How to use Makefile

What is GCC?

- 1、GCC 的意思也只是 GNU Compiler Collection ，可以用来编译c语言、Ada 语言、C++ 语言、Java 语言、Objective C 语言、Pascal 语言、COBOL语言，以及支持函数式编程和逻辑编程的 Mercury 语言等等。

2、一个示例程序test.c:

```
#include <stdio.h>
int main(void)
{
   printf("Hello World!\n");
   return 0;
}
```

使用gcc编译，一步到位的命令为： gcc test.c -o test, 就可以生成可执行文件test，输入命令./test，就可以看到“Hello world！ “。

3、其实编译分为4步执行，即预处理(也称预编译，Preprocessing)、编译(Compilation)、汇编 (Assembly)和连接(Linking)。

GCC的编译的四个阶段

- 1、预处理：gcc -E test.c -o test.i
- 2、编译为汇编代码(Compilation)：gcc -S test.i -o test.s或直接从.c文件编译，gcc -S test.c -o test.s
- 3、汇编(Assembly)：gcc -c test.s -o test.o或gcc -c test.c -o test.o
- 4、连接(Linking)：gcc test.o -o test

一些GCC规则

- 1、多个程序文件的编译：gcc test1.c test2.c -o test
- 2、检错：gcc -pedantic illcode.c -o illcode
- 3、警告：gcc -Wall illcode.c -o illcode
- 4、警告处停止编译：gcc -Werror test.c -o test

GDB的使用（1）

- 一个调试示例
- 源程序：test.c

```
1 #include <stdio.h>
2
3 int func(int n)
4 {
5         int sum=0,i;
6         for(i=0; i<n; i++)
7         {
8                 sum+=i;
9         }
10        return sum;
11 }
```

GDB的使用（2）

```
14 main()
15 {
16          int i;
17          long result = 0;
18          for(i=1; i<=100; i++)
19          {
20                  result += i;
21          }
22
23        printf("result[1-100] = %d /n", result );
24        printf("result[1-250] = %d /n", func(250) );
25 }
```

GDB的使用（3）

- 1、在之前介绍的gcc的使用中，可以生成执行文件：gcc -g test.c -o test
- 2、在终端中键入命令：gdb test会进入如下的界面：

GDB的使用（4）

- 3、显示源程序，键入：l
- 4、设置断点：b 行数/函数名
- 5、显示断点信息：info b
- 6、执行：r
- 7、执行断点后一句：n
- 8、执行完成断点后的部分：c
- 9、显示变量的值：p 变量名
- 10、推出gdb：q

What is Makefile?

- 当一个工程规模达到一定的大小时，工程中会出现很多的源文件，而源文件会根据功能、类型、模块分别存放在不同的文件目录。makefile制定的规则可以帮助你决定哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译。
- makefile的一个好处就是“自动化编译”，只需要一个make命令就可以实现整个工程的自动编译。Make是一个解释makefile中指令的命令工具。
- 接下来就让我们来看看makefile如何工作的吧～～

1、一个简单的例子（样例1）

- 1、一个main.c源文件:
- #include<stdio.h>
- int main()
- {

  printf("hello,world!\n");

  return 0;
- }
- 2、其对应的Makefile文件（此文件取名为Makefile或者makefile，这样make命令就能自动识别出）：
- result:main.c

  cc -o result main.c

如何使用Makefile

- 如样例1中所示，只需要进入到该文件夹的目录下。

- 第一步键入make，就可以执行makefile自动完成编译，生成可执行文件result。

- 第二步键入./result，就会显示结果：
- “hello ,world!”

解释下Makefile的基本语句

- 1、回顾前面提到的Makefile：
- result:main.c
- 	cc -o result main.c
- 2、形如：
- target: prerequisite

  command
- 其中target是生成的目标文件，比如上面的最终可执行文件result。
- prerequisite是生成target需要依赖的文件，包括.c ，.o ，.h文件。
- command是命令，是用来执行的语句（任意的shell命令），比如上面的cc -o result main.c，（注意，command这行开头必须Tab键空格，这样才会被识别为需要执行的命令）。

一个稍微复杂的例子（样例2）

- 这是个验证用户是不是“LiMing”，如果是”LiMing”就可以登入，并且选择要看的课程。该工程中包含以下的源文件：
- 1、2个头文件：hello.h, header.h
- 2、4个.c文件：call.c, function.c ,verify.c, main.c
- 3、hello.h中声明了四个函数：command(), display(), getName(), getPassword,
- 这四个函数在function.c中实现。
- 4、header.h中声明了两个函数call(), verify(),分别在call.c以及function.c中实现。
- 5、main.c，会去调用以上相关的函数。

