写在前面的话

本节视频来自2018年为2440的裸机加强版视频，非常适合本教程。
要注意几点：

- 以前源码目录是`003_Makefile`，现在目录改为`04_2018_Makefile`
  - GIT仓库里：`01_all_series_quickstart\04_嵌入式Linux应用开发基础知识\source\04_2018_Makefile`
- 本节视频配套的文档，就是本文档，位于:
  - GIT仓库里：`01_all_series_quickstart\04_嵌入式Linux应用开发基础知识\doc_pic\04.2018_Makefile`

Makefile实例

前面讲了那么多Makefile的知识，现在开始做一个实例。

之前编译的程序`002_syntax`，有个缺陷，将其复制出来，新建一个`003_example`文件夹，放在里面。
在`c.c`里面，包含一个头文件`c.h`，在`c.h`里面定义一个宏，把这个宏打印出来。
c.c:

```
#include <stdio.h>
#include <c.h>

void func_c()
{
    printf("This is C = %d\n", C);
}
```

c.h:

```
#define C 1
```

然后上传编译，执行`./test`,打印出：

```
This is B
This is C =1
```

测试没有问题，然后修改`c.h`：

```
#define C 2
```

重新编译，发现没有更新程序，运行，结果不变，说明现在的Makefile存在问题。

为什么会出现这个问题呢， 首先我们test依赖c.o，c.o依赖c.c，如果我们更新c.c，会重新更新整个程序。
但c.o也依赖c.h，我们更新了c.h，并没有在Makefile上体现出来，导致c.h的更新，Makefile无法检测到。
因此需要添加:

```
c.o : c.c c.h
```

现在每次修改c.h，Makefile都能识别到更新操作，从而更新最后输出文件。

这样又冒出了一个新的问题，我们怎么为每个.c文件添加.h文件呢？对于内核，有几万个文件，不可能为每个文件依次写出其头文件。
因此需要做出改进，让其自动生成头文件依赖，可以参考这篇文章：http://blog.csdn.net/qq1452008/article/details/50855810

```
gcc -M c.c // 打印出依赖

gcc -M -MF c.d c.c  // 把依赖写入文件c.d

gcc -c -o c.o c.c -MD -MF c.d  // 编译c.o，把依赖写入文件c.d
```

修改Makefile如下：

```
objs = a.o b.o c.o

dep_files := $(patsubst %,.%.d, $(objs))
dep_files := $(wildcard $(dep_files))

test: $(objs)
    gcc -o test $^

ifneq ($(dep_files),)
include $(dep_files)
endif

%.o : %.c
    gcc -c -o $@ $< -MD -MF .$@.d

clean:
    rm *.o test

distclean:
    rm $(dep_files)

.PHONY: clean
```

首先用obj变量将.o文件放在一块。
利用前面讲到的函数，把obj里所有文件都变为.%.d格式，并用变量dep_files表示。
利用前面介绍的wildcard函数，判断dep_files是否存在。
然后是目标文件test依赖所有的.o文件。
如果dep_files变量不为空，就将其包含进来。
然后就是所有的.o文件都依赖.c文件，且通过-MD -MF生成.d依赖文件。
清理所有的.o文件和目标文件
清理依赖.d文件。

现在我门修改了任何.h文件，最终都会影响最后生成的文件，也没任何手工添加.h、.c、.o文件，完成了支持头文件依赖。

下面再添加CFLAGS，即编译参数。比如加上编译参数-Werror，把所有的警告当成错误。

```
CFLAGS = -Werror -Iinclude

……


%.o : %.c
    gcc $(CFLAGS) -c -o $@ $< -MD -MF .$@.d
```

现在重新make，发现以前的警告就变成了错误，必须要解决这些错误编译才能进行。在`a.c`里面声明一下函数：

```
void func_b();
void func_c();
```

重新make，错误就没有了。

除了编译参数-Werror，还可以加上-I参数，指定头文件路径，-Iinclude表示当前的inclue文件夹下。此时就可以把c.c文件里的`#include ".h"`改为`#include <c.h>`，前者表示当前目录，后者表示编译器指定的路径和GCC路径。