目錄

DOS批处理教程

原作者:龙卷风

原作者博客:http://xiangkg.blog.163.com/

根据原作者的文章“批处理高级教程精选合编”编写的GitBook版本。一些示例进行了优化。

本文档使用GitBook发布:https://www.gitbook.com/book/izombielandgit/dos-bat-tutorial/details

本文档GitHub:https://github.com/izombielandgit/DOS-BAT-Tutorial

1. 批处理基础

1.1. 常用批处理内部命令简介

批处理定义：顾名思义，批处理文件是将一系列命令按一定的顺序集合为一个可执行的文本文件，其扩展名为BAT或者CMD。这些命令统称批处理命令。

小知识：可以在键盘上按下Ctrl+C组合键来强行终止一个批处理的执行过程。

了解了大概意思后，我们正式开始学习。先看一个简单的例子！

```
@echo off
echo "欢迎来到非常 BAT!"
pause
```

把上面的3条命令保存为文件test.bat或者test.cmd然后执行，他就会在屏幕上显示以下二行话：

```
欢迎来到非常 BAT!
请按任意键继续...
```

这就是一个简单批处理文件了，这个批处理文件一共就用了2条命令"echo"和"pause"还有一个特殊符号"@"

从上面这个简单的批处理中,我们可以发现其实批处理就是运用一些含有特殊意义的符号和一些完成指定功能的命令组合而成,那么在批处理中有多少这样的特殊符号和功能命令呢？

我们现在就来仔细了解一下一些最常用的！

(以下内容来源网络,请各位仔细阅读,好进入下节的实例说明)

批处理的常见命令（未列举的命令还比较多，请查阅帮助信息）

1. REM 和 ::
2. ECHO 和 @
3. PAUSE
4. ERRORLEVEL
5. TITLE
6. COLOR

7. mode 配置系统设备
8. GOTO 和 :
9. FIND
10. START
11. assoc 和 ftype
12. pushd 和 popd
13. CALL
14. shift
15. IF
16. setlocal 与 变量延迟

介绍命令

1. REM 和 ::

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 和 @

打开回显或关闭回显功能，或显示消息。如果没有任何参数，echo命令将显示当前回显设置。

@字符放在命令前将关闭该命令回显，无论此时echo是否为打开状态。

语法：

```
echo [ON | OFF]
echo [message]
```

`echo.` 此用法将显示一空行，相当于回车，非常有用。

执行echo off将关闭回显，它后面的所有命令都不显示命令本身，只显示执行后的结果，除非执行echo on命令。

执行@echo off不但关闭以后命令的回显，连echo off命令本身也不显示了。通常以@echo off作为批处理程序的首行。

一般用ECHO MESSAGE来显示一个特定的消息。例：

```
@Echo off
Echo hello
Pause
```

运行显示：

```
hello
```

3. PAUSE

PAUSE，玩游戏的人都知道，暂停的意思。在这里就是停止系统命令的执行并显示下面的内容。例：

```
@Echo off
PAUSE
```

运行显示：

```
请按任意键继续...
```

要显示其他提示语，可以这样用：

```
@Echo off
Echo 其他提示语 & pause>nul
```

笔者注：扩展介绍：nul解释是“空设备”，`>nul` 可以扩展到其他命令，实际应有 `1>nul` ， `2>nul` 等用法， `1>nul` 表示把正确的信息隐藏， `2>nul` 代表把错误信息隐藏，默认 `>nul` 即为 `1>nul` 。若正确信息和错误信息都隐藏则写为 `pause 1>nul 2>nul` 。由于pause没什么错误信息，所以 `pause 1>nul` 通常写为 `pause>nul` 。

4. 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。

5. TITLE

设置cmd窗口的标题

```
@Echo off
title 新标题
pause
```

6. COLOR

设置默认的控制台前景和背景颜色。

```
COLOR [attr]
```

`attr` 指定控制台输出的颜色属性

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
- 1 = 蓝色
- 2 = 绿色
- 3 = 湖蓝色
- 4 = 红色
- 5 = 紫色
- 6 = 黄色
- 7 = 白色
- 8 = 灰色
- 9 = 淡蓝色
- A = 淡绿色
- B = 淡浅绿色
- C = 淡红色
- D = 淡紫色
- E = 淡黄色
- 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

7. mode 配置系统设备

配置系统设备。

串行口：

```
MODE COMm[:] [BAUD=b] [PARITY=p] [DATA=d] [STOP=s]
            [to=on|off] [xon=on|off] [odsr=on|off]
            [octs=on|off] [dtr=on|off|hs]
            [rts=on|off|hs|tg] [idsr=on|off]
```

设备状态：

```
MODE [device] [/STATUS]
```

打印重定向：

```
MODE LPTn[:]=COMm[:]
```

选择代码页：

```
MODE CON[:] CP SELECT=yyy
```

代码页状态：

```
MODE CON[:] CP [/STATUS]
```

显示模式：

```
MODE CON[:] [COLS=c] [LINES=n]
```

击键率：

```
MODE CON[:] [RATE=r DELAY=d]
```

例：

```
mode con cols=113 lines=15 & color 9f
```

此命令设置DOS窗口大小：15行，113列

8. GOTO 和：

GOTO会点编程的朋友就会知道这是跳转的意思。

在批处理中允许以 `:XXX` 来构建一个标号，然后用 `GOTO XXX` 跳转到标号 `:XXX` 处，然后执行标号后的命令。例：

```
if {%1}=={} goto noparms
if "%2"=="" goto noparms
```

标签的名字可以随便起，但是最好是有意义的字符串啦，前加个冒号用来表示这个字符串是标签，goto命令就是根据这个冒号（:）来寻找下一步跳到到那里。最好有一些说明这样你别人看起来才会理解你的意图啊。例：

```
@echo off
:start
set /a var+=1
echo %var%
if %var% leq 3 GOTO start
pause
```

运行显示：

```
1
2
3
4
请按任意键继续...
```

9. FIND

在文件中搜索字符串。

```
FIND [/V] [/C] [/N] [/I] [/OFF[LINE]] "string" [[drive:][path]fi
lename[ ...]]
```

```
/V                          #显示所有未包含指定字符串的行。
/C                          #仅显示包含字符串的行数。
/N                          #显示行号。
/I                          #搜索字符串时忽略大小写。
/OFF[LINE]                  #不要跳过具有脱机属性集的文件。
"string"                    #指定要搜索的文本字符串。
[drive:][path]filename      #指定要搜索的文件。
```

如果没有指定路径，FIND将搜索在提示符处键入的文本或者由另一命令产生的文本。

Find常和type命令结合使用

```
Type [drive:][path]filename | find "string" [>tmpfile] #挑选包含string的行
Type [drive:][path]filename | find /v "string" #剔除文件中包含string的行
Type [drive:][path]filename | find /c #显示文件行数
```

以上用法将去除find命令自带的提示语（文件名提示）

例：

```
@echo off
echo 111 >test.txt
echo 222 >>test.txt
find "111" test.txt
del test.txt
pause
```

运行显示如下：

```
---------- TEST.TXT
111
请按任意键继续...
```

例：

```
@echo off
echo 111 >test.txt
echo 222 >>test.txt
type test.txt|find "111"
del test.txt
pause
```

运行显示如下：

```
111
请按任意键继续...
```

10. START

批处理中调用外部程序的命令（该外部程序在新窗口中运行，批处理程序继续往下执行，不理会外部程序的运行状况），如果直接运行外部程序则必须等外部程序完成后才继续执行剩下的指令。

例：

```
start explorer d:\
```

调用图形界面打开D盘

11. assoc 和 ftype

文件关联

`assoc` 设置“文件扩展名”关联，关联到“文件类型”

`ftype` 设置“文件类型”关联，关联到“执行程序和参数”

当你双击一个“.txt”文件时，windows并不是根据“.txt”直接判断用“notepad.exe”打开，而是先判断“.txt”属于txtfile“文件类型”，再调用 txtfile关联的命令行 `txtfile=%SystemRoot%\system32\NOTEPAD.EXE %1` 可以在“文件夹选项”->“文件类型”里修改这2种关联。

```
assoc           #显示所有'文件扩展名'关联
assoc .txt      #显示.txt 代表的'文件类型'，结果显示 .txt=txtfile
assoc .doc      #显示.doc 代表的'文件类型'，结果显示 .doc=Word.Documen
t.8
assoc .exe      #显示.exe 代表的'文件类型'，结果显示 .exe=exefile
ftype           #显示所有'文件类型'关联
ftype exefile   #显示 exefile 类型关联的命令行，结果显示 exefile="%1"
%*
```

`assoc .txt=Word.Document.8` 设置“.txt”为word类型的文档，可以看到“.txt”文件的图标都变了

`assoc .txt=txtfile` 恢复“.txt”的正确关联

`ftype exefile="%1" %*` 恢复“exefile”的正确关联。如果该关联已经被破坏，可以运行 `command.com` ，再输入这条命令。

12. pushd 和 popd

切换当前目录

```
c: & cd\ & md mp3  #在C:\建立mp3文件夹
md d:\mp4          #在D:\建立mp4文件夹
cd /d d:\mp4       #更改当前目录为d:\mp4
pushd c:\mp3       #保存当前目录，并切换当前目录为c:\mp3
popd               #恢复当前目录为刚才保存的d:\mp4
```

