C/C++ 通用 Makefile

◆

C/C++ 通用 Makefile

Generic Makefile for C/C++ Program

==========

Keywords: Makefile, make, Generic, C/C++

Author: whyglinux (whyglinux AT hotmail DOT com)

Date: 2006-03-04

==========

本文提供了一个用于对 C/C++ 程序进行编译和连接以产生可执行程序的通用 Makefile。

在使用 Makefile 之前，只需对它进行一些简单的设置即可；而且一经设置，即使以后对源程序文件有所增减一般也不再需要改动 Makefile。因此，即便是一个没有学习过 Makefile 书写规则的人，也可以为自己的 C/C++ 程序快速建立一个可工作的 Makefile。

这个 Makefile 可以在 GNU Make 和 GCC 编译器下正常工作。但是不能保证对于其它版本的 Make 和编译器也能正常工作。

如果你发现了本文中的错误，或者对本文有什么感想或建议，可通过 whyglinux AT hotmail DOT com 邮箱和作者联系。

此 Makefile 的使用方法如下：

- 程序目录的组织

尽量将自己的源程序集中在一个目录中，并且把 Makefile 和源程序放在一起，这样用起来比较方便。当然，也可以将源程序分类存放在不同的目录中。

在程序目录中创建一个名为 Makefile 的文本文件，将后面列出的 Makefile 的内容复制到这个文件中。（注意：在复制的过程中，Makfile 中各命令前面的 Tab 字符有可能被转换成若干个空格。这种情况下需要把 Makefile 命令前面的这些空格替换为一个 Tab。）

将当前工作目录切换到 Makefile 所在的目录。目前，这个 Makefile 只支持在当前目录中的调用，不支持当前目录和 Makefile 所在的路径不是同一目录的情况。

- 指定可执行文件

程序编译和连接成功后产生的可执行文件在 Makefile 中的 PROGRAM 变量中设定。这一项不能为空。为自己程序的可执行文件起一个有意义的名子吧。

指定源程序

要编译的源程序由其所在的路径和文件的扩展名两项来确定。由于头文件是通过包含来使用的，所以在这里说的源程序不应包含头文件。

程序所在的路径在 SRCDIRS 中设定。如果源程序分布在不同的目录中，那么需要在 SRCDIRS 中一一指定，并且路径名之间用空格分隔。

在 SRCEXTS 中指定程序中使用的文件类型。C/C++ 程序的扩展名一般有比较固定的几种形

式：.c、.C、.cc、.cpp、.CPP、.c++、.cp、或者.cxx（参见 man gcc）。扩展名决定了程序是 C 还是 C++ 程序：.c 是 C 程序，其它扩展名表示 C++ 程序。一般固定使用其中的一种扩展名即可。但是也有可能需要使用多种扩展名，这可以在 SOURCE_EXT 中一一指定，各个扩展名之间用空格分隔。

虽然并不常用，但是 C 程序也可以被作为 C++ 程序编译。这可以通过在 Makefile 中设置 CC = $(CXX) 和 CFLAGS = $(CXXFLAGS) 两项即可实现。

这个 Makefile 支持 C、C++ 以及 C/C++ 混合三种编译方式：

- 如果只指定 .c 扩展名，那么这是一个 C 程序，用 $(CC) 表示的编译命令进行编译和连接。
- 如果指定的是除 .c 之外的其它扩展名（如 .cc、.cpp、.cxx 等），那么这是一个 C++ 程序，用 $(CXX) 进行编译和连接。
- 如果既指定了 .c，又指定了其它 C++ 扩展名，那么这是 C/C++ 混合程序，将用 $(CC) 编译其中的 C 程序，用 $(CXX) 编译其中的

C++ 程序，最后再用 $(CXX) 连接程序。

这些工作都是 make 根据在 Makefile 中提供的程序文件类型（扩展名）自动判断进行的，不需要用户干预。

- 指定编译选项

编译选项由三部分组成：预处理选项、编译选项以及连接选项，分别由 CPPFLAGS、CFLAGS 与 CXXFLAGS、LDFLAGS 指定。

CPPFLAGS 选项可参考 C 预处理命令 cpp 的说明，但是注意不能包含 -M 以及和 -M 有关的选项。如果是 C/C++ 混合编程，也可以在这里设置 C/C++ 的一些共同的编译选项。

