

# autoconf und automake

Andreas Krennmair

ak@synflood.at

7. Mai 2004

## Zusammenfassung

Praktisch jede Software unter Linux wird mit `./configure ; make ; make install` konfiguriert, kompiliert und installiert. Ich möchte heute zeigen, wie man selbst mit Autoconf und Automake den Konfigurations- und Übersetzungsprozess automatisieren kann, und was das für portable Software bringen kann.

## Agenda

- Einführung
- Makefiles
- Autoconf alleine
- Autoconf und Automake

## Einführung: Geschichte der automatisierten Compilierung unter Unix

- Anfangs: Shellskripte namens `make` im Sourceverzeichnis (bis Unix V6)
- Ab Unix V7: Programm `make`, Information aus Datei `makefile`
- `makefile` enthält Informationen, welche Dateien (Targets) aus welchen Dateien (Dependencies) erzeugt werden, und welche Kommandos dazu aufgerufen werden müssen.
- Makefiles funktionieren anfangs ganz gut, bis die ersten Unix-Varianten erschienen, die subtil anders als die bisherigen Unixe waren. Software sollte aber portabel bleiben.
- Einfache Lösung: ein Target pro Zielplattform oder ein Makefile pro Zielplattform. Nachteil: bei mehr Plattformen ansteigender Wartungsaufwand

## Einführung: erste automatisierte Konfiguration

- Makefile-Lösung hörte auf zu skalieren, als immer mehr und immer obskurre Unix-Varianten auftauchten, auf die die Entwickler von Software noch dazu keinen Zugriff mehr hatten.
- Erste Lösung Mitte bis Ende der 80er Jahre: `Configure`
- Larry Wall wollte, dass seine Software (insbesondere Perl) portabel auf möglichst vielen Unix-Plattformen läuft
- Schrieb Shellskript `Configure`, das Informationen über das System sammelte, und aus `*.SH`-Dateien dementsprechende Dateien generierte (`Makefile.SH` → `Makefile`)
- Vorteil: Perl konnte ohne grossen Portierungsaufwand auf vielen, teilweise recht obskuren Unix-Systemen betrieben werden (“Congratulations. You aren’t running Eunice.”)

## Einführung: Konfiguration für GNU

- GNU-Software sollte möglichst portabel sein
- GNU-Projekt griff Larry Wall's Idee auf, und realisierte im wesentlichen zwei Frameworks, um das Konfigurieren und Übersetzen von GNU-Software möglichst portabel und einfach wartbar zu halten.
- Konfiguration: `autoconf`
- Übersetzung: `automake`
- Status heute: Autoconf und Automake sind ein Quasi-Standard bei Freier Software/Open Source

## Einfache Makefiles

Makefiles bestehen im wesentlichen aus zwei Bereichen:

- Variablendefinitionen
- Zieldefinitionen

Variablen werden verwendet, um gleiche “Textbausteine”, die öfters im Makefile vorkommen, zusammenzufassen und parametrisierbar zu machen, z.B. Compilerkommando, Compilerflags, Ausgabedatei, ...

Zieldefinitionen geben an, welche Datei erzeugt werden soll, von welchen Dateien diese Datei abhängig ist, und mit welchem Kommando die Datei aus den Abhängigkeiten erzeugt wird. Diese Zieldefinition definiert ein sog. “Target”. Wird ein Target aufgerufen, so wird das Kommando nur ausgeführt, wenn die zu generierende Datei noch nicht existiert, oder wenn eine der Abhängigkeiten erst generiert werden muss, oder wenn eine der Abhängigkeiten neuer ist als die bestehende Datei. So werden unnötige Compile-Vorgänge vermieden.

