

# Урок bat-аники

Автор: Алексей Александров  
Источник: RSDN Magazine #2-2005

Опубликовано: 11.07.2005  
Версия текста: 1.0

## Введение

Как экранировать символ?  
Как перенести длинную строку?  
Как определить имя каталога, в котором находится запущенный командный файл?  
Как получить короткое (8.3) имя файла?  
Как перенаправить стандартный вывод в файл?  
Как сложить два числа?  
А можно создать в bat-файле функцию?  
Как можно избежать использования goto?  
Как обработать текстовый файл?  
Что это за упомянутые ранее операторы объединения команд?  
Можно ли написать на bat-языке серьезную программу?  
Материалы по теме



## ВВЕДЕНИЕ

Мы все любим писать серьезные вещи на серьезных языках. Шаблоны, C++, Reflection, Perl и многое другое – вот то, что мы любим, то, чему посвящаем длинные сообщения в форумах, то, что снится нам по ночам.

Однако в нашей повседневной деятельности встречаются и вещи, которые не так интересны и интеллектуальны. Мы не очень любим говорить об этом, делаем вид, что Это – грязно, нечистоплотно и недостойно нашего внимания. Однако, приходит день, приходит час и перст Судьбы находит нас – нам надо написать еще один батничек... Иногда это запускатка для построения проекта, которая должна при ошибке компиляции скопировать логи на сетевой диск, иногда – запуск обновления исходных текстов из SVN. Иногда – что-нибудь еще.

К чему я это все? А к тому, что поговорим мы о полезных хитростях при написании файлов сценариев на встроенном командном языке Windows. К счастью, это занятие не является доминирующим в профессиональной деятельности автора, так что я не обязуюсь заполнить абсолютно все пробелы в данной области. Кроме того, рожденный ползать летать не может, и из cmd.exe, увы, не получится ни /usr/bin/perl, ни даже /bin/sh. Так что, все нижеприведенное – просто некоторые интересные факты из жизни файлов с расширением bat, на которые автор обратил внимание во время решения различных практических задач автоматизации.

Наш урок будет построен по сугубо практическому принципу, известному в народе как Cookbook. Иными словами, я не буду вдаваться в синтаксические и семантические дебри командного языка Windows, а лишь продемонстрирую его возможности (хотел написать «мощь», но все-таки передумал). Именно поэтому большинство следующих заголовков будет начинаться со слова «Как». Впрочем, для полноты по ходу развития событий будут даваться подробные комментарии, в том числе и по языковым конструкциям.

### ПРЕДУПРЕЖДЕНИЕ

Практически все описанные здесь рецепты подойдут только для Windows 2000 и старше. Bat-язык Windows 9x, к счастью, можно считать почившим, так что здесь он не рассматривается. Более того, диалекты cmd.exe операционных систем Windows 2000, Windows XP и Windows Server 2003 также немного различаются. Все приведенное ниже создано и проверено на компьютере под управлением операционной системы Windows XP. За подробной информацией по различиям в реализации той или иной команды обращайтесь к [1].

## КАК ЭКРАНИРОВАТЬ СИМВОЛ?

В командном языке Windows существует некоторый набор символов с высоким приоритетом, которые всегда трактуются как спецсимволы. К ним, в частности, относятся:

- Операторы перенаправления ввода-вывода <, >, >>.

- Оператор конвейера |.
- Операторы объединения команд ||, & и &&.
- Оператор разыменования переменной %...%.

В случае если символ, относящийся к одному из таких операторов, должен быть включен в вашу команду в его literalном смысле, вас ждут определенные неожиданности. Например, при выполнении вот такой строки

```
echo The ratio should be up to 10%.
```

символ процента будет «съеден» интерпретатором, который решит, что это попытка вывода значения какой-то переменной. В случае со знаком процента решение довольно хорошо известно и состоит в удвоении этого символа:

```
echo The ratio should be up to 10%%.
```

