needed source files
nc85_wzm_corba_SOURCES = \
    $(source) \
    $(hmg_util) \
    $(awd_util) \
    $(util) \
    $(os_spez) \
    ../../include/config-win32.h \
    ../../include/config_hawk_ppc.h \
    ../../include/d_kflib.h \
    ../../include/wz_db.h \
    ../../include/wz_db85.h \
    ../../include/wzm_db.h \
    ../../include/wzm_defs.h \
    ../../include/wzm_diag_recorder.h \
    ../../include/wzm_func.h \
    ../../include/wzm_isg.h \
    ../../include/wzm_main.h \
    ../../include/wzm_other_func.h \
    ../../include/wzm_sps.h \
    ../../include/wzm_tdef.h \
    ../../include/wzm_texte.h \
    ../../include/wzm_tool_life.h \
    ../../include/wzm_var.h \
    ../../src/corba/OrbRunner.h \
```

```
  ../../src/corba/SignalHandler.h

## linker flags
nc85_wzm_corba_LDFLAGS = \
 -L../../../../../ACE_wrappers_vob/ACE_wrappers/build/linux/lib
-L../../../../../ACE_wrappers_vob/HomagCorbaUtils/release/i686_linux32
-L../../../../../Corba_vob/DIAG_IDL/release/i686_linux32

## needed libraries
nc85_wzm_corba_LDADD = \
 -lACE -lACEXML -lACEXML_Parser -lTAO_Messaging -lTAO -lTAO_CosNaming
-lTAO_CosNotification -lTAO_Svc_Utils -lTAO_IORTable -lTAO_PortableServer
-lTAO_DynamicAny -lTAO_ETCL -lHomagCorbaUtils -lDIAG_IDL -lrt -lm

ACLOCAL = @ACLOCAL@
ACLOCAL_AMFLAGS = -I m4

## Clean up template repositories, etc.
clean-local:
      -rm -f *~ *.bak *.rpo *.sym lib*.*_pure_* core core.*
      -rm -f gcctemp.c gcctemp so_locations *.ics
      -rm -rf cxx_repository ptrepository ti_files
      -rm -rf templateregistry ir.out
      -rm -rf ptrepository SunWS_cache Templates.DB
```

Ein ausführliches Buch zu den meisten Aspekten der GNU Toolchain ist auch online verfügbar :
http://sources.redhat.com/autobook/autobook/autobook.html

4.6 Aufruf GNU Toolchain

Es hat sich als praktisch erwiesen ein Script für den Build Vorgang der Sourcen zu schreiben, das als Ergebnis ein Makefile liefert. Hier ist wieder exemplarisch das Werzeugverwaltungsprojekt der Homag AG aufgeführt:

```
 1    libtoolize
 2    aclocal
 3    automake --add-missing --foreign
 4    autoconf
 5
 6    mkdir -p debug
 7    cd debug
 8    ../configure CPPFLAGS="-x c++" CXXFLAGS="-g -O0 -Wall" \
 9        --program-suffix=_d --build=i686-pc-cygwin --host=i686-pc-linux-gnu \
10        --disable-static
11    cd ..
```

In Zeile 6 wird ein neues Verzeichnis angelegt und in diesem Verzeichnis wird das configure Script aufgerufen. Dadurch ist es möglich, verschiedene Konfigurationen (z.B. debug / release) desselben Programms zu erstellen.

Es ist wichtig, hier die host und build Bezeichnungen nicht durcheinanderzubringen: Bei normalen Applikationen ist das Build-System das System, auf welchem der Build Vorgang stattfindet, während das Host-System das Sytem ist, auf dem die Applikation später ausgeführt werden soll.