# [C编译: makefile基础](http://www.cnblogs.com/vamei/archive/2013/04/29/3051062.html)

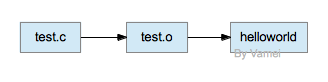
作者：Vamei 出处：http://www.cnblogs.com/vamei 欢迎转载，也请保留这段声明。谢谢！

在编译一个大型项目的时候，往往有很多目标文件、库文件、头文件以及最终的可执行文件。不同的文件之间存在依赖关系(dependency)。比如当我们使用下面命令编译时:

$gcc -c -o test.o test.c

$gcc -o helloworld test.o

可执行文件helloworld依赖于test.o进行编译的，而test.o依赖于test.c。



 依赖关系

在我们编译一个大型项目时，我们往往要很多次的调用编译器，来根据依赖关系，逐步编译整个项目。这样的方式是自下而上的，即先编译下游文件，再编译上游文件。

UNIX系统下的make工具用于自动记录和处理文件之间的依赖关系。我们不用输入大量的"gcc"命令，而只需调用make就可以完成整个编译过程。所有的依赖关系都记录在makefile文本文件中。我们只需要make helloworld，make会根据依赖关系，自上而下的找到编译该文件所需的所有依赖关系，最后再自下而上的编译。

(make有多个版本，本文将基于GNU make。make会自动搜索当前目录下的makefile, Makefile或者GNUmakefile)



依赖

### 基本概念

我们使用一个示例C语言文件:

[复制代码](javascript:void(0);)

#include <stdio.h>

/\*

\* By Vamei

\* test.c for makefile demo

\*/

int main()

{

printf("Hello world!\n");

return 0;

}

[复制代码](javascript:void(0);)

下面是一个简单的makefile

[复制代码](javascript:void(0);)

# helloworld is a binary file

helloworld: test.o  
　　echo "good"

　　gcc -o helloworld test.o

test.o: test.c

　　gcc -c -o test.o test.c

[复制代码](javascript:void(0);)

观察上面的makefile

* #号起始的行是注释行
* target: prerequisite为依赖关系，即目标文件(target)依赖于前提文件(prerequisite)。可以有多个前提文件，用空格分开。
* 依赖关系后面的<Tab>缩进行是实现依赖关系进行的操作，即正常的UNIX命令。一个依赖关系可以附属有多个操作。

用直白的话说，就是:

* 想要helloworld吗？那你必须有test.o，并执行附属的操作。
* 如果没有test.o，那你必须搜索其他依赖关系，并创建test.o。

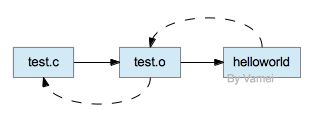
我们执行

$make helloworld

来创建helloworld。

make是一个[递归](http://www.cnblogs.com/vamei/archive/2013/03/30/2989930.html)创建的过程:

* Base Case 1: 如果当前依赖关系中没有说明前提文件，那么直接执行操作。
* Base Case 2: 如果当前依赖关系说明了目标文件，而目标文件所需的前提文件已经存在，而且前提文件与上次make时没有发生改变(根据最近写入时间判断)，也直接执行该依赖关系的操作。
* 如果当前目标文件依赖关系所需的前提文件不存在，或者前提文件发生改变，那么以前提文件为新的目标文件，寻找依赖关系，创建目标文件。



虚线: 依赖关系检索

上面是make的核心功能。有了上面的功能，我们可以记录项目中所有的依赖关系和相关操作，并使用make进行编译。下面的内容都是在此核心内容上的拓展。

### 宏

make中可以使用宏(MACRO)。宏类似于文本类型的变量。比如下面的CC:

[复制代码](javascript:void(0);)

CC = gcc  
  
# helloworld is a binary file

helloworld: test.o  
　　echo "good"

　　$(CC) -o helloworld test.o

test.o: test.c

　　$(CC) -c -o test.o test.c

[复制代码](javascript:void(0);)

我们用CC来代表"gcc"。在makefile中，使用$(CC)的方式来调用宏的值。make会在运行时，使用宏的值(gcc)来替代$(CC)。

shell的环境变量可以直接作为宏调用。如果同一个自定义的宏同时也有同名环境环境变量，make将优先使用自定义宏。

(可以使用$make -e helloworld来优先使用环境变量)

类似于C语言的宏，makefile中的宏可以方便的管理一些固定出现的文本，并方便替换操作。比如我们未来使用ifort编译器时，只需要更改宏定义为:

CC = ifort

就可以了

### 内部宏

make中有内部定义的宏，可以直接使用。$@中包含有当前依赖关系的目标文件名，而$^包含当前目标的前提文件:

[复制代码](javascript:void(0);)

CC = gcc

# helloworld is a binary file

helloworld: test.o

　　echo $@

　　$(CC) -o $@ $^

test.o: test.c

　　$(CC) -c -o $@ $^

[复制代码](javascript:void(0);)

内部宏       功能

$\*          当前依赖关系中的目标文件名，不包括后缀。

$\*          当前依赖关系中，发生改变的前提文件

$$          字符"$"

如果目标或者前提文件是一个完整路径，我们可以附加D和F来提取文件夹部分和文件名部分，比如$(@F)表示目标文件的文件名部分。

### 后缀依赖

在makefile中使用

.SUFFIXES: .c .o

来说明.c和.o是后缀。

我们可以使用后缀依赖的方式，比如:

[复制代码](javascript:void(0);)

CC = gcc

.SUFFIXES: .c .o

.c.o:

$(CC) -c -o $@ $^

#--------------------------

# helloworld is a binary file

helloworld: test.o

echo $@

$(CC) -o $@ $^

test.o: test.c

[复制代码](javascript:void(0);)

我们定义.c和.o为后缀。并有后缀依赖关系.c.o:。前者为前提，后者为目标。(注意，与一般的依赖关系顺序不同)

上面的test.o和test.c有依赖关系，但没有操作。make会发现该依赖关系符合.c.o的后缀依赖，并执行该后缀依赖后面的操作。

如果项目很大型的时候，后缀依赖非常有用。符合后缀依赖的文件往往有类似的操作，我们可以将这些操作用后缀依赖表示，而避免重复输入。

### 其他

makefile的续行符为\

makefile中经常会定义下面依赖关系:

all:

如果make后没有跟随文件名，那么将执行该依赖关系。

clean:

常用于清理历史文件。

比如:

[复制代码](javascript:void(0);)

CC = gcc

.SUFFIXES: .c .o

.c.o:

$(CC) -c -o $@ $^

#--------------------------

all: helloworld

@echo "ALL"

# helloworld is a binary file

helloworld: test.o

@echo $@

$(CC) -o $@ $^

test.o: test.c

clean:

-rm helloworld \*.o

[复制代码](javascript:void(0);)

注意: echo前面的@和rm前面的-。@后的命令将不显示命令本身。-后面的命令将忽略错误(比如删除不存在的文件)。

### 总结

make的核心功能是根据依赖关系来实现编译管理。

make的其他功能是让用户可以更加便捷的写出makefile。

参考

http://oreilly.com/linux/excerpts/9780596100292/gnu-make-utility.html