转载：http://www.cnblogs.com/ericdream/archive/2011/12/09/2282359.html

前言：

       这次task，我大概用了4天的时间去完成。四天的时间内，我不停地去查资料，不停的去做小Demo，不停的总结，终于做完了这次的作业。下面的内容记录了我做这次Makefile作业的学习之路。

一、    相关概念的介绍

什么是Makefile？怎么书写Makefile？竟然有工具可以自动生成Makefile?怎么生成啊？开始的时候，我有这么多疑问，所以，必须得先把基本的概念搞个清楚。

1. Makefile

　　makefile用来定义整个工程的编译规则。一个工程中的源文件计数，其按类型、功能、模块分别放在若干个目录中，makefile定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作，因为 makefile就像一个Shell脚本一样，其中也可以执行操作系统的命令。

　　makefile带来的好处就是——“自动化编译”，一旦写好，只需要一个make命令，整个工程完全自动编译，极大的提高了软件开发的效率。make是一个命令工具，是一个解释makefile中指令的命令工具，一般来说，大多数的IDE都有这个命令，比如：Delphi的make，Visual C++的nmake，Linux下GNU的make。可见，makefile都成为了一种在工程方面的编译方法。

2. Autoconf

       Autoconf是一个用于生成可以自动地配置软件源代码包以适应多种Unix类系统的 shell脚本的工具。由Autoconf生成的配置脚本在运行的时候与Autoconf是无关的， 就是说配置脚本的用户并不需要拥有Autoconf。

       对于每个使用了Autoconf的软件包，Autoconf从一个列举了该软件包需要的，或者可以使用的系统特征的列表的模板文件中生成配置脚本。在shell代码识别并响应了一个被列出的系统特征之后，Autoconf允许多个可能使用（或者需要）该特征的软件包共享该特征。 如果后来因为某些原因需要调整shell代码，就只要在一个地方进行修改； 所有的配置脚本都将被自动地重新生成以使用更新了的代码。

3. Automake

       Automake是一个从文件`Makefile.am'自动生成`Makefile.in' 的工具。每个`Makefile.am'基本上是一系列make的宏定义 （make规则也会偶尔出现）。生成的`Makefile.in'服从GNU Makefile标准。GNU Makefile标准文档长、复杂，而且会发生改变。Automake的目的就是解除个人GNU维护者维护Makefile的负担 （并且让Automake的维护者来承担这个负担）。典型的Automake输入文件是一系列简单的宏定义。处理所有这样的文件以创建 `Makefile.in'。在一个项目（project）的每个目录中通常包含一个 `Makefile.am'。Automake在几个方面对一个项目做了限制；例如它假定项目使用Autoconf并且对`configure.in'的内容施加了某些限制。

       Automake支持三种目录层次： “flat”、“shallow”和“deep”。一个flat（平）包指的是所有文件都在一个目录中的包。为这类包提供的`Makefile.am' 缺少宏SUBDIRS。这类包的一个例子是termutils。一个deep（深）包指的是所有的源代码都被储存在子目录中的包；顶层 目录主要包含配置信息。GNU cpio 是这类包的一个很好的例子，GNU tar也是。deep包的顶层`Makefile.am'将包括 宏SUBDIRS，但没有其它定义需要创建的对象的宏。一个shallow（浅）包指的是主要的源代码储存在顶层目录中，而 各个部分（典型的是库）则储存在子目录中的包。Automake本身就是这类包（GNU make也是如此，它现在已经不使用automake）。

下面，就以这三种目录层次结构给大家介绍

二、    Flat目录结构：

1. 目录结构：

Helloworld

|-mytest.h

|-mytest.c

|-mymain.c

　　顶级目录helloworld，该目录下存在三个文件。mytest.h头文件声明了sayhello()方法；mytest.c中实现了sayhello()方法；mymain.c中的main调用了sayhello()方法。

2. 执行步骤：

2.1. Autoscan

　　 在helloworld目录下执行autoscan命令，其中生成一个configure.scan的文件。

2.2. 将configure.scan文件更名为configure.in文件

2.3. 打开configure.in文件，修改文件内容

复制代码

1 # -\*- Autoconf -\*-  
 2 # Process this file with autoconf to produce a configure script.  
 3   
 4 #AC\_INIT([2.68])  
 5 AC\_INIT([hello], [1.0], [\*\*@126.com])  
 6 AC\_CONFIG\_SRCDIR([mymain.c])  
 7 #AC\_CONFIG\_HEADERS([config.h])  
 8   
 9 AM\_INIT\_AUTOMAKE(hello, 1.0)  
10   
11 # Check for programs  
12 AC\_PROG\_CC  
13   
14 # Check for libraries  
15 # Check for header files  
16 # Check for typedefs, structures, and compiler characteristics.  
17 # Check for library functions.  
18   
19 AC\_OUTPUT(Makefile)

