ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
"МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ им. Н.П. ОГАРЕВА"

А.В. ПОПОВ

ЯЗЫК КОМАНДНОГО ПРОЦЕССОРА CMD.EXE ОПЕРАЦИОННОЙ СИСТЕМЫ WINDOWS

УЧЕБНОЕ ПОСОБИЕ

САРАНСК
ИЗДАТЕЛЬСТВО МОРДОВСКОГО УНИВЕРСИТЕТА
2009

УДК 004.384 (075.8)
ББК 397
П58

Рецензенты:
кафедра информационно-вычислительных систем Саранского кооперативного института (филиал) АНО ВПО ЦС РФ "Российский университет кооперации";
директор ИМЦ МРИО, заслуженный работник образования Республики Мордовия *Т.П. Лунина*

П58 Язык командного процессора Cmd.exe операционной системы Windows : учеб. пособие / А.В. Попов. – Саранск : Изд-тво Мордов. ун-та, 2009. – 56 с.
ISBN 978-5-7103-20

В пособии рассматриваются возможности командного процессора Cmd.exe, являющегося стандартным инструментом автоматизации работы в операционной системе Windows.

Предназначено для студентов специальностей "Математика" и "Прикладная математика и информатика".

УДК 004.384 (075.8)
ББК 397

ISBN 978-5-7103-20

Лекция 1. Эволюция инструментов для автоматизации работы в Microsoft Windows

В настоящее время графический интерфейс Windows стал настолько привычным, что многие пользователи и начинающие администраторы даже не задумываются (а зачастую и просто не знают) об альтернативных способах управления данной операционной системой, связанных с *командной строкой* (command line) и различными *сценариями* (scripts), о тех преимуществах, которые дают эти инструменты с точки зрения *автоматизации работы*, то есть решения различных задач в автоматическом режиме, без участия человека. Подобная ситуация обусловлена тем, что исторически командная строка всегда была слабым местом операционной системы Windows (по сравнению с Unix-системами). Причиной этого, в основном, является то, что изначально компания Microsoft ориентировалась на широкую аудиторию неискушенных пользователей, не желающих особо вникать в технические детали выполнения тех или иных действий в системе. Поэтому основные усилия разработчиков операционной системы направлялись на улучшение графической оболочки для более комфортной работы непрофессионалов, а не на создание рабочей среды для специалистов или опытных пользователей.

Как показало время, с коммерческой точки зрения на рынке персональных (домашних или офисных) компьютеров эта стратегия оказалась более чем успешной – миллионы людей используют графический интерфейс Windows для запуска нужных им программ, работы в офисных пакетах, просмотра фильмов и т.п. Да и управлять одним Windows-сервером сегодня несложно – операционная система предлагает удобные графические средства для настройки различных параметров и выполнения ежедневных администраторских задач, а с помощью службы терминалов легко можно работать на удаленном сервере, физически расположенном на другом континенте.

Однако подобная модель управления не является масштабируемой: если администрировать не один, а десять серверов, используя стандартные графические инструменты, то одну и ту же последовательность изменения элементов управления в диалоговых окнах придется повторить десять раз, следовательно, в этом случае остро встает вопрос об автоматизации выполнения рутинных операций (например, проведение инвентаризации оборудования и программного обеспечения, мониторинг работы служб, анализ журналов событий и т.д.) на множестве компьютеров. Помочь в этом могут либо специальные (как правило тяжеловесные и недешевые) приложения типа Microsoft Systems Management Server (SMS), либо сценарии, которые пишутся администраторами самостоятельно (на языке оболочки командной строки или на специальных языках сценариев) и поддерживаются непосредственно операционной системой, без установки сторонних программных продуктов.

Поэтому для профессионала, занимающегося администрированием информационных систем на базе Windows, знание возможностей командной строки, сценариев и технологий автоматизации, поддерживаемых данной операционной системой, просто необходимо.

При этом, однако, неправильно было бы думать, что командная строка или сценарии нужны только администраторам. Ведь ежедневные рутинные задачи пользователей (связанные, например, с копированием или архивированием файлов, подключением или отключением сетевых ресурсов и т.п.), которые обычно выполняются с помощью графического интерфейса проводника Windows, можно полностью самостоятельно автоматизировать, написав нехитрый командный файл, состоящий всего из нескольких строчек! Более того, для человека, не знающего основные команды Windows и такие базовые возможности операционной системы, как перенаправление ввода/вывода и конвейеризация команд, некоторые простейшие задачи могут показаться нетривиальными. Попробуйте, например, пользуясь только графическими средствами, сформировать файл, содержащий имена файлов из всех подкаталогов какого-либо каталога! А ведь для этого достаточно выполнить единственную команду `DIR` (с определенными ключами) и перенаправить вывод этой команды в нужный текстовый файл.

Каким же нам хотелось бы видеть инструмент для автоматизации работы в операционной системе, какими возможностями он должен обладать? Желательно, чтобы в нем было реализовано следующее:

- работа в разных версиях операционной системы (желательно во всех) без установки какого-либо дополнительного программного обеспечения;
- интеграция с командной строкой (непосредственное выполнение вводимых с клавиатуры команд);
- согласованный и непротиворечивый синтаксис команд и утилит;
- наличие подробной встроенной справки по командам с примерами использования;
- возможность выполнения сценариев, составленных на простом для изучения языке;
- возможность использования всех технологий, поддерживаемых операционной системой.

В Unix-системах в качестве инструмента автоматизации выступает стандартная оболочка (shell) или ее модификации (bashell, kshell, cshell и т.д.), причем этот аспект операционной системы стандартизирован в рамках POSIX (стандарт мобильных систем).

В операционной системе Windows дело обстоит сложнее. На сегодняшний день одного "идеального" средства автоматизации, удовлетворяющего сразу всем перечисленным выше требованиям, в Windows нет; в последних версиях операционной системы поддерживаются несколько стандартных инструментов автоматизации, сильно отличающихся друг от друга: оболочка командной строки cmd.exe, среда выполнения сценариев Windows Script Host и оболочка Microsoft PowerShell. Поэтому администратору или пользователю Windows приходится выбирать, каким

именно подходом воспользоваться для решения определенной задачи, а для этого желательно иметь четкое представление о сильных и слабых сторонах данных средств автоматизации. Здесь мы кратко обсудим достоинства и недостатки каждого из них.

Оболочка командной строки command.com/cmd.exe

Во всех версиях операционной системы Windows поддерживается интерактивная *оболочка командной строки* (command shell) и по умолчанию устанавливается определенный набор утилит командной строки (количество и состав этих утилит зависит от версии операционной системы) [1], [2]. Вообще, любую операционную систему можно представить в виде совокупности *ядра системы*, которое имеет доступ к аппаратуре и оперирует файлами и процессами, и *оболочки* (*командного интерпретатора*) с утилитами, которые позволяют пользователю получить доступ к функциональности ядра операционной системы. Механизм работы оболочек в разных системах одинаков: в ответ на приглашение ("подсказку", prompt), выдаваемое находящейся в ожидании оболочкой, пользователь вводит некоторую команду (функциональность этой команды может быть реализована либо самой оболочкой, либо определенной внешней утилитой), оболочка выполняет ее, при необходимости выводя на экран какую-либо информацию, после чего снова выводит приглашение и ожидает ввода следующей команды. С технической точки зрения оболочка представляет собой построчный интерпретатор простого языка сентенциального (директивного) программирования, в качестве операторов которого могут использоваться исполняемые программы.

Наряду с интерактивным режимом работы оболочки, как правило, поддерживают и пакетный режим, в котором система последовательно выполняет команды, записанные в текстовом файле-сценарии. Оболочка Windows не является исключением, с точки зрения программирования язык командных файлов Windows может быть охарактеризован следующим образом:

- реализация сентенциальной (директивной) парадигмы программирования;
- выполнение в режиме построчной интерпретации;
- наличие управляющих конструкций;
- поддержка нескольких видов циклов (в том числе специальных циклов для обработки текстовых файлов);
- наличие оператора присваивания (установки значения переменной);
- возможность использования внешних программ (команд) операционной системы в качестве операторов и обработки их кодов возврата;
- наличие нетипизированных переменных, которые декларируются первым упоминанием (значение переменных могут интерпретироваться как числа и использоваться в выражениях целочисленной арифметики).

Начиная с версии Windows NT, оболочка командной строки представляется интерпретатором Cmd.exe, который расширяет возможности оболочки command.com операционной системы MS-DOS. В свою очередь

функциональность командного интерпретатора command.com была позаимствована из операционной системы CP/M, оболочка которой представляла собой значительно упрощенный и урезанный вариант оболочки Unix-систем.

Таким образом, оболочка командной строки MS-DOS изначально уступала Unix-оболочкам по удобству работы и развитости языка сценариев; в командной оболочке Windows (cmd.exe), несмотря на все сделанные улучшения, не удалось преодолеть данное отставание ни в режиме интерактивной работы (например, в cmd.exe отсутствует поддержка псевдонимов для длинных названий команд и не реализовано автоматическое завершение команд при вводе их с клавиатуры), ни в синтаксисе или возможностях языка командных файлов. Ситуация усугублялась тем, что Windows всегда проигрывала Unix-системам в количестве и функциональных возможностях стандартных (не требующих дополнительной установки) утилит командной строки, а также в качестве и полноте встроенной справочной системы по командам оболочки.

На практике проблему отсутствия нужной функциональности у стандартных команд приходится решать либо с помощью утилит пакета Windows Resource Kit для соответствующей версии операционной системы, либо путем поиска подходящей утилиты сторонних производителей. Кроме того, в Windows можно пользоваться POSIX-совместимыми утилитами и оболочками с помощью пакета Microsoft Services For Unix (SFU). Данный продукт разрабатывался еще для Windows NT и первоначально не входил в состав операционной системы, его нужно было приобретать за отдельную плату. В дальнейшем пакет SFU стал бесплатным и даже был включен в состав операционной системы Windows Server 2003 R2.

Итак, учитывая все сказанное выше, мы можем сделать следующий вывод: оболочка командной строки cmd.exe и командные файлы – наиболее универсальные и простые в изучении средства автоматизации работы в Windows, доступные во всех версиях операционной системы, которые, однако, существенно проигрывают аналогичным инструментам в Unix-системах и не обеспечивают доступ к объектным моделям, поддерживаемым операционной системой (COM, WMI, .NET).

Поддержка языков сценариев. Сервер сценариев Windows Script Host

Следующим шагом в развитии средств и технологий автоматизации в операционной системе Windows стало появление сервера сценариев Windows Script Host (WSH). Этот инструмент разработан для всех версий Windows и позволяет непосредственно в операционной системе выполнять сценарии на полноценных языках сценариев (по умолчанию, VBScript и JScript), которые до этого были доступны только внутри HTML-страниц и работали в контексте безопасности веб-браузера (в силу этого подобные сценарии, например, могли не иметь доступа к файловой системе локального компьютера).

По сравнению с командными файлами интерпретатора cmd.exe сценарии WSH имеют несколько преимуществ.

Во-первых, VBScript и JScript – это полноценные алгоритмические языки, имеющие встроенные функции и методы для обработки символьных строк, выполнения математических операций, обработки исключительных ситуаций и т.д.; кроме того, для написания сценариев WSH может использоваться любой другой язык сценариев (например, широко распространенный в Unix-системах Perl), для которого установлен соответствующий модуль поддержки.

Во-вторых, WSH поддерживает несколько собственных объектов, свойства и методы которых позволяют решать некоторые часто возникающие повседневные задачи администратора операционной системы: работа с сетевыми ресурсами, переменными среды, системным реестром, ярлыками и специальными папками Windows, запуск и управление работой других приложений.

В-третьих, из сценариев WSH можно обращаться к службам любых приложений-серверов автоматизации (например, программ из пакета Microsoft Office), которые регистрируют в операционной системе свои объекты.

Наконец, сценарии WSH позволяют работать с объектами информационной модели Windows Management Instrumentation (WMI), обеспечивающей программный интерфейс управления всеми компонентами операционной модели, а также с объектами службы каталогов Active Directory Service Interface (ADSI) (объектные модели WMI и ADSI будут обсуждаться подробнее в следующих лекциях).

Следует также отметить, что технология WSH поддерживается в Windows уже довольно давно, в Интернете (в том числе на сайте Microsoft) можно найти множество готовых сценариев, выполняющих ту или иную операцию и при определенных навыке и знаниях быстро "подогнать" эти сценарии под свои конкретные задачи.