## Beispiel 1: einfaches Makefile

```
# Kommentar
LATEX=pdflatex           # Variablendefinition

ac-am.pdf: ac-am.tex     # Zieldefinition
    $(LATEX) ac-am.tex   # <Tabulator>Kommando
```

## Beispiel 2: gleichartige Targets zusammenfassen

```
LATEX=pdflatex
```

```
RM=rm -f
```

```
PDFFILES=ac-am.pdf
```

```
all: $(PDFFILES)
```

```
%.pdf: %.tex
```

```
    $(LATEX) $<
```

```
clean:
```

```
    $(RM) $(PDFFILES) *.aux *.log
```

```
.PHONY: all clean # phony targets werden nicht als Dateien generiert
```

## Beispiel 3: modulares C-Programm übersetzen

```
CC=gcc
```

```
CFLAGS=-Os -Wall
```

```
OBJS=foo.o bar.o baz.o quux.o
```

```
OUTPUT=xyzyzy
```

```
all: $(OUTPUT)
```

```
$(OUTPUT): $(OBJS)
```

```
$(CC) $(CFLAGS) $(LDFLAGS) -o $(OUTPUT) $(OBJS) $(LIBS)
```

```
%.o: %.c
```

```
$(CC) $(CFLAGS) $(DEFINES) -c $<
```

```
clean:
```

```
$(RM) $(OBJS) $(OUTPUT) core *.core
```

```
.PHONY: all clean
```

## Limitationen von Makefiles

Makefiles funktionieren zwar bei kleineren, einfachen Programmen, wer jedoch groessere, portable Software schreiben will, stößt mit `make` und Makefiles schnell an Grenzen.

Die Unterschiede zwischen den einzelnen Unix-Systeme sind z.B. folgende:

- Strukturen unterscheiden sich
- Funktionen sind unterschiedlich deklariert
- `#defines` sind anders benannt oder existieren nicht
- Manche Funktionen sind nicht mehr im der `libc`, sondern in externe Libraries ausgelagert (z.B. Sockets nach `libsocket`).

Auf diese Unterschiede kann `make` nicht eingehen. Deswegen muss man einen Konfigurationsmechanismus einführen, der dies kann.

## Autoconf, Schritt 1: configure-Skript erzeugen

Autoconf bietet die Möglichkeit, auf eine große Anzahl von Kommandos und Tests zurückzugreifen, um möglichst alle relevanten Systemparameter abzurufen. Diese Tests werden in einer Datei `configure.in` abgelegt, aus dem dann mit dem Kommando `autoconf` die Datei `configure` erzeugt wird. Mit dem Kommando `autoheader` wird die Datei `config.h.in` erzeugt.

Ruft man `./configure` auf, so sammelt das `configure`-Skript die Konfigurationsinformationen, und generiert aus `config.h.in` die Datei `config.h` sowie alle in `configure.in` angegebenen zu konfigurierenden Dateien, das ist meistens `Makefile.in`, aus der `Makefile` erzeugt wird.

Die `configure.in`-Datei lässt sich übrigens erzeugen, indem man `autoscan` aufruft, und die resultierende Datei `configure.scan` in `configure.in` umbenennt.

## Autoconf, Schritt 2: Makefile.in erstellen

Die Datei `Makefile.in` wird wie ein normales Makefile geschrieben, mit dem Unterschied, dass für bestimmte Variablen, deren Wert vom `configure`-Skript bestimmt werden, spezielle Platzhalter eingefügt werden. Das sieht dann z.B. so aus:

```
CC=@CC@
CFLAGS=@CFLAGS@ @DEFS@
LDFLAGS=@LDFLAGS@
LIBS=@LIBS@
```

Der Rest des Makefile sieht wie ein normales Makefile aus. Um auf sämtliche ermittelten Parameter zugreifen zu können, müssen die einzelnen C-Sourcefiles nur noch die Datei `config.h` inkludieren. Damit ist Autoconf vollständig integriert und das Buildsystem darauf angepasst.

