<https://www.sohu.com/a/191735643_505857>

在Linux下经常要安装部署一些软件包或者工具，拿到安装包之后一看，简单，configure,make, make install即可搞定。

有时候我就在想，这个configure,make ,make install是什么意思呢，configure是测试存在的特性，然后make开始编译，make install生成相应的可执行文件。但是一个工具只是记住了其中的拼写部分或是基本的概念，但是对于原理知之甚少，也是需要补补了。

几个构建编译隐藏的命令

要先说这个编译安装过程，使用命令aclocal会生成m4文件，aclocal本质上是一个perl脚本。先提提m4, m4是一种宏处理器，它是 POSIX 标准的一部分。为什么叫m4呢，全称是macro,m后面有4个字母，据说是这样的，哈哈。摘录一段对于m4的描述:从图灵的角度来看 m4，输入流与输出流可以衔接起来构成一条无限延伸的纸带，m4 是这条纸带的读写头，所以 m4 是一种图灵机。m4 的计算能力与任何一种编程语言等同，区别只体现在编程效率以及所编写的程序的运行效率方面。

然后是autoconf，是生成configure文件的，configure是一个脚本，它能设置源程序来适应各种不同的操作系统平台，并且根据不同的系统来产生合适的Makefile，从而可以使你的源代码能在不同的操作系统平台上被编译出来。

最后是automake用来生成Makefile.in文件

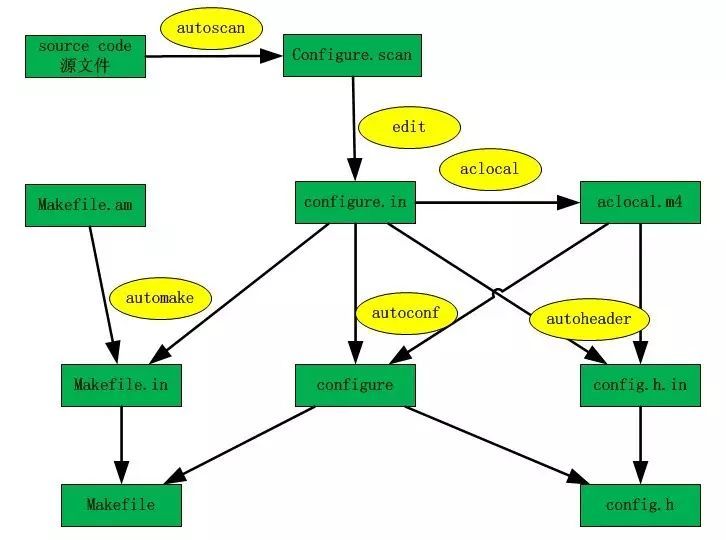
简单总结一下，这个编译过程涉及几个命令工具，大体的功能点如下。

aclocal # 产生 aclocal.m4

autoconf # 根据 configure.in 生成configure

automake --add-missing # 根据 Makefile.am生成Makefile.in

网上找到一张总结的很牛的图，很全面。



构建过程环境准备

我们写个简单的Hello world来了解下整个过程吧。

我写了一段非常简单的c程序，就凑合着编译着用吧。文件为main.c

#include <stdio.h>

int main(int argc, const char \*argv[])

{

printf("Hello world ,a new testn");

return 0;

}

可以看出，程序运行后的输出就是Hello world,a new test

我们看看构建GNU程序中如何按照规范来模拟这个过程

我们创建一个文件configure.ac，里面是一些宏，是接下俩的autoconf来处理的需要的，然后交给automake来处理，最终完成这个检查。

AC\_INIT([helloworld],[0.1],[xxx@xxx.com])

AM\_INIT\_AUTOMAKE

AC\_PROG\_CC

AC\_CONFIG\_FILES([Makefile])

AC\_OUTPUT

比如AC\_INIT([helloworld],[0.1],[xxx@xxx.com])的含义是autoconf生成包的名字，版本（这个可以自己定义），反馈邮箱，

AM\_INIT\_AUTOMAKE是检查automake尝试Makefile时的工具，AC\_PROG\_CC是编译器检测，AC\_CONFIG\_FILES是automake构建出类似.in的文件。

然后就是Makefile的文件，我们设定名字为Makefile.am，这部分的内容和上面的配置是密切相关的。

[root@oel64 tmp]# cat Makefile.am

AUTOMAKE\_OPTIONS=foreign

bin\_PROGRAMS = helloworld

helloworld\_SOURCES = main.c

automake提供了3种软件等级：foreign、gnu和gnits。默认等级是gnu。此处AUTOMAKE\_OPTIONS使用的是foreign，表示只检测必要的文件。

bin\_PROGRAMS定义了要产生的执行文件名，这里我们定义为helloworld

file\_SOURCES定义file这个执行程序的依赖文件，其中“file\_SOURCES”中的前部分“file”要改写成可执行文件名，即与bin\_PROGRAMS定义的名称一致，此处就是helloworld了。如果有多个可执行文件，那就要定义相应的file\_SOURCES。

