使用 GNU autotools 改造一个软件项目

及永刚

jungle@soforge.com

2006 年 3 月 24 日 版本：0.3

本文不是一篇规范的教程，而是用一个软件项目作为例子，演示使用 GNU autotools 进行软件管理的思路和过程。

目 录

1 示例项目

这里借用了 Wei Mingzhi <whistler_wmz@users.sf.net> 开发的麻将游戏来进行演示，在此，先对他表示感谢!

示例软件下载:

http://planet.time.net.my/TechnologyPark/semj/mahjong.tar.bz2

2 软件布局

将下载的软件包解压到一个目录

```
$ cd ~/work
$ tar xjf mahjong.tar.bz2
```

可以看到这是一个典型的 Windows 风格的软件项目布局，在 mahjong 目录下放着程序的源码，程序运行时使用的数据文件放在子目录下面。

作者还提供了一个 Makefile，一个 DOS 风格的 !play.bat 批处理文件，一个编译好的 mj.exe 可执行文件。在 Win32 平台上运行 !play.bat 就可以直接运行程序。在 Unix/Linux 系统上，进入 mahjong 目录，键入 make 命令，**如果**一切顺利的话，将生成 mj 可执行文件，然后在命令行上运行 ./mj 程序，也可以启动麻将游戏。

对于一个 Windows 程序来讲，该软件布局可以说非常清晰明了。但在 Unix/Linux 系统上，执行 make 命令就可能遇到问题：编译找不到头文件，或者连接找不到库文件。而在 make 成功以后，运行麻将程序必须先进入该 majiong 目录，才能执行程序。如果要像其他的程序一样，可以在任意目录使用，就要专门为这一个程序修改 PATH 环境变量，或者再写一个启动脚本，并将它复制到 /usr/bin 这样的目录下。

3 Makefile 分析

```
1  #
2  # Copyright (c) 2005, Wei Mingzhi. All rights reserved.
3  #
4  # Use, redistributions and modifications of this file is
5  # unrestricted provided the above copyright notice is
6  # preserved.
```

```
 7  #
 8
 9  OBJ = \
10          bot.o config.o game.o general.o hand.o ini.o main.o \
11          player.o text.o tile.o util.o
12
13  HEADERS = \
14          bot.h game.h general.h hand.h ini.h main.h player.h \
15          tile.h
16
17  CC = gcc
18  CXX = g++
19
20  TARGET = mj
21
22  BASEFLAGS = -g3 -D_DEBUG=1
23  #BASEFLAGS = -s -O3
24
25  CFLAGS = ${BASEFLAGS} `sdl-config --cflags`
26  LDFLAGS = ${BASEFLAGS} `sdl-config --libs` -lSDL_image \
27            -lSDL_mixer -lSDL_ttf
28
29  all: ${TARGET}
30
31  ${TARGET}: ${OBJ}
32          ${CXX} ${LDFLAGS} -o ${TARGET} ${OBJ}
33
34  clean:
35          rm -f *.o ${TARGET}
36
37  distclean:
38          rm -f *.o ${TARGET}
39
40  %.o:    %.cpp ${HEADERS}
41          ${CXX} ${CFLAGS} -c $< -o $@
42
43  %.o:    %.c ${HEADERS}
44          ${CC} ${CFLAGS} -c $< -o $@
```

Makefile 很清楚：第 20 行定义 TARGET 变量为 mj，第 29 行表明 make 默认的 target 也就是生成 'mj'；第 17 行加入编译时的调试信息；第 25、26、27 行使用了 sdl-config 工具侦测 SDL 开发库编译链接信息，在 26、27 行还指明需要连接 SDL_image、SDL_mixser 和 SDL_ttf 库。

4 GNU 的软件风格

一个标准的 GNU 软件，编译安装都是使用下面三个步骤:

```
$ ./configure
$ make
$ make install
```

configure 脚本运行时可以侦测系统的环境，确定软件安装目录，然后生成 Makefile 文件。make 调用系统中的编译器进行编译和连接。make install 将软件安装到设定的目录。

用户执行 configure 时可以通过它的命令行参数指定自己所需的编译选项，比如安装目录通过 `--prefix=PREFIX` 设置，如果不指定，缺省情况下 PREFIX 是 /usr/local。默认安装时，执行文件安装到 /usr/local/bin 目录，库安装到 /usr/local/lib 目录，数据文件安装到 /usr/local/share 目录。

