Makefile使用

**作者:向仔州**

Makefile变量使用.......................................................................................................................2

环境变量使用..............................................................................................................................3

通配符使用..................................................................................................................................4

Makefile里面嵌套shell脚本语言..............................................................................................4

Ifeq else 的用法，类似(C语言的if …else..)，ifneq，ifdef.....................................................5

上级目录Makefile编译下级多个目录的Makefile....................................................................7

findstring用法：查找变量字符串，返回一个状态...................................................................8

MAKEFLAGS系统提前定义好的环境变量使用..........................................................................9

wildcard关键字使用...................................................................................................................10

notdir关键字使用………………………………………………………………………………………………………………....10

origin使用：告诉变量是从什么位置来....................................................................................11

VPATH(大写)指定源文件目录来编译……………………………………………………………………………………..11

vpath关键字使用,指定编译源码的目录...................................................................................13

include使用…………………………………………………………………………………………………………………………...13

VPATH,include,wildcard,notdir在Makefile项目工程当中的配合应用.....................................15

subst字符串替换........................................................................................................................18

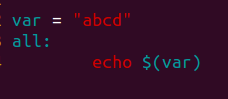
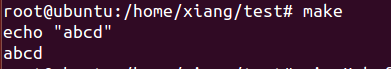
Makefile 编译下执行shell 比如@$(MKCONFIG) $(@:\_config=)…………………………………………..18

2.我们试试Makefile变量值替换规则 : 冒号……………………………………………………………………...18

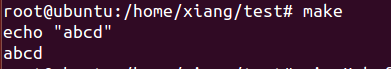
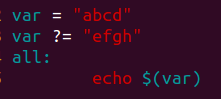
addprefix 增加字符的前缀…………………………………………………………………………………………………….19

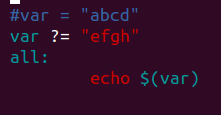
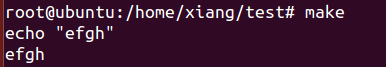
**Makefile变量使用**

**Makefile变量定义和shell一样**

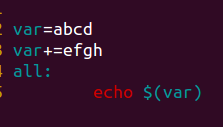
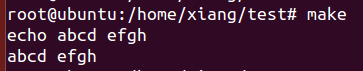
**?= 是同一个变量前面赋值了，后面就不能再赋值了**



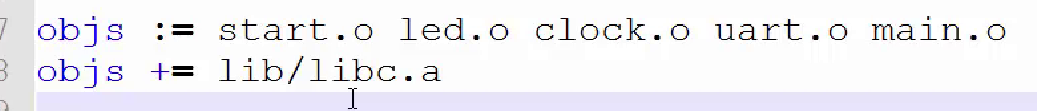
 

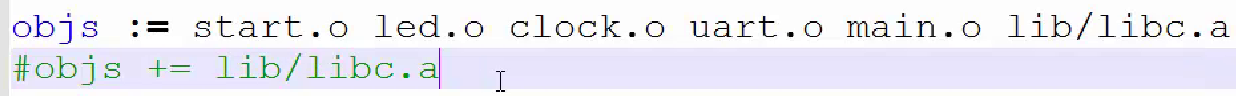
**这就是前面没有给变量赋值**

**+= 就是变量var被赋值之后，我突然又想给var加点参数，但是前面给var赋值的参数不能被覆盖，我就用+=**

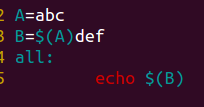
 

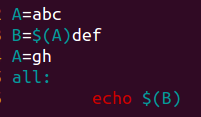
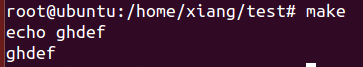
+=这种就是我什么时候需要什么时候写。



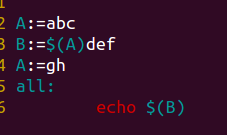
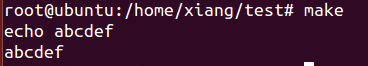


这就是+=的好处

**我们发现是最后一次给A变量赋值算作数**

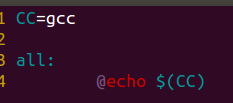
**用:= A变量就是第一次赋值作数，后面再给A赋值不作数**

所以如果变量是重复赋值的话， = := 要注意使用

**环境变量使用**

Makefile里定义的环境变量可以给工程中其它Makefile使用，用export来导出环境变量

局部变量是在自己的Makefile使用，所以环境变量最好用大写，而且要小心不能和工程中其他Makefile里面的环境变量重复。

**Makefile可以从外部传入参数**

**@这个符号是要求语句执行的时候不打印到屏幕上影响眼睛，但是语句还是在执行**



这就是编译的时候CC接受参数，然后CC变量就是echo $(arm-linux)，这也是环境变量的一种形式

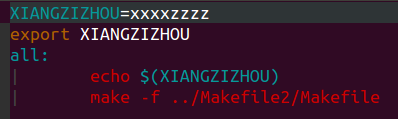
export导出的自定义环境变量只能在Makefile的指定的子Makefile中才生效。