после чего все заработает так, как надо. Однако в других случаях все менее очевидно. Рассмотрим командный сценарий, который генерирует незатейливый HTML-файл:

```
@echo off
set OUTPUTFILE=%1

echo <html>                                >%OUTPUTFILE%
echo <head>                                >>%OUTPUTFILE%
echo <title>This is a greeting page</title> >>%OUTPUTFILE%
echo </head>                                >>%OUTPUTFILE%
echo <body>                                >>%OUTPUTFILE%
echo Hello World!                          >>%OUTPUTFILE%
echo </body>                                >>%OUTPUTFILE%
echo </html>                                >>%OUTPUTFILE%
```

К сожалению, при попытке запуска этого "чуда инженерного разума" нас постигнет неудача в виде сообщения об ошибке

```
> was unexpected at this time.
```

Оно и понятно: командный интерпретатор не в силах разобраться, где его просят вывести на экран символ HTML-тега, а где перенаправить вывод. В нормальных языках программирования эта проблема обычно решается обрамлением строковых литералов кавычками. Отчасти это помогает и в bat-файлах. Но лишь отчасти. Выполнение строки

```
echo "<html>"                                >%OUTPUTFILE%
```

приведет к тому, что в выходной файл будут записаны и сами кавычки. Это явно не совсем то, что требуется.

К счастью, есть один малоизвестный способ, позволяющий добиться требуемого результата. Символ ^ позволяет экранировать любой другой символ с безусловным приоритетом. Таким образом, вышеприведенный пример генерации HTML может быть успешно записан так:

```
@echo off
set OUTPUTFILE=%1

echo ^<html^>                                >%OUTPUTFILE%
echo ^<head^>                                >>%OUTPUTFILE%
echo ^<title^>This is a greeting page^</title^> >>%OUTPUTFILE%
echo ^</head^>                                >>%OUTPUTFILE%
echo ^<body^>                                >>%OUTPUTFILE%
echo Hello World!                          >>%OUTPUTFILE%
echo ^</body^>                                >>%OUTPUTFILE%
echo ^</html^>                                >>%OUTPUTFILE%
```

Таким же способом можно экранировать любой другой специальный символ. Очевидно, можно экранировать и сам ^. Не очень эстетично, зато дешево и практично. Слово «надежно» я пропустил умышленно...

## КАК ПЕРЕНЕСТИ ДЛИННУЮ СТРОКУ?

Совет по поводу экранирующего символа ^ имеет еще одно применение: перенос строк. Я (как и многие из вас, наверное) люблю, чтобы любой исходный текст, который я пишу, выглядел красиво – даже \*.bat-файлы. Одним из обязательных условий красоты и удобочитаемости кода для меня является его

ширина: все строки должны уместиться в 78 столбцов. Можно поспорить по поводу числа 78, но в одном я непреклонен – ограничение на ширину текста кода должно быть, иначе это не код, а макароны.

Так вот долгое время \*.bat-файлы портили мне жизнь тем, что иногда приходилось писать длинную строку – например, вызов какой-нибудь другой программы с кучей опций, и я не знал, что с этим делать. Происходило это нечасто, но всегда было неприятно. Но, к счастью, моя жизнь изменилась с тех пор, как я открыл для себя Супер-Символ ^:

```
packagebin.exe --recursive-search=yes --files-mask=exe,dll,pdb,obj ^
--archive-type=zip --archive-level=max --deliver-method=ftp ^
--deliver-target=ftp://ftp.site.com
```

Помните лишь, что чудо-символ должен быть последним в строке – скажите «Нет!» концевым пробелам.

## КАК ОПРЕДЕЛИТЬ ИМЯ КАТАЛОГА, В КОТОРОМ НАХОДИТСЯ ЗАПУЩЕННЫЙ КОМАНДНЫЙ ФАЙЛ?

Иногда сценарию надо знать полный путь к себе самому и/или к каталогу, в котором он находится. Это может понадобиться по разным причинам. Например, он должен достать из системы контроля версий исходники в каталог <script-dir>/src рядом с собой. Или, запускаются тесты из каталога <script-dir>/tests, и перед их запуском надо добавить каталог <script-dir>/bin в переменную PATH.

