automake, autoconf, libtool

Benny Siegert ‹bsiegert@gmail.com›
The MirOS Project (http://www.mirbsd.org)

FrOSCon 2007

Gliederung

1. Historischer Überblick
2. Vorstellung der einzelnen Programme
3. Ein einfaches Beispiel
4. Schlechte Beispiele
5. Zusammenfassung

Makefiles

- seit Version 7 AT&T UNIX (ca. 1977)
- Ziele (Dateinamen) mit Abhängigkeiten und Compiler-Kommandos
- Pseudo-Dateinamen wie „install”

```
# extrem simples Makefile
foo: foo.o
	cc -o foo foo.o

foo.o: foo.c
	cc -c -o foo.o foo.c

install: foo
	install -m 555 foo /usr/local/bin
```

… End of story?

Problem: *Optionen*

- Compilerflags, spezielle Anpassungen
- Pfade: Header, Installationsverzeichnis?

➡ Benutzer muss Optionen *anpassen* können

- manuell Makefiles bearbeiten:
  heute unzumutbar

imake

- Lösung bei X11
- Imakefile mit C-Präprozessor bearbeitet ergibt Makefile
- einleuchtende Idee, aber in der Praxis ein Alptraum
- Xorg benutzt jetzt autotools, sie wissen warum

Schily-Make

- die Lösung von Jörg Schilling (cdrtools)
- hat ein riesiges Verzeichnis voller Options-Makefiles, eins für jede Plattform
- funktioniert nicht mit BSD Make
- beleidigt als erstes den Benutzer („*ich* finde, du hast das falsche make”)
- wegen Lizenz: niemand außer Jörg Schilling kann den cdrtools-Source legal vertreiben

Das GNU-Buildsystem

- heute fast ubiquitär im Einsatz
- einfach für den Benutzer:

```
% ./configure ; make ; make install
```

- nur wenige Entwickler verstehen das System
- immer mehr „einfachere" Alternativen
- Wirklich unnötig kompliziert?
  - Nein, denn ein *portables* Buildsystem ist schwierig!

Vorstellung der einzelnen Programme

configure

- verarbeitet Optionen (Zielverzeichnis, Compilerflags, Module?)
- sucht Libraries, Header, Pfade usw.
- erstellt config.status
- config.status handhabt *Ersetzungsvariablen* (z.B. CC, CFLAGS, LIBS)

autoconf

- configure.ac enthält Makroaufrufe, die in ein Shell-Skript gewandelt werden
- Makros, die nicht zu autoconf gehören, kommen aus aclocal.m4
- Dokumentation: info autoconf

autoheader

- erstellt automatisch eine Vorlage für config.h
- config.h enthält Präprozessorsymbole, z.B. HAVE_FOO_H
- Im Quellcode: am Anfang jeder Datei #include “config.h”
- Dokumentation: info autoheader

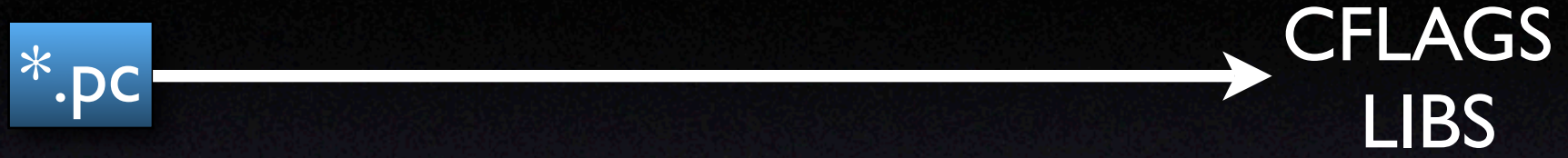
automake

- nimmt dem Entwickler die Aufgabe ab ein portables Makefile zu schreiben
- Makefile.am ist kurz und simpel
- Makefile.in sieht dann furchterregend aus (Tipp: einfach nicht anfassen)
- Dokumentation: info automake

aclocal

- bei automake dabei
- macht autoconf lokale Makros (auch die von automake!) bekannt
- nur die, die auch verwendet werden, landen in aclocal.m4
- acinclude.m4 für eigene Makros
- Dokumentation: info aclocal

pkgconfig



- erleichtert Auffinden und Verwenden von Shared Libraries
- ersetzt gtk-config und viele andere
- `PKG_CHECK_MODULES(FOO, glib2.0 >= 2.8.0)` setzt FOO_CFLAGS, FOO_LIBS
- Dokumentation: man pkgconfig

libtool

- Unterstützung für statische/Shared Libraries auf allen Plattformen
  - kapselt plattformspezifische Kommandos
- Man tut so, als würde man statische Libraries mit Endung .la bauen (sog. *libtool libraries*)
- Kommandos mit `libtool --mode=foo` präfixen
- libtool *verändert* Kommandos
- Dokumentation: info libtool (außer Mac OS)

