Autoconf Tutorial

Mark Galassi

1 Some background on autoconf

Autoconf is a set of tools (the programs are actually called `autoconf`, `autoheader`, `autoscan`, `autoreconf`, `autoupdate`, `ifnames`) which helps make your code configurable and portable to various versions of UNIX.

You probably have had the experience of installing a typical GNU utility, such as *GNU Emacs*. You might remember that, even though emacs is a gigantic and complex program, the installation procedure consisted symply of typing something like:

```
gzcat emacs-19.30.tar.gz | tar xvf -
cd emacs-19.30
./configure --prefix=/packages            # or whatever prefix you use
gmake install
```

and that was all the active work you had to do.

The `autoconf` tools help you prepare your software in this manner.

The autoconf documentation is quite complete and detailed, but it does not contain any simple concrete examples to get a beginner off the ground. I recently taught myself how to use autoconf on new programs and how to convert existing programs.

I taught myself by converting Gnudl and Dominion to use autoconf. Then I converted xyplot to autoconf and in doing that took notes. The notes, with a few embelishments, are what you find here. This document should allow you to take your old programs and add configure scripts to them, so that they will have that robustness and configurability.

Some other background details:

- There are two other automatic configuration systems: Larry Wall's *metaconfig* (used by rn, perl, patch...) and the X Window Consortium's *imake* utility. Both are also very good, but I have a strong preference for autoconf. The reasons are pretty much those stated in the autoconf manual, so I will not repeat them here.
- People writing GNU utilities (and other tools) usually use autoconf and follow many of the GNU coding standards (there is a document describing those as well). There are debateable points in the GNU coding standards, but overall they really make large software packages easy to install and maintain. I will call a program that uses the GNU coding standards and autoconf *GNUlitically correct*, and I mean it as a compliment: GNUitically correct programs are usually quality pieces of software, and when I install a program with a good configure script, I usually think "Aha, this is probably a good program".
- Many people, including myself, have come up with clever ideas for their '`Makefile`' and '`config.h`' files; ways to use the C preprocessor to decide which architecture we are on. I had come up with a foolproof way of guessing all architectures I had access to (which was very many versions of UNIX) for the *dominion* distribution. But little changes in new revisions would foil my scheme, and I saw that I was not that clever. Also, it was a lot of work. I have since converted dominion to use autoconf, and it is much more robust this way.

2 Converting an existing large program

This is the sequence of steps I did to convert James Theiler's *xyplot* to use autoconf. There are a few subdirectories involved: '`xydoc`', '`xyplot`', '`xysee`'.

2.1 Dumb work

First you have to do the mechanical stuff.

2.1.1 In the subdirectories

```
cd xydoc
cp Makefile Makefile.orig              # save the originals
mv Makefile Makefile.in
autoheader
cd ..
```

Then do the same in the other subdirectories (xyplot and xysee).

2.1.2 At the top level

At the end, I did the top level directory:

```
cd xydoc
cp Makefile Makefile.orig              # save the originals
mv Makefile Makefile.in
autoscan
mv configure.scan configure.in
autoheader
autoconf
cd ..
```

Note that I only did '`autoheader`' in the subdirectories, but I did the full '`autoconf`' in the top level directory.

2.2 Touching up configure.in

Now some things have to be changed in '`configure.in`':

- Toward the top of '`configure.in`' (after AC_INIT(...)) I put in these lines:

  ```
  dnl Most of these things are boiler plate (from autoscan);
  dnl but here's some of my stuff
  dnl BEGIN MARK's BLOCK OF STUFF

  dnl NOTHING IN HERE YET

  dnl END MARK's BLOCK OF STUFF
  ```

- At the very bottom, I replace the line

  ```
  AC_OUTPUT(xyview/Makefile xysee/Makefile xydoc/Makefile xyplot/Makefile Makefile)
  ```

  with

  ```
  AC_OUTPUT(xyview/Makefile xysee/Makefile xydoc/Makefile xyplot/Makefile Makefile,
  ```

  This will allow me to create rules for re-configuring automatically when various '`.in`' files change.