Поговорим теперь о слабых местах WSH. Прежде всего, сам по себе WSH – это только среда выполнения сценариев, а не оболочка; WSH не интегрирован с командной строкой, то есть отсутствует режим, в котором можно было вводить команды с клавиатуры и сразу видеть результат их выполнения.

Большим минусом является то, что в операционной системе по умолчанию нет полноценной подробной справочной информации по объектам WSH и языкам VBScript/JScript (документацию приходится искать в Интернете на сайте Microsoft). Другими словами, если вы, например, не помните синтаксис определенной команды VBScript/JScript или точное название свойства объекта WSH, под рукой у вас нет распечатанной документации, а компьютер не имеет выхода в Интернет, то написать корректный сценарий вам просто не удастся (в данном аспекте командные файлы более универсальны, так как практически у всех команд есть по крайней мере встроенное описание используемых ими ключей, а в

операционной системе имеется справочный файл с информацией о всех стандартных командах).

Наконец, сценарии WSH представляют собой довольно серьезную потенциальную угрозу с точки зрения безопасности, известно большое количество вирусов, использующих WSH для выполнения деструктивных действий.

Таким образом, можно дать следующую общую оценку: сценарии WSH – это универсальный инструмент, который в любой версии операционной системы Windows позволяет решать задачи автоматизации практически любой степени сложности, но требует при этом большой работы по изучению самих языков сценариев и ряда смежных технологий управления операционной системой (WMI, ADSI и т.п.).

Командная оболочка Microsoft PowerShell

Итак, к началу XXI века ситуацию со средствами автоматизации работы в Windows нельзя было назвать совсем хорошей. С одной стороны функциональности и гибкости языка оболочки cmd.exe было явно недостаточно, а с другой стороны сценарии WSH, работающие с объектными моделями ADSI и WMI, оказались слишком сложными для пользователей среднего уровня и начинающих администраторов.

В 2000 году была начата разработка новой оболочки для доступа к объектам WMI из командной строки (WMI Command-line, WMIC). Этот продукт оказался не особенно удачным, так как в нем акцент был сделан на функциональные особенности WMI, а не на удобство работы пользователя. Начав дорабатывать WMIC, специалисты Microsoft поняли, что можно реализовать оболочку, которая не ограничивалась бы только работой с объектами WMI, а также предоставляла бы доступ к любым классам платформы .NET Framework, обеспечивая тем самым возможность пользоваться из командной строки всеми мощными функциональными возможностями данной среды.

Перед разработчиками новой оболочки, получившей название Windows PowerShell, стояли следующие основные цели и задачи, которые были успешно решены:

- применение командной строки в качестве основного интерфейса администрирования;
- реализация модели ObjectFlow (элементом обмена информации является объект);
- переработка существующих команд, утилит и оболочки;
- интеграция командной строки, объектов COM, WMI и .NET;
- работа с произвольными источниками данных в командной строке по принципу файловой системы.

Вообще, самая важная идея, заложенная в PowerShell, состоит в том, что в командной строке вывод результатов команды представляет собой не текст (в смысле последовательности байтов), а объект (данные вместе со свойственными им методами). В силу этого работать в PowerShell становится

проще, чем в традиционных оболочках, так как не нужно выполнять никаких манипуляций по выделению нужной информации из символьного потока.

Отметим, что PowerShell одновременно является и оболочкой командной строки (пользователь работает в интерактивном режиме) и средой выполнения сценариев, которые пишутся на специальном языке PowerShell.

Интерактивный сеанс в PowerShell похож на работу в оболочке Unix-систем: все команды в PowerShell имеют подробную встроенную справку (для большинства команд приводятся примеры их использования), поддерживается функция автоматического завершения названий команд и их параметров при вводе с клавиатуры, для многих команд имеются псевдонимы, аналогичные названиям Unix-утилит (`ls`, `pwd`, `tee` и т.д.).

В целом, оболочка PowerShell намного удобнее и мощнее своих предшественников (cmd.exe и WSH), а основным недостатком, сдерживающим распространение нового инструмента, является тот факт, что PowerShell работает не во всех версиях операционной системы Windows. Оболочкой можно пользоваться только на версиях не ниже Windows XP Service Pack 2 с установленным пакетом .NET Framework 2.0.

Контрольные вопросы

Вопрос 1.

Вариант 1. В каких версиях операционной системы Windows можно пользоваться командными файлами?

a. во всех версиях Windows
b. в Windows NT и выше
c. в Windows XP и выше

Вариант 2. Какие версии операционной системы Windows поддерживают сервер сценариев WSH?

a. Windows 2000 и выше
b. все 32-разрядные версии Windows
c. Windows NT и выше

Вопрос 2.

Вариант 1. Какие языки можно использовать для написания сценариев WSH?

a. Microsoft VBScript
b. Microsoft JScript
c. Microsoft C#

Вариант 2. Можно ли написать сценарий WSH на языке Perl?

a. нет
b. да, Perl поддерживается по умолчанию
c. да, но требуется установка специального модуля поддержки Perl

---

Вопрос 3.

Вариант 1. Какое средство автоматизации предлагает собственную объектную модель?

a. Cmd.exe
b. Windows PowerShell
c. WSH

Вариант 2. Можно ли написать сценарий WSH на языке Python?

a. нет
b. да, Python поддерживается по умолчанию
c. да, но только после установки модуля поддержки Python

Лекция 2. Оболочка командной строки Windows. Интерпретатор Cmd.exe

В операционной системе Windows, как и в других операционных системах, интерактивные (набираемые с клавиатуры и сразу же выполняемые) команды выполняются с помощью так называемого *командного интерпретатора*, иначе называемого *командным процессором* или *оболочкой командной строки* (command shell). Командный интерпретатор или оболочка командной строки — это программа, которая, находясь в оперативной памяти, считывает набираемые вами команды и обрабатывает их. В Windows 9x, как и в MS-DOS, командный интерпретатор по умолчанию был представлен исполняемым файлом `command.comКомандный интерпретатор:command.com`. Начиная с версии Windows NT, в операционной системе реализован интерпретатор команд `Cmd.exe`, обладающий гораздо более мощными возможностями.

Запуск оболочки

В Windows NT/2000/XP файл `Cmd.exe`, как и другие исполняемые файлы, соответствующие внешним командам операционной системы, находятся в каталоге `%SystemRoot%\SYSTEM32` (значением переменной среды `%SystemRoot%` является системный каталог Windows, обычно `C:\Windows` или `C:\WinNT`). Для запуска командного интерпретатора (открытия нового сеанса командной строки) можно выбрать пункт **Выполнить…** (Run) в меню **Пуск** (Start), ввести имя файла `Cmd.exe` и нажать кнопку **OK**. В результате откроется новое окно, в котором можно запускать команды и видеть результат их работы.

Внутренние и внешние команды. Структура команд

Некоторые команды распознаются и выполняются непосредственно самим командным интерпретатором — такие команды называются *внутренними* (например, `COPY` или `DIR`) Другие команды операционной системы представляют собой отдельные программы, расположенные по умолчанию в том же каталоге, что и `Cmd.exe`, которые Windows загружает и выполняет аналогично другим программам. Такие команды называются *внешними* (например, `MORE` или `XCOPY`).

Рассмотрим структуру самой командной строки и принцип работы с ней. Для того чтобы выполнить команду, вы после приглашения командной строки (например, `C:\>`) вводите *имя* этой команды (регистр не важен), ее *параметры* и *ключи* (если они необходимы) и нажимаете клавишу <Enter>. Например:

```
C:\>COPY C:\myfile.txt A:\ /V
```

Имя команды здесь — `COPY`, параметры — `C:\myfile.txt` и `A:\`, а ключом является `/V`. Отметим, что в некоторых командах ключи могут начинаться не с символа `/`, а с символа – (минус), например, `-V`.

Многие команды Windows имеют большое количество дополнительных параметров и ключей, запомнить которые зачастую бывает трудно.

Большинство команд снабжено встроенной справкой, в которой кратко описываются назначение и синтаксис данной команды. Получить доступ к такой справке можно путем ввода команды с ключом /?. Например, если выполнить команду `ATTRIB /?`, то в окне MS-DOS мы увидим следующий текст:

```
Отображение и изменение атрибутов файлов.
ATTRIB [+R|-R] [+A|-A] [+S|-S] [+H|-H] [[диск:][путь]имя_файла]
[/S]
       +   Установка атрибута.
       -   Снятие атрибута.
       R   Атрибут "Только чтение".
       A   Атрибут "Архивный".
       S   Атрибут "Системный".
       H   Атрибут "Скрытый".
       /S  Обработка файлов во всех вложенных папках указанного
пути.
```

Для некоторых команд текст встроенной справки может быть довольно большим и не умещаться на одном экране. В этом случае помощь можно выводить последовательно по одному экрану с помощью команды `MORE` и *символа конвейеризации* |, например:

```
XCOPY /? | MORE
```

В этом случае после заполнения очередного экрана вывод помощи будет прерываться до нажатия любой клавиши. Кроме того, используя *символы перенаправления вывода* > и >>, можно текст, выводимый на экран, направить в текстовый файл для дальнейшего просмотра. Например, для вывода текста справки к команде `XCOPY` в текстовый файл `xcopy.txt`, используется следующая команда:

```
XCOPY /? > XCOPY.TXT
```

Замечание

Вместо имени файла можно указывать обозначения устройств компьютера. В Windows поддерживаются следующие имена устройств: `PRN` (принтер), `LPT1–LPT3` (соответствующие параллельные порты), `AUX` (устройство, присоединяемое к последовательному порту 1), `COM1–COM3` (соответствующие последовательные порты), `CON` (терминал: при вводе это клавиатура, при выводе — монитор), `NUL` (пустое устройство, все операции ввода/вывода для него игнорируются).

Перенаправление ввода/вывода и конвейеризация (композиция) команд

Рассмотрим более подробно поддерживаемые в Windows UNIX-подобные концепции переназначения устройств стандартного ввода/вывода и конвейерного выполнения команд.

С помощью *переназначения устройств ввода/вывода* одна программа может направить свой вывод на вход другой или перехватить вывод другой программы, используя его в качестве своих входных данных. Таким образом, имеется возможность передавать информацию от процесса к процессу при минимальных программных издержках. Практически это означает, что для

программ, которые используют стандартные входные и выходные устройства, операционная система позволяет:

- выводить сообщения программ не на экран (стандартный выходной поток), а в файл или на принтер (перенаправление вывода);
- читать входные данные не с клавиатуры (стандартный входной поток), а из заранее подготовленного файла (перенаправление ввода);
- передавать сообщения, выводимые одной программой, в качестве входных данных для другой программы (*конвейеризация* или *композиция команд*).

Из командной строки эти возможности реализуются следующим образом. Для того чтобы перенаправить текстовые сообщения, выводимые какой-либо командой, в текстовый файл, нужно использовать конструкцию

команда > имя_файла

Если при этом заданный для вывода файл уже существовал, то он перезаписывается (старое содержимое теряется), если не существовал — создается. Можно также не создавать файл заново, а дописывать информацию, выводимую командой, в конец существующего файла. Для этого команда перенаправления вывода должна быть задана так:

команда >> имя_файла

С помощью символа < можно прочитать входные данные для заданной команды не с клавиатуры, а из определенного (заранее подготовленного) файла:

```
команда < имя_файла
```

Приведем несколько примеров перенаправления ввода/вывода.

1. Вывод встроенной справки для команды `COPY` в файл `copy.txt`:

```
COPY /? > copy.txt
```

2. Добавление текста справки для команды `XCOPY` в файл `copy.txt`:

```
XCOPY /? >> copy.txt
```

3. Ввод новой даты из файла `date.txt` (`DATE` — это команда для просмотра и изменения системной даты):

```
DATE < date.txt
```

Если при выполнении определенной команды возникает ошибка, то сообщение об этом по умолчанию выводится на экран. В случае необходимости сообщения об ошибках (стандартный поток ошибок) можно перенаправить в текстовый файл с помощью конструкции

команда 2> имя_файла

В этом случае стандартный вывод будет производиться на экран. Также имеется возможность информационные сообщения и сообщения об ошибках выводить в один и тот же файл. Делается это следующим образом:

команда > имя_файла 2>&1

Например, в приведенной ниже команде стандартный выходной поток и стандартный поток ошибок перенаправляются в файл `copy.txt`:

```
XCOPY A:\1.txt C: > copy.txt 2>&1
```

Наконец, с помощью конструкции

```
команда1 | команда2
```

можно использовать сообщения, выводимые первой командой, в качестве входных данных для второй команды (конвейер команд).