**Makefile2为子Makefile**

**Makefile1为主Makefile**

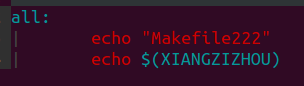




**在执行完主Makefile1的程序后必须在最后一句用make –f指定子Makefile2执行makefile**

**主Makefile导出的环境变量，必须由主Makefile指定的子Makefile才能使用**





**这样Makefile2的Makefile程序才能获得export导出的环境变量值**

所以export必须是两个Makefile有父子关系才有用

**通配符使用**

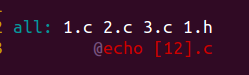
我创建了这多个文件



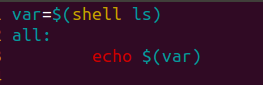
**\*表示当前目录下所有都是同一个后缀的文件放进来展开**

 你看h文件就不行。

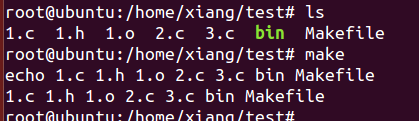
**[]中括号,就是选择指定几个.c文件的放进来展开，不像\*号选择所有.c文件**

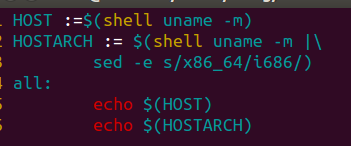
 

**Makefile里面嵌套shell脚本语言**

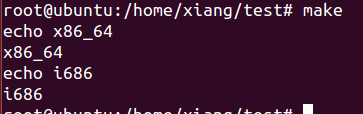


**Makefile执行shell语句,一定要将shell语句用$()括起来**



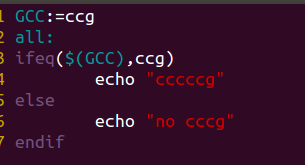


**\这个是接续符号，如果你一行命令写不完，就用这个接续符号换行继续写命令**



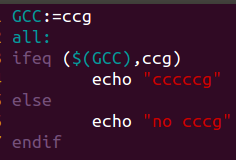
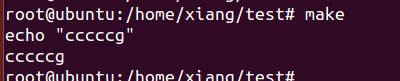
这就是Makefile的shell实际应用

**Ifeq else 的用法，类似(C语言的if …else..)，ifneq，ifdef**



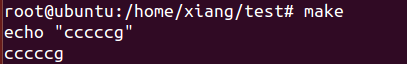
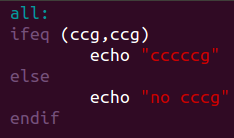
Ifeq(值1，值2)，如果值1等于值2就执行ifeq下的语句，否则执行else下的语句

为什么make出错？

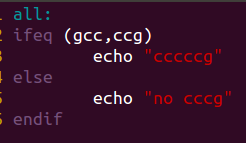
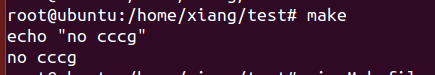
 

你看输出正确了

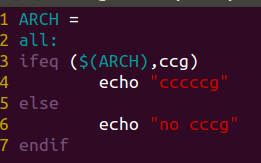
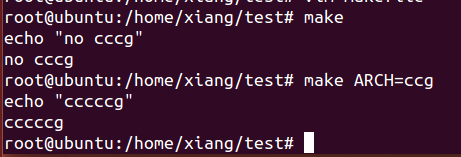
**这里要加空格，ifeq和括号之间用空格隔开，这就是makefile奇葩之处**



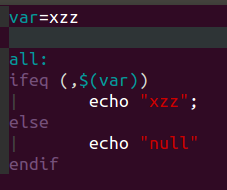
把变量展开就是这样的

把变量展开就是这样的

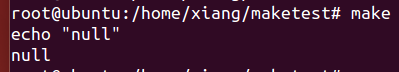
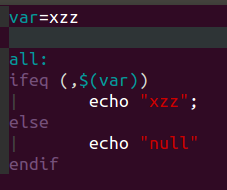
 

如果变量在Makefile里面不赋值，那么在make 执行后面要给变量赋值



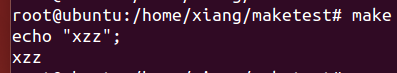
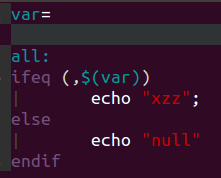
Ifeq ，前面没有变量就表示为空

比如我给var变量赋值了，那么var就不为空，就应该执行else



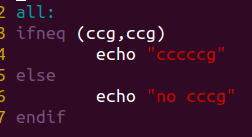
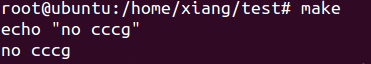
这里var什么值都没有就表示为空