一般用处不大，在当前目录名不确定时，会有点帮助。（dos编程中很有用）

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 [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
: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
```

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. IF

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
```

下面逐一介绍，更详细的分析请看后面章节。

IF [NOT] ERRORLEVEL number command

IF ERRORLEVEL这个句子必须放在某一个命令的后面，执行命令后由IF ERRORLEVEL来判断命令的返回值。

Number的数字取值范围0~255，判断时值的排列顺序应该由大到小。返回的值大于等于指定的值时，条件成立。例：

```
@echo off
dir c:
rem 退出代码为>=1 就跳至标题 1 处执行，>=0 就跳至标题 0 处执行
IF ERRORLEVEL 1 goto 1
IF ERRORLEVEL 0 goto 0
Rem 上面的两行不可交换位置，否则失败了也显示成功。
:0
echo 命令执行成功！
Rem 程序执行完毕跳至标题 exit 处退出
goto exit
:1
echo 命令执行失败！
Rem 程序执行完毕跳至标题 exit 处退出
goto exit
:exit
pause
```

运行显示：

```
命令执行成功！
请按任意键继续...
```

IF [NOT] string1==string2 command

string1和string2都为字符的数据，英文内字符的大小写将看作不同，这个条件中的等于号必须是两个（绝对相等的意思）

条件相等后即执行后面的command

检测当前变量的值做出判断，为了防止字符串中含有空格，可用以下格式

```
if [NOT] {string1}=={string2} command
if [NOT] [string1]==[string2] command
if [NOT] "string1"=="string2" command
```

这种写法实际上将括号或引号当成字符串的一部分了，只要等号左右两边一致就行了，比如下面的写法就不行：

```
if {string1}==[string2] command
```

IF [NOT] EXIST filename command

EXIST filename为文件或目录存在的意思

```
@echo off
IF EXIST autoexec.bat echo 文件存在！
IF not EXIST autoexec.bat echo 文件不存在！
pause
```

这个批处理大家可以放在C盘和D盘分别执行，看看效果（笔者注：C盘好像不一定有autoexec.bat，理解这个意思就行了。）

16. setlocal 与 变量延迟

要想进阶，变量延迟是必过的一关！所以这一部分希望你能认真看。

为了更好的说明问题，我们先引入一个例子。

例1：

```
@echo off
set a=4
set a=5 & echo %a%
pause
```

结果：

```
4
```

解说：为什么是4而不是5呢？在echo之前明明已经把变量a的值改成5了？让我们先了解一下批处理运行命令的机制：

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。

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。

例3：

```
@echo off
setlocal enabledelayedexpansion
for /l %%i in (1,1,5) do (
set a=%%i
echo !a!
)
pause
```

结果：

```
1
2
3
4
5
```

解说：本例开启了变量延迟并用“!!”将变量扩起来，因此得到我们预期的结果。如果不用变量延迟会出现什么结果呢？结果是这样的：

```
ECHO 处于关闭状态。
ECHO 处于关闭状态。
ECHO 处于关闭状态。
ECHO 处于关闭状态。
ECHO 处于关闭状态。
```

即没有感知到for语句中的动态变化。

1.2. 常用特殊符号

1. @ 命令行回显屏蔽符
2. % 批处理变量引导符
3. > 重定向符
4. >> 重定向符
5. < >& <& 重定向符
6. | 命令管道符
7. ^ 转义字符
8. & 组合命令
9. && 组合命令
10. || 组合命令
11. "" 字符串界定符
12. , 逗号
13. ; 分号
14. () 括号
15. ! 感叹号
16. 批处理中可能会见到的其它特殊标记符:（略）

1. @ 命令行回显屏蔽符

这个字符在批处理中的意思是关闭当前行的回显。我们从前几课知道 `ECHO OFF` 可以关闭掉整个批处理命令的回显，但不能关掉 `ECHO OFF` 这个命令，现在我们在 `ECHO OFF` 这个命令前加个 `@` ，就可以达到所有命令均不回显的要求。

2. % 批处理变量引导符

这个百分号严格来说是算不上命令的，它只是批处理中的参数而已（多个%一起使用的情况除外，以后还将详细介绍）。

引用变量用 `%var%` ，调用程序外部参数用 `%1` 至 `%9` 等等

`%0 %1 %2 %3 %4 %5 %6 %7 %8 %9 %*` 为命令行传递给批处理的参数

`%0` 批处理文件本身，包括完整的路径和扩展名

`%1` 第一个参数

`%9` 第九个参数

`%*` 从第一个参数开始的所有参数

参数 `%0` 具有特殊的功能，可以调用批处理自身，以达到批处理本身循环的目的，也可以复制文件自身等等。

例：最简单的复制文件自身的方法

```
copy %0 d:\wind.bat
```

3. > 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）

4. >> 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。用法同上

同样拿1.txt做例子，使用命令：

```
echo hello > 1.txt
echo world >>1.txt
```

这时候1.txt内容如下:

```
hello
world
```

5. < >& <& 重定向符

这三个命令也是管道命令，但它们一般不常用，你只需要知道一下就OK了，当然如果想仔细研究的话，可以自己查一下资料。（本人已查过，网上也查不到相关资料）

< 输入重定向命令，从文件中读入命令输入，而不是从键盘中读入。

```
@echo off
echo 2005-05-01>temp.txt
date <temp.txt
del temp.txt
pause
```

这样就可以不等待输入直接修改当前日期

>& 将一个句柄的输出写入到另一个句柄的输入中。

<& 刚好和>&相反，从一个句柄读取输入并将其写入到另一个句柄输出中。

常用句柄：0、1、2，未定义句柄：3—9

`1>nul` 表示禁止输出正确的信息

`2>nul` 表示禁止输出错误信息。

其中的1与2都是代表某个数据流输入输出的地址（NT CMD称之为句柄，MSDOS称之为 设备）。

句柄0：标准输入stdin，键盘输入

句柄1：标准输出stdout，输出到命令提示符窗口（console，代码为CON）

句柄2：标准错误stderr，输出到命令提示符窗口（console，代码为CON）

其中的stdin可被<重定向，stdout可被 `>` 、 `>>` 重定向，而stderr在DOS下不可直接重定向，只有通过ctty或其它命令将系统控制权转交给其它设备的方式，来间接完成。

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，在格盘时要输入 `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，测试时请慎重）

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，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 呵呵^
哈哈^
额
pause
```

不用多说，自己试一下就明白了。

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；

提示：组合命令和重定向命令一起使用必须注意优先级

管道命令的优先级高于重定向命令，重定向命令的优先级高于组合命令

问题：把C盘和D盘的文件和文件夹列出到a.txt文件中。你将如何来搞定这道题？有朋友说，这还不是很简单的问题吗？同时执行两个dir，然后把得到的结果 `>` 到a.txt 里就OK了嘛，

```
dir c:\ && dir d:\ > a.txt
```

仔细研究一下这句执行后的结果，看看是否能达到题目的要求！错了！这样执行后a.txt里只有D盘的信息！为什么？就因为这里 `&&` 命令和 `>` 命令不能同时出现一个句子里（批处理把一行看成一个句子）！！组合命令 `&&` 的优先级没有管道命令 `>` 的优先级高（自己总结的，不妥的地方请指正）！所以这句在执行时将本行分成这两部分： `dir c:\` 和 `dir d:\ > a.txt` ，而并不是如你想的这两部分： `dir c:\ && dir d:\` 和 `> a.txt` 。要使用组合命令 `&&` 达到题目的要求，必须得这么写：

```
dir c:\ > a.txt && dir d:\ >> a.txt
```

这样，依据优先级高低，DOS将把这句话分成以下两部分： `dir c:\ > a.txt` 和 `dir d:\ >> a.txt` 。

当然这里还可以利用 `&` 命令（自己想一下道理哦）：

```
dir c:\ > a.txt & dir d:\ >> a.txt
```

11. "" 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。

12. , 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:\
```

13. ; 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 z:\
```

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。

为什么？如果目标路径不存在，则终止执行；如果路径存在，文件不存在，则继续执行。

14. () 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：

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。

注意：这种多条命令被视为一条命令行时，如果其中有变量，就涉及到变量延迟的问题。

15. ! 感叹号

没啥说的，在变量延迟问题中，用来表示变量，即 `%var%` 应该表示为 `!var!` ，请看前面的setlocal命令介绍。

16. 批处理中可能会见到的其它特殊标记符:（略）

```
CR(0D)          #命令行结束符
Escape(1B)      #ANSI 转义字符引导符
Space(20)       #常用的参数界定符
Tab(09) ; =     #不常用的参数界定符
+ COPY          #命令文件连接符
* ?            #文件通配符
/               #参数开关引导符
:               #批处理标签引导符
```

2. FOR命令详解

讲FOR之前，先告诉各位新手朋友，如果你有什么命令不懂，直接在CMD下面输入：`name /?`这样的格式来看系统给出的帮助文件，比如`for /?`就会把FOR命令的帮助全部显示。

笔者注：初学时，不明白各命令是如何运行的可以尝试把`@echo off`去掉来观察

FOR这条命令基本上都被用来处理文本，我们除了要说他处理文本的作用外还要讲他的其他一些好用的功能！

看看他的基本格式（这里我引用的是批处理中的格式，直接在命令行只需要一个`%`号）：

