原文地址：<https://blog.csdn.net/initphp/article/details/43705765>

对原文有做修改

安装包

sudo apt-get install autoconf

### 关于Autotools

**autotools工具**，专门用来生成Makefile

Autotools并不是一个工具，而是一系列工具：

1. autoscan

2. aclocal

3. autoconf

4. autoheader

5. automake

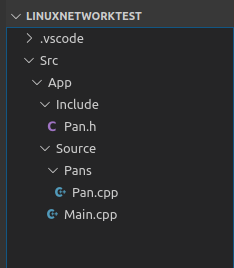
### c源文件同一目录下Autotools的使用

如果你的源文件都放在同一个目录下面，那么使用Autotools的时候会相对简单很多。

下面会按照步骤来实现同一目录下的Autotools工具的使用。

#### 1. 源代码例子

项目结构



App文件夹当作一个模块

切换到项目目录（LinuxNetWorkTest）

#### 2. Autoscan命令

第一步，执行**autoscan**命令。

这个命令主要用于扫描工作目录，并且生成**configure.scan文件**。

configure.scan需要重命名为**configure.ac**，然后编辑这个配置文件，如下。

# -\*- Autoconf -\*-

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

# 声明autoconf要求的版本号

AC\_PREREQ([2.69])

# 定义软件名称、版本号、联系方式

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

# 必须要的，参数为软件名称和版本号

AM\_INIT\_AUTOMAKE(Main, 1.0)

# 宏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([Src/App/Include/Pan.h])

# 宏用于生成config.h文件，以便 autoheader 命令使用。

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

# Checks for programs.

# 指定编译器

AC\_PROG\_CXX

AC\_PROG\_CC

# Checks for libraries.

# Checks for header files.

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.

# 指定要生成的的Makefile文件，不同文件夹下的Makefile通过空格分隔。例如：AC\_CONFIG\_FILES([Makefile, src/Makefile])

# 这里我们需要为App模块生成一个Makefile

AC\_CONFIG\_FILES([Src/App/Makefile])

# 用来设定 configure 所要产生的文件，如果是makefile，configure 会把它检查出来的结果带入makefile.in文件产生合适的makefile。

AC\_OUTPUT

configure.ac标签说明：

|  |  |
| --- | --- |
| 标签 | 说明 |
| AC\_PREREQ | 声明autoconf要求的版本号 |
| AC\_INIT | 定义软件名称、版本号、联系方式 |
| AM\_INIT\_AUTOMAKE | 必须要的，参数为软件名称和版本号 |
| AC\_CONFIG\_SCRDIR | 宏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.。此处为当前目录下main.c。 |
| AC\_CONFIG\_HEADER | 宏用于生成config.h文件，以便 autoheader 命令使用。 |
| AC\_PROG\_CC | 指定编译器，默认GCC |
| AC\_CONFIG\_FILES | 生成相应的Makefile文件，不同文件夹下的Makefile通过空格分隔。例如：AC\_CONFIG\_FILES([Makefile, src/Makefile]) |
| AC\_OUTPUT | 用来设定 configure 所要产生的文件，如果是makefile，configure 会把它检查出来的结果带入makefile.in文件产生合适的makefile。 |

#### 3. Aclocal命令

执行**aclocal**命令。扫描 configure.ac 文件生成 aclocal.m4文件, 该文件主要处理本地的宏定义

#### 4. Autoconf命令

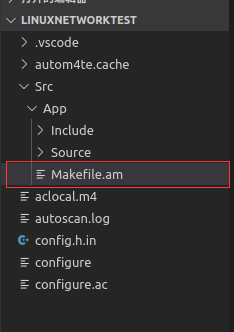
执行**autoconf**命令。这个命令将 configure.ac 文件中的宏展开，生成 configure 脚本。这个过程可能要用到aclocal.m4中定义的宏。

