链滴

一个 mini Autotool 项目实践

作者：ReyRen

原文链接：https://ld246.com/article/1567155773944

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

****

用户提供的输入文件

在这个项目中我们需要用户仅仅提供两个文件. 剩下的全部都交给Autotools去处理和生成:

- "Makefile.am" 是automake的输入文件
- "configure.in" 是autoconf的输入文件

我喜欢把"Makefile.am"想象成一个项目构建要求的最基本的规范: 需要编译什么, 以及它安装在哪? 可能是Automake的最大优势 - 描述尽可能简单, 但最终产品是一个带有一系列方便make target的"Mkefile".

"configure.in"是宏调用和shell代码片段的模板, "autoconf"使用它来生成"configure"脚本. "autocon"将"configure.in"的内容复制到"configure"中, 并且扩展输入中出现的宏. 其他文本是逐字复制的.

首先来看看用户提供的"Makfile.am":

```
## Makefile.am -- Process this file with automake to produce Makefile.in
    bin_PROGRAMS = foonly
    foonly_SOURCES = main.c foo.c foo.h nly.c scanner.l parser.y
    foonly_LDADD = @LEXLIB@
```

这个"Makefile.am"指定我们当运行make install时在"bin"目录中编译和安装一个名为"foonly"的程序. 用于编译"foonly"的源文件是C源文件'main.c', 'foo.c', 'nly.c'和'foo.h', 'scanner.l'中的lex程序和 'parer.y'中的yacc语法. 这也引出了Automake非常好的一个方面: 因为lex和yacc都从它们的输入文件生中间C程序, Automake知道如何构建这样的中间文件并将它们链接到最终的可执行文件中. 最后, 我们须记住链接一个合适的lex库.

接下来是"configure.in":

```
dnl Process this file with autoconf to produce a configure script.
    AC_INIT(main.c)
```

```
AM_INIT_AUTOMAKE(foonly, 1.0)
AC_PROG_CC
AM_PROG_LEX
AC_PROG_YACC
AC_OUTPUT(Makefile)
```

这个"configure.in"调用一些强制的Autoconf和Automake初始化宏, 然后调用AC_PROG系列的一些utoconf宏来找到合适的C编译器, lex和yacc程序. 最后, AC_OUTPUT宏用于使生成的"configure"脚输出"Makefile" - 但是来自哪里呢? 它是从"Makefile.in"处理的, "Automake"根据你的"Makefile.am而生成. 也就是说"Makefile.in"是输入给configure的, 然后生成"Makefile".

```
graph TD;
    makefile.am --> AUTOMAKE;
    AUTOMAKE --> makefile.in;
    configure.in --> AUTOCONF;
    AUTOCONF --> configure;
    makefile.in --> configure;
    configure --> Makefile;
```

生成输出文件

在之后的文章中会介绍到从输入文件到输出文件需要什么命令去执行. 在这里我们先说"configure"的成:

```
$ aclocal
$ autoconf
```

因为"configure.in"所包含的宏调用好多是Autoconf自身不认识的(因为前面提到了, configure.in会用一些Autoconf和Automake强制初始化宏)--"AM_INIT_AUTOMAKE"就是一个典型的例子. 所以很必要为Autoconf收集所有的宏定义, 让它可以在生成"configure"的时候用. 这个就是通过aclocal这个序完成的. 名字的由来是由于它生成了"aclocal.m4". 如果你看生成出来的"aclocal.m4", 就会发现"AM_NIT_AUTOMAKE"的定义.

在运行autoconf之后, 你就会发现"configure"生成了出来, 但是先运行aclocal是很重要的, 因为"autoake"依赖于"configure.in"和"aclocal.m4". 关系是不是有些凌乱了, 的确有点恶心. 其实各个文件的出顺序是这样的:

autoscan生成configure.ac --> 调整configure.ac中的一些参数 --> autoconf先生成一波autom4teconfigure --> 然后编辑Makefile.am --> 再次编辑configure.ac, 比如引入AUTOMAKE, OUTPUT的宏 --> aclocal --> automake --add-missing 这一步是需要有根据之前配置好的configure.ac, 因里面写了再哪些字目录中有Makefile.am --> 最后autoconf生成最终版configure --> ./configure成出Makefile.

