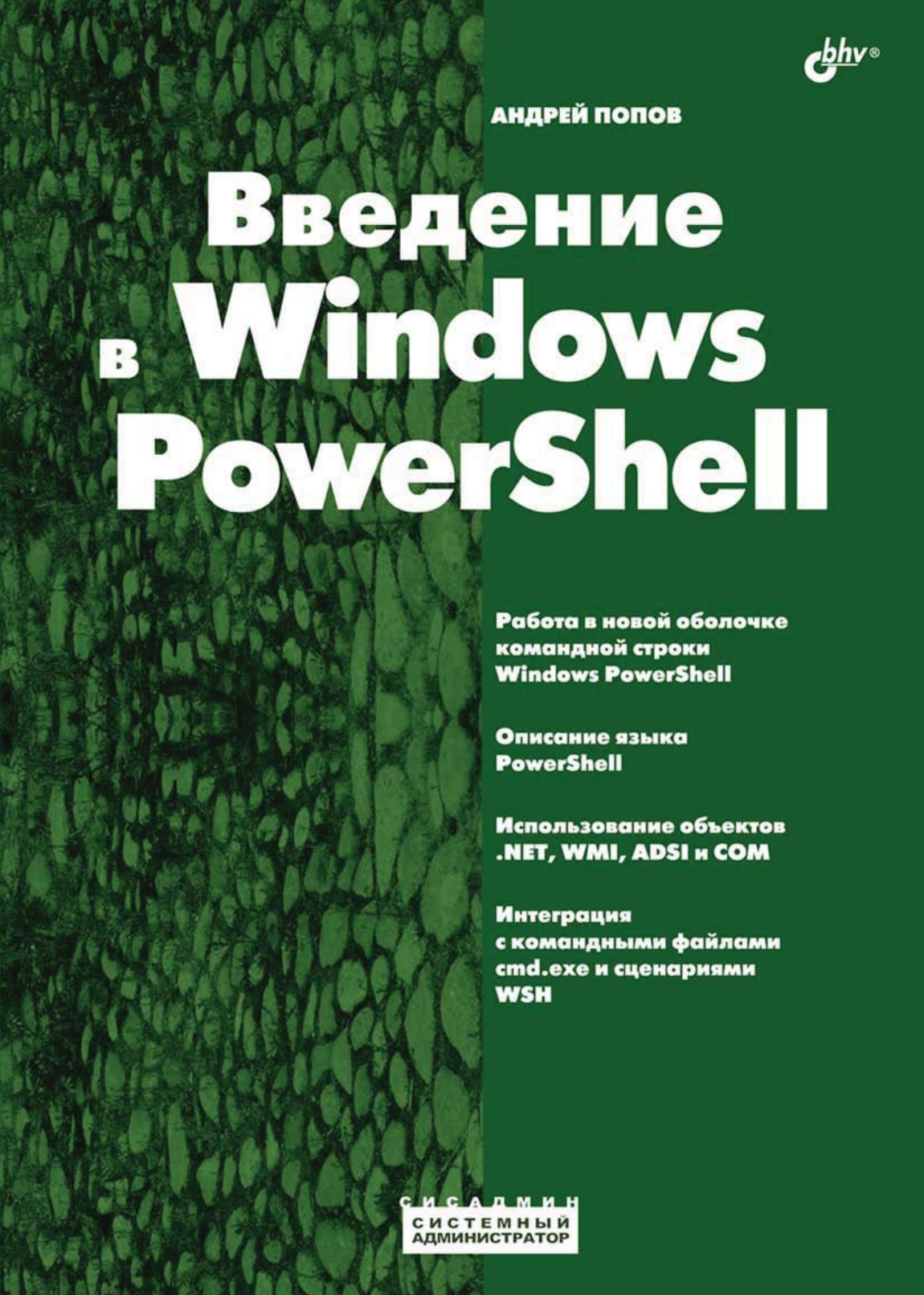
bhv
АНДРЕЙ ПОПОВ
Введение в Windows PowerShell
Работа в новой оболочке командной строки Windows PowerShell
Описание языка PowerShell
Использование объектов .NET, WMI, ADSI и COM
Интеграция с командными файлами cmd.exe и сценариями WSH
СИСАДМИН
СИСТЕМНЫЙ АДМИНИСТРАТОР

УДК 681.3.06
ББК 32.973.26-018.2
П58

Попов А. В.

