**工具简介**

所必须的软件：autoconf/automake/m4/perl/libtool（其中libtool非必须）。

Automake 是一种帮助『自动』产生 Makefile 文件的软件，并且让开发出来的的软件可以象 Apache，MySQL 和常见的 GNU 软件一样，只需要写一些预先定义好的宏 (macro)，提交给Automake处理后会产生一个可以供 Autoconf 使用的 Makefile.in文件。再配合利用 Autoconf产生的自动配置设置文件 configure 即可产生一份符合 GNU Makefile 惯例的 Makeifle

**生成 Makefile 的来龙去脉**

首先进入 project 目录，在该目录下运行一系列命令，创建和修改几个文件，就可以生成符合该平台的Makefile文件，操作过程如下：

1) 运行autoscan命令

2) 将configure.scan 文件重命名为configure.in，并修改configure.in文件

3) 在project目录下新建Makefile.am文件

4) 在project目录下新建NEWS、 README、 ChangeLog 、AUTHORS文件

5) 将/usr/share/automake-1.X/目录下的depcomp和complie文件拷贝到本目录下

6) 运行aclocal命令

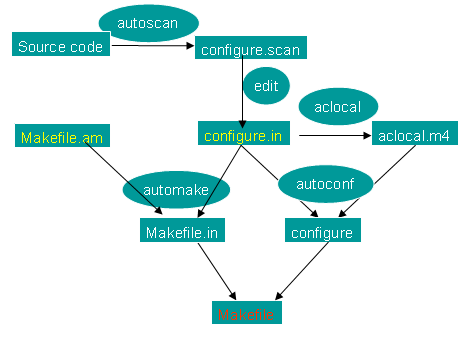
7) 运行autoconf命令

8) 运行automake -a命令

9) 运行./confiugre脚本

可以通过图2看出产生Makefile的流程，如图所示：

**图 2生成Makefile流程图**



**Configure.in**

当我们利用autoscan工具生成confiugre.scan文件时，我们需要将confiugre.scan重命名为confiugre.in文件。confiugre.in调用一系列autoconf宏来测试程序需要的或用到的特性是否存在，以及这些特性的功能。