Then re-run `autoconf` in the top level directory.

2.3 Try it out

At this point you should be able to try it out by typing something like

```
./configure --prefix=`pwd`/work
make
```

and if all goes well, you should get almost exactly the same results as before you started this exercise.

2.4 Moving toward the GNU coding standards

I then went in to make the Makefiles look a bit more like standard GNU makefiles, with all the targets and variables.

2.4.1 Makefile variables

I changed all the ‘`Makefile.in`’ files to have the following boiler-plate stuff at the top:

```
SHELL = /bin/sh
VPATH = @srcdir@

subdirs = @subdirs@
top_srcdir = @top_srcdir@
srcdir = @srcdir@
prefix = @prefix@
exec_prefix = @exec_prefix@
bindir = $(exec_prefix)/bin
infodir = $(prefix)/info
libdir = $(prefix)/lib/gnudl
mandir = $(prefix)/man/man1

CC = @CC@
CPPFLAGS = @CPPFLAGS@
CFLAGS = $(CPPFLAGS) @CFLAGS@
LDFLAGS = @LDFLAGS@
LIBS = @LIBS@
INSTALL = @INSTALL@
```

And this stuff at the bottom, so that "make" will automatically reconfigure if the ‘`.in`’ files change.

```
# automatic re-running of configure if the ocnfigure.in file has changed
${srcdir}/configure: configure.in aclocal.m4
cd ${srcdir} && autoconf

# autoheader might not change config.h.in, so touch a stamp file
${srcdir}/config.h.in: stamp-h.in
${srcdir}/stamp-h.in: configure.in aclocal.m4
cd ${srcdir} && autoheader
echo timestamp > ${srcdir}/stamp-h.in

config.h: stamp-h
stamp-h: config.h.in config.status
./config.status
Makefile: Makefile.in config.status
./config.status
```

```
config.status: configure
./config.status --recheck
```

Having added these GNU-friendly variables to the 'Makefile.in', I removed the original 'Makefile.in' material that did similar things, and replaced it with these variables. This is almost always straightforward.

The effect of adding the rules for 'config.h', 'configure' and so forth is that the programmer never has to remember if she or he has run autoconf or configure recently: the make will figure it out and run what has to be run.

2.4.2 Adding various Makefile targets

It would help if you were to add some of the standard GNU targets to your 'Makefile.in', targets such as clean, distclean, install, uninstall.

Here are examples of how that was done for xyplot. Note that xyplot has several 'Makefile.in' files. I will show these rules from the top level '$(srcdir)/Makefile.in' (which does very little work: it just invokes the rules from the subdirectories and knows how to make snapshots) and the '$(srcdir)/xyplot/Makefile.in' which is the main workhorse. Keep in mind that xyplot is free software, and you can find it by anonymous ftp from ftp://nis-ftp.lanl.gov/pub/users/jt/Software

Here are some targets from the top level '$(srcdir)/Makefile.in':

```
all:
@for dir in ${subdirs}; do \
  (cd $$dir && $(MAKE) all) \
  || case "$(MFLAGS)" in *k*) fail=yes;; *) exit 1;; esac; \
done && test -z "$$fail"

install:
@for dir in ${subdirs}; do \
  (cd $$dir && $(MAKE) install) \
  || case "$(MFLAGS)" in *k*) fail=yes;; *) exit 1;; esac; \
done && test -z "$$fail"
clean:
/bin/rm -f *~
@for dir in ${subdirs}; do \
  (cd $$dir && $(MAKE) clean) \
  || case "$(MFLAGS)" in *k*) fail=yes;; *) exit 1;; esac; \
done && test -z "$$fail"

distclean: clean
/bin/rm -f Makefile config.h config.status config.cache config.log
@for dir in ${subdirs}; do \
  (cd $$dir && $(MAKE) distclean) \
  || case "$(MFLAGS)" in *k*) fail=yes;; *) exit 1;; esac; \
done && test -z "$$fail"

# a rule to make snapshots
snapshot: $(SOURCES) $(DOCS) $(OTHERFILES)
@echo
@echo "->Note: The version for now is hacked into Makefile.in as"
@echo "->" $(VERS)
@echo
@echo "->copying all release files to the directory " xyplot-$(VERS)
@echo
tar cf - $(SOURCES) $(DOCS) $(OTHERFILES) | gzip > xyplot-$(VERS).tar.gz
-mkdir xyplot-$(VERS)
gzcat xyplot-$(VERS).tar.gz | (cd xyplot-$(VERS); tar xf -)
```

```
/bin/rm -f xyplot-$(VERS).tar.gz
@echo
@echo "->making the compressed tar file " xyplot-$(VERS).tar.gz
@echo
tar cf - xyplot-$(VERS) | gzip > xyplot-$(VERS).tar.gz
@echo
# @echo "->placing the snapshot for anonymous ftp in " $(FTPDIR)
# @echo
# rcp xyplot-$(VERS).tar.gz $(FTPDIR)
echo "->removnig the temporary directory " xyplot-$(VERS)
/bin/rm -rf xyplot-$(VERS)             # remove the old directory
```

