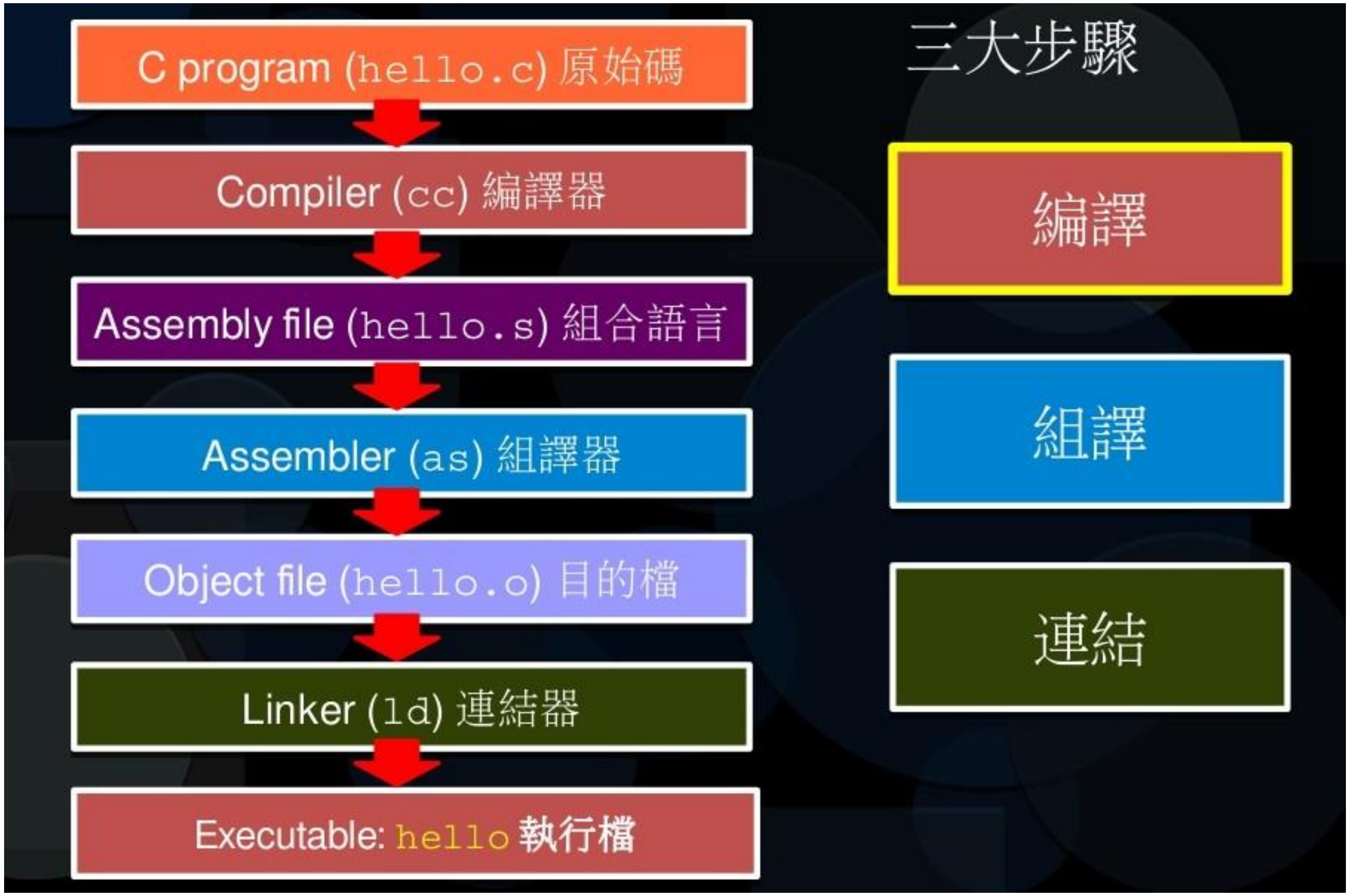
Makefile

2017.05.02

Outline

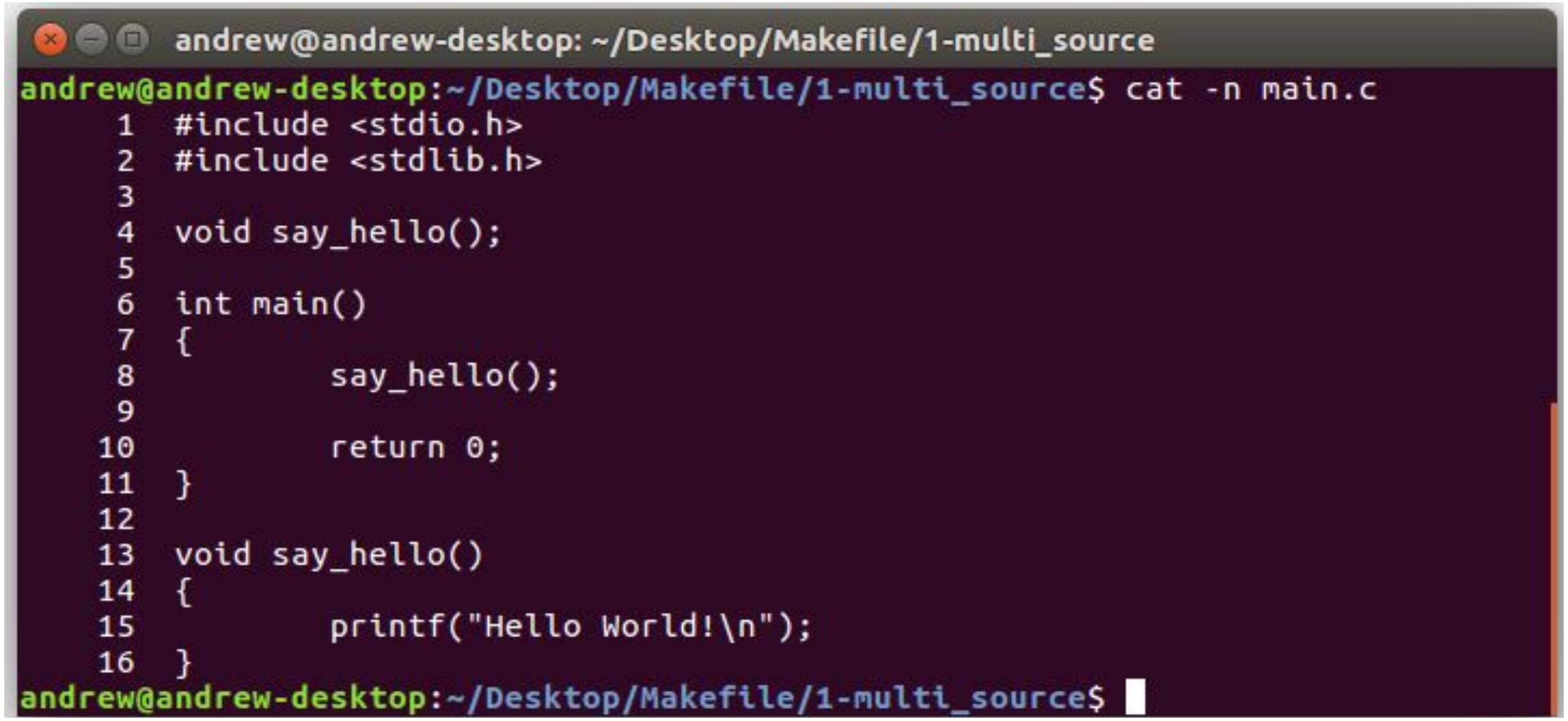
- 程式編譯的流程
- 如何把 source code 分成多個檔案？
- 最簡單的 Makefile
- 增加一些 Makefile 的規則
- After You Know How to Split Your Program
    - Folder Structure for a C Project
    - Create Your Own Library
    - Using Other’s Library
    - How to Make Your Library Useful