下面我们就来目睹一下confiugre.scan的庐山真面目：

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

**AC\_PREREQ(2.59)**

**AC\_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)**

**AC\_CONFIG\_SRCDIR([config.h.in])**

**AC\_CONFIG\_HEADER([config.h])**

**# Checks for programs.**

**AC\_PROG\_CC**

**# Checks for libraries.**

**# FIXME: Replace `main' with a function in `-lpthread':**

**AC\_CHECK\_LIB([pthread], [main])**

**# Checks for header files.**

**# Checks for typedefs, structures, and compiler characteristics.**

**# Checks for library functions.**

**AC\_OUTPUT**

**每个configure.scan文件都是以AC\_INIT开头，以AC\_OUTPUT结束。我们不难从文件中看出confiugre.in文件的一般布局：**

**AC\_INIT**

**测试程序**

**测试函数库**

**测试头文件**

**测试类型定义**

**测试结构**

**测试编译器特性**

**测试库函数**

**测试系统调用**

**AC\_OUTPUT**

上面的调用次序只是建议性质的，但我们还是强烈建议不要随意改变对宏调用的次序。

现在就开始修改该文件：

**$mv configure.scan configure.in**

**$vim configure.in**

**修改后的结果如下：**

**# -\*- Autoconf -\*-**

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

**AC\_PREREQ(2.59)**

**AC\_INIT(test, 1.0, normalnotebook@126.com)**

**AC\_CONFIG\_SRCDIR([src/ModuleA/apple/core/test.c])**

**AM\_CONFIG\_HEADER(config.h)**

**AM\_INIT\_AUTOMAKE(test,1.0)**

**# Checks for programs.**

**AC\_PROG\_CC**

**# Checks for libraries.**

**# FIXME: Replace `main' with a function in `-lpthread':**

**AC\_CHECK\_LIB([pthread], [pthread\_rwlock\_init])**

**AC\_PROG\_RANLIB**

**# Checks for header files.**

**# Checks for typedefs, structures, and compiler characteristics.**

**# Checks for library functions.**

**AC\_OUTPUT([Makefile**

**src/lib/Makefile**

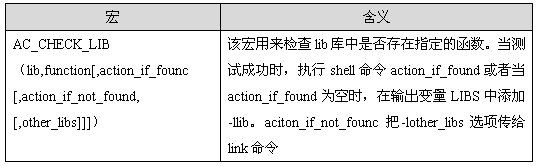
**src/ModuleA/apple/core/Makefile**

**src/ModuleA/apple/shell/Makefile**

**])**

其中要将AC\_CONFIG\_HEADER([config.h])修改为：AM\_CONFIG\_HEADER(config.h), 并加入AM\_INIT\_AUTOMAKE(test,1.0)。由于我们的测试程序是基于多线程的程序，所以要加入AC\_PROG\_RANLIB，不然运行automake命令时会出错。在AC\_OUTPUT输入要创建的Makefile文件名。

由于我们在程序中使用了读写锁，所以需要对库文件进行检查，即AC\_CHECK\_LIB([pthread], [main])，该宏的含义如下：

****

其中，LIBS是link的一个选项，详细请参看后续的Makefile文件。由于我们在程序中使用了读写锁，所以我们测试pthread库中是否存在pthread\_rwlock\_init函数。

由于我们是基于deep类型来创建makefile文件，所以我们需要在四处创建Makefile文件。即：project目录下，lib目录下，core和shell目录下。

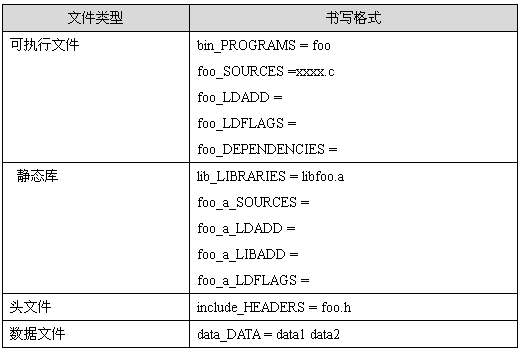
Autoconf提供了很多内置宏来做相关的检测，限于篇幅关系，我们在这里对其他宏不做详细的解释，具体请参看参考文献1和参考文献2，也可参看autoconf信息页。

**Makefile.am**

Makefile.am是一种比Makefile更高层次的规则。只需指定要生成什么目标，它由什么源文件生成，要安装到什么目录等构成。

表一列出了可执行文件、静态库、头文件和数据文件，四种书写Makefile.am文件个一般格式。

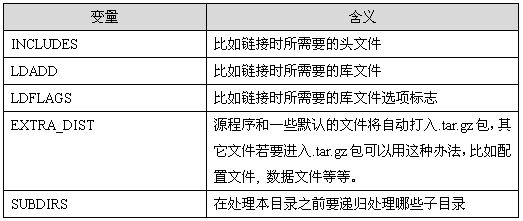
**表 1Makefile.am一般格式**



对于可执行文件和静态库类型，如果只想编译，不想安装到系统中，可以用noinst\_PROGRAMS代替bin\_PROGRAMS，noinst\_LIBRARIES代替lib\_LIBRARIES。

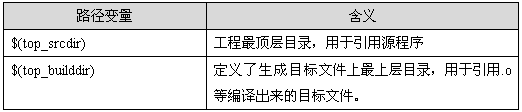
Makefile.am还提供了一些全局变量供所有的目标体使用：

**表 2 Makefile.am中可用的全局变量**



在Makefile.am中尽量使用相对路径，系统预定义了两个基本路径：

**表 3Makefile.am中可用的路径变量**



**编译步骤：**

编译一个简单的源文件main.c,自动生成一个makefile,以下是步骤：

第一步：

----------

在/root/project/main目录下创建一个文件main.c,其内容如下：

------------------------------------------------

#include <stdio.h>

int main(int argc, char\*\* argv)

{

    printf("Hello, Auto Makefile!\n");

    return 0;

