跟我一起写 Makefile

作者：陈皓

整理：祝冬华

Makefile的规则:

target … : prerequisites … ;command

command

command

target是目标集合

prerequisites是生成target所依赖的文件

command是此条规则若需要执行时的执行shell命令,若需要执行是指prerequisites中有一个以上的文件比targe文件新的话,command就会被执行.

make的工作过程

1:读入Makefile文件,读入include的其他makefile文件

2:初始化文件中的变量

3:推导隐晦规则,并分析所有规则

4:为所有的目标文件创建依赖关系链

5:根据依赖关系,决定哪些目标要重新生成

6:执行生成命令

备注:

make会在当前的目录下找Makefile或者makefile的文件,若指定则用-f参数

若找到,则文件中的第一个规则的第一个目标就是最终的目标文件

找寻依赖文件过程中,若出现依赖文件找不到,则make直接退出,并报错,若command执行出错,make不会理会

Makefile文件中包含什么

1:显示规则

2:隐晦规则

3变量定义

4:文件指示

5:注释

**注释:**

1:

Makefile中只有行注释,用#字符

2:

make规则中的command执行时,

**变量的定义**

1:

Makefile中的变量一般都是字符串,可理解为C中的宏,当makefile被执行时,其中的变量会被扩展到相应的引用位置上,与宏区别之处是可以在makefile中改变其值

定义变量object=main.o kdb.o

引用变量edit:$(object)

2:

变量命名规则:可包含数字,字符,下划线,不可包含”: # =”或空字符,大小写敏感

3:

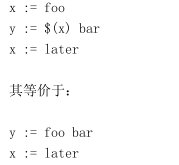
变量可用在目标,依赖,命令以及新的变量中

4:

变量在声明时需要给予初值,使用时需要在变量名前加$,最好用小括号或大括号把变量包括起来,若要使用$字符,用$$表示

5:变量中的变量

变量简单的使用是使用=号,=左侧是变量名,右侧是变量的值,右侧的变量的值可以定义在文件的任何一处,而且不一定非要是已定义好的值,也可以是后面定义的值,这种方法有好处也有坏处,未避免这种情况的发送,可以使用:=操作符,如:



另外?=操作符表示,如果变量没定义过,则定义之,如果定义过,则忽略

6:变量的高级用法

第一种:变量值的替换

$(var:a=b)或${var:a=b}或$(var:%.a=%.b)

把变量var中的所有以a字串结尾的a替换为b字串

如:foo : =a.o b.o c.o

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

则bar变量的值是a.c b.c c.c

第二种:变量的值再当成变量

x = y

y = z

a := $($(x))

a的值是z

7:追加变量值用+=

objects = main.o foo.o bar.o utils.o

objects += another.o

我们的$(objects)值变成：“main.o foo.o bar.o utils.o another.o”

8:

如果有变量是通过make的命令行设置的,则makefile中对这个变量的赋值会被忽略,如果想在makefile中设置这类参数的值,那么得使用override指示符,语法为

override <variable>=<variable>

override <variable>+=<variable>

override <variable>:=<variable>

默认情况下，只有通过命令行设置的变量会被传递。而定义在文件中的变量，如果要向下层 Makefile 传递，则需要使用exprot 关键字来声明。

9:目标变量(局部变量)

相对前面所说的全局变量,可以定义针对某个规则的目标文件

<target …>:<variable>

如:

prog : CFLAGS = -g

prog : prog.o foo.o bar.o

$(CC) $(CFLAGS) prog.o foo.o bar.o

prog.o : prog.c

$(CC) $(CFLAGS) prog.c

foo.o : foo.c

$(CC) $(CFLAGS) foo.c

bar.o : bar.c

$(CC) $(CFLAGS) bar.c

在这个示例中，不管全局的$(CFLAGS)的值是什么，在 prog 目标，以及其所引发的所有

规则中（prog.o foo.o bar.o 的规则），$(CFLAGS)的值都是“-g”

