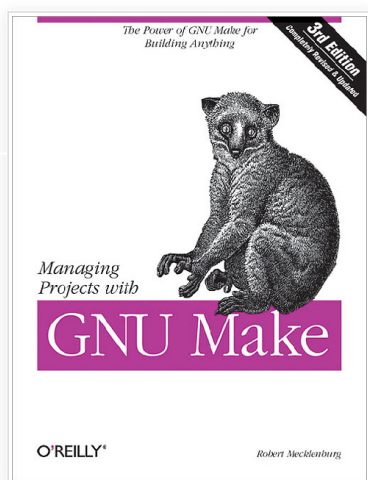




[Back to Home »](#) [Makefile »](#) [Makefile \(Part 1\)](#)

## Makefile (Part 1)

Thursday, June 25, 2015



Bài viết liên quan

+ [Cross compiler](#)

### 1. Đặt vấn đề

**Những vấn đề khi biên dịch**

- + Một chương trình đơn giản => chỉ có một vài file
- + Một chương trình "không đơn giản"
  - Nhiều dòng lệnh
  - Nhiều module
  - Nhiều người tham gia viết

**Vấn đề xảy ra:**

- + Khó quản lý một file lớn (cả người và máy)
- + Mỗi thay đổi cần thời gian biên dịch lâu
- + Nhiều người lập trình không thể thay đổi cùng một file đồng thời
- + Chương trình được phân ra thành nhiều module

**Giải pháp:** chia project ra thành nhiều file

**Mục tiêu:**

- + Chia thành các module một cách đúng đắn
- + Thời gian biên dịch ngắn nếu có sự thay đổi

Visitors	
 17,344	 106
 3,796	 101
 545	 96
 429	 73
 172	 64
 150	 47
FLAG counter	

## Labels

[Applications \(7\)](#)

[azure \(1\)](#)

[Beginning Linux Programming \(21\)](#)

[bootloader \(1\)](#)

[C \(28\)](#)

[CPP \(18\)](#)

[Cross compiler \(3\)](#)

[Database \(1\)](#)

[DirectFB \(4\)](#)

[DLib \(1\)](#)

[docker \(3\)](#)

[Face detection \(2\)](#)

[Face recognition \(2\)](#)

[FriendlyARM \(6\)](#)

[Gallery \(5\)](#)

[IPFire \(1\)](#)

[kernel \(1\)](#)

[Library \(16\)](#)

[Linux Basic \(33\)](#)

[Linux PC \(20\)](#)

+ Dễ dàng bảo trì cấu trúc project và sự phụ thuộc

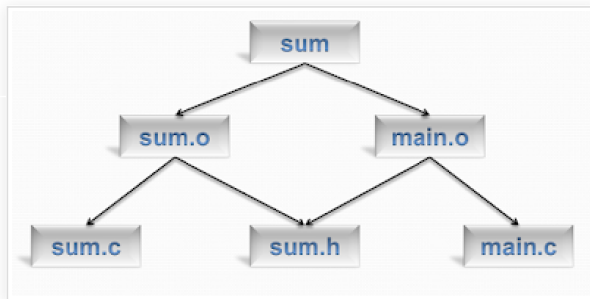
## 2. Makefile

### 2.1 Makefile là gì?

- + Makefile là một file dạng **script** chứa các thông tin:
  - Cấu trúc project (file, sự phụ thuộc)
  - Các lệnh để tạo file
- + Lệnh **make** sẽ đọc nội dung Makefile, hiểu kiến trúc của project và thực thi các lệnh

### 2.2 Cấu trúc project

- + Cấu trúc và sự phụ thuộc của project có thể được biểu diễn bằng một DAG (Directed Acyclic Graph)
- + Thí dụ:
  - Chương trình chứa 3 file: **main.c**, **sum.c**, **sum.h**
  - File **sum.h** được dùng bởi cả 2 file **main.c** và **sum.c**
  - File thực thi là **sum**



sum.h

```

#ifndef SUM_H_
#define SUM_H_
#include <stdio.h>
int sum(int a, int b);
#endif /* SUM_H_ */