```
FOR 参数 %%变量名 IN (相关文件或命令) DO 执行的命令
```

参数：FOR有4个参数：

1. /d
2. /r
3. /l
4. /f

他们的作用我在下面用例子解释。

%%变量名：这个变量名可以是单个的小写a-z或者大写A-Z，需区分大小写，FOR会把每个读取到的值给他！

IN：命令的格式,照写就是了！

(相关文件或命令**)**：FOR要把什么东西读取然后赋值给变量，不懂的话看下面的例子。

DO：命令的格式,照写就是了！

执行的命令：对每个变量的值要执行什么操作就写在这。

看不懂这些说明，可以在CMD输入`for /?`看系统提供的帮助！这里也给出来，大家对照：

```
FOR %variable IN (set) DO command [command-parameters]

  %variable           指定一个单一字母可替换的参数。
  (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。
```

2.1. FOR命令中的参数

1. /d

```
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。

如果Set（也就是上面写的“相关文件或命令”）包含通配符（ * 和 ？ ），将对与 Set 相匹配的每个目录（而不是指定目录中的文件组）执行指定的Command。

这个参数主要用于目录搜索，不会搜索文件，看这样的例子：

```
@echo off
for /d %%i in (c:\*) do echo %%i
pause
```

运行会把C盘根目录下的全部目录名字打印出来，而文件名字一个也不显示！

再来一个，比如我们要把当前路径下文件夹的名字只有1-3个字母的打出来：

```
@echo off
for /d %%i in (???) do echo %%i
pause
```

这样的话如果你当前目录下有目录名字只有1-3个字母的，就会显示出来，没有就不显示了。

这里解释下 * 号和 ？ 号的作用， * 号表示任意N个字符，而 ？ 号只表示任意一个字符

知道作用了，给大家个思考题目：

```
@echo off
for /d %%i in (window?) do @echo %%i
pause
```

保存到C盘下执行，会显示什么呢？自己看吧！显示：

```
windows
```

`/D` 参数只能显示当前目录下的目录名字，这个大家要注意！

2. /r

```
FOR /R [[drive:]path] %variable IN (set) DO command [command-par
ameters]
```

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。

递归

上面我们知道， `/D` 只能显示当前路径下的目录名字，那么现在这个 `/R` 也是和目录有关，他能干嘛呢？放心他比 `/D` 强大多了！

他可以把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，有什么用看例子！请注意2点：

1. set中的文件名如果含有通配符(`？` 或 `*`)，则列举 `/R` 参数指定的目录及其下面的所有子目录中与set相符合的所有文件，无相符文件的目录则不列举。
2. 相反，如果set中为具体文件名，不含通配符，则枚举该目录树（即列举该目录及其下面的所有子目录），而不管set中的指定文件是否存在。这与前面所说的单点 `.` 枚举目录树是一个道理，单点代表当前目录，也可视为一个文件。

例：

```
@echo off
for /r c:\ %%i in (*.exe) do echo %%i
pause
```

把这个BAT保存到D盘随便哪里然后执行，我们会看到，它把C盘根目录，和每个目录的子目录下面全部的EXE文件都列出来了！

例：

```
@echo off
for /r %%i in (*.exe) do @echo %%i
pause
```

参数不一样了！这个命令前面没加 `C:\` ，也就是搜索路径，这样他就会以当前目录为搜索路径，比如你这个BAT你把他放在 `d:\test` 目录下执行，那么他就会把 `d:\test` 目录和下面的子目录的全部EXE文件列出来！

例：

```
@echo off
for /r c:\ %%i in (boot.ini) do echo %%i
pause
```

运行本例发现枚举了C盘所有目录，为了只列举存在boot.ini的目录，可改成下面这样：

```
@echo off
for /r c:\ %%i in (boot.ini) do if exist %%i echo %%i
pause
```

用这条命令搜索文件真不错。

3. /l

```
FOR /L %variable IN (start,step,end) DO command [command-paramet
ers]
```

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)。

使用迭代变量设置起始值(Start#)，然后逐步执行一组范围的值，直到该值超过所设置的终止值(End#)。 `/L` 将通过对Start#与End#进行比较来执行迭代变量。如果Start#小于End#，就会执行该命令。如果迭代变量超过End#，则命令解释程序退出

此循环。还可以使用负的Step#以递减数值的方式逐步执行此范围内的值。例如，(1,1,5)生成序列(1 2 3 4 5)，而(5,-1,1)则生成序列(5 4 3 2 1)。

例：

```
@echo off
for /l %%i in (1,1,5) do @echo %%i
pause
```

保存执行看效果：

```
1
2
3
4
5
请按任意键继续...
```

(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!

例：

```
@echo off
for /l %%i in (1,1,5) do start cmd
pause
```

执行后是不是吓了一跳，怎么多了5个CMD窗口！如果把那个(1,1,5)改成(1,1,65535)会有什么结果，我先告诉大家，会打开65535个CMD窗口...这么多你不死机算你强！当然我们也可以把那个 `start cmd` 改成 `md %i` 这样就会建立指定个目录了，名字为1-65535。

4. /f

迭代及文件解析

使用文件解析来处理命令输出、字符串及文件内容。使用迭代变量定义要检查的内容或字符串，并使用各种options选项进一步修改解析方式。使用options令牌选项指定哪些令牌应该作为迭代变量传递。请注意：在没有使用令牌选项时，`/F` 将只检查第一个令牌。文件解析过程包括读取输出、字符串或文件内容，将其分成独立的文本行以及再将每行解析成零个或更多个令牌。然后通过设置为令牌的迭代变量值，调用for循环。默认情况下，`/F` 传递每个文件每一行的第一个空白分隔符号。跳过空行。

详细的帮助格式为：

```
FOR /F ["options"] %variable IN (file-set) DO command [command-p
arameters]
FOR /F ["options"] %variable IN ("string") DO command [command-p
arameters]
FOR /F ["options"] %variable IN ('command') DO command [command-
parameters]
```

带引号的字符串 `"options"` 包括一个或多个指定不同解析选项的关键字。这些关键字为：

eol=c 指一个行注释字符的结尾(就一个)

skip=n 指在文件开始时忽略的行数。

delims=xxx 指分隔符集。这个替换了空格和制表符的默认分隔符集。

tokens=x,y,m-n 指每行的哪一个符号被传递到每个迭代的for本身。这会导致额外变量名称的分配。m-n格式为一个范围。通过nth符号指定mth。如果符号字符串中的最后一个字符星号，那么额外的变量将在最后一个符号解析之后分配并接受行的保留文本。

usebackq 使用后引号（键盘上数字1左边的那个键`）。未使用参数usebackq 时：file-set表示文件，但不能含有空格；双引号表示字符串，即"string"；单引号表示执行命令，即'command'。使用参数usebackq时：file-set和"file-set"都表示文件，当文件路径或名称中有空格时，就可以用双引号括起来；单引号表示字符串，即'string'；后引号表示命令执行，即`command`

For命令例1：

```
@echo off
rem 首先建立临时文件 test.txt
echo ;注释行,这是临时文件,用完删除 >test.txt
echo 11 段 12 段 13 段 14 段 15 段 16 段 >>test.txt
echo 21 段,22 段,23 段,24 段,25 段,26 段 >>test.txt
echo 31 段-32 段-33 段-34 段-35 段-36 段 >>test.txt
FOR /F "eol=; tokens=1,3* delims=,- " %%i in (test.txt) do echo
%%i %%j %%k
Pause
Del test.txt
```

运行显示结果：

```
11 12 段 13 段 14 段 15 段 16 段
21 22 段,23 段,24 段,25 段,26 段
31 32 段-33 段-34 段-35 段-36 段
请按任意键继续...
```

解释：

`eol=;` 分号开头的行为注释行 `tokens=1,3*` 将每行第1段,第3段和剩余字段分别赋予变量%%i，%%j，%%k `delims=,-` （减号后有一空格）以逗号减号和空格为分隔符，空格必须放在最后

笔者注：如果不好理解可以更改相应值来观察编号。

For命令例2：

```
@echo off
FOR /F "eol= delims=" %%i in (test.txt) do echo %%i
Pause
```

运行将显示test.txt全部内容，包括注释行。

另外 `/F` 参数还可以输出命令的结果，For命令例3：

```
@echo off
FOR /F "delims=" %%i in ('net user') do @echo %%i
pause
```

这样你本机全部帐号名字就出来了。把扩号内的内容用两个单引号引起来就表示那个当命令执行，FOR会返回命令的每行结果，加"delims="是为了让含有空格的行能整行显示出来，不加就只显示空格左边一列！

2.2. FOR命令中的变量

先把FOR的变量全部列出来：

1. ~I 删除任何引号(")，扩展 %I
2. %~fI 将 %I 扩展到一个完全合格的路径名
3. %~dI 仅将 %I 扩展到一个驱动器号
4. %~pI 仅将 %I 扩展到一个路径
5. %~nI 仅将 %I 扩展到一个文件名
6. %~xI 仅将 %I 扩展到一个文件扩展名
7. %~sI 扩展的路径只含有短名
8. %~aI 将 %I 扩展到文件的文件属性
9. %~tI 将 %I 扩展到文件的日期/时间
10. %~zI 将 %I 扩展到文件的大小
11. %~$PATH:I 查找列在路径环境变量的目录，并将 %I 扩展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，此组合键会扩展到空字符串

我们可以看到每行都有一个大写字母“I”,这个I其实就是我们在FOR带入的变量,我们FOR语句代入的变量名是什么,这里就写什么.