П58 Введение в Windows PowerShell. — СПб.: БХВ-Петербург, 2009. — 464 с.: ил. — (Системный администратор)

ISBN 978-5-9775-0283-2

Рассматривается новая объектно-ориентированная оболочка командной строки Microsoft Windows PowerShell и ее возможности для автоматизации повседневных задач администрирования. Описываются основные элементы и конструкции языка PowerShell. Приводятся примеры использования объектов .NET, WMI, ADSI и COM. Обсуждаются вопросы совместного использования PowerShell, командных файлов интерпретатора cmd.exe и сценариев Windows Script Host. Даются примеры решения с помощью PowerShell задач администратора Windows.

Для администраторов, программистов и опытных пользователей Windows

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Юрий Якубович*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.08.08.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 37,41.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0283-2

Оглавление

Введение **1**

Что это за книга и для кого она предназначена 2
Структура книги 3
Принятые в книге соглашения 5

Часть I. Изучаем PowerShell **7**

Глава 1. Windows PowerShell — результат развития технологий автоматизации **9**

Предшественники PowerShell в мире Windows 9
Оболочка командной строки command.com/cmd.exe 12
Сервер сценариев Windows Script Host (WSH) 14
Оболочка WMI Command-line (WMIC) 17
Причины и цели создания оболочки PowerShell 18
Отличие PowerShell от других оболочек — ориентация на объекты 22

Глава 2. Первые шаги в PowerShell. Основные понятия **25**

Загрузка и установка PowerShell 25
Запуск оболочки 26
Работают ли знакомые команды? 26
Вычисление выражений 28
Типы команд PowerShell 29
Командлеты 30
Функции 35
Сценарии 36
Внешние исполняемые файлы 36
Псевдонимы команд 36
Диски PowerShell 40
Провайдеры PowerShell 41

Навигация по дискам PowerShell 43
Просмотр содержимого дисков и каталогов 44
Создание дисков 46

Глава 3. Приемы работы в оболочке 48

Редактирование в командном окне PowerShell 48
Автоматическое завершение команд 50
Справочная система PowerShell 52
Получение справки о командлетах 53
Справочная информация, не связанная с командлетами 55
История команд в сеансе работы 58
Протоколирование действий в сеансе работы 62

Глава 4. Настройка оболочки 65

Настройка ярлыка PowerShell 65
Программное изменение свойств консоли PowerShell 67
Цвета текста и фона 68
Заголовок командного окна 68
Размеры командного окна 69
Приглашение командной строки 70
Настройка пользовательских профилей 72
Политики выполнения сценариев 75

Глава 5. Работа с объектами 78

Конвейеризация объектов в PowerShell 78
Просмотр структуры объектов (командлет *Get-Member*) 80
Фильтрация объектов (командлет *Where-Object*) 82
Сортировка объектов (командлет *Sort-Object*) 85
Выделение объектов и свойств (командлет *Select-Object*) 87
Выполнение произвольных действий над объектами в конвейере (командлет *ForEach-Object*) 90
Группировка объектов (командлет *Group-Object*) 91
Измерение характеристик объектов (командлет *Measure-Object*) 92
Вызов статических методов 93
Управление выводом команд в PowerShell 95
Форматирование выводимой информации 96
Перенаправление выводимой информации 99

Глава 6. Переменные, массивы и хэш-таблицы 102

Числовые и символьные литералы 102
Числовые литералы 102
Символьные строки 103

Переменные PowerShell ... 107
Переменные оболочки PowerShell ... 108
Пользовательские переменные. Типы переменных ... 111
Переменные среды Windows ... 115
Массивы в PowerShell ... 117
Обращение к элементам массива ... 117
Операции с массивом ... 118
Хэш-таблицы (ассоциативные массивы) ... 121
Операции с хэш-таблицей ... 124

Глава 7. Операторы и управляющие инструкции ... 126

Арифметические операторы ... 126
Оператор сложения ... 127
Оператор умножения ... 129
Операторы вычитания, деления и остатка от деления ... 130
Операторы присваивания ... 131
Операторы сравнения ... 133
Операторы проверки на соответствие шаблону ... 135
Логические операторы ... 138
Управляющие инструкции языка PowerShell ... 139
Инструкция *If ... ElseIf ... Else* ... 139
Цикл *While* ... 140
Цикл *Do ... While* ... 141
Цикл *For* ... 141
Цикл *ForEach* ... 142
Метки циклов, инструкции *Break* и *Continue* ... 145
Инструкция *Switch* ... 146