CFLAGS 和 CXXFLAGS 两个变量通常用来指定编译选项。前者仅仅用于指定 C 程序的编译选项，后者仅仅用于指定 C++ 程序的编译选项。其实也可以在两个变量中指定一些预处理选项（即一些本来应该放在 CPPFLAGS 中的选项），和 CPPFLAGS 并没有明确的界限。

连接选项在 LDFLAGS 中指定。如果只使用 C/C++ 标准库，一般没有必要设置。如果使用了非标准库，应该在这里指定连接需要的选项，如库所在的路径、库名以及其它联接选项。

现在的库一般都提供了一个相应的 .pc 文件来记录使用库所需要的预编译选项、编译选项和连接选项等信息，通过 pkg-config 可以动态提取这些选项。与由用户显式指定各个选项相比，使用 pkg-config 来访问库提供的选项更方便、更具通用性。在后面可以看到一个 GTK+ 程序的例子，其编译和连接选项的指定就是用 pkg-config 实现的。

- 编译和连接

上面的各项设置好之后保存 Makefile 文件。执行 make 命令，程序就开始编译了。

命令 make 会根据 Makefile 中设置好的路径和文件类型搜索源程序文件，然后根据文件的类型调用相应的编译命令、使用相应的编译选项对程序进行编译。

编译成功之后程序的连接会自动进行。如果没有错误的话最终会产生程序的可执行文件。

注意：在对程序编译之后，会产生和源程序文件一一对应的 .d 文件。这是表示依赖关系的文件，通过它们 make 决定在源程序文件变动之后要进行哪些更新。为每一个源程序文件建立相应的 .d 文件这也是 GNU Make 推荐的方式。

