Linux项目自动化构建工具-make/Makefile

> • 会不会写makefifile，从一个侧面说明了一个人是否具备完成大型工程的能力
> • 一个工程中的源文件不计数，其按类型、功能、模块分别放在若干个目录中，makefifile定义了一系列的规则来指定，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，甚至于进行更复杂的功能操作
> • makefifile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，极大的提高了软件开发的效率。
> • make是一个命令工具，是一个解释makefifile中指令的命令工具，一般来说，大多数的IDE都有这个命令，比如：Delphi的make，Visual C++的nmake，Linux下GNU的make。可见，makefile都成为了一种在工程方面的编译方法。
> • make是一条命令，makefifile是一个文件，两个搭配使用，完成项目自动化构建。

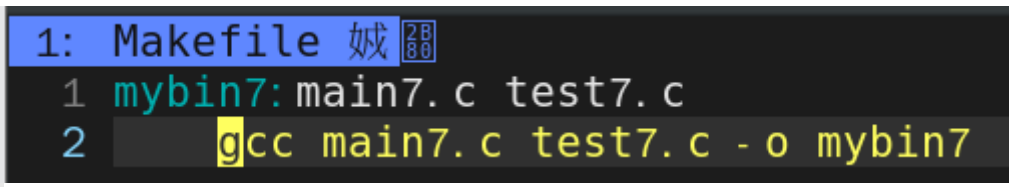
编译：

```
gcc main7.c test7.c -o mybin7
```

缺陷：当源文件有成百上千个时，gcc就很繁琐

Makefile文件：

```
touch Makefile
```




```
make -- 编译
```

在Makefile中写入：**(固定的套路)**

删除文件就可以直接：make clean

Makefile:

编写Makefile，本质上是在编写依赖关系和依赖方法！！！

1.依赖关系

- 上面的文件 hello ,它依赖 hell.o
- hello.o ， 它依赖 hello.s hello.s ， 它依赖 hello.i
- hello.i ， 它依赖 hello.c

2.依赖方法:

- gcc hello.* -option hello.* ,就是与之对应的依赖关系

.PHONY：可以理解为makefile的“关键字”，.PHONY：clean（伪目标）：总是被执行的!

原理

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“Makefifile”或“makefifile”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“hello”这个文件，并把这个文件作为最终的目标文件。
3. 如果hello文件不存在，或是hello所依赖的后面的hello.o文件的文件修改时间要比hello这个文件新（可以用touch 测试），那么，他就会执行后面所定义的命令来生成hello这个文件。
4. 如果hello所依赖的hello.o文件不存在，那么make会在当前文件中找目标为hello.o文件的依赖性，如果找到则再根据那一个规则生成hello.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 hello.o 文件，然后再用 hello.o 文件声明make的终极任务，也就是执行文件hello了。
6. 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。
7. 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。
8. 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。

实际的工作过程:

最终版本:

- $^:依赖文件列表
- $@:目标文件
- %.c:当前目录下的所有的.c文件展开
- %.o:对应的.c形成的.o

$<:%.c所代表的源文件，一个一个的拿出来，用gcc进行编译，形成同名的.o

项目清理

> • 工程是需要被清理的
> • 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。
> • 但是一般我们这种clean的目标文件，我们将它设置为伪目标,用 .PHONY 修饰,伪目标的特性是，总是被执行的。
> • 可以将我们的 hello 目标文件声明成伪目标，测试一下。