Weitere

autom4te: Wrapper um m4, von autoconf verwendet

- Bei Problemen: autom4te.cache löschen

autoscan: erstellt automatisch configure.ac

libtoolize, gettextize: libtool- bzw. gettext-Dateien hinzufügen

autoreconf: notwendige Dateien regenerieren

Beispiel:
ein einfaches Projekt

Quelltext

```c
#include <stdio.h>
#include "hello.h"

void hello()
{
   printf("Hello World!\n");
}
```

hello.c

```c
void hello();
```

hello.h

```c
#include "hello.h"

int main(int argc, char **argv)
{
   hello();
   return 0;
}
```

main.c

Traditionelles Makefile

```
hw: main.o libhello.a
        cc -o hw main.o libhello.a

libhello.a: hello.o
        ar cru libhello.a hello.o
        ranlib libhello.a

.SUFFIXES: .c .o
.c.o:
        cc -c -o $@ $<

clean:
        rm -f hw main.o hello.o libhello.a
```

Makefile mit libtool

```
hw: main.lo libhello.la
	./libtool --mode=link cc -o hw main.lo libhello.la

libhello.la: hello.lo
	./libtool --mode=link cc -rpath /usr/local/lib \
	-o libhello.la hello.lo

.SUFFIXES: .c .lo
.c.lo:
	./libtool --mode=compile cc -c -o $@ $<

clean:
	./libtool --mode=clean rm -f hw main.lo hello.lo \
	libhello.la
```

autoconf + automake

- Für mehr Übersicht: Quellcode nach src/
- „autoreconf -v -i” aufrufen

```
AC_INIT(hw, 0.1)
AM_INIT_AUTOMAKE(foreign)

AC_PROG_CC
AC_STDC_HEADERS

AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

configure.ac

```
SUBDIRS = src
```

Makefile.am

```
bin_PROGRAMS = hw
hw_SOURCES = main.c hello.c hello.h
```

src/Makefile.am

Benutzen von config.h

```
AC_INIT(hw, 0.1)
AM_INIT_AUTOMAKE
AC_CONFIG_HEADERS(config.h)

[…]
```

configure.ac

- autoheader aufrufen
- Damit config.h auch benutzt wird:
  in jeder Quelltextdatei oben ergänzen

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif
```

… Demo …

Features (bis jetzt)

- benutzt CC und CFLAGS des Benutzers
- Installation (make install), Deinstallation
  - Zielverzeichnis kann angegeben werden
- Möglichkeit ein Build-Verzeichnis zu nutzen
- Dependency Tracking (deaktivierbar)
- Tarball erstellen (make dist)
- make clean

libtool!

- Ziel wie vorher: libhello als Shared Library

- libtool wird von configure erstellt (Makro AC_PROG_LIBTOOL)
- libtool.m4 und ltmain.sh aus dem libtool-Source kopieren (oder „libtoolize --copy”)
- am Ende wieder „autoreconf -v -i” aufrufen

```
[…]
AC_PROG_CC
AC_STDC_HEADERS
AC_PROG_LIBTOOL
[…]
```

configure.ac

```
ACLOCAL_AMFLAGS = -I .

SUBDIRS = src
```

Makefile.am

```
bin_PROGRAMS = hw
hw_SOURCES = main.c
hw_LDADD = libhello.la

lib_LTLIBRARIES = libhello.la
libhello_la_SOURCES = hello.c hello.h
```

src/Makefile.am

Ersetzungsvariablen

- Ziel: Versionsinfo von libhello soll im configure gesetzt werden

- AC_SUBST(*variable*) ersetzt @*variable*@ durch den Inhalt der Variablen
- automake definiert *variable* automatisch für uns
- -version-info in die LDFLAGS der Library

current:revision:age

- Versionsinfo für libtool-Libraries
- drei Zahlen $\in \mathbb{N}_0$
- *current* ist die aktuelle API-Versionsnummer
- *revision* steht für Bugfixes o.ä.
  ohne Auswirkungen auf das API
- *age* gibt an, zu wie vielen Versionen wir
  binärkompatibel sind

- Bei Bugfixes usw.: *revision* erhöhen
- Funktion hinzugefügt:
  current und *age* erhöhen, *revision* auf 0
- Funktion entfernt oder anderer Prototyp:
  current erhöhen, *age* und *revision* rücksetzen

```
[…]
LIBHELLO_CURRENT=1
LIBHELLO_REVISION=0
LIBHELLO_AGE=1

LIBHELLO_VER=$LIBHELLO_CURRENT:\
$LIBHELLO_REVISION:$LIBHELLO_AGE
AC_SUBST(LIBHELLO_VER)

AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

configure.ac

