简单的makefile编写

闲扯

以前写代码的时候都是用codeblocks的现在用文本编辑器了还有点不适应。尤其是编译运行的时候都要输入一大行的命令烦死了，我开始也是了解到make的但是我一直以为他是一个特别高大上的东西就一直都没有学会用，今天我尝试用了一下，嘿嘿那么简单我还有点不适应了呢。好了闲扯就这些了。。。。虽然我只会一点简单的。。。

正文

Makefile 介绍

> make命令执行时，需要一个 Makefile 文件，以告诉make命令需要怎么样的去编译和链接程序。 只要我们的Makefile写得够好，所有的这一切，我们只用一个make命令就可以完成，make命令会自动智能地根据当前的文件修改的情况来确定哪些文件需要重编译，从而自己编译所需要的文件和链接目标程序。

Makefile的规则

在讲述这个Makefile之前，还是让我们先来粗略地看一看Makefile的规则。

```
target ... : prerequisites ...
command
...
...
```

target也就是一个目标文件，可以是Object File，也可以是执行文件。还可以是一个标签（Label).

prerequisites就是，要生成那个target所需要的文件或是目标。

command也就是make需要执行的命令。（任意的Shell命令）

这是一个文件的依赖关系，也就是说，target这一个或多个的目标文件依赖于prerequisites中的文件，其生成规则定义在command中。说白一点就是说，prerequisites中如果有一个以上的文件比target文件要新的话，command所定义的命令就会被执行。这就是Makefile的规则。也就是Makefile中最核心的内容。

说到底，Makefile的东西就是这样一点，好像我的这篇文档也该结束了.

```
all: 1.cpp
    g++ 1.cpp -o 1.out
run: 1.out
    ./1.out
clean:
    rm -f 1.out
```

一个简单的makefile就这样诞生了

但没有结束

一个示例

如果一个工程有3个头文件，和8个C文件，我们为了完成前面所述的那三个规则，我们的Makefile应该是下面的这个样子的。 edit : main.o kbd.o command.o display.o / insert.o search.o files.o utils.o cc -o edit main.o kbd.o command.o display.o / insert.o search.o files.o utils.o

```
main.o : main.c defs.h
cc -c main.c
```

```
kbd.o : kbd.c defs.h command.h
cc -c kbd.c
command.o : command.c defs.h command.h
cc -c command.c
display.o : display.c defs.h buffer.h
cc -c display.c
insert.o : insert.c defs.h buffer.h
cc -c insert.c
search.o : search.c defs.h buffer.h
cc -c search.c
files.o : files.c defs.h buffer.h command.h
cc -c files.c
utils.o : utils.c defs.h
cc -c utils.c
clean :
rm edit main.o kbd.o command.o display.o /
insert.o search.o files.o utils.o
```

> 反斜杠（/）是换行符的意思。这样比较便于Makefile的易读。

我们可以把这个内容保存在文件为“Makefile”或“makefile”的文件中，然后在该目录下直接输入命令“make”就可以生成执行文件edit。如果要删除执行文件和所有的中间目标文件，那么，只要简单地执行一下“make clean”就可以了。

在这个makefile中，目标文件（target）包含：执行文件edit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。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 edit 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，换言之，目标文件是哪些文件更新的。

在定义好依赖关系后，后续的那一行定义了如何生成目标文件的操作系统命令，一定要以一个Tab键作为开头。记住，make并不管命令是怎么工作的，他只管执行所定义的命令。make会比较targets文件和prerequisites文件的修改日期，如果prerequisites文件的日期要比targets文件的日期要新，或者target不存在的话，那么，make就会执行后续定义的命令。

这里要说明一点的是，clean不是一个文件，它只不过是一个动作名字，有点像C语言中的lable一样，其冒号后什么也没有，那么，make就不会自动去找文件的依赖性，也就不会自动执行其后所定义的命令。要执行其后的命令，就要在make命令后明显得指出这个lable的名字。这样的方法非常有用，我们可以在一个makefile中定义不用的编译或是和编译无关的命令，比如程序的打包，程序的备份，等等。

---

make是如何工作的

> 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“edit”这个文件，并把这个文件作为最终的目标文件。
3. 如果edit文件不存在，或是edit所依赖的后面的 .o 文件的文件修改时间要比edit这个文件新，那么，他就会执行后面所定义的命令来生成edit这个文件
4. 如果edit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 文件生命make的终极任务，也就是执行文件edit了。

这就是整个make的依赖性，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。

通过上述分析，我们知道，像clean这种，没有被第一个目标文件直接或间接关联，那么它后面所定义的命令将不会被自动执行，不过，我们可以显示要make执行。即命令——“make clean”，以此来清除所有的目标文件，以便重编译。

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file.c，那么根据我们的依赖性，我们的目标file.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file.o的文件也是最新的啦，于是file.o的文件修改时间要比edit要新，所以edit也会被重新链接了（详见edit目标文件后定义的命令）。

而如果我们改变了“command.h”，那么，kdb.o、command.o和files.o都会被重编译，并且，edit会被重链接。

作者语

> 关于makefile 我就看懂了这些 所以就写了这些 还有一些我现在还用不到所以不写 等到我用到了再说 嘿嘿 嘿 $\int{10}^{20}$