比如:

```
FOR /F %%z IN ('set') DO @echo %%z
```

这里我们代入的变量名是z那么我们就要把那个I改成z，例如 `%~fI` 改为 `%~fz` 至于前面的 `%~p` 这样的内容就是语法了！

1. ~I

删除任何引号(")，扩展 %I

这个变量的作用就如他的说明，删除引号！

我们来看这个例子：

首先建立临时文件temp.txt，内容如下：

```
"1111
"2222"
3333"
"4444"44
"55"55"55
```

建立一个BAT文件代码如下：

```
@echo off
echo ^"1111>temp.txt
echo "2222">>temp.txt
echo 3333^">>temp.txt
echo "4444"44>>temp.txt
echo ^"55"55"55>>temp.txt
rem 上面建立临时文件，注意不成对的引号要加转义字符^，重定向符号前不要留空格
FOR /F "delims=" %%i IN (temp.txt) DO echo %%~i
pause
del temp.txt
```

执行后，我们看CMD的回显如下：

```
1111       #字符串前的引号被删除了
2222       #字符串首尾的引号都被删除了
3333"      #字符串前无引号，后面的引号保留
4444"44    #字符串前面的引号删除了，而中间的引号保留
55"55"55   #字符串前面的引号删除了，而中间的引号保留
请按任意键继续...
```

和之前temp.txt中的内容对比一下，我们会发现第1、2、4、5行的引号都消失了，这就是删除引号~i的作用了！

删除引号规则如下：

1. 若字符串首尾同时存在引号，则删除首尾的引号；
2. 若字符串尾不存在引号，则删除字符串首的引号；
3. 如果字符串中间存在引号，或者只在尾部存在引号，则不删除。

2. %~fI

将 %I 扩展到一个完全合格的路径名

例：

把test.bat保存放在D:\，内容如下：

```
@echo off
FOR /F "delims==" %%i IN ('dir /b') DO @echo %%~fi
pause
```

执行后显示内容如下：

```
D:\test.bat
请按任意键继续...
```

当我把代码中的 `%~fi` 直接改成 `%i`

```
@echo off
FOR /F "delims==" %%i IN ('dir /b') DO @echo %%i
pause
```

执行后就会显示以下内容：

```
test.bat
请按任意键继续...
```

通过对比，我们很容易就看出没有路径了，这就是“将 %I 扩展到一个完全合格的路径名”的作用。

也就是如果 `%i` 变量的内容是一个文件名的话，他就会把这个文件所在的绝对路径打印出来，而不只单单打印一个文件名，自己动手动实验下就知道了。

3. %~dI

仅将 %I 扩展到一个驱动器号

例：

代码如下，放到桌面执行。

```
@echo off
FOR /F "delims==" %%i IN ('dir /b') DO @echo %%~di
pause
```

执行后CMD里显示如下：

```
C:
请按任意键继续...
```

桌面就一个文件test.bat，`%%~di`作用是，如果变量`%%i`的内容是一个文件或者目录名，他就会把他这文件或者目录所在的盘符号打印出来。

4. %~pI

仅将 %I 扩展到一个路径

这个用法和上面一样，他只打印路径不打印文件名字。

```
@echo off
FOR /F "delims==" %%i IN ('dir /b') DO @echo %%~pi
pause
```

自己复制代码看结果，下面几个都是这么个用法，代码给出来，大家自己看结果吧。

5. %~nI

仅将 %I 扩展到一个文件名

只打印文件名字：

```
@echo off
FOR /F "delims==" %%i IN ('dir /b') DO @echo %%~ni
pause
```

6. %~xI

仅将 %I 扩展到一个文件扩展名

只打印文件的扩展名：

```
@echo off
FOR /F "delims==" %%i IN ('dir /b') DO @echo %%~xi
pause
```

7. %~sI

扩展的路径只含有短名

打印绝对路径短文件名（短路径名）：

```
@echo off
FOR /F "delims==" %%i IN ('dir /b') DO @echo %%~si
pause
```

8. %~aI

将 %I 扩展到文件的文件属性

打印文件的属性：

```
@echo off
FOR /F "delims==" %%i IN ('dir /b') DO @echo %%~ai
pause
```

9. %~tI

将 %I 扩展到文件的日期/时间

打印文件建立的日期：

```
@echo off
FOR /F "delims==" %%i IN ('dir /b') DO @echo %%~ti
pause
```

10. %~zI

将 %I 扩展到文件的大小

打印文件的大小：

```
@echo off
FOR /F "delims==" %%i IN ('dir /b') DO @echo %%~zi
pause
```

补充：上面例子中的"delims=="可以改为"delims="，即不要分隔符。

11. %~$PATH:I

查找列在路径环境变量的目录，并将 %I 扩展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，此组合键会扩展到空字符串

这是最后一个，和上面那些都不一样。

例：

```
@echo off
FOR /F "delims=" %%i IN ("notepad.exe") DO echo %%~$PATH:i
pause
```

上面代码显示结果为

```
C:\Windows\System32\notepad.exe
请按任意键继续...
```

意思是在PATH变量里指定的路径里搜索notepad.exe文件，如果有notepad.exe则会把他所在绝对路径打印出来，没有就打印一个错误！

3. 批处理中的变量

批处理中的变量，我把他分为两类，分别为“系统变量”和“自定义变量”

我们现在来详解这两种变量。

3.1. 系统变量

他们的值由系统将其根据事先定义的条件自动赋值，也就是这些变量系统已经给他们定义了值，不需要我们来给他赋值，我们只需要调用。把他们全部列出来：

- %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。

这么多系统变量，我们如何知道他的值是什么呢？

在CMD里输入 `echo %WINDIR%` ，就能显示一个变量的值了。

举个实际例子，比如我们要复制文件到当前帐号的启动目录里就可以这样：

```
copy d:\1.bat "%USERPROFILE%\「开始」菜单\程序\启动\"
```

注意有空格的目录要用引号引起来。

笔者注：系统版本不同，以上目录可能有所不同。

另外还有一些系统变量，他们是代表一个意思，或者一个操作：

他们分别是 `%0` `%1` `%2` `%3` `%4` `%5` ...一直到 `%9` 还有一个 `%*`

- %0 这个有点特殊，有几层意思，先讲 %1-%9 的意思
- %1 返回批处理的第一个参数
- %2 返回批处理的第二个参数
- %3-%9 依此推类

反回批处理参数，到底怎么个返回法？

看这个例子，把下面的代码保存为test.bat然后放到C盘下：

```
@echo off
echo %1 %2 %3 %4
echo %1
echo %2
echo %3
echo %4
```

进入CMD，输入 `cd c:\`

然后输入 `test.bat 参数1 参数2 参数3 参数4`

注意中间的空格，我们会看到这样的结果：

```
参数1 参数2 参数3 参数4
参数1
参数2
参数3
参数4
```

对比下代码, `%1` 就是“参数1”， `%2` 就是“参数2”以此类推。

这些 `%1` 和 `%9` 可以让批处理也能带参数运行，大大提高批处理功能。

还有一个 `%*` ，作用是返回参数，不过它是一次返回全部参数的值，不用在输入 `%1` `%2` 来确定一个一个的值。例：

```
@echo off
echo %*
```

同样保存为test.bat放到C盘

进入CMD，输入 `cd c:\`

然后输入 `test.bat 参数1 参数2 参数3 参数4`

可以看到他一次把全部参数都显示出来了

现在讲比较特殊的 `%0`

`%0` 这个不是返回参数的值了，他有两层意思：

第一层意思:返回批处理所在绝对路径

例：

```
@echo off
echo %0
pause
```

保存为test.bat放在D:\运行，会显示如下结果：

```
D:\test.bat
请按任意键继续
```

它把当前批处理执行的所在路经打印出来了，这就是返回批处理所在绝对路径的意思。

第二层意思:无限循环执行**BAT**

例：

```
@echo off
net user
%0
```

保存为BAT执行，他就会无限循环执行net user这条命令，直到你手动停止。

其实 `%0` 就是第一参数 `%1` 前面那个参数，当然就是批处理文件名（包括路径）。

以上就是批处理中的一些系统变量，另外还有一些变量，他们也表示一些功能，FOR命令中的那些就是，FOR变量已经说过，就不讲了。

3.2. 自定义变量

顾名思义：自定义变量就是由我们自己来给他赋予值的变量

要使用自定义变量就得使用set命令了，看例子：

```
@echo off
set var=我是值
echo %var%
pause
```

执行结果：

```
我是值
请按任意键继续...
```

var为变量名，=号右边是给变量的值。

这是最简单的一种设置变量的方法。

如果我们想让用户手工输入变量的值，而不是在代码里指定，可以用用set命令的 `/p` 参数。

例子:

```
@echo off
set /p var=请输入变量的值:
echo 您输入的值是:%var%
pause
```

var为变量名，=号右边是提示语，不是变量的值

变量的值是批处理运行后自己用键盘输入。

4. SET命令详解

可以使用 `set /?` 查看SET的帮助。

在上一章中简单的介绍了一下SET设置自定义变量的作用，现在具体讲一下SET的功能。

4.1. 用SET命令设置自定义变量

显示、设置或删除 cmd.exe 环境变量。

```
SET [variable=[string]]
```

variable 指定环境变量名。

string 指定要指派给变量的一系列字符串。

要显示当前环境变量，键入不带参数的SET。

SET命令不允许变量名含有等号。

例：

```
@echo off
set var=我是值
echo %var%
pause
```

请看 `set var=我是值` ，这就是在批处理中设置变量的方法。

`set` 是命令， `var` 是变量名，=号右边的 `我是值` 是变量的值。

在批处理中我们要引用这个量就把变量名用两个%（百分号）扩起来,如 `%var%`

SET还可以提供一个交互界面，让用户自己输入变量的值，然后再根据这个值来做相应操作，SET的这种语法，只需要加一个 `/P` 参数就可以了。

```
SET /P variable=[promptString]
```

