**Makefile**

dir函数的用法：取目录函数

格式：$(dir <names…..>)

从文件名序列<names>中取出目录部分，目录部分是指最后一个反斜杠（“/”）之前的部分。如果没有反斜杠，就返回“./”。

实例：

$(dir src/foo.c hacks) //返回“src/ ./”

notdir函数的用法：

格式：$(notdir <names….>)

从文件名序列names中取出非目录的部分，目录部分是指最后一个“/”（包括“/”）之前的部分。删除所有文件名中的目录部分，只保留所有的文件名中非目录的部分。如果“names”中有不包含“/”的文件名，则不改变这个文件名。

实例：

$(notdir src/foo.c hacks) //返回值为：“foo.c hacks”

patsubst函数的用法：模式替换函数

格式：$(patsubst <pattern>, <replacement>,<text>)

查找text中的单词（单词以“空格”、“Tab”、或者“回车”“换行”分割）是否符合模式<pattern>，如果匹配的话，则以<replacement>替换。其中，<pattern>可以包括通配符“%”，表示任意长度的字符串。如果<replacement>中也包含“%”，那么<replacement>中的“%”僵尸<pattern>中的那个“%”所代表的字符串。（可以使用“\”来转移，以“%”来表示真实含义的“%”字符）。函数返回被替换后的字符串。

实例：

$(patsubst %.c,%.o,x.c.c bar.c) //把字符串“x.c.c bar.c”符合模式[%.c]的单词替换为[%.o]，返回结果是“x.c.o bar.o”

wildcard函数的用法：

在Makefile中，通配符会被自动展开。但是在变量的定义和函数引用时，通配符将失效，这种情况下就要使用函数“wildcard”。

格式：$(wildcard PATTERN…..)

在Makefile中，他被展开为已经存在的、使用空格分开的、匹配此模式的所有文件列表。如果不存在任何符合此模式的文件，函数会忽略模式字符并返回空。

实例：

一般我们可以用“$(wildcard \*.c)”来获取工作目录下的所有.c文件列表。复杂一些用法：可以使用“$(patsubst %.c,%.o,$(wildcard.c))”，首先使用“wildcard”函数来获取工作目录下的.c文件列表；之后将列表中的所有文件名的后缀替换为.o。这样我们就可以得到当前目录可生成.o文件列表

filter函数的用法：

格式：$(filter <pattern…>, <text>)

以<pattern>模式过滤<text>字符串中的单词，保留符合模式<pattern>的单词。可以有多个模式。返回符合模式<pattern>的字符串。

实例：

sources := foo.c bar.c baz.s ugh.h

foo:$(sources)

cc $(filter %.c %.s,$(sources)) -o $@ //$(filter %.c %.s,$(sources))返回“foo.c bar.c baz.s”

filter-out函数的用法：

格式：$(filter-out,<pattern…>,<text>)

过滤掉字符串“text”中所有符合模式“pattern”的单词，保留所有不符合此模式的单词。可以有多个模式。存在多个模式时，模式表达式之间使用空格分割。

实例：

objects = main1.o foo.o main2.o bar.o

mains = main1.o main2.o

$(filter-out $(mains),$objects)//返回“foo.o bar.o”

strip函数的用法：

格式：$(strip STRINT)

去掉字符串（若干单词，使用若干空字符分割）“STRINT”开头和结尾的空字符，并将其中多个连续的空字符合并为一个空字符。返回无前导和结尾空字符、使用单一空格分隔的多单词字符串。

实例：

STR = a b c

LOSTR = $(strip $(STR)) //返回结果是“a b c”

变量替换引用：

对于一个已经定义的变量，可以使用“替换引用”将其值中的后缀字符（串）使用指定的字符（字符串）替换。格式“$(VAR:A=B)”（或者“${VAR:A=B}”），意思是，替换变量“VAR”中所有“A”字符结尾的字为“B”结尾的字。“结尾”的含义是空格之前（变量值多个字之间使用空格分开）。而对于变量其他部分的“A”字符不进行替换。

实例：

foo := a.o b.o c.o

bar := $(foo:.o=.c)

在这个定义中，变量“bar”的值就为“a.c b.c c.c”。使用变量的替换引用将变量“foo”以空格分开的值中的所有的字的尾字符“o”替换为“c”，其他部分不变。如果在变量“foo”中存在“o.o”时，那么变量“bar”的值为“a.c b.c c.c o.c”，而不是“a.c b.c c.c c.c”。

for、和foreach循环：

for循环属于shell命令，必须在同一行中写完整个for语句，在其中引用变量时，需要使用$$，并且变量一定不可以用括号括起来。（网上有说需要用括号括起来，但是实际使用时，括起来是达不到想要的目的，或许与make版本有关，我使用的是GNU Make 4.1）

实例：此Makefile一次实现编译当前目录下所有的.c文件。完成这个任务可以直接在TEST目标后面将$(TARGETS)这些依赖写上，使用默认规则就可以完成这个工作，不需要for、foreach循环这些执行语句即可。只是为了学习for和foreach。

CC = gcc

FILES := $(wildcard \*.c)

TARGETS := $(filter-out current,$(patsubst %.c,%,$(FILES)))

.PHONY: TEST

TEST:

for i in $(TARGETS);do \

$(CC) -o $$i $$i.c; \

done

.PHONY: clean

clean:

rm -rf modules.order $(TARGETS)

foreach循环是makefile的内置函数，语法格式：$(foreach i,$(list),<tex>[;<tex>])

其中foreach前面的$不可以去掉。

实例：

CC = gcc

FILES := $(wildcard \*.c)

TARGETS := $(filter-out current,$(patsubst %.c,%,$(FILES)))

.PHONY: TEST

TEST:

@$(foreach i,$(TARGETS),$(shell $(CC) -o $i $i.c))

module:

make -C $(KERNEL) M=`pwd` modules

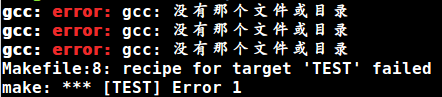
.PHONY: clean

clean:

rm -rf $(TARGETS)

在这个foreach循环中，@$(foreach i,$(TARGETS),$(shell $(CC) -o $i $i.c))，后面的执行语句如果不加shell将会报错。比如：

$(foreach i,$(TARGETS), $(CC) -o $i $i.c)



似乎gcc接收的参数都是gcc，不知道为什么会是这样。