}

------------------------------------------------

此时状态如下：

[root@localhost main]# **pwd**

/root/project/main

[root@localhost main]# **ls**

**main.c**

[root@localhost main]#

第二步：

----------

运行 **autoscan** , 自动创建两个文件： autoscan.log configure.scan

此时状态如下：

[root@localhost main]# **autoscan**

[root@localhost main]# **ls**

**autoscan.log configure.scan** main.c

[root@localhost main]#

第三步：

----------

修改configure.scan的文件名为configure.in

查看**configure.in**的内容：

------------------------------------------------

#                                               -\*- Autoconf -\*-

# Process this file with autoconf to produce a configure script.

AC\_PREREQ(2.61)

AC\_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)

AC\_CONFIG\_SRCDIR([main.c])

AC\_CONFIG\_HEADER([config.h])

# Checks for programs.

AC\_PROG\_CC

# Checks for libraries.

# Checks for header files.

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.

AC\_OUTPUT

------------------------------------------------

解读以上的文件：

------------------------------------------------

#                                               -\*- Autoconf -\*-

# Process this file with autoconf to produce a configure script.

# AC\_PREREQ:

# 确保使用的是足够新的Autoconf版本。如果用于创建configure的Autoconf的版

# 本比version 要早，就在标准错误输出打印一条错误消息并不会创建configure。

AC\_PREREQ(2.61)

#

# 初始化,定义软件的基本信息,包括设置包的全称,版本号以及报告BUG时需要用的邮箱地址

#

AC\_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)

#

# 用来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

#

AC\_CONFIG\_SRCDIR([main.c])

#

# 用于生成config.h文件，以便autoheader使用

#

AC\_CONFIG\_HEADER([config.h])

# Checks for programs.

AC\_PROG\_CC

# Checks for libraries.

# Checks for header files.

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.

#

# 创建输出文件。在`configure.in'的末尾调用本宏一次。

#

AC\_OUTPUT

------------------------------------------------

修改动作:

    1.修改AC\_INIT里面的参数: AC\_INIT(main,1.0, xxx@163.com)

    2.添加宏AM\_INIT\_AUTOMAKE, 它是automake所必备的宏，也同前面一样，PACKAGE是所要产生软件套件的名称，VERSION是版本编号。

    3.在AC\_OUTPUT后添加输出文件Makefile

修改后的结果:

------------------------------------------------

#                                               -\*- Autoconf -\*-

# Process this file with autoconf to produce a configure script.

AC\_PREREQ(2.61)

**AC\_INIT(main, 1.0, pgpxc@163.com)**

AC\_CONFIG\_SRCDIR([main.c])

AC\_CONFIG\_HEADER([config.h])

**AM\_INIT\_AUTOMAKE(main,1.0)**

# Checks for programs.

AC\_PROG\_CC

# Checks for libraries.

# Checks for header files.

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.

**AC\_OUTPUT([Makefile])**

------------------------------------------------

第四步:

运行 **aclocal**, 生成一个“**aclocal.m4**”文件和一个缓冲文件夹**autom4te.cache**，该文件主要处理本地的宏定义。

此时的状态是：

[root@localhost main]# **aclocal**

[root@localhost main]# **ls**

**aclocal.m4 autom4te.cache** autoscan.log configure.in configure.in~ main.c

[root@localhost main]#

第五步：

运行 **autoconf**, 目的是生成 configure

此时的状态是：

[root@localhost main]# **autoconf**

[root@localhost main]# **ls**

aclocal.m4      autoscan.log configure.in   main.c

autom4te.cache **configure**     configure.in~

[root@localhost main]#

第六步：

运行 **autoheader**，它负责生成config.h.in文件。该工具通常会从“acconfig.h”文件中复制用户附加的符号定义，因此此处没有附加符号定义，所以不需要创建“acconfig.h”文件。

此时的状态是：

[root@localhost main]# **autoheader**

[root@localhost main]# **ls**

aclocal.m4      autoscan.log configure     configure.in~

autom4te.cache **config.h.in**   configure.in main.c

[root@localhost main]#

第七步：

下面即将运行 **automake**, 但在此之前应该做一下准备工作!