例：

```
@echo off
set /p var=请输入变量的值:
echo 您输入的值是:%var%
pause
```

`set /p` 是命令语法， `var` 是变量名，=号右边的 `请输入变量的值:` 是提示语，不是变量的值。

运行后，我们在提示语后面直接输入 `1` ，显示结果：

```
请输入变量的值:1
您输入的值是:1
请按任意键继续...
```

4.2. 用SET命令进行简单计算

语法：

```
SET /A expression
```

`/A` 命令行开关指定等号右边的字符串为被评估的数字表达式。该表达式 评估器很简单并以递减的优先权顺序支持下列操作：

```
()                    - 分组
! ~ -                 - 一元运算符
* / %                 - 算数运算符
+ -                   - 算数运算符
<< >>                 - 逻辑移位
&                     - 按位“与”
^                     - 按位“异”
|                     - 按位“或”
= *= /= %= += -=      - 算数赋值
&= ^= |= <<= >>=      - 二进制运算赋值
,                     - 表达式分隔符
```

下面简单解释一下：

SET的 `/A` 参数就是让SET可以支持数学符号进行加减等一些数学运算。

例：

```
@echo off
set /p input=请输入计算表达式:
set /a var=%input%
echo 计算结果:%input%=%var%
pause
```

注意：**DOS**计算只能精确到整数

请看下面几个运算过程：

```
请输入计算表达式:1+9+20+30-10
计算结果:1+9+20+30-10=50
请按任意键继续...
```

```
请输入计算表达式:10/3
计算结果:10/3=3   #DOS计算精确到整数，小数舍了。
请按任意键继续...
```

```
请输入计算表达式:-100+62
计算结果:-100+62=-38
请按任意键继续...
```

```
请输入计算表达式:100%3  #求余数
计算结果:100%3=1
请按任意键继续...
```

```
请输入计算表达式:(25+75)*2/(15+5)
计算结果:(25+75)*2/(15+5)=10
请按任意键继续...
```

```
请输入计算表达式:1234567890*9876543210
无效数字。数字精确度限为 32 位。
计算结果:1234567890*9876543210=
请按任意键继续...
```

注意：上面的计算过程显示，DOS计算只能精确到32位，这个32位是指二进制32位，其中最高位为符号位（0为正，1为负），低位31位为数值。31个1换成十进制为2147483647，所以DOS计算的有效值范围是-2147483647至2147483647，超出该数值范围时计算出错，请看下面的计算过程：

```
请输入计算表达式:2147483647-1  #最大值减1，值有效
计算结果:2147483647-1=2147483646
请按任意键继续...
```

运行：

```
set /a a=1+1,b=2+1,c=3+1
```

后会显示一个 `4` （笔者注：好像打开批处理文件不能显示 `4` ，需要在CMD窗口直接输入命令来观察）。但我们用 `echo %a% %b% %c%` 后看结果，会发现其他数学运算也有效果，这就是逗号 `,` 的作用。

有时候我们需要直接在原变量进行加减操作就可以用这种语法：

```
set /a var+=1
```

这样的语法对应原始语法就是：

```
set /a var = %var% + 1
```

都是一样的结果，以原变量的值为初始值再进行数学运算，不过这样写简单一点。只要帮助里有这个语法，都可以这样用，比如：`set /a var*=2`。

另外还有一些用逻辑或取余操作符，这些符号，按照上面的使用方法会报错的。

比如我们在CMD里输入`set /a var=1 & 1`（与运算），并不会显示为`1`，而是报错。

为什么？对于这样的“逻辑或取余操作符”，我们需要把他们用双引号`"`引起来，也可以用转义字符`^`，例：

与运算：

```
@echo off
set /a var= 1 "&" 1
echo %var%
pause
```

其他逻辑或取余操作符用法：

异运算：

```
set /a var= 1 "^" 1
```

取模运算：（笔者注：批处理文件中`"%"`要写为`"%%"`）

```
set /a var= 1 "%" 1
```

左移位运算，3的二进制为11，左移2位为1100，换成十进制就是12：

```
set /a var= 3 "<<" 2
```

右移位运算，4 的二进制为 100，右移动 2 位为 1，结果为1：

```
set /a var= 4 ">>" 2
```

凡是按位计算均需换算成二进制，下面行中的符号均针对二进制这些符号也可以用 `&=` `^=` `|=` `<<=` `>>=` 。这样的简单用法，如：（注意引号）

```
set /a var"&=" 1
```

相当于：

```
set /a var = %var% "&" 1
```

思考题：求2的n次方

参考答案：

```
@echo off
set /p n=请输入2的几次方:
set /a num=1^<^<n
echo %num%
pause
```

（ `set /?` 有一段描述：在表达式中的任何非数字字符串键作为环境变量名称，这些环境变量名称的值已在使用前转换成数字。如果指定了一个环境变量名称，但未在当前环境中定义，那么值将被定为零。这使你可以使用环境变量值做计算而不用键入那些 % 符号来得到它们的值。即是此例中 `set /a num=1^<^<n` 的 `n` 不需要写成 `%n%` 。）

或：

```
@echo off
set /p n=请输入2的几次方:
set /a num=1 "<<" n
echo 2的%n%次方为:%num%
pause
```

4.3. 用命令进行字符串处理

4.3.1. 字符串替换

语法：

```
%PATH:str1=str2%
```

意思是：将字符串变量 `%PATH%` 中的 `str1` 替换为 `str2`

这个是替换变量值的内容，看例子：

```
@echo off
set a= www. google. com
echo 替换前的值:"%a%"
set var=%a: =%
echo 替换后的值:"%var%"
pause
```

运行显示：

```
替换前的值:" www. google. com"
替换后的值:"www.google.com"
请按任意键继续...
```

对比一下，发现它是把变量 `%a%` 的空格给替换掉了，从这个例子，我们就可以发现 `%PATH:str1=str2%` 这个操作就是把变量 `%PATH%` 的里的 `str1` 全部用 `str2` 替换。

把上面的例子改成这样：

```
@echo off
set a=www.google.com
echo 替换前的值:"%a%"
set var=%a:.=测试%
echo 替换后的值:"%var%"
pause
```

运行显示：

```
替换前的值:"www.google.com"
替换后的值:"www测试google测试com"
请按任意键继续...
```

`set` 是命令，`var` 是变量名，`a` 是要进行字符替换的变量的值，`.` 为要替换的值，`测试` 为替换后的值。

执行后就会把变量 `%a%` 里面的 `.` 全部替换为 `测试` 。

这就是SET的替换字符功能。

4.3.2. 字符串截取

语法：

```
%a:~[m[,n]]%
```

方括号 `[]` 表示可选，`%` 为变量标识符，`a` 为变量名（不可少），冒号 `:` 用于分隔变量名和说明部分，符号 `~` 可以简单理解为“偏移”即可，m为偏移量（缺省为0），n为截取长度（缺省为全部）。

`%PATH:~10,5%` 解释看例1。

例1：

```
@echo off
set a=www.google.com
set var=%a:~3,2%
echo %var%
pause
```

运行结果：

```
.g
请按任意键继续...
```

分析 `set var=%a:~3,2%` ：

`set` 是命令， `var` 是变量值， `a` 是要进行字符操作的变量， `3` 是从变量 `a` 的第几位开始显示（首位从 `0` 开始计算）， `2` 表示显示几位。

合起来就是把变量 `a` 的值从第4位（偏移量3）开始,把2个字符赋予给变量 `var` 。

其他两种语法

```
%PATH:~-10%
%PATH:~0,-2%
```

`%PATH:~-10%` 解释看例2，例3。

例2：

```
@echo off
set a=www.google.com
set var=%a:~-4%
echo %var%
pause
```

运行结果：

```
.com
请按任意键继续...
```

这个就是把变量 `a` 倒数 `4` 位的值给变量 `var` 。

例3：

```
@echo off
set a=www.google.com
set var=%a:~3%
echo %var%
pause
```

运行结果：

```
.google.com
请按任意键继续...
```

这个就是把变量 `a` 从第 `3` 位开始后面全部的值给变量 `var` 。

`%PATH:~0,-2%` 解释看例4，例5。

例4：

```
@echo off
set a=www.google.com
set var=%a:~0,-4%
echo %var%
pause
```

运行结果：

```
www.google
请按任意键继续...
```

从结果分析，分析出：这是把变量 `a` 的值从 `0` 位开始，到倒数第 `4` 位之前的值全部赋予给 `var` 。

如果改成这样，例5：

```
@echo off
set a=www.google.com
set var=%a:~2,-3%
echo %var%
pause
```