Используя механизмы перенаправления ввода/вывода и конвейеризации, можно из командной строки посылать информацию на различные устройства и автоматизировать ответы на запросы, выдаваемые командами или программами, использующими стандартный ввод. Для решения таких задач подходит команда

```
ECHO [сообщение]
```

которая выводит `сообщение` на экран. Рассмотрим примеры использования этой команды.

1. Посылка символа прогона на принтер:

```
ECHO ^L > PRN
```

2. Удаление всех файлов в текущем каталоге без предупреждения (автоматический положительный ответ на запрос об удалении):

```
ECHO y | DEL *.*
```

3. Соединение по телефону из командной строки (модем связан с портом `COM2`):

```
ECHO ATDT 1(123)555-1234 > COM2
```

Команды `MORE` и `SORT`

Одной из наиболее часто использующихся команд, для работы с которой применяется перенаправление ввода/вывода и конвейеризация, является `MORE`. Эта команда считывает стандартный ввод из конвейера или перенаправленного файла и выводит информацию частями, размер каждой из которых не больше размера экрана. Используется `MORE` обычно для просмотра длинных файлов. Возможны три варианта синтаксиса этой команды:

```
MORE [диск:][путь]имя_файла
MORE < [диск:][путь]имя_файла
имя_команды | MORE
```

Параметр `[диск:][путь]имя_файла` определяет расположение и имя файла с просматриваемыми на экране данными. Параметр `имя_команды` задает команду, вывод которой отображается на экране (например, `DIR` или команда `TYPE`, использующаяся для вывода содержимого текстового файла на экран). Приведем два примера.

Для поэкранного просмотра помощи команды `DIR` используется команда:

```
XCOPY /? | MORE
```

Для поэкранного просмотра текстового файла news.txt возможны следующие варианты команд:

```
MORE news.txt
MORE < news.txt
TYPE news.txt | MORE
```

Другой распространенной командой, использующей перенаправление ввода/вывода и конвейеризацию, является `SORT`. Эта команда работает как

фильтр: она считывает символы в заданном столбце, упорядочивает их в возрастающем или убывающем порядке и выводит отсортированную информацию в файл, на экран или другое устройство. Возможны два варианта синтаксиса этой команды:

```
SORT [/R] [/+n] [[диск1:][путь1]файл1] [> [диск2:][путь2]файл2]
```

или

```
[команда |] SORT [/R] [/+n] [> [диск2:][путь2]файл2]
```

В первом случае параметр *[диск1:][путь1]файл1* определяет имя файла, который нужно отсортировать. Во втором случае будут отсортированы выходные данные указанной команды. Если параметры файл1 или команда не заданы, то SORT будет считывать данные с устройства стандартного ввода.

Параметр *[диск2:][путь2]файл2* задает файл, в который будет направляется сортированный вывод; если этот параметр не задан, то вывод будет направлен на устройство стандартного вывода.

По умолчанию сортировка выполняется в порядке возрастания. Ключ /R позволяет изменить порядок сортировки на обратный (от Z к A и затем от 9 до 0). Например, для поэкранного просмотра отсортированного в обратном порядке файла price.txt, нужно задать следующую команду:

```
SORT /R < price.txt |MORE
```

Ключ /+n задает сортировку в файле по символам n-го столбца. Например, /+10 означает, что сортировка должна осуществляться, начиная с 10-й позиции в каждой строке. По умолчанию файл сортируется по первому столбцу.

Условное выполнение и группировка команд

В командной строке Windows NT/2000/XP можно использовать специальные символы, которые позволяют вводить несколько команд одновременно и управлять работой команд в зависимости от результатов их выполнения. С помощью таких символов условной обработки можно содержание небольшого пакетного файла записать в одной строке и выполнить полученную составную команду.

Используя символ амперсанта &, можно разделить несколько утилит в одной командной строке, при этом они будут выполняться друг за другом. Например, если набрать команду DIR & PAUSE & COPY /? и нажать клавишу <Enter>, то вначале на экран будет выведено содержимое текущего каталога, а после нажатия любой клавиши — встроенная справка команды COPY.

Символ ^ позволяет использовать командные символы как текст, то есть при этом происходит игнорирование значения специальных символов. Например, если ввести в командной строке

```
ECHO Абв & COPY /?
```

и нажать клавишу <Enter>, то произойдет выполнение подряд двух команд: ECHO Абв и COPY /? (команда ECHO выводит на экран символы, указанные в командной строке после нее). Если же выполнить команду

```
ECHO Абв ^& COPY /?
```

то на экран будет выведено

```
Абв & COPY /?
```

В этом случае просто выполняется одна команда `ECHO` с соответствующими параметрами.

Условная обработка команд в Windows осуществляется с помощью символов `&&` и `||` следующим образом. Двойной амперсант `&&` запускает команду, стоящую за ним в командной строке, только в том случае, если команда, стоящая перед амперсантами была выполнена успешно. Например, если в корневом каталоге диска `C:` есть файл `plan.txt`, то выполнение строки `TYPE C:\plan.txt && DIR` приведет к выводу на экран этого файла и содержимого текущего каталога. Если же файл `C:\plan.txt` не существует, то команда `DIR` выполняться не будет.

Два символа `||` осуществляют в командной строке обратное действие, т.е. запускают команду, стоящую за этими символами, только в том случае, если команда, идущая перед ними, не была успешно выполнена. Таким образом, если в предыдущем примере файл `C:\plan.txt` будет отсутствовать, то в результате выполнения строки `TYPE C:\plan.txt || DIR` на экран выведется содержимое текущего каталога.

Отметим, что условная обработка действует только на ближайшую команду, то есть в строке

```
TYPE C:\plan.txt && DIR & COPY /?
```

команда `COPY /?` запустится в любом случае, независимо от результата выполнения команды `TYPE C:\plan.txt`.

Несколько утилит можно сгруппировать в командной строке с помощью скобок. Рассмотрим, например, две строки:

```
TYPE C:\plan.txt && DIR & COPY /?
TYPE C:\plan.txt && (DIR & COPY /?)
```

В первой из них символ условной обработки `&&` действует только на команду `DIR`, во второй — одновременно на две команды: `DIR` и `COPY`.

Контрольные вопросы

Вопрос 1.

Вариант 1. В каком каталоге в Windows XP хранится файл cmd.exe?

a. %SystemRoot%
b. %SystemRoot%\System32
c. %SystemRoot%\System

Вариант 2. С помощью каких символов можно перенаправить выходной поток команды во внешний файл с сохранением прежнего содержимого этого файла?

a. >
b. >>
c. |

Вопрос 2.

Вариант 1. Какой командой можно создать файл Dir_Help.txt с описанием команды DIR?

a. dir /? | Dir_Help.txt
b. dir > Dir_Help.txt
c. dir /? > Dir_Help.txt

Вариант 2. Какой командой можно создать файл xcopy_help.txt с описанием всех параметров команды XCOPY?

a. xcopy -help > xcopy_help.txt
b. xcopy /? > xcopy_help.txt
c. xcopy /? & xcopy_help.txt

Вопрос 3.

Вариант 1. Что выведется на экран в результате выполнения команды: echo 1111 > c:\klop.txt & type c:\klop.txt > nul && echo 2222 ?

a. ничего не выведется
b. строка 2222
c. строки 1111 и 2222

Вариант 2. Что выведется на экран в результате выполнения команды: echo 1111 > c:\klop.txt & type c:\klop.txt > nul || echo 2222 ?

a. ничего не выведется
b. строка 2222
c. строки 1111 и 2222

Лекция 3. Команды для работы с файловой системой

Рассмотрим некоторые наиболее часто используемые команды для работы с файловой системой. Отметим сначала несколько особенностей определения путей к файлам в Windows.

Пути к объектам файловой системы

Напомним, что файловая система логически имеет древовидную структуру и имена файлов задаются в формате *[диск:][путь\]имя_файла*, то есть обязательным параметром является только имя файла. При этом, если путь начинается с символа "\", то маршрут вычисляется от корневого каталога, иначе — от текущего каталога. Например, имя `C:123.txt` задает файл `123.txt` в текущем каталоге на диске `C:`, имя `C:\123.txt` — файл `123.txt` в корневом каталоге на диске `C:`, имя `ABC\123.txt` — файл `123.txt` в подкаталоге `ABC` текущего каталога.

Существуют особые обозначения для текущего каталога и родительского каталогов. Текущий каталог обозначается символом `.` (точка), его родительский каталог — символами `..` (две точки). Например, если текущим каталогом является `C:\WINDOWS`, то путь к файлу `autoexec.bat` в корневом каталоге диска `C:` может быть записан в виде `..\autoexec.bat`.

В именах файлов (но не дисков или каталогов) можно применять так называемые групповые символы или шаблоны: `?` (вопросительный знак) и `*` (звездочка). Символ `*` в имени файла означает произвольное количество любых допустимых символов, символ `?` — один произвольный символ или его отсутствие. Скажем, под шаблон `text??1.txt` подходят, например, имена `text121.txt` и `text11.txt`, под шаблон `text*.txt` — имена `text.txt`, `textab12.txt`, а под шаблон `text.*` — все файлы с именем `text` и произвольным расширением.

Для того чтобы использовать длинные имена файлов при работе с командной строкой, их нужно заключать в двойные кавычки. Например, чтобы запустить файл с именем `'Мое приложение.exe'` из каталога `'Мои документы'`, нужно в командной строке набрать `"C:\Мои документы\Мое приложение.exe"` и нажать клавишу <Enter>.

Перейдем теперь непосредственно к командам для работы с файловой системой.

Команда CD

Текущий каталог можно изменить с помощью команды

```
CD [диск:][путь\]
```

Путь к требуемому каталогу указывается с учетом приведенных выше замечаний. Например, команда `CD \` выполняет переход в корневой каталог текущего диска. Если запустить команду `CD` без параметров, то на экран будут выведены имена текущего диска и каталога.

Команда COPY

Одной из наиболее часто повторяющихся задач при работе на компьютере является копирование и перемещение файлов из одного места в другое. Для копирования одного или нескольких файлов используется команда `COPY`.

Синтаксис этой команды:

```
COPY [/A|/B] источник [/A|/B] [+ источник [/A|/B] [+ ...]] [результат [/A|/B]] [/V][/Y|/-Y]
```

Краткое описание параметров и ключей команды `COPY` приведено в табл. 2.1.

Таблица 2.1. *Параметры и ключи команды* `COPY`

Параметр	Описание
`источник`	Имя копируемого файла или файлов
`/A`	Файл является текстовым файлом ASCII, то есть конец файла обозначается символом с кодом ASCII 26 (<Ctrl>+<Z>)
`/B`	Файл является двоичным. Этот ключ указывает на то, что интерпретатор команд должен при копировании считывать из источника число байт, заданное размером в каталоге копируемого файла
`результат`	Каталог для размещения результата копирования и/или имя создаваемого файла
`/V`	Проверка правильности копирования путем сравнения файлов после копирования
`/Y`	Отключение режима запроса подтверждения на замену файлов
`/-Y`	Включение режима запроса подтверждения на замену файлов

Приведем примеры использования команды `COPY`.

1. Копирование файла `abc.txt` из текущего каталога в каталог `D:\PROGRAM` под тем же именем:

```
COPY abc.txt D:\PROGRAM
```

2. Копирование файла `abc.txt` из текущего каталога в каталог `D:\PROGRAM` под новым именем `def.txt`:

```
COPY abc.txt D:\PROGRAM\def.txt
```

3. Копирование всех файлов с расширением `txt` с диска `A:` в каталог `'Мои документы'` на диске `C:`

```
COPY A:\*.txt "C:\Мои документы"
```

Если не задать в команде целевой файл, то команда `COPY` создаст копию файла-источника с тем же именем, датой и временем создания, что и исходный файл, и поместит новую копию в текущий каталог на текущем диске. Например, для того чтобы скопировать все файлы из корневого каталога диска `A:` в текущий каталог, достаточно выполнить такую краткую команду:

```
COPY A:\*.*
```

В качестве источника или результата при копировании можно указывать имена не только файлов, но и устройств компьютера. Например, для того чтобы распечатать файл `abc.txt` на принтере, можно

воспользоваться командой копирования этого файла на устройство PRN: COPY abc.txt PRN

Другой интересный пример: создадим новый текстовый файл и запишем в него информацию, без использования текстового редактора. Для этого достаточно ввести команду COPY CON my.txt, которая будет копировать то, что вы набираете на клавиатуре, в файл my.txt (если этот файл существовал, то он перезапишется, иначе — создастся). Для завершения ввода необходимо ввести символ конца файла, то есть нажать клавиши <Ctrl>+<Z>.