执行ifeq下面的语句

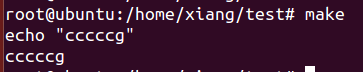
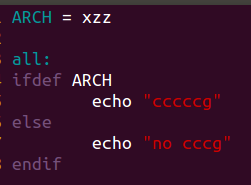


Ifeq的反作用是用ifneq

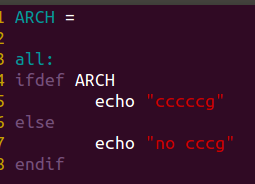
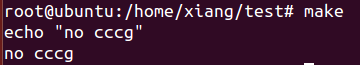
Ifneq就是两边相等就返回假，执行else下的代码，两边不相等为真

Ifdef，测量一个变量是否有值



如果ARCH变量得到xzz值，执行ifdef下的语句

如果ARCH变量没有值，执行else下的语句

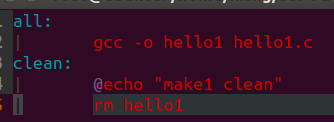
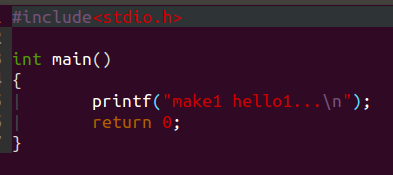
**上级目录Makefile编译下级多个目录的Makefile**



顶层目录有两个子目录make1，make2

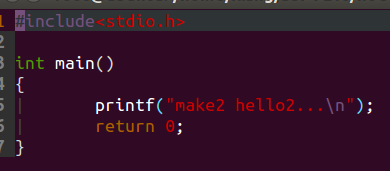
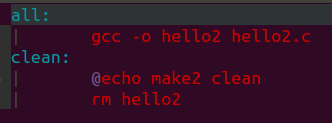


make1目录有c文件和Makefile文件





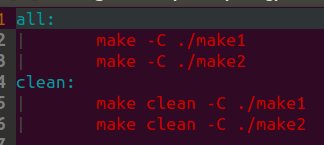
make2目录有c文件和Makefile文件

顶层的Makefile就是控制make1，make2目录里面的makefile文件执行的



make –C 指定目录，就是执行指定目录里面的Makefile文件

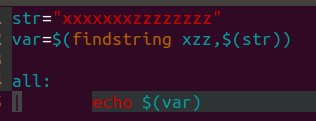


make clean –C 指定目录，就是执行指定目录Makefile文件里面的clean命令

这里指定的目录是make1，make2 。./就是当前目录下

的make1，make2目录

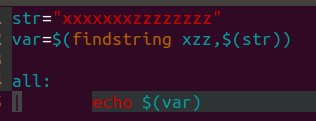
**findstring用法：查找变量字符串，返回一个状态**

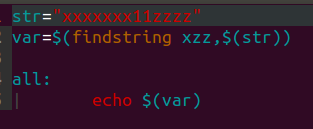
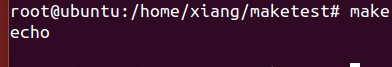


如果str变量里面有xzz字符串，findstring就返回xzz，这个返回的xzz字符就是findstring定义的

要查找的字符串xzz

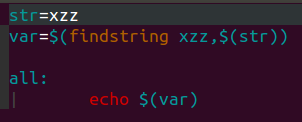
传入变量，看变量里面有没有xzz字符串

  你看返回字符串

你看str没有和findstring定义的相同字符串xzz，所以返回空

不用””字符串，直接给字符值也可以

**MAKEFLAGS系统提前定义好的环境变量使用**

MAKEFLAGS和我们自定义的环境变量不一样，MAKEFLAGS是系统定义好的，你只能使用，所以你在自己定义环境变量名字的时候不要和MAKEFLAGS这些名字冲突。

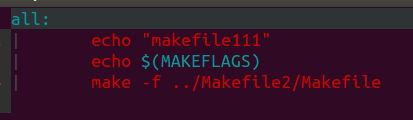
需要注意的是，有两个变量，一个是 SHELL，一个是 MAKEFLAGS，这两个变量不管你是否export，其总是要传递到子 Makefile 中

