

Makefile

传统的编译方式

- 编译全部文件并链接



Suppose we have four source files as follows:

```
// factorial.cpp
#include "functions.h"
int factorial(int n)
{
    if (n == 1)
        return 1;
    else
        return n * factorial(n - 1);
}
```

```
// main.cpp
#include <iostream>
#include "functions.h"
using namespace std;

int main()
{
    print_hello();

    cout << "This is main:" << endl;
    cout << "The factorial of 5 is: " << factorial(5) << endl;

    return 0;
}
```

```
// printhello.cpp
#include <iostream>
#include "functions.h"
using namespace std;

void print_hello()
{
    cout << "Hello World!" << endl;
}
```

```
// functions.h
void print_hello();
int factorial(int n);
```

Normally, you can compile these files by the following command:

```
$ g++ -o hello main.cpp printhello.cpp factorial.cpp
```



```
lane@Lane:~$ cd Makefile_test/
lane@Lane:~/Makefile_test$ g++ -o hello main.cpp printhello.cpp factorial.cpp
lane@Lane:~/Makefile_test$ ./hello
Hello World!
This is main:
The factorial of 5 is: 120
```

- 编译单个文件+链接
 - 如果我只对少数文件进行了修改，只需要编译少数文件，最后进行链接
 - 节省了其余大多数文件编译所用的时间

```
● lane@Lane:~/Makefile_test$ g++ factorial.cpp -c
● lane@Lane:~/Makefile_test$ g++ main.cpp -c
● lane@Lane:~/Makefile_test$ g++ printhello.cpp -c
● lane@Lane:~/Makefile_test$ g++ *.o -o out
● lane@Lane:~/Makefile_test$ ./out
Hello World!
This is main:
The factorial of 5 is: 120
```

- 编译单个文件 `g++ factorial.cpp -c`
- 链接所有 `.o` 文件 `g++ *.o -o out` (规定生成的可执行文件叫out)

Makefile

- Makefile是一个文件，放在与程序文件在同一目录下
- 注释以 `#` 开头

version 1

▼ 示例

```
1 hello: main.cpp printhello.cpp factorial.cpp
2     g++ -o hello main.cpp printhello.cpp factorial.cpp
```

- 运行make的时候自动检查时间戳，若依赖文件中任意文件最后一次修改时间比目标文件晚，则运行command，反之不运行
- 也就是说如果我只改了一个文件，它也会全部编译并链接
- 其实就是把传统的编译方式写成了脚本，敲命令的时候只需要输入make，但没有减少不必要的编译时间

version 2


▼ 示例

```
1 CXX = g++
2 TARGET = hello
3 OBJ = main.o printhello.o factorial.o
4
```

```

5 $(TARGET): $(OBJ)
6     $(CXX) -o $(TARGET) $(OBJ)
7
8 main.o: main.cpp
9     $(CXX) -c main.cpp
10
11 printhello.o: printhello.cpp
12     $(CXX) -c printhello.cpp
13
14 factorial.o: factorial.cpp
15     $(CXX) -c factorial.cpp

```

- 看上去比version 1复杂，实际上类似于  编译单个文件+链接
- 每次执行make命令，会检查每个文件是否比目标文件新，只编译并链接修改后的文件，减少不必要的编译耗时

version 3

▼ 示例

```

1 CXX = g++
2 TARGET = hello
3 OBJ = main.o printhello.o factorial.o
4
5 $(TARGET): $(OBJ)
6     $(CXX) -o $@ $^      # $@=TARGET  $^=OBJ
7
8 %.o: %.cpp              # %.o %.cpp 通配符
9     $(CXX) -c $<        # $<=%.cpp (单个)
10
11 .PHONY: clean           # 如果没有这行，当目录中存在clean同名文件，就会产生歧义
12 clean:
13     rm -f *.o $(TARGET)

```

- 相比于version 2，那三个依赖文件被抽象成了通配符，写Makefile文件更简单
- 后面关于clean可不考虑

version 4

▼ 示例

```

1 CXX = g++
2 TARGET = hello
3 SRC = $(wildcard *.cpp)      # 当前目录下所有.cpp文件


```

```
4 OBJ = $(patsubst %.cpp, %.o, $(SRC)) # SRC中文件后缀由.cpp改成.o
5
6 $(TARGET): $(OBJ)
7     $(CXX) -o $@ $^
8
9 %.o: %.cpp
10     $(CXX) -c $<
11
12 .PHONY: clean
13 clean:
14     rm -f *.o $(TARGET)
```

- 这个版本与version 3相比就是把上面OBJ文件的获取变成了先获得SRC，再改后缀两步
- 无需手动输入依赖文件，只需放到当前目录下即可

参考

BV188411L7d2

 [示例文件](#) 5 项内容