Команда COPY может также объединять (склеивать) нескольких файлов в один. Для этого необходимо указать единственный результирующий файл и несколько исходных. Это достигается путем использования групповых знаков (? и *) или формата файл1 + файл2 + файл3. Например, для объединения файлов 1.txt и 2.txt в файл 3.txt можно задать следующую команду:

```
COPY 1.txt+2.txt 3.txt
```

Объединение всех файлов с расширением dat из текущего каталога в один файл all.dat может быть произведено так:

```
COPY /B *.dat all.dat
```

Ключ /B здесь используется для предотвращения усечения соединяемых файлов, так как при комбинировании файлов команда COPY по умолчанию считает файлами текстовыми.

Если имя целевого файла совпадает с именем одного из копируемых файлов (кроме первого), то исходное содержимое целевого файла теряется. Если имя целевого файла опущено, то в его качестве используется первый файл из списка. Например, команда COPY 1.txt+2.txt добавит к содержимому файла 1.txt содержимое файла 2.txt. Командой COPY можно воспользоваться и для присвоения какому-либо файлу текущей даты и времени без модификации его содержимого. Для этого нужно ввести команду типа

```
COPY /B 1.txt +,,
```

Здесь запятые указывают на пропуск параметра приемника, что и приводит к требуемому результату.

Команда COPY имеет и свои недостатки. Например, с ее помощью нельзя копировать скрытые и системные файлы, файлы нулевой длины, файлы из подкаталогов. Кроме того, если при копировании группы файлов COPY встретит файл, который в данный момент нельзя скопировать (например, он занят другим приложением), то процесс копирования полностью прервется, и остальные файлы не будут скопированы.

Команда XCOPY

Указанные при описании команды COPY проблемы можно решить с помощью команды XCOPY, которая предоставляет намного больше возможностей при копировании. Необходимо отметить, правда, что XCOPY может работать только с файлами и каталогами, но не с устройствами.

Синтаксис этой команды:

```
XCOPY источник [результат] [ключи]
```

Команда XCOPY имеет множество ключей, мы коснемся лишь некоторых из них. Ключ /D[:[дата]] позволяет копировать только файлы, измененные не ранее указанной даты. Если параметр дата не указан, то копирование будет производиться только если источник новее результата. Например, команда

```
XCOPY "C:\Мои документы\*.*" "D:\BACKUP\Мои документы" /D
```

скопирует в каталог 'D:\BACKUP\Мои документы' только те файлы из каталога 'C:\Мои документы', которые были изменены со времени последнего подобного копирования или которых вообще не было в 'D:\BACKUP\Мои документы'.

Ключ /S позволяет копировать все непустые подкаталоги в каталоге-источнике. С помощью же ключа /E можно копировать вообще все подкаталоги, включая и пустые.

Если указан ключ /C, то копирование будет продолжаться даже в случае возникновения ошибок. Это бывает очень полезным при операциях копирования, производимых над группами файлов, например, при резервном копировании данных.

Ключ /I важен для случая, когда копируются несколько файлов, а файл назначения отсутствует. При задании этого ключа команда XCOPY считает, что файл назначения должен быть каталогом. Например, если задать ключ /I в команде копирования всех файлов с расширением txt из текущего каталога в несуществующий еще подкаталог TEXT,

```
XCOPY  *.txt TEXT /I
```

то подкаталог TEXT будет создан без дополнительных запросов.

Ключи /Q, /F и /L отвечают за режим отображения при копировании. При задании ключа /Q имена файлов при копировании не отображаются, ключа /F — отображаются полные пути источника и результата. Ключ /L обозначает, что отображаются только файлы, которые должны быть скопированы (при этом само копирование не производится).

С помощью ключа /H можно копировать скрытые и системные файлы, а с помощью ключа /R — заменять файлы с атрибутом "Только для чтения". Например, для копирования всех файлов из корневого каталога диска C: (включая системные и скрытые) в каталог SYS на диске D:, нужно ввести следующую команду:

```
XCOPY C:\*.* D:\SYS /H
```

Ключ /T позволяет применять XCOPY для копирования только структуры каталогов источника, без дублирования находящихся в этих каталогах файлов, причем пустые каталоги и подкаталоги не включаются. Для того чтобы все же включить пустые каталоги и подкаталоги, нужно использовать комбинацию ключей /T /E.

Используя XCOPY, можно при копировании обновлять только уже существующие файлы (новые файлы при этом не записываются). Для этого применяется ключ /U. Например, если в каталоге C:\2 находились файлы a.txt и b.txt, а в каталоге C:\1 — файлы a.txt, b.txt, c.txt и d.txt, то после выполнения команды

```
XCOPY C:\1 C:\2 /U
```

в каталоге C:\2 по-прежнему останутся лишь два файла a.txt и b.txt, содержимое которых будет заменено содержимым соответствующих файлов из каталога C:\1.Если с помощью XCOPY копировался файл с атрибутом "Только для чтения", то по умолчанию у файла-копии этот атрибут снимется. Для того чтобы копировать не только данные, но и полностью атрибуты файла, необходимо использовать ключ /K.

Ключи /Y и /-Y определяют, нужно ли запрашивать подтверждение перед заменой файлов при копировании. /Y означает, что такой запрос нужен, /-Y — не нужен.

Команда DIR

Еще одной очень полезной командой является DIR [диск:][путь][имя_файла] [ключи], которая используется для вывода информации о содержимом дисков и каталогов. Параметр [диск:][путь] задает диск и каталог, содержимое которого нужно вывести на экран. Параметр [имя_файла] задает файл или группу файлов, которые нужно включить в список. Например, команда

```
DIR C:\*.bat
```

выведет на экран все файлы с расширением bat в корневом каталоге диска C:. Если задать эту команду без параметров, то выводится метка диска и его серийный номер, имена (в коротком и длинном вариантах) файлов и подкаталогов, находящихся в текущем каталоге, а также дата и время их последней модификации. После этого выводится число файлов в каталоге, общий объем (в байтах), занимаемый файлами, и объем свободного пространства на диске. Например:

```
Том в устройстве C имеет метку PHYS1_PART2
 Серийный номер тома: 366D-6107
 Содержимое папки C:\aditor
.              <ПАПКА>      25.01.00  17:15 .
..             <ПАПКА>      25.01.00  17:15 ..
TEMPLT02 DAT           227  07.08.98   1:00 templt02.dat
UNINST1  000         1 093  02.03.99   8:36 UNINST1.000
HILITE   DAT         1 082  18.09.98  18:55 hilite.dat
TEMPLT01 DAT            48  07.08.98   1:00 templt01.dat
UNINST0  000        40 960  15.04.98   2:08 UNINST0.000
TTABLE   DAT           357  07.08.98   1:00 ttable.dat
ADITOR   EXE       461 312  01.12.99  23:13 aditor.exe
README   TXT         3 974  25.01.00  17:26 readme.txt
ADITOR   HLP        24 594  08.10.98  23:12 aditor.hlp
TEKCTO~1 TXT                 0   11.03.01    9:02 Текстовый
файл.txt
          11 файлов        533 647 байт
           2 папок     143 261 696 байт свободно
```

С помощью ключей команды DIR можно задать различные режима расположения, фильтрации и сортировки. Например, при использовании ключа /W перечень файлов выводится в широком формате с максимально возможным числом имен файлов или каталогов на каждой строке. Например:

```
Том в устройстве C имеет метку PHYS1_PART2
```

```
 Серийный номер тома: 366D-6107
 Содержимое папки C:\aditor
[.]             [..]            TEMPLT02.DAT    UNINST1.000
HILITE.DAT
TEMPLT01.DAT    UNINST0.000     TTABLE.DAT      ADITOR.EXE
README.TXT
ADITOR.HLP      ТЕКСТО~1.TXT
        11 файлов          533 647 байт
         2 папок     143 257 600 байт свободно
```

С помощью ключа `/A[[:]атрибуты]` можно вывести имена только тех каталогов и файлов, которые имеют заданные атрибуты (`R` — "Только чтение", `A` — "Архивный", `S` — "Системный", `H` — "Скрытый", префикс "-" имеет значение НЕ). Если ключ `/A` используется более чем с одним значением атрибута, будут выведены имена только тех файлов, у которых все атрибуты совпадают с заданными. Например, для вывода имен всех файлов в корневом каталоге диска `C:`, которые одновременно являются скрытыми и системными, можно задать команду

```
DIR C:\ /A:HS
```

а для вывода всех файлов, кроме скрытых — команду

```
DIR C:\ /A:-H
```

Отметим здесь, что атрибуту каталога соответствует буква `D`, то есть для того чтобы, например, вывести список всех каталогов диска `C:`, нужно задать команду

```
DIR C: /A:D
```

Ключ `/O[[:]сортировка]` задает порядок сортировки содержимого каталога при выводе его командой `DIR`. Если этот ключ опущен, `DIR` печатает имена файлов и каталогов в том порядке, в котором они содержатся в каталоге. Если ключ `/O` задан, а параметр `сортировка` не указан, то `DIR` выводит имена в алфавитном порядке. В параметре сортировка можно использовать следующие значения: `N` — по имени (алфавитная), `S` — по размеру (начиная с меньших), `E` — по расширению (алфавитная), `D` — по дате (начиная с более старых), `A` — по дате загрузки (начиная с более старых), `G` — начать список с каталогов. Префикс "-" означает обратный порядок. Если задается более одного значения порядка сортировки, файлы сортируются по первому `критерию`, затем по второму и т.д.

Ключ `/S` означает вывод списка файлов из заданного каталога и его подкаталогов.

Ключ `/B` перечисляет только названия каталогов и имена файлов (в длинном формате) по одному на строку, включая расширение. При этом выводится только основная информация, без итоговой. Например:

```
UNINST1.000
hilite.dat
templt01.dat
UNINST0.000
ttable.dat
aditor.exe
readme.txt
aditor.hlp
Текстовый файл.txt
```

Команды MKDIR и RMDIR

Для создания нового каталога и удаления уже существующего пустого каталога используются команды `MKDIR [диск:]путь` и `RMDIR [диск:]путь [ключи]` соответственно (или их короткие аналоги `MD` и `RD`). Например:

```
MKDIR "C:\Примеры"
RMDIR "C:\Примеры"
```

Команда `MKDIR` не может быть выполнена, если каталог или файл с заданным именем уже существует. Команда `RMDIR` не будет выполнена, если удаляемый каталог не пустой.

Команда DEL

Удалить один или несколько файлов можно с помощью команды

```
DEL [диск:][путь]имя_файла [ключи]
```

Для удаления сразу нескольких файлов используются групповые знаки `?` и `*`. Ключ `/S` позволяет удалить указанные файлы из всех подкаталогов, ключ `/F` – принудительно удалить файлы, доступные только для чтения, ключ `/A[[:]атрибуты]` – отбирать файлы для удаления по атрибутам (аналогично ключу `/A[[:]атрибуты]` в команде `DIR`).

Команда REN

Переименовать файлы и каталоги можно с помощью команды `RENAME` (`REN`). Синтаксис этой команды имеет следующий вид:

```
REN [диск:][путь][каталог1|файл1] [каталог2|файл2]
```

Здесь параметр `каталог1|файл1` определяет название каталога/файла, которое нужно изменить, а `каталог2|файл2` задает новое название каталога/файла. В любом параметре команды `REN` можно использовать групповые символы `?` и `*`. При этом представленные шаблонами символы в параметре `файл2` будут идентичны соответствующим символам в параметре `файл1`. Например, чтобы изменить у всех файлов с расширением `txt` в текущей директории расширение на `doc`, нужно ввести такую команду:

```
REN *.txt *.doc
```

Если файл с именем `файл2` уже существует, то команда `REN` прекратит выполнение, и произойдет вывод сообщения, что файл уже существует или занят. Кроме того, в команде `REN` нельзя указать другой диск или каталог для создания результирующих каталога и файла. Для этой цели нужно использовать команду `MOVE`, предназначенную для переименования и перемещения файлов и каталогов.

Команда MOVE

Синтаксис команды для перемещения одного или более файлов имеет вид:

```
MOVE [/Y|/-Y] [диск:][путь]имя_файла1[,...] результирующий_файл
```

Синтаксис команды для переименования папки имеет вид:

```
MOVE [/Y|/-Y] [диск:][путь]каталог1 каталог2
```

Здесь параметр `результирующий_файл` задает новое размещение файла и может включать имя диска, двоеточие, имя каталога, либо их сочетание.