复制代码

2.4. 然后分别执行以下两个命令：

aclocal

autoconf

2.5. 在helloworld文件夹下创建一个名为Makefile.am的文件，并输入一下内容：

1 AUTOMAKE\_OPTIONS=foreign  
2 bin\_PROGRAMS=hello  
3 hello\_SOURCES=mymain.c mytest.c mytest.h

2.6. 执行命令“automake --add-missing”，automake 会根据Makefile.am 文件产生一些文件，其中包含最重要的Makefile.in

2.7. 执行“./configure”命令生成Makefile文件

2.8. 执行“make”命令来编译hello.c程序，从而生成可执行程序hello。生成可执行程序hello后，执行“./hello”。

 哈哈，一定看到你想要的结果了吧。

三、    shallow目录结构

1. 目录结构

helloworld

|-mymain.c

|head

||-mytest.h

||-mytest.c

 　　顶级目录helloworld，该目录下存在一个主文件mymain.c和一个目录head。head目录中，mytest.h头文件声明了sayhello()方法；mytest.c中实现了sayhello()方法；mymain.c中的main调用了sayhello()方法。

2. 执行步骤：

2.1. 在顶层目录下运行autoscan产生configure.scan文件

2.2. 将configure.scan文件更名为configure.in文件

2.3. 打开configure.in文件，修改文件内容

复制代码

1 # -\*- Autoconf -\*-  
 2 # Process this file with autoconf to produce a configure script.  
 3   
 4 #AC\_INIT([2.68])  
 5 AC\_INIT([hello], [1.0], [\*\*@126.com])  
 6 AC\_CONFIG\_SRCDIR([mymain.c])  
 7 #AC\_CONFIG\_HEADERS([config.h])  
 8   
 9 AM\_INIT\_AUTOMAKE(hello, 1.0)  
10   
11 # Check for programs  
12 AC\_PROG\_CC  
13 #使用静态库编译，需要此宏定义  
14 AC\_PROG\_RAMLIB  
15   
16 # Check for libraries  
17 # Check for header files  
18 # Check for typedefs, structures, and compiler characteristics.  
19 # Check for library functions.  
20   
21 AC\_OUTPUT(Makefile head/Makefile)

复制代码

2.4. 然后分别执行以下两个命令：

aclocal

autoconf

2.5. 在head文件夹下创建Makefile.am文件，内容如下：

1 AUTOMAKE\_OPTIONS=foreign  
2 noinst\_LIBRARIES=libmytest.a  
3 libmytest\_a\_SOURCES=mytest.h mytest.c

2.6. 在helloworld文件夹下创建Makefile.am文件，内容如下：

1 AUTOMAKE\_OPTIONS=foreign  
2 SUBDIRS=head  
3 bin\_PROGRAMS=hello  
4 hello\_SOURCES=mymain.c  
5 hello\_LDADD=head/mytest.a

2.7. 执行命令“automake –add-missing”，automake会根据Makefile.am 文件产生一些文件，其中包含最重要的Makefile.in

2.8. 执行“./configure”命令生成Makefile文件

2.9. 执行“make”命令来编译hello.c程序，从而生成可执行程序hello。生成可执行程序hello后，执行“./hello”。

哈哈，shallow的目录结构也搞定了哦~~

四、    Deep目录结构

1. 目录结构

helloworld

|head

||-mytest.h

||-mytest.c

|src

||-mymain.c

　　顶级目录helloworld，该目录下存在两个目录src和head。Head目录中，mytest.h头文件声明了sayhello()方法；mytest.c中实现了sayhello()方法；src目 录中的mymain.c中的main调用了sayhello()方法。

2. 执行步骤

2.1.  在顶层目录下运行autoscan产生configure.scan文件

2.2.  将configure.scan文件更名为configure.in文件

2.3.  打开configure.in文件，修改文件内容

复制代码

1 # -\*- Autoconf -\*-  
 2 # Process this file with autoconf to produce a configure script.  
 3   
 4 #AC\_INIT([2.68])  
 5 AC\_INIT([hello], [1.0], [\*\*@126.com])  
 6 AC\_CONFIG\_SRCDIR([src/mymain.c])  
 7 #AC\_CONFIG\_HEADERS([config.h])  
 8   
 9 AM\_INIT\_AUTOMAKE(hello, 1.0)  
10   
11 # Check for programs  
12 AC\_PROG\_CC  
13 #使用静态库编译，需要此宏定义  
14 AC\_PROG\_RAMLIB  
15   
16 # Check for libraries  
17 # Check for header files  
18 # Check for typedefs, structures, and compiler characteristics.  
19 # Check for library functions.  
20   
21 AC\_OUTPUT(Makefile head/Makefile src/Makefile)