由于 GNU 的软件风格方便易用，通用性好，可移植性高，现在大多数 Unix/Linux 系统上的自由软件都采用这种方式分发软件。

5 准备 autotools

GNU autotools 主要包含三个软件：autoconf，automake 和 libtool。当前流行的有新旧两个版本，本例采用的是新版本，分别对应的是：autoconf 2.59，automake 1.9.6 和 libtool 1.5.18。

很多 linux 发行版都会默认安装这几个工具。本例是在 NetBSD 下进行操作，安装这几个软件包是通过 pkgsrc，它们在 pkgsrc 目录为 devel/autoconf、devel/automake 和 devel/libtool。

6 改造文件布局

原来软件的根目录下面放的是程序的源码，按照 GNU 的习惯，将它们放到 src 子目录，根目录留给 configure 这类文件使用，其他的数据文件保持不变，仍然放在各自的子目录。

先创建一个目录 majiang，然后根据需要将 mahjong 目录下的文件复制过来。由于是为 Unix/Linux 系统进行改写，原目录里的 win32 相关文件就不用复制到新目录。

```
$ cd ~/work/majiang
$ ls
data/  docs/  fonts/  images/  music/  sound/  src/
```

7 autoscan

autoconf 软件包里面的 autoscan 工具可以扫描工作目录，生成一个 configure.ac 的模板文件 configure.scan。

```
$ cd ~/work/majiang
$ autoscan
```

autoscan 命令在当前目录生成的 configure.scan 文件内容为:

```
 1  #                                               -*- Autoconf -*-
 2  # Process this file with autoconf to produce a configure script.
 3
 4  AC_PREREQ(2.59)
 5  AC_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)
 6  AC_CONFIG_SRCDIR([src/bot.h])
 7  AC_CONFIG_HEADER([config.h])
 8
 9  # Checks for programs.
10  AC_PROG_CXX
11  AC_PROG_CC
12
13  # Checks for libraries.
14
15  # Checks for header files.
16  AC_HEADER_STDC
17  AC_CHECK_HEADERS([limits.h malloc.h stdlib.h string.h unistd.h])
18
19  # Checks for typedefs, structures, and compiler characteristics.
20  AC_HEADER_STDBOOL
21  AC_C_CONST
22  AC_C_INLINE
```

```
23
24  # Checks for library functions.
25  AC_FUNC_MALLOC
26  AC_FUNC_REALLOC
27  AC_CHECK_FUNCS([memset strcasecmp strchr strdup])
28  AC_OUTPUT
```

号开始的行是注释，其他都是 m4 宏命令。将它改名为 configure.ac，然后在此基础上进行修改。

8 configure.ac 的基本结构

configure.ac 文件是 autoconf 的输入文件，经过 autoconf 处理，展开里面的 m4 宏，输出的是 configure 脚本。

第 4 行声明本文件要求的 autoconf 版本，因为本例使用了新版本 2.59，所以在此注明。

第 5 行 AC_INIT 宏用来定义软件的名称和版本等信息，本例写成:

```
AC_INIT(majiang, 1.0)
```

这里省略了 BUG-REPORT-ADDRESS 参数，它是可选项，一般写成作者的邮件地址。

第 6 行 AC_CONFIG_SRCDIR 宏通过侦测所指定的源码文件是否存在，来确定源码目录的有效性。可以选择源码目录中的任何一个文件作为代表，比如将 autoscan 选择的 bot.h 文件改成 main.cpp:

```
AC_CONFIG_SRCDIR([src/main.cpp])
```

宏参数中使用‘[]’，是为了表明其中的字符串是一个整体。

第 7 行的 AC_CONFIG_HEADER 宏用于生成 config.h 文件，里面存放 configure 脚本侦测到的信息。如果程序需要使用其中的定义，就在源码中加入

```
#include <config.h>
```

其他的一些宏是标准的侦测过程，可以保留不动。

configure.ac 文件要求 AC_INIT 宏必须放在开头位置，AC_OUTPUT 放在文件末，中间用来检测编译环境的各种宏没有特别的先后次序要求，由宏之间相互关系决定。

9 Makefile 文件的产生

前面 configure.ac 里面的宏，主要作用是侦测系统，并没有编译相关的设置。因为这些信息是写在 Makefile.am 里面，然后用 automake 工具转换成 Makefile.in，configure 脚本执行时再读取 Makefile.in，并与侦测信息一起写到 Makefile 文件。

