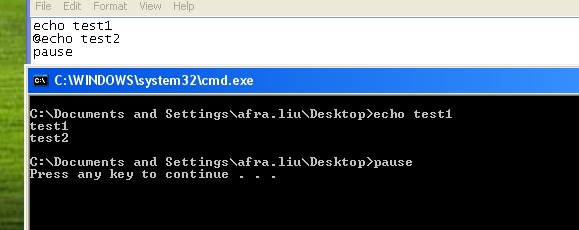
**常用批处理**

1. 特殊符号

@符号在批处理中的作用是关闭当前行命令的回显,也就是不显示执行的是什么命令,只显示命令的结果

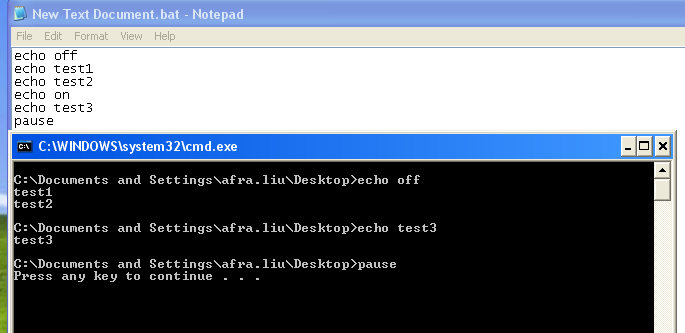


（ECHO on  
在DOS中默认是打开ECHO命令显示功能，通俗的说，就是显示现在正在执行的是什么命令，包括命令名称和参数等；  
ECHO off   
关闭命令显示功能，执行命令时只显示执行结果（如是否删除成功），不显示正在执行的命令。  
这样可以提高命令执行速度。

