**13、CALL**  
CALL命令可以在批处理执行过程中调用另一个批处理，当另一个批处理执行完后，再继续执行原来的批处理  
CALL command  
调用一条批处理命令，和直接执行命令效果一样，特殊情况下很有用，比如变量的多级嵌套，见教程后面。在批处理编程中，可以根据一定条件生成命令字符串，用call可以执行该字符串，见例子。  
CALL [drive:][path]filename [batch-parameters]  
调用的其它批处理程序。filename 参数必须具有 .bat 或 .cmd 扩展名。  
CALL :label arguments  
调用本文件内命令段，相当于子程序。被调用的命令段以标签:label开头  
以命令goto :eof结尾。  
另外，批脚本文本参数参照(%0、%1、等等)已如下改变:  
     批脚本里的 %\* 指出所有的参数(如 %1 %2 %3 %4 %5 ...)  
     批参数(%n)的替代已被增强。您可以使用以下语法:（看不明白的直接运行后面的例子）  
         %~1         - 删除引号(")，扩充 %1  
         %~f1        - 将 %1 扩充到一个完全合格的路径名  
         %~d1        - 仅将 %1 扩充到一个驱动器号  
         %~p1        - 仅将 %1 扩充到一个路径  
         %~n1        - 仅将 %1 扩充到一个文件名  
         %~x1        - 仅将 %1 扩充到一个文件扩展名  
         %~s1        - 扩充的路径指含有短名  
         %~a1        - 将 %1 扩充到文件属性  
         %~t1        - 将 %1 扩充到文件的日期/时间  
         %~z1        - 将 %1 扩充到文件的大小  
         %~$PATH : 1 - 查找列在 PATH 环境变量的目录，并将 %1  
                       扩充到找到的第一个完全合格的名称。如果环境  
                       变量名未被定义，或者没有找到文件，此组合键会  
                       扩充到空字符串  
    可以组合修定符来取得多重结果:  
        %~dp1       - 只将 %1 扩展到驱动器号和路径  
        %~nx1       - 只将 %1 扩展到文件名和扩展名  
        %~dp$PATH:1 - 在列在 PATH 环境变量中的目录里查找 %1，  
                       并扩展到找到的第一个文件的驱动器号和路径。  
        %~ftza1     - 将 %1 扩展到类似 DIR 的输出行。  
    在上面的例子中，%1 和 PATH 可以被其他有效数值替换。  
%~ 语法被一个有效参数号码终止。%~ 修定符不能跟 %\*使用  
注意：参数扩充时不理会参数所代表的文件是否真实存在，均以当前目录进行扩展  
要理解上面的知识，下面的例子很关键。  
例：  
@echo off  
Echo 产生一个临时文件 > tmp.txt  
Rem 下行先保存当前目录，再将c:\windows设为当前目录  
pushd c:\windows  
Call :sub tmp.txt  
Rem 下行恢复前次的当前目录  
Popd  
Call :sub tmp.txt  
pause  
Del tmp.txt  
exit  
:sub  
Echo 删除引号： %~1  
Echo 扩充到路径： %~f1  
Echo 扩充到一个驱动器号： %~d1  
Echo 扩充到一个路径： %~p1   
Echo 扩充到一个文件名： %~n1  
Echo 扩充到一个文件扩展名： %~x1  
Echo 扩充的路径指含有短名： %~s1   
Echo 扩充到文件属性： %~a1   
Echo 扩充到文件的日期/时间： %~t1   
Echo 扩充到文件的大小： %~z1   
Echo 扩展到驱动器号和路径：%~dp1  
Echo 扩展到文件名和扩展名：%~nx1  
Echo 扩展到类似 DIR 的输出行：%~ftza1  
Echo.  
Goto :eof  
例：  
set aa=123456  
set cmdstr=echo %aa%  
call %cmdstr%  
pause  
本例中如果不用call，而直接运行%cmdstr%，将显示结果%aa%，而不是123456

**14、shift**  
更改批处理文件中可替换参数的位置。  
SHIFT [/n]  
如果命令扩展名被启用，SHIFT 命令支持/n 命令行开关；该命令行开关告诉  
命令从第 n 个参数开始移位；n 介于零和八之间。例如:  
    SHIFT /2  
会将 %3 移位到 %2，将 %4 移位到 %3，等等；并且不影响 %0 和 %1。

**15、setlocal 与 变量延迟**

@echo off  
set a=4  
set a=5 & echo %a%  
pause  
结果：4

例2:  
@echo off  
setlocal enabledelayedexpansion  
set a=4  
set a=5 & echo !a!  
pause   
结果：5  
解说：启动了变量延迟，得到了正确答案。变量延迟的启动语句是“setlocal enabledelayedexpansion”，并且变量要用一对叹号“!!”括起来（注意要用英文的叹号），否则就没有变量延迟的效果。  
分析一下例2，首先“setlocal enabledelayedexpansion”开启变量延迟，然后“set a=4”先给变量a赋值为  
4，“set a=5 & echo !a!”这句是给变量a赋值为5并输出（由于启动了变量延迟，所以批处理能够感知到动态变化，即不是先给该行变量赋值，而是在运行过程中给变量赋值，因此此时a的值就是5了）。  
再举一个例子巩固一下。