- Makefile 目标（Targets）

下面是关于这个 Makefile 提供的目标以及它所完成的功能：

make

编译和连接程序。相当于 make all。

- make objs

仅仅编译程序产生 .o 目标文件，不进行连接（一般很少单独使用）。

- make clean

删除编译产生的目标文件和依赖文件。

- make cleanall

删除目标文件、依赖文件以及可执行文件。

- make rebuild

重新编译和连接程序。相当于 make clean && make all。

关于这个 Makefile 的实现原理不准备详细解释了。如果有兴趣的话，可参考文末列出的“参考资料”。

Makefile 的内容如下：

```
[ - ]CODE:#############################################################################
#
# Generic Makefile for C/C++ Program
#
# Author: whyglinux (whyglinux AT hotmail DOT com)
# Date: 2006/03/04

# Description:
# The makefile searches in directories for the source files
# with extensions specified in , then compiles the sources
# and finally produces the , the executable file, by linking
# the objectives.

# Usage:
# $ make compile and link the program.
# $ make objs compile only (no linking. Rarely used).
# $ make clean clean the objectives and dependencies.
# $ make cleanall clean the objectives, dependencies and executable.
# $ make rebuild rebuild the program. The same as make clean && make all.
#============================================================================
=

## Customizing Section: adjust the following if necessary.
##============================================================================
=

# The executable file name.
# It must be specified.
# PROGRAM := a.out # the executable name
PROGRAM :=

# The directories in which source files reside.
# At least one path should be specified.
```

```
# SRCDIRS := . # current directory
SRCDIRS :=

# The source file types (headers excluded).
# At least one type should be specified.
# The valid suffixes are among of .c, .C, .cc, .cpp, .CPP, .c++, .cp, or .cxx.
# SRCEXTS := .c # C program
# SRCEXTS := .cpp # C++ program
# SRCEXTS := .c .cpp # C/C++ program
SRCEXTS :=

# The flags used by the cpp (man cpp for more).
# CPPFLAGS := -Wall -Werror # show all warnings and take them as errors
CPPFLAGS :=

# The compiling flags used only for C.
# If it is a C++ program, no need to set these flags.
# If it is a C and C++ merging program, set these flags for the C parts.
CFLAGS :=
CFLAGS +=

# The compiling flags used only for C++.
# If it is a C program, no need to set these flags.
# If it is a C and C++ merging program, set these flags for the C++ parts.
CXXFLAGS :=
CXXFLAGS +=

# The library and the link options ( C and C++ common).
LDFLAGS :=
LDFLAGS +=

## Implict Section: change the following only when necessary.
##==========================================================================
=
# The C program compiler. Uncomment it to specify yours explicitly.
#CC = gcc

# The C++ program compiler. Uncomment it to specify yours explicitly.
#CXX = g++

# Uncomment the 2 lines to compile C programs as C++ ones.
#CC = $(CXX)
#CFLAGS = $(CXXFLAGS)
```

```
# The command used to delete file.
#RM = rm -f

## Stable Section: usually no need to be changed. But you can add more.
##================================================================================
=
SHELL = /bin/sh
SOURCES = $(foreach d,$(SRCDIRS),$(wildcard $(addprefix $(d)/*,$(SRCEXTS))))
OBJS = $(foreach x,$(SRCEXTS), \
$(patsubst %$(x),%.o,$(filter %$(x),$(SOURCES))))
DEPS = $(patsubst %.o,%.d,$(OBJS))

.PHONY : all objs clean cleanall rebuild

all : $(PROGRAM)

# Rules for creating the dependency files (.d).
#---------------------------------------------------
%.d : %.c
@$(CC) -MM -MD $(CFLAGS) $<

%.d : %.C
@$(CC) -MM -MD $(CXXFLAGS) $<

%.d : %.cc
@$(CC) -MM -MD $(CXXFLAGS) $<

%.d : %.cpp
@$(CC) -MM -MD $(CXXFLAGS) $<

%.d : %.CPP
@$(CC) -MM -MD $(CXXFLAGS) $<

%.d : %.c++
@$(CC) -MM -MD $(CXXFLAGS) $<

%.d : %.cp
@$(CC) -MM -MD $(CXXFLAGS) $<

%.d : %.cxx
@$(CC) -MM -MD $(CXXFLAGS) $<

# Rules for producing the objects.
#---------------------------------------------------
```

```
    objs : $(OBJS)

    %.o : %.c
    $(CC) -c $(CPPFLAGS) $(CFLAGS) $
```

```
<

%.o : %.C
$(CXX) -c $(CPPFLAGS) $(CXXFLAGS) $<

%.o : %.cc
$(CXX) -c $(CPPFLAGS) $(CXXFLAGS) $<

%.o : %.cpp
$(CXX) -c $(CPPFLAGS) $(CXXFLAGS) $<

%.o : %.CPP
$(CXX) -c $(CPPFLAGS) $(CXXFLAGS) $<

%.o : %.c++
$(CXX -c $(CPPFLAGS) $(CXXFLAGS) $<

%.o : %.cp
$(CXX) -c $(CPPFLAGS) $(CXXFLAGS) $<

%.o : %.cxx
$(CXX) -c $(CPPFLAGS) $(CXXFLAGS) $<

# Rules for producing the executable.
#----------------------------------------------
$(PROGRAM) : $(OBJS)
ifeq ($(strip $(SRCEXTS)), .c) # C file
$(CC) -o $(PROGRAM) $(OBJS) $(LDFLAGS)
else # C++ file
$(CXX) -o $(PROGRAM) $(OBJS) $(LDFLAGS)
endif

-include $(DEPS)

rebuild: clean all

clean :
@$(RM) *.o *.d
```

```
cleanall: clean
@$(RM) $(PROGRAM) $(PROGRAM).exe

### End of the Makefile ## Suggestions are welcome ## All rights reserved ###
#################################################################################
```

下面提供两个例子来具体说明上面 Makefile 的用法。

例一 Hello World 程序

这个程序的功能是输出 Hello, world! 这样一行文字。由 hello.h、hello.c、main.cxx 三个文件组成。前两个文件是 C 程序，后一个是 C++ 程序，因此这是一个 C 和 C++ 混编程序。

[-]CODE:/* File name: hello.h

```
* C header file
*/

#ifndef HELLO_H
#define HELLO_H

#ifdef __cplusplus
extern "C" {
#endif

void print_hello();

#ifdef __cplusplus
}
#endif

#endif
```

[-]CODE:/* File name: hello.c

```
* C source file.
*/
#include "hello.h"
#include

 void print_hello()
{
puts( "Hello, world!" );
}
```

[-]CODE:/* File name: main.cxx
* C++ source file.
*/
#include "hello.h"

int main()
{
print_hello();

return 0;
}
建立一个新的目录，然后把这三个文件拷贝到目录中，也把 Makefile 文件拷贝到目录中。之后，对 Makefile 的相关项目进行如下设置：