）

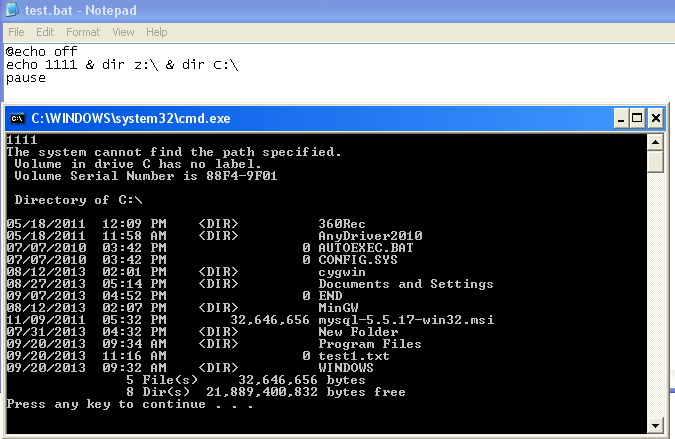
Echo off 相当于在每一行加上了@

Echo on 打开回显设置

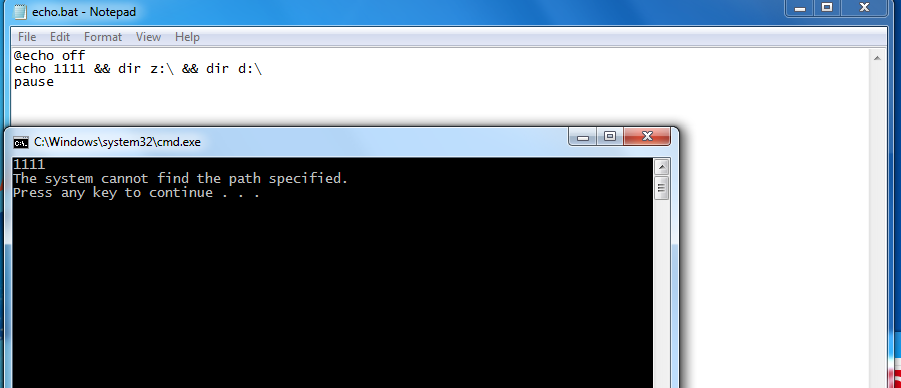


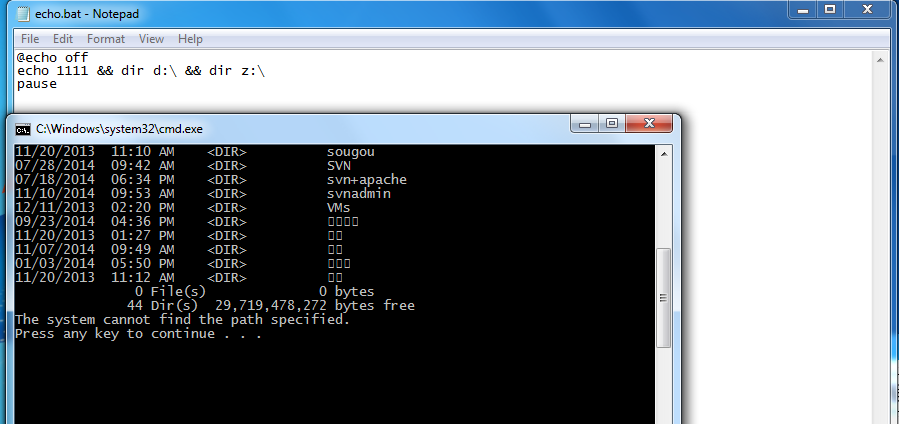
如果不想显示echo off ，将第一条命令改成 @echo off即可。批处理会显示得更简洁

**&** 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，而不管命令是否执行成功

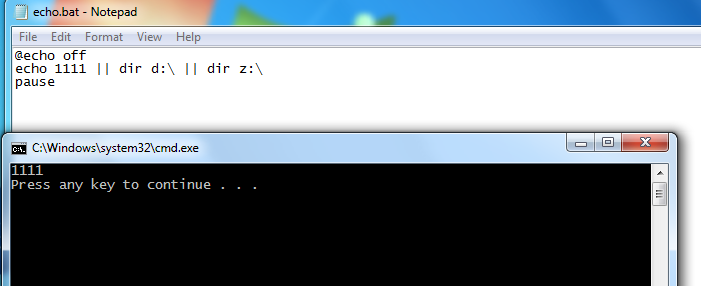
****

&& 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；





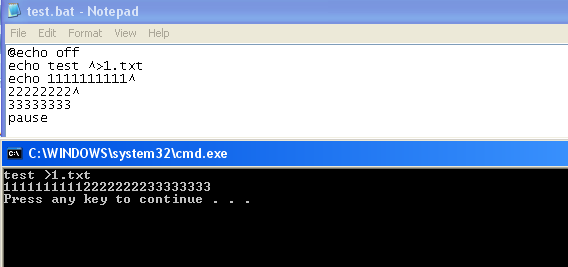
|| 第一条命令 || 第二条命令 [|| 第三条命令...]  
用这种方法可以同时执行多条命令，当碰到执行正确的命令后将不执行后面的命令，如果没有出现正确的命令则一直执行完所有命令；



:: 注释

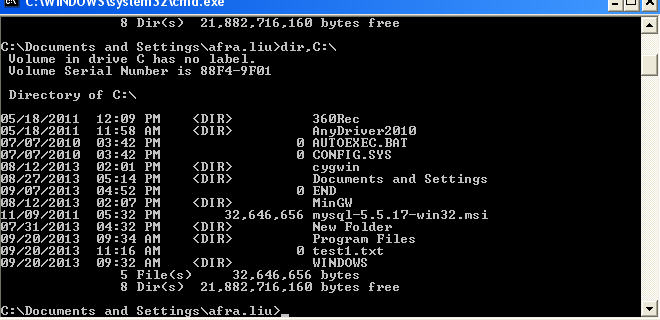
>   重定向符  
输出重定向命令  
这个字符的意思是传递并且覆盖，他所起的作用是将运行的结果传递到后面的范围（后边可以是文件，也可以是默认的系统控制台）  
    在NT系列命令行中，重定向的作用范围由整个命令行转变为单个命令语句，受到了命令分隔符&,&&,||和语句块的制约限制。  
比如：  
使用命令：echo hello >1.txt将建立文件1.txt，内容为”hello “（注意行尾有一空格）  
使用命令：echo hello>1.txt将建立文件1.txt，内容为”hello“（注意行尾没有空格）  
  
>>  重定向符  
输出重定向命令  
这个符号的作用和>有点类似，但他们的区别是>>是传递并在文件的末尾追加，而>是覆盖

|  命令管道符  
格式：第一条命令 | 第二条命令 [| 第三条命令...]  
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  
（这条命令有危害性，测试时请慎）

