在makefile中，会经常使用shell命令，也经常见到$var 和 $$var的情况，有什么区别呢，区别大了。不要认为在makefile的规则的命令行中使用$var就是将makefile的变量和shell共享了，这里仅仅是读取makefile的变量然后扩展开，将其值作为参数传给了一个shell命令。而$$var是在访问一个shell命令内定义的变量，而非makefile的变量。此外，如果某规则有n个shell命令行构成，而相互之间没有用';'和'\'连接起来的话，就是相互之间没有关联的shell命令，相互之间也不能变量共享。看如下例子：

makefile代码段1：

```
VAR=3
target: prerequsite1 prerequsite2
        echo $VAR          (1)
        VAR=4                  (2)
        echo $VAR               (3)
        echo $$VAR               (4)
```

在代码段1中，(1)的结果是3，显然makefile利用自己的变量将$VAR扩展成3之后传递给这个echo这个shell命令。

(2)中，是一个独立的shell命令自己第一了一个shell变量，名字也叫VAR，且其值为4，不会影响到makefile中的VAR。

(3)中，同(1)，makefile中的变量VAR的值依然是3

(4)makefile将$$VAR先执行一次扩展得到如下shell命令：

echo $VAR然后交给shell去解释执行，可是对于这个shell命令来说VAR是一个为定义的变量，因此输出的结果就是个空行。

makefile代码段2：

```
VAR=3
target: prerequsite1 prerequsite2
        echo $VAR;\         (1')
        VAR=4;\                 (2')
        echo $VAR;\              (3')
```

echo $$VAR (4')

在代码段2中，所有的shell命令都被连接起来了，那么执行的结果就有变化了：

(1')结果同(1)，$VAR被替换成了3

(2')结果同(2)

(3')输出3，因为虽然shell中有VAR变量，可是makefile先要进行扩展，扩展的结果就是echo 3。

(4')输出4，因为makefile扩展结果为echo $VAR，而shell中已经有了变量VAR，且其值为4.

类似的例子还有如下：

makefile代码片断3：

```
SUBDIRS=tools examples src
target: prerequsite1 prerequsite2
     for dir in $SUBDIRS; do $(MAKE) -C $$dir;done
```

make首先将这个命令扩展成：

```
for dir in tools examples src; do make -C $dir; done
```

然后交给shell解释执行，可见dir就是一个shell中的变量。

总之，在makefile的shell命令中，党要引用shell变量的时候，要使用$$VAR格式。

此外注意一点，在makefile中，$$还被用来做SECONDEXPANSION,即二次扩展，一般是作为prerequsites.

如：

```
.SECONDEXPANSION （要使用这个feature就的加上这一行）
main_objs := main.o try.o test.o
lib_objs := lib.o api.o
main lib:     $$($$@_objs)
```

$$($$@_objs)第一阶段的扩展结果为$($@_objs)，第2阶段的扩展结果为:$@被替换为main lib，与_objs连接成main_objs lib_objs。加上外面$的扩展，就是$main_objs $lib_objs,最终结果为main.o try.o test.o lib.o api.o