Можно, конечно, рассчитывать на то, что командный файл был вызван из того же каталога, где он находится, и тогда в качестве вышеупомянутого <script-dir> можно использовать переменную окружения %CD% - полный путь к текущему каталогу. Однако любые допущения в нашем деле недопустимы (хороший каламбур, однако!). Поэтому приведу более надежное решение.

Прежде всего, вспоминаем, что переменная %0 в bat-файле соответствует нулевому аргументу командной строки, т.е. имени самого файла. После этого читаем скудную документацию для команды call:

```
call /?
```

и обнаруживаем, что при использовании нумерованных переменных %0-%9 можно использовать некоторые модификаторы:

%~l	- разворачивает %1, удаляя кавычки (")
%~fl	- разворачивает %1 в полный квалифицированный путь
%~dl	- разворачивает %1 в букву диска
%~pl	- разворачивает %1 в путь
%~nl	- разворачивает %1 в имя файла
%~xl	- разворачивает %1 в расширение файла
%~sl	- развернутый путь будет содержать только короткие имена
%~al	- разворачивает %1 в атрибуты файла
%~tl	- разворачивает %1 в дату/время создания файла
%~zl	- разворачивает %1 в размер файла
%~\$PATH:1	- Ищет в каталогах, перечисленных в переменной среды PATH, и разворачивает %1 в полное квалифицированное имя первого совпадения. Если имя переменной среды не определено, или если файл не найден, этот модификатор вернет пустую строку

и, более того:

Модификаторы можно объединять для получения сложных результатов:

%~dp1	- разворачивает %1 в букву диска и путь
%~nx1	- разворачивает %1 в имя файла с расширением
%~dp\$PATH:1	- ищет %1 в каталогах, перечисленных в переменной среды PATH, и разворачивает в букву диска и путь к первому найденному файлу.
%~ftzal	- разворачивает %1 в строку, подобную DIR

Таким образом, правильным будет использовать в качестве тега <script-dir> сочетание %~dp0, которое будет раскрыто в полный путь к каталогу, где находится сценарий. Например,

```
"%~dp0\packagebin.exe" --recursive-search=yes --files-mask=exe,dll,pdb,obj ^
--archive-type=zip --archive-level=max --deliver-method=ftp ^
--deliver-target=ftp://ftp.site.com --deliver-source="%~dp0\bin"
```

Обратите внимание на использование кавычек – потенциально каталог может иметь в своем пути пробел. Кавычки избавят от проблем в этом случае.

### ПРЕДУПРЕЖДЕНИЕ

Опасайтесь бездумного применения команды cd %~dp0 без проверки результата выполнения. Теоретически, эта команда должна сменить текущий каталог на каталог, в котором расположен командный файл. Как правило, это работает. Однако

возможны неожиданности. Однажды был написан простой командный сценарий, задача которого была просто удалить все каталоги рядом с собой. В «свою» директорию он переходил как раз через `cd %~dp0`. Все было проверено на локальной машине – работало замечательно. После этого сценарий был помещен на файл-сервер, где ему и полагалось быть. Я зашел с помощью `Far` в сетевой каталог, и для контрольной проверки решил запустить файл еще раз. Дальнейшее словно в тумане. `cmd.exe` правильно определил местонахождение `bat`-файла: `\\servername\sharename\directory`. Однако при попытке сделать туда `cd`, он сказал, что UNC-пути в качестве текущих каталогов не поддерживаются и лучше он сменит текущий каталог на `C:\WINDOWS...` Это было действительно мудрое решение... Часть сценария, отвечавшая за удаление всех каталогов, сработала отлично – хорошо, что я успел вовремя остановить это безумие. В тот день я узнал, что такое System Restore...

## КАК ПОЛУЧИТЬ КОРОТКОЕ (8.3) ИМЯ ФАЙЛА?

«А зачем? – спросите вы – Ведь мы живем в мире Интернета, Web-сервисов и NTFS с длинными именами файлов». Это действительно так, но иногда встречаются программы, которые отчаянно сопротивляются прогрессу, и в частности, не любят имен файлов и полных путей с пробелами. Одной из таких программ, кстати, является утилита `build.exe` из Windows DDK... В таких ситуациях спасает использование короткого, «беспробельного» DOS-имени для файла.

