<https://blog.csdn.net/runfarther/article/details/50036115>

我们先看三段C++程序：

一、line1的源码

line1.h

**[cpp]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. #ifndef \_LINE\_1\_H
2. #define \_LINE\_1\_H
3. **void** line1\_print(**const** **char** \*strMsg);
4. #endif

line1.cpp

**[cpp]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. #include "line1.h"
2. #include <stdio.h>
3. **void** line1\_print(**const** **char** \*strMsg)
4. {
5. printf("This is line1 print %s.\r\n",strMsg);
6. }

二、line2的源码

line2.h

**[cpp]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. #ifndef \_LINE\_2\_H
2. #define \_LINE\_2\_H
3. **void** line2\_print(**const** **char** \*strMsg);
4. #endif

line2.cpp

**[cpp]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. #include "line2.h"
2. #include <stdio.h>
3. **void** line2\_print(**const** **char** \*strMsg)
4. {
5. printf("This is line2 print %s.\r\n",strMsg);
6. }

三、main的源码

main.cpp

**[cpp]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. #include "line1.h"
2. #include "line2.h"
4. **int** main(**int** argc,**char** \*\*argv)
5. {
6. line1\_print("hello runfarther");
7. line2\_print("hello runfarther");
9. **return** 0;
10. }

对上面的代码，在不用Makefile时，我们可以直接用命令行来编译，得到我们的可执行程序main.out：

**[plain]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. g++ -c main.c
2. g++ -c line1.c
3. g++ -c line2.c
4. g++ -o main.out main.o line1.o line2.o

为了编译工作更加方便，我们通常会编写Makefile来完成编译，我们先看一个用于编译和链接上面代码的例子：

**[plain]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. main.out:main.o line1.o line2.o
2. g++ -o main.out main.o line1.o line2.o
3. main.o:main.c line1.h line2.h
4. g++ -c main.c
5. line1.o:line1.c line1.h
6. g++ -c line1.c
7. line2.o:line2.c line2.h
8. g++ -c line2.c

从例子可以看出，Makefile一般的格式是：

**[html]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. target:components
2. rule

第一行表示的是依赖关系，第二行是规则，特别要注意，rule这行必须是TAB键开头。

比如说我们上面的那个Makefile文件的前面二行：

**[plain]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. main.out:main.o line1.o line2.o
2. g++ -o main.out main.o line1.o line2.o

表示我们的目标(target)main.out的依赖对象(components)是main.o line1.o line2.o，当依赖的对象在被修改的话，就要去执行规则一行所指定的命令g++ -o main.out main.o line1.o line2.o。注意规则这行是以一个TAB键开头。  
  
接下来我来介绍下Makefile中的四个有用的特殊符号意义和使用，他们分别是@、$@、$^、$<

一、@

这个符串通常用在“规则”行中，表示不显示命令本身，而只显示它的结果，例如Makefile中的内容为：

**[plain]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. DIR\_OBJ=./obj
2. CMD\_MKOBJDIR=if [ -d ${DIR\_OBJ} ]; then exit 0; else mkdir ${DIR\_OBJ}; fi
4. mkobjdir:
5. @${CMD\_MKOBJDIR}

命令行执行如下：

**[html]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. make mkobjdir

此时不会显示在命令行不会显示出if [ -d ${DIR\_OBJ} ]; then exit 0; else mkdir ${DIR\_OBJ}; fi，但如果规则行的TAB后没有以@开头，则会显示，不信可以试试。

二、$@、$^、$<

这三个分别表示：

$@          --代表目标文件(target)

$^            --代表所有的依赖文件(components)

$<           --代表第一个依赖文件(components中最左边的那个)。

好了，知道了他们的意义后，如果使用上面三个变量，那么简化的Makefile文件为：

**[plain]** [view plain](https://blog.csdn.net/runfarther/article/details/50036115) [copy](https://blog.csdn.net/runfarther/article/details/50036115)

1. main.out:main.o line1.o line2.o
2. g++ -o $@ $^
3. main.o:main.c line1.h line2.h
4. g++ -c $<
5. line1.o:line1.c line1.h
6. g++ -c $<
7. line2.o:line2.c line2.h
8. g++ -c $<

当然，还可以进一步简化，有兴趣可以再去研究下，这里就不展示了。