样例2对应的Makefile

- result:call.o function.o main.o verify.o

  cc -o result -Wall main.o call.o function.o verify.o
- main.o:main.c header.h hello.h

  cc -c -Wall main.c
- call.o:call.c header.h

  cc -c -Wall call.c
- function.o:function.c hello.h

  cc -c -Wall function.c
- verify.o:verify.c header.h

  cc -c -Wall verify.c
- clean:

  rm result verify.o function.o\

  call.o main.o

该Makefile内容的解释（1）

- 1、在这个makefile中：
- 目标文件（target）包含：执行文件result和中间目标文件（*.o）。
- 依赖文件（prerequisites）就是冒号后面的那些 .c 文件和 .h文件以及.o文件。每一个 .o 文件都有一组依赖文件，而这些 .o 文件又是执行文件 result 的依赖文件。依赖关系的实质上就是说明了目标文件是由哪些文件生成的。

Makefile中内容的解释（2）

- 2、
- verify.o:verify.c header.h

  cc -c -Wall verify.c
- 注意到这里的target是verify.o，.o文件是编译生成的中间代码。编译生成.o文件执行命令是：cc -c …。
- 这里的prerequisite包括verify.c 和header.h，是因为verify.c 中的函数是在header.h中声明的，所以这两个文件都是依赖文件。
- 注意到命令中加了 -Wall，这是让编译器将那些warning显示出来，当然你可以选择去掉-Wall，这样就不会显示warning，不过还是建议加上这个命令，避免一些可能隐藏的错误。

- 3、result:call.o function.o main.o verify.o

  cc -o result -Wall main.o call.o function.o verify.o

- 这里result的生成依赖于main.o，call.o,function.o,verify.o

Makefile中内容的解释（3）

- 5、看这三行代表性代码：
- clean:

```
rm result verify.o function.o\
        call.o main.o
```

- 这三行代码跟我们前面讲的形式有些差别。首先是符号“\”，这是用来换行的，这样比较便于makefile的易读。
- 注意到这里的prerequisite内容即“：”后面的内容为空，这说明clean不依赖于任何文件，所以makefile执行时不会去主动执行这段命令，即键入make时不会执行这段命令。注意到这段命令是帮助我们删除生成的执行文件result以及*.o文件（这样我们就可以重新生成新的.o文件以及执行文件了），只需要在命令行键入“make clean”就会去执行这段命令，完成删除功能。

make是如何工作的（1）

- 在默认的方式下，也就是我们只输入make命令。那么：
- 1、make会在当前目录下找名字叫“Makefile”或“makefile”的文件。
- 2、如果找到，它会找文件中的第一个目标文件（target），在上面的例子中，他会找到“result”这个文件，并把这个文件作为最终的目标文件。
- 3、如果result文件不存在，或是result所依赖的后面的 .o 文件的文件修改时间要比result这个文件新，那么，他就会执行后面所定义的命令来生成result这个文件。
- 4、如果result所依赖的.o文件也不存在，那么make会在当前文件中找目标为.o文件的依赖性，如果找到则再根据那一个规则生成.o文件。（这有点像一个堆栈的过程）
- 5、当然，你的C文件和H文件是存在的啦，于是make会生成 .o 文件，然后再用 .o 文件生成make的终极任务，也就是执行文件result了

make是如何工作的（2）

- 1、make会一层又一层地去找文件的依赖关系，直到最终编译出第一个目标文件。在找寻的过程中，如果出现错误，比如最后被依赖的文件找不到，那么make就会直接退出，并报错，而对于所定义的命令的错误，或是编译不成功，make根本不理。make只管文件的依赖性，即，如果在我找了依赖关系之后，冒号后面的文件还是不在，那么对不起，我就不工作啦。

- 2、于是在我们编程中，如果这个工程已被编译过了，当我们修改了其中一个源文件，比如call.c，那么根据我们的依赖性，我们的目标call.o会被重编译（也就是在这个依性关系后面所定义的命令），于是.call.o的文件也是最新的啦，于是call.o的文件修改时间要比result要新，所以result也会被重新链接了（详见result目标文件后定义的命令）。

Makefile中变量的使用

- 1、在上面的变量中，让我们先看看result的规则：
- result:call.o function.o main.o verify.o

  cc -o result -Wall main.o call.o function.o verify.o
- 2、我们可以看到*.o的字符串被重复了两次，如果下次还要有.o 文件要加入，那么需要在三个地方增加，除了这边两个外还有clean那个地方。要是少加了一个地方就会出现一些错误。所以如果将这些.o文件定义为一个变量的内容，那么只要在这个变量赋值的地方增加或者删除相应的.o文件就行了。
- 3、我们在makefile的开头加上一句:objects = main.o call.o function.o verify.o，然后将之前对应的三处改为$(objects)

改良版的makefile

- objects = call.o function.o main.o verify.o
- result:$(objects)

  cc -o result -Wall $(objects)
- main.o:main.c header.h hello.h

  cc -c -Wall main.c
- call.o:call.c header.h

  cc -c -Wall call.c
- function.o:function.c hello.h

  cc -c -Wall function.c
- verify.o:verify.c header.h

  cc -c -Wall verify.c
- Clean:

  rm result $(objects)
- 如果要加入或者删去.o文件，只需要改objects就行了～～

获取更多Makefile的知识

- 1、联系我：xingealpha@gmail.com
- 2、使用google