Если перемещается только один файл, допускается указать его новое имя. Это позволяет сразу переместить и переименовать файл. Например,

```
MOVE "C:\Мои документы\список.txt" D:\list.txt
```

Если указан ключ `/-Y`, то при создании каталогов и замене файлов будет выдаваться запрос на подтверждение. Ключ `/Y` отменяет выдачу такого запроса.

Контрольные вопросы

Вопрос 1.

Вариант 1. Пусть текущим каталогом является C:\folder1\folder2\folder3. Какой командой можно переместиться в корневой каталог диска C?

a. cd \
b. cd c:
c. cd ..\..\..

Вариант 2. Пусть текущим каталогом является C:\folder1\folder2\folder3, а в каталоге folder2 имеется еще один подкаталог folder4. Какой командой можно переместиться в каталог folder4?

a. cd c:\folder1\folder2\folder4
b. cd ..\folder4
c. cd .\folder4
c. строки 1111 и 2222

Вопрос 2.

Вариант 1. Какой командой можно изменить расширения на js для файлов, находящихся в текущем каталоге, имеющих расширение vbs и имя которых состоит из одного или двух символов?

a. ren ??.vbs ??.js
b. ren ??.vbs *.js
c. ren *.vbs *.js
d. move ??.vbs *.js

Вариант 2. Пусть в текущем каталоге на диске C: хранятся файлы с расширением vbs. Какой командой можно переместить эти файлы в каталог E:\folder1 (с удалением первоначальных файлов)?

a. ren *.vbs e:\folder1\
b. move *.vbs e:\folder1\
c. copy *.vbs e:\folder1\
d. copy *.vbs e:\folder1\ & del *.vbs

Вопрос 3.

Вариант 1. Требуется сформировать файл doc_info.txt со списком всех файлов с расширением doc, находящихся в папке "C:\Мои документы" и всех ее подпапках. Какой командой можно это сделать?

a. dir /s /b C:\Мои документы*.doc > doc_info.txt
b. dir /b "C:\Мои документы*.doc" > doc_info.txt
c. dir /s /b "C:\Мои документы*.doc" > doc_info.txt

Вариант 2. Требуется сформировать файл hidden _info.txt со списком всех скрытых файлов, находящихся в папке "C:\Мои документы" и всех ее подпапках. Какой командой можно это сделать?

a. dir /s /a:h "C:\Мои документы*.*" > hidden_info.txt
b. dir /a:h "C:\Мои документы*.*" > hidden_info.txt
c. dir /s /a:h C:\Мои документы*.* > hidden_info.txt

Лекция 4. Командные файлы. Работа с переменными среды и параметрами командной строки

Язык оболочки командной строки (shell language) в Windows реализован в виде **командных** (или **пакетных**) **файлов**. Командный файл в Windows — это обычный текстовый файл с расширением `bat` или `cmd`, в котором записаны допустимые команды операционной системы (как внешние, так и внутренние), а также некоторые дополнительные инструкции и ключевые слова, придающие командным файлам некоторое сходство с алгоритмическими языками программирования. Например, если записать в файл `deltmp.bat` следующие команды:

```
C:\
CD %TEMP%
DEL /F *.tmp
```

и запустить его на выполнение (аналогично исполняемым файлам с расширением `com` или `exe`), то мы удалим все файлы во временной директории Windows. Таким образом, исполнение командного файла приводит к тому же результату, что и последовательный ввод записанных в нем команд. При этом не проводится никакой предварительной компиляции или проверки синтаксиса кода; если встречается строка с ошибочной командой, то она игнорируется. Очевидно, что если вам приходится часто выполнять одни и те же действия, то использование командных файлов может сэкономить много времени.

Вывод сообщений и дублирование команд

По умолчанию команды пакетного файла перед исполнением выводятся на экран, что выглядит не очень эстетично. С помощью команды `ECHO OFF` можно отключить дублирование команд, идущих после нее (сама команда `ECHO OFF` при этом все же дублируется). Например,

```
REM Следующие две команды будут дублироваться на экране …
DIR C:\
ECHO OFF
REM А остальные уже не будут
DIR D:\
```

Для восстановления режима дублирования используется команда ECHO ON. Кроме этого, можно отключить дублирование любой отдельной строки в командном файле, написав в начале этой строки символ `@`, например:

```
ECHO ON
REM Команда DIR C:\ дублируется на экране
DIR C:\
REM А команда DIR D:\ — нет
@DIR D:\
```

Таким образом, если поставить в самое начало файла команду

```
@ECHO OFF
```

то это решит все проблемы с дублированием команд.

В пакетном файле можно выводить на экран строки с сообщениями. Делается это с помощью команды

```
ECHO сообщение
```

Например,

```
@ECHO OFF
ECHO Привет!
```

Команда `ECHO.` (точка должна следовать непосредственно за словом "`ECHO`") выводит на экран пустую строку. Например,

```
@ECHO OFF
ECHO Привет!
ECHO.
ECHO Пока!
```

Часто бывает удобно для просмотра сообщений, выводимых из пакетного файла, предварительно полностью очистить экран командой `CLS`.

Используя механизм перенаправления ввода/вывода (символы > и >>), можно направить сообщения, выводимые командой `ECHO`, в определенный текстовый файл. Например:

```
@ECHO OFF
ECHO Привет! > hi.txt
ECHO Пока! >> hi.txt
```

Использование параметров командной строки

При запуске пакетных файлов в командной строке можно указывать произвольное число параметров, значения которых можно использовать внутри файла. Это позволяет, например, применять один и тот же командный файл для выполнения команд с различными параметрами.

Для доступа из командного файла к параметрам командной строки применяются символы `%0`, `%1`, ..., `%9` или `%*`. При этом вместо `%0` подставляется имя выполняемого пакетного файла, вместо `%1`, `%2`, ..., `%9` — значения первых девяти параметров командной строки соответственно, а вместо `%*` — все аргументы. Если в командной строке при вызове пакетного файла задано меньше девяти параметров, то "лишние" переменные из `%1` – `%9` замещаются пустыми строками. Рассмотрим следующий пример. Пусть имеется командный файл `copier.bat` следующего содержания:

```
@ECHO OFF
CLS
ECHO Файл %0 копирует каталог %1 в %2
XCOPY %1 %2 /S
```

Если запустить его из командной строки с двумя параметрами, например

```
copier.bat C:\Programs D:\Backup
```

то на экран выведется сообщение

```
Файл copier.bat копирует каталог C:\Programs в D:\Backup
```

и будет выполнено копирование каталога `C:\Programs` со всеми его подкаталогами в `D:\Backup`.

При необходимости можно использовать более девяти параметров командной строки. Это достигается с помощью команды SHIFT, которая изменяет значения замещаемых параметров с %0 по %9, копируя каждый параметр в предыдущий, то есть значение %1 копируется в %0, значение %2 – в %1 и т.д. Замещаемому параметру %9 присваивается значение параметра, следующего в командной строке за старым значением %9. Если же такой параметр не задан, то новое значение %9 — пустая строка.

Рассмотрим пример. Пусть командный файл my.bat вызван из командной строки следующим образом:

```
my.bat p1 p2 p3
```

Тогда %0=my.bat, %1=p1, %2=p2, %3=p3, параметры %4 – %9 являются пустыми строками. После выполнения команды SHIFT значения замещаемых параметров изменятся следующим образом: %0=p1, %1=p2, %2=p3, параметры %3 – %9 – пустые строки.

При включении расширенной обработки команд SHIFT поддерживает ключ /n, задающий начало сдвига параметров с номера n, где n может быть числом от 0 до 9.

Например, в следующей команде:

```
SHIFT /2
```

параметр %2 заменяется на %3, %3 на %4 и т.д., а параметры %0 и %1 остаются без изменений.

Команда, обратная SHIFT (обратный сдвиг), отсутствует. После выполнения SHIFT уже нельзя восстановить параметр (%0), который был первым перед сдвигом. Если в командной строке задано больше десяти параметров, то команду SHIFT можно использовать несколько раз.

В командных файлах имеются некоторые возможности синтаксического анализа заменяемых параметров. Для параметра с номером n (%n) допустимы синтаксические конструкции (операторы), представленные в табл. 3.1.

Таблица 3.1. *Операторы для заменяемых параметров*

Операторы	**Описание**
%~Fn	Переменная %n расширяется до полного имени файла
%~Dn	Из переменной %n выделяется только имя диска
%~Pn	Из переменной %n выделяется только путь к файлу
%~Nn	Из переменной %n выделяется только имя файла
%~Xn	Из переменной %n выделяется расширение имени файла
%~Sn	Значение операторов N и X для переменной %n изменяется так, что они работают с кратким именем файла
%~$PATH:n	Проводится поиск по каталогам, заданным в переменной среды PATH, и переменная %n заменяется на полное имя первого найденного файла. Если переменная PATH не определена или в результате поиска не найден ни один файл, эта конструкция заменяется на пустую строку. Естественно, здесь переменную PATH можно заменить на любое другое допустимое значение

Данные синтаксические конструкции можно объединять друг с другом, например:

`%~DPn` — из переменной `%n` выделяется имя диска и путь,

`%~NXn` — из переменной `%n` выделяется имя файла и расширение.

Рассмотрим следующий пример. Пусть мы находимся в каталоге `C:\TEXT` и запускаем пакетный файл с параметром `Рассказ.doc` (`%1=Рассказ.doc`). Тогда применение операторов, описанных в табл. 4.1, к параметру `%1` даст следующие результаты:

```
%~F1=C:\TEXT\Рассказ.doc
%~D1=C:
%~P1=\TEXT\
%~N1=Рассказ
%~X1=.doc
%DP1=C:\TEXT\
%NX1=Рассказ.doc
```

Работа с переменными среды

Внутри командных файлов можно работать с так называемыми *переменными среды* (или *переменными окружения*), каждая из которых хранится в оперативной памяти, имеет свое уникальное имя, а ее значением является строка. Стандартные переменные среды автоматически инициализируются в процессе загрузки операционной системы. Такими переменными являются, например, `WINDIR`, которая определяет расположение каталога Windows, `TEMP`, которая определяет путь к каталогу для хранения временных файлов Windows или `PATH`, в которой хранится *системный путь* (путь поиска), то есть список каталогов, в которых система должна искать выполняемые файлы или файлы совместного доступа (например, динамические библиотеки). Кроме того, в командных файлах с помощью команды `SET` можно объявлять собственные переменные среды.

Получение значения переменной

Для получения значения определенной переменной среды нужно имя этой переменной заключить в символы `%`. Например:

```
@ECHO OFF
CLS
REM Создание переменной MyVar
SET MyVar=Привет
REM Изменение переменной
SET MyVar=%MyVar%!
ECHO Значение переменной MyVar: %MyVar%
REM Удаление переменной MyVar
SET MyVar=
ECHO Значение переменной WinDir: %WinDir%
```

При запуске такого командного файла на экран выведется строка

```
Значение переменной MyVar: Привет!
Значение переменной WinDir: C:\WINDOWS
```

Преобразования переменных как строк

С переменными среды в командных файлах можно производить некоторые манипуляции. Во-первых, над ними можно производить операцию конкатенации (склеивания). Для этого нужно в команде `SET` просто написать рядом значения соединяемых переменных. Например,

```
SET A=Раз
SET B=Два
SET C=%A%%B%
```

После выполнения в файле этих команд значением переменной `C` будет являться строка '`РазДва`'. Не следует для конкатенации использовать знак `+`, так как он будет воспринят просто в качестве символа. Например, после запуска файл следующего содержания

```
SET A=Раз
SET B=Два
SET C=A+B
ECHO Переменная C=%C%
SET D=%A%+%B%
ECHO Переменная D=%D%
```

на экран выведутся две строки:

```
Переменная C=A+B
Переменная D=Раз+Два
```

Во-вторых, из переменной среды можно выделять подстроки с помощью конструкции `%имя_переменной:~n1,n2%`, где число `n1` определяет смещение (количество пропускаемых символов) от начала (если `n1` положительно) или от конца (если `n1` отрицательно) соответствующей переменной среды, а число `n2` – количество выделяемых символов (если `n2` положительно) или количество последних символов в переменной, которые не войдут в выделяемую подстроку (если `n2` отрицательно). Если указан только один отрицательный параметр `-n`, то будут извлечены последние `n` символов. Например, если в переменной `%DATE%` хранится строка "21.09.2007" (символьное представление текущая дата при определенных региональных настройках), то после выполнения следующих команд

```
SET dd1=%DATE:~0,2%
SET dd2=%DATE:~0,-8%
SET mm=%DATE:~-7,2%
SET yyyy=%DATE:~-4%
```

новые переменные будут иметь такие значения: `%dd1%`=21, `%dd2%`=21, `%mm%=09`, `%yyyy%`=2007.