#### 5. Autoheader命令

执行**autoheader**命令。该命令生成 config.h.in 文件。

#### 6. 创建Makefile.am文件

创建**Makefile.am**文件。



Makefile.am的目录与我们App模块的Makefile同在一个目录

Automake工具会根据 configure.in 中的参量把 Makefile.am 转换成 Makefile.in 文件。

Makefile.am这个文件非常重要，定义了一些生成Makefile的规则

Makefile.am：

# automake提供了3中软件等级:foreign, gnu和gnits, 供用户选择。默认级别是gnu.1. 在本例中， 使用了foreign等级, 它只检测必须的文件。

AUTOMARK\_OPTIONS = foreign

# 生成可执行文件的名称

bin\_PROGRAMS = Main

# 生成Main需要用到的源码文件，这里Main\_SOURCES中的Main与上面定义的相同

Main\_SOURCES = ./Source/Main.cpp ./Source/Pans/Pan.cpp ./Include/Pan.h

#### 7. Automake命令

执行**automake --add-missing**命令。该命令生成 Makefile.in 文件。使用选项 "--add-missing" 可以让 Automake 自动添加一些必需的脚本文件。

#### 8. configure命令

执行**./configure**这个命令很熟悉吧。./**configure**主要把 Makefile.in 变成最终的 Makefile 文件。

#### 9. make命令

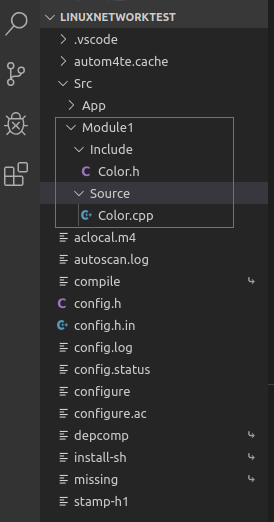
进入App目录，执行make命令，执行make命令后，就生成了可执行文件。

### c源文件不同目录下Autotools的使用

在多模块的情况下，我们会在每个模块生成Makefile，入口模块会引用其他模块

#### 1. 源代码例子

新增模块，如下



#### 2. 创建Makefile.am文件

在Module1下创建创建**Makefile.am**文件。

**# 生成的静态库文件名称，noinst加上之后是只编译，不安装到系统中**

**noinst\_LIBRARIES=libModule1.a**

**# 这个静态库文件需要用到的依赖**

**libModule1\_a\_SOURCES=./Source/Color.cpp ./Include/Color.h**

**# 导入需要外部依赖的头文件**

**# include\_HEADERS=../include/common.h ../include/sum.h ../include/get.h ../include/val.h**

**说明：**不加**include\_HEADERS**也是可以运行的，但是在使用make dist打包命令后，并不会将../include/文件夹打包进去，所以还是需要加上include\_HEADERS。

修改App下的**Makefile.am**文件。

# automake提供了3中软件等级:foreign, gnu和gnits, 供用户选择。默认级别是gnu.1. 在本例中， 使用了foreign等级, 它只检测必须的文件。

AUTOMARK\_OPTIONS = foreign

# 先扫描的目录

# 我们的App模块依赖于Module1模块，所有这里指定模块依赖

SUBDIRS = ../Module1

# 生成可执行文件的名称

bin\_PROGRAMS = Main

# 生成Main需要用到的源码文件，这里Main\_SOURCES中的Main与上面定义的相同

Main\_SOURCES = ./Source/Main.cpp ./Source/Pans/Pan.cpp ./Include/Pan.h

# 静态连接方式 连接Module1下生成的libModule1.a文件

Main\_LDADD = ../Module1/libModule1.a

# 如果有动态连接应指定下面的属性，如下表示用到了lib event和pthread，这个是动态连接

# LIBS = -l pthread -l event

#### 3. 执行Autoscan命令

修改configure.ac

# -\*- Autoconf -\*-

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

# 声明autoconf要求的版本号

AC\_PREREQ([2.69])

# 定义软件名称、版本号、联系方式

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

# 必须要的，参数为软件名称和版本号

AM\_INIT\_AUTOMAKE(Main, 1.0)

# 如果我们使用的静态库，应该如下来生成静态库

AC\_PROG\_RANLIB

# 宏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([Src/App/Include/Pan.h])

# 宏用于生成config.h文件，以便 autoheader 命令使用。

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

# Checks for programs.

# 指定编译器

AC\_PROG\_CXX

AC\_PROG\_CC

# Checks for libraries.

# Checks for header files.

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.