运行结果：

```
w.google.
请按任意键继续...
```

就是显示从第3位（偏移量2）开始减去倒数三位字符的值赋给变量 `var` 。

小结如下：

```
a=www.google.com

%a:~3,2%  = ".g"            #偏移量3，从第四位向右取2位
%a:~-4%   = ".com"          #偏移量-4，截取倒数4位（也可理解为从右边开始截取4位）
%a:~3%    = ".google.com"   #偏移量3，可理解为去掉左边3位，右取全部
%a:~0,-4% = "www.google"    #偏移量0，右取长度至-4，即倒数4位
%a:~2,-3% = "w.google."     #偏移量2，右取长度至-3，即倒数3位
```

上面所述用法其实相当于 vbs 函数 mid、left、right

```
%a:~0,n%  相当于函数left(a,n)，取左边n位
%a:~-m%   相当于函数right(a,m)，取右边m位
%a:~m,n%  相当于函数mid(a,m+1,n)，从m+1位开始取n位
%a:~m,-n% 相当于函数mid(a,m+1,len(a)-m-n)
%a:~m %   相当于函数mid(a,m+1,len(a)-m)或者right(a,len(a)-m)
```

思考题：输入任意字符串，求字符串的长度

参考答案：

```
@echo off
set /p str=请输入任意长度的字符串:
echo 你输入了字符串:"%str%"
if not defined str (pause & goto :eof)
set num=0
:len
set /a num+=1
set str=%str:~0,-1%     #也可以%str:~1%
if defined str goto len
echo 字符串长度为：%num%
pause
```

5. IF命令讲解

可以使用 `if /?` 查看IF的帮助。

IF有三种基本的用法。

执行批处理程序中的条件处理。

```
IF [NOT] ERRORLEVEL number command
IF [NOT] string1==string2 command
IF [NOT] EXIST filename command
```

NOT 指定只有条件为false的情况下，Windows才应该执行该命令。

ERRORLEVEL number 如果最后运行的程序返回一个等于或大于指定数字的退出代码，指定条件为true。

string1==string2 如果指定的文字字符串匹配，指定条件为 true。

EXIST filename 如果指定的文件名存在，指定条件为 true。

command 如果符合条件，指定要执行的命令。如果指定的条件为FALSE，命令后可跟ELSE命令，该命令将在ELSE关键字之后执行该命令。

ELSE 子句必须出现在同一行上的IF之后。例如：

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

5.1. ERRORLEVEL

```
IF [NOT] ERRORLEVEL number command
```

基本作用是判断上一条命令执行结果的代码，以决定下一个步骤。

例：

```
@echo off
net user
IF %ERRORLEVEL%==0 echo net user 执行成功了!
pause
```

这是个简单判断上条命令是否执行成功。

这个用法和帮助里的用法不太一样，按照帮助里的写法 `IF %ERRORLEVEL% == 0 echo net user 执行成功了!` 这一句应该写成： `IF ERRORLEVEL 0 echo net user 执行成功了!` 。但用这种语法，不管你的上面的命令是否执行成功，都会显示“执行成功”。

这是 `IF ERRORLEVEL` 语句的特点：当使用 `IF ERRORLEVEL 0` 的句式时，它的含义是：如果错误码的值大于或等于0的时候，将执行某个操作；当使用 `IF %ERRORLEVEL%==0` 的句式时，它的含义是：如果错误码的值等于0的时候，将执行某个操作。因为这两种句式含义的差别，如果使用前一种句式的时候，错误码语句的排列顺序是从大到小排列。 `%ERRORLEVEL%` 是系统变量，返回上条命令的执行结果代码。“成功”用0表示，“失败”用1表示。当然还有其他参数，用的时候基本就这两数字。

这只是一般的情况，实际上，ERRORLEVEL返回值可以在0~255之间。比如：xcopy默认的ERRORLEVEL值就有5个，分别表示5种执行状态：

```
退出码 说明
0       文件复制没有错误。
1       if errorlevel 2 echo。
2       用户按 CTRL+C 终止了 xcopy。
4       出现了初始化错误。没有足够的内存或磁盘空间，或命令行上输入了无效的驱动器名称或语法。
5       出现了磁盘写入错误。
```

要判断上面xcopy命令的5种退出情况，应写成：

```
if errorlevel 5 echo 出现了磁盘写入错误
if errorlevel 4 echo 出现了初始化错误
if errorlevel 2 echo 用户按 CTRL+C 终止了 xcopy
if errorlevel 1 echo if errorlevel 2 echo
if errorlevel 0 echo 文件复制没有错误。
```

才能正确执行。

再举几个例子：

```
@echo off
net usertest
IF %ERRORLEVEL%==1 echo net user 执行失败了!
pause
```

判断上一条命令是否执行失败的。

```
@echo off
set /p var=随便输入一个命令:
%var%
IF %ERRORLEVEL%==0 goto yes
goto no
:yes
echo "%var%"执行成功了.
pause
:no
echo "%var%"执行失败了.
pause
```

判断你输入的命令是成功还是失败了。

简化版：

```
@echo off
set /p var=随便输入一个命令:
%var%
IF %ERRORLEVEL%==0 (echo "%var%"执行成功了.) ELSE echo "%var%"执行失败了.
pause
```

`ELSE` 后面写上执行失败后的操作。

当然我门还可以把IF ELSE这样的语句分成几行写出来,使他看上去好看点：

```
@echo off
set /p var=随便输入一个命令:
%var%
IF %ERRORLEVEL%==0 (
echo "%var%"执行成功了.
) ELSE (
echo "%var%"执行失败了.
)
pause
```

这里介绍的两种简写对IF的三种语法都可以套用,不只是在 `IF [NOT] ERRORLEVEL number command` 这种语法上才能使用。

5.2. string

```
IF [NOT] string1==string2 command
```

用来比较变量或者字符的值是不是相等的。

例：

```
@echo off
set /p var=请输入第一个比较字符:
set /p var2=请输入第二个比较字符:
IF %var% == %var2% (echo 我们相等) ELSE echo 我们不相等
pause
```

上面这个例子可以判断你输入的值是不是相等。但是如果输入相同的字符，而其中一个后面带一个空格，还是会认为相等，如何让有空格的输入不相等呢？我们在比较字符上加个双引号就可以了：

```
@echo off
set /p var=请输入第一个比较字符:
set /p var2=请输入第二个比较字符:
IF "%var%" == "%var2%" (echo 我们相等) ELSE echo 我们不相等
pause
```

5.3. EXIST

```
IF [NOT] EXIST filename command
```

判断某个文件或者文件夹是否存在的语法

例：

```
@echo off
IF EXIST "c:\test" (echo 存在文件) ELSE echo 不存在文件
pause
```

判断的文件路径加引号是为了防止路径有空格，如果路径有空格加个双引号就不会出现判断出错了。

另外我们看到每条IF用法后都有个 `[NOT]` 语句，加上的话，表示先判断我们的条件不成立时，没加他默认是先判断条件成立时，比如上面这个例改为：

```
@echo off
IF NOT EXIST "c:\test" (echo 存在文件) ELSE echo 不存在文件
pause
```

执行后，如果C盘下没有test,他还是会显示 `存在文件` ，这就表示加了NOT就 会先判断条件失败的情况。上面例子改成这样就正确了：

```
@echo off
IF NOT EXIST "c:\test" (echo 不存在文件) ELSE echo 存在文件
pause
```

5.4. IF的增强用法

```
IF [/I] string1 compare-op string2 command
IF CMDEXTVERSION number command
IF DEFINED variable command
```

后面两个用法，因为它们和上面的用法表示的意义基本一样，只简单说说。

`IF [/I] string1 compare-op string2 command` 这个语句在判断字符时不区分字符的大小写。

`CMDEXTVERSION` 条件的作用跟ERRORLEVEL的一样，除了它是在跟与命令扩展有关联的内部版本号比较。第一个版本是1。每次对命令扩展有相当大的增强时，版本号会增加一个。命令扩展被停用时，CMDEXTVERSION条件不是眞的。

如果已定义环境变量，`DEFINED`条件的作用跟EXIST的一样，除了它取得一个环境变量，返回的结果是true。

例：

```
@echo off
if a == A (echo 我们相等) ELSE echo 我们不相等
pause
```

执行后显示：

```
我们不相等
请按任意键继续...
```

加上`/I`不区分大小写就相等了。

```
@echo off
if /i a == A (echo 我们相等) ELSE echo 我们不相等
pause
```

下面是一些用来判断数字的符号：

```
EQU - 等于
NEQ - 不等于
LSS - 小于
LEQ - 小于或等于
GTR - 大于
GEQ - 大于或等于
```

例：

```
@echo off
set /p var=请输入一个数字:
if %var% LEQ 4 (echo 我小于等于4) ELSE echo 我不小于等于4
pause
```