編譯程式的流程



來源 : https://www.slideshare.net/jserv/how-a-compiler-works-gnu-toolchain

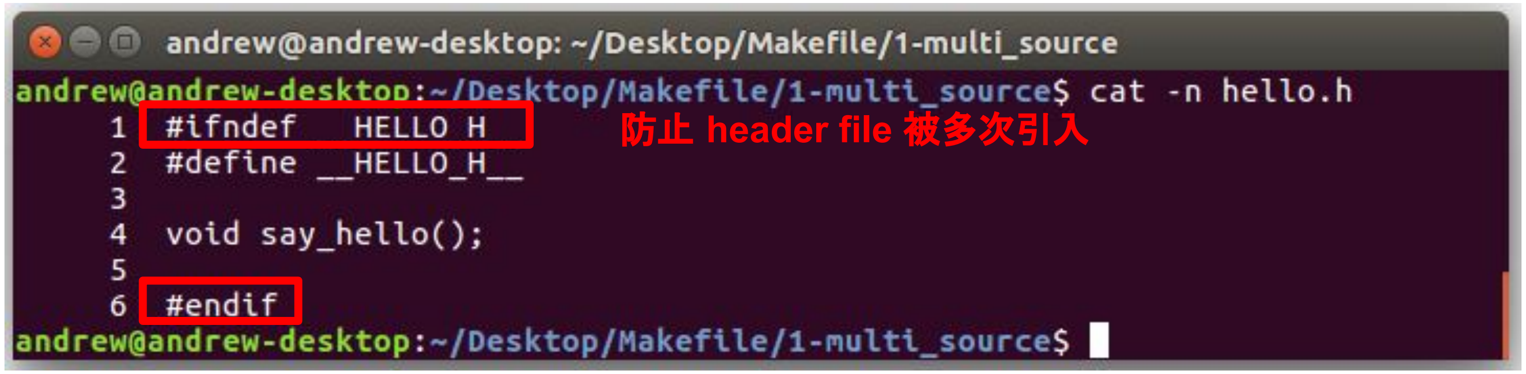
如何把 source code 分成多個檔案？



```
andrew@andrew-desktop:~/Desktop/Makefile/1-multi_source$ cat -n main.c
     1	#include <stdio.h>
     2	#include <stdlib.h>
     3	
     4	void say_hello();
     5	
     6	int main()
     7	{
     8		say_hello();
     9	
    10		return 0;
    11	}
    12	
    13	void say_hello()
    14	{
    15		printf("Hello World!\n");
    16	}
andrew@andrew-desktop:~/Desktop/Makefile/1-multi_source$ 
```

如何把 source code 分成多個檔案？

Step1: 新增一個 header file (.h 檔)



```
andrew@andrew-desktop: ~/Desktop/Makefile/1-multi_source
andrew@andrew-desktop:~/Desktop/Makefile/1-multi_source$ cat -n hello.h
     1  #ifndef __HELLO_H__
     2  #define __HELLO_H__
     3
     4  void say_hello();
     5
     6  #endif
andrew@andrew-desktop:~/Desktop/Makefile/1-multi_source$
```

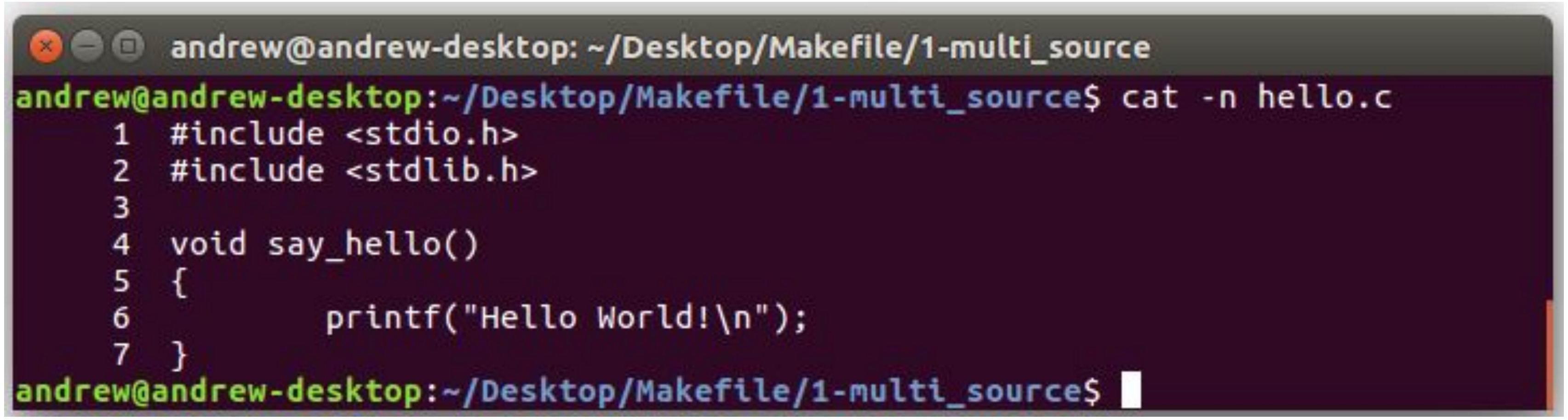
防止 header file 被多次引入

Header file 可以包含的東西：

1. 其他 .c 檔需要用到的 function 的 function declaration
2. 一些常數 (const int MAX_LEN = 1000;) 或是 macro (#define TRUE 1)...
3. struct declaration