这里我写了一篇autoconf&automake实战性指导, 具体可以帮忙理清楚它们之间的关系, 穿插在这里因为我在学习这里的时候也是凌乱了. 于是在这里阅读链接文章比较舒服.

我们在看看Automake:

```
$ automake --add-missing
    automake: configure.in: installing ./install-sh
    automake: configure.in: installing ./mkinstalldirs
    automake: configure.in: installing ./missing
    automake: Makefile.am: installing ./INSTALL
    automake: Makefile.am: required file ./NEWS not found
```

```
automake: Makefile.am: required file ./README not found
automake: Makefile.am: installing ./COPYING
automake: Makefile.am: required file ./AUTHORS not found
automake: Makefile.am: required file ./ChangeLog not found
```

"--add-missing"会将一些样板文件从Automake的安装中拷贝到当前目录. "COPYING"文件包含GP规则, 并且变化是不频繁的, 所以可以不需要用户介入的生成. 一系列的脚本文件也会生成, 它们都是为给生成的"Makefile"使用的. 尤其是给"install" target. 但是可以看到一些Requred的文件仍然是丢失的:

NEWS

用户可见的包更改记录. 格式不严格, 但对最新版本的更改应显示在文件的顶部.

README

用户首先要查看包的目的, 以及可能的特殊安装说明.

AUTHORS

维护者的邮箱

ChangeLog

这个格式是比较严格的, 这也是相当重要的一个文件. 记录了这个包的变化.

我们先来欺骗一下Automake:

```
touch NEWS README AUTHORS ChangeLog
automake --add-missing
```

到目前为止, 目录的内容看起来相当完整, 可以让你联想到你之前安装的GNU项目的顶级目录情况:

```
AUTHORS    INSTALL     NEWS        install-sh    mkinstalldirs
COPYING    Makefile.am  README      configure     missing
ChangeLog  Makefile.in  aclocal.m4  configure.in
```

接下来就可以将这些所有文件打包到一个tar文件中, 然后给其他人在他们自己的系统上进行安装. 在由utomake生成的"Makefile.in"中有个make的target会很容易的帮助生成不同的发行版. 关于这块儿的西, 在以后会单独谈到. 当用户拿到后, 只需要打开tar包, 然后:

```
$ ./configure
    creating cache ./config.cache
    checking for a BSD compatible install... /usr/bin/install -c
    checking whether build environment is sane... yes
    checking whether make sets ${MAKE}... yes
    checking for working aclocal... found
    checking for working autoconf... found
    checking for working automake... found
    checking for working autoheader... found
    checking for working makeinfo... found
    checking for gcc... gcc
    checking whether the C compiler (gcc  ) works... yes
    checking whether the C compiler (gcc  ) is a cross-compiler... no
    checking whether we are using GNU C... yes
    checking whether gcc accepts -g... yes
    checking how to run the C preprocessor... gcc -E
    checking for flex... flex
    checking for flex... (cached) flex
    checking for yywrap in -lfl... yes
```

```
    checking lex output file root... lex.yy
    checking whether yytext is a pointer... yes
    checking for bison... bison -y
    updating cache ./config.cache
    creating ./config.status
    creating Makefile

$ make all
    gcc -DPACKAGE=\"foonly\" -DVERSION=\"1.0\" -DYYTEXT_POINTER=1  -I. -I. \

-g -O2 -c main.c
    gcc -DPACKAGE=\"foonly\" -DVERSION=\"1.0\" -DYYTEXT_POINTER=1  -I. -I. \

-g -O2 -c foo.c
    flex   scanner.l && mv lex.yy.c scanner.c
    gcc -DPACKAGE=\"foonly\" -DVERSION=\"1.0\" -DYYTEXT_POINTER=1  -I. -I. \

-g -O2 -c scanner.c
    bison -y   parser.y && mv y.tab.c parser.c
    if test -f y.tab.h; then \

if cmp -s y.tab.h parser.h; then rm -f y.tab.h; \

else mv y.tab.h parser.h; fi; \
    else :; fi
    gcc -DPACKAGE=\"foonly\" -DVERSION=\"1.0\" -DYYTEXT_POINTER=1  -I. -I. \

-g -O2 -c parser.c
    gcc  -g -O2  -o foonly  main.o foo.o scanner.o parser.o -lfl
```

