Make file
(Compiler with GCC/G++)

Introduction

* Small program → single file → compile very easy.
* Example:

```c
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
}
```

```
$> gcc -o HelloWorld HelloWorld.c
$> ./HelloWorld
```

Introduction (cont.)

* “Not so small” program:
  * Many lines of code;
  * Multiple components;
  * More than one programmer.

Introduction (cont.)

* Problems:
  * Long file are harder to manage (for both programmers and machines);
  * Every change requires long compilation;
  * Many programmers can not modify the same file simultaneously.
  * Divison to component is module.
* Solution: Make file.

GCC Compiler

gcc [options] sources with options:

* -o: general ouput file (as *.exe in Windows);
* -c: general object file (*.o as *.class file in Java);
* -I (i upper): folder contain <include> file;
* -l: library name;
* -L: path to library.

Make file

* Done in Unix by the Makefile mechanism
* A make file is a file (script) containing:
  * Project structure (files, dependencies);
  * Instructions for file creation.
* The make command reads a makefile, understand the project structure and make up the executable.
* Note that the Makefile mechanism is not limited to C program.

The basic Makefile*

* The basic makefile is composed of:

target: dependencies

[tab] system command

Rule

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello World!\n");
6  }
7
8
```

```
1  #makefile
2  CC = gcc
3  all:HelloWorld.o
4      $(CC) -o all HelloWorld.o
5  HelloWorld.o:HelloWorld.c
6      $(CC) -c HelloWorld.c
```

(*)http://www.gnu.org/software/make/manual/make.html

Example Makefile

* Edit two files: main.c and func.c:

```c
/*main.c file*/

int main()
{
    hi();
}

```

```c
/*func.c file*/
#include <stdio.h>

void hi()
{
    printf("HelloWorld!\n");
}

```

Example Makefile (cont.)

* Edit Makefile (save with name Makefile or makefile not extend file):

```
1  #makefile
2  CC = gcc
3  all:main.o func.o
4      $(CC) -o main main.o func.o
5  main.o:main.c
6      $(CC) -c main.c
7  func.o:func.c
8      $(C)C -c func.c
9
10
```

* Type: make to compile in terminal

Target in Makefile

* Type make is simulate type make all.

Note: Default the first target is execute.

* Type make clear.

```
1  #makefile
2  CC = gcc
3  all:main.o func.o
4      $(CC) -o main main.o func.o
5  main.o:main.c
6      $(CC) -c main.c
7  func.o:func.c
8      $(C)C -c func.c
9  clear:
10     rm -f *.o all
11
```

Mechanism of Makefile

File	Last modified
all	10:10
main.o	09:58
func.o	09:30
main.c	10:09
func.c	10:00

Library with Makefile in Linux

* Static library: contain object file (*.o) is created by ar tool.
  * Syntax: ar rcs libfunc.a func.o (libfunc.a);
  * Use: gcc –o main main.c libfunc.a
    or gcc –o main main.c –lfunc.
* Dynamic library:
  * Syntax: gcc –c-fPIC func.c (func.o)
    gcc shared –o libfunc.so func.o (libfunc.so).
  * Use: gcc –o main main.c libfunc.so
    or gcc –o main main.c –lfunc.