```

sum.c

```

#include "sum.h"

int sum(int a, int b){
    return (a+b);
}

```

main.c

```

#include <stdio.h>
#include "sum.h"

int main(int argc, char **argv){

    int x;
    x= sum(1, 2);
    printf("x = %d \n", x);

    return 1;
}

```

Makefile

Makefile (4)

Networking (10)

OpenCV (5)

OpenGL (5)

OpenVPN (1)

OpenWrt (1)

Programming techniques (13)

QT Framework (5)

Raspberry PI (7)

rootfs (1)

Shell (18)

tensorflow (1)

Tools (10)

## References

<http://www.makelinux.net/books/>

Embedded247

<http://linux.die.net/>

<http://www.tutorialspoint.com/unix>

<http://www.cprogramming.com/>

<http://www.tutorialspoint.com/cprogramming>

<http://www.tutorialspoint.com/cplusplus>

<http://www.cplusplus.com/>

```

sum: main.o sum.o
    gcc -o sum main.o sum.o
main.o: main.c sum.h
    gcc -c main.c
sum.o: sum.c sum.h
    gcc -c sum.c

```

### Phân tích cấu trúc Makefile:

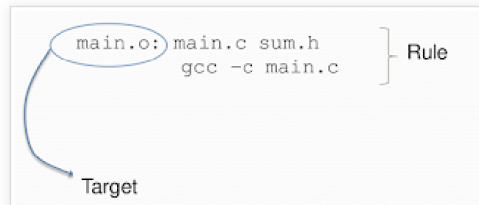
```

main.o: main.c sum.h
    gcc -c main.c

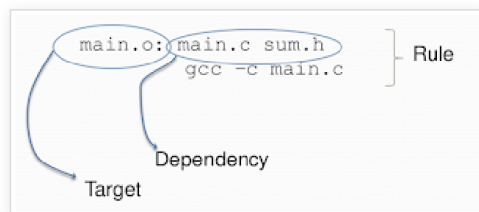
```

} Rule

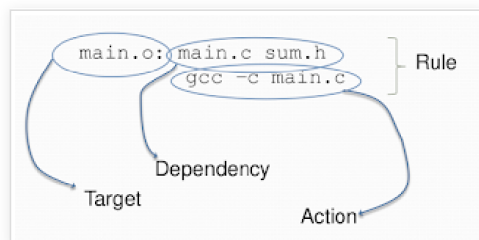
Rule: Có thể có nhiều Rule, trong ví dụ trên có 3 rule



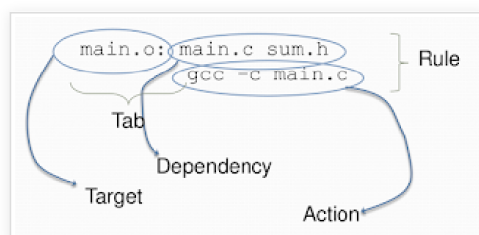
Target



Dependency: Cần thiết để tạo ra Target



Action: Câu lệnh compile để tạo ra Target từ Dependency,  
option là **"-c"** để tạo ra file object từ file source code,  
option là **"-o"** để tạo ra file chương trình nhị phân từ file object  
Mỗi Rule có thể không có hoặc có nhiều Action



Action được thực lùi vào một Tab (bằng 4 lần blank space) so với Target

**Thứ tự thực hiện:**

Khi bạn thực hiện lệnh make, chương trình make sẽ nhảy đến target đầu tiên là sum với mục đích để tạo ra nó, để làm được điều đó make đi kiểm tra lần lượt (từ trái qua phải: main.o -> sum.o) xem các dependency của sum đã tồn tại chưa. Dependency đầu tiên là main.o chưa có, cần phải tìm rule nào đó mà ở đó main.o đóng vai trò là target, make tìm ra rule thứ 2 và nó nhảy đến thực hiện rule thứ 2 để tạo ra main.o (lưu ý khi nó chạy rule 2 thì cũng giống y như khi chạy rule đầu tiên, có thể coi như là đệ quy). Sau khi tạo ra main.o, make trở về rule 1 để tiến hành kiểm tra tiếp xem dependency thứ hai là sum.o đã tồn tại chưa, sum.o chưa có vì thế make tiến hành các bước tương tự như đối với main.o. Sau khi tất cả các dependency được tạo ra, make mới có thể tạo ra file chạy cuối cùng là sum.