在 autotools 的命名习惯中，后缀 ac 的文件是 autoconf 的输入文件，后缀 am 的文件是 automake 的输入文件，后缀 in 的文件是 configure 的输入文件。autoconf 旧版本中 configure.in 等同于 configure.ac，虽然新版本也可以识别，但它不符合命名规则，所以新版本的文件应该使用 ac 后缀。

简单的 Makefile.in 可以手动编写，如果使用 automake 产生，需要在 configure.ac 里面加入 AM_INIT_AUTOMAKE 宏进行声明。

要输出 Makefile，还需要在 configure.ac 中使用 AC_CONFIG_FILES 宏指明。该宏并不是只处理 Makefile，而是将 FILE.in 文件转换为 FILE 文件。因为 make 可以遍历子目录，如果子目录中存在 Makefile，也将同时处理。在本例中 src 目录下是源码，其他是数据文件，可以使用单独一个 Makefile 放在根目录下面，也可以用多个 Makefile。由于每个子目录的 Makefile 只处理本目录的文件，分工明确，是模块化的方法，推荐使用。因此在 configure.ac 里面增加下面的宏，表示软件根目录和子目录中都需要生成 Makefile 文件:

```
AC_CONFIG_FILES([Makefile
                 src/Makefile
                 data/Makefile
                 docs/Makefile
                 fonts/Makefile
                 images/Makefile
                 music/Makefile
                 sound/Makefile])
```

10 编写 Makefile.am

10.1 软件根目录 Makefile.am

由于该目录下面保存的是与 autotools 相关的文件，没有需要编译安装的文件，所以只注明需要进一步处理的子目录信息:

```
SUBDIRS = src data docs fonts images music sound
```

10.2 src/Makefile.am

此目录里是源代码，最终生成 mj 可执行文件，在其 Makefile.am 中写入

```
bin_PROGRAMS = mj

mj_SOURCES = bot.h \
             bot.cpp \
             config.cpp \
             game.h \
             game.cpp \
             general.h \
             general.cpp \
             hand.h \
             hand.cpp \
             ini.h \
             ini.cpp \
             main.h \
             main.cpp \
             player.h \
             player.cpp \
             text.cpp \
             tile.h \
             tile.cpp \
             util.cpp
```

am 文件里变量通过命名判断其含义，保留的字符串间用下划线分隔。

bin_PROGRAMS 表示列出二进制的程序，值为多个空格分开的程序列表，这里仅有一个 mj。

mj_SOURCES 列出的是组成 mj 程序的文件，文件比较多的时候，每个文件写成一行容易看清楚。

10.3 data/Makefile.am

本目录的文件是 mj 运行时读取的数据，它的 Makefile.am 可以这样写

```
mjdatadir = $(pkgdatadir)/data
mjdata_DATA = mj.ini titles.txt
EXTRA_DIST = $(mjdata_DATA)
```

因为 datadir 是保留的关键字，所以用 mjdatadir 代替，pkgdatadir 指向 $prefix/share/FULL-PACKAGE-NAME 目录，因为在 AC_INIT 中已经声明 FULL-PACKAGE-NAME 为 majiang，pkgdatadir 就等于 $prefix/share/majiang 目录。

其中 mjdatadir 让 data 目录下的文件安装到 $prefix/share/majiang/data 目录里面。

mjdata_DATA 列出此目录下需要安装的文件，然后用 EXTRA_DIST 变量注明。

余下几个子目录都与 data 目录类似。

10.4 docs/Makefile.am

```
docsdir = $(pkgdatadir)/docs
docs_DATA = gkai00mp.txt  gpl.html  readme.txt
EXTRA_DIST = $(docs_DATA)
```

10.5 fonts/Makefile.am

```
fontsdir = $(pkgdatadir)/fonts
fonts_DATA = brush.ttf  gkai00mp.ttf
EXTRA_DIST = $(fonts_DATA)
```

10.6 images/Makefile.am

```
imagesdir = $(pkgdatadir)/images
images_DATA =   bgame.jpg \
```

```
                mjgirl1a.jpg \
                mjgirl2a.jpg \
                mjgirl3a.jpg \
                mjgirl4a.jpg \
                tiles.jpg \
                electron.jpg \
                mjgirl1b.jpg \
                mjgirl2b.jpg \
                mjgirl3b.jpg \
                mjgirl4b.jpg \
                gameover.jpg \
                mjgirl1c.jpg \
                mjgirl2c.jpg \
                mjgirl3c.jpg \
                mjgirl4c.jpg
EXTRA_DIST = $(images_DATA)
```