**17、ATTRIB  显示或更改文件属性**  
ATTRIB [+R|-R] [+A|-A] [+S|-S] [+H|-H] [[drive:] [path] filename] [/S [/D]]  
  +   设置属性。  
  -    清除属性。  
  R   只读文件属性。  
  A   存档文件属性。  
  S   系统文件属性。  
  H   隐藏文件属性。  
  [drive:][path][filename]  
      指定要处理的文件属性。  
  /S  处理当前文件夹及其子文件夹中的匹配文件。  
  /D  也处理文件夹。

**6、|  命令管道符**  
格式：第一条命令 | 第二条命令 [| 第三条命令...]  
将第一条命令的结果作为第二条命令的参数来使用，记得在unix中这种方式很常见。  
例如：  
dir c:\|find "txt"  
以上命令是：查找C：\所有，并发现TXT字符串。  
FIND的功能请用 FIND /? 自行查看  
在不使format的自动格式化参数时，我是这样来自动格式化A盘的  
echo y|format a: /s /q /v:system  
用过format的都知道，再格盘时要输入y来确认是否格盘，这个命令前加上echo y并用|字符来将echo y的结果传给format命令  
从而达到自动输入y的目的

**7、^  转义字符**  
^是对特殊符号<,>,&的前导字符，在命令中他将以上3个符号的特殊功能去掉，仅仅只把他们当成符号而不使用他们的特殊意义。  
比如  
echo test ^>1.txt  
结果则是：test > 1.txt  
他没有追加在1.txt里，呵呵。只是显示了出来  
另外，此转义字符还可以用作续行符号。  
举个简单的例子：  
@echo off  
echo 英雄^  
是^  
好^  
男人  
pause  
不用多说，自己试一下就明白了。  
为什么转义字符放在行尾可以起到续行符的作用呢？原因很简单，因为每行末尾还有一个看不见的符号，即回车符，转义字符位于行尾时就让回车符失效了，从而起到了续行的作用。

**8、& 组合命令**  
语法：第一条命令 & 第二条命令 [& 第三条命令...]  
&、&&、||为组合命令，顾名思义，就是可以把多个命令组合起来当一个命令来执行。这在批处理脚本里是允许的，而且用的非常广泛。因为批处理认行不认命令数目。  
这个符号允许在一行中使用2个以上不同的命令，当第一个命令执行失败了，也不影响后边的命令执行。  
这里&两边的命令是顺序执行的，从前往后执行。  
比如：  
dir z:\ & dir y:\ & dir c:\  
以上命令会连续显示z,y,c盘的内容，不理会该盘是否存在  
  
  
**9、&& 组合命令**  
语法：第一条命令 && 第二条命令 [&& 第三条命令...]  
用这种方法可以同时执行多条命令，当碰到执行出错的命令后将不执行后面的命令，如果一直没有出错则一直执行完所有命令  
这个命令和上边的类似，但区别是，第一个命令失败时，后边的命令也不会执行  
dir z:\ && dir y:\ && dir c:\  
  
  
**10、||  组合命令**  
语法：第一条命令 || 第二条命令 [|| 第三条命令...]  
用这种方法可以同时执行多条命令，当一条命令失败后才执行第二条命令，当碰到执行正确的命令后将不执行后面的命令，如果没有出现正确的命令则一直执行完所有命令；

**二、参数 /d**  
FOR /D %%variable IN (set) DO command [command-parameters]

For /d %%i in (1,1,5) do @each %%i

(1,1,5)这个参数也就是表示从1开始每次加1直到5终止!

字符编码

由于计算机是美国人发明的，因此，最早只有127个字符被编码到计算机里，也就是大小写英文字母、数字和一些符号，这个编码表被称为ASCII编码，比如大写字母A的编码是65，小写字母z的编码是122。

但是要处理中文显然一个字节是不够的，至少需要两个字节，而且还不能和ASCII编码冲突，所以，中国制定了GB2312编码，用来把中文编进去。

你可以想得到的是，全世界有上百种语言，日本把日文编到Shift\_JIS里，韩国把韩文编到Euc-kr里，各国有各国的标准，就会不可避免地出现冲突，结果就是，在多语言混合的文本中，显示出来会有乱码。

因此，Unicode应运而生。Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。

Unicode标准也在不断发展，但最常用的是用两个字节表示一个字符（如果要用到非常偏僻的字符，就需要4个字节）。现代操作系统和大多数编程语言都直接支持Unicode。

现在，捋一捋ASCII编码和Unicode编码的区别：ASCII编码是1个字节，而Unicode编码通常是2个字节。

汉字中已经超出了ASCII编码的范围，用Unicode编码是十进制的20013，二进制的01001110 00101101。

你可以猜测，如果把ASCII编码的A用Unicode编码，只需要在前面补0就可以，因此，A的Unicode编码是00000000 01000001。

新的问题又出现了：如果统一成Unicode编码，乱码问题从此消失了。但是，如果你写的文本基本上全部是英文的话，用Unicode编码比ASCII编码需要多一倍的存储空间，在存储和传输上就十分不划算。

所以，本着节约的精神，又出现了把Unicode编码转化为“可变长编码”的UTF-8编码。UTF-8编码把一个Unicode字符根据不同的数字大小编码成1-6个字节，常用的英文字母被编码成1个字节，汉字通常是3个字节，只有很生僻的字符才会被编码成4-6个字节。如果你要传输的文本包含大量英文字符，用UTF-8编码就能节省空间：

在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。

用记事本编辑的时候，从文件读取的UTF-8字符被转换为Unicode字符到内存里，编辑完成后，保存的时候再把Unicode转换为UTF-8保存到文件：