### ПРЕДУПРЕЖДЕНИЕ

Доступ к файлу по короткому имени может быть не всегда возможен. На файловой системе NTFS создание коротких псевдонимов для файлов может быть отключено путем установки в единицу значения «`NtfsDisable8dot3NameCreation`» в ключе реестра «`HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\FileSystem`».

Итак, все же (в предположении, что надругательства над NTFS не было) – как? Внимательный читатель должен был заметить в предыдущем разделе, что при обращении к переменным `%0 - %9` можно использовать префикс

```
%~s1 - expanded path contains short names only
```

который нам как раз мог бы помочь. Но есть засада – все эти полезные префиксы нельзя использовать с произвольной переменной окружения, а присваивание переменным `%0 - %9` не поддерживается. К счастью, описываемые префиксы можно еще использовать с переменными цикла `for`, и это дает нам способ достичь требуемого результата. Например, вот так можно получить 8.3-путь к “Program Files”:

```
for /d %i in ("%PROGRAMFILES%") do (
    set PROGRAMFILESSHORT=%~s1
)

echo 8.3-имя для каталога "%PROGRAMFILES%" -^> "%PROGRAMFILESSHORT%"
```

Этот и другие модификаторы можно использовать и с любой другой формой цикла `for`, подробнее о которых можно узнать из:

```
for /?
```

## КАК ПЕРЕНАПРАВИТЬ СТАНДАРТНЫЙ ВЫВОД В ФАЙЛ?

Плоха та короткая программа, которая не стремится стать большой. К сожалению, это правило применимо и к командным файлам Windows тоже – иногда `bat`-файлы вырастают до довольно больших размеров. Если при этом результат выполняемых команд должен журналироваться, то все становится совсем плохо – почти каждая строка имеет хвостик типа

```
echo Cleaning up the target directory >>%LOGFILE%
...
echo The target directory has been cleaned >>%LOGFILE%
```

Гораздо проще было бы перенаправить стандартный вывод в файл, чтобы все команды `echo` и вообще, все, что выводится на экран, автоматически попадали в журнальный файл. Сделать это можно следующим образом (рассмотрим на знакомом примере генерации HTML-файла):

```
@echo off
set OUTPUT=out.html

if "%STDOUT_REDIRECTED%" == "" (
    set STDOUT_REDIRECTED=yes
    cmd.exe /c %0 %* >%OUTPUT%
    exit /b %ERRORLEVEL%
)
```

```
echo ^<html^>
echo ^<head^>
echo ^<title^>This is a greeting page^</title^>
echo ^</head^>
echo ^<body^>
echo Hello World!
echo ^</body^>
echo ^</html^>
```

Здесь делается то же, что и раньше, но с перенаправлением стандартного вывода в файл out.html. Делается это простым способом – перезапуском сценарием самого себя. Сначала проверяется, не установлена ли переменная окружения STDOUT\_REDIRECTED. Если она установлена, значит, сценарий уже перезапущен с перенаправленным выводом и можно просто продолжить работу. Если же переменная не установлена, то мы ее устанавливаем и перезапускаем скрипт (cmd.exe /c %0) с таким же набором параметров, что и исходная команда (%\*) и перенаправленным в файл стандартным выводом (>%OUTPUT%). После завершения выполнения «перенаправленной» команды выходим.

Такое «единовременное» перенаправление имеет и еще один неочевидный плюс: файл открывается и закрывается только один раз, и всем командам и дочерним процессам передается дескриптор уже открытого файла. Во-первых, это чуть-чуть улучшит производительность (жизнь удалась – сроду бы не подумал, что буду когда-нибудь писать о производительности в bat-файлах). Во-вторых, это поможет избежать проблемы с невозможностью открыть файл для записи. Такое может случиться, если после выполнения одной из команд останется «висеть» какой-нибудь процесс. Он будет держать дескриптор интересующего нас файла и перенаправление вывода в этот файл для всех последующих команд провалится. Проблема может показаться надуманной, но однажды она украла у меня 2 часа жизни...