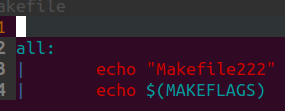
我make –s，那么s就会传递给MAKEFLAGS变量

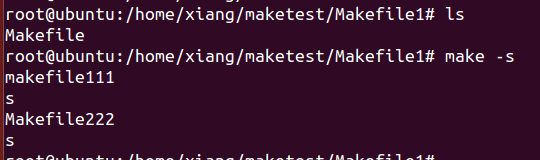
 

还有make –k 也可以将k传递进MAKEFLAGS

比如我一个主Makefile1和一个子Makefile2

主Makefile1

子Makefile2

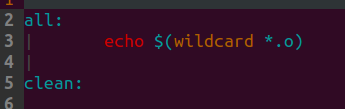


我在主makefile执行make –s，那么s就会传给主makefile和子makefile

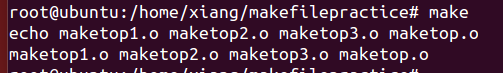
**wildcard关键字使用**



当前目录下有这些文件，我想把.o为后缀的文件用变量接受进来

Makefile用wildcard 来搜索

wildcard格式: wildcard \*. 文件后缀



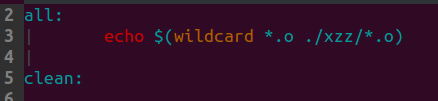
将我要的.o文件放入变量了，其他目录文件和不是.o的文件都将忽略掉。

wildcard搜索指定目录下的.o文件

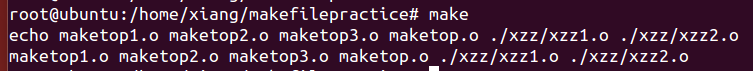


xzz目录下有两个.o文件，但是我的Makefile在上一级目录，看我如何用wildcard指定目录的

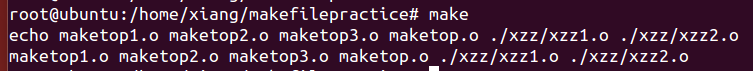
表示xzz目录下所有.o文件



wildcard指定目录格式: wildcard \*.文件后缀 指定搜索文件.o所在的目录

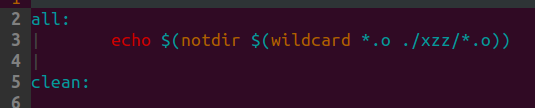


我们发现除了把xzz目录下的.o文件放入变量，当前Makefile所在目录下的.o文件也放入了变量。所以wildcard不管你去搜索哪个目录的文件，当前目录下的文件也会搜索一遍。

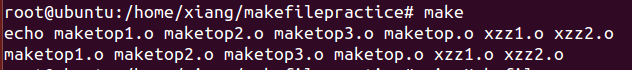


我想把xzz目录路径去掉，只保留xzz目录下的.o文件名放入变量

**notdir关键字使用**



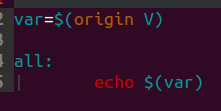
notdir格式: notdir 变量名



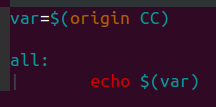
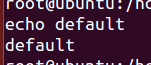
将变量名里面包含的路径字符去掉，比如./xzz/xzz1.o，就是将./xzz去掉，只保留xzz1.o字符串给变量

**origin使用：告诉变量是从什么位置来**

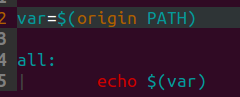
Origin 判断V变量从哪里来，但是因为环境变量和Makefile局部都没有定义V变量，所以返回undefined

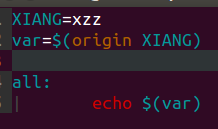
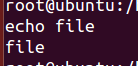
CC为系统自己提供的默认变量，所以返回default

PATH是系统自带的环境变量，可以用在任何Makefile，不像export这种导出的变量只能用在子Makefile

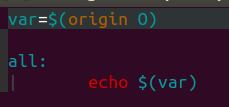
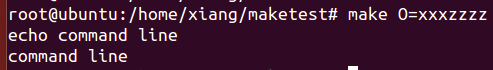
 