And here are some targets from '`$(srcdir)/xyplot/Makefile.in`':

```
install: all
$(top_srcdir)/mkinstalldirs $(bindir)
$(top_srcdir)/mkinstalldirs $(libdir)
$(INSTALL) xyplot $(bindir)
$(INSTALL) xyps $(bindir)
$(INSTALL) xyug $(bindir)
$(INSTALL) xypost $(libdir)

uninstall:
-/bin/rm -f $(bindir)/xyplot
-/bin/rm -f $(bindir)/xyps
-/bin/rm -f $(bindir)/xyug
-/bin/rm -f $(libdir)/xypost

# removes whatever can be built with make except xypost
clean:
/bin/rm -f *.o *~ xyplot xyps squeeze xyug

distclean:
/bin/rm -f Makefile config.h config.status config.cache config.log
```

2.4.3 Adding some utility programs to your distribution

You might have noticed that the `install:` rules used a couple of programs '`mkinstalldirs`' and `$(INSTALL)`. You should include a copy of '`mkinstalldirs`' in the top level directory of your distribution. The `$(INSTALL)` variable gets set by '`configure`' to be a Berkeley–style install program. If the user does not have a BSD–compatible install program, it is important for you provide the '`install-sh`' shell script in the top level directory.

So you should grab '`mkinstalldirs`' and '`install-sh`' from some other GNU program (such as autoconf, for example) and put them in your distribution.

2.4.4 Cutting a release

Another nice thing that GNU programs almost always do is to create a '`program-versionnumber`' directory when you unbundle them. The best way to do this is to have a script or a '`Makefile.in`' target which makes a directory called program-version and copies all the files that need to be distributed into that directory, and then makes a gzipped tar archive of that directory.

You can look at the `snapshot:` target shown above for xyplot. It first uses a '`tar`' pipeline to copy the essential files into an appropriately named directory (in this case '`xyplot-2.6.0`'). Then it uses '`tar`' and '`gzip`' to make the file '`xyplot-2.6.0.tar.gz`', which is ready to be sent out to the world.

It goes without saying that you should try the installation procedure with this '`.tar.gz`' file on as many versions of UNIX as possible before you publicize the release.

3 Using autoconf on a new program

Things are much easier if you are writing a brand new program. It would be nice if the '`autoconf`' distribution or the GNU coding standards were shipped with a template for '`Makefile.in`', so you could just fill in the blanks. (Note that there is no need for '`configure.in`' or '`config.h.in`' templates, since '`autoscan`' and '`autoheader`' make those.)