^  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  


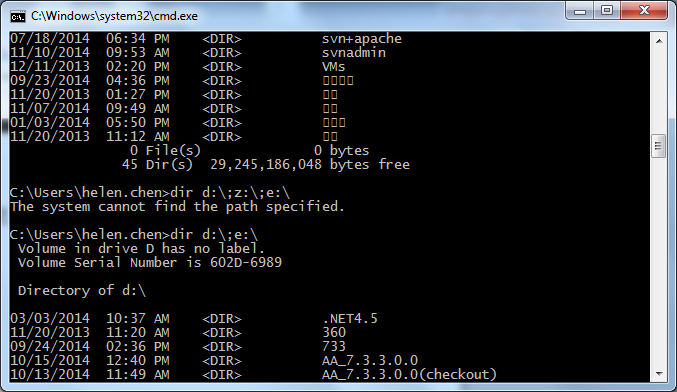
""  字符串界定符  
双引号允许在字符串中包含空格，进入一个特殊目录可以用如下方法  
cd "program files"  
cd progra~1  
cd pro\*  
以上三种方法都可以进入program files这个目录  
为什么转义字符放在行尾可以起到续行符的作用呢？原因很简单，因为每行末尾还有一个看不见的符号，即回车符，转义字符位于行尾时就让回车符失效了，从而起到了续行的作用。

, 逗号  
逗号相当于空格，在某些情况下“,”可以用来当做空格使  
比如  
dir,c:\

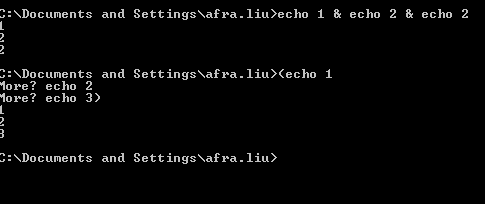


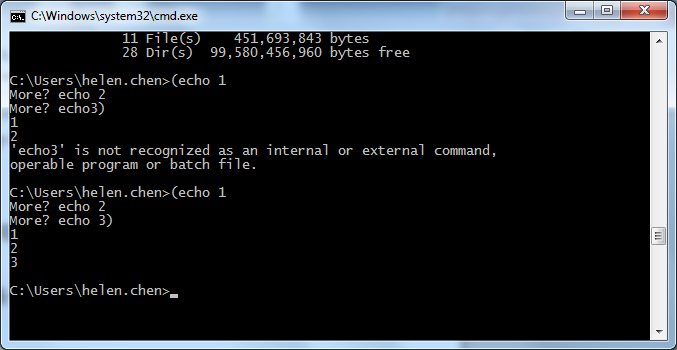
; 分号  
分号，当命令相同时，可以将不同目标用；来隔离，但执行效果不变，如执行过程中发生错误，则只返回错误报告，但程序仍会执行。

。（有人说不会继续执行，其实测试一下就知道了）  
比如：  
dir c:\;d:\;e:\;z:\  
以上命令相当于  
dir c:\  
dir d:\  
dir e:\  
dir f:\  
如果其中z盘不存在，运行显示：系统找不到指定的路径。然后终止命令的执行。  
例：dir c:\;d:\;e:\1.txt

  
以上命令相当于  
dir c:\  
dir d:\  
dir e:\1.txt  
其中文件e:\1.txt不存在，但e盘存在，有错误提示，但命令仍会执行。

() 括号  
小括号在批处理编程中有特殊的作用，左右括号必须成对使用，括号中可以包括多行命令，这些命令将被看成一个整体，视为一条命令行。  
    括号在for语句和if语句中常见，用来嵌套使用循环或条件语句，其实括号()也可以单独使用，请看例子。  
例：  
命令：echo 1 & echo 2 & echo 3  
可以写成：  
(  
echo 1  
echo 2  
echo 3  
)  
上面两种写法效果一样，这两种写法都被视为是一条命令行。

