参考：https://my.oschina.net/u/1413984/blog/199029s

Makefile有三个非常有用的变量。分别是$@，$^，$<代表的意义分别是：

假设我们有下面这样的一个程序，源代码如下： 

|  |
| --- |
| /\* main.c \*/  #include "mytool1.h"  #include "mytool2.h"  int main(int argc，char \*\*argv)  {  mytool1\_print("hello")；  mytool2\_print("hello")；  }  /\* mytool1.h \*/  #ifndef \_MYTOOL\_1\_H  #define \_MYTOOL\_1\_H  void mytool1\_print(char \*print\_str)；  #endif  /\* mytool1.c \*/  #include "mytool1.h"  void mytool1\_print(char \*print\_str)  {  printf("This is mytool1 print %s "，print\_str)；  }  /\* mytool2.h \*/  #ifndef \_MYTOOL\_2\_H  #define \_MYTOOL\_2\_H  void mytool2\_print(char \*print\_str)；  #endif  /\* mytool2.c \*/  #include "mytool2.h"  void mytool2\_print(char \*print\_str)  {  printf("This is mytool2 print %s "，print\_str)；  } |

当然由于这个程序很短，我们可以这样来编译:  

|  |
| --- |
| gcc -c main.c  gcc -c mytool1.c  gcc -c mytool2.c  gcc -o main main.o mytool1.o mytool2.o |

 这样的话我们也可以产生main程序，而且也不是很麻烦。但是如果我们考虑一下如果有一天我们修改了其中的一个文件(比如说mytool1.c)那么我们难道还要重新输入上面的命令?也许你会说，这个很容易解决啊，我写一个 SHELL脚本，让它帮我去完成不就可以了。是的对于这个程序来说，是可以起到作用的。但是当我们把事情想的更复杂一点，如果我们的程序有几百个源程序的时候，难道也要编译器重新一个一个的去编译?   
  
为此，聪明的程序员们想出了一个很好的工具来做这件事情，这就是make。我们只要执行以下make，就可以把上面的问题解决掉。在我们执行make之前，我们要先编写一个非常重要的文件。--Makefile。对于上面的那个程序来说，可能的一个Makefile的文件是：   
  
# 这是上面那个程序的Makefile文件: 

|  |
| --- |
| main：main.o mytool1.o mytool2.o  gcc -o main main.o mytool1.o mytool2.o  main.o：main.c mytool1.h mytool2.h  gcc -c main.c  mytool1.o：mytool1.c mytool1.h  gcc -c mytool1.c  mytool2.o：mytool2.c mytool2.h  gcc -c mytool2.c |

有了这个Makefile文件，不论我们什么时候修改了源程序当中的什么文件，我们只要执行make命令，我们的编译器都只会去编译和我们修改的文件有关的文件，其它的文件它连理都不想去理的。   
  
下面我们学习Makefile是如何编写的。   
  
在Makefile中也#开始的行都是注释行.Makefile中最重要的是描述文件的依赖关系的说明。一般的格式是：   
  
target：components   
  
TAB rule   
  
第一行表示的是依赖关系。第二行是规则。   
  
比如说我们上面的那个Makefile文件的第二行。   
  
main：main.o mytool1.o mytool2.o   
  
表示我们的目标(target)main的依赖对象(components)是main.o mytool1.omytool2.o 当倚赖的对象在目标修改后修改的话，就要去执行规则一行所指定的命令。就象我们的上面那个Makefile第三行所说的一样要执行 gcc-o main main.o mytool1.o mytool2.o 注意规则一行中的TAB表示那里是一个TAB键   
  
Makefile有三个非常有用的变量。分别是$@，$^，$<代表的意义分别是：   
  
$@--目标文件，$^--所有的依赖文件，$<--第一个依赖文件。   
  
如果我们使用上面三个变量，那么我们可以简化我们的Makefile文件为：   
  
# 这是简化后的Makefile   
  
main：main.o mytool1.o mytool2.o   
  
gcc -o $@ $^   
  
main.o：main.c mytool1.h mytool2.h   
  
gcc -c $<   
  
mytool1.o：mytool1.c mytool1.h   
  
gcc -c $<   
  
mytool2.o：mytool2.c mytool2.h   
  
gcc -c $<   
  
经过简化后,我们的Makefile是简单了一点，不过人们有时候还想简单一点。这里我们学习一个Makefile的缺省规则   
  
.c.o：   
  
gcc -c $<   
  
这个规则表示所有的 .o文件都是依赖与相应的.c文件的。例如mytool.o依赖于mytool.c这样Makefile还可以变为：   
  
# 这是再一次简化后的Makefile   
  
main：main.o mytool1.o mytool2.o   
  
gcc -o $@ $^   
  
.c.o：   
  
gcc -c $<   
  
好了，我们的Makefile 也差不多了，如果想知道更多的关于Makefile的规则，可以查看相应的文档。