6. DOS编程高级技巧

6.1. 交互界面设计

例：

```
@echo off
cls
title 终极多功能修复
:menu
cls
color 0A
echo.
echo ================================
echo 请选择要进行的操作，然后按回车
echo ================================
echo.
echo 1.网络修复及上网相关设置,修复 IE,自定义屏蔽网站
echo.
echo 2.病毒专杀工具，端口关闭工具,关闭自动播放
echo.
echo 3.清除所有多余的自启动项目，修复系统错误
echo.
echo 4.清理系统垃圾,提高启动速度
```

```
echo.
echo Q.退出
echo.
echo.
:cho
set choice=
set /p choice= 请选择:
IF NOT "%choice%"=="" SET choice=%choice:~0,1%
if /i "%choice%"=="1" goto ip
if /i "%choice%"=="2" goto setsave
if /i "%choice%"=="3" goto kaiji
if /i "%choice%"=="4" goto clean
if /i "%choice%"=="Q" goto end
echo 选择无效，请重新输入
echo.
goto cho
:ip
echo 您选择了 1
pause&goto menu
:setsave
echo 您选择了 2
pause&goto menu
:kaiji
echo 您选择了 3
pause&goto menu
:clean
echo 您选择了 4
pause&goto menu
:end
echo 点击任意键退出&pause>nul&goto :eof
```

只要学完本教程前面的章节，上面的程序应该能看懂了。

6.2. IF...ELSE...条件语句

前面已经谈到，DOS 条件语句主要有以下形式

```
IF [NOT] ERRORLEVEL number command
IF [NOT] string1==string2 command
IF [NOT] EXIST filename command
```

增强用法：

```
IF [/I] string1 compare-op string2 command
```

增强用法中加上 `/I` 就不区分大小写了。

增强用法中还有一些用来判断数字的符号：

```
EQU - 等于
NEQ - 不等于
LSS - 小于
LEQ - 小于或等于
GTR - 大于
GEQ - 大于或等于
```

上面的 `command` 命令都可以用小括号来使用多条命令的组合，包括ELSE子句。组合命令中可以嵌套使用条件或循环命令。

例：

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

也可写成：

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

但这种写法不适合命令太多或嵌套命令的使用。

6.3. 循环语句

6.3.1. 指定次数循环

```
FOR /L %variable IN (start,step,end) DO command [command-paramet
ers]
```

组合命令：

```
FOR /L %variable IN (start,step,end) DO (
Command1
Command2
......
commandN
)
```

6.3.2. 对某集合执行循环语句

```
FOR %%variable IN (set) DO command [command-parameters]
```

`%%variable` 指定一个单一字母可替换的参数。

`(set)` 指定一个或一组文件。可以使用通配符。

`command` 对每个文件执行的命令，可用小括号使用多条命令组合。

```
FOR /R [[drive:]path] %variable IN (set) DO command [command-par
ameters]
```

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。同前面一样， `command` 可以用括号来组合：

```
FOR /R [[drive:]path] %variable IN (set) DO (
Command1
Command2
......
commandN
)
```

6.3.3. 条件循环

利用goto语句和条件判断，DOS可以实现条件循环，例：

```
@echo off
set var=0
rem ************循环开始了
:continue
set /a var+=1
echo 第%var%次循环
if %var% lss 100 goto continue
rem ************循环结束了
echo 循环执行完毕
pause
```

6.4. 子程序

在批处理程序中可以调用外部可运行程序，比如exe程序，也可调用其他批处理程序，这些也可以看作子程序，但是不够方便，如果被调用的程序很多，就显得不够简明了，很繁琐。

批处理可以调用本程序中的一个程序段，相当于子程序，这些子程序一般放在主程序后面。

子程序调用格式：

```
CALL :label arguments
```

子程序语法：

```
:label
command1
command2
......
commandN
goto :eof
```

传至子程序的参数在CALL语句中指定，在子程序中用%1、%2 至%9 的形式调用，而子程序返回主程序的数据只需在调用结束后直接引用就可以了，当然也可以指定返回变量，请看下面的例子。

子程序例1：

```
@echo off
call :sub return 你好
echo 子程序返回值:%return%
pause
:sub
set %1="%2"
rem %1即return，%2即你好
goto :eof
```

运行结果：

```
子程序返回值:"你好"
请按任意键继续...
```

子程序例2：（求多个整数相加的子程序）

```
@echo off
set sum=0
call :sub sum 10 20 35
echo 数据求和结果:%sum%
pause
:sub
rem 参数1为返回变量名称
set /a %1+=%2
shift /2
if not "%2"=="" goto sub
goto :eof
```

运行结果：

```
65
```

在Win98系统中，不支持上面这种标号调用，须将子程序单独保存为一个批处理程序，然后调用。

6.5. 用ftp命令实现自动下载

ftp是常用的下载工具，ftp界面中有40多个常用命令，不在这里介绍。这里介绍 如何用DOS命令行调用ftp命令，实现ftp自动登录，上传下载，并自动退出ftp程序。其实可以将ftp命令组合保存为一个文本文件，然后用以下命令调用即可。

```
ftp -n -s:[[drive:]path]filename
```

上面的 `filename` 为ftp命令文件，包括登录IP地址，用户名，密码，操作命令等

例：

```
open 90.52.8.3  #打开ip
user username   #用户为username
password1234    #密码
bin             #二进制传输模式
prompt
cd tmp1         #切换至username用户下的tmp1目录
pwd
lcd d:\download #本地目录
mget *          #下载tmp1目录下的所有文件
bye             #退出ftp
```

笔者注：现在很少用ftp了吧，就当参考思路。

6.6. 用7-ZIP实现命令行压缩和解压功能

语法格式：（详细情况见7-zip帮助文件，如果有难度可以先跳过，用到的时候再学）

```
7z <command> [<switch>...] <base_archive_name> [<arguments>...]
```

7z.exe的每个命令都有不同的参数 `<switch>` ，请看帮助文件

`<base_archive_name>` 为压缩包名称

`<arguments>` 为文件名称，支持通配符或文件列表

其中， `7z` 是指命令行压缩解压程序7z.exe， `<command>` 是7z.exe包含的命令，列举如下：

a：Adds files to archive. 添加至压缩包

a命令可用参数：

```
-i (Include)
-m (Method)
-p (Set Password)
-r (Recurse)
-sfx (create SFX)
-si(use StdIn)
-so (use StdOut)
-ssw (Compressshared files)
-t (Type of archive)
-u (Update)
-v (Volumes)
-w (Working Dir)
-x (Exclude)
```

b：Benchmark

d：Deletes files from archive. 从压缩包中删除文件

d命令可用参数：

```
-i (Include)
-m (Method)
-p (Set Password)
-r (Recurse)
-u (Update)
-w (Working Dir)
-x (Exclude)
```

e：Extract 解压文件至当前目录或指定目录

e命令可用参数：

```
-ai (Include archives)
-an (Disable parsing of archive_name)
-ao (Overwrite mode)
-ax (Exclude archives)
-i (Include)
-o (Set Output Directory)
-p (Set Password)
-r (Recurse)
-so (use StdOut)
-x (Exclude)
-y (Assume Yes on all queries)
```

l：Lists contents of archive.

t：Test

u：Update

x：eXtract with full paths 用文件的完整路径解压至当前目录或指定目录

x命令可用参数：

```
-ai (Include archives)
-an (Disable parsing of archive_name)
-ao (Overwrite mode)
-ax (Exclude archives)
-i (Include)
-o (Set Output Directory)
-p (Set Password)
-r (Recurse)
-so (use StdOut)
-x (Exclude)
-y (Assume Yes on all queries)
```

6.7. 调用 VBScript 程序

使用Windows脚本宿主，可以在命令提示符下运行脚本。CScript.exe提供了用于设置脚本属性的命令行开关。

用法：

```
CScript 脚本名称 [脚本选项...] [脚本参数...]
```

选项：

- //B 批模式：不显示脚本错误及提示信息
- //D 启用 Active Debugging
- //E:engine 使用执行脚本的引擎
- //H:CScript 将默认的脚本宿主改为 CScript.exe
- //H:WScript 将默认的脚本宿主改为 WScript.exe （默认）
- //I 交互模式（默认，与 //B 相对)
- //Job:xxxx 执行一个 WSF 工作
- //Logo 显示徽标（默认）
- //Nologo 不显示徽标：执行时不显示标志
- //S 为该用户保存当前命令行选项
- //T:nn 超时设定秒：允许脚本运行的最长时间
- //X 在调试器中执行脚本
- //U 用 Unicode 表示来自控制台的重定向 I/O

`脚本名称` 是带有扩展名和必需的路径信息的脚本文件名称，如d:\admin\vbscripts\chart.vbs。

`脚本选项和参数` 将传递给脚本。脚本参数前面有一个斜杠(/)。每个参数都是可选的；但不能在未指定脚本名称的情况下指定脚本选项。如果未指定参数，则CScript将显示CScript语法和有效的宿主参数。

6.8. 将批处理转化为可执行文件

由于批处理文件是一种文本文件，任何人都可以对其进行随便编辑，不小心就会把里面的命令破坏掉，所以如果将其转换成.com格式的可执行文件，不仅执行效率会大大提高，而且不会破坏原来的功能，更能将优先级提到最高。Bat2Com就可以完成这个转换工作。

小知识：在DOS环境下，可执行文件的优先级由高到低依次为.com>.exe>.bat>.cmd，即如果在同一目录下存在文件名相同的这四类文件，当只键入文件名时，DOS执行的是name.com，如果需要执行其他三个文件，则必须