# 指定要生成的的Makefile文件，不同文件夹下的Makefile通过空格分隔。例如：AC\_CONFIG\_FILES([Makefile, src/Makefile])

# 这里我们需要为App模块生成一个Makefile

AC\_CONFIG\_FILES([Src/App/Makefile Src/Module1/Makefile])

# 用来设定 configure 所要产生的文件，如果是makefile，configure 会把它检查出来的结果带入makefile.in文件产生合适的makefile。

AC\_OUTPUT

#### 4. Aclocal命令

执行**aclocal**命令。扫描 configure.ac 文件生成 aclocal.m4文件,

#### 5. Autoconf命令

执行**autoconf**命令。这个命令将 configure.ac 文件中的宏展开，生成 configure 脚本。

#### 6. Autoheader命令

第四步，执行**autoheader**命令。该命令生成 config.h.in 文件。

#### 7. Automake命令

执行**automake --add-missing**命令。该命令生成 Makefile.in 文件。

#### 8. configure命令

执行**./configure**命令。./**configure**主要把 Makefile.in 变成最终的 Makefile 文件。

#### ****9. make命令****

进入App文件夹，执行**make**命令。make执行后，会生成的可执行文件。

**根据上面两个示例我们知道，生成Makefile只需要configure.ac和Makefile.am文件，于是我们根据这两个文件的改动来合并一下我们的命令**

如果改动了configure.ac文件，我们执行如下命令更新makefile

**aclocal && autoconf && autoheader && automake --add-missing && ./configure**

如果只改动Makefile.am文件，我们执行如下命令更新makefile

**automake --add-missing && ./configure**

### Autotools运行流程

#### 流程总结：

1. 执行**autoscan**命令。这个命令主要用于扫描工作目录，并且生成**configure.scan文件**。

2. 修改configure.scan为**configure.ac**文件，并且修改配置内容。

3. 执行**aclocal**命令。扫描 configure.ac 文件生成 aclocal.m4文件。

4. 执行**autoconf**命令。这个命令将 configure.ac 文件中的宏展开，生成 configure 脚本。

5. 执行**autoheader**命令。该命令生成 config.h.in 文件。

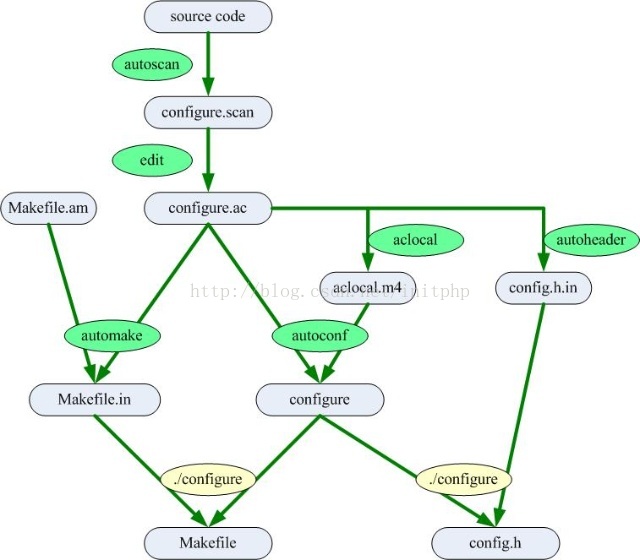
6. 新增Makefile.am文件，修改配置内容

7. 执行**automake --add-missing**命令。该命令生成 Makefile.in 文件。

8. 执行 **./congigure**命令。将Makefile.in命令生成Makefile文件。

9. 执行**make**命令。生成可执行文件。

#### 流程图：



### Make命令详解

1. make命令：编译文件。make命令主要通过Makefile文件生成可执行文件。

2. make clean命令。清楚编译的文件，包括目标文件\*.o和可执行文件

3. make install 命令把目标文件安装到系统中。默认安装到/usr/local/bin目录下面。

4. make uninstall 命令，把目标文件从系统中卸载。

5. make dist 命令，打包发布。

**如何使用发布的文件：**

1. 下载到“hello-1.0.tar.gz”压缩文档

2. 使用“ tar -zxvf hello-1.0.tar.gz ”命令解压

3. 使用 “./configure” 命令，主要是生成Makefile命令，已经一些配置初始化。

4. 使用 “make” 命令编译源代码文件生成软件包。

5. 使用“make install ”命令来安装编译后的软件包到系统中。

### Makefile.am属性解读

#### 1. 可执行文件类型（我们的模块是可执行文件，如我们的App模块）

|  |  |
| --- | --- |
| 书写格式 | 说明 |