Глава 8. Функции, фильтры и сценарии ... 152

Функции в PowerShell ... 152
Обработка аргументов функций с помощью переменной *$Args* ... 153
Формальные параметры функций ... 155
Возвращаемые значения ... 160
Функции внутри конвейера команд ... 162
Фильтры в PowerShell ... 163
Функции в качестве командлетов ... 164
Сценарии PowerShell ... 166
Создание и запуск сценариев PowerShell ... 167
Передача аргументов в сценарии ... 169
Выход из сценариев ... 170
Оформление сценариев. Комментарии ... 171

Глава 9. Обработка ошибок и отладка 175

Обработка ошибок 175
Объект ErrorRecord и поток ошибок 176
Сохранение объектов, соответствующих ошибкам 179
Мониторинг возникновения ошибок 182
Режимы обработок ошибок 183
Обработка "критических" ошибок (исключений) 185
Отладка сценариев 187
Вывод диагностических сообщений 187
Командлет Set-PSDebug 188
Трассировка выполнения команд 189
Пошаговое выполнение команд 191
Вложенная командная строка и точки прерывания 191

Часть II. Используем PowerShell 195

Глава 10. Доступ из PowerShell к внешним объектам (COM, WMI, .NET и ADSI) 197

Работа с COM-объектами 197
Внешние серверы автоматизации на примере Microsoft Office 202
Доступ к объектам WMI 208
Подключение к подсистеме WMI. Получение списка классов 209
Получение объектов WMI 211
Выполнение WQL-запросов 214
Использование объектов .NET 215
Доступ к службе каталогов ADSI 218

Глава 11. Работа с файловой системой 222

Навигация в файловой системе 222
Получение списка файлов и каталогов 223
Определение размера каталогов 227
Создание файлов и каталогов 228
Чтение и просмотр содержимого файлов 229
Запись файлов 230
Копирование файлов и каталогов 232
Переименование и перемещение файлов и каталогов 235
Удаление файлов и каталогов 236
Поиск текста в файлах 237
Замена текста в файлах 241

Глава 12. Управление процессами и службами 243

Управление процессами 243
Просмотр списка процессов 245
Определение библиотек, используемых процессом 250
Остановка процессов 252
Запуск процессов, изменение приоритетов выполнения 253
Завершение неотвечающих процессов 255
Управление службами 255
Просмотр списка служб 256
Остановка и приостановка служб 258
Запуск и перезапуск служб 259
Изменение параметров службы 259

Глава 13. Работа с системным реестром 261

Структура реестра 261
Просмотр локального реестра 263
Просмотр удаленного реестра 265
Модификация реестра 268
Создание нового раздела 269
Копирование разделов 269
Переименование раздела 270
Удаление раздела 270
Создание параметра 270
Изменение значения параметра 271
Переименование параметра 272
Копирование параметров 272
Очистка значения параметра 273
Удаление параметра 273

Глава 14. Работа с журналами событий 275

Инструменты для обработки журналов событий 277
Список журналов событий на локальном компьютере 279
Список журналов событий на удаленном компьютере 281
Просмотр событий из локального журнала 282
Вывод событий определенного типа 285
Отбор событий по идентификатору 286
Отбор событий по датам 287
Группировка событий по источнику возникновения 288
Просмотр событий из удаленного журнала 289

Настройка журналов событий....292
Установка максимального размера журналов....292
Установка режима хранения журналов....292
Очистка журнала....293

Глава 15. Управление рабочими станциями. Получение и анализ системной информации....295

Завершение сеанса пользователя....295
Перезагрузка и выключение компьютера....297
Получение информации о BIOS....298
Вывод списка команд, выполняемых при загрузке системы....299
Вывод свойств операционной системы....301
Вывод списка установленных программных продуктов....303
Вывод списка установленных обновлений операционной системы....306

Глава 16. Инвентаризация оборудования....309

Получение информации о физической памяти....309
Преобразование отчета в формат HTML....311
Получение информации о процессорах....313
Получение списка устройств Plug-and-Play....316
Получение информации о звуковой карте....319
Получение информации о видеокарте....320
Получение информации о сетевых адаптерах....323

Глава 17. Настройка сетевых параметров. Работа с электронной почтой....325

Получение и настройка сетевых параметров....325
Получение списка IP-адресов компьютера....326
Вывод параметров протокола TCP/IP....327
Настройка DHCP....332
Отправка сообщений по электронной почте....338

Глава 18. PowerShell, cmd.exe и VBScript: совместное использование....341

Сравнение языков PowerShell и cmd.exe....342
Различия в синтаксисе команд....342
Работа с переменными....344
Использование циклов....346
Вывод текста и запуск программ....347
Запуск команд cmd.exe из PowerShell....347

