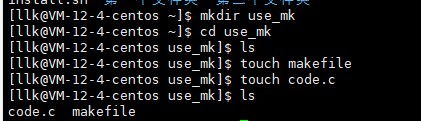
##### 一、make和makefile

###### 1、这两个都是什么

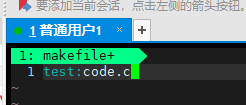
1.首先需要清楚的是make是一条指令，makefile是一个文件，并且makefile是当前目录下面的一个文件。两个搭配使用，完成项目自动化构建

2. makefile这个文件是和源文件同级别的，并且这个文件只有这个名字，而且这个名字可以是makefile或者是Makefile

//也就是说这个文件名的首字母是不区分大小写的

###### 2.编写makefile文件

1. 首先进入这个文件，打开并在第一行编写依赖关系



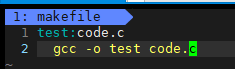
（1）冒号前面的是编译后的可执行程序，这个名字是自己定义的

（2）冒号后面的是想要进行编译的源代码文件。这个源代码文件的名字就是当前目录下面的文件

（3）这两个名字中间使用英文的 ： 进行隔开

（4）test可执行文件是依赖于code.c文件，并通过下面的依赖方法生成的

2.在第二行进行依赖方法的编写

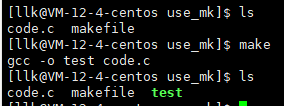


（1）第二行中使用就是前面的命令，用gcc -o 的命令，自定义编译后的可执行文件的名字，最好和上面的依赖关系名字保持一致

（2）并且依赖方法后面必须是 tab 键开头的

###### 2.使用make命令进行编译项目

1.在上面的makefile文件编写完成以后，就可以在项目中使用make指令进行源文件的编译，编译后就就会生成makefile指定名字的可执行文件



（1）但是make只能使用一次，编译成功过后，只要可执行文件存在，并且没有变更源代码，就不会再进行编译

###### 3.使用make命令对可执行文件或者源文件进行删除

1.可以在makefile文件中继续定义依赖方法和依赖关系 ps：依赖方法就是进行某个操作的名字（或者说是指令）

//另外后面另个并没有依赖关系

（1）使用 make clear 命令，会将可执行文件进行删除

（2）使用 make delete 命令，会将可执行文件，以及源文件一并删除。

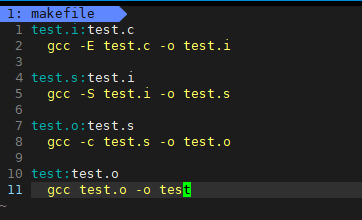
（3）这些依赖关系的名字都是自己定义的，并不是固定的

###### 4.make 会自动推导makefile中的依赖关系

1.源文件生成可执行程序需要进行4个步骤，我们将这四个步骤单独的拆开分别设置依赖关系和依赖方法，使用make依然能够生成可执行文件

//并且这四个步骤的顺序在makefile中可以任意顺序的书写

比如 test:test.o gcc test.o -o test .将四个步骤全部书写，任意顺序还是会生成可执行文件，并且也会生成其中的中间文件



2. 当然还是能够和上面一样设置方法将除可执行文件以外的进行删除，或者指定某些中间文件进行删除

###### 5.make默认执行的关系

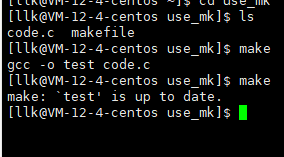
1.make默认值行的就是makefile中，最上面的一个依赖关系

2.如果将clean依赖关系放在makefile中最上面，那么只使用make指令默认执行的就是clean。

3.也就是说最好将编译成可执行程序的依赖关系放在最上面

###### 6.为什么指令只能执行一次

1.短时间在不修改源文件的时候执行make指令就会出现这种情况



2.但是在修改了源文件之后就会允许再次使用make了

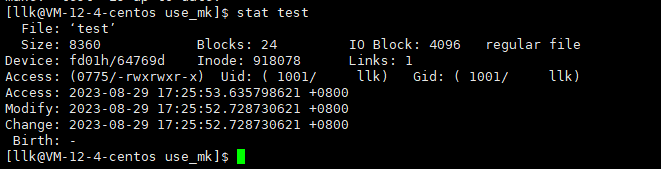
3.原因 ：源文件中的最近修改时间和可执行文件的最近修改时间作对比

//也就是说如果我们更改了源文件，历史上还有可执行文件，那么源文件的最近修改时间一定要比可执行文件的时间要新

//源文件和源文件编译的可执行程序一般情况下最后修改时间是不一样的

//看完7. 以后就只知道了比较的是修改时间 modify

###### 7.使用 stat 指令查看文件的修改访问等时间



1. access:：访问 这个代表最后访问这个文件的时间 比如cat打印，vim进入修改都是访问，这个时间修改的频率是特别高的