## Как сложить два числа?

Краткий ответ – смотри:

```
set /?
```

Длинный ответ таков. В bat-файлах можно производить довольно-таки продвинутые вычисления – продвинутые не в сравнении с другими языками, а в сравнении с отсутствием возможности что-либо вычислить вообще. Вычисление осуществляется командой set, если она выполняется с ключом /a. Поддерживается практически полный набор операторов языка C, включая шестнадцатеричный модификатор 0x. Переменные окружения в выражении не обязательно заключать в знаки процента – все, что не является числом, считается переменной. Подробнее – все-таки в man set, тьфу, то есть в set /?. А здесь напоследок – просто несколько примеров.

```
@echo off

set ARG=1

rem Переменные окружения в выражении не обязательно заключать в %...%
set /a RESULT=ARG + 2
echo %RESULT%

rem Если выражение содержит какие-либо из символов non grata, надо
rem заключить его в кавычки
set /a RESULT="ARG << 2"
echo %RESULT%

rem Шестнадцатеричная арифметика
set /a RESULT=0x1234 + 0x6786
echo %RESULT%

rem И многое-многое другое...
```

## А можно создать в бат-файле функцию?

Да, можно. Более того, иногда даже нужно. Правда, функциями это можно назвать условно. Есть особый синтаксис команды call, который позволяет перейти на метку в этом же bat-файле с запоминанием места, откуда был произведен этот вызов:

```
call :метка аргументы
```

Возврат из функции производится командой:

```
exit /b [опциональный код возврата]
```

Ключ /b здесь очень важен: без него будет произведен выход не из функции, а из сценария вообще.

За подробностями обращайтесь к:

```
call /?  
exit /?
```

Что интересно, команда call с таким синтаксисом поддерживает рекурсивные вызовы с автоматическим созданием нового фрейма для переменных аргументов %0-%9. Иногда это может быть полезным. Вот классический пример рекурсивного подсчета факториала на командном языке:

```
@echo off  
  
call :factorial %1  
echo %RESULT%  
exit  
  
rem Функция для подсчета значения факториала  
rem Вход:      %1          Число, для которого необходимо подсчитать факториал  
rem Выход:     %RESULT%   Значение факториала  
rem :factorial  
  
if %1 == 0 (  
    set RESULT=1  
    exit /b  
)  
  
if %1 == 1 (  
    set RESULT=1  
    exit /b  
)  
  
set /a PARAM=%1 - 1  
call :factorial %PARAM%  
  
set /a RESULT=%1 * %RESULT%  
  
exit /b
```

Пример работы:

```
> factorial.bat 10  
3628800
```

## КАК МОЖНО ИЗБЕЖАТЬ ИСПОЛЬЗОВАНИЯ GOTO?

Любой хоть сколько-то осмысленный \*.bat-файл длиной больше 50 строк является ярким лозунгом в поддержку работы Дейкстры «О вреде оператора goto». Мешанина из переходов вперед и назад действительно является кодом «только для записи». Можно ли что-то предпринять по этому поводу?

На самом деле можно. Как правило, большинство меток и переходов используются для организации ветвлений при проверке условий, т.е. банальных if-then-else блоков. В оригинале, bat-язык поддерживал только одну команду в блоке then, что автоматически приводило к идиомам вида:

```
if condition goto :THEN  
rem Команды ветки 'else'  
rem ...  
goto IF_END  
:THEN  
rem Команды ветки 'then'  
rem ...  
:IF_END
```

Но к счастью, командный интерпретатор cmd.exe современных ОС Windows 2000 и старше поддерживает блоки команд в конструкциях ветвления, что устраняет необходимость применения меток. Блоки команд заключаются в круглые скобки. Выглядит это так (имитируя C/C++ indentation style):

```
if condition (  
    rem Команды ветки 'then'  
    rem ...  
) else (  
    rem Команды ветки 'else'  
    rem ...  
)
```

Конкретный пример использования:

```
@echo off

set BUILDMODE=%1

if "%BUILDMODE%" == "" (
    echo FAIL: Аргумент является обязательным ^(--debug, --release^)
    exit /b 1
)

rem Удаляем из аргумента все дефисы для упрощения обработки
set BUILDMODE=%BUILDMODE:--%

if "%BUILDMODE%" == "debug" (
    echo INFO: Устанавливаем debug-режим окружения
    set CCFLAGS=/Od /MDd /Z7
) else (
    echo INFO: Устанавливаем release-режим окружения
    set CCFLAGS=/O2 /MD
)
```