Сравнение языков PowerShell и VBScript 349
Обращение к функциям, командам и методам 349
Работа с переменными, массивами и объектами 351
Использование символьных строк 351
Прочие замечания по синтаксису 352
Аналоги PowerShell для функций VBScript 353
Математические функции 353
Символьные функции 355
Функции для работы с датами и временем 361
Использование из PowerShell кода VBScript 368
Использование из PowerShell кода JScript 370

Заключение 371

П Р И Л О Ж Е Н И Я 373

Приложение 1. Объектная модель WMI 375

Общая структура WMI 376
Ядро WMI 377
Провайдеры WMI 378
Менеджер объектов CIM 379
Репозиторий CIM. Пространства имен 381
Путь к классам и объектам CIM 384
Безопасность при работе с WMI 385
Структура классов WMI 389
Основные типы классов CIM 389
Свойства классов WMI 391
Методы классов WMI 397
Квалификаторы классов, свойств и методов 399
Интерактивная работа с объектами WMI 403
Тестер WMI (WBEMTest) 403
Административные утилиты WMI (WMI Tools) 404

Приложение 2. Полезные COM-объекты и примеры их использования 411

Управление проводником Windows с помощью объекта *Shell.Application* 411
Отображение специальных окон Проводника 413
Вызов элементов панели управления 418
Управление открытыми окнами 420

Использование объектов Windows Script Host 423
Работа с ресурсами локальной сети (объект *WScript.Network*) 423
Вывод информационного окна (объект *WScript.Shell*) 428
Переключение между приложениями, имитация нажатий клавиш (объект *WScript.Shell*) 430
Доступ к специальным папкам Windows (объект *WScript.Shell*) 436
Удаление некорректных ярлыков (объект *WScript.Shell*) 438

Ссылки на ресурсы Интернета 441

Сайты компании Microsoft 441
Другие сайты 441
Группы новостей 442
Блоги 442

Список литературы 443

Предметный указатель 445

Глава 1

Windows PowerShell — результат развития технологий автоматизации

Прежде чем приступить непосредственно к изучению оболочки командной строки Windows PowerShell, попробуем ответить на ряд вопросов. Для чего, собственно, компании Microsoft потребовалось создавать этот совершенно новый инструмент и язык программирования? Какую пользу он может принести обычным пользователям и системным администраторам? Почему недостаточно было существующих средств?

Чтобы понять это, посмотрим, как в операционной системе Windows обстояло дело с *автоматизацией работы*, то есть решением различных задач в автоматическом режиме, без участия человека, до появления PowerShell.

Предшественники PowerShell в мире Windows

В настоящее время графический интерфейс Windows стал настолько привычным, что многие пользователи и начинающие администраторы даже не задумываются об альтернативных способах управления операционной системой с помощью *командной строки* (command line) и различных *сценариев* (scripts). Зачастую они просто не знают о тех преимуществах, которые дают эти инструменты с точки зрения автоматизации работы.

Подобная ситуация обусловлена тем, что исторически командная строка всегда была слабым местом операционной системы Windows (по сравнению с UNIX-системами). Причиной этого, прежде всего, является то, что компания Microsoft изначально 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 хоть на другом континенте.

Однако подобная модель управления не является масштабируемой: если с помощью стандартных графических инструментов администрировать не один, а десять серверов, то последовательность изменений настроек в диалоговых окнах придется повторить десять раз. Следовательно, в этом случае остро встает вопрос об автоматизации выполнения рутинных операций (например, проведения инвентаризации оборудования и программного обеспечения, мониторинга работы служб, анализа 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 рутинные ежедневные 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! Однако 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 единст-

венную команду `dir` (с определенными ключами) и перенаправить вывод этой команды в нужный текстовый файл. Например, следующая команда создаст текстовый файл c:\list_mp3.txt, в котором будут записаны имена всех файлов с расширением mp3, находящихся в каталоге c:\music или в каком-либо его подкаталоге:

```
dir /s /b c:\music\*.mp3 > c:\list_mp3.txt
```

Задумаемся теперь, 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 (в идеальном случае 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 (sh) или ее модификации (bash, ksh, csh 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 одновременно поддерживаются несколько стандартных инструментов автоматизации, сильно отличающихся друг от друга: оболочка командной строки cmd.exe, среда выполнения сценариев Windows Script Host (WSH), оболочка WMI Command-line (WMIC) и, наконец, новинка и предмет нашего изучения — 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 всех доступных средств автоматизации.