首先

创建一个 **Makefile.am**.这一步是创建Makefile很重要的一步，automake要用的脚本配置文件是Makefile.am，用户需要自己创建相应的文件。之后，automake工具转换成Makefile.in。

这个Makefile.am的内容如下：

------------------------------------------------

AUTOMAKE\_OPTIONS=foreign

bin\_PROGRAMS=main

main\_SOURCES=main.c

------------------------------------------------

下面对该脚本文件的对应项进行解释。

    其中的AUTOMAKE\_OPTIONS为设置automake的选项。由于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，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。在本例使用foreign等级，它只检测必须的文件。

    bin\_PROGRAMS定义要产生的执行文件名。如果要产生多个执行文件，每个文件名用空格隔开。

    main\_SOURCES定义“main”这个执行程序所需要的原始文件。如果”main”这个程序是由多个原始文件所产生的，则必须把它所用到的所有原 始文件都列出来，并用空格隔开。例如：若目标体“main”需要“main.c”、“sunq.c”、“main.h”三个依赖文件，则定义 main\_SOURCES=main.c sunq.c main.h。要注意的是，如果要定义多个执行文件，则对每个执行程序都要定义相应的file\_SOURCES。

其次

使用automake对其生成“configure.in”文件，在这里使用选项“—adding-missing”可以让automake自动添加有一些必需的脚本文件。

运行后的状态是：

------------------------------------------------

[root@localhost main]# **automake --add-missing**

configure.in:8: installing `./missing'

configure.in:8: installing `./install-sh'

Makefile.am: installing `./depcomp'

[root@localhost main]# **ls**

aclocal.m4      config.h.in   configure.in~ main.c        Makefile.in

autom4te.cache configure     **depcomp        Makefile.am missing**

autoscan.log    configure.in **install-sh**     Makefile.am~

[root@localhost main]#

------------------------------------------------

第八步

运行**configure**，在这一步中，通过运行自动配置设置文件configure，把Makefile.in变成了最终的Makefile。

运行的结果如下：

------------------------------------------------

[root@localhost main]# **./configure**

checking for a BSD-compatible install... /usr/bin/install -c

checking whether build environment is sane... yes

checking for a thread-safe mkdir -p... /bin/mkdir -p

checking for gawk... gawk

checking whether make sets $(MAKE)... yes

checking for gcc... gcc

checking for C compiler default output file name... a.out

checking whether the C compiler works... yes

checking whether we are cross compiling... no

checking for suffix of executables...

checking for suffix of object files... o

checking whether we are using the GNU C compiler... yes

checking whether gcc accepts -g... yes

checking for gcc option to accept ISO C89... none needed

checking for style of include used by make... GNU

checking dependency style of gcc... gcc3

configure: creating ./config.status

**config.status: creating Makefile**

config.status: creating config.h

config.status: executing depfiles commands

[root@localhost main]# ls

aclocal.m4      config.h.in    configure.in   main.c        Makefile.in

autom4te.cache config.log     configure.in~ Makefile     **missing**

autoscan.log    config.status **depcomp**        Makefile.am  **stamp-h1**

**config.h**        configure      **install-sh**     Makefile.am~

[root@localhost main]#

------------------------------------------------

第九步

运行 **make**，对配置文件Makefile进行测试一下

此时的状态如下:

------------------------------------------------

[root@localhost main]# **make**

cd . && /bin/sh /root/project/main/missing --run aclocal-1.10

cd . && /bin/sh /root/project/main/missing --run automake-1.10 --foreign

cd . && /bin/sh /root/project/main/missing --run autoconf

/bin/sh ./config.status --recheck

running CONFIG\_SHELL=/bin/sh /bin/sh ./configure   --no-create --no-recursion

checking for a BSD-compatible install... /usr/bin/install -c

checking whether build environment is sane... yes

checking for a thread-safe mkdir -p... /bin/mkdir -p

checking for gawk... gawk

checking whether make sets $(MAKE)... yes

checking for gcc... gcc

checking for C compiler default output file name... a.out

checking whether the C compiler works... yes

checking whether we are cross compiling... no

checking for suffix of executables...