10.7 music/Makefile.am

```
musicdir = $(pkgdatadir)/music
music_DATA =    bet.ogg \
                bonus.ogg \
                music.ogg \
                musicb.ogg \
                musice.ogg \
                win.ogg \
                bgame.ogg \
                gameover.ogg \
                music1.ogg \
                musicc.ogg \
                musicp.ogg
EXTRA_DIST = $(music_DATA)
```

10.8 sound/Makefile.am

```
sounddir = $(pkgdatadir)/sound
sound_DATA =    boom.wav \
                ding.wav \
                discard.wav \
                discard2.wav \
                flash.wav \
                snd1.wav \
```

```
                snd2.wav \
                snd3.wav \
                snd4.wav
EXTRA_DIST = $(sound_DATA)
```

11 运行 autotools

准备好 configure.ac 和 Makefile.am，就可以用 autotools 的命令处理这些文件。开始可能会出现错误，不过没关系，可以按照错误信息的提示逐步进行修正。

首先要使用的是 aclocal 命令，它根据 configure.ac 的定义，将需要使用的 m4 宏定义复制到 aclocal.m4 里面。缺省时，搜索 m4 宏是从 autoconf 的安装目录和系统的 aclocal 目录。如果需要使用其他路径下的宏，可以通过命令行的 -I 选项指定。

接着使用 autoheader 命令，它负责生成 config.h.in 文件，这里面的 C 语言宏定义也是通过解析 configure.ac 产生。

下来运行 automake 命令处理 Makefile.am，生成 Makefile.in。GNU 对自己发布的软件有严格的规范，比如必须附带许可证声明文件 COPYING 等等，否则 automake 执行时会报错。automake 提供了三种软件等级：foreign、gnu 和 gnits，让用户选择采用，默认等级为 gnu。本例使用 foreign 等级，它只检测必须的文件。有一些必需的脚本文件可以从 automake 软件包里复制过来，在执行时使用 `--add-missing` 选项可以让 automake 自动添加，默认方式是采用符号链接，如加上 `--copy` 选项则可以使用复制方式。本例中，automake 的命令如下:

```
$ automake --foreign --add-missing --copy
```

最后，使用 autoconf 命令生成 configure 脚本文件。

12 SDL 库的侦测

这个麻将游戏是基于 SDL 库开发的，一般系统默认不会安装，因此 configure 脚本的一个任务就是检查用户的系统中是否有该软件包。

autoconf 提供了很多宏可以实现侦测功能，但首先应该查看 SDL 软件包是否已经提供相应的宏。通过 pkgsrc 的工具可以看到:

```
$ pkg_info -L SDL|grep m4
/usr/pkg/share/aclocal/sdl.m4
```

即 SDL 软件包提供了一个 sdl.m4 宏，放在系统的 aclocal 目录下。

在这个宏文件的注释中说明了使用的方法:

```
dnl AM_PATH_SDL([MINIMUM-VERSION,[ACTION-IF-FOUND[,ACTION-IF-NOT-FOUND]]])
dnl Test for SDL, and define SDL_CFLAGS and SDL_LIBS
```

也就是说在 configure.ac 里面调用 AM_PATH_SDL 宏，就可以侦测 SDL。找到 SDL 库以后，该宏还输出 SDL_CFLAGS 和 SDL_LIBS 编译连接选项，它们实际上就是调用 ‘sdl-config --cflags‘ 和 ‘sdl-config --libs‘。

于是在 configure.ac 里面加入 AM_PATH_SDL 宏

```
# Checks for libraries.
SDL_VERSION=1.2.0
AM_PATH_SDL($SDL_VERSION,
            :,
    AC_MSG_ERROR([*** SDL version $SDL_VERSION not found!])
)
```

当前 SDL 的版本为 1.2.9，于是 MINIMUM-VERSION 就设为 1.2.0。如果在系统中侦测到需要的库，没什么额外的操作，假如没有找到，则给出错误信息。

AM_PATH_SDL 输出 SDL_CFLAGS 和 SDL_LIBS 编译参数，需要添加到 src/Makefile.am 里面:

```
mj_CPPFLAGS = @SDL_CFLAGS@
mj_LDFLAGS = @SDL_LIBS@
```