3.1 The Makefile.in template

I will provide the beginnings of a '`Makefile.in`' template here. I have marked with my usual *???* code the parts you will want to modify, or where you will need to add your own files.

```
# Main Makefile for Mark Galassi's stupid ``marklib'' library
# Copyright (C) 1996 Mark Galassi.

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2, or (at your option)
# any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

SHELL = /bin/sh
top_srcdir = @top_srcdir@
srcdir = @srcdir@
VPATH = @srcdir@

.SUFFIXES:
.SUFFIXES: .c .o

OPT=-g -O

AR = ar
AR_FLAGS = rc
RANLIB = @RANLIB@

CC = @CC@
CFLAGS = -I. @CFLAGS@
LDFLAGS = @LDFLAGS@
LIBS = @LIBS@
INSTALL = @INSTALL@
prefix = @prefix@
exec_prefix = @exec_prefix@
bindir = $(exec_prefix)/bin
libdir = $(prefix)/lib
infodir = $(prefix)/info
```

```
# ??? replace these with your own list of files
SOURCES=file1.c file2.c ...
DOCS=PROG.texi PROG.info
MISC=configure mkinstalldirs install-sh aclocal.m4
OBJS=file1.o file2.o ...
LIB_OBJS=libf1.o libf2.o ...

# ??? replace with your targets
all: libMYPROG.a PROG PROG.info

# ??? here I make the bindir, libdir and infodir directories; you
# might not need all of these.  also, I assumed the names PROG and
# libMYPROG.a for the program and library.
install: all
$(top_srcdir)/mkinstalldirs $(bindir)
$(top_srcdir)/mkinstalldirs $(libdir)
$(top_srcdir)/mkinstalldirs $(infodir)
$(INSTALL) PROG $(bindir)/PROG
$(INSTALL) libMYPROG.a $(libdir)/libMYPROG.a
$(INSTALL) PROG.info $(infodir)/PROG.info

uninstall:
/bin/rm -f $(bindir)/PROG
/bin/rm -f $(libdir)/libMYPROG.a
        /bin/rm -f $(infodir)/PROG.info

libMYPROG.a: $(OBJS)
/bin/rm -f libMYPROG.a
$(AR) $(AR_FLAGS) libMYPROG.a $(LIB_OBJS)
$(RANLIB) libMYPROG.a

PROG: $(OBJS) libMYPROG.a
$(CC) $(CFLAGS) -o PROG $(OBJS) $(LIBS)

clean:
/bin/rm -f core *.o $(OBJS) $(LIB_OBJS) libMYPROG.a

distclean: clean
/bin/rm -f Makefile config.h config.status config.cache config.log \
marklib.dvi

mostlyclean: clean

maintainer-clean: clean

PROG.info: PROG.texi
makeinfo PROG.texi

# automatic re-running of configure if the ocnfigure.in file has changed
${srcdir}/configure: configure.in aclocal.m4
cd ${srcdir} && autoconf

# autoheader might not change config.h.in, so touch a stamp file
${srcdir}/config.h.in: stamp-h.in
${srcdir}/stamp-h.in: configure.in aclocal.m4
```

```
cd ${srcdir} && autoheader
echo timestamp > ${srcdir}/stamp-h.in

config.h: stamp-h
stamp-h: config.h.in config.status
./config.status
Makefile: Makefile.in config.status
./config.status
config.status: configure
./config.status --recheck
```

3.2 What else

I must build up a hello world style program here so that I can give a few more concrete examples. The program should ideally have a library, a texinfo file, an executable and a '`.h`' file.

4 Concluding remarks

There are a couple of other steps involved in making your program GNUitically correct.

If you intend the program to be free software, you should seriously consider putting it under the GNU General Public License (GPL). The net is full of people criticizing it, but the fact is that the GPL works fine and takes little effort. Also, when people think they have a really clever licensing idea, they usually really don't.

If you are writing documentation for your program, you should consider writing it *texinfo*. Once again, there are criticisms to texinfo, and a lot of people have cleverer ideas, but I know of no other documentation system that has the following features (which, once the fluff gets old, are what really matters):

- Everything involved is free software with source code available, and ports to all UNIX variants.
- You can generate good looking plain ascii files.
- You can generate very high quality printed manuals with TeX.
- You can generate HTML documents for people on the web.
- You can generate *info* files for browsing with emacs or the stand alone info program.
- You can link your own program with my cinfo library (needs a lot of work) to have embedded context sensitive help. (Treat this item as vaporware for now.)

The limitations and obsolete limitations in texinfo are being discussed and addressed these days.

Table of Contents