[-]CODE:PROGRAM := hello # 设置运行程序名
SRCDIRS := . # 源程序位于当前目录下
SRCEXTS := .c .cxx # 源程序文件有 .c 和 .cxx 两种类型
CFLAGS := -g # 为 C 目标程序包含 GDB 可用的调试信息
CXXFLAGS := -g # 为 C++ 目标程序包含 GDB 可用的调试信息
由于这个简单的程序只使用了 C 标准库的函数（puts），所以对于 CFLAGS 和 CXXFLAGS 没有过多的要求，LDFLAGS 和 CPPFLAGS 选项也无需设置。

经过上面的设置之后，执行 make 命令就可以编译程序了。如果没有错误出现的话，./hello 就可以运行程序了。

如果修改了源程序的话，可以看到只有和修改有关的源文件被编译。也可以再为程序添加新的源文件，只要它们的扩展名是已经在 Makefile 中设置过的，那么就没有必要修改 Makefile。

例二 GTK+ 版 Hello World 程序

这个 GTK+ 2.0 版的 Hello World 程序可以从下面的网址上得到：http://www.gtk.org/tutorial/c58.html#SEC-HELLOWORLD。当然，要编译 GTK+ 程序，还需要你的系统上已经安装好了 GTK+。

跟第一个例子一样，单独创建一个新的目录，把上面网页中提供的程序保存为 main.c 文件。对 Makefile 做如下设置：

[-]CODE:PROGRAM := hello # 设置运行程序名
SRCDIRS := . # 源程序位于当前目录下
SRCEXTS := .c # 源程序文件只有 .c 一种类型

CFLAGS := `pkg-config --cflags gtk+-2.0` # CFLAGS
LDFLAGS := `pkg-config --libs gtk+-2.0` # LDFLAGS
这是一个 C 程序，所以 CXXFLAGS 没有必要设置——即使被设置了也不会被使用。

编译和连接 GTK+ 库所需要的 CFLAGS 和 LDFLAGS 由 pkg-config 程序自动产生。

现在就可以运行 make 命令编译、./hello 执行这个 GTK+ 程序了。

参考资料：

- Multi-file projects and the GNU Make utility
Author: George Foot
http://www.elitecoders.de/mags/cscene/CS2/CS2-10.html

- GNU Make Manual
http://www.gnu.org/software/make/manual/

◆
◆
不错
但建议 LZ 改一下题目，这个文件不是在所有 Unix 下适用的，比如:=的写法 ◆
◆
FH 于 2006-3-4 21:58 发表
不错
但建议 LZ 改一下题目，这个文件不是在所有 Unix 下适用的，比如:=的写法
关于这一点文中已有说明：“这个 Makefile 可以在 GNU Make 和 GCC 编译器下正常工作。但是不能保证对于其它版本的 Make 和编译器也能正常工作。”

它应该可以适用于各个系统，包括 Linux、Unix、Windows、Mac，只要在系统上存在 GNU Make 和 GCC 编译器，或者和 GNU Make 和 GCC 兼容。因此，在上述前提下是可以跨平台的。

除了平台移植性之外，“通用”也可以指这个 Makefile 能快速适用于各种不同的应用程序。
◆
◆

QUOTE:原帖由 *whyglinux* 于 2006-3-4 23:07 发表

关于这一点文中已有说明：“这个 Makefile 可以在 GNU Make 和 GCC 编译器下正常工作。但是不能保证对于其它版本的 Make 和编译器也能正常工作。”

它应该可以适用于各个系统，包括 Linux、Unix、Windows、 ...

哦，那就改叫 GNU 通用好了
如果没有 GNU，要做出一个通用的来根本就是不可能的，各编译器的参数选项都千差万别。
◆
◆
如果有几层目录呢? ◆
◆
恩，学习。 ◆
◆
不错！谢谢！ ◆
◆
我就不明白了，那么多 L(U)nix 高人，怎么就没人愿意将生成 MAKE 文件的过程自动化啊？
很难吗？还是不屑于做？

当 linux 程序员还在工具怎么使用的时候，而 WINDOWS 程序员已经开始解决实际问题了。

海，不明白啊，不明白!!!! ◆
◆
这个东西好！ ◆
◆
谢一楼好文。

QUOTE:原帖由 *liuty2006* 于 2006-3-10 01:35 发表
当 linux 程序员还在工具怎么使用的时候，而 WINDOWS 程序员已经开始解决实际问题了。
这就是 Linux 的问题，Linux 目前还要靠开源免费来吸引眼球，而 M$已经赚了多少？
Linux/Unix 的高手高不可攀，对初级入门不肖一助，而初哥不得其门而入。 ◆
◆
◆

