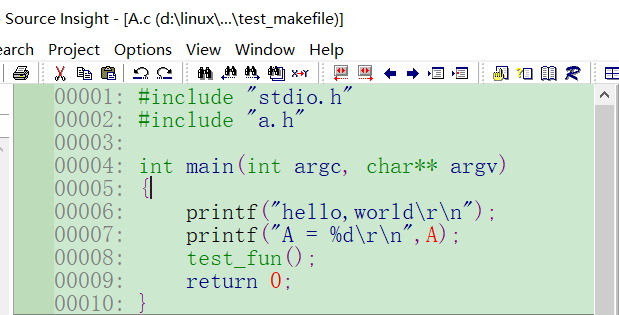
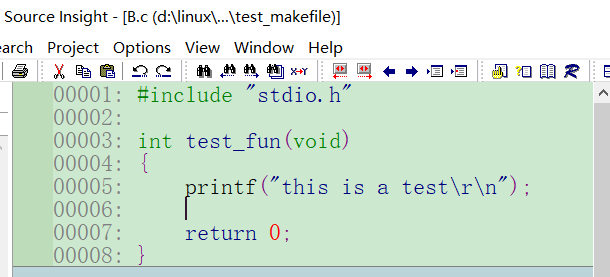
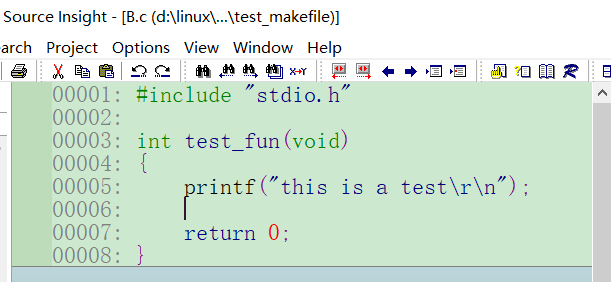
首先简单理解makefile中依赖关系，某个文件依赖于a.h，如果a.h文件被改动，所以依赖于a.h的文件会被全部重新编译，反之如果不动则不会。举例如下

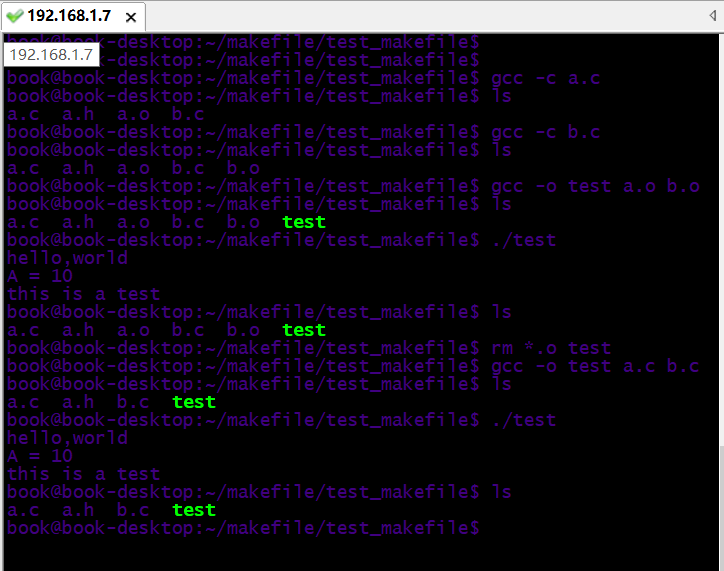
有 a.c b.c a.h三个文件







ubuntu中操作如图



gcc –c a.c/gcc a.c/gcc -o

gcc –c只编译C文件，成功输出文件a.o

gcc –o是将目标文件连接成可执行二进制文件

-c -o是不能同时使用的

gcc –o test a.c b.c

对于a.c b.c均会执行预处理，编译，汇编的操作

最后用-o链接成可执行文件

写makefile

核心：规则

目标：依赖1 依赖2 依赖3……

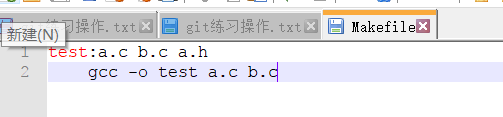
TAB命令

命令执行的条件：

i 依赖 比 目标 的时间新

ii 没有“目标”这个文件

举例



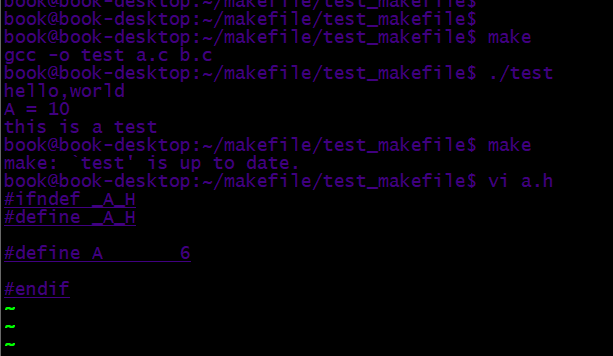
我们有makefile这个文件后，一执行make操作就回去自动读取当前路径下的makefile，然后根据规则看是否执行。如上图所示，test是目标文件，没有的话就回去执行gcc的操作。