На мой взгляд, с этим уже вполне можно жить. Но, как всегда, жизнь не так проста, как кажется. Есть одна проблема. Переменные, использующиеся в блоках then и else, раскрываются перед началом выполнения этих блоков, а не в процессе выполнения. В приведенном примере это не вызывает никаких проблем, однако в следующем вызовет:

```
if "%BUILDMODE%" == "debug" (
    echo INFO: Устанавливаем debug-режим окружения
    set OPTFLAGS=/Od
    set CCFLAGS=%OPTFLAGS% /MDd /Z7
) else (
    echo INFO: Устанавливаем release-режим окружения
    set OPTFLAGS=/O2
    set CCFLAGS=%OPTFLAGS% /MD
)
```

Загвоздка в том, что в обоих блоках подстановка переменной OPTFLAGS произойдет до того, как она будет изменена в процессе выполнения этого блока. Соответственно, в CCFLAGS будет подставлено то значение, которое OPTFLAGS имела на момент начала выполнения данного if-блока.

Решается эта проблема путем использования отложенного раскрытия переменных. Переменные, заключенные в !...! вместо %...%, будут раскрыты в их значения только в момент непосредственного использования. Данный режим по умолчанию отключен. Включить его можно либо использованием ключа /V:ON при вызове cmd.exe, либо использованием команды

```
setlocal enabledelayedexpansion
```

в тексте самого bat-файла. Второй способ мне представляется более удобным – не очень здорово требовать от кого-то запуска твоего сценария с определенным параметром.

С учетом сказанного предыдущий «неправильный» пример может быть исправлен так:

```
setlocal enabledelayedexpansion

rem ...

if "%BUILDMODE%" == "debug" (
    echo INFO: Setting up debug mode environment
    set OPTFLAGS=/Od
    set CCFLAGS=!OPTFLAGS! /MDd /Z7
) else (
    echo INFO: Setting up release mode environment
    set OPTFLAGS=/O2
    set CCFLAGS=!OPTFLAGS! /MD
)
```

Вот теперь это почти полноценный if-then-else блок. Почти, потому что если в одной из команд echo у вас встретится закрывающая круглая скобка, то вам необходимо заэкранировать ее символом ^, иначе синтаксический анализатор путается...

Но в любом случае, это гораздо лучше безумного количества меток и переходов.

## КАК ОБРАБОТАТЬ ТЕКСТОВЫЙ ФАЙЛ?

Иногда в командном файле необходимо получить доступ к содержимому некоторого текстового файла и некоторым образом это содержимое обработать. Например, прочитать файл настроек программы.

Для привнесения еще большей конкретики в процесс изучения зададимся целью прочитать файл с настройками следующего содержания:

```
# Это простой файл с настройками

# Режим сборки
buildmode=release

# Компилятор
compiler=cl.exe

# Архитектура
arch=x86
```

Ничего сверхъестественного – простой key=value формат с возможностью вставки Unix-style комментариев. Помочь в чтении и обработке этого файла нам сможет команда `for`. Ее дополнительные опции позволяют задать и разделители, и символ начала комментария, и кое-что еще. Вот командный файл, который выполняет поставленную задачу:

```
@echo off

rem Читаем настройки из файла settings.txt, который должен располагаться в
rem том же каталоге, что и bat-файл. Если не удалось распарсить настройки -
rem выходим с ненулевым кодом возврата.
call :read_settings %~dp0\settings.txt || exit /b 1

rem Прочитанные настройки:
echo Build mode   : %BUILDMODE%
echo Compiler     : %COMPILER%
echo Architecture: %ARCH%

rem Выход из сценария. Дальше - только функции.
exit /b 0

rem
rem Функция для чтения настроек из файла.
rem Вход:
rem     %1                - Имя файла с настройками
rem
rem :read_settings
rem
set SETTINGSFILE=%1

rem Проверка существования файла
if not exist %SETTINGSFILE% (
    echo FAIL: Файл с настройками отсутствует
    exit /b 1
)

rem Обработка файла с настройками
rem Здесь:
rem     eol=# указывает на то, что содержимое строки начиная с символа #
rem     и до ее конца может быть пропущено как комментарий.
rem
rem     delims== указывает, что разделителем значений является символ =
rem
rem     tokens=1,2 приводит к тому, что в переменную %%i будет занесен первый
rem     токен, а в %%j - второй.
rem
for /f "eol=# delims== tokens=1,2" %%i in (%SETTINGSFILE%) do (
    rem В переменной i - ключ
    rem В переменной j - значение
    rem Мы транслируем это в переменные окружения
    set %%i=%%j
)

exit /b 0
```