//但是为了提高整机效率，可能是在下面两个时间修改了几次以后，这个才会进行改变。但是早期的Linux系统就是修改非常频繁

2. modify：修改 这个代表最后修改这个文件的时间 这个修改是代表对文件的内容进行修改

3. change：改变 这个是对文件的属性进行改变 文件内容外的一切和这个文件相关的都是文件的属性，比如文件的权限

4. 文件=文件内容+文件属性

（1）一般情况下这三者并不是互相割裂的，而是一个动，可能另外一个也会改变，比如访问时间的改变，进行修改内容后，modify改变。内存变大，文件属性中的change时间也会改变

###### 8.使用touch命令手动的更改时间

1. touch 已经存在的文件名 ：这样的指令会更新三个时间

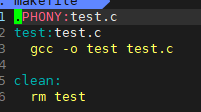
2. touch -a 已经存在的文件名 ： 只更新访问时间

3. touch -m 已经存在的文件名 ： 只更新修改的时间

###### 9.强制make每次都进行编译（可以短时间内连续的使用）

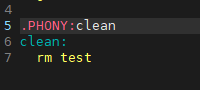
1.还是在makefile文件中进行修改

2. 使用 .PHONY: 对后面的文件进行修改



（1）后面的文件叫做 伪目标，make总是会执行

（2）但是不建议对源文件进行这样的修饰，一般情况下是对clean进行修饰



###### 10.makefile中方法的简写

1.对makefile中的方法进行书写的时候，有一种方便的写法就是 $@ $^



（1）$@ : 代表冒号左边的可执行文件

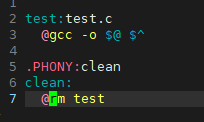
（2）$^ :代表冒号右边的源文件

###### 11.取消方法回显

1.在执行make的时候，会将方法显示出来



（1）在执行这个命令的时候，会将方法显示出来，在方法的前面加上@符号，再去使用命令的时候就不会显示出来了



//此时就不会再显示出来了

##### 二、制作一个Linux小程序 进度条

###### 1.回车换行

1.回车和换行是两个概念。

（1）在显示器上面，换行是光标跳转到下一行

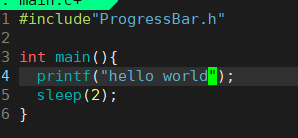
（2）而回车是光标在换到下一行后处于最前面的位置

（3）在写程序的时候，\n 默认为换行回车

2.只回车的符号是 \r

###### 2.缓冲区

1.为什么在屏幕上执行完这个程序后的现象是先休眠两秒钟，然后才开始打印helloworld

 //但是打印出来的结果是这样的

2.不用质疑的是，程序肯定是先执行的输出语句，再执行的sleep。但是为什么在现象上却显示的是先睡眠两秒再去输出。

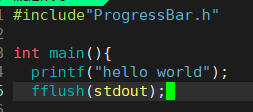
（1）这个期间，hello world 就在缓冲区中。就在缓冲区中被保存起来了 ----这个缓冲区就是c语言提供的

（2）这个所谓的缓冲区就是c语言维护的一段内存

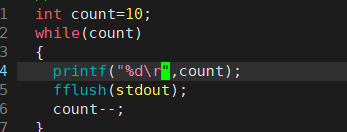
（3）

3.强制刷新缓冲区中的内容（将里面的数据刷新出去）

（1）使用函数 fflush 。输入输出缓冲区详细见c语言中的笔记

  
（2）在编写程序的时候，默认会打开三个标准文件 标准输入，标准输出（显示器，stdout），标准错误

4.先写出来倒计时（在同一行的同一个位置不断变换数字）



（1）注意好 \r 是只回车不换行，一直在这一行的最前方进行计数

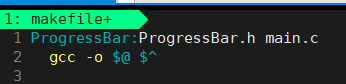
（2）但是这样在显示器上输出的是 10 90 80 70.....这样是因为10输出的是两个字符，而后面的数字覆盖的是第一个字符

（3）此时可以用 %2d 来进行控制，也就是只输出两个字符的位置

（4）并且还可以继续用 %-2d 来进行控制，也就是左对齐的方式输出，显示的数字都紧贴着最左方

###### 3.开始编写项目ProgressBar

1.首先makefile文件中的依赖关系还能有多个



（1）并且 $^ 可以指代冒号后面的两个源文件