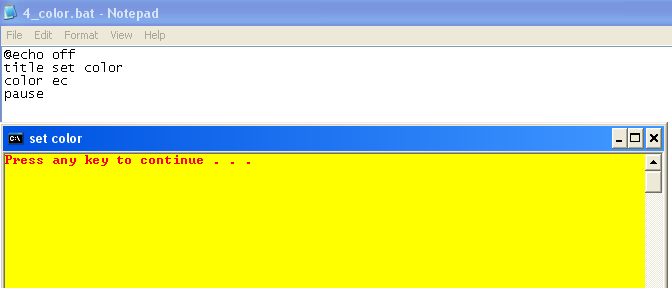




Echo 1回车就有“More?”生成。

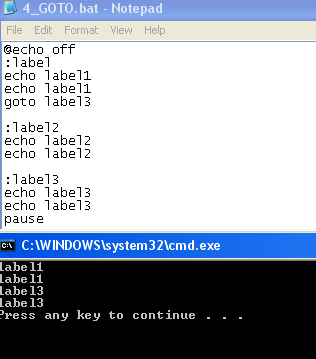
! 变量延迟里面会讲

1. errorlevel  
   程序返回码  
   echo %errorlevel%  
   每个命令运行结束，可以用这个命令行格式查看返回码  
   用于判断刚才的命令是否执行成功  
   默认值为0，一般命令执行出错会设 errorlevel 为1
2. title  
   设置cmd窗口的标题  
   title 新标题        #可以看到cmd窗口的标题栏变了
3. COLOR  
   设置默认的控制台前景和背景颜色。  
   颜色属性由两个十六进制数字指定 -- 第一个为背景，第二个则为  
   前景。每个数字可以为以下任何值之一:  
       0 = 黑色       8 = 灰色  
       1 = 蓝色       9 = 淡蓝色  
       2 = 绿色       A = 淡绿色  
       3 = 湖蓝色     B = 淡浅绿色  
       4 = 红色       C = 淡红色  
       5 = 紫色       D = 淡紫色  
       6 = 黄色       E = 淡黄色  
       7 = 白色       F = 亮白色  
   如果没有给定任何参数，该命令会将颜色还原到 CMD.EXE 启动时  
   的颜色。这个值来自当前控制台窗口、/T 开关或  
   DefaultColor 注册表值。  
   如果用相同的前景和背景颜色来执行 COLOR 命令，COLOR 命令  
   会将 ERRORLEVEL 设置为 1。  
   例如: "COLOR fc" 在亮白色上产生亮红色



1. REM为注释命令，一般用来给程序加上注解，该命令后的内容不被执行，但能回显。  
   其次, :: 也可以起到rem 的注释作用, 而且更简洁有效; 但有两点需要注意：  
   第一, 任何以冒号:开头的字符行, 在批处理中都被视作标号, 而直接忽略其后的所有内容。  
   有效标号：冒号后紧跟一个以字母数字开头的字符串，goto语句可以识别。  
   无效标号：冒号后紧跟一个非字母数字的一个特殊符号，goto无法识别的标号，可以起到注释作用，所以 :: 常被用作注释符号，其实 :+ 也可起注释作用。  
   第二, 与rem 不同的是, ::后的字符行在执行时不会回显, 无论是否用echo on打开命令行回显状态, 因为命令解释器不认为他是一个有效的命令行, 就此点来看, rem 在某些场合下将比 :: 更为适用; 另外, rem 可以用于 config.sys 文件中。
2. ECHO 回显命令，参考1. 在DOS 窗口打印Message
3. GOTO 跳转

