行文件中。这个库如果被﷽﷽﷽﷽所需生成的可执行文件中。这个库函数源码编译及库函数

在讲链接的文章里已经比较细致地说了多个源文件是通过什么方式来链接的，可以通过符号表和重定位两个关键词来简单回忆一下有关链接的过程。了解链接最主要的好处是能够理解定义在不同源文件中的变量和函数是如何通过符号表来完成共享的。不过本文将会从最基本的源码安装过程来补充剩下的一些内容。

在linux下如果是通过源码安装软件，一般都会经历下面几个步骤：

* ./congigure
* make clean
* make
* make install

首先是执行configure检测程序，该程序会检测linux系统以及相关软件属性，之后通过configure程序生成接下来程序安装所需的信息，也就是通常我们所说的Makefile文件。

那么什么是Makefile文件呢？通常在写一些小代码，只包含一个源文件时，只需要通过gcc \*.c即可完成编译工作，即使是包含几个头文件这么做依旧不会太麻烦。但是当工程比较大时，再每个源文件都写到命令行下就很不现实了，这时候需要有一个独立于源代码之外的文件，用来指导整个编译过程，这就是Makefile文件的作用。可以说Makefile工程编译的说明书，配合make就可以完成自动化编译工作。

通常从github上克隆的工程在执行上述configure程序之后都会生成一个Makefile文件，一般如果是直接用vim去看这个文件一般都是一头雾水，越大的工程越是如此。如果是小一些的工程，并且对各个模块功能比较熟悉又能了解链接过程的话，即使不能完全看懂Makefile文件，看个八九不离十也是可以的。

在一个包含多个源文件的工程中，编译的过程就是将各个源文件通过编译器转成二进制的目标文件（以.o结尾），之后通过链接器将各个目标文件链接成完整的可执行文件。这个过程如果单独去做的话，则需要通过编译器逐个源文件去生成目标文件，然后再去逐一去链接，而这里要说到的Makefile就是为了解决这方面问题的。

不过之前说到的用configure是使用别人的东西，这里想要说的是如果自己写了个大一点的工程怎么去写Makefile文件。要自己写Makefile文件，首先得知道Makefile的一些基本语法规则。

Makefile规则：

目标（target）：需要的条件

命令（前面必须是Tab键开头）

1. 目标可以为一个或多个，可以是目标文件或可执行文件
2. 需要的条件就是生成目标所需要的文件
3. 命令就是生成目标所需要执行的脚本（通常为命令行指令）

简而言之就是目标的生成依赖于所需的条件，生成的规则则由命令来描述。

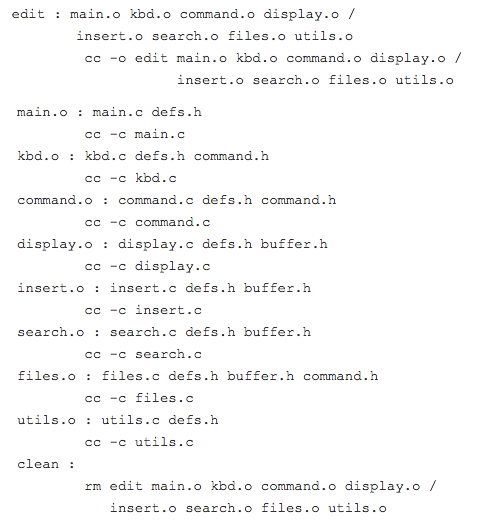
假定一个工程有3个头文件，8个源文件的话，可以用以下方式来完成。 

图1：Makefile例子

上面的例子说明了编译后生成了8个目标文件，之后再由这8个目标文件生成了可执行文件edit。在我们键入make命令后就可以执行子都怪你编译了。那么，输入make之后是怎么执行的呢？

