Using Makefile

羅嘉諄

When your program is too sophisticated to compile !

```
$ g++ -c a.cpp -o a.o
$ g++ -c b.cpp -o b.o
$ g++ -c c.cpp -o c.o
$ g++ -c main.cpp -o main.o
$ g++ a.o b.o c.o main.o -o main
```

Compile :

$ make

Delete files :

$ make clean

Basic structure

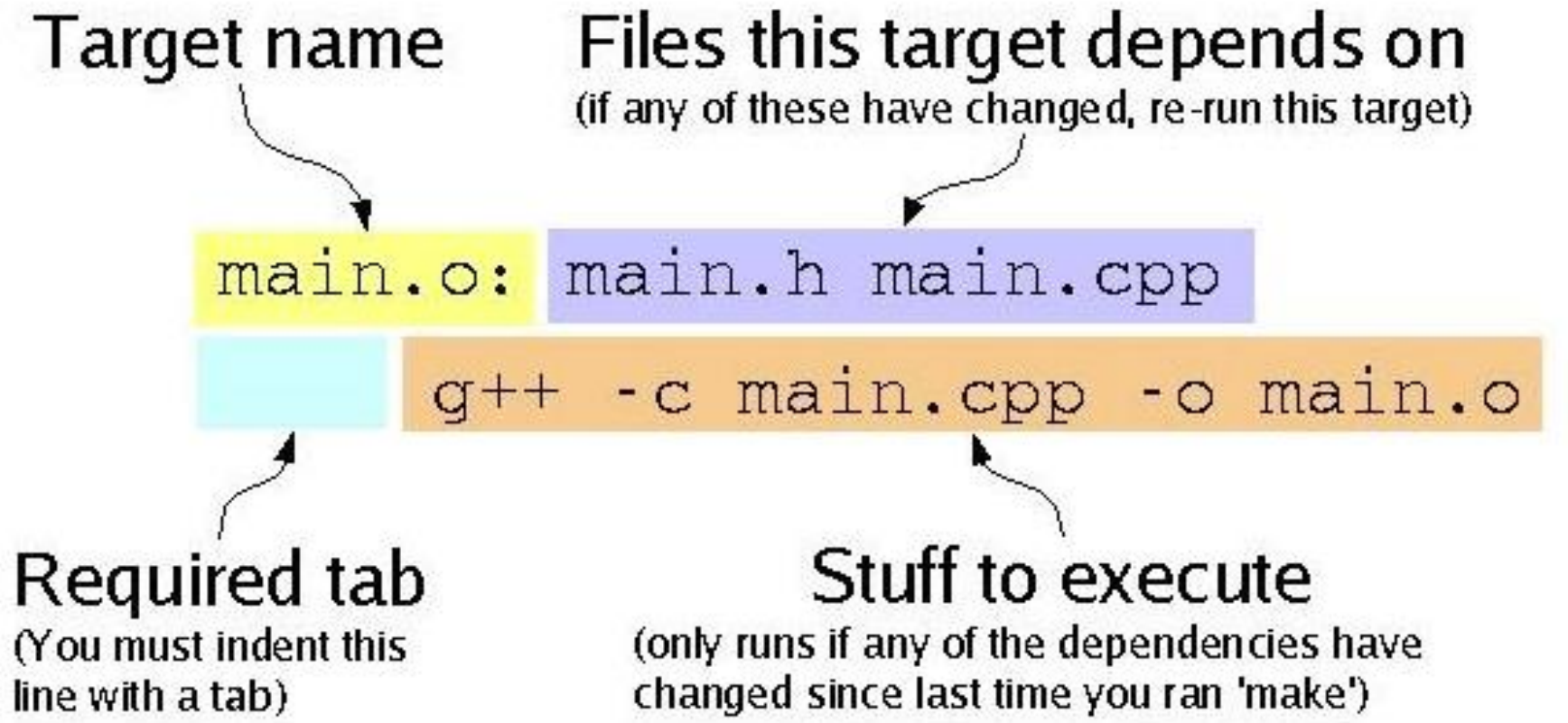
```
LIBS = -lm
OBJS = main.o haha.o sin_value.o cos_value.o
main: ${OBJS}
        gcc -o main ${OBJS} ${LIBS}
clean:
        rm -f main ${OBJS}
```

-o：制定目標名稱

變數宣告

制定規則

Creation rule



Basics

- # 後面加一些東東 → 註解
- 變數名稱 = 值 → 變數宣告
- $(變數名稱) → 使用先前宣告過的變數，若未曾被宣告，視為空字串
- -Wall → g++參數，允許發出g++提供的所有警告。
- -werror → 將警告轉成錯誤
- -Ox (x = 0, 1 , 2 , 3) → 編譯器優化選項，0為不優化，其餘數字越高，優化級別越高

參考資料：https://www.cnblogs.com/lidan/archive/2011/05/25/2239517.html

Basics (conti.)

- $^: a variable containing the list of prerequisites
- $@: target name
- $<: the first prerequisite
- $?: only the dependencies that are out of date
- $+: all dependencies including duplicates
- $|: all of the 'order only' prerequisites
- * : a wildcard, which means all
- % : one to one

An example

```
1  CC=g++
2  CFLAGS=-Wall -std=c++11 -Ofast
3  INCLUDES=-I../include/
4  HEADERS=a.h b.h c.h d.h e.h
5  LFLAGS=-L../lib/
6  LIBS=-lm -lsystemc
7  SOURCES=a.cpp b.cpp c.cpp d.cpp e.cpp
8  OBJECTS=$(SOURCES:.cpp=.o)
9  EXECUTABLE=main
10
11 all: $(SOURCES) $(EXECUTABLE)
12
13 $(EXECUTABLE): $(OBJECTS)
14     $(CC) $(CFLAGS) $(INCLUDES) $(OBJECTS) -o $@ $(LFLAGS) $(LIBS)
15
16 %.o:  %.cpp  $(HEADERS)
17     $(CC) $(CFLAGS) $(INCLUDES) -c $< -o $@
18
19 clean:
20     rm -rf *.o $(EXECUTABLE)
```