 不管是自己进行项目开发还是安装应用软件，我们都经常要用到Make或Make install。利用Make工具，我们可以将大型的开发项目分解成为多个更易于管理的模块，对于一个包括几百个源文件的应用程序，使用Make命令和 Makefile工具就可以简洁明快地理顺各个源文件之间纷繁复杂的相互关系。 而且如此多的源文件，如果每次都要键入gcc命令进行编译的话，那对程序员来说简直就是一场灾难。而Make工具则可自动完成编译工作，并且可以只对程序员在上次编译后修改过的部分进行编译。因此，有效的利用Make命令和Makefile文件工具可以大大提高项目开发的效率。同时掌握Make 和Makefile文件之后，您也不会再面对着Linux下的应用软件手足无措了。 但令人遗憾的是，在许多讲述Linux应用的书籍上都没有详细介绍这个功能强大但又非常复杂的编译工具。在这里我就向大家详细介绍一下Make及其描述文件 Makefile。 Makefile文件 Make工具最主要也是最基本的功能就是通过Makefile文件来描述源程序之间的相互关系并自动维护编译工作。而Makefile 文件需要按照某种语法进行编写，文件中需要说明如何编译各个源文件并连接生成可执行文件，并要求定义源文件之间的依赖关系。Makefile 文件是许多编译器--包括 Windows NT 下的编译器--维护编译信息的常用方法，只是在集成开发环境中，用户通过友好的界面修改Makefile文件而已。在 UNIX 系统中，习惯使用Makefile作为makfile 文件。如果要使用其他文件作为 Makefile，则可利用类

似下面的 Make 命令选项指定Makefile文件： 1.$ Make -f Makefile.debug 例如，一个名为prog的程序由三个C源文件filea.c、fileb.c和filec.c以及库文件LS编译生成，这三个文件还分别包含自己的头文件a.h 、b.h和c.h。通常情况下，C编译器将会输出三个目标文件filea.o、fileb.o和filec.o。假设filea.c和fileb.c都要声明用到一个名为defs的文件，但filec.c不用。即在filea.c和fileb.c里都有这样的声明： 1.#include "defs" 那么下面的文档就描述了这些文件之间的相互联系: 1.#It is a example for describing Makefile 2.prog : filea.o fileb.o filec.o 3.cc filea.o fileb.o filec.o -LS -o prog 4.filea.o : filea.c a.h defs 5.cc -c filea.c 6.fileb.o : fileb.c b.h defs 7.cc -c fileb.c 8.filec.o : filec.c c.h 9.cc -c filec.c 这个描述文档就是一个简单的Makefile文件。从上面的例子注意到，第一个字符为 # 的行为注释行。第一个非注释行指定prog由三个目标文件filea.o、fileb.o和filec.o链接生成。第三行描述了如何从prog所依赖的文件建立可执行文件。接下来的4、6、8行分别指定三个目标文件，以及它们所依赖的.c和.h文件以及defs文件。而5、7、9行则指定了如何从目标所依赖的文件建立目标。当filea.c或a.h文件在编译之后又被修改，则 Make 工具可自动重新编译filea.o，如果在前后两次编译之间，filea.C 和a.h 均没有被修改，而且 test.o 还存在的话，就没有必要重新编译。 这种依赖关系在多源文件的程序编译中尤其重要。通过这种依赖关系的定义，Make 工具可避免许多不必要的编译工作。当然，利用 Shell脚本也可以达到自动编译的效果，但是，Shell 脚本将全部编译任何源文件，包括哪些不必要重新编译的源文件，而 Make 工具则可根据目标上一次编译的时间和目标所依赖的源文件的更新时间而自动判断应当

编译哪个源文件。 Makefile文件作为一种描述文档一般需要包含以下内容: ◆ 宏定义 ◆ 源文件之间的相互依赖关系 ◆ 可执行的命令 Makefile中允许使用简单的宏指代源文件及其相关编译信息，在Linux中也称宏为变量。在引用宏时只需在变量前加$符号，但值得注意的是，如果变量名的长度 超过一个字符，在引用时就必须加圆括号（）。 下面都是有效的宏引用：1.$(CFLAGS) 2.$2 3.$Z 4.$(Z) 其中最后两个引用是完全一致的。需要注意的是一些宏的预定义变量，在Unix系统中，$*、$@、$?和$ 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com