1. 首先make会在当前目录下搜索Makefile文件
2. 如果找到则会找到文件中第一个目标文件，上例中是edit，并将其作为最终的目标文件
3. 如果edit不存在或者后面依赖的.o文件修改时间比较新，则执行其后面的命令来生成edit文件。
4. 如果edit所依赖的.o文件也不存在，则根据其依赖的文件和命令去生成。这里输入的内容一定是.c和.h文件，如果Makefile中所需的源文件和头文件也不存在，就编译失败。

从上面的描述可以看出，整个make按照Makefile去执行编译的过程非常树的遍历过程，叶子节点就是最基本的源文件和头文件，中间节点就是各目标文件，最顶端的根节点是最终的可执行文件。执行make的过程就是按照Makefile从叶子节点到根节点逐层编译，满足所有依赖关系的过程。

不过，上面的Makefile文件写起来似乎依旧比较麻烦，并且在重复的工作中只要有一个地方写少了就会导致整个编译的失败，这就需要我们使用变量来化简。

在上例中我们可以看到，最终可执行文件edit的条件部分和命令部分重复了两次，我们可以使用objects变量来指代这部分内容。

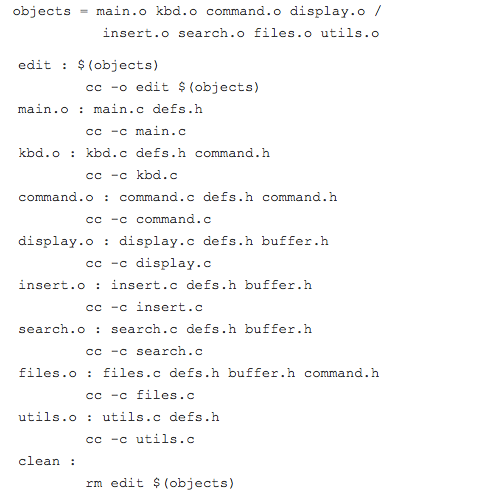


图2：Makefile中使用变量

上面的Makefile已经简化了很多，不过还可以更简单一些，make提供了自动推导文件及文件后面依赖关系的命令，于是可以继续简化，将推导的工作交给make。

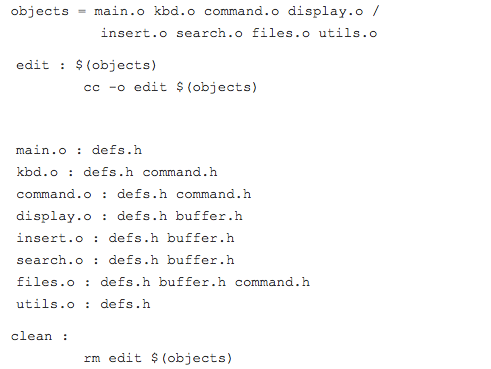


图3：make自动推导

简单介绍了make之后顺带提一下几种常见库函数形式。

静态库：静态库的扩展名以.a结尾，每一个.a结尾的静态库文件可以理解为含有多个.o目标文件的集合。如果我们想把写得代码打包为静态库的话，只需要使用ar工具就可以做到，这里就不具体展开了。静态库最大的特点是在链接会将静态库中被引用的模块完整地拷贝到被所需生成的可执行文件中，这里所做的事情实际上完整的数据拷贝而非地址引用，这就导致使用静态库第一个问题就是程序会比较大，如果多个程序使用同一个静态库的话更是如此。还有一个问题就是如果静态库更新了，则依赖于该库的多个程序都需要重新编译。不过相比于其他库方式，静态库因为一次性包含了所有所需的内容，所以每个程序没有了其他依赖性，可以直接执行。

动态库：动态库的扩展名以.so结尾。动态库相比于静态库最大的区别是动态库只是在链接时加入引用的位置，只有当使用到了库时才会去取库的内容。也正因如此，使用动态库的程序大小会比较小，而且在库进行更新时不需要重新编译每个程序。除了上面提到的这些内容，对于动态库，我们可以将最常用的动态库加载到内存中用来加快访问速度，我们可以将打算加载到内存中的动态库写入/etc/ls.so.conf中，之后执行ldconf即可完成动态库加载进内存的工作。