Vậy make thực hiện theo nguyên tắc / thứ tự như sau:

+ Tạo ra các file object trước (**main.o, sum.o**)

+ Tạo ra chương trình nhị phân cuối cùng từ các file object đã được tạo ra trước đó (**sum**)

```
sum: main.o sum.o      > 3
gcc -o sum main.o sum.o
main.o: main.c sum.h   > 1
gcc -c main.c
sum.o: sum.c sum.h     > 2
gcc -c sum.c
```

**Compile & Execute:**

"cd" vào thư mục chứa project và thực hiện lệnh **make**, chương trình **sum** sẽ được tạo ra cùng với 2 file object là **sum.o** và **main.o**, chạy thử chương trình:

```
[ninhld@localhost ~]$
[ninhld@localhost ~]$ cd /home/ninhld/Github/eslinuxprogramming/Makefile
[ninhld@localhost Makefile]$
[ninhld@localhost Makefile]$ make
gcc -c main.c
gcc -c sum.c
gcc -o sum main.o sum.o
[ninhld@localhost Makefile]$
[ninhld@localhost Makefile]$
[ninhld@localhost Makefile]$ ./sum
x = 3
[ninhld@localhost Makefile]$
[ninhld@localhost Makefile]$
```

Location: /home/ninhld/Github/eslinuxprogramming/Makefile		
Name	Size	Type
main.c	135 bytes	C source code
main.o	1.6 kB	object code
Makefile	112 bytes	Makefile
sum	8.6 kB	executable
sum.c	60 bytes	C source code
sum.h	94 bytes	C header
sum.o	1.2 kB	object code

**2.3 Nguyên lý biên dịch lại của Makefile**

Việc compile lại project dựa vào hai yếu tố:

- + Thời gian chỉnh sửa (date modified)
- + Cây phụ thuộc trong Makefile

Có nghĩa là một khi Dependency thay đổi thì Target tương ứng cũng phải được compile lại.

Theo Makefile như trong ví dụ trên:

- + Nếu chỉnh sửa **sum.h** thì **main.o** và **sum.o** phải được tạo lại, mặt khác **main.o** & **sum.o** lại là dependency của **sum** nên **sum** sẽ được tạo lại

```
[ninhld@localhost Makefile]$ make
gcc -c main.c
gcc -c sum.c
gcc -o sum main.o sum.o
```

- + Nếu chỉnh sửa **sum.c** thì **sum.o** được tạo lại, đương nhiên **sum** phụ thuộc **sum.o** nên **sum** sẽ được tạo lại

```
[ninhld@localhost Makefile]$ make
gcc -c sum.c
gcc -o sum main.o sum.o
```

Khi viết Makefile tránh gộp tất cả lại làm một như dưới đây, vì khi đó nếu một trong các file source code thay đổi thì cũng phải compile lại tất cả các file khác, điều đó làm mất nhiều thời gian:

Makefile

```
sum: main.c sum.c sum.h
    ${CC} -o sum main.c sum.c sum.h
```

Xem tiếp [Makefile \(Part 2\)](#)

Share This



0 bình luận

Sắp xếp theo


Cũ nhất

Thêm bình luận...

[Plugin bình luận trên Facebook](#)

Leave a Reply

Enter your comment...

 Comment as: huongpv@gma ▼

Sign out

Publish

Preview

☐ Notify me

[Subscribe to Posts](#) | [Subscribe to Comments](#)

[PREV](#)[NEXT](#)

---

- Copyright © 2020 Lập trình hệ thống nhúng Linux . Powered by Luong Duy Ninh -