指定文件的全名，如name.bat。

笔者注：未测试，也不知道生成的可执行文件会不会被安全软件阻止。

6.9. 时间延迟

时间延迟，就是执行一条命令后延迟一段时间再进行下一条命令。

6.9.1. 利用ping命令延时

例：

```
@echo off
echo 延时前...
ping /n 5 127.0.0.1 >nul
echo 延时后...
pause
```

解释：用到了ping命令的 `/n` 参数，表示要发送多少次请求到指定的ip。本例中要发送 `5` 次请求到本机的ip（127.0.0.1）。127.0.0.1可简写为127.1。 `>nul` 就是屏蔽掉ping命令所显示的内容。

6.9.2. 利用for命令延时

例：

```
@echo off
echo 延时前...
for /l %%i in (1,1,5000) do echo %%i>nul
echo 延时后...
pause
```

解释：利用一个计次循环并屏蔽它所显示的内容来达到延时的目的。

6.10. 模拟进度条

下面给出一个模拟进度条的程序。如果将它运用在你自己的程序中，可以使你的程序更漂亮。

```
@echo off
mode con cols=113 lines=15 &color f2
cls
echo.
echo 程序正在初始化. . .
echo.
echo ┌──────────────────────────────────────────────┐
set/p= ■<nul
for /L %%i in (1 1 38) do set /p a=■<nul&ping /n 1 127.0.0.1>nul
echo 100%%
echo └──────────────────────────────────────────────┘
pause
```

解释：`set /p a=■<nul` 的意思是：只显示提示信息 `■` 且不换行，也不需手工输入任何信息，这样可以使每个 `■` 在同一行逐个输出。`ping /n 0 127.1>nul` 是输出每个 `■` 的时间间隔，即每隔多少时间输出一个 `■` 。

6.11. 企业微信发送消息

先注册企业微信，创建一个应用，在应用的可见范围设置权限。

```
@echo off
::批处理文件编码需要存为UTF-8无BOM格式，中文消息推送才能正常
::填入企业ID
set CorpID=xxxx
::填入应用Secret
set Secret=xxxx
set GURL=https://qyapi.weixin.qq.com/cgi-bin/gettoken?corpid=%Co
rpID%^&corpsecret=%Secret%

::acccess_token
FOR /F tokens^=10^ delims^=^" %%i in ('curl -s -G "%GURL%"') do
set Token=%%i
set PURL="https://qyapi.weixin.qq.com/cgi-bin/message/send?acces
s_token=%Token%"

::填入应用AgentId
set wxAppID=1
::企业微信中部门成员ID(企业微信成员信息中称为帐号)
set wxUserID=1
::发送的消息内容
set wxMsg=测试
set Body={ \"touser\":\"%wxUserID%\", \"msgtype\":\"text\", \"ag
entid\":\"%wxAppID%\", \"text\":{\"content\":\"%wxMsg%\"}, \"saf
e\":\"0\" }

curl --data-ascii "%Body%" %PURL%

pause
```

一个应用实例：

系统Windows2008R2，可能系统不同，具体变量，参数或设置的地方有可能不同。

下载CURL，设置好系统变量，如果没有设置，将下面的curl命令替换为绝对路径。

因批处理文件编码需要存为UTF-8无BOM格式，中文消息推送才能正常，所以不要用自带的记事本编辑，可以使用Notepad++。

关机提醒脚本

新建 `shutdown-notify.bat` ：

```
@echo off&setlocal enabledelayedexpansion

set CorpID=
set Secret=
set GURL=https://qyapi.weixin.qq.com/cgi-bin/gettoken?corpid=%Co
rpID%^&corpsecret=%Secret%

FOR /F tokens^=10^ delims^=^" %%i in ('curl -s -G "%GURL%"') do
set Token=%%i
set PURL="https://qyapi.weixin.qq.com/cgi-bin/message/send?acces
s_token=%Token%"

set wxAppID=1
set wxUserID=1
FOR /F "tokens=1 delims= " %%i in ('echo %DATE%') do set RIQI=%%
i
FOR /F "tokens=1 delims=." %%i in ('echo %TIME%') do set SHIJIAN
=%%i
set wxMsg=服务器状态提醒：\n主机名：%COMPUTERNAME%\n状态类型：关机\n状
态时间：%RIQI% %SHIJIAN%
set Body={ \"touser\":\"%wxUserID%\", \"msgtype\":\"text\", \"ag
entid\":\"%wxAppID%\", \"text\":{\"content\":\"%wxMsg%\"}, \"saf
e\":\"0\" }

curl --data-ascii "%Body%" %PURL% >nul
exit
```

关机时执行脚本设置：

gpedit.msc -> 计算机配置 -> Windows设置 -> 脚本(启动/关机) -> 关机

设置完成后重启（该次关机不会执行），重新启动完成后再关机尝试。

开机提醒脚本

新建 `boot-notify.bat` ：

```
@echo off&setlocal enabledelayedexpansion

set CorpID=
set Secret=
set GURL=https://qyapi.weixin.qq.com/cgi-bin/gettoken?corpid=%Co
rpID%^&corpsecret=%Secret%

FOR /F tokens^=10^ delims^=^" %%i in ('curl -s -G "%GURL%"') do
set Token=%%i
set PURL="https://qyapi.weixin.qq.com/cgi-bin/message/send?acces
s_token=%Token%"

set wxAppID=1
set wxUserID=1
FOR /F "tokens=1 delims= " %%i in ('echo %DATE%') do set RIQI=%%
i
FOR /F "tokens=1 delims=." %%i in ('echo %TIME%') do set SHIJIAN
=%%i
set wxMsg=服务器状态提醒：\n主机名：%COMPUTERNAME%\n状态类型：开机\n状
态时间：%RIQI% %SHIJIAN%
set Body={ \"touser\":\"%wxUserID%\", \"msgtype\":\"text\", \"ag
entid\":\"%wxAppID%\", \"text\":{\"content\":\"%wxMsg%\"}, \"saf
e\":\"0\" }

curl --data-ascii "%Body%" %PURL% >nul
exit
```

启动时执行脚本设置：

与关机设置相同位置的“启动”那里添加后，并未生效，不知道什么原因（可能是该处设置时，执行脚本的优先级较高，在还未连上网络时就执行了），改为使用“任务计划程序”。

新建一个任务计划：

- 常规 -> 运行时使用帐户administrator -> 不管用户是否登录都要运行
- 触发器 -> 启动时

登录提醒脚本

新建 `login-notify.bat` ：

```
@echo off&setlocal enabledelayedexpansion
::批处理文件编码需要为UTF-8，中文消息推送才能正常

::设置为当前文件所在目录
set BAT_HOME=C:\weixin-notify

set CorpID=
set Secret=
set GURL=https://qyapi.weixin.qq.com/cgi-bin/gettoken?corpid=%Co
rpID%^&corpsecret=%Secret%

FOR /F tokens^=10^ delims^=^" %%i in ('curl -s -G "%GURL%"') do
set Token=%%i
set PURL="https://qyapi.weixin.qq.com/cgi-bin/message/send?acces
s_token=%Token%"

set wxAppID=1
set wxUserID=1
::根据系统日志来获取登录用户的用户名，不使用系统变量%USERNAME%是因为考虑用
户执行权限的问题，所以统一由administrator来执行脚本
for /f "delims=" %%a in ('WEVTUTIL qe Security ^| findstr winlog
on.exe') do (
  set "str=%%a"
  set str=!str:TargetUserName^'^>=#!
  for /f "tokens=2 delims=#" %%A in ("!str!") do (echo %%A>%BAT_
HOME%\usertmp.txt)
)
::同用户名，获取登录用户的IP地址(如果有更方便的获取方式可自行替换)
for /f "delims=" %%a in ('WEVTUTIL qe Security ^| findstr winlog
on.exe') do (
  set "str=%%a"
  set str=!str:IpAddress^'^>=#!
  for /f "tokens=2 delims=#" %%A in ("!str!") do (echo %%A>%BAT_
HOME%\iptmp.txt)
)
for /f "tokens=1 delims=<" %%i in ('type %BAT_HOME%\usertmp.txt'
) do (set USER=%%i & goto :enduser)
:enduser
```

```
for /f "tokens=1 delims=<" %%i in ('type %BAT_HOME%\iptmp.txt')
do (set IP=%%i & goto :endip)
:endip
del %BAT_HOME%\usertmp.txt & del %BAT_HOME%\iptmp.txt
FOR /F "tokens=1 delims= " %%i in ('echo %DATE%') do set RIQI=%%
i
FOR /F "tokens=1 delims=." %%i in ('echo %TIME%') do set SHIJIAN
=%%i
::冒号不能统一使用英文或中文状态, 会出现乱码, 不知道原因
set wxMsg=服务器登录提醒：\n主机名：%COMPUTERNAME%\n登录用户^: %USER%
\n登录IP^: %IP%\n登录时间^: %RIQI% %SHIJIAN%
set Body={ \"touser\":\"%wxUserID%\", \"msgtype\":\"text\", \"ag
entid\":\"%wxAppID%\", \"text\":{\"content\":\"%wxMsg%\"}, \"saf
e\":\"0\" }

curl --data-ascii "%Body%" %PURL% >nul
exit
```

新建一个任务计划：

- 常规 -> 运行时使用帐户administrator -> 不管用户是否登录都要运行
- 触发器 -> 所有用户登录时
- 设置 -> 并行运行新实例

在帐户administrator修改密码后要在任务计划里重新输入