关于输入文件的维护

如果在项目中更改了GNU Autotool中的任何一个输入文件, 那么必须重新生成由机器生成的这些文才能使得改动生效.

当然我们也可以通过一次运行一个所需的工具来重新生成这些文件. 但是, 正如我们上面所看到的, 计依赖关系可能很困难 - 比如说特定的更改是否需要运行aclocal? 特定更改是否需要运行autoconf? 针于这个问题有两种解决方案:

- **第一种解决方案是使用autoreconf命令** 此工具通过以正确的顺序重新运行所有必需的工具来重新成所有派生文件. 它有点像一个暴力的解决方案, 但它工作得很好, 特别是如果你不想容纳其他维护者, 定期维护会很麻烦.
- **另一种选择是Automake的"维护者模式"** 通过从"configure.in"中调用AM_MAINTAINER_MOD宏, "automake"将在"configure"中激活--enable-maintainer-mode选项. 关于这个功能的细节, 再后会说到.

打包生成的文件

关于如何处理生成的文件的争论是在相关的邮件列表上争论的热火朝天的. 归纳总结下基本有两种观点:

- **一个论点是生成的文件不应该包含在包中, 而应该只包含源代码的"首选形式"** 根据这个定义, "confi ure"是一个派生文件, 就像一个目标文件, 它不应该包含在包中. 因此, 用户应该在构建软件包之前使用 NU Autotools来自我引导. 我相信这种纯粹的方法有一些优点, 因为它不鼓励打包衍生文件的做法.

● **另一个论点是提供这些文件的优点远远超过了上面提到的对良好软件工程实践的违背** 通过包括生成文件, 用户可以方便地无需关注和使用的所有不同版本的工具保持同步. 对于Autoconf尤其如此, 因为"onfigure"脚本通常由使用本地修改版本的autoconf和本地安装的宏的维护者生成. 如果用户重新生成"onfigure", 则结果可能与预期的不同. 当然, 这是一种糟糕的做法, 但它恰好反映了现实.

我相信答案是当包将要分发给广泛的用户时, 在包中包含生成的文件. 对于内部包, 前一个论点可能更意义, 因为这些工具也可以在版本控制下进行.

记录和changelog

在"ChangeLog"中记录更改时, 一个条目是由一个人完成的. 逻辑更改组合在一起, 而逻辑上不同的更改即"更改集")由单个空行分隔. 以下是Automake自己的"ChangeLog"的示例:

```
2019-09-02  Yuan Ren  <reyren179@gmail.com>

 * automake.in (finish_languages): Only generate suffix rule
         when not doing dependency tracking.

 * m4/init.m4 (AM_INIT_AUTOMAKE): Use AM_MISSING_INSTALL_SH.
         * m4/missing.m4 (AM_MISSING_INSTALL_SH): New macro.

 * depend2.am: Use @SOURCE@, @OBJ@, @LTOBJ@, @OBJOBJ@,
         and @BASE@.  Always use -o.
```

关于"ChangeLog"条目的另一个要点是它们应该言简意骇. 条目没有必要详细解释为什么要进行更改. NU编码标准提供了一整套保存"ChangeLog"的指南. 尽管可以使用任何文本编辑器创建ChangeLog目, 但Emacs提供了一种主模式来帮助编写它们.

OK, 这一篇信息量不小. 接下来会深入这里涉及的没个文件进行细节描述.