所以环境变量返回environment

自己在Makefile定义的局部变量返回file

在make时候跟着的变量O传入进来，就返回command line

如果make没有传入O变量，就返回undefined

**VPATH(大写)指定源文件目录来编译**

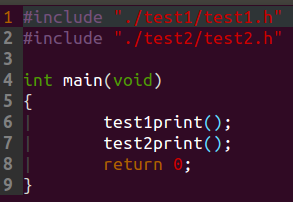
多文件原始编译方法



test1目录有c文件和h文件 test2目录有c文件和h文件

然后main.c文件会去调用test1.c和test2.c文件里面的子函数



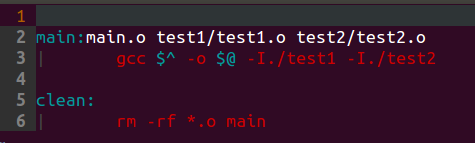
在main.c文件里面还必须指定你头文件的绝对路径

这是main文件要调用的函数

这时候Makefile怎么写呢？

1.编译出执行文件main

2.加入编译出执行文件的主文件main.o也就是等用于main.c

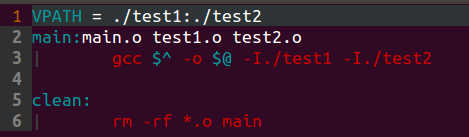
这就是makefile原始写法

4.而且编译的时候还要加入-I 头文件绝对路径

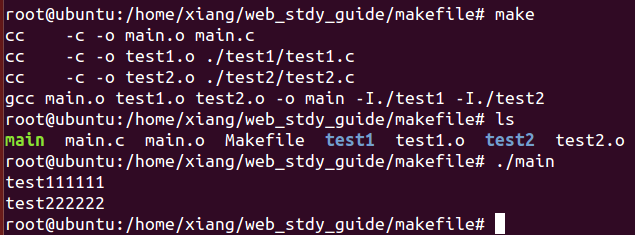
3.还要加入main.c调用子函数test1.c和test2.c的绝对路径

这种情况下假如test1,test2下面还有很多很多目录，我是不是要在test1/…..test1.c依赖里面加很多路径呢？岂不是手都要敲打疼

VPATH关键字指定C文件的目录路径，冒号是如果有多个目录，用冒号分割

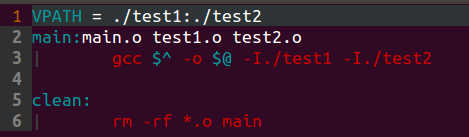


这里头文件-I路径还是要加上，因为VPATH关键字是告诉Makefile 源码路径。而不是告诉GCC源码路径,所以GCC要加上-I



编译成功

这些子文件.o怎么会在顶层当前目录呢？这是因为我的Makefile依赖文件没有指定目录

没有指定.o生成的目录路径。

感觉变化不大啊，只不过把目录路径给了VPATH变量了而已。如果有多级目录岂不是还要手动在VPATH后面加目录路径?

确实VPATH只能做到这个程度

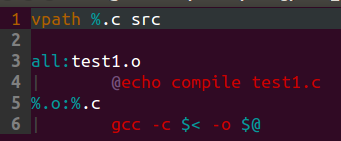
**vpath关键字使用,指定编译源码的目录**



现在我将自己写的多个c文件放在src目录下

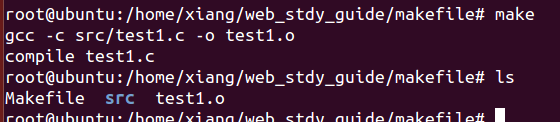


现在我想编译src目录下的每个c文件，第一种方法是我在src目录下写个Makefile然后编译，第二种方法是我在src上级目录写个makefile用vpath关键字来包含编译



我的src目录有两个文件，你要编译哪一个文件，你在all后面指定就行

用vpath关键字指定你要编译的源文件目录,%.c意思是只编译src目录的.c文件



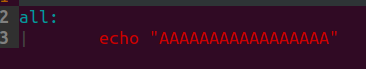
test1文件编译出来了,编译test2就要去Makefile里面把test1.o改成test2.o

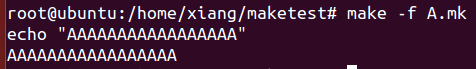
但是这有什么用?我直接去src目录里面去用Makefile编译不是一样的效率吗？

**include使用**

include主要用来包含 .mk 的Makefile文件

我们先看看.mk怎么编译



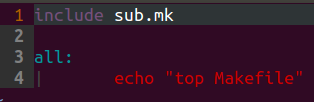
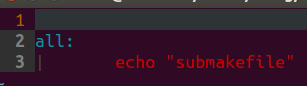
make 用-f来指定.mk文件才能把.mk文件当作Makefile编译

下面我创建一个主Makefile和一个子Makefile(子Makefile都是文件名.mk)

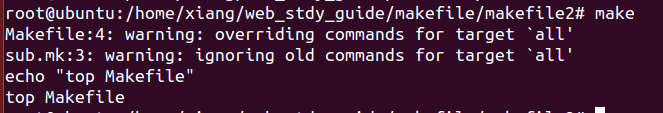


子makefile文件

主makefile文件

主Makefile文件的内容 子sub.mk文件的内容



编译的时候主Makefile确实按照要求去先执行了include指定的mk文件

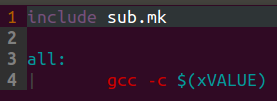
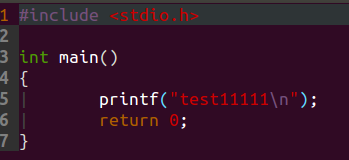
但是多个Makefile文件相互调用的情况下只能有一个all标志，所以Makefile程序发现sub.mk里面的all是不合法的，但是没有退出顶层Makefile，而是忽略include的mk文件继续向下执行，所以主Makefile文件的后面内容成功执行

如果是这样的话,那么我写子mk文件有什么意义呢？

子mk里面可以定义很多变量，然后主Makefile用include包含子.mk文件的时候，会将子.mk文件里面的内容在主Makefile里面展开，这样主Makefile就可以使用子.mk文件里面的变量了。