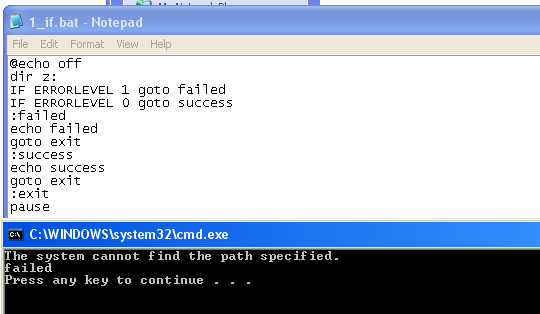
以“:XXX”来构建一个标号，然后用GOTO XXX直接来执行标号后的命令。

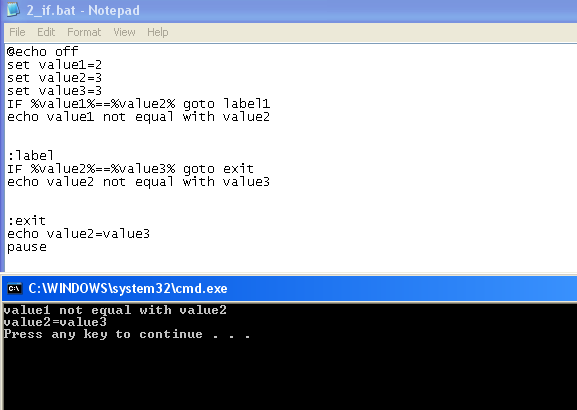


1. CALL命令可以在批处理执行过程中调用另一个批处理，当另一个批处理执行完后，再继续执行原来的

Start 命令也是在批处理执行过程中调用另一个批处理,但是不会等这个批处理执行完。

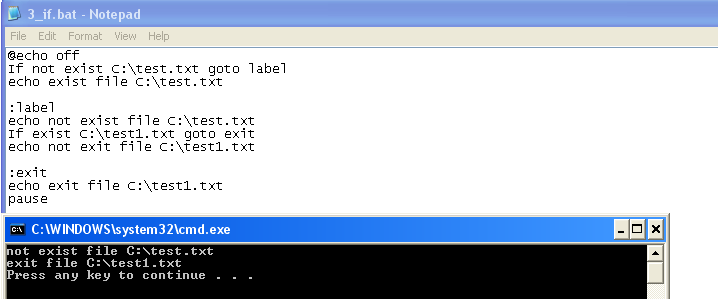
Start的时候会启两个dos窗口， call只启一个。

1. Pause 在这里就是停止系统命令的执行并显示下面的内容
2. Exit 结束命令….
3. IF 条件判断语句，语法格式如下：   
   IF [NOT] ERRORLEVEL number command   
   IF [NOT] string1==string2 command   
   IF [NOT] EXIST filename command
4. IF [NOT] ERRORLEVEL number command   
   检测命令执行完后的返回值做出判断。 0 是成功，1 是失败  
   
5. **IF string1==string2 command**   
   **检测当前变量的值做出判断**



(3). **IF [NOT] EXIST filename command**

**判断文件是否存在**

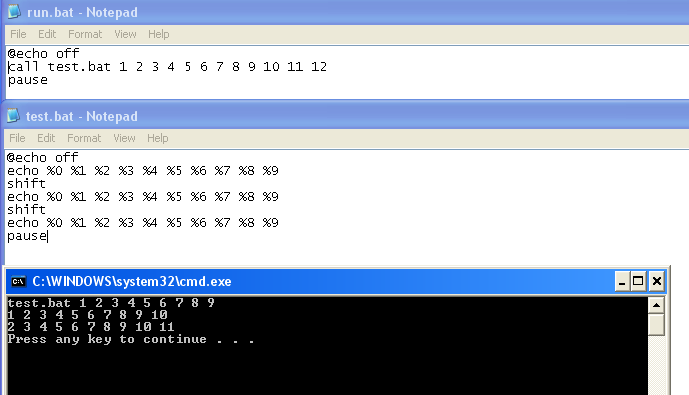


If…..else….

IF EXIST filename (  
        del filename  
    ) ELSE (  
        echo filename missing  
    )

1. Shift 命令

SHIFT命令可以让在命令上的的命令使用超过10个（%0~%9）以上的可替代参数例：



1. format (外部命令)格式化硬盘 format c: /q/u/autotest/q表示快速格式化，/autotest表示自动格式化，不需要按 Y 确认/u表示每字节用
2. md创建目录

md abc #在当前目录里建立子目录 abc

md d:\a\b\c #如果 d:\a 不存在，将会自动创建

1. rd删除目录

rd abc #删除当前目录里的 abc 子目录，要求为空目录

rd /s/q d:\temp #删除 d:\temp 文件夹及其子文件夹和文件，不需要按 Y 确认

1. ren文件重命名

ren 1.txt 2.bak #把 1.txt 更名为 2.bak

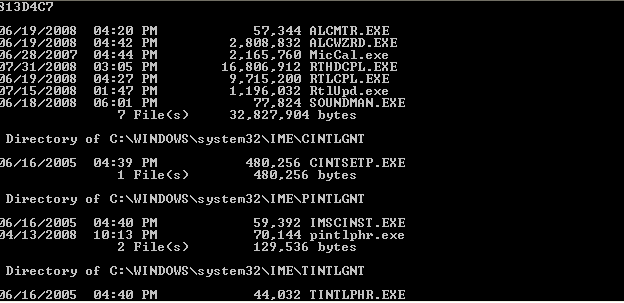
ren \*.txt \*.ini #把当前目录里所有.txt文件改成.ini文件

