Makefile 编写指南

- 概述

什么是 Makefile? Makefile 定义了一系列的规则来指定,哪些文件需要先变异，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作。

Makefile 带来的好处就是——自动化编译，一旦写好，只需要一个 make 命令，整个工程完全自动编译，极大地提高了软件开发效率。make 是一个命令工具，是一个解释 Makefile 中指令的命令工具。

- Makefile 规则

```
target ...: prerequisites...
[TAB]command...
```

target 可以是 Object File，也可以是执行文件，还可以是一个标签。
我们用一个简单的例子来说明这个规则。首先假设我们有下列文件。
User.h
User.cpp
Main.cpp

现在我们要编译出名为 UserInfo 的可执行文件,那么 Makefile 如下(方括号是为了突出这里是 TAB 而不是空格):

```
UserInfo: User.o Main.cpp
[TAB]g++ -o UserInfo Main.cpp User.o
User.o: User.h User.cpp
[TAB]g++ -c User.cpp
clean:
[TAB]rm User.o UserInfo
```

下面逐一解释,make 会在 Makefile 中寻找第一个目标,在这里就是 UserInfo，这个目标被作为最终的目标文件。冒号后面是依赖，在这里， UserInfo 依赖于 User.o 与 Main.cpp，也就是说，当这两个文件发生了变化，那么 make 就会执行下面的命令。User.o 以此类推。最后的 clean，不是一个目标文件，它是一个标签，是一个动作名字，冒号后面什么都没有，make 即不会去寻找其依赖。这里 clean 相当于一个命令，可以像下面这样执行：

```
make clean
```

此时即会删除掉 User.o 和 UserInfo。在这里，标签是可以自行定义的。

- Makefile 中的变量

话说，那如果是比较大的工程，这么写，那岂不是写到蛋碎都写不完嘛？
好的，Makefile 是有变量的。

我们继续以一个简单的例子来学习。
假设有以下文件：
    Date.h
    Date.cpp
    User.h
    User.cpp
    Main.cpp
那么，Makefile 有如下：

```
OBJECTS=Date.o\
        User.o
EXEC=UserInfo
CC=g++

$(EXEC): Date.o User.o Main.cpp
[TAB]$(CC) -o $(EXEC) Main.cpp Date.o User.o
Date.o: Date.h Date.cpp
[TAB]$(CC) -c Date.cpp
User.o: Date.h Date.cpp User.h User.cpp
[TAB]$(CC) -c User.cpp
clean:
[TAB]rm $(OBJECTS) $(EXEC)
```

如果你有 Shell 的使用经验，那么这种写法你肯定不会陌生。下面是解释：首先，我们定义了 3 个变量，分别是 OBJECTS，EXEC，CC 来表示我们的 object files，executable file，以及 compiler。注意：这里等号两边无空格。

之后，变量通过 $(Val_Name) 的形式来使用。

在这里，User 的剧情设定是它要调用 Date 里面的东东，所以存在一个依赖关系。

以上就是简单的 Makefile 的使用，至于 Makefile 的其它功能，我们也留给大家去挖掘学习，而 make 命令的更多选项，我们也建议大家在终端输入

man make 来查看。