复制代码

2.4.  然后分别执行以下两个命令：

aclocal

autoconf

2.5.  在head文件夹下创建Makefile.am文件，内容如下：

1 AUTOMAKE\_OPTIONS=foreign  
2 noinst\_LIBRARIES=libmytest.a  
3 libmytest\_a\_SOURCES=mytest.h mytest.c

2.6.  在src文件夹下创建Makefile.am文件，内容如下：

1 AUTOMAKE\_OPTIONS=foreign  
2 bin\_PROGRAMS=hello  
3 hello\_SOURCES=mymain.c  
4 hello\_LDADD=../head/libmytest.a

2.7.  在helloworld文件夹下创建Makefile.am文件，内容如下：

1 AUTOMAKE\_OPTIONS=foreign  
2 SUBDIRS=head src

2.8.  执行命令“automake –add-missing”，automake会根据Makefile.am 文件产生一些文件，其中包含最重要的Makefile.in

2.9.  执行“make”命令来编译hello.c程序，从而生成可执行程序hello。生成可执行程序hello后，执行“./hello”。

 哈哈，deep目录下的编译与链接也搞定了！

五、    总结：

归纳一下以上所有例子的流程：

（1）在存放源代码的顶层目录下执行autoscan命令生成configure.scan文件。

（2）将configure.scan文件改名为configure.in，并对其默认配置进行修改。

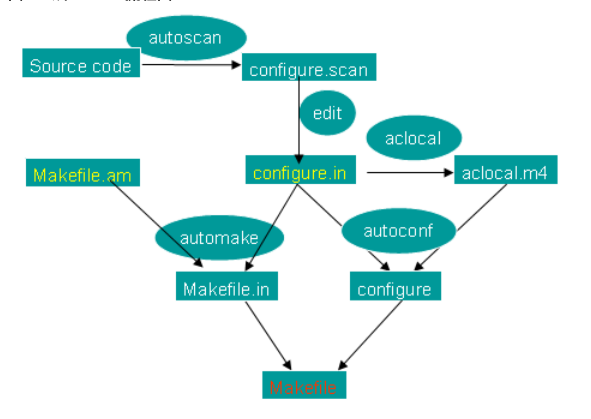
（3）执行aclocal、autoconf两个命令，分别生成aclocal.m4、configure文件。

（4）在每个目录下创建一个名为Makefile.am的文件，并输入相应的内容。

（5）执行automake --add-missing，它根据Makefile.am文件，生成Makefile.in。

（6）执行./configure脚本文件，它根据Makefile.in文件，生成最终的Makefile文件。

（7）生成Makefile之后，执行“make”编译工程并且生成可执行程序。



六、    能力进阶

以上的的程序还只是处于初级阶段，并且生成的是静态库。我们可以发现，用autoconf和automake生成Makefile的关键在于configure.in和Makefile.am的文件的书写。所以，要想使自己的功力更上一层，需要熟悉autoconf和automake这两个工具的使用，其中有很多重要的宏需要我们了解。这里时具体的参考手册：

autoconf手册

       英文版：<http://www.gnu.org/software/autoconf/manual/autoconf.html>

       中文版：<http://www.linuxforum.net/books/autoconf.html>

automake手册

       中文版：<http://www.linuxforum.net/books/automake.html>

七、    Configure.in文件解析

autoconf是用来产生“configure”文件的工具。“configure”是一个Shell脚本，它可以自动设定一些编译参数使程序能够在不同平台上进行编译。autoconf读取configure.in 文件然后产生’configure’这个Shell脚本。

configure.in 文件的内容是一系列GNU m4 的宏，这些宏经autoconf处理后会变成检查系统特性的Shell脚本。configure.in文件中宏的顺序并没有特别的规定，但是每一个configure.in 文件必须以宏AC\_INIT开头，以宏AC\_OUTPUT结束。一般可先用autoscan这个工具扫描原始文件以产生一个configure.scan 文件，再对configure.scan 作些修改，从而生成 configure.in 文件。