用 ‘@’ 包围的变量会在 configure 执行时被替换。

从 mahjong 的 Makefile 中看到，这个软件还要使用 SDL_image、SDL_mixser 和 SDL_ttf 库，但它们不属于 SDL 软件包，需要另外安装。由于这些库在 sdl.m4 中也没有进行侦测，所以自己要写一些脚本。

autotools 提供了一个 AC_CHECK_LIB 宏可以用来检测库，现在就使用它来检测这几个 SDL 库。该宏的语法为:

```
AC_CHECK_LIB (LIBRARY, FUNCTION, [ACTION-IF-FOUND],
          [ACTION-IF-NOT-FOUND], [OTHER-LIBRARIES])
```

第一个参数是库名，第二个参数是库中的一个函数，第三个参数是检测到以后进行的动作，第四个参数是未检测到以后的动作，第五个参数是其他的库。

对于 SDL_image、SDL_mixer 和 SDL_ttf 对应的使用方法如下:

```
# Check for SDL_image library
AC_CHECK_LIB(SDL_image, IMG_Load, , AC_MSG_ERROR([
*** Unable to find SDL_image libary with PNG support
(http://www.libsdl.org/projects/SDL_image/)
]), `sdl-config --libs`)

# Check for SDL_mixer library
AC_CHECK_LIB(SDL_mixer, Mix_LoadMUS, , AC_MSG_ERROR([
*** Unable to find SDL_mixer libary with OGG support
(http://www.libsdl.org/projects/SDL_mixer/)
]), `sdl-config --libs`)

# Check for SDL_ttf library
AC_CHECK_LIB(SDL_ttf, TTF_OpenFont, , AC_MSG_ERROR([
*** Unable to find SDL_ttf libary
(http://www.libsdl.org/projects/SDL_ttf/)
]), `sdl-config --libs`)
```

其中IMG_Load、Mix_LoadMUS 和TTF_OpenFont 分别是源码中调用的函数。

13 软件使用的数据文件

原来 mj 读取数据是从执行时目录的子目录中读取，但现在将数据放到 $prefix/share/majiang 目录下，需要通过一种途径让程序可以知道数据文件被安放的位置。

要达到这个目的有很多方法，这里采用最直接的一种：将数据文件安装目录变量通过 CPPFLAGS 编译参数传递给程序。

于是修改 src/Makefile.am 的 CPPFLAGS:

```
mj_CPPFLAGS = @SDL_CFLAGS@ -DDATA_DIR=\"${datadir}/majiang\"
```

相应地修改 src 目录下的源码，在读取数据文件的地方，将读取的路径改成 DATA_DIR 里对应的子目录。例如，原先 config.cpp 中是:

```
void LoadCfg()
{
   cfg.Load("data/mj.ini");
}
```

现改成:

```
void LoadCfg()
{
   cfg.Load(DATA_DIR"data/mj.ini");
}
```

14 configure 选项

原来 mahjong 的 Makefile 第 22 行定义了 debug 调试选项，虽然也可以照样放到 src/Makefile.am 的 CPPFLAGS 里面实现，但 autotools 提供了一种更灵活的机制。

configure 脚本可以通过选项来设置编译参数，现增加一个 `--enable-debug` 选项，需要 DEBUG 时，在命令行上加上它来打开，默认则关闭。

这项功能是使用 AC_ARG_ENABLE 宏实现:

```
AC_ARG_ENABLE (FEATURE, HELP-STRING, [ACTION-IF-GIVEN],
          [ACTION-IF-NOT-GIVEN])
```

其中 FEATURE 是名称，HELP_STRING 为说明信息，在使用 `./configure --help` 时可以看到。最后两个分别对应打开和关闭时的操作。

现在将 DEBUG 功能加入 configure.ac:

```
AC_ARG_ENABLE(debug,
      [  --enable-debug           turn on debug],
      CXXFLAGS="$CXXFLAGS -g3 -D_DEBUG=1")
```

15 autotools 脚本

每次修改了 configure.ac 或 Makefile.am 等 autotools 输入文件后都需要再次运行 aclocal、automake、autoconf 这些命令，为了方便起见，可以将他们放到一个 shell 脚本里面，例如:

```
#! /bin/sh
set -x
aclocal
autoheader
automake --foreign --add-missing --copy
autoconf
```