Обильные комментарии должны помочь легко разобраться, что к чему. За подробностями, как обычно, отошлю к:

```
for /?
```

Кстати, возможности команды `for` не ограничиваются чтением из файла. Возможно также чтение вывода другой команды. Например, так:

```
@echo off

for /f "tokens=* usebackq" %%i in (`cmd.exe /c ver`) do (
```



```
)      set VERSION=%i
)

echo %VERSION%
```

Особенно меня умиляет наличие опции “usebackq”, которая делает синтаксис отдаленно похожим на юниксовый. И в стенах царства Билла есть граждане, скучающие по /bin/sh и пытающиеся хоть как-то скрасить существование свое и окружающих. Следующий совет это также косвенно подтверждает.

## ЧТО ЭТО ЗА УПОМЯНУТЫЕ РАНЕЕ ОПЕРАТОРЫ ОБЪЕДИНЕНИЯ КОМАНД?

Это операторы &, && и ||. Они практически совсем не освещены в документации, но полезны в повседневности. Они позволяют объединять несколько команд в одну, т.е. примерно так:

```
command1 & command2
command1 && command2
command1 || command2
```

Форма этих операторов весьма соответствует их содержанию. В случае, пожалуй, наименее полезного оператора & вторая команда будет просто выполнена после первой, т.е. это равносильно простой записи:

```
command1
command2
```

Оператор && гарантирует, что вторая команда будет выполнена только, если первая была выполнена успешно, т.е. с нулевым кодом возврата (он же %errorlevel%). Такие конструкции очень популярны в мире shell-сценариев Unix. Например:

```
cd sources && make clean
```

Я был приятно удивлен, узнав, что cmd.exe тоже умеет выполнять такие конструкции. Это безопаснее и правильнее, нежели простое последовательное выполнение этих команд, и короче и проще, чем строгая проверка и обработка кодов возврата. Очень удобно при написании на скорую руку. Не менее полезен иногда и оператор ||. Суть его тоже логична – выполнить вторую команду, если первая дала сбой. Часто встречается в таких идиомах:

```
cd sources || exit 1
```

Если перейти в каталог sources не удастся, то будет произведен выход с кодом ошибки 1. Если же первая команда отработает нормально, то вторая выполнена не будет. Например, такая простейшая защита помогла бы в случае с cd по UNC-адресу, описанному ранее.

## МОЖНО ЛИ НАПИСАТЬ НА БАТ-ЯЗЫКЕ СЕРЬЕЗНУЮ ПРОГРАММУ?

Пожалуй, нет. Серьезная программа должна все-таки выглядеть серьезно. А все написанное на командном языке Windows таковым назвать можно лишь с о-о-о-чень большой натяжкой. Так что для решения более сложных задач автоматизации лучше все-таки взять что-нибудь более функциональное:

- Perl
- Python
- Ruby
- JScript / VBScript

Последние, кстати, присутствуют в Windows 2000/XP по умолчанию (с некоторыми функциональными различиями) и в целом могут считаться заменой \*.bat языку. Однако сдается мне, что \*.bat-файлы проживут еще очень долго.

Дай Бог, чтобы я ошибся...

## МАТЕРИАЛЫ ПО ТЕМЕ

1. Windows Batch Files – Commands Reference.
2. Windows 2000. Команды: Карманный справочник. Э. Фриш. – М.: Мир, 2003.