如何把 source code 分成多個檔案？

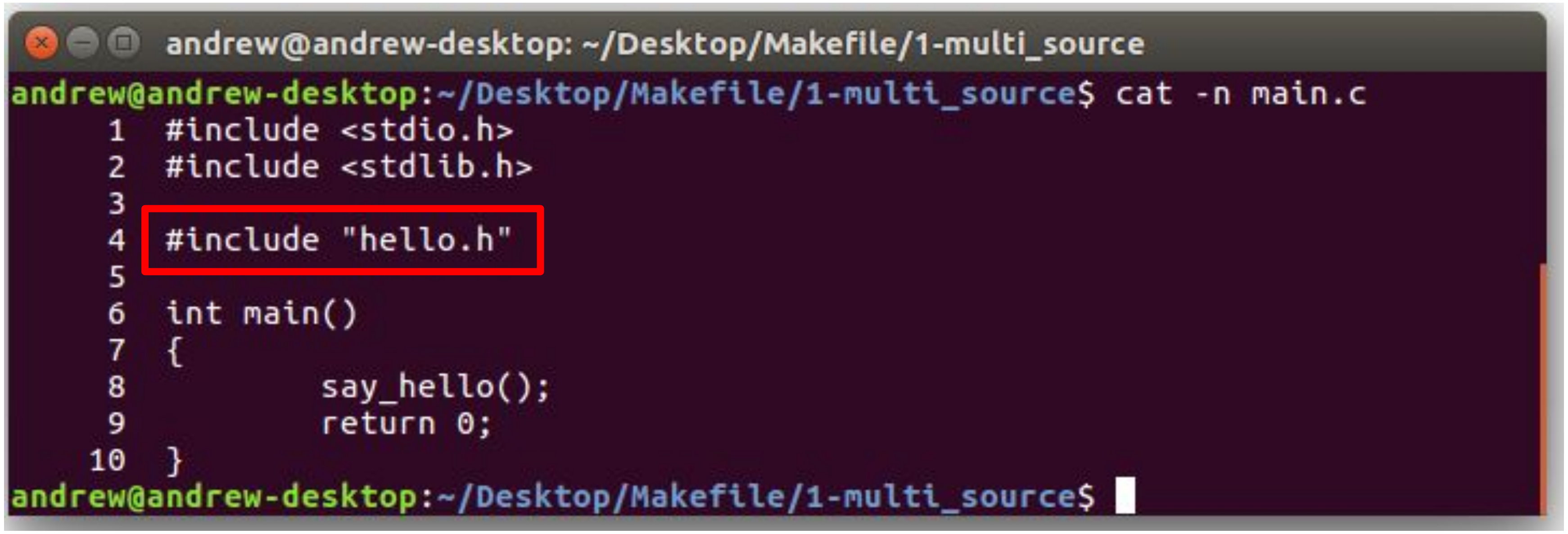
Step2: 將剛剛 header file 裡有宣告的 function 放到一個或多個 .c 檔裡頭



```
andrew@andrew-desktop: ~/Desktop/Makefile/1-multi_source
andrew@andrew-desktop:~/Desktop/Makefile/1-multi_source$ cat -n hello.c
     1	#include <stdio.h>
     2	#include <stdlib.h>
     3	
     4	void say_hello()
     5	{
     6		printf("Hello World!\n");
     7	}
andrew@andrew-desktop:~/Desktop/Makefile/1-multi_source$ 
```

如何把 source code 分成多個檔案？

Step3: 有用到在 header file 裡的 function 的 .c 檔需要 include 那個 header file



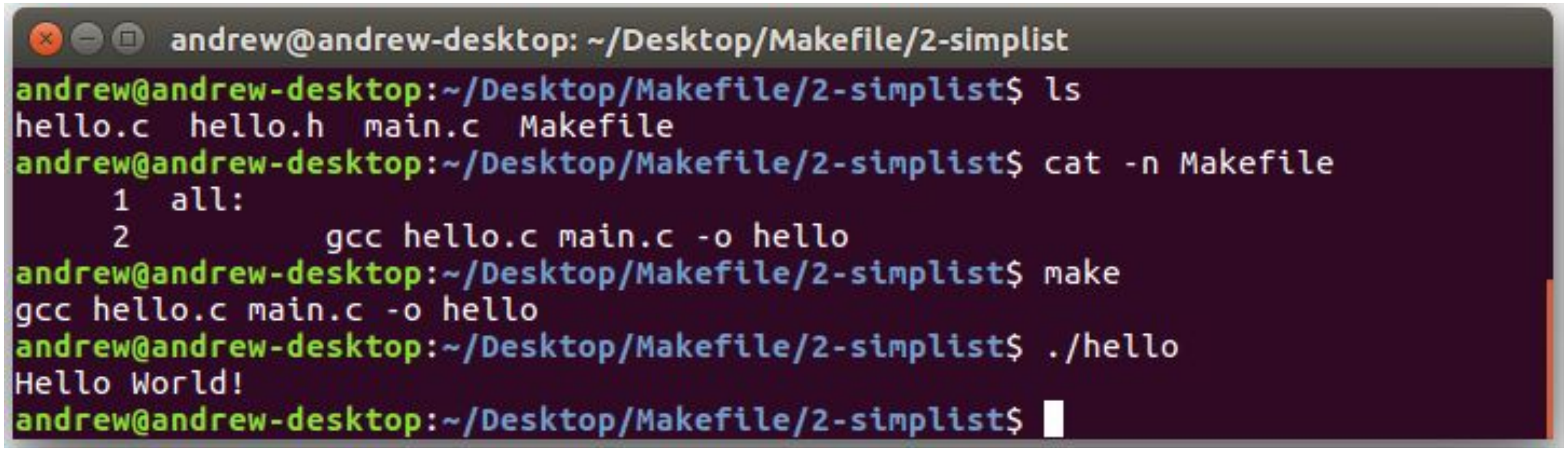
```
andrew@andrew-desktop:~/Desktop/Makefile/1-multi_source$ cat -n main.c
     1	#include <stdio.h>
     2	#include <stdlib.h>
     3	
     4	#include "hello.h"
     5	
     6	int main()
     7	{
     8	        say_hello();
     9	        return 0;
    10	}
andrew@andrew-desktop:~/Desktop/Makefile/1-multi_source$ 
```