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). Вообще, любую операционную систему можно представить в виде совокупности *ядра системы*, которое имеет доступ к аппаратуре и управляет 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 (рис. 1.1).

Замечание

С технической точки зрения оболочка представляет собой построчный интерпретатор простого языка сентенциального (директивного) программирования, в качестве операторов которого могут использоваться исполняемые программы.

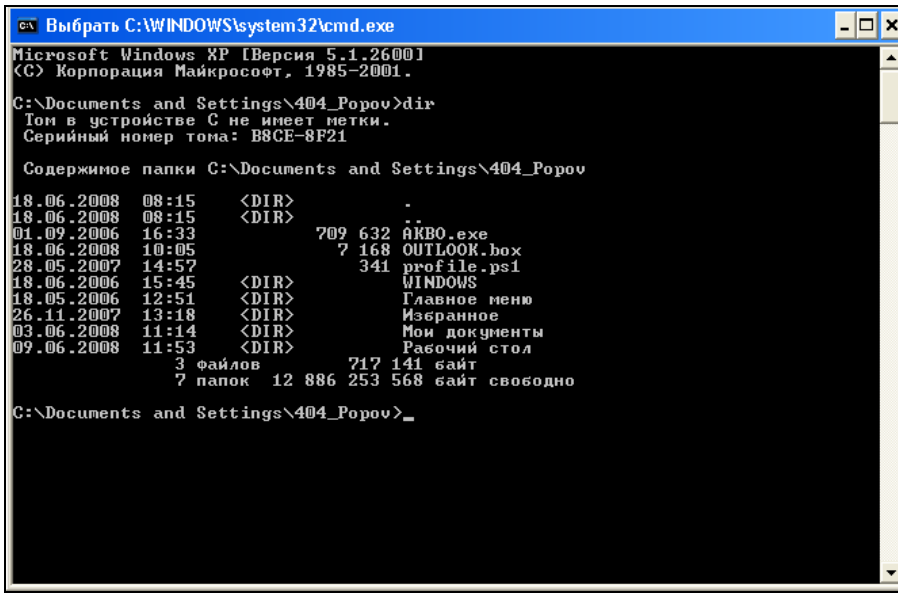

Рис. 1.1. Результат выполнения команды dir в оболочке cmd.exe

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, командный интерпретатор которой представлял 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 NT (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.

> ***Замечание***
>
> Для того чтобы прочитать встроенную справку для определенной команды, нужно вызвать эту команду с ключом `/?` (например, `xcopy /?`). Общий справочник по командной строке находится в файле %WinDir%\Help\Ntcmds.chm.

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 ранее, 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. При этом, однако, оболочка cmd.exe и командные файлы существенно проигрывают аналогичным инструментам в UNIX-системах и не обеспечивают доступ к объектным моделям, поддерживаемым операционной системой (COM, WMI, .NET).

Сервер сценариев Windows Script Host (WSH)

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. Например, в сценарии MakeShortcut.vbs с помощью объекта WshShell создается ярлык на сетевой ресурс: сайт компании Microsoft (листинг 1.1).

Листинг 1.1. Создание ярлыка из сценария (VBScript)

```
'*****************************************************************
'* Имя: MakeShortcut.vbs
'* Язык: VBScript
'* Описание: Создание ярлыков из сценария
'*****************************************************************
Dim WshShell,oUrlLink
' Создаем объект WshShell
Set WshShell=WScript.CreateObject("WScript.Shell")
' Создаем ярлык на сетевой ресурс
Set oUrlLink = WshShell.CreateShortcut("Microsoft Web Site.URL")
' Устанавливаем URL
oUrlLink.TargetPath = "http://www.microsoft.com"
' Сохраняем ярлык
oUrlLink.Save
'************* Конец ********************************************
```

В-третьих, из сценариев WSH можно обращаться к службам любых приложений-серверов автоматизации, которые регистрируют в операционной системе свои объекты (скажем, программ из пакета Microsoft Office). Например, в сценарии PrintInWord.vbs происходит подключение к серверу автоматизации Microsoft Word и вывод строк текста в окно этого приложения (листинг 1.2).