将上面内容保存到 autogen.sh 文件，并修改文件属性为 755。每次需要重新生成 configure 脚本时，执行 ./autogen.sh 即可。

16 使用 configure 产生的 Makefile

现在执行 ./autogen.sh 得到的 configure 脚本已经可以正常工作了，进入 ~/work/majiang 目录，执行 ./configure，可以看到它检查系统的过程，包括 SDL 和 SDL_image 等库的侦测结果。使用 ./configure –help 可以看到 autotools 提供的帮助信息。

configure 执行的完毕，输出软件根目录和几个子目录下面的 Makefile 文件。这些 Makefile 有几个常用的 target:

- make all

  不加任何 target，默认就是 all，作用是编译软件

- make install

  安装软件包，如果安装到系统目录，需要 root 权限

- make clean

  清除编译产生的目标文件

- make distclean

  可以同时清除编译的结果和 configure 输出的文件

- make tags

  生成 etags 使用的 TAGS 文件

- make dist

  生成软件发布包，为 tar.gz 格式的压缩包，文件名由软件包名和版本组成。

17 最终的 configure.ac 文件

```
#                                                -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

AC_PREREQ(2.59)
AC_INIT([majiang], [1.0])
AC_CONFIG_SRCDIR([src/main.cpp])
AC_CONFIG_HEADER([config.h])

AC_CANONICAL_HOST
AC_CANONICAL_TARGET
AM_INIT_AUTOMAKE

# Checks for programs.
AC_PROG_CXX
AC_PROG_CC

AC_LANG(C++)

# Checks for libraries.
SDL_VERSION=1.2.0
AM_PATH_SDL($SDL_VERSION,
            :,
    AC_MSG_ERROR([*** SDL version $SDL_VERSION not found!])
)

# Check for SDL_image library
AC_CHECK_LIB(SDL_image, IMG_LoadPNG_RW, , AC_MSG_ERROR([
*** Unable to find SDL_image libary with PNG support
(http://www.libsdl.org/projects/SDL_image/)
]), `sdl-config --libs`)

# Check for SDL_mixer library
AC_CHECK_LIB(SDL_mixer, Mix_LoadOGG_RW, , AC_MSG_ERROR([
```

```
*** Unable to find SDL_mixer libary with OGG support
(http://www.libsdl.org/projects/SDL_mixer/)
]), `sdl-config --libs`)

# Check for SDL_ttf library
AC_CHECK_LIB(SDL_ttf, TTF_OpenFont, , AC_MSG_ERROR([
*** Unable to find SDL_ttf libary
(http://www.libsdl.org/projects/SDL_ttf/)
]), `sdl-config --libs`)

# Checks for header files.
AC_HEADER_STDC
AC_CHECK_HEADERS([limits.h malloc.h stdlib.h string.h unistd.h])

# Checks for typedefs, structures, and compiler characteristics.
AC_HEADER_STDBOOL
AC_C_CONST
AC_C_INLINE

# Checks for library functions.
AC_FUNC_MALLOC
AC_FUNC_REALLOC
AC_CHECK_FUNCS([memset strcasecmp strchr strdup])

AC_ARG_ENABLE(debug,
      [  --enable-debug          turn on debug],
      CXXFLAGS="$CXXFLAGS -g3 -D_DEBUG=1")

AC_CONFIG_FILES([Makefile
                 src/Makefile
                 data/Makefile
                 docs/Makefile
                 fonts/Makefile
                 images/Makefile
                 music/Makefile
                 sound/Makefile])
AC_OUTPUT
```

18 结束语

GNU 的很多工具经常给人一种感觉: 功能很强大，但也很难学。autotools 可以说是这类工具的一个典型，它需要用户对 shell、make、软件编译、m4 宏语言，以及 Unix/Linux 操作系统各方面知识都有一定的了解。使用时又要 autoconf、automake、libtool 多个工具相互配合[1]，如果要给软件增加国际化功能，还要再了解和掌握 gettext、po 等工具和规则。

与学习其他知识一样，所谓**难**，其实是不了解，不熟悉。本文通过一个范例演示使用 autotools 的过程，是让不了解的人熟悉这个工具。但真正的理解，还需要将它运用到自己的软件项目当中，不断地实践，不断地思考和总结。

[1]由于本例软件中没有生成库文件，所以没有涉及 libtool 工具的使用。