為何需要學 Makefile ？

1. 節省時間
   只要輸入 make 就可以自動編譯 source code，可以節省重複輸入的時間。

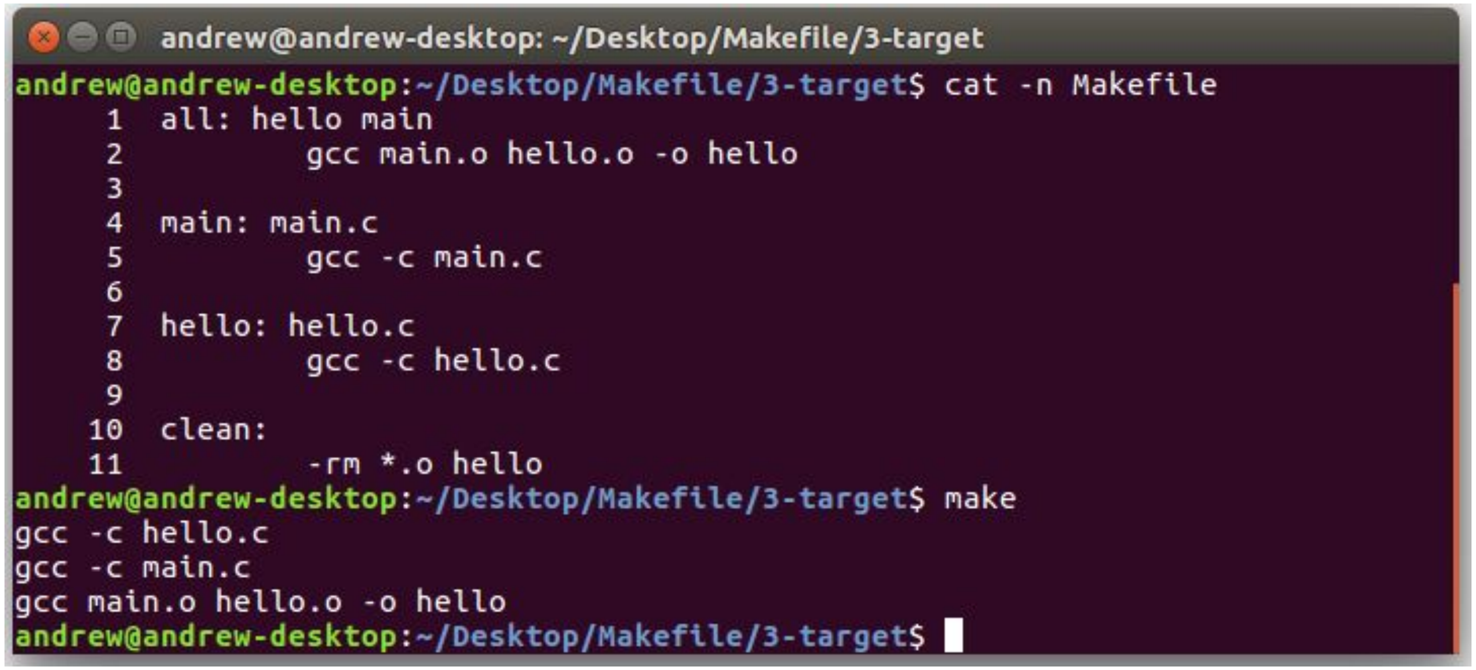
2. 許多 open source 的 project 編譯都會使用 Makefile，如果了解 Makefile，在編譯軟體時若遇到問題時就有機會可以解決。

最簡單的 Makefile



```
andrew@andrew-desktop:~/Desktop/Makefile/2-simplist$ ls
hello.c  hello.h  main.c  Makefile
andrew@andrew-desktop:~/Desktop/Makefile/2-simplist$ cat -n Makefile
     1	all:
     2		gcc hello.c main.c -o hello
andrew@andrew-desktop:~/Desktop/Makefile/2-simplist$ make
gcc hello.c main.c -o hello
andrew@andrew-desktop:~/Desktop/Makefile/2-simplist$ ./hello
Hello World!
andrew@andrew-desktop:~/Desktop/Makefile/2-simplist$ 
```

增加一些 Makefile 的規則 (1)

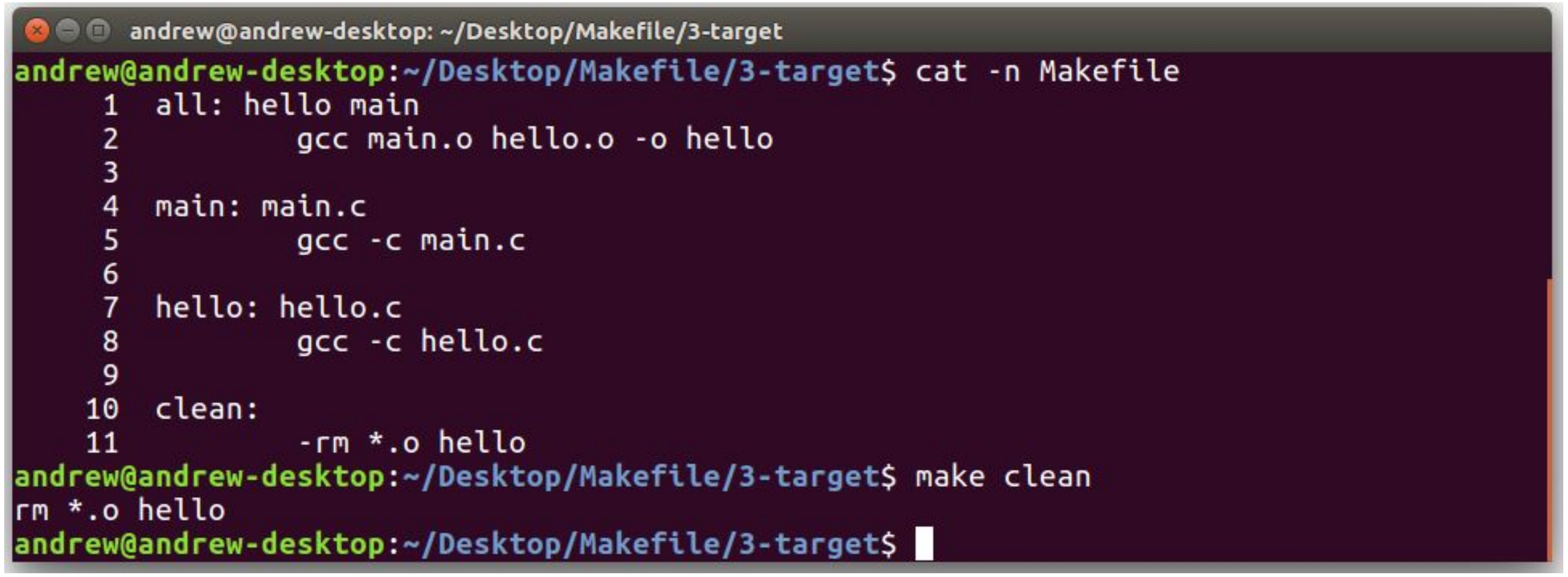


```
andrew@andrew-desktop:~/Desktop/Makefile/3-target$ cat -n Makefile
     1	all: hello main
     2		gcc main.o hello.o -o hello
     3
     4	main: main.c
     5		gcc -c main.c
     6
     7	hello: hello.c
     8		gcc -c hello.c
     9
    10	clean:
    11		-rm *.o hello
andrew@andrew-desktop:~/Desktop/Makefile/3-target$ make
gcc -c hello.c
gcc -c main.c
gcc main.o hello.o -o hello
andrew@andrew-desktop:~/Desktop/Makefile/3-target$
```

1. 執行 target all 時，發現它有兩個 prerequisites，所以就先跳去執行 target hello 和 target main
2. target hello 有一個 prerequisite hello.c，make 就照著下面的 command "gcc -c hello.c" 產生 object file
3. target hello 和 target main 都執行完後，就跳回來執行target all 下面的 command

增加一些 Makefile 的規則 (1)