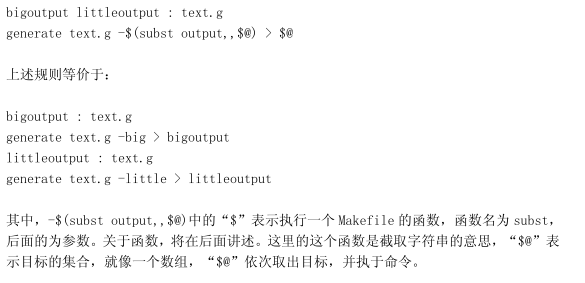
还有模式变量,即将变量定义到符合某个模式目标上

<pattern …>:<variable>

如%.o:CFLAGS=-O

**显示规则**

**1:多目标**



**2:静态模式:**

<targets …>:<target-pattern>:<prereq-patterns …>

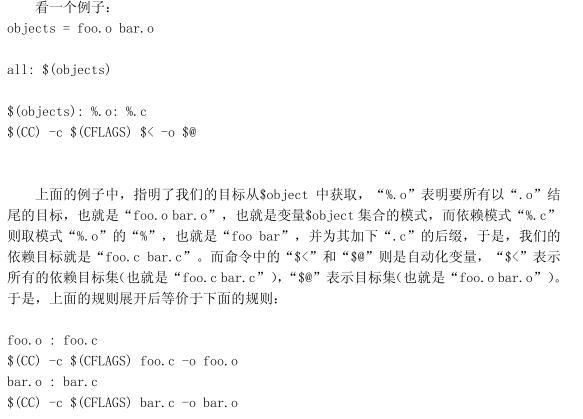
<command>

…

targets是一系列文件的集合,可用通配符

target-parrtern指明了目标集的模式,

prereq-parrterns是目标的依赖文件集模式



**3:嵌套执行make**

总控makefile中会调用子目录的makefile,类似如下

subsystem:

cd subdir && $(MAKE)

总控makefile中的变量可以传递到下级makefile中(需要显示声明),但是不会覆盖下级makefile中定义的变量,除非指定-e参数

如果想传递变量到下级makefile中,你可以使用如下声明

export <variable …>

如果你不想让某些变量传递到下级makefile中,可以使用如下声明

unexport <variable …>

如果要传递所有的变量,只需要一个export就可以了

需要注意的是

1:有两个变量,SHELL,MAKEFLAGS,这两个变量不管export与否,总是会传递给下次makefile中的

2:make命令中的几个参数不会往下传递,它们是 “-C –f –h –o -W

**4:函数的使用**

语法:$(<function> <arguments>)或${<function> <arguments>}

<function>就是函数名，make 支持的函数不多。<arguments>是函数的参数，参数间

以逗号“,”分隔，而函数名和参数之间以“空格”分隔。函数调用以“$”开头，以圆括号

或花括号把函数名和参数括起。函数中的参数可以使用变量

函数分字符串处理函数和文件名操作函数及其他几个函数

其他几个函数的讲解:

foreach

foreach 函数和别的函数非常的不一样。因为这个函数是用来做循环用的,Makefile中的 foreach 函数几乎是仿照于 Unix 标准 Shell中的 for 语句

它的语法是：

$(foreach <var>,<list>,<text>)

这个函数的意思是，把参数<list>中的单词逐一取出放到参数<var>所指定的变量中，

然后再执行<text>所包含的表达式。每一次<text>会返回一个字符串，循环过程中，<text>

的所返回的每个字符串会以空格分隔，最后当整个循环结束时，<text>所返回的每个字符串

所组成的整个字符串（以空格分隔）将会是 foreach 函数的返回值。

所以，<var>最好是一个变量名，<list>可以是一个表达式，而<text>中一般会使用<var>

这个参数来依次枚举<list>中的单词。举个例子：

names := a b c d

files := $(foreach n,$(names),$(n).o)

上面的例子中，$(name)中的单词会被挨个取出，并存到变量“n”中，“$(n).o”每次

根据“$(n)”计算出一个值，这些值以空格分隔，最后作为 foreach 函数的返回，所以，

$(files)的值是“a.o b.o c.o d.o”。

注意，foreach 中的<var>参数是一个临时的局部变量，foreach 函数执行完后，参数

<var>的变量将不在作用，其作用域只在 foreach 函数当中。

if函数

if 函数很像 GNU 的 make 所支持的条件语句——ifeq

$(if <condition>,<then-part>)

或是

$(if <condition>,<then-part>,<else-part>)

if 函数可以包含“else”部分，或是不含。即 if 函数的参数可以是两个，也可

以是三个。<condition>参数是 if 的表达式，如果其返回的为非空字符串，那么这个表达式

就相当于返回真，于是，<then-part>会被计算，否则<else-part>会被计算

而 if 函数的返回值是，如果<condition>为真（非空字符串），那个<then-part>会是整

个函数的返回值，如果<condition>为假（空字符串），那么<else-part>会是整个函数的返

回值，此时如果<else-part>没有被定义，那么，整个函数返回空字串。

**隐晦规则**

make可以自动从目标文件推导出其所依赖的文件和command

这里我们将讲述所有预先设置（也就是 make 内建）的隐含规则，如果我们不明确地写

下规则，那么，make 就会在这些规则中寻找所需要规则和命令。当然，我们也可以使用 make

的参数“-r”或“--no-builtin-rules”选项来取消所有的预设置的隐含规则。

还是先来看一看常用的隐含规则吧。

编译 C 程序的隐含规则

“<n>.o”的目标的依赖目标会自动推导为“<n>.c”，并且其生成命令是“$(CC) –c

$(CPPFLAGS) $(CFLAGS)”