checking for suffix of object files... o

checking whether we are using the GNU C compiler... yes

checking whether gcc accepts -g... yes

checking for gcc option to accept ISO C89... none needed

checking for style of include used by make... GNU

checking dependency style of gcc... gcc3

configure: creating ./config.status

/bin/sh ./config.status

config.status: creating Makefile

config.status: creating config.h

config.status: config.h is unchanged

config.status: executing depfiles commands

cd . && /bin/sh /root/project/main/missing --run autoheader

rm -f stamp-h1

touch config.h.in

make all-am

make[1]: Entering directory `/root/project/main'

gcc -DHAVE\_CONFIG\_H -I.     -g -O2 -MT main.o -MD -MP -MF .deps/main.Tpo -c -o main.o main.c

mv -f .deps/main.Tpo .deps/main.Po

gcc -g -O2   -o main main.o

cd . && /bin/sh ./config.status config.h

config.status: creating config.h

config.status: config.h is unchanged

make[1]: Leaving directory `/root/project/main'

[root@localhost main]# **ls**

aclocal.m4      autoscan.log config.h.in config.status configure.in   depcomp    **main    main.o**    Makefile.am   Makefile.in stamp-h1

autom4te.cache config.h      config.log   configure      configure.in~ install-sh main.c Makefile Makefile.am~ missing

[root@localhost main]#

------------------------------------------------

第十步

运行生成的文件 main：

------------------------------------------------

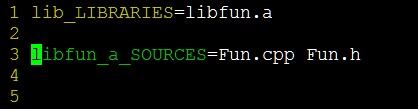
[root@localhost main]#**./main**

**例子：（同时生成静态库，动态库）**

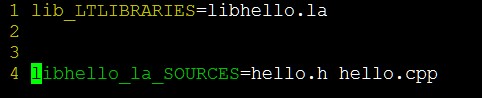
 假设我们的项目根目录为project，下面有三个子目录hello，fun，src，其中hello目录包含hello.cpp和hello.h，用来生成动态链接库。fun目录下面有Fun.cpp和Fun.h，用来生成静态链接库，src目录下面含有man.cpp用来调用其他两个库。

    实现的过程如下：

* 在fun目录下新建Makefile.am，文件内容和生成静态库一样。



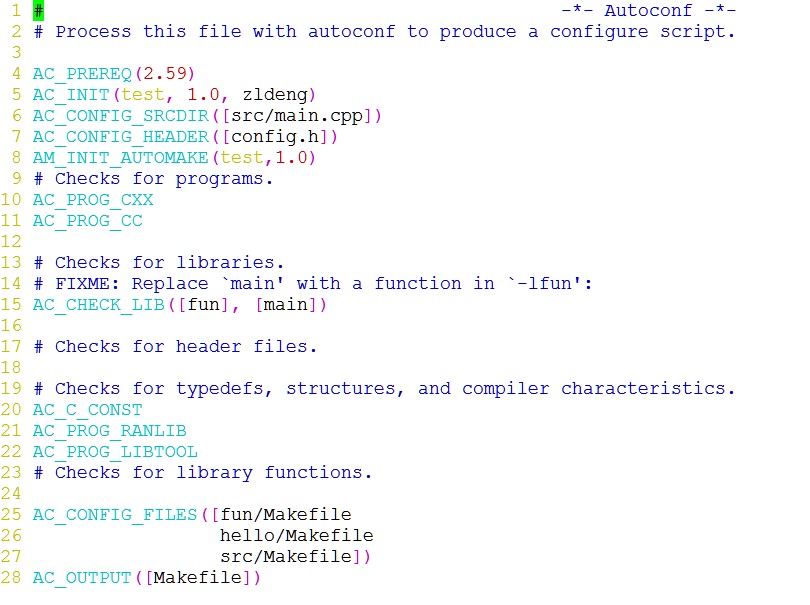
* 在hello目录下新建Makefile.am文件，文件内容为：



* 在src目录下新建Makefile.am文件，文件内容为：



* 在project目录下运行autoscan命令，将configure.scan改为configure.in文件，修改文件内容：



          这里我同时加了AC\_PROG\_RANLIB和AC\_PROG\_LIBTOOL，其实只需要加入AC\_PROG\_LIBTOOL即可，加入的AC\_PROG\_RANLIB其实也会被AC\_PROG\_LIBTOOL覆盖。

* 运行aclocal、autoiconf、autoheader、automake -a。
* 运行./configure --prefix=‘pwd’
* 运行make && make install命令。

这时候fun和hello目录下的库文件就生成了。src下面就生成了可执行文件。