QUOTE:原帖由 *liuty2006* 于 2006-3-10 01:35 发表
我就不明白了，那么多 L(U)nix 高人，怎么就没人愿意将生成 MAKE 文件的过程自动化啊？
很难吗？还是不屑于做？

当 linux 程序员还在工具怎么使用的时候，而 WINDOWS 程序员已经开始解决实际问题了。

海，不明白啊 ...
GNU 工具不但早就将生成 Makefile 的过程自动化了，还把检测环境，连接库等等的工作自动化了。Windows 呢？
GNU 工具能够自己动化的不加改动在 N 多个平台下编译。Windows 下的 Projects 能吗？
记得某牛人都提倡 Windows 程序员学会 nmake 和 lc 的吧？

`Windows 程序员已经开始解决的实际问题'，是 UI 的问题吧？ ◆
◆
初学者可以接触一下 automake 可以自动生产 makefile ◆

◆
好东西呀，学习一下 ◆
◆
顶，学习 ◆
◆

QUOTE:原帖由 *liuty2006* 于 2006-3-10 01:35 发表
我就不明白了，那么多 L(U)nix 高人，怎么就没人愿意将生成 MAKE 文件的过程自动化啊？
很难吗？还是不屑于做？

当 linux 程序员还在工具怎么使用的时候，而 WINDOWS 程序员已经开始解决实际问题了。

海，不明白啊 ...
你没有用过
./configure
make
make install
吗?
这就是自动生成.

如果更自动一点,还有

aclocal
automake
autoconf

自己用 man 看看, 到底是什么用的. ◆
◆
不错，收藏了！ ◆
◆
还是 aclocal
automake
autoconf 还用， 跨平台，自动增加变异参数，连接库。便于发布。 ◆
◆
做个记号，好东西 ◆
◆
如果有很多目录,怎么办? ◆
◆
学习中 ◆
◆
◆
Mark 我想我很快就能用上这些东西了，要学的东西太多了！ ◆
◆

Makefile 在 Linux 下，是可以自动生成的，你只需要修改或添加一些针对性的参数就可以了。
◆
◆
多谢 LZ
收下先 ◆
◆

QUOTE:原帖由 *kingsz1* 于 2006-3-11 06:41 发表
谢一楼好文。

这就是 Linux 的问题，Linux 目前还要靠开源免费来吸引眼球，而 M$已经赚了多少？
Linux/Unix 的高手高不可攀，对初级入门不肖一助，而初哥不得其门而入。
深有感触 ◆
◆
好贴,多谢. ◆
◆
请教一下要在这里加入 include 目录如何加入?

我定义了一个变量:
INCCDIRS := -I/include/

然后再:
%.o : %.cpp
$(CXX) -c &(INCDIRS) $(CPPFLAGS) $(CXXFLAGS) $<
(假设我需要编译的是*.cpp 的文件)

可是不行... ◆
◆

QUOTE:原帖由 *converse* 于 2007-3-8 18:08 发表
请教一下要在这里加入 include 目录如何加入?

我定义了一个变量:
INCCDIRS := -I/include/

然后再:
%.o : %.cpp
$(CXX) -c &(INCDIRS) $(CPPFLAGS) $(CXXFLAGS) $<
(假设我需要编译的是*.cpp 的 ...
应该是：$(INCDIRS)吧？ ◆
◆

头文件的包含路径设置（如 -I/include/）也可集中放到 CFLAGS（C 程序）或者 CXXFLAGS（C++程序）变量的后面，这样就可省略引入自己定义的变量了。 ◆
◆

QUOTE:原帖由 *rrrrrrrr8* 于 2007-3-7 19:58 发表
好贴,多谢.
一年前的贴被你找出来了，赞。 ◆
◆

QUOTE:原帖由 *liuty2006* 于 2006-3-10 01:35 发表
我就不明白了，那么多 L(U)nix 高人，怎么就没人愿意将生成 MAKE 文件的过程自动化啊？很难吗？还是不屑于做？