| bin\_PROGRAMS | 生成的可执行文件名称。如果生成的可执行文件名称为多个，则可以通过空格的方式分隔。 当运行make install命令的时候，可执行文件会默认安装到linux系统的**/usr/local/bin**目录下面 **noinst\_PROGRAMS**：如果make install的时候不想被安装，可以使用noinst\_PROGRAMS命令。 例子：bin\_PROGRAMS=hello |
| hello\_SOURCES | 编译成可执行文件所依赖的.c源文件。多个源文件之间用空格分隔。hello为可执行文件名称。 |
| hello\_LDADD | 编译成可执行文件过程中，连接所需的库文件，包括**\*.so**的动态库文件和**.a**的静态库文件。 |
| hello\_LDFLAGS | 连接的时候所需库文件的标识 |

#### 2. 静态库文件类型（如我们的Module1模块）

模块会被编译成静态链接库 \*.a格式的文件。

注意：**静态库使用中，需要对configure.ac中加入AC\_PROG\_RANLIB**

|  |  |
| --- | --- |
| 书写格式 | 说明 |
| noinst\_LIBRARIES | 生成静态库（\*.a）或者动态库(\*.so)的名称。 库文件一般以lib\*.a或者lib\*.so来命名。 **noinst\_LIBRARIES**:当运行make install的时候，库文件不会被安装到linux默认的/usr/local/lib目录下。 **lib\_LIBRARIES**:当运行make intsall的时候，则会被安装到**/usr/local/lib**目录下。 |
| libpro\_a\_SOURCES | 所需的源文件。多个文件用空格分开。 |
| libpro\_a\_LDADD | 加载所需的库文件。 |
| libpro\_a\_LDFLAGS | 编译的时候的连接标识。 |

#### 3. 头文件

我们一般需要导入一些\*.h的头文件，如果你在Makefile.am中没有标识需要导入的头文件，可能在make dist打包的时候出现问题，头文件可能不会被打进包里面。

include\_HEADERS=../include/common.h ../include/sum.h ../include/get.h ../include/val.h #可以将头文件引入

make install，头文件默认会被安装到linux系统**/usr/local/include**

#### 4. 数据文件

data\_DATA = data1 data2

#### 5. 常用变量



AUTOMAKE\_OPTIONS=foreign #软件等级

SUBDIRS=src #先扫描子目录，多个目录用空格隔开

LIBS = -l pthread -l event #动态连接库

EXTRA\_DIST = conf #打包一些配置文件

#### 6. 安装目录

我们知道，默认情况下，执行make install命令，则会将文件安装到/usr/local/bin   /usr/local/include  /usr/local/lib目录下面。

我们可以通过命令**./configure --prefix=** 生成Makefile文件的时候，配置make install命令执行的时候的文件安装路径。

下面这个例子，我们在执行make install的时候，程序会被安装到/home/test目录下面。

./configure --prefix=/home/test

执行下面一系列命令：

./configure --prefix=/home/test

sudo make install

我们可以进入/home/test目录下看到相应的bin文件已经生成：

[admin@localhost test]$ ls

bin include lib

下面这些变量是已经定义好的安装路径的变量。

用户也可以修改这些变量（makefile.am中）。

bindir = Build/Debug/bin

libdir = Build/Debug/lib

datadir=Build/Debug/share

sysconfdir=Build/Debug/etc

includedir=Build/Debug/include

假如我们有自定义的文件夹，我们需要将这个文件夹下的内容安装到安装目录，

则需要配置一个自定义的文件夹目录confdir

confdir=${prefix}/conf #conf为名称 dir为每个文件夹变量必须带上

conf\_DATA=conf/\* #这个是将conf/目录下的内容安装到confdir目录下

EXTRA\_DIST=conf #在make dist打包的时候 也要将扩展文件夹打包进去

confdir为需要创建的文件夹目录。

conf\_DATA为需要拷贝的文件内容到${prefix}/conf目录中去