註：
在 terminal 可以直接輸入 make + <target>，就可以直接執行 Makefile 裡的那個 target。

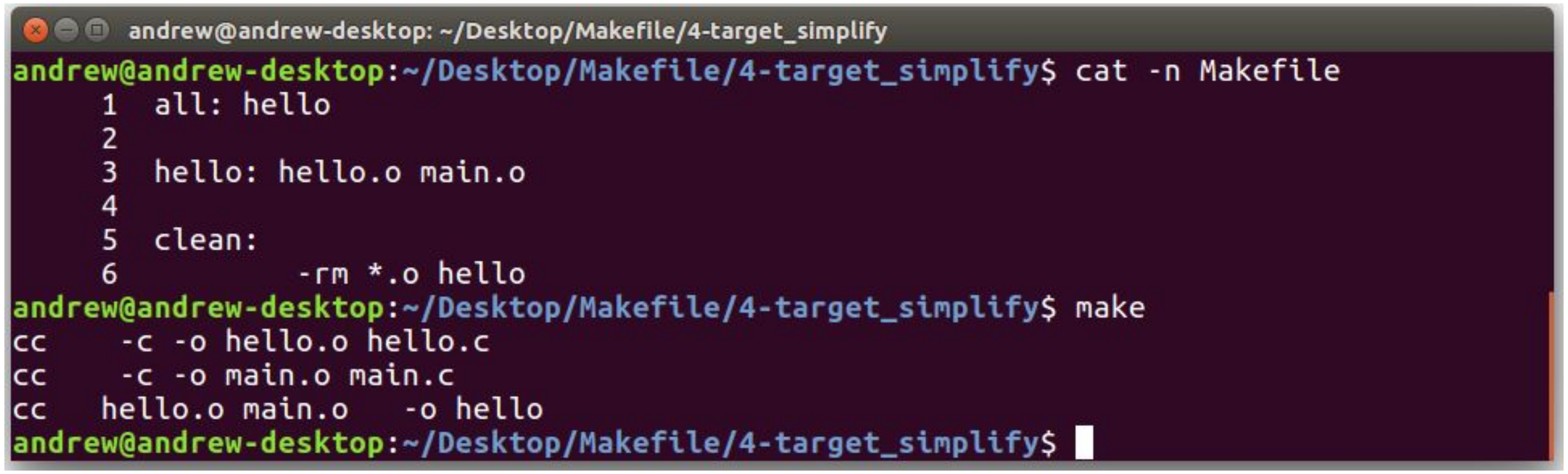


andrew@andrew-desktop: ~/Desktop/Makefile/3-target

```
andrew@andrew-desktop:~/Desktop/Makefile/3-target$ cat -n Makefile
     1	all: hello main
     2		gcc main.o hello.o -o hello
     3	
     4	main: main.c
     5		gcc -c main.c
     6	
     7	hello: hello.c
     8		gcc -c hello.c
     9	
    10	clean:
    11		-rm *.o hello
andrew@andrew-desktop:~/Desktop/Makefile/3-target$ make clean
rm *.o hello
andrew@andrew-desktop:~/Desktop/Makefile/3-target$ 
```

增加一些 Makefile 的規則 (1)

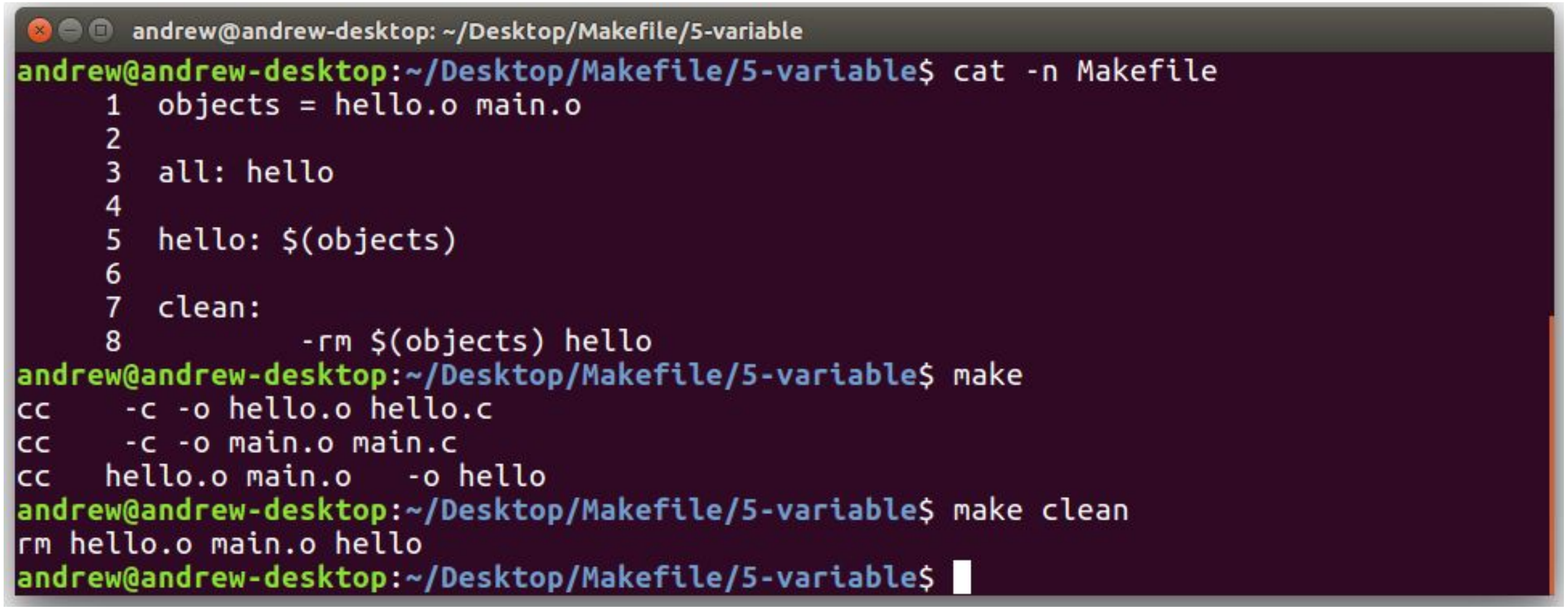
其實可以簡化 (using implicit rule)：



```
andrew@andrew-desktop:~/Desktop/Makefile/4-target_simplify$ cat -n Makefile
     1	all: hello
     2	
     3	hello: hello.o main.o
     4	
     5	clean:
     6		-rm *.o hello
andrew@andrew-desktop:~/Desktop/Makefile/4-target_simplify$ make
cc    -c -o hello.o hello.c
cc    -c -o main.o main.c
cc   hello.o main.o   -o hello
andrew@andrew-desktop:~/Desktop/Makefile/4-target_simplify$ 
```

增加一些 Makefile 的規則 (2)

