Makefile.am

by Jian Lee

一般格式

文件类型	书写格式
可执行文件	bin_PROGRAMES = foo
	foo_SOURCES = xxxx.c
	foo_LDADD =
	foo_LDFLAGS =
	foo_DEPENDENCIES =
静态库	lib_LIBRARIES = libfoo.a
	foo_a_SOURCES =
	foo_a_LDADD =
	foo_a_LIBADD =
	foo_a_LDFALGS =
头文件	include_HEADERS = foo.h
数据文件	data_DATA = data1 data2

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。

全局变量

Makefile.am还提供了一些全局变量供所有的目标体使用：

变量	含义
INCLUDES	比如链接时所需要的头文件
LDADD	比如链接时所需要的库文件
LDFLAGS	比如链接时所需要的库文件选项标志
EXTRA_DIST	源程序和一些默认的文件将自动打入 .tar.gz 包，其他文件若要进入 .tar.gz 包可以使用这种方法，如配置文件，数据文件等。
SUBDIRS	处理本目录前要递归处理哪些子目录

automake 安装路径

automake设置了默认的安装路径：

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。
```

定义一个新的安装路径

比如test, 可定义

```
testdir = $(prefix)/test,
```

然后 test_DATA =test1 test2，则 test1，test2 会作为数据文件安装到 $(prefix)/test 目录下。

示例

我们首先需要在工程顶层目录下创建一个 Makefile.am 来指明包 含的子目录：

```
SUBDIRS=src/lib src/ModuleA/apple/shell src/ModuleA/apple/
CURRENTPATH=$(shell /bin/pwd)
INCLUDES=-I$(CURRENTPATH)/src/include -I$(CURRENTPATH)/src
export INCLUDES
```

由于每个源文件都会用到相同的头文件，所以我们在最顶层的Makefile.am中包含 了编译源文件时所用到的头文件，并导出。

我们将 lib 目录下的 swap.c 文件编译成 libswap.a 文件，被 apple/shell/apple.c 文件调用，那么lib目录下的 Makefile.am 如下所示：

```
noinst_LIBRARIES=libswap.a
libswap_a_SOURCES=swap.c
INCLUDES=-I$(top_srcdir)/src/includ
```

这里使用 noinst_LIBRARIES , 是因为如果只想编译，而不想安装到系统中，就 用 noinst_LIBRARIES 代替 bin_LIBRARIES ，对于可执行文件就用 noinst_PROGRAMS 代替 bin_PROGRAMS 。对于安装的情况，库将会安装到 $(prefix)/lib 目录下，可执行文件将会安装到 ${prefix}/bin 。如果想安装该 库，则 Makefile.am 示例如下：

```
bin_LIBRARIES=libswap.a
libswap_a_SOURCES=swap.c
INCLUDES=-I$(top_srcdir)/src/include
swapincludedir=$(includedir)/swap
swapinclude_HEADERS=$(top_srcdir)/src/include/swap.h
```

最后两行的意思是将 swap.h 安装到 ${prefix}/include/swap 目录下。

接下来，对于可执行文件类型的情况，我们将讨论如何写 Makefile.am . 对于编 译 apple/core目录下的文件，我们写成的Makefile.am如下所示：

```
noinst_PROGRAMS=test
test_SOURCES=test.c
test_LDADD=$(top_srcdir)/src/ModuleA/apple/shell/apple.o $
test_LDFLAGS=-D_GNU_SOURCE
DEFS+=-D_GNU_SOURCE
#LIBS=-lpthread
```

由于我们的 test.c 文件在链接时，需要 apple.o 和 libswap.a 文件，所以我 们需要在 test_LDADD 中包含这两个文件。对于 Linux 下的信号量/读写锁文件 进行编译，需要在编译选项中指明 -D_GNU_SOURCE 。所以在 test_LDFLAGS 中指 明。而 test_LDFLAGS 只是链接时的选项，编译时同样需要指明该选项，所以需 要 DEFS 来指明编译选项，由于 DEFS 已经有初始值，所以这里用 += 的形式指 明。从这里可以看出，Makefile.am 中的语法与 Makefile 的语法一致，也可以 采用条件表达式。如果你的程序还包含其他的库，除了用 AC_CHECK_LIB 宏来指 明外，还可以用LIBS来指明。

如果你只想编译某一个文件，那么 Makefile.am 如何写呢？这个文件也很简单， 写法跟可执行文件的差不多，如下例所示：

```
noinst_PROGRAMS=apple
apple_SOURCES=apple.c
DEFS+=-D_GNU_SOURCE
```

我们这里只是欺骗 automake ，假装要生成apple文件，让它为我们生成依赖关系 和执行命令。所以当你运行完 automake 命令后，然后修改 apple/shell/ 下的 Makefile.in 文件，直接将LINK语句删除，即：

```
clean-noinstPROGRAMS:
    -test -z "$(noinst_PROGRAMS)" || rm -f $(noinst_PROGRA

apple$(EXEEXT): $(apple_OBJECTS) $(apple_DEPENDENCIES)
    @rm -f apple$(EXEEXT)

#$(LINK) $(apple_LDFLAGS) $(apple_OBJECTS) $(apple_LDADD)
```