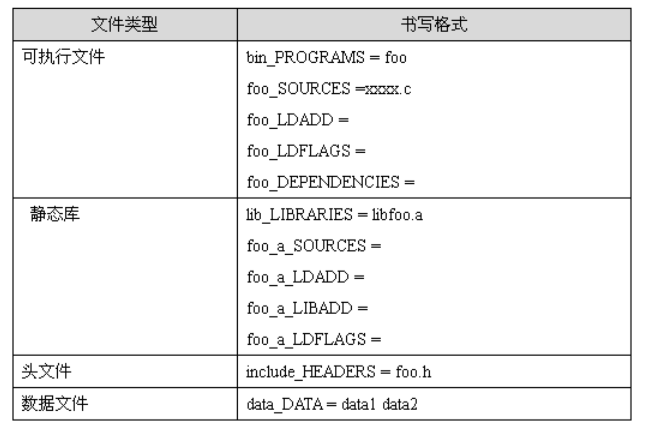
configure.in 文件中一些宏的含义如下：

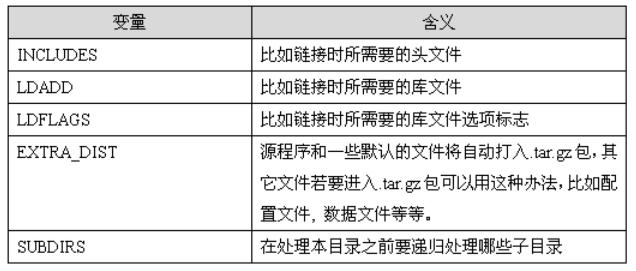
|  |  |
| --- | --- |
| #或dnl | #或dnl后面的内容作为注释不会被处理，它们是注释的起始标志 |
| AC\_INIT([FULL-PACKAGE-NAME], [VERSION], [BUG-REPORT-ADDRESS]) |  |
| AM\_INIT\_AUTOMAKE(PACKAGE,VERSION) | 这个是后面运行automake命令所必需的宏，PACKAGE指明要产生软件的名称，VERSION 是其版本号 |
| AC\_PROG\_CC | 检查系统可用的C编译器，若源代码是用C语言编写的就需要这个宏 |
| AC\_OUTPUT(FILE) | 设置configure命令所要产生的文件。我们最终期望产生Makefile  这个文件，因此一般将其设置为AC\_OUTPUT(Makefile) |

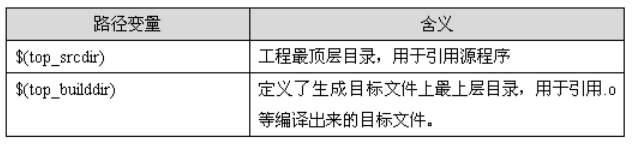
在运行automake命令时，还需要一些其他的宏，这些额外的宏由aclocal产生。执行aclocal会产生aclocal.m4文件，如果没有特别的要求，无需修改它。用 aclocal产生的宏将会提示automake如何动作。

另一个重要的文件是Makefile.am。automake根据configure.in中的宏并在perl的帮助下把Makefile.am转成Makefile.in文件。Makefile.am 文件定义所要产生的目标。

八、    Makefile.am







　　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。

automake设置了默认的安装路径：

1) 标准安装路径

默认安装路径为：$(prefix) = /usr/local，可以通过./configure --prefix=<new\_path>的方法来覆盖。

其它的预定义目录还包括：bindir = $(prefix)/bin, libdir = $(prefix)/lib, datadir = $(prefix)/share, sysconfdir = $(prefix)/etc等等。

2) 定义一个新的安装路径

比如test, 可定义builddir = $(prefix)/build, 然后test\_LIBRARIES =mytest.h mytest.c，则mytest.h mytest.c 会作为静态库安装到$(prefix)/build目录下。

九、    如何使用产生的Makefile文件

执行configure脚本文件所产生的Makefile文件有几个预定的选项可供使用：

make all：产生设定的目标，即生成所有的可执行文件。使用make也可以达到此目的。

make clean：删除之前编译时生成的可执行文件及目标文件（形如\*.o的中间文件）。

make distclean：除了删除可执行文件和目标文件以外，把configure所产生的 Makefile文件也清除掉。通常在发布软件前执行该命令。

make install：将使用make all或make命令产生的可执行文件以软件的形式安装到系统中。若使用bin\_PROGRAMS宏，程序将会被安装到 /usr/local/bin下，否则安装到预定义的目录下。

make dist：将程序和相关的文档包装为一个压缩文档以供发布。执行完该命令，在当前目录下会产生一个名为PACKAGE-VERSION.tar.gz的文件。PACKAGE 和 VERSION 这两个参数是来自configure.in文件中的AM\_INIT\_AUTOMAKE(PACKAGE,

VERSION)。如在上个例子中执行make dist命令，会产生名为“hello-1.0.tar.gz”的文件。

make distcheck：与make dist类似，但是加入了检查包装以后的压缩文件是否正常。

十、    动态库编译

需要在Makefile.am中指定：

lib\_LTLIBRARIES=libhello.al

libhello\_al\_SOURCES=mytest.h mytest.c

在根目录下的configure.in中加AC\_PROG\_LIBTOOL

动态库编译之前，需要安装libtool工具：apt-get install libtool。

若出现：“required file `./ltmain.sh' not found”错误，是因为libtool的配置问题。

解决方法：

$libtoolize --automake --debug --copy –force