Using Variable：

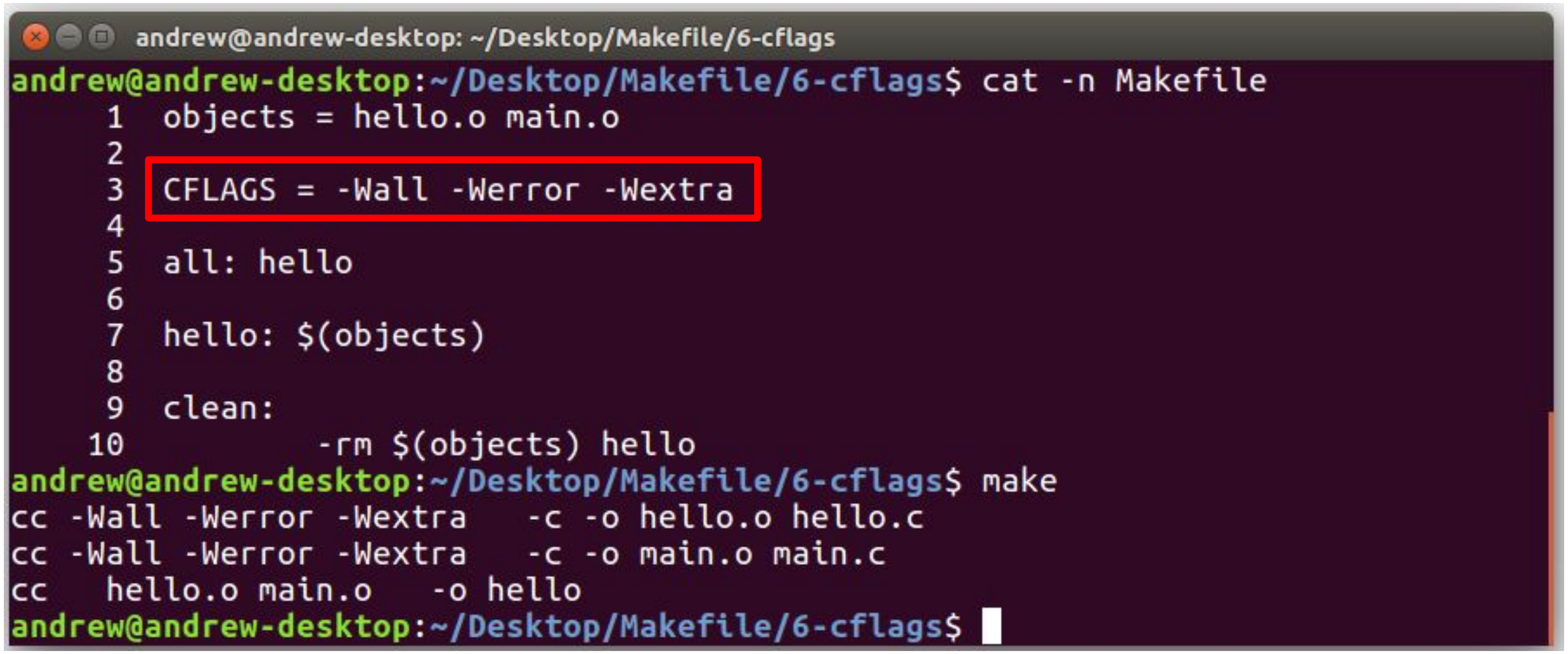


```
andrew@andrew-desktop:~/Desktop/Makefile/5-variable$ cat -n Makefile
     1	objects = hello.o main.o
     2	
     3	all: hello
     4	
     5	hello: $(objects)
     6	
     7	clean:
     8		-rm $(objects) hello
andrew@andrew-desktop:~/Desktop/Makefile/5-variable$ make
cc    -c -o hello.o hello.c
cc    -c -o main.o main.c
cc   hello.o main.o   -o hello
andrew@andrew-desktop:~/Desktop/Makefile/5-variable$ make clean
rm hello.o main.o hello
andrew@andrew-desktop:~/Desktop/Makefile/5-variable$ 
```

1. 定義時：變數 ＝ 值
2. 使用時：$(變數) 或 ${變數}

增加一些 Makefile 的規則 (2)

How to add -Wall -Wextra -Werror ?

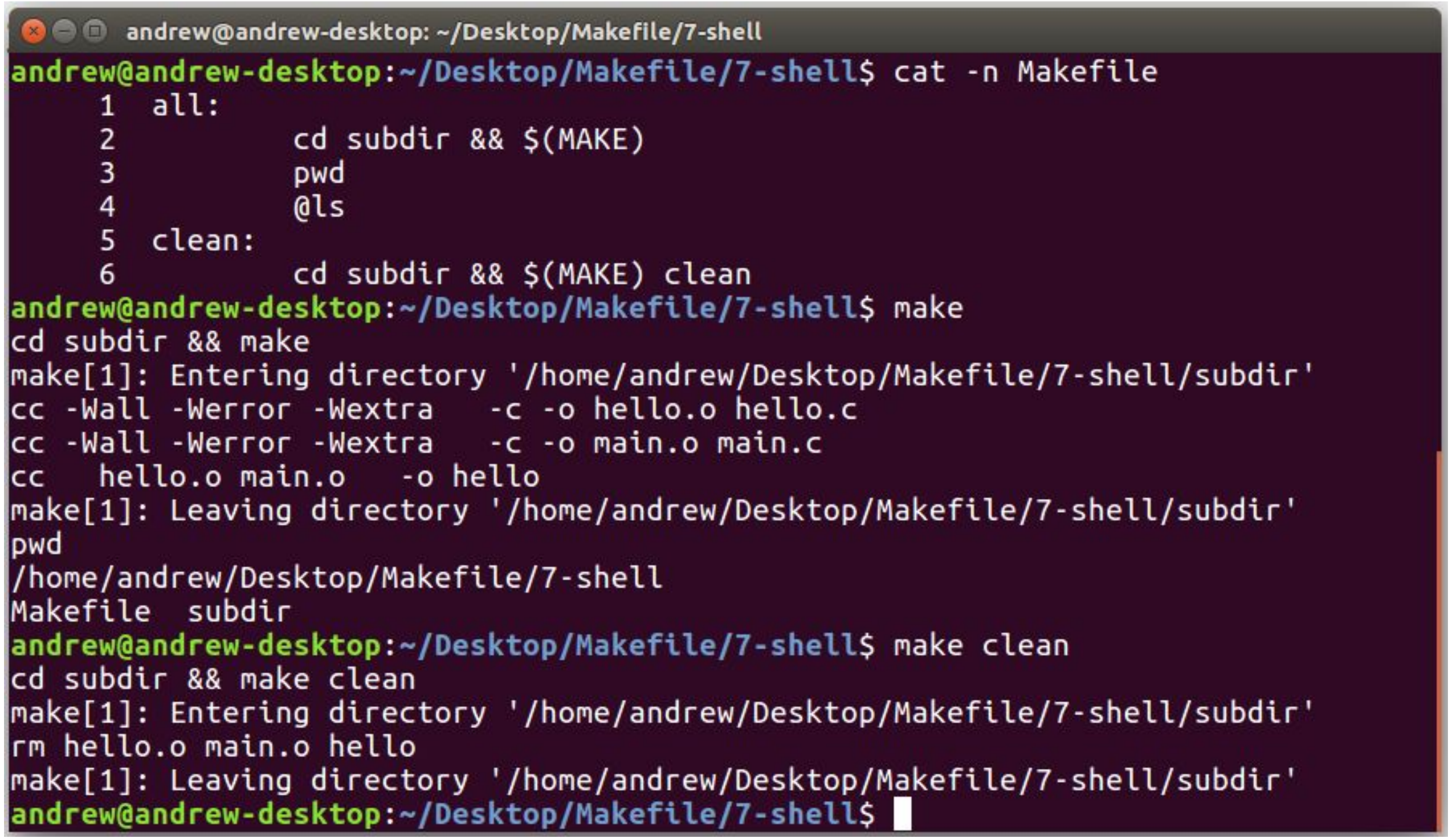


```
andrew@andrew-desktop:~/Desktop/Makefile/6-cflags$ cat -n Makefile
     1	objects = hello.o main.o
     2	
     3	CFLAGS = -Wall -Werror -Wextra
     4	
     5	all: hello
     6	
     7	hello: $(objects)
     8	
     9	clean:
    10		-rm $(objects) hello
andrew@andrew-desktop:~/Desktop/Makefile/6-cflags$ make
cc -Wall -Werror -Wextra   -c -o hello.o hello.c
cc -Wall -Werror -Wextra   -c -o main.o main.c
cc   hello.o main.o   -o hello
andrew@andrew-desktop:~/Desktop/Makefile/6-cflags$ 
```