我建立了主Makefile,子Makefile文件sub.mk，和一个c文件

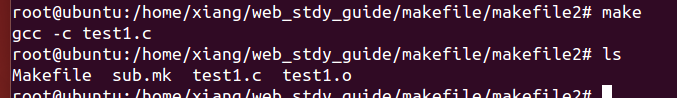
主Makefile文件内容 sub.mk文件内容 test1.c文件

现在我将test1.c文件编译成.o文件:

1.首先在sub.mk文件里面声明了变量来代表test1.c字符

2.主Makefile用include关键字将sub.mk里面的所有变量获取出来。

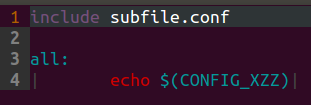
3.然后主Makefile就可以使用sub.mk里面的变量了。



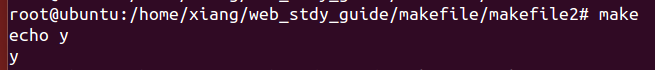
编译出了.o文件

而且include不止可以包含.mk格式的文件，还可以包含.conf格式的文件



主Makefile文件内容 subfile.conf文件内容



Include可以把.conf文件的变量包含进主Makefile，供其主Makefile使用

如果subfile是普通文件，include还能包含吗？



把subfile.conf修改为普通文件subfile



主Makefile的include还是可以包含，而且还是承认CONFIG\_XZZ为一个变量。

以上就是include关键字的功能，让一些不经常变化或者固定的变量，写在配置文件中，然后主Makefile需要用这些变量的时候，就用include将这些配置文件包含进来。

**VPATH,include,wildcard,notdir在Makefile项目工程当中的配合应用**



debug目录里面是A工程师设计的c程序，现在我要使用

test目录里面是B工程师设计的c程序，现在我要使用

app目录是我的程序文件，我要调用A工程师和B工程师里面的程序



test目录里面的程序Makefile是B工程师写好了的

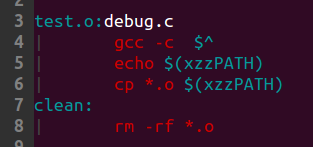
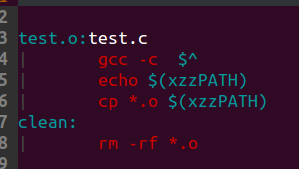


debug目录里面的程序Makefile是A工程师写好了的

现在我们在顶层Makefile目录来建立两个文件，一个是srco文件用来存放AB工程师Makefile编译出来的.o文件，因为我自己写的app主程序会去调用debug.c和test.c里面的函数，也就是间接依靠debug.o和test.o文件去调用的debug.c和test.c，所以debug.c和test.c要比app.c文件提前编译好。



inc目录用来存放A,B工程师debug,test程序的头文件，这样我写顶层Makefile的时候就去填写inc路经就可以了，不用去A,B工程师这么深的目录去找头文件路径，而且主程序app.c文件头文件包含也就只包含inc路经就可以了，不用去A,B工程师这么深的目录去包含头文件路径。

B工程师编译完成后，直接用cp把.o文件复制到xzzPATH指定的顶层srco目录

xzzPATH是顶层Makefile定义的路径变量

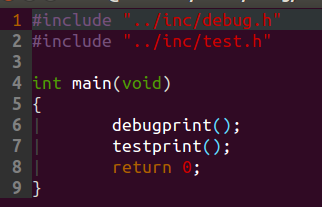
A工程师编译完成后，直接用cp把.o文件复制到xzzPATH指定的顶层srco目录