当 linux 程序员还在工具怎么使用的时候，而 WINDOWS 程序员已经开始解决实际问题了。

海，不明白啊 ...
早就有了 autotool 系列。别抱怨了! ◆
◆
◆
请教 whyglinux 一个问题,生成的目标文件默认和源代码文件在一个目录里,我想自己设置目标文件的目录,在你的模版中加入了以下代码,但是不行:

```
OBJDIRS := obj
#以下修改了 OBJS 的写法
OBJS1 = $(foreach x,$(SRCEXTS), \
$(patsubst %$(x),%.o,$(filter %$(x),$(SOURCES))))
OBJS = $(foreach x,$(SRCDIRS), \
$(patsubst %$(x), $(OBJDIRS),$(filter %$(x),$(OBJS1))))
```

第一个OBJS1 是把*.$(SRCEXTS)替换成*.o 文件,可以成功,这个是你的模版中原有的了,第二个想把 SRCDIRS 替换成 OBJDIRS 但是不行,不知道为什么?

另外,你的这个文件要求生成的 PRAGRAMM 和目标文件在一个目录下,不知道怎么改变这个设置?

[*本帖最后由 converse 于 2007-3-15 15:10 编辑*] ◆
◆
我得 Makefile 里面有解决这个问题 ◆
◆
请注意一下下面这两行

[-]CODE:$(OBJ_DIR)%.o:$(SRC_DIR)%.c $(OBJ_DIR)%.d
$(CC) $(FLAG_COMPLE) $< -o $@
◆
◆
这里是带路经的，所以能够正确匹配

否则不能成功

（我测试的时候发现必须这样） ◆
◆

QUOTE:原帖由 *converse* 于 2007-3-15 15:08 发表
请教 whyglinux 一个问题,生成的目标文件默认和源代码文件在一个目录里,我想自己设置目标文件的目录,在你的模版中加入了以下代码,但是不行:

OBJDIRS := obj
#以下修改了 OBJS 的写法
OBJS1 = $(foreach x,$ ...
假设上面的变量 x 表示 abc 目录，则 %$(x) 就是 %abc，表示的特征是以 abc 结尾的字符串。有这样特征的目标文件存在吗？

其实有更简单的实现方法：OBJS = $(addprefix $(SRCDIRS), $(notdir $(OBJS1)))

要给生成的可执行文件加路径，在 PROGRAM 中直接设定即可；或者在连接目标文件的时候将 $(PROGRAM) 改为 (XX_PATH)(PROGRAM)。 ◆
◆

QUOTE:原帖由 *whyglinux* 于 2007-3-20 22:00 发表

假设上面的变量 x 表示 abc 目录，则 %$(x) 就是 %abc，表示的特征是以 abc 结尾的字符串。有这样特征的目标文件存在吗？

其实有更简单的实现方法：OBJS = $(addprefix $(SRCDIRS), $(notdir $(OBJS1)))
...
谢谢，我明天回去试试看，准备大规模的把这套模板用在工作中 ◆
◆

QUOTE:原帖由 *converse* 于 2007-3-20 23:36 发表

谢谢，我明天回去试试看，准备大规模的把这套模板用在工作中:em18::em18:
如果你的软件需要发布，有可能工作在不同的 Unix/Linux 平台下，那么这个 Makefile 不适合，因为它缺乏基本的探测系统环境的能力，而且许多功能也需要添加，此 Makefile 的跨平台性也没有经过测试。所以，这时，Autotools（或者叫 GNU Build System）是更加合适的选择。

我写这个 Makefile 的目的是为了得到一个一劳永逸的软件构建环境。也就是说，在为某一类程序（比如使用 GTK+ 库的程序）配置好 Makefile 之后，即使程序的源文件被改名、或者增加减少之后也不需要改动这个 Makefile，仍然可以用它来构建程序；而且，也可以用这个 Makefile 构建其它类似的程序（比如使用 GTK+ 库的程序。）。如果没有什么特殊的要求，通常你需要做的只是把它拷贝到程序的目录下、或者仅仅需要做一个符号连接。

无论在工作还是在学习中，我们都要经常自己编写或者实验他人的代码。这时，这个 Makefile 的优越性就可以体现出来了。Autotools 工具虽然功能强大，但是：一是你要学习它的使用，这对初学者有点困难；二是源程序文件变了，还需要相应地改动 Autotools 的配置文件，很难具备我上面提到的利便性。因此，除了正规的项目之外，一些实验性的代码我总是用我的这个 Makefile 来构建。