ren d:\temp tmp #支持对文件夹的重命名

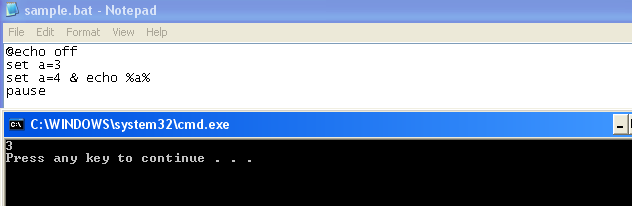
1. dir 显示目录中的文件和子目录列表。

Dir C:\

Dir C:\\*.exe /s



1. tree 以图形显示驱动器或路径的文件夹结构。
2. xcopy 复制文件和目录树。
3. del 删除文件
4. setlocal 与 变量延迟



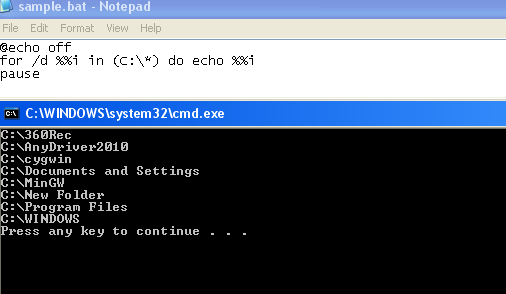
批处理读取命令时是按行读取的（另外例如for命令等，其后用一对圆括号闭合的所有语句也当作一行），在处理之前要完成必要的预处理工作，这其中就包括对该行命令中的变量赋值。我们现在分析一下例1，批处理在运行到这句“set a=5 & echo %a%”之前，先把这一句整句读取并做了预处理——对变量a赋了值，那么%a%当然就是4了！（没有为什么，批处理就是这样做的。）  
而为了能够感知环境变量的动态变化，批处理设计了变量延迟。简单来说，在读取了一条完整的语句之后，不立即对该行的变量赋值，而会在某个单条语句执行之前再进行赋值，也就是说“延迟”了对变量的赋值。  
那么如何开启变量延迟呢？变量延迟又需要注意什么呢？举个例子说明一下：  
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。

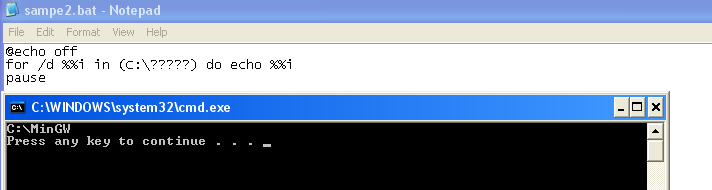
13．For循环内容太多，大家可以参考

FOR %%variable IN (set) DO command [command-parameters]  
  %%variable  指定一个单一字母表示可替换的参数。  
  (set)      指定一个或一组文件。可以使用通配符。  
  command    指定对每个文件执行的命令。  
  command-parameters  
             为特定命令指定参数或命令行开关。

FOR有4个参数 /d   /l   /r   /f   他们的作用我在下面用例子解释

FOR /D %%variable IN (set) DO command [command-parameters]  
    如果集中包含通配符，则指定与目录名匹配，而不与文件  
    名匹配。

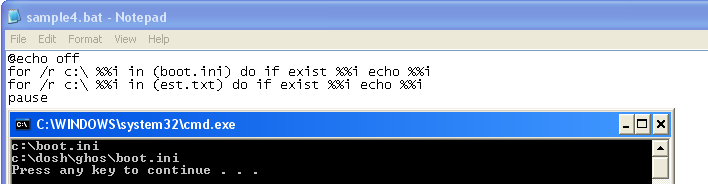
****



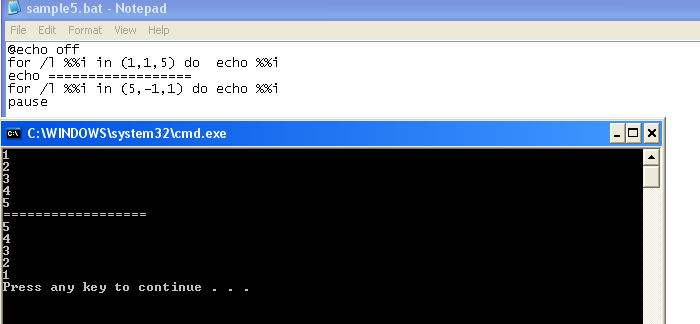
\*号表示任意N个字符,而?号只表示任意一个字符

FOR /R [[drive:]path] %%variable IN (set) DO command [command-parameters]  
    检查以 [drive:]path 为根的目录树，指向每个目录中的  
    FOR 语句。如果在 /R 后没有指定目录，则使用当前  
目录。如果集仅为一个单点(.)字符，则枚举该目录树。