```
lib_LTLIBRARIES = libhello.la
libhello_la_SOURCES = hello.c
libhello_la_LDFLAGS = \
   -version-info ${LIBHELLO_VER}
```

src/Makefile.am

Fortgeschrittenes

Header-Files

- AC_CHECK_HEADERS überprüft, ob Header-File vorhanden ist *und funktioniert*
- Falls ein Header einen anderen benötigt, vierten Parameter verwenden

```
[…]
AC_CHECK_HEADERS([sys/param.h sys/mount.h],
[], [], [#ifdef HAVE_SYS_PARAM_H
#include <sys/param.h>
#endif
])
[…]
```

configure.ac

configure-Argumente

- Zusätzliche Features (--enable-*feature*) oder externe Pakete (--with-*package*)
- AC_ARG_ENABLE oder AC_ARG_WITH, funktionieren gleich
- Argumente: Name, Hilfe-String und zwei Shellfragmente (angegeben, nicht angegeben)
- AS_HELP_STRING formatiert diesen

Einfaches Beispiel

```
[…]
AC_ARG_ENABLE([debug],
   AS_HELP_STRING([--enable-debug],
      [Enable debugging code (default=no)]),
   [enable_debug=$enableval],
   [enable_debug=no])
if test "x$enable_debug" = xyes; then
   AC_DEFINE(DEBUG, 1, [Enable debugging code])
fi
[…]
```

configure.ac

- AC_DEFINE: Setzt ein Präprozessorsymbol (in config.h)

Komplexes Beispiel

```
AC_ARG_WITH([glib2],
    AS_HELP_STRING([--with-glib2], [Enable support for glib 2.0
@<:@default=auto@:>@]),
    [with_glib2=$withval],
    [with_glib2=auto])
if test "x$with_glib2" != xno; then
    PKG_CHECK_MODULES(GLIB, [glib-2.0 >= 2.8], [have_glib2=yes],
[have_glib2=no])
else
    have_glib2=no
fi
if test "x$with_glib2" = xyes -a "x$have_glib2" = xno; then
    AC_MSG_ERROR([Library requirements (glib-2.0 >= 2.8) not met;
consider adjusting the PKG_CONFIG_PATH environment variable if your
libraries are in a nonstandard prefix so pkg-config can find them.])
fi
if test "x$have_glib2" = xyes; then
    AC_DEFINE([HAVE_GLIB2], [1], [Define if you have the glib2 library.])
fi
```

Wie man es *nicht*
machen sollte …

Bakefile

- Ersatz für automake
- wird von wxWidgets benutzt
- pkgsrc hat gepatchte wxWidgets-Makefiles, die libtool benutzen: 3 MiB großer Patch!
- Beispiel-Bakefile, aus dem Tutorial:

```
<?xml version="1.0"?>

<makefile>
  <include file="presets/simple.bkl"/>

  <exe id="hello" template="simple">
    <sources>hello.c</sources>
  </exe>
</makefile>
```

wx-config

- sehr komplexes Skript statt .pc-Datei(en)
- muss hinter configure herräumen:

```
# We evidently can't trust people not to duplicate things in
# configure, or to keep them in any sort of sane order overall,
# so only add unique new fields here even if it takes us a while.
[…]
# This is not a licence
# for sloppy work elsewhere though and @GUI_TK_LIBRARY should
# be fixed.
[…]
# of course, this duplication is bad but I'll leave to somebody else the care
# of refactoring this as I don't see any way to do it - VZ.

# This (and the other cruft to support it) should be removed with
# reference to the FIXME above when configure stops piping us a slurry
# of options that need to be decomposed again for most practical uses - RL.
```

„Plattformliste”

- autoconf soll helfen, von langen Listen mit Plattformen und passenden Optionen loszukommen
- einige configures tun das trotzdem

```
case "${host}" in
[…]
  *-*-openbsd*)
    USE_BSD=1
    USE_OPENBSD=1
    AC_DEFINE(__OPENBSD__)
    AC_DEFINE(__BSD__)
    DEFAULT_DEFAULT_wxUSE_GTK=1
  ;;

  *)
    AC_MSG_ERROR(unknown system type ${host}.)
esac
```

Makefiles von a2ps

- Letztes Release (4.13b) war 2000
- benutzt autoconf 2.14a, nie offiziell released
- verwendet nicht aclocal, sondern:

```
## I use special autoconf/make macros
ACLOCAL_AMFLAGS = --version >/dev/null && cat m4/*.m4 >aclocal.m4
```

- bringt sein eigenes automake,
  verteilt auf 17 (!) m4-Dateien, mit

Conclusio

Die goldenen Regeln

1. Nicht alle haben Linux i386, gcc, GNU Make
2. Nicht das Rad neu erfinden wollen
3. Nicht hardcoden, was getestet werden kann
4. Die autoconf-Infrastruktur muss vom Benutzer regenerierbar sein
5. Keine generierten Files verändern