Practice

增加一些 Makefile 的規則 (3)

在 Makefile 裡使用 shell



```
andrew@andrew-desktop:~/Desktop/Makefile/7-shell$ cat -n Makefile
     1	all:
     2		cd subdir && $(MAKE)
     3		pwd
     4		@ls
     5	clean:
     6		cd subdir && $(MAKE) clean
andrew@andrew-desktop:~/Desktop/Makefile/7-shell$ make
cd subdir && make
make[1]: Entering directory '/home/andrew/Desktop/Makefile/7-shell/subdir'
cc -Wall -Werror -Wextra   -c -o hello.o hello.c
cc -Wall -Werror -Wextra   -c -o main.o main.c
cc   hello.o main.o   -o hello
make[1]: Leaving directory '/home/andrew/Desktop/Makefile/7-shell/subdir'
pwd
/home/andrew/Desktop/Makefile/7-shell
Makefile  subdir
andrew@andrew-desktop:~/Desktop/Makefile/7-shell$ make clean
cd subdir && make clean
make[1]: Entering directory '/home/andrew/Desktop/Makefile/7-shell/subdir'
rm hello.o main.o hello
make[1]: Leaving directory '/home/andrew/Desktop/Makefile/7-shell/subdir'
andrew@andrew-desktop:~/Desktop/Makefile/7-shell$ 
```

Folder Structure for a C Project

src/	- source files
lib/	- required libraries
doc/	- documentation
tests/	- test files
build/	- where we build
README	- how to use, how to install, ……
Makefile	

Create Your Own Library

假設你非常喜歡你寫的 say_hello() 這個 function (我也很喜歡 :)), 然後你希望把它包成一個 library 讓大家都能一起用, 那應該怎麼做呢？

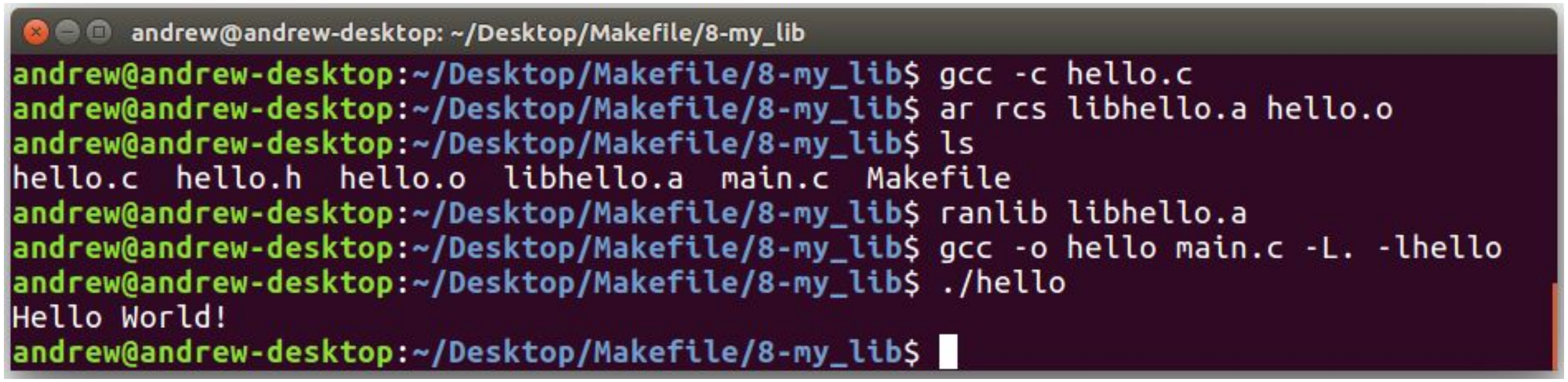
Create Your Own Library

首先要知道的是, library 有兩種:

1. 靜態 (.a 檔)
2. 動態 (.so 檔)

Create Your Own Library

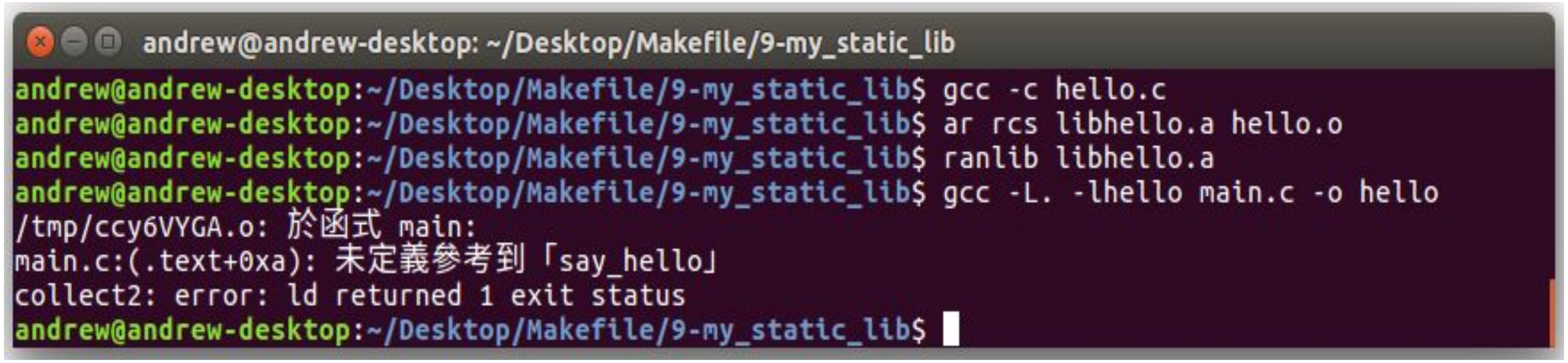
如何產生靜態 library 呢？



andrew@andrew-desktop: ~/Desktop/Makefile/8-my_lib

```
andrew@andrew-desktop:~/Desktop/Makefile/8-my_lib$ gcc -c hello.c
andrew@andrew-desktop:~/Desktop/Makefile/8-my_lib$ ar rcs libhello.a hello.o
andrew@andrew-desktop:~/Desktop/Makefile/8-my_lib$ ls
hello.c  hello.h  hello.o  libhello.a  main.c  Makefile
andrew@andrew-desktop:~/Desktop/Makefile/8-my_lib$ ranlib libhello.a
andrew@andrew-desktop:~/Desktop/Makefile/8-my_lib$ gcc -o hello main.c -L. -lhello
andrew@andrew-desktop:~/Desktop/Makefile/8-my_lib$ ./hello
Hello World!
andrew@andrew-desktop:~/Desktop/Makefile/8-my_lib$ 
```