В-третьих, можно выполнять процедуру замены подстрок с помощью конструкции `%имя_переменной:s1=s2%` (в результате будет возвращена строка, в которой каждое вхождение подстроки `s1` в соответствующей переменной среды заменено на `s2`). Например, после выполнения команд

```
SET a=123456
SET b=%a:23=99%
```

в переменной b будет храниться строка "199456". Если параметр s2 не указан, то подстрока s1 будет удалена из выводимой строки, т.е. после выполнения команды

```
SET a=123456
SET b=%a:23=%
```

в переменной b будет храниться строка "1456".

Операции с переменными как с числами

При включенной расширенной обработке команд (этот режим в Windows XP используется по умолчанию) имеется возможность рассматривать значения переменных среды как числа и производить с ними арифметические вычисления. Для этого используется команда SET с ключом /A. Приведем пример пакетного файла add.bat, складывающего два числа, заданных в качестве параметров командной строки, и выводящего полученную сумму на экран:

```
@ECHO OFF
REM В переменной M будет храниться сумма
SET /A M=%1+%2
ECHO Сумма %1 и %2 равна %M%
REM Удалим переменную M
SET M=
```

Локальные изменения переменных

Все изменения, производимые с помощью команды SET над переменными среды в командном файле, сохраняются и после завершения работы этого файла, но действуют только внутри текущего командного окна. Также имеется возможность локализовать изменения переменных среды внутри пакетного файла, то есть автоматически восстанавливать значения всех переменных в том виде, в каком они были до начала запуска этого файла. Для этого используются две команды: SETLOCAL и ENDLOCAL. Команда SETLOCAL определяет начало области локальных установок переменных среды. Другими словами, изменения среды, внесенные после выполнения SETLOCAL, будут являться локальными относительно текущего пакетного файла. Каждая команда SETLOCAL должна иметь соответствующую команду ENDLOCAL для восстановления прежних значений переменных среды. Изменения среды, внесенные после выполнения команды ENDLOCAL, уже не являются локальными относительно текущего пакетного файла; их прежние значения не будут восстановлены по завершении выполнения этого файла.

Связывание времени выполнения для переменных

При работе с составными выражениями (группы команд, заключенных в круглые скобки) нужно учитывать, что переменные среды в командных файлах используются в режиме раннего связывания. С точки зрения логики выполнения командного файла это может привести к ошибкам. Например, рассмотрим командный файл 1.bat со следующим содержимым:

```
SET a=1
ECHO a=%a%
```

```
SET a=2
ECHO a=%a%
```

и командный файл `2.bat`:

```
SET a=1
ECHO a=%a%
(SET a=2
ECHO a=%a% )
```

Казалось бы, результат выполнения этих двух файлов должен быть одинаковым: на экран выведутся две строки: "a=1" и "a=2". На самом же деле таким образом сработает только файл `1.bat`, а файл `2.bat` два раза выведет строку "a=1"!

Данную ошибку можно обойти, если для получения значения переменной вместо знаков процента (%) использовать восклицательный знак (!) и предварительно включить режим связывания времени выполнения командой `SETLOCAL ENABLEDELAYEDEXPANSION`. Таким образом, для корректной работы файл `2.bat` должен иметь следующий вид:

```
SETLOCAL ENABLEDELAYEDEXPANSION
SET a=1
ECHO a=%a%
(SET a=2
ECHO a=!a! )
```

Контрольные вопросы

Вопрос 1.

Вариант 1. Какое расширение могут иметь командные файлы?

a. cmd
b. txt
c. bat

Вариант 2. Можно ли в командных файлах пользоваться внутренними командами интерпретатора Cmd.exe и каким образом?

a. да, нужно просто указать требуемую команду
b. да, но для вызова внутренней команды нужно запускать новую копию командного интерпретатора
c. нет, нельзя

Вопрос 2.

Вариант 1. Какой смысл в командном файле имеет параметр %0 ?

a. имя запускаемого файла
b. первый параметр командной строки
c. последний параметр командной строки

Вариант 2. С помощью какого оператора можно выделить имя файла из первого параметра командной строки?

a. %~P1
b. %~N1
c. %~D1

Вопрос 3.

Вариант 1. Пусть значением переменной %SystemRoot% является строка "C:\Windows" (без кавычек). Какое значение будет иметь переменная s после выполнения в пакетном файле команды set s=%SystemRoot:~0,3% ?

a. C:
b. ows
c. C:\

Вариант 2. Пусть значением переменной %SystemRoot% является строка "C:\Windows" (без кавычек). Какое значение будет иметь переменная s после выполнения в пакетном файле команды set s=%SystemRoot:~-3% ?

a. C:\
b. ows
c. C:

Лекция 5. Управление работой командных файлов

Интерпретатор Cmd.exe обладает некоторыми возможностями для управления ходом выполнения командных файлов.

Приостановка выполнения командных файлов

Для того чтобы вручную прервать выполнение запущенного `bat`-файла, нужно нажать клавиши <Ctrl>+<C> или <Ctrl>+<Break>. Однако часто бывает необходимо программно приостановить выполнение командного файла в определенной строке с выдачей запроса на нажатие любой клавиши. Это делается с помощью команды `PAUSE`. Перед запуском этой команды полезно с помощью команды `ECHO` информировать пользователя о действиях, которые он должен произвести. Например:

```
ECHO Вставьте дискету в дисковод A: и нажмите любую клавишу
PAUSE
```

Команду `PAUSE` обязательно нужно использовать при выполнении потенциально опасных действий (удаление файлов, форматирование дисков и т.п.). Например,

```
ECHO Сейчас будут удалены все файлы в C:\Мои документы!
ECHO Для отмены нажмите Ctrl-C
PAUSE
DEL "C:\Мои документы\*.*"
```

Вызов внешних командных файлов

Из одного командного файла можно вызвать другой, просто указав его имя. Например:

```
@ECHO OFF
CLS
REM Вывод списка log-файлов
DIR C:\*.log
REM Передача выполнения файлу f.bat
f.bat
COPY A:\*.* C:\
PAUSE
```

Однако в этом случае после выполнения вызванного файла управление в вызывающий файл не передается, то есть в приведенном примере команда

```
COPY A:\*.* C:\
```

(и все следующие за ней команды) никогда не будет выполнена.

Для того чтобы вызвать внешний командный файл с последующим возвратом в первоначальный файл, нужно использовать специальную команду

```
CALL файл
```

Например:

```
@ECHO OFF
CLS
REM Вывод списка log-файлов
```

```
DIR C:\*.log
REM Передача выполнения файлу f.bat
CALL f.bat
COPY A:\*.* C:\
PAUSE
```

В этом случае после завершения работы файла `f.bat` управление вернется в первоначальный файл на строку, следующую за командой `CALL` (в нашем примере это команда `COPY A:\*.* C:\`).

Операторы перехода

Командный файл может содержать метки и команды `GOTO` перехода к этим меткам. Любая строка, начинающаяся с двоеточия `:`, воспринимается при обработке командного файла как метка. Имя метки задается набором символов, следующих за двоеточием до первого пробела или конца строки. Приведем пример.

Пусть имеется командный файл следующего содержания:

```
@ECHO OFF
COPY %1 %2
GOTO Label1
ECHO Эта строка никогда не выполнится
:Label1
REM Продолжение выполнения
DIR %2
```

После того, как в этом файле мы доходим до команды

```
GOTO Label1
```

его выполнение продолжается со строки

```
REM Продолжение выполнения
```

В команде перехода внутри файла `GOTO` можно задавать в качестве метки перехода строку `:EOF`, которая передает управление в конец текущего пакетного файла (это позволяет легко выйти из пакетного файла без определения каких-либо меток в самом его конце).

Также для перехода к метке внутри текущего командного файла кроме команды `GOTO` можно использовать и рассмотренную выше команду `CALL`:

```
CALL :метка аргументы
```

При вызове такой команды создается новый контекст текущего пакетного файла с заданными аргументами, и управление передается на инструкцию, расположенную сразу после метки. Для выхода из такого пакетного файла необходимо два раза достичь его конца. Первый выход возвращает управление на инструкцию, расположенную сразу после строки `CALL`, а второй выход завершает выполнение пакетного файла. Например, если запустить с параметром "`Копия-1`" командный файл следующего содержания:

```
@ECHO OFF
ECHO %1
CALL :2 Копия-2
:2
```

```
ECHO %1
```

то на экран выведутся три строки:

```
Копия-1
Копия-2
Копия-1
```

Таким образом, подобное использование команды CALL очень похоже на обычный вызов подпрограмм (процедур) в алгоритмических языках программирования.

Операторы условия

С помощью команды IF … ELSE (ключевое слово ELSE может отсутствовать) в пакетных файлах можно выполнять обработку условий нескольких типов. При этом если заданное после IF условие принимает истинное значение, система выполняет следующую за условием команду (или несколько команд, заключенных в круглые скобки), в противном случае выполняется команда (или несколько команд в скобках), следующие за ключевым словом ELSE.

Проверка значения переменной

Первый тип условия используется обычно для проверки значения переменной. Для этого применяются два варианта синтаксиса команды IF:

```
IF [NOT] строка1==строка2 команда1 [ELSE команда2]
```

(квадратные скобки указывают на необязательность заключенных в них параметров) или

```
IF [/I] [NOT] строка1 оператор_сравнения строка2 команда
```

Рассмотрим сначала первый вариант. Условие строка1==строка2 (здесь необходимо писать именно два знака равенства) считается истинным при точном совпадении обеих строк. Параметр NOT указывает на то, что заданная команда выполняется лишь в том случае, когда сравниваемые строки не совпадают.

Строки могут быть литеральными или представлять собой значения переменных (например, %1 или %TEMP%). Кавычки для литеральных строк не требуются. Например,

```
IF %1==%2 ECHO Параметры совпадают!
IF %1==Петя ECHO Привет, Петя!
```

Отметим, что при сравнении строк, заданных переменными, следует проявлять определенную осторожность. Дело в том, что значение переменной может оказаться пустой строкой, и тогда может возникнуть ситуация, при которой выполнение командного файла аварийно завершится. Например, если вы не определили с помощью команды SET переменную MyVar, а в файле имеется условный оператор типа

```
IF %MyVar%==C:\ ECHO Ура!!!
```

то в процессе выполнения вместо %MyVar% подставится пустая строка и возникнет синтаксическая ошибка. Такая же ситуация может возникнуть, если одна из сравниваемых строк является значением параметра командной строки, так как этот параметр может быть не указан при запуске командного

файла. Поэтому при сравнении строк нужно приписывать к ним в начале какой-нибудь символ, например:

```
IF -%MyVar%==-C:\ ECHO Ура!!!
```

С помощью команд `IF` и `SHIFT` можно в цикле обрабатывать все параметры командной строки файла, даже не зная заранее их количества. Например, следующий командный файл (назовем его `primer.bat`) выводит на экран имя запускаемого файла и все параметры командной строки:

```
@ECHO OFF
ECHO Выполняется файл: %0
ECHO.
ECHO Файл запущен со следующими параметрами...
REM Начало цикла
:BegLoop
IF -%1==- GOTO ExitLoop
ECHO %1
REM Сдвиг параметров
SHIFT
REM Переход на начало цикла
GOTO BegLoop
:ExitLoop
REM Выход из цикла
ECHO.
ECHO Все.
```

Если запустить `primer.bat` с четырьмя параметрами:

```
primer.bat А Б В Г
```

то в результате выполнения на экран выведется следующая информация:

```
Выполняется файл: primer.bat

Файл запущен со следующими параметрами:
А
Б
В
Г

Все.
```

Рассмотрим теперь оператор `IF` в следующем виде:

```
IF [/I] строка1 оператор_сравнения строка2 команда
```

Синтаксис и значение `операторов_сравнения` представлены в табл. 3.2.