Листинг 1.2. Использование сервера автоматизации Microsoft Word (VBScript)

```
'*****************************************************************
' Имя: PrintInWord.vbs
' Язык: VBScript
' Описание: Использование из сценария внешнего объекта
'           автоматизации (Microsoft Word)
'*****************************************************************
Option Explicit

Dim WA,WD,Sel  ' Объявляем переменные
```

```
'Создаем объект-приложение Microsoft Word
Set WA=WScript.CreateObject("Word.Application")
' Можно было использовать конструкцию
' Set WA=CreateObject("Word.Application")

Set WD=WA.Documents.Add  'Создаем новый документ (объект Document)
WA.Visible=true  ' Делаем Word видимым
Set Sel=WA.Selection  'Создаем объект Selection
Sel.Font.Size=14 'Устанавливаем размер шрифта
Sel.ParagraphFormat.Alignment=1  'Выравнивание по центру
Sel.Font.Bold=true  'Устанавливаем полужирный шрифт
Sel.TypeText "Привет!" & vbCrLf  'Печатаем строку текста
Sel.Font.Bold=false  'Отменяем полужирный шрифт
Sel.ParagraphFormat.Alignment=0  'Выравнивание по левому краю
'Печатаем строку текста
Sel.TypeText "Эти строки напечатаны с помощью WSH."
WD.PrintOut   'Выводим документ на принтер
'*************  Конец ******************************************
```

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) (объектная модель WMI подробно обсуждается в *приложении 1*).

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 навыках 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 для WSH 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 обо 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.).

Оболочка WMI Command-line (WMIC)

Как уже упоминалось, в операционной системе Windows поддерживается информационная модель Windows Management Instrumentation (WMI), которая занимает важное место среди технологий, инструментов и средств автоматизации. В основе данной технологии лежит схема CIM (Common Information Model), которая представляет физическую и логическую структуры компьютерной системы в виде единой расширяемой объектно-ориентированной информационной модели и определяет единые интерфейсы для получения информации о любом компоненте этой модели.

Изначально работать с WMI можно было либо с помощью специальных графических утилит, либо путем составления довольно сложных сценариев WSH. В состав операционных систем Windows XP и Windows Server 2003 была включена утилита WMIC (WMI Command-line), позволяющая обращаться к подсистеме WMI непосредственно из командной строки. Оболочка WMIC поддерживает навигацию по информационной схеме WMI локального или удаленного компьютера, позволяя выполнять WQL-запросы к классам и объектам WMI. При этом вместо сложных названий классов WMI используются простые псевдонимы, причем можно создавать собственные псевдонимы, что делает информационную схему WMIC расширяемой. Например, классу `Win32_OperatingSystem` соответствует псевдоним `OS`. Если набрать в командной строке WMIC команду `OS` и нажать <Enter>, то мы увидим на экране свойства операционной системы, установленной на компьютере (рис. 1.2).

По умолчанию WMIC поддерживает около 80 псевдонимов, с помощью которых можно выполнить полторы сотни методов и получить значения множества свойств. Важной особенностью WMIC является то, что вывод команд

может быть организован в различные форматы: на экран, в текстовый файл, в XML- и HTML-документы, в MOF-файл, в текстовый файл с разделителями или в любой другой формат, определяемый пользователем с помощью таблиц стилей XSL (eXtensible Stylesheet Language).

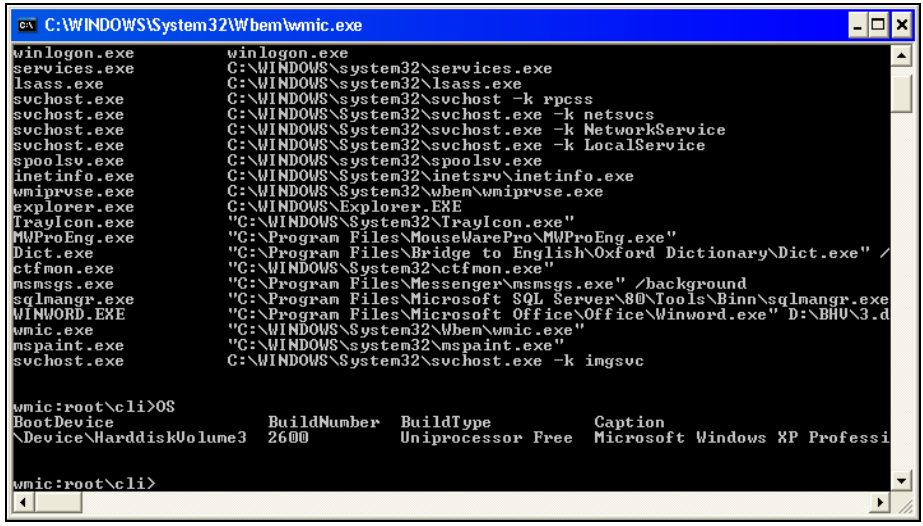

Рис. 1.2. Результат выполнения команды OS в оболочке WMIC

Одна команда WMIC может быть применена сразу к нескольким удаленным компьютерам с любой 32-разрядной версией Windows, при этом наличие WMIC на удаленной машине не требуется, необходима только установка ядра WMI и соответствующая настройка прав доступа к WMI. Кроме этого, команды WMI могут использоваться в пакетных файлах Windows, что позволяет простыми средствами автоматизировать работу с WMI на локальных или удаленных компьютерах.