（借此机会回答了“已经有了 Autotools 工具了，还需要一个手工创建的 Makefile 吗？”类似的疑问，同时说明了这个通用 Makefile 的一般应用场景。希望有用。） ◆
◆
别让这个沉了，不好找，

仍然学习中 ◆
◆
这个比 automake 怎么样？ ◆
◆
哈哈，太好了！正是我要找的!!!谢谢楼主，我要认真研究研究！ ◆
◆
◆
C/C++ 通用 Makefile 新版本 0.2 发布了。

这个版本修正了上一版本中多目录连接的错误，在易用性上做了进一步的改进。

我已经在 SourceForge 上为此 Makefile 申请建立了一个项目：gcmakefile。新版本将在近期不断发布，欢迎下载测试并提出改进意见。

2007-3-25 12:19
下载次数: 86 gcmakefile-0.2.tar.gz (2.77 KB)
也可在此下载 gcmakefile 0.2： ◆
◆

QUOTE:原帖由 *whyglinux* 于 2007-3-24 20:19 发表

C/C++ 通用 Makefile 新版本 0.2 发布了。

这个版本修正了上一版本中多目录连接的错误，在易用性上做了进一步的改进。

我已经在 SourceForge 上为此 Makefile 申请建立了一个项目：[url=http://sourceforge ...
不错，谢谢楼主。 ◆
◆
很好 ◆
◆
已经下载，lz 太强了，向你学习!! 呵呵 ◆
◆
正在学习 makefile，看了楼主的，一下子明白了很多。顶!! ◆
◆
占个位 ◆
◆

QUOTE:原帖由 *whyglinux* 于 2007-3-25 12:19 发表
C/C++ 通用 Makefile 新版本 0.2 发布了。

这个版本修正了上一版本中多目录连接的错误，在易用性上做了进一步的改进。

我已经在 SourceForge 上为此 Makefile 申请建立了一个项目：[url=http://sourceforge ...
要积分才能下吗?! ◆
◆
我建了一个测试目录 test 把你的 Makefile（gcmakefile-0.3）放在 test 下。
在 test 目录下建了两个子目录 main 和 tool，main 目录下有 mytest.cpp mytest.h
tool 目录下有 tool.cpp tool.h 。
Makefile 的添加设置是
CPPFLAGS = -Wall -I main -I tool
SRCDIRS = tool main
PROGRAM = mytest
CFLAGS = -g
CXXFLAGS= -g
CXX = g++
执行 gmake 可以生成执行文件。
但只修改 mytest.h 的话，执行 gmake 不会重新编译程序。
如果把 mytest.cpp mytest.h too.cpp tool.h 都放在 test 目录下，
不设置 SRCDIRS ，只修改 mytest.h，会重新编译程序。
我是在 freebsd5.4 上测试的。gmake 为 GNU Make 3.80 ◆
◆

QUOTE:原帖由 *wxycyel* 于 2007-6-25 19:39 发表

我建了一个测试目录 test 把你的 Makefile（gcmakefile-0.3）放在 test 下。
在 test 目录下建了两个子目录 main 和 tool，main 目录下有 mytest.cpp mytest.h
tool 目录下有 tool.cpp tool.h 。
Makefile 的添加设置是
CPPF ...
谢谢 wxycyel 的测试和指正。问题在于源程序文件位于多目录时产生的依赖文件中路径的缺失。已经作了相应的修改，并添加了在线帮助功能，更新到 gcmakefile 0.4 版，欢迎下载测试与使用。 ◆
◆
鼓励一下。 ◆
◆
◆

QUOTE:原帖由 *liuty2006* 于 2006-3-10 01:35 发表
我就不明白了，那么多 L(U)nix 高人，怎么就没人愿意将生成 MAKE 文件的过程自动化啊？很难吗？还是不屑于做？

当 linux 程序员还在工具怎么使用的时候，而 WINDOWS 程序员已经开始解决实际问题了。

海，不明白 ...
首先不要用 win 的观点看待 UNIX。在很多 UNIX 爱好者心中，windows 是垃圾，往往总在做无用而非真正有用的事情，如同 windows 下程序员刚看待 UNIX 一个样子，他们可能会认为这个操作系统居然可以用？
观点不同，立场不同。
但是就你的问题，其实是有解决的，只是你不知道。 ◆
◆
看到老贴，支持一个。 ◆
◆
◆