Таблица 3.2. *Операторы сравнения в* `IF`

Оператор	**Значение**
`EQL`	Равно
`NEQ`	Не равно

LSS	Меньше
LEQ	Меньше или равно
GTR	Больше
GEQ	Больше или равно

Приведем пример использования операторов сравнения:

```
@ECHO OFF
CLS
IF -%1 EQL -Вася ECHO Привет, Вася!
IF -%1 NEQ -Вася ECHO Привет, но Вы не Вася!
```

Ключ `/I`, если он указан, задает сравнение текстовых строк без учета регистра. Ключ `/I` можно также использовать и в форме `строка1==строка2` команды `IF`. Например, условие

```
IF /I DOS==dos …
```

будет истинным.

Проверка существования заданного файла

Второй способ использования команды `IF` — это проверка существования заданного файла. Синтаксис для этого случая имеет вид:

```
IF [NOT] EXIST файл команда1 [ELSE команда2]
```

Условие считается истинным, если указанный файл существует. Кавычки для имени файла не требуются. Приведем пример командного файла, в котором с помощью такого варианта команды `IF` проверяется наличие файла, указанного в качестве параметра командной строки.

```
@ECHO OFF
IF -%1==- GOTO NoFileSpecified
IF NOT EXIST %1 GOTO FileNotExist

REM Вывод сообщения о найденном файле
ECHO Файл '%1' успешно найден.
GOTO :EOF

:NoFileSpecified
REM Файл запущен без параметров
ECHO В командной строке не указано имя файла.
GOTO :EOF

:FileNotExist
REM Параметр командной строки задан, но файл не найден
ECHO Файл '%1' не найден.
```

Проверка наличия переменной среды

Аналогично файлам команда `IF` позволяет проверить наличие в системе определенной переменной среды:

```
IF DEFINED переменная команда1 [ELSE команда2]
```

Здесь условие DEFINED применяется подобно условию EXISTS наличия заданного файла, но принимает в качестве аргумента имя переменной среды и возвращает истинное значение, если эта переменная определена. Например:

```
@ECHO OFF
CLS
IF DEFINED MyVar GOTO :VarExists
ECHO Переменная MyVar не определена
GOTO :EOF
:VarExists
ECHO Переменная MyVar определена,
ECHO ее значение равно %MyVar%
```

Проверка кода завершения предыдущей команды

Еще один способ использования команды IF — это проверка *кода завершения* (*кода выхода*) предыдущей команды. Синтаксис для IF в этом случае имеет следующий вид:

```
IF [NOT] ERRORLEVEL число команда1 [ELSE команда2]
```

Здесь условие считается истинным, если последняя запущенная команда или программа завершилась с кодом возврата, равным либо превышающим указанное число.

Составим, например, командный файл, который бы копировал файл my.txt на диск C: без вывода на экран сообщений о копировании, а в случае возникновения какой-либо ошибки выдавал предупреждение:

```
@ECHO OFF
XCOPY my.txt C:\ > NUL
REM Проверка кода завершения копирования
IF ERRORLEVEL 1 GOTO ErrOccurred
ECHO Копирование выполнено без ошибок.
GOTO :EOF

:ErrOccurred
ECHO При выполнении команды XCOPY возникла ошибка!
```

В операторе IF ERRORLEVEL … можно также применять операторы сравнения чисел, приведенные в табл. 3.2. Например:

```
IF ERRORLEVEL LEQ 1 GOTO Case1
```

Замечание.

Иногда более удобным для работы с кодами завершения программ может оказаться использование переменной %ERRORLEVEL%. (строковое представление текущего значения кода ошибки ERRORLEVEL).

Проверка версии реализации расширенной обработки команд

Наконец, для определения внутреннего номера версии текущей реализации расширенной обработки команд применяется оператор IF в следующем виде:

```
IF CMDEXTVERSION число команда1 [ELSE команда2]
```

Здесь условие CMDEXTVERSION применяется подобно условию ERRORLEVEL, но число сравнивается с вышеупомянутым внутренним номером

версии. Первая версия имеет номер 1. Номер версии будет увеличиваться на единицу при каждом добавлении существенных возможностей расширенной обработки команд. Если расширенная обработка команд отключена, условие `CMDEXTVERSION` никогда не бывает истинно.

Контрольные вопросы

Вопрос 1.

Вариант 1. Какой командой можно приостановить работу пакетного файла?

a. stop
b. pause
c. wait

Вариант 2. С помощью какого оператора можно завершить работу командного файла?

a. call :eof
b. goto :end
c. goto :eof

Вопрос 2.

Вариант 1. С помощью какого варианта команды if можно проверить наличие определенного файла?

a. if exists …
b. if defined …
c. if errorlevel …

Вариант 2. С помощью какого варианта команды if можно проверить наличие определенной переменной среды?

a. if exists …
b. if defined …
c. if errorlevel …

Лекция 6. Организация циклов в командных файлах

В командных файлах для организации циклов используются несколько разновидностей оператора FOR, которые обеспечивают следующие функции:

- выполнение заданной команды для всех элементов указанного множества;
- выполнение заданной команды для всех подходящих имен файлов;
- выполнение заданной команды для всех подходящих имен каталогов;
- выполнение заданной команды для определенного каталога, а также всех его подкаталогов;
- получение последовательности чисел с заданными начальным и конечным значениями, а также шагом приращения;
- чтение и обработка строк из текстового файла;
- обработка строк вывода определенной команды.

Цикл FOR … IN … DO …

Самый простой вариант синтаксиса команды FOR для командных файлов имеет следующий вид:

```
FOR %%переменная IN (множество) DO команда [параметры]
```

Внимание

Перед названием переменной должны стоять именно два знака процента (%%), а не один, как это было при использовании команды FOR непосредственно из командной строки.

Сразу приведем пример. Если в командном файле заданы строки

```
@ECHO OFF
FOR %%i IN (Раз,Два,Три) DO ECHO %%i
```

то в результате его выполнения на экране будет напечатано следующее:

```
Раз
Два
Три
```

Параметр множество в команде FOR задает одну или более текстовых строк, разделенных запятыми, которые вы хотите обработать с помощью заданной команды. Скобки здесь обязательны. Параметр команда [параметры] задает команду, выполняемую для каждого элемента *множества*, при этом вложенность команд FOR на одной строке не допускается. Если в строке, входящей во множество, используется запятая, то значение этой строки нужно заключить в кавычки. Например, в результате выполнения файла с командами

```
@ECHO OFF
FOR %%i IN ("Раз,Два",Три) DO ECHO %%i
```

на экран будет выведено

```
Раз,Два
Три
```

Параметр %%переменная представляет подставляемую переменную (счетчик цикла), причем здесь могут использоваться только имена переменных, состоящие из одной буквы. При выполнении команда FOR заменяет подставляемую переменную текстом каждой строки в заданном *множестве*, пока команда, стоящая после ключевого слова DO, не обработает все такие строки.

Замечание.
Чтобы избежать путаницы с параметрами командного файла %0 — %9, для переменных следует использовать любые символы кроме 0 – 9.

Параметр множество в команде FOR может также представлять одну или несколько групп файлов. Например, чтобы вывести в файл список всех файлов с расширениями txt и prn, находящихся в каталоге C:\TEXT, без использования команды DIR, можно использовать командный файл следующего содержания:

```
@ECHO OFF
FOR %%f IN (C:\TEXT\*.txt C:\TEXT\*.prn) DO ECHO %%f >>
list.txt
```

При таком использовании команды FOR процесс обработки продолжается, пока не обработаются все файлы (или группы файлов), указанные во *множестве*.

Цикл FOR /D … IN … DO …

Следующий вариант команды FOR реализуется с помощью ключа /D:

```
FOR /D %переменная IN (набор) DO команда [параметры]
```

В случае, если набор содержит подстановочные знаки, то команда выполняется для всех подходящих имен каталогов, а не имен файлов. Скажем, выполнив следующий командный файл:

```
@ECHO OFF
CLS
FOR /D %%f IN (C:\*.*) DO ECHO %%f
```

мы получим список всех каталогов на диске C:, например:

```
C:\Arc
C:\CYR
C:\MSCAN
C:\NC
C:\Program Files
C:\TEMP
C:\TeX
C:\WINNT
```

Цикл FOR /R … IN … DO …

С помощью ключа /R можно задать рекурсию в команде: FOR:

```
FOR /R [[диск:]путь] %переменная IN (набор) DO команда
[параметры]
```

В этом случае заданная команда выполняется для каталога [*диск*:]*путь*, а также для всех подкаталогов этого пути. Если после ключа /R не указано

имя каталога, то выполнение команды начинается с текущего каталога. Например, для распечатки всех файлов с расширением txt в текущем каталоге и всех его подкаталогах можно использовать следующий пакетный файл:

```
@ECHO OFF
CLS
FOR /R %%f IN (*.txt) DO PRINT %%f
```

Если вместо набора указана только точка (.), то команда проверяет все подкаталоги текущего каталога. Например, если мы находимся в каталоге C:\TEXT с двумя подкаталогами BOOKS и ARTICLES, то в результате выполнения файла:

```
@ECHO OFF
CLS
FOR /R %%f IN (.) DO ECHO %%f
```

на экран выведутся три строки:

```
C:\TEXT\.
C:\TEXT\BOOKS\.
C:\TEXT\ARTICLES\.
```

Цикл FOR /L … IN … DO …

Ключ /L позволяет реализовать с помощью команды FOR арифметический цикл, в этом случае синтаксис имеет следующий: вид:

```
FOR /L %переменная IN (начало,шаг,конец) DO команда [параметры]
```

Здесь заданная после ключевого слова IN тройка (начало,шаг,конец) раскрывается в последовательность чисел с заданными началом, концом и шагом приращения. Так, набор (1,1,5) раскрывается в (1 2 3 4 5), а набор (5,-1,1) заменяется на (5 4 3 2 1). Например, в результате выполнения следующего командного файла:

```
@ECHO OFF
CLS
FOR /L %%f IN (1,1,5) DO ECHO %%f
```

переменная цикла %%f пробежит значения от 1 до 5, и на экране напечатаются пять чисел:

```
1
2
3
4
5
```

Числа, получаемые в результате выполнения цикла FOR /L, можно использовать в арифметических вычислениях. Рассмотрим командный файл my.bat следующего содержания:

```
@ECHO OFF
CLS
FOR /L %%f IN (1,1,5) DO CALL :2 %%f
GOTO :EOF
:2
```

```
SET /A M=10*%1
ECHO 10*%1=%M%
```

В третьей строке в цикле происходит вызов нового контекста файла `my.bat` с текущим значением переменной цикла `%%f` в качестве параметра командной строки, причем управление передается на метку `:2` (*см. описание* `CALL` *в разделе "Изменения в командах перехода"*). В шестой строке переменная цикла умножается на десять, и результат записывается в переменную `M`. Таким образом, в результате выполнения этого файла выведется следующая информация:

```
10*1=10
10*2=20
10*3=30
10*4=40
10*5=50
```

Цикл FOR /F … IN … DO …

Самые мощные возможности (и одновременно самый запутанный синтаксис) имеет команда: `FOR` с ключом `/F`:

```
FOR /F ["ключи"] %переменная IN (набор) DO команда [параметры]
```

Здесь параметр `набор` содержит имена одного или нескольких файлов, которые по очереди открываются, читаются и обрабатываются. Обработка состоит в чтении файла, разбиении его на отдельные строки текста и выделении из каждой строки заданного числа подстрок. Затем найденная подстрока используется в качестве значения переменной при выполнении основного тела цикла (заданной команды).

По умолчанию ключ `/F` выделяет из каждой строки файла первое слово, очищенное от окружающих его пробелов. Пустые строки в файле пропускаются. Необязательный параметр `"ключи"` служит для переопределения заданных по умолчанию правил обработки строк. Ключи представляют собой заключенную в кавычки строку, содержащую приведенные в табл. 3.3 ключевые слова:

Таблица 3.3. *Ключи в команде* `FOR /F`

Ключ	Описание
`EOL=C`	Определение символа комментариев в начале строки (допускается задание только одного символа)
`SKIP=N`	Число пропускаемых при обработке строк в начале файла
`DELIMS=XXX`	Определение набора разделителей для замены заданных по умолчанию пробела и знака табуляции
`TOKENS=X,Y,M-N`	Определение номеров подстрок, выделяемых из каждой строки файла и передаваемых для выполнения в тело цикла

При использовании ключа `TOKENS=X,Y,M-N` создаются дополнительные переменные. Формат `M-N` представляет собой диапазон подстрок с номерами от `M` до `N`. Если последний символ в строке `TOKENS=` является звездочкой, то создается дополнительная переменная, значением которой будет весь текст, оставшийся в строке после обработки последней подстроки.