xzzPATH是顶层Makefile定义的路径变量

A工程师debug程序的Makefile B工程师test程序的Makefile

然后A工程师和B工程师把debug.h和test.h文件复制到顶层的inc目录

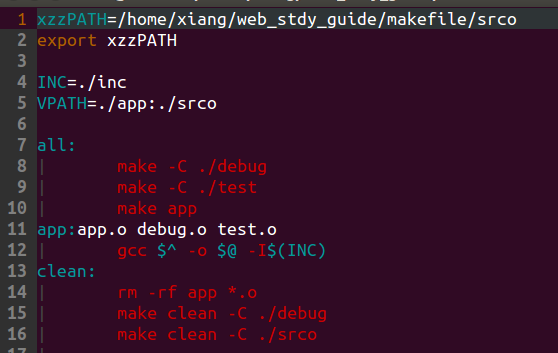
下面我用app.c主函数去调用debug和test程序



调用A,B工程师实现的程序

主函数调用debug和test的程序，那么我们只需要包含顶层inc目录下的头文件就是了，因为这两个头文件是AB工程复制到inc目录下的

下面是顶层Makefile



你的app.o程序调用了哪些c文件程序就要把那些c程序的.o文件依赖进来

因为Makefile只能有一个目标，所以如果先执行了all就不会执行下面的app，所以这里在all里面指定接着执行app

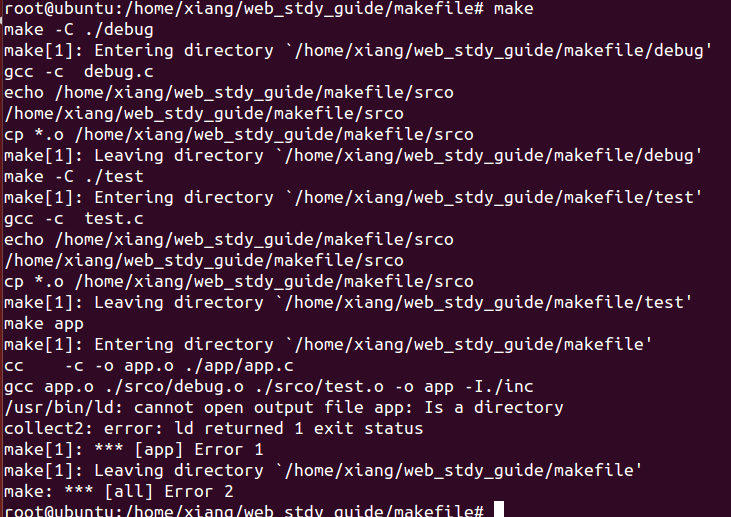
执行make –C 先把AB工程师的程序编译成.o了来

指定主文件路径和需要调用的C文件变成.o文件的路径

GCC编译需要头文件路径，AB工程师的程序头文件都要求放在inc目录下，这里指定头文件路径

这就是传给A,B工程师存放.o文件的路径

导出环境变量AB工程师的Makefile才可以用

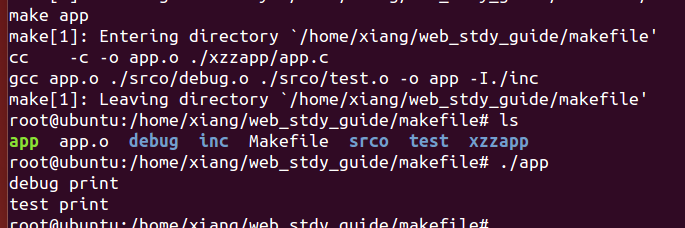
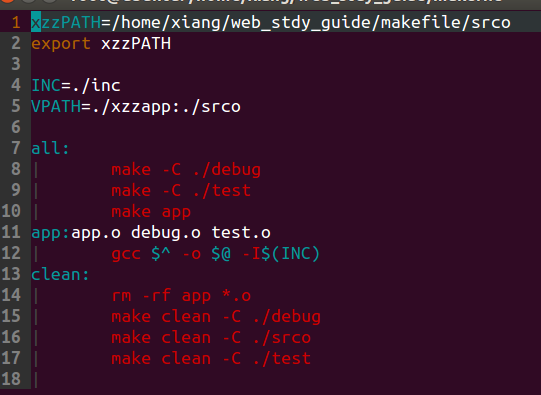
这个错误不是Makefile语法问题

编译的时候出现了ld returned 1 exit status错误



这个问题是因为你生成的二进制文件名是app，但是在二进制app文件目录下有个名叫app的目录，所以冲突了。



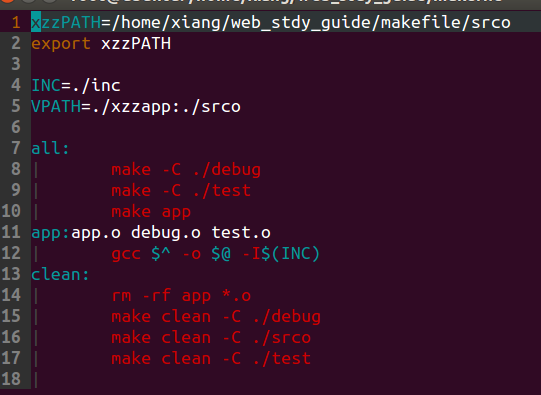


Makefile再修改下

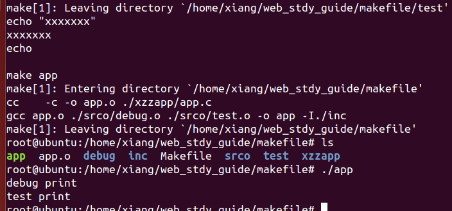
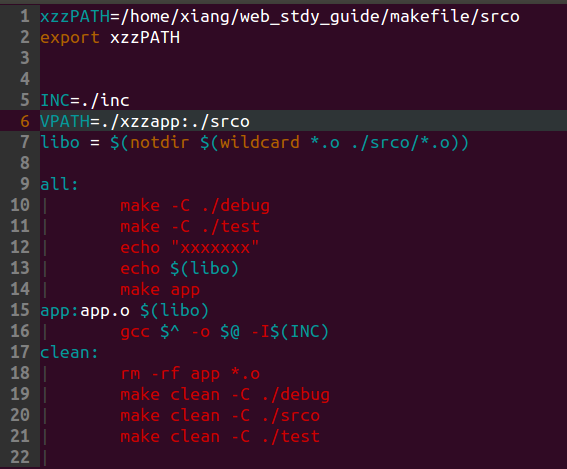
把app.c文件的目录名改成xzzapp