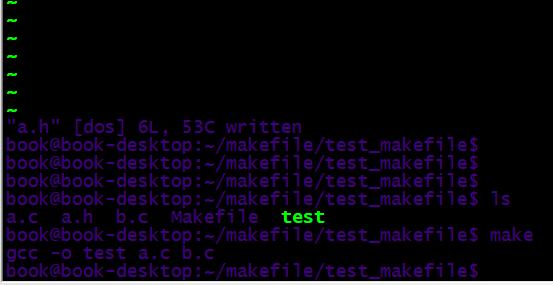
命令执行的条件

i 没有test

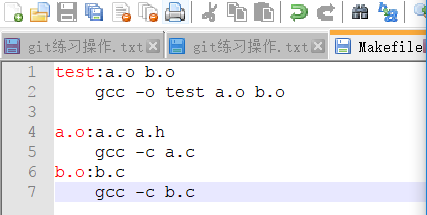
ii 依赖比目标文件新

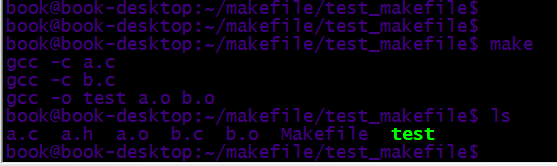
如下图所示，分别是没有目标文件以及依赖更新的情况下执行make操作的结果：





这是第一个makefile，比较low，我们可以继续优化，如下所示：





首先用a.o b.o去生成test，a.o不存在继续往下找，找到为止。

所以执行的顺序为

gcc -c a.c

gcc -c b.c

gcc -o test a.o b.o

核心：

目标：依赖1 依赖2

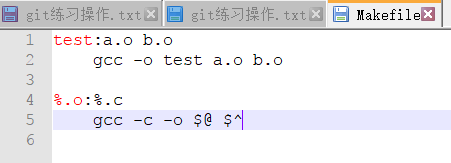
生成目标命令

依赖1：依赖1.1 依赖1.2

生成依赖1命令

这个makefile还是有缺陷，如果我有很多依赖怎么办？

通配符的引入

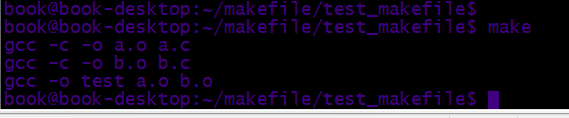


$@:目标文件

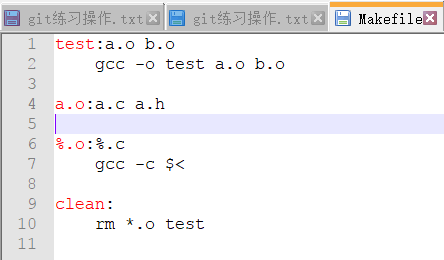
$<:第一个依赖

$^:所有的依赖

%：通配符，所有的

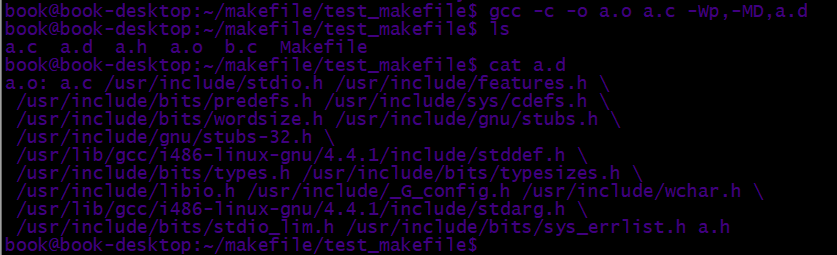


这个makefile更智能一点，但是有个地方不对，a.o依赖于a.h的，没有写进去，怎么办呢，这个时候你去修改a.h是不会有任何效果的。还需要改进一下。

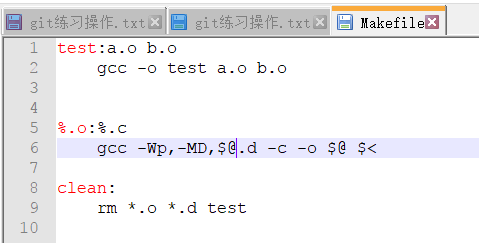


添加了一个进去之后就可以了。但是文件多了我们不可能每次都自己添加依赖吧，所以我们要想办法生成一个依赖文件

如，查看生成a.o的依赖文件 gcc –c –o a.o a.c -Wp,-Mp,a.d

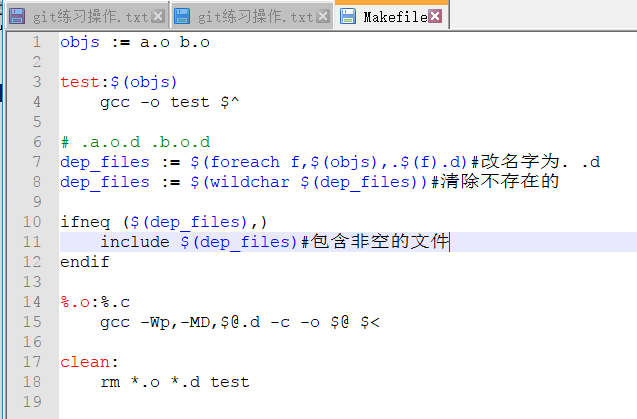


所以我们可以用命令自动生成依赖文件，修改makefile如下



之后会生成.o.d文件，这就是依赖了，我们之后要把依赖加入到include里面去，开始使用makefile函数了，go on

foreach wildchar两个函数



看多层次的makefile每个的最底层是这样的

obj-y += file1.o

obj-y += file2.o

它的上一层是这样的

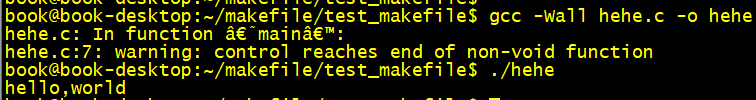
obj-y += dir/

插讲一点编译知识

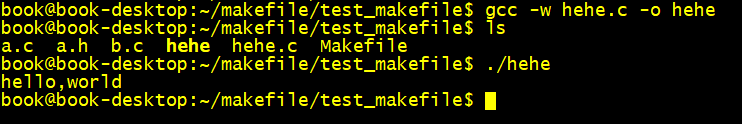
编译的时候指定目录 -I 头文件目录 -L库文件目录

-Wall选项 列出所有警告

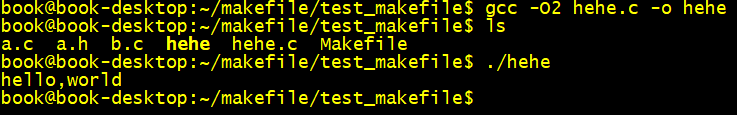