定义模式规则

你可以使用模式规则来定义一个隐含规则。一个模式规则就好像一个一般的规则，只是

在规则中，目标的定义需要有"%"字符。"%"的意思是表示一个或多个任意字符。在依赖目标

中同样可以使用"%"，只是依赖目标中的"%"的取值，取决于其目标。

自动化变量

在上述的模式规则中，目标和依赖文件都是一系例的文件，那么我们如何书写一个命令

来完成从不同的依赖文件生成相应的目标？因为在每一次的对模式规则的解析时，都会是不

同的目标和依赖文件。

自动化变量就是完成这个功能的。

所谓自动化变量，就是这种变量会把模式中所定义的一系

列的文件自动地挨个取出，直至所有的符合模式的文件都取完了。

这种自动化变量只应出现在规则的**命令中**

$@

表示规则中的目标文件集。在模式规则中，如果有多个目标，那么，"$@"就是匹配于

目标中模式定义的集合。

$<

依赖目标中的第一个目标名字。如果依赖目标是以模式（即"%"）定义的，那么"$<"将

是符合模式的一系列的文件集。注意，其是一个一个取出来的。

一般来说，一个目标的模式有一个有前缀或是后缀的"%"，或是没有前后缀，直接就是

一个"%"。因为"%"代表一个或多个字符，所以在定义好了的模式中，我们把"%"所匹配的内

容叫做"茎"，例如"%.c"所匹配的文件"test.c"中"test"就是"茎"。因为在目标和依赖目标

中同时有"%"时，依赖目标的"茎"会传给目标，当做目标中的"茎"。

当一个模式匹配包含有斜杠（实际也不经常包含）的文件时，那么在进行模式匹配时，

目录部分会首先被移开，然后进行匹配，成功后，再把目录加回去。在进行"茎"的传递时，

我们需要知道这个步骤。例如有一个模式"e%t"，文件"src/eat"匹配于该模式，于是"src/a"

就是其"茎"，如果这个模式定义在依赖目标中，而被依赖于这个模式的目标中又有个模式

"c%r"，那么，目标就是"src/car"。（"茎"被传递）

**文件指示**

包含三个方面:

1:makefile文件包含另一个makefile文件,类似include

2:指定makefile中的有效部分,类似#if等

3:定义一个多行命令

1:

makefile中可用关键词include把别的Makefile包含进来,被包含的文件会原模原样放在当前文件包含的位置

语法:

include <filename>

filename可包含路径和通配符,在include前可有空字符,但不能以Tab键开始,include与filename之间可以用一个或多个空格隔开

filename若无路径,则首先会在当前目录查找,若无,会在make执行时所带-I参数指定的目录查找,继而回去系统文件查找,如找不到,则出现致命信息,如想忽略找不到的文件,请在include前加减号”-”

2:条件判断

可以使用条件判断让make不同情况选择不同执行分支,条件表达式可以比较变量的值,或比较变量和常量的值

语法:

<conditional-directive>

<text-if-true>

else

<text-if-false>

endif

其中<conditional-directive>表示条件关键字。这个关键字有四个

第一个是我们前面所见过的“ifeq”

ifeq (<arg1>, <arg2>)

ifeq '<arg1>' '<arg2>'

ifeq "<arg1>" "<arg2>"

ifeq "<arg1>" '<arg2>'

ifeq '<arg1>' "<arg2>"

第二个条件关键字是“ifneq”。语法是：

ifneq (<arg1>, <arg2>)

ifneq '<arg1>' '<arg2>'

ifneq "<arg1>" "<arg2>"

ifneq "<arg1>" '<arg2>'

ifneq '<arg1>' "<arg2>"

第三个条件关键字是“ifdef”。语法是：

ifdef <variable-name>

第四个条件关键字是“ifndef”。其语法是：

ifndef <variable-name>

特别注意的是，make 是在读取 Makefile 时就计算条件表达式的值，并根据条件表达式

的值来选择语句，所以，你最好不要把自动化变量（如“$@”等）放入条件表达式中，因为

自动化变量是在运行时才有的。