В качестве недостатка WMIC можно отметить отсутствие встроенной полноценной поддержки и обработки событий WMI. Как показало время, оболочка WMIC оказалась не особенно удачной, так как в этом продукте акцент был сделан на функциональные особенности WMI, а не на удобстве работы пользователя.

Причины и цели создания оболочки PowerShell

Итак, к началу XXI века в операционной системе Windows поддерживались три разных инструмента для автоматизации работы: оболочки командной

строки cmd.exe и WMIC, а также сервер сценариев WSH. Зачем же компании Microsoft понадобилась разработка еще одной совершенно новой оболочки командной строки со своим языком сценариев?

Дело в том, что у каждого из перечисленных инструментов автоматизации имелись довольно серьезные недостатки, не позволявшие сказать, что Windows обладает по-настоящему мощным и эффективным средством для работы с командной строкой и написания сценариев (см. табл. 1.1). 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.

Таблица 1.1. *Требования к инструменту автоматизации*

Требование	**cmd.exe**	**WSH**	**WMIC**
Работа во всех версиях операционной системы без установки дополнительного программного обеспечения	Да	Да	Нет (только Windows XP и выше)
Интеграция с командной строкой	Да	Нет	Да
Согласованный и непротиворечивый синтаксис команд и утилит	Нет	Нет	Да
Поддержка *псевдонимов* (кратких синонимов) для длинных названий команд	Нет	Нет	Да
Автоматическое завершение команд и имен файлов при вводе их с клавиатуры	Частично (автоматическое завершение имен файлов и папок)	Нет	Нет
Поддержка *истории введенных команд* с возможностью их повторного вызова, просмотра и редактирования	Да	Нет	Да
Наличие подробной встроенной справки по командам с примерами использования	Частично	Нет	Да

Таблица 1.1 *(окончание)*

Требование	cmd.exe	WSH	WMIC
Возможность автоматического выполнения сценариев	Да (язык командных файлов)	Да (языки сценариев VBScript, JScript и т. д.)	Частично (команды WMIC можно встраивать в командные файлы)
Доступ и использование всех технологий и возможностей, поддерживаемых операционной системой	Нет (нет прямого доступа к объектам COM, WMI, ADSI, .NET)	Да	Нет (доступ только к объектам WMI)

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.

Новая оболочка *Windows PowerShell* (первоначально она называлась Monad) была задумана разработчиками Microsoft как более мощная среда для написания сценариев и работы из командной строки. Разработчики PowerShell преследовали несколько целей. Главная и наиболее амбициозная из них — создать среду составления сценариев, которая наилучшим образом подходила бы для современных версий операционной системы Windows и была бы более функциональной, расширяемой и простой в использовании, чем какой-либо аналогичный продукт для любой другой операционной системы. В первую очередь эта среда должна была подходить для решения задач, стоящих перед системными администраторами (тем самым Windows получила бы дополнительное преимущество в борьбе за сектор корпоративных платформ), а также удовлетворять требованиям разработчиков программного обеспечения, предоставляя им средства для быстрой реализации интерфейсов управления создаваемыми приложениями.

Для достижения этих целей были решены следующие задачи:

- **Обеспечение прямого доступа из командной строки к объектам COM, WMI и .NET.** В новой оболочке присутствуют команды, позволяющие в интерактивном режиме работать с COM-объектами, а также с экземплярами классов, определенных в информационных схемах WMI и .NET.
- **Организация работы с произвольными источниками данных в командной строке по принципу файловой системы.** Например, навигация по системному реестру или хранилищу цифровых сертификатов выполняется

из командной строки с помощью аналога команды `cd` интерпретатора cmd.exe.