## Autoconf, Zusammenfassung

In einer Minute<sup>1</sup> zur Sourcekonfiguration mit Autoconf:

- `$ autoscan && mv configure.scan configure.in – configure.in generieren`
- `$ $EDITOR configure.in – configure.in anpassen`
- `$ autoconf – configure erzeugen`
- `$ autoheader – config.h.in erzeugen`
- `$ $EDITOR Makefile.in – Makefile.in erzeugen`

---

<sup>1</sup>Wenn Makefile.in schon existiert ;-)

## Beispiel für configure.in (1)

```
AC_PREREQ(2.57)
AC_INIT(akpop3d, 0.7.7, ak@synflood.at)
AC_CONFIG_SRCDIR([authenticate.c])
AC_CONFIG_HEADER([config.h])

# Checks for programs.
AC_PROG_CC
AC_PROG_INSTALL
```

## Beispiel für configure.in (2)

```
# Checks for header files.
AC_HEADER_STDC
AC_HEADER_SYS_WAIT
AC_CHECK_HEADERS([arpa/inet.h fcntl.h limits.h netdb.h
  netinet/in.h shadow.h stdlib.h string.h sys/file.h
  sys/socket.h sys/time.h syslog.h unistd.h])

# Checks for typedefs, structures, and compiler
# characteristics.
AC_C_CONST
AC_TYPE_UID_T
AC_TYPE_OFF_T
AC_TYPE_PID_T
AC_TYPE_SIZE_T
AC_HEADER_TIME
```

## Beispiel für configure.in (3)

```
# Checks for library functions.
AC_FUNC_ALLOCA
AC_FUNC_FORK
AC_FUNC_REALLOC
AC_FUNC_SELECT_ARGTYPES
AC_FUNC_STAT
AC_CHECK_FUNCS([atexit dup2 getsipnam inet_ntoa
    memchr memset select socket strchr strerror
    strncasecmp strrchr])

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

## Weitere nützliche Autoconf-Funktionen

- `AC_CHECK_LIB(library, symbol)`: wenn `symbol` in Library `library` gefunden wird, wird `-llibrary` zu den `LDFLAGS` hinzugefügt und `HAVE_LIBLIBRARY=1` in `config.h` definiert.
- `AC_DEFINE([KEY], [VALUE], [Beschreibung])`: in `config.h` wird `#define KEY VALUE` eingetragen.
- `AC_ARG_WITH(option, [beschreibung])`: das `configure`-Skript um eine `-with-option` Option erweitern.
- `AC_ARG_ENABLE(option, [beschreibung])`: das `configure`-Skript um eine `-enable-option` Option erweitern.

## Beispiele zu nützlichen Autoconf-Funktionen

```
AC_ARG_WITH(openssl, [ --with-openssl use OpenSSL])
if test "$with_openssl" != "no" ; then
    AC_CHECK_LIB(crypto, BIO_new)
    AC_CHECK_LIB(ssl, SSL_new)
fi
AC_ARG_ENABLE(rfc2449, [ --enable-rfc2449 enable RFC 2449 support])
if test "$enable_rfc2449" != "no" ; then
    AC_DEFINE([ENABLE_RFC2449], [1], [rfc2449])
fi
```

## Funktionsweise von Autoconf

- Vorgefertigte Tests in Form von m4-Makros verfügbar
- Autoconf lässt configure.in durch m4 laufen, daraus entsteht configure-Skript, was nicht anderes als ein Shellskript ist.
- → man kann durch Einfügen von eigenem Shellcode eigene Tests durchführen.
- → oder man greift auf <http://ac-archive.sourceforge.net/> zurück, einem umfangreichen Archiv von hunderten Autoconf-Macros.

```
if test x`uname -s` = "xDarwin" ; then
  AC_DEFINE([HAVE_DARWIN],[1],[define whether we have Darwin])
fi
```

## Autoconf-Makros selbst schreiben

Autoconf-Makros sind ein Mischmasch aus m4-Skript und Shellskript.

```
AC_DEFUN([AC_C_LONG_LONG],
[AC_CACHE_CHECK(for long long int, ac_cv_c_long_long,
[if test "$GCC" = yes; then
  ac_cv_c_long_long=yes
else
  AC_TRY_COMPILE(, [long long int i;],
  ac_cv_c_long_long=yes,
  ac_cv_c_long_long=no)
fi])
if test $ac_cv_c_long_long = yes; then
  AC_DEFINE(HAVE_LONG_LONG)
fi
])
```

## Automake: Einführung

Automake ist dafür gedacht, den eigentlichen Übersetzungsprozess so weit wie möglich zu vereinfachen, und dem User das Schreiben von eigenen `Makefile.in`'s abzunehmen. Automake setzt Autoconf voraus.

Die `Makefile.am`-Datei besteht wie ein Makefile aus Targets, die quasi beliebig benannt werden können, und alle Kommandos enthalten wie auch ein Target in einem Makefile oder Makefile.in.

Zusätzlich existieren eine Reihe von speziellen Variablen, mit denen das Übersetzen von Software einfacher wird.