Разберем применение этой команды на примере пакетного файла parser.bat, который производит разбор файла myfile.txt:

```
@ECHO OFF
IF NOT EXIST myfile.txt GOTO :NoFile
     FOR /F "EOL=; TOKENS=2,3* DELIMS=, " %%i IN (myfile.txt)
DO @ECHO %%i %%j %%k
GOTO :EOF
:NoFile
ECHO Не найден файл myfile.txt!
```

Здесь во второй строке производится проверка наличия файла myfile.txt; в случае отсутствия этого файла выводится предупреждающее сообщение. Команда FOR в третьей строке обрабатывает файл myfile.txt следующим образом:

- Пропускаются все строки, которые начинаются с символа точки с запятой (EOL=;).
- Вторая и третья подстроки из каждой строки передаются в тело цикла, причем подстроки разделяются пробелами (по умолчанию) и/или запятыми (DELIMS=,).
- В теле цикла переменная %%i используется для второй подстроки, %%j — для третьей, а %%k получает все оставшиеся подстроки после третьей.

В нашем примере переменная %%i явно описана в инструкции FOR, а переменные %%j и %%k описываются неявно с помощью ключа TOKENS=. Например, если в файле myfile.txt были записаны следующие три строки:

```
ААА БББББ ВВВВ,ГГГГГ ДДДД
ЕЕЕЕЕ,ЖЖЖЖ ЗЗЗЗ
;КККК ЛЛЛЛЛ ММММММ
```

то в результате выполнения пакетного файла parser.bat на экран выведется следующее:

```
БББББ ВВВВ ГГГГГ ДДДД
ЖЖЖЖ ЗЗЗЗ
```

Замечание

Ключ TOKENS= позволяет извлечь из одной строки файла до 26 подстрок, поэтому запрещено использовать имена переменных, начинающиеся не с букв английского алфавита (a–z). Следует помнить, что имена переменных FOR являются глобальными, поэтому одновременно не может быть активно более 26 переменных.

Команда FOR /F также позволяет обработать отдельную строку. Для этого следует ввести нужную строку в кавычках вместо набора имен файлов в скобках. Строка будет обработана так, как будто она взята из файла. Например, файл следующего содержания:

```
@ECHO OFF
     FOR /F "EOL=; TOKENS=2,3* DELIMS=, " %%i IN ("ААА БББББ
ВВВВ,ГГГГГ ДДДД") DO @ECHO %%i %%j %%k
```

при своем выполнении напечатает

```
БББББ ВВВВ ГГГГГ ДДДД
```

Вместо явного задания строки для разбора можно пользоваться переменными среды, например:

```
@ECHO OFF
SET M=AAA БББББ ВВВВ,ГГГГГ ДДДД
     FOR /F "EOL=; TOKENS=2,3* DELIMS=, " %%i IN ("%M%") DO
@ECHO %%i %%j %%k
```

Наконец, команда `FOR /F` позволяет обработать строку вывода другой команды. Для этого следует вместо набора имен файлов в скобках ввести строку вызова команды в апострофах (не в кавычках!). Строка передается для выполнения интерпретатору команд `cmd.exe`, а вывод этой команды записывается в память и обрабатывается так, как будто строка вывода взята из файла. Например, следующий командный файл:

```
@ECHO OFF
CLS
ECHO Имена переменных среды:
ECHO.
FOR /F "DELIMS==" %%i IN ('SET') DO ECHO %%i
```

выведет перечень имен всех переменных среды, определенных в настоящее время в системе.

В цикле `FOR` допускается применение тех же синтаксических конструкций (операторов), что и для заменяемых параметров (табл. 3.4).

Таблица 3.4. *Операторы для переменных команды* `FOR`

Операторы	Описание
`%~Fi`	Переменная `%i` расширяется до полного имени файла
`%~Di`	Из переменной `%i` выделяется только имя диска
`%~Pi`	Из переменной `%i` выделяется только путь к файлу
`%~Ni`	Из переменной `%i` выделяется только имя файла
`%~Xi`	Из переменной `%i` выделяется расширение имени файла
`%~Si`	Значение операторов `N` и `X` для переменной `%i` изменяется так, что они работают с кратким именем файла

Замечание

Если планируется использовать расширения подстановки значений в команде `FOR`, то следует внимательно подбирать имена переменных, чтобы они не пересекались с обозначениями формата.

Например, если мы находимся в каталоге `C:\Program Files\Far` и запустим командный файл следующего содержания:

```
@ECHO OFF
CLS
FOR %%i IN (*.txt) DO ECHO %%~Fi
```

то на экран выведутся полные имена всех файлов с расширением `txt`:

```
C:\Program Files\Far\Contacts.txt
C:\Program Files\Far\FarFAQ.txt
C:\Program Files\Far\Far_Site.txt
```

```
C:\Program Files\Far\License.txt
C:\Program Files\Far\License.xUSSR.txt
C:\Program Files\Far\ReadMe.txt
C:\Program Files\Far\register.txt
C:\Program Files\Far\WhatsNew.txt
```

Циклы и связывание времени выполнения для переменных

Как и в рассмотренном выше примере с составными выражениями, при обработке переменных среды внутри цикла могут возникать труднообъяснимые ошибки, связанные с ранними связыванием переменных. Рассмотрим пример. Пусть имеется командный файл следующего содержания:

```
SET a=
FOR %%i IN (Раз,Два,Три) DO SET a=%a%%%i
ECHO a=%a%
```

В результате его выполнения на экран будет выведена строка "a=Три", то есть фактически команда

```
FOR %%i IN (Раз,Два,Три) DO SET a=%a%%%i
```

равносильна команде

```
FOR %%i IN (Раз,Два,Три) DO SET a=%%i
```

Для исправления ситуации нужно, как и в случае с составными выражениями, вместо знаков процента (%) использовать восклицательные знаки и предварительно включить режим связывания времени выполнения командой `SETLOCAL ENABLEDELAYEDEXPANSION`. Таким образом, наш пример следует переписать следующим образом:

```
SETLOCAL ENABLEDELAYEDEXPANSION
SET a=
FOR %%i IN (Раз,Два,Три) DO SET a=!a!%%i
ECHO a=%a%
```

В этом случае на экран будет выведена строка "a=РазДваТри".

Контрольные вопросы

Вопрос 1.

Вариант 1. Какой из указанных ниже операторов командного файла выведет на экран список всех папок на диске D:?

Ответ 1. for /d %%f in (D:*.*) do echo %f
+Ответ 2. for /d %%f in (D:*.*) do echo %%f
Ответ 3. for /r %%f in (D:*.*) do echo %%f

Вариант 2. Какой из указанных ниже операторов командного файла выведет на экран имена всех переменных среды?

+Ответ 1. for /f "delims==" %%i in ('set') do echo %%i
Ответ 2. for /f "delims==" %%i in ("set") do echo %%i
Ответ 3. for /f "delims==" %%i in (set) do echo %%i

Вопрос 2.

Вариант 1. Какой из указанных ниже операторов командного файла выведет на экран значения всех переменных среды?

Ответ 1. for /f %%i in ('set') do echo %%i

+Ответ 2. for /f "tokens=2 delims==" %%i in ('set') do echo %%i

Ответ 3. for /f "delims==" %%i in ('set') do echo %i

Вариант 2. С помощью какого из указанных ниже операторов командного файла можно изменить расширение с txt на bak у файлов в текущем каталоге и всех его подкаталогах?

a. for /R %%f in (*.txt) do ren "%%~NXf" "%%~Nf.bak"

b. for %%f in ('*.txt') do ren "%%~NXf" "%%~Nf.bak"

c. for %%f in (*.txt) do ren "%%~NXf" "%%~Nf.bak"

Практические задания

Вариант 1

1. Пусть даны текстовые файлы 1.txt, 2.txt, 3.txt следующего содержания:

```
1.txt
     Иванов              01  34
     Алексеев            02  20
2.txt
     Петров              10  22
     Сидоров             08  45
3.txt
     Абрамов             04  22
     Яковлев             34  68
```

   С помощью командного файла выполнить следующие действия. Скопировать файлы 1.txt, 2.txt, 3.txt в файл all.txt без символа конца файла. Отсортировать all.txt по фамилиям, результат поместить в файл names.txt. Убрать из names.txt данные на Иванова, отсортировать строки по возрастанию первого числового кода.
2. Написать командный файл для изменения в файле, имя которого указывается в качестве параметра командной строки, всех символов a на b.
3. Написать командный файл, который при запуске выводил бы на экран свое имя и все параметры командной строки с указанием их порядкового номера.
4. Написать командный файл, который печатал бы общее число переменных среды, определенных в системе, и после нажатия клавиши выводил на экран имена этих переменных (без значений) вместе с порядковым номером. Таким образом, на экран должна выводиться информация следующего вида:

```
Количество переменных в системе: 33
-----------------------------------
CLASSPATH
CLIPPER
COMPUTERNAME
COMSPEC
HOMEDRIVE
HOMEPATH
:
```

5. Создать командный файл, который выводил бы последнюю строку текстового файла, имя которого задается в качестве параметра командной строки. Если нужный файл не удастся найти, вывести соответствующее сообщение и прервать работу файла.
6. Написать командный файл для вычисления факториала натурального числа, который будет указываться в качестве параметра командной строки.

7. Пусть в каталоге записаны файлы вида ММДД*.* (номера месяца и дня в месяце, после которых идут произвольные символы). Нужно с помощью командного файла для каждой встречающейся даты создать подкаталог (имя ММДД) и переместить туда все соответствующие файлы.

Вариант 2

1. Написать командный файл, который будет копировать из текущего каталога все файлы с расширением txt, кроме одного файла, указанного в качестве второго параметра командной строки, в каталог, указанный первым параметром. Если имя каталога, в который должно производиться копирование, не задано, то вывести сообщение об этом и прервать выполнение файла.
2. С помощью командного файла создать в текущем каталоге подкаталог с именем в формате ГГГГММДД, совпадающим с текущей датой.
3. Написать командный файл для переименования файлов путем замены всех пробелов в их именах на символы подчеркивания. Имя изменяемого файла должно указываться в качестве параметра командной строки.
4. Вывести на рабочий стол пользователя текстовый файл, в названии которого был бы написан IP-адрес и имя компьютера. Работа данного командного файла не должна зависеть от имени активного пользователя.
5. Написать командный файл, который запускается с одним параметром командной строки и последовательно со смещением выводит подстроки этого параметра, например:

```
1.bat abcdef
abcdef
bcdef
cdef
def
ef
f
```

6. Написать команду для удаления лидирующих нулей из переменной или параметра командной строки, соответствующей числу (количество нулей может быть любым). Например, строка 00000123 должна поменяться на 123.
7. Найти в текущем каталоге и всех его подкаталогах все файлы нулевого размера и удалить их.

Ответы к контрольным вопросам

Лекция 1

Вопрос	Вариант 1	Вариант 2
1	a	b
2	a, b	c
3	c	c

Лекция 2

Вопрос	Вариант 1	Вариант 2
1	b	b
2	c	b
3	b	a

Лекция 3

Вопрос	Вариант 1	Вариант 2
1	a, c	a, b
2	a, b	b, d
3	c	a

Лекция 4

Вопрос	Вариант 1	Вариант 2
1	a, c	a
2	a	b
3	c	b

Лекция 5

Вопрос	Вариант 1	Вариант 2
1	b	c
2	a	b

Лекция 6

Вопрос	Вариант 1	Вариант 2
1	b	a
2	b	a

Библиографический список

1. Командные файлы и сценарии Windows Script Host / А.В. Попов. – СПб.: БХВ-Петербург, 2002. – 320 с.
2. Командная строка Microsoft Windows. Справочник администратора. / У.Р. Станек. – М.: Изд.-торг. дом "Русская редакция", 2004. – 480 с.

Содержание

Учебное издание

ПОПОВ Андрей Владимирович

ЯЗЫК КОМАНДНОГО ПРОЦЕССОРА CMD.EXE ОПЕРАЦИОННОЙ СИСТЕМЫ WINDOWS

Учебное пособие

Печатается в авторской редакции в соответствии с представленным оригиналом-макетом

Дизайн обложки

Подписано в печать 00.06.09. Формат 60×84 $\frac{1}{16}$. Усл. печ. л. 00,00.
Тираж ____ экз. Заказ №

Издательство Мордовского университета
Типография Мордовского университета
430005, г. Саранск, ул. Советская, 24