- **Разработка интуитивно понятной унифицированной структуры встроенных команд, основанной на их функциональном назначении.** В новой оболочке имена всех внутренних команд (в PowerShell они называются *командлетами*) соответствуют шаблону "глагол-существительное", например, `Get-Process` (получить информацию о процессе), `Stop-Service` (остановить службу), `Clear-Host` (очистить экран консоли) и т. д. Для одинаковых параметров внутренних команд используются стандартные имена, структура параметров во всех командах идентична, все команды обрабатываются одним синтаксическим анализатором. В результате облегчается изучение и запоминание команд.
- **Обеспечение возможности расширения встроенного набора команд.** Внутренние команды PowerShell могут дополняться командами, создаваемыми пользователем. При этом они полностью интегрируются в оболочку, информация о них может быть получена из стандартной справочной системы PowerShell.
- **Организация поддержки знакомых команд из других оболочек.** В PowerShell на уровне псевдонимов собственных внутренних команд поддерживаются наиболее часто используемые стандартные команды из оболочки cmd.exe и UNIX-оболочек. Например, если пользователь, привыкший работать с UNIX-оболочкой, выполнит `ls`, то он получит ожидаемый результат: список файлов в текущем каталоге (то же самое относится к команде `dir`).
- **Разработка полноценной встроенной справочной системы для внутренних команд.** Для большинства внутренних команд в справочной системе дано подробное описание и примеры использования. В любом случае встроенная справка по любой внутренней команде будет содержать краткое описание всех ее параметров.
- **Реализация автоматического завершения при вводе с клавиатуры имен команд, их параметров, а также имен файлов и папок.** Данная возможность значительно упрощает и ускоряет ввод команд с клавиатуры.

Разработчики старались собрать в PowerShell все лучшие аспекты других оболочек командной строки из разных операционных систем. По их словам, сильное влияние на PowerShell оказали следующие продукты:

- bash, ksh (конвейеризация или композиция команд);
- AS/400, VMS (стандартные названия команд, ускоряющие изучение);
- Tcl, WSH (поддержка встраиваемости и нескольких языков);
- Perl, Python (выразительность и стиль).

Отметим, что PowerShell одновременно является и оболочкой командной строки (пользователь может работать 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.).

Отдельное внимание было уделено вопросам безопасности при работе со сценариями (например, запустить сценарий можно только с указанием полного пути к нему, а по умолчанию запуск сценариев PowerShell в системе вообще запрещен).

Язык PowerShell несложен для изучения, писать на нем сценарии, обращающиеся к внешним объектам, проще, чем на VBScript или JScript. 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.

Главной особенностью среды PowerShell, отличающей ее от всех других оболочек командной строки, является то, что единицей обработки и передачи информации здесь является объект, а не строка текста. В командной строке PowerShell 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.

Отличие PowerShell от других оболочек — ориентация на объекты

При разработке любого языка программирования одним из основных является вопрос о том, какие типы данных и каким образом будут в нем представлены. При создании PowerShell разработчики решили не изобретать ничего нового и воспользоваться унифицированной объектной моделью .NET. Данный выбор был сделан по нескольким причинам.

Во-первых, *платформа .NET* повсеместно используется при разработке программного обеспечения для Windows и предоставляет, в частности, общую информационную схему, с помощью которой разные компоненты операционной системы могут обмениваться данными друг с другом.

Во-вторых, объектная модель .NET является *самодокументируемой*: каждый объект .NET содержит информацию о своей структуре. При интерактивной работе это очень полезно, так как появляется возможность непосредственно из командной строки выполнить запрос к определенному объекту и увидеть описание его свойств и методов, то есть понять, какие именно манипуляции можно проделать с данным объектом, не изучая дополнительной документации с его описанием.

В-третьих, работая в оболочке с объектами, можно с помощью их свойств и методов легко получать нужные данные, не занимаясь разбором и анализом символьной информации, как это происходит во всех традиционных оболочках командной строки, ориентированных на текст. Рассмотрим пример. В Windows XP есть консольная утилита tasklist.exe, которая выдает информацию о процессах, запущенных в системе:

```
C:\> tasklist

Имя образа                   PID Имя сессии       № сеанса       Память
========================= ====== ================ ======== ============
System Idle Process            0                         0         16 КБ
System                         4                         0         32 КБ
smss.exe                     560                         0         68 КБ
csrss.exe                    628                         0      4 336 КБ
winlogon.exe                 652                         0      3 780 КБ
services.exe                 696                         0      1 380 КБ
lsass.exe                    708                         0      1 696 КБ
svchost.exe                  876                         0      1 164 КБ
svchost.exe                  944                         0      1 260 КБ
svchost.exe                 1040                         0     10 144 КБ
svchost.exe                 1076                         0        744 КБ
svchost.exe                 1204                         0        800 КБ
spoolsv.exe                 1296                         0      1 996 КБ
kavsvc.exe                  1516                         0      9 952 КБ
klnagent.exe                1660                         0      5 304 КБ
klswd.exe                   1684                         0         64 КБ
```