```
bin_PROGRAMS = hello
hello_SOURCES = hello.c main.c
EXTRA_DIST = hello.h
```

## Automake: mehr Kommandos

- `$ $EDITOR Makefile.am` – **Makefile.am erstellen**
- `$ autoscan && mv configure.scan configure.in`
- `$ autoheader`
- `$ aclocal`
- `AM_INIT_AUTOMAKE (programmname, version)` in `configure.in` eintragen
- `$ automake -a` – **Makefile.in aus Makefile.am generieren**
- `$ autoconf`

```
$ ls -l Makefile.in configure
-rw-r--r--  1 ak  staff   16048 16 Mar 20:03 Makefile.in
-rwxr-xr-x  1 ak  staff  123354 16 Mar 20:03 configure
$
```

## Automake: Primaries

Die `_PROGRAMS`, `_SOURCES`, etc. Suffixe, die wie vorher gesehen haben, nennen sie übrigens "Primaries". Weitere Primaries sind z.B.:

- `DATA`: gibt Datendateien an, die 1:1 mitinstalliert, ansonsten aber ignoriert werden
- `HEADER`: damit werden Headerfiles spezifiziert, die zusammen mit Libraries installiert werden sollen.
- `SCRIPTS`: ausführbare Skripte, die ebenfalls installiert werden, jedoch nicht kompiliert oder gestripped
- `MANS`: gibt Manpages an, die ebenfalls mitinstalliert werden.

Die Grundbedürfnisse für einfache und problemlose Konfigurations-, Übersetzungs- und Installationsroutinen wäre damit gedeckt.

## Rekursives Automake

Um den Inhalt von Unterverzeichnissen in den Automake-Vorgang miteinzubeziehen, muss man lediglich alle relevanten Unterverzeichnisse über die `$SUBDIRS`-Variable angeben.

```
SUBDIRS = m4 src doc
```

In jedem Unterverzeichnis muss natürlich wiederum eine `Makefile.am` angelegt und daraus eine `Makefile.in` erzeugt werden. Ausserdem muss das dann zu erzeugende `Makefile` in der `configure.in` angegeben werden, und zwar via `AC_CONFIG_FILES`.

## Resümee

- `make` mag veraltet und eingerostet wirken (wird seit Ende der 1970er eingesetzt), bietet aber ein mächtiges System, um Abhängigkeiten zu überprüfen, und unnötige Compilevorgänge zu minimieren.
- Autoconf bietet ein mächtiges System, um vielerlei systemabhängige Konfigurationspunkte in Erfahrung zu bringen, was wiederum einen Eckpfeiler für systemnahe und portable Programmierung bildet
- Automake macht es für Entwickler besonders einfach, Softwarepakete in eine Form zu bringen, dass sie übersetzt und installiert werden können
- Autoconf und Automake mögen suboptimale Lösungen sein (`./configure` dauert lange, `configure` und `Makefile.in` sind bei Automake extrem gross), stellen jedoch eine frei verfügbare, einfach anzuwendende und vor allem weit verbreitete Lösung dar

## Literaturempfehlungen

- Das “Autobook”: Autoconf, Automake and Libtool:  
[http://sources.redhat.com/autobook/autobook/autobook\\_toc.html](http://sources.redhat.com/autobook/autobook/autobook_toc.html)
- Autoconf Dokumentation:  
[http://www.delorie.com/gnu/docs/autoconf/autoconf\\_toc.html](http://www.delorie.com/gnu/docs/autoconf/autoconf_toc.html)
- Automake Dokumentation:  
[http://www.delorie.com/gnu/docs/automake/automake\\_toc.html](http://www.delorie.com/gnu/docs/automake/automake_toc.html)

**Und jetzt...**

**zum praktischen Teil!**