gcc –Wall test.c –o test



-w选项 关闭编译时的警告，编译后不显示任何warning



-O –O1 –O2 –O3不同的优化程度，具体详细百度



-g是加上调试信息gdb用，不然不用

make –C是指进入某个特定的目录再执行make

**make命令的使用**

**make[option] [target]**

**-C dir**：在读入makefile之前，把路径切换到dir下。如果同时使用几个‘-C’选项，则eachis interpreted relative to the previous one。

**-d**：在正常处理后打印调试信息。

**-e**：设置环境变量的优先权高于makefile文件变量的优先权。

**-f file**：将file设置为makefile文件。

**-i**：忽略在执行重建文件命令时产生的所有错误。

**-Idir**：指定搜寻makefile文件的路径。如果同时使用几个‘-I’选项，则按照次序搜寻这些路径。

**-k**：在出现错误后，尽可能的继续执行。也就是说当一个目标创建失败后，所有依靠它的目标文件将不能重建，而这些目标的其它依赖则可继续处理。

**-n**：打印要执行的命令，但却不执行它们。

**-ofile**：即使文件file比它的依赖旧，也不重建该文件。

**-p**：打印数据库，其中的数据来自读入makefile文件的结果；打印之后执行。

make –qp                           打印数据库后不执行。

make –p –f/dev/null       打印预定义的规则和变量的数据库。

**-q**：不打印也不执行命令。如果所有目标都已经更新到最新，make的退出状态是0；如果一部分需要更新，退出状态是1；如果make遇到错误，退出状态是2。

**-r**：禁止使用预定义的隐含规则，同时也清除了缺省的后缀列表和后缀规则。注意缺省的变量仍然有效。

**-R**：禁止使用内建的规则变量。‘-R’自动使‘-r’生效。

**-s**：不回显执行的命令。

**-S**：使‘-k’失效。除非在递归调用make时，通过变量MAKEFLAGS从上层make继承‘-k’，或环境中设置了选项‘-k’，否则没有必要使用该选项。

**-t**：标记文件已经更新到最新，但实际却没有更新它们。

**-w**：打印执行makefile文件时涉及的所有工作目录。

**-Wfile**：Pretendthat the file has been just modified。在和‘-n’一起使用时，将表明更改该文件会发生什么。如果没有和‘-n’一起使用，那么它和在运行make之前对该文件使用touch命令的结果几乎一样，但使用该选项make只是在想象中更改该文件的时间戳。

:= += ？= =三者的区别



如何寻找makefile中第一个target

target : 依赖

行，从现在开始我们来写通用makefile