构建过程实践

到目前为止，我们创建了3个文件 main.c,configure.ac,Makefile.am

[root@oel64 c]# ll

-rwxr-xr-x. 1 root root 108 Sep 13 12:13 configure.ac

-rw-r--r--. 1 root root 105 Sep 13 12:13 main.c

-rw-r--r--. 1 root root 79 Sep 13 12:13 Makefile.am

首先使用aclocal来得到m4文件。这里生成了2个文件，一个是aclocal.m4,另外一个是cache文件autom4te.cache

[root@oel64 c]# aclocal

[root@oel64 c]# ll

total 56

-rw-r--r--. 1 root root 34611 Sep 13 12:14 aclocal.m4

drwxr-xr-x. 2 root root 4096 Sep 13 12:14 autom4te.cache

-rwxr-xr-x. 1 root root 108 Sep 13 12:13 configure.ac

-rw-r--r--. 1 root root 105 Sep 13 12:13 main.c

-rw-r--r--. 1 root root 79 Sep 13 12:13 Makefile.am

然后使用autoconf得到configure文件

[root@oel64 c]# autoconf

[root@oel64 c]# ll

-rw-r--r--. 1 root root 34611 Sep 13 12:14 aclocal.m4

drwxr-xr-x. 2 root root 4096 Sep 13 12:14 autom4te.cache

-rwxr-xr-x. 1 root root 135288 Sep 13 12:14 configure

-rwxr-xr-x. 1 root root 108 Sep 13 12:13 configure.ac

-rw-r--r--. 1 root root 105 Sep 13 12:13 main.c

-rw-r--r--. 1 root root 79 Sep 13 12:13 Makefile.am

然后使用automake来构建模块

[root@oel64 c]# automake --add-missing

configure.ac:2: installing `./install-sh'

configure.ac:2: installing `./missing'

Makefile.am: installing `./depcomp'

整个过程完成之后，就是我们平常执行的操作了。

执行configure的结果如下：

[root@oel64 c]# ./configure

checking for a BSD-compatible install... /usr/bin/install -c

checking whether build environment is sane... yes

checking for a thread-safe mkdir -p... /bin/mkdir -p

checking for gawk... gawk

checking whether make sets $(MAKE)... yes

checking for gcc... gcc

checking for C compiler default output file name... a.out

checking whether the C compiler works... yes

checking whether we are cross compiling... no

checking for suffix of executables...

checking for suffix of object files... o

checking whether we are using the GNU C compiler... yes

checking whether gcc accepts -g... yes

checking for gcc option to accept ISO C89... none needed

checking for style of include used by make... GNU

checking dependency style of gcc... gcc3

configure: creating ./config.status

config.status: creating Makefile

config.status: executing depfiles commands

[root@oel64 c]#

然后是make，这个过程你可以清晰的看到gcc开始编译。

[root@oel64 c]# make

gcc -DPACKAGE\_NAME="helloworld" -DPACKAGE\_TARNAME="helloworld" -DPACKAGE\_VERSION="0.1" -DPACKAGE\_STRING="helloworld 0.1" -DPACKAGE\_BUGREPORT="xxx@xxx.com" -DPACKAGE="helloworld" -DVERSION="0.1" -I. -g -O2 -MT main.o -MD -MP -MF .deps/main.Tpo -c -o main.o main.c

mv -f .deps/main.Tpo .deps/main.Po

gcc -g -O2 -o helloworld main.o

最后是make install,有了可执行的程序文件。

[root@oel64 c]# make install

make[1]: Entering directory `/root/c'

test -z "/usr/local/bin" || /bin/mkdir -p "/usr/local/bin"

/usr/bin/install -c helloworld '/usr/local/bin'

make[1]: Nothing to be done for `install-data-am'.

make[1]: Leaving directory `/root/c'

比如编译后的main.o,如果使用strings来查看内容就是执行后的结果。

[root@oel64 c]# strings main.o

Hello world ,a new test

如果查看可执行程序helloworld的内容，里面是有相应的堆栈的。

[root@oel64 c]# strings helloworld

/lib64/ld-linux-x86-64.so.2

\_\_gmon\_start\_\_

libc.so.6

puts

\_\_libc\_start\_main

GLIBC\_2.2.5

fff.

手工执行一下 。

[root@oel64 c]# ./helloworld

Hello world ,a new test[返回搜狐，查看更多](https://www.sohu.com/?strategyid=00001)

声明：该文观点仅代表作者本人，搜狐号系信息发布平台，搜狐仅提供信息存储空间服务。