当前或者你指定路径下的文件名字全部读取,注意是文件名字



参数 /L  
FOR /L %%variable IN (start,step,end) DO command [command-parameters]  
    该集表示以增量形式从开始到结束的一个数字序列。  
    因此，(1,1,5) 将产生序列 1 2 3 4 5，(5,-1,1) 将产生  
    序列 (5 4 3 2 1)。



系统变量：

%ALLUSERSPROFILE% 本地 返回“所有用户”配置文件的位置。  
%APPDATA% 本地 返回默认情况下应用程序存储数据的位置。  
%CD% 本地 返回当前目录字符串。  
%CMDCMDLINE% 本地 返回用来启动当前的 Cmd.exe 的准确命令行。  
%CMDEXTVERSION% 系统 返回当前的“命令处理程序扩展”的版本号。  
%COMPUTERNAME%  系统 返回计算机的名称。  
%COMSPEC%  系统 返回命令行解释器可执行程序的准确路径。  
%DATE%  系统 返回当前日期。使用与 date /t 命令相同的格式。由 Cmd.exe 生成。有关  
date 命令的详细信息，请参阅 Date。  
%ERRORLEVEL%  系统 返回上一条命令的错误代码。通常用非零值表示错误。  
%HOMEDRIVE%  系统 返回连接到用户主目录的本地工作站驱动器号。基于主目录值而设置。用  
户主目录是在“本地用户和组”中指定的。  
%HOMEPATH%  系统 返回用户主目录的完整路径。基于主目录值而设置。用户主目录是在“本地用户和组”中指定的。  
%HOMESHARE%  系统 返回用户的共享主目录的网络路径。基于主目录值而设置。用户主目录是  
在“本地用户和组”中指定的。  
%LOGONSERVER%  本地 返回验证当前登录会话的域控制器的名称。  
%NUMBER\_OF\_PROCESSORS%  系统 指定安装在计算机上的处理器的数目。  
%OS%  系统 返回操作系统名称。Windows 2000 显示其操作系统为 Windows\_NT。  
%PATH% 系统 指定可执行文件的搜索路径。  
%PATHEXT% 系统 返回操作系统认为可执行的文件扩展名的列表。  
%PROCESSOR\_ARCHITECTURE%  系统 返回处理器的芯片体系结构。值：x86 或 IA64 基于  
Itanium  
%PROCESSOR\_IDENTFIER% 系统 返回处理器说明。  
%PROCESSOR\_LEVEL%  系统 返回计算机上安装的处理器的型号。  
%PROCESSOR\_REVISION% 系统 返回处理器的版本号。  
%PROMPT% 本地 返回当前解释程序的命令提示符设置。由 Cmd.exe 生成。  
%RANDOM% 系统 返回 0 到 32767 之间的任意十进制数字。由 Cmd.exe 生成。  
%SYSTEMDRIVE% 系统 返回包含 Windows server operating system 根目录（即系统根目录）  
的驱动器。  
%SYSTEMROOT%  系统 返回 Windows server operating system 根目录的位置。  
%TEMP% 和 %TMP% 系统和用户 返回对当前登录用户可用的应用程序所使用的默认临时目录。  
有些应用程序需要 TEMP，而其他应用程序则需要 TMP。  
%TIME% 系统 返回当前时间。使用与 time /t 命令相同的格式。由 Cmd.exe 生成。有关  
time 命令的详细信息，请参阅 Time。  
%USERDOMAIN% 本地 返回包含用户帐户的域的名称。  
%USERNAME% 本地 返回当前登录的用户的名称。  
%USERPROFILE% 本地 返回当前用户的配置文件的位置。  
%WINDIR% 系统 返回操作系统目录的位置。

批处理修改IE代理:

批处理添加环境变量:

批处理修改机器名：

参考: <http://cvs-server.achievo.com:8001/svn/projects/AUTOTEST/Tool/ConfigVM/bat/VMTool.bat>