Create Your Own Library



andrew@andrew-desktop: ~/Desktop/Makefile/9-my_static_lib

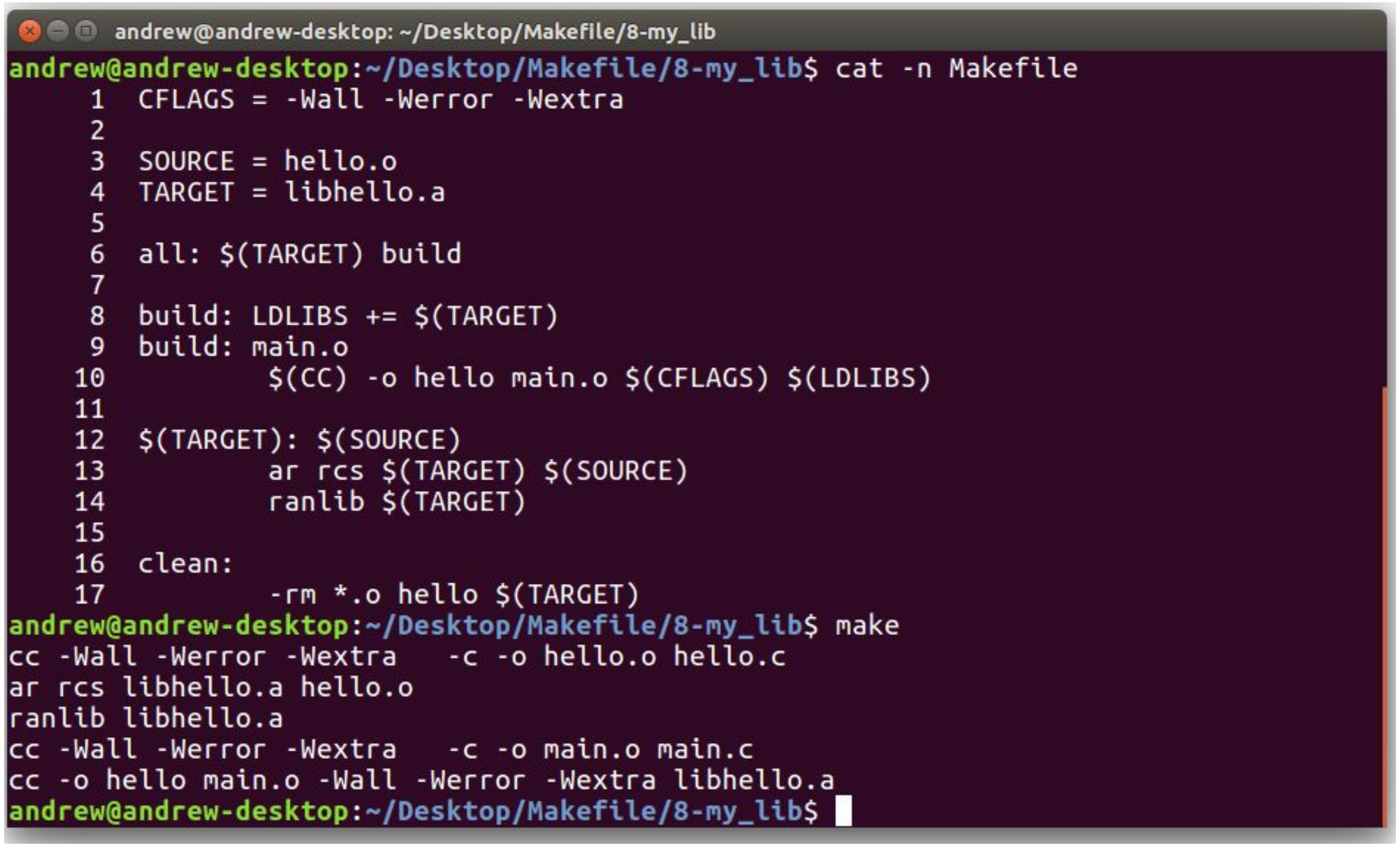
```
andrew@andrew-desktop:~/Desktop/Makefile/9-my_static_lib$ gcc -c hello.c
andrew@andrew-desktop:~/Desktop/Makefile/9-my_static_lib$ ar rcs libhello.a hello.o
andrew@andrew-desktop:~/Desktop/Makefile/9-my_static_lib$ ranlib libhello.a
andrew@andrew-desktop:~/Desktop/Makefile/9-my_static_lib$ gcc -L. -lhello main.c -o hello
/tmp/ccy6VYGA.o: 於函式 main:
main.c:(.text+0xa): 未定義參考到「say_hello」
collect2: error: ld returned 1 exit status
andrew@andrew-desktop:~/Desktop/Makefile/9-my_static_lib$ 
```

What happened?

Practice 鳨兔

Create Your Own Library

改寫 Makefile！



```
andrew@andrew-desktop: ~/Desktop/Makefile/8-my_lib
andrew@andrew-desktop:~/Desktop/Makefile/8-my_lib$ cat -n Makefile
     1	CFLAGS = -Wall -Werror -Wextra
     2	
     3	SOURCE = hello.o
     4	TARGET = libhello.a
     5	
     6	all: $(TARGET) build
     7	
     8	build: LDLIBS += $(TARGET)
     9	build: main.o
    10		$(CC) -o hello main.o $(CFLAGS) $(LDLIBS)
    11	
    12	$(TARGET): $(SOURCE)
    13		ar rcs $(TARGET) $(SOURCE)
    14		ranlib $(TARGET)
    15	
    16	clean:
    17		-rm *.o hello $(TARGET)
andrew@andrew-desktop:~/Desktop/Makefile/8-my_lib$ make
cc -Wall -Werror -Wextra   -c -o hello.o hello.c
ar rcs libhello.a hello.o
ranlib libhello.a
cc -Wall -Werror -Wextra   -c -o main.o main.c
cc -o hello main.o -Wall -Werror -Wextra libhello.a
andrew@andrew-desktop:~/Desktop/Makefile/8-my_lib$ 
```

Using Other’s Library

<stdio.h>

<stdlib.h>

<string.h>

“list.h”

“list_algos.h”

How to Make Your Library Useful

You should make your data type **ABSTRACT** !!!

```c
typedef struct ListNode {
        struct ListNode *next;
        struct ListNode *prev;
        void *value;
} ListNode;

typedef struct List {
        int count;
        ListNode *first;
        ListNode *last;
} List;
```

Summary

- 編譯流程
- 如果你完全不知道今天上課在做什麼......
- make + <target>
- $(variable)