再次make，成功编译，成功执行。

我们下面来优化Makefile文件



我现在不想每次调用一个头文件，就要去srco目录里面去找对应的.o文件写入依赖中，这样调用的文件多了，我这里要写一堆字符，好烦

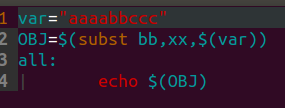


将libo放入依赖，展开就是app.o debug.o test.o 这样就可以编译出执行程序了

用wildcard关键字去指定目录提取所以.o文件的字符串给变量，然后用notdir将.o文件路径去掉，这样libo变量就=debug.o,test.o了

这样AB工程师想srco目录放库文件，我主程序调用库文件就是了。

**subst字符串替换**



像subst这种命令是不放在all目标下执行的，因为不属于编译命令，所以放在任何地方都可以执行，就是不能放在TAB下

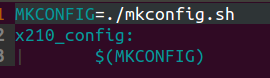
把变量var的bb字符串替换成xx字符串

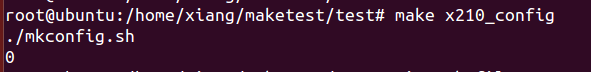


**Makefile 编译下执行shell** 比如@$(MKCONFIG) $(@:\_config=)

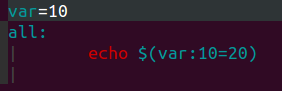
1.我们先试试Makefile的变量调用shell

目录下有两个文件

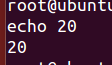
这是Makefile执行shell的方法



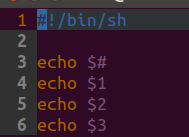
**2.我们试试Makefile变量值替换规则 : 冒号**

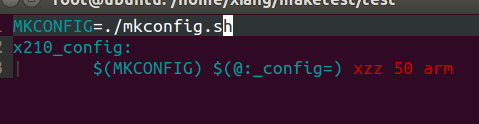


变量后边的 : 符号就是说将变量的10替换成20

输出正确

3.我们来试试@$(MKCONFIG) $(@:\_config=)替换目标字符，传参进shell

我们的mkconfig.sh只输出传入的参数变量

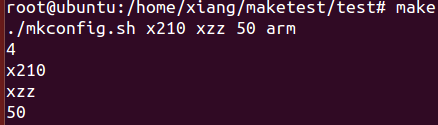


为什么是空，因为按照变量替换规则$(var:10=20)我这个=后面没有东西

这里的\_config将x210\_config的\_config字符串替换为空

$(@获取目标 : 左边的字符

替换完成之后就是$(MKCONFIG) x210 xzz 50 arm，这样了。将参数传入进shell

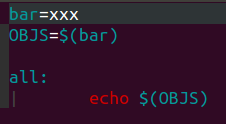
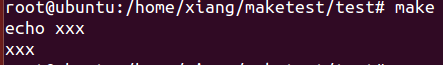


然后提取$1是x210

$2是xzz，以此类推

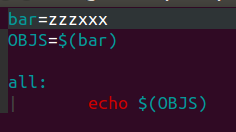
Shell要求$#就是传入shell字符个数，这里是4个参数

**addprefix 增加字符的前缀**

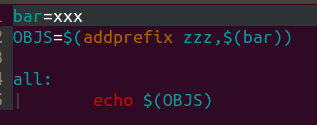
 

我输出最简单的字符

我现在想给字符前面在加点字符，

这样修改很普遍

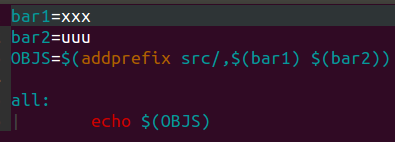
 

但是我想用addprefix来修改

addprefix 第一项写的zzz字符，就是加给逗号后面变量的字符串前缀

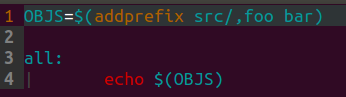
这样做有什么意义呢？

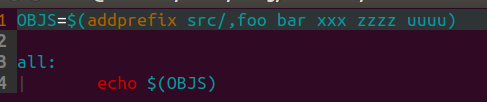
如果我们所有文件只有一个目录(比如src目录)，但是src目录里面的文件在不断的修改和增加，但是编译的时候每个文件都需要指定src目录，我是不是每个文件参数前面都去写src呢？这样好累



这就是每个文件字符串前面都加src，每个文件(字符串)用空格隔离开

我们也可以不用变量，直接写文件名





可以写多个文件名，每个文件名都加上src前缀字符

