

Windows PowerShell



*Введение в технологии
языка сценариев для
пользователей без
базовых знаний*

MICROSOFT SWITZERLAND

1 октября 2007

Франк Кох (БЕРН)
Разработчик и пропагандист платформы

Windows PowerShell

Чего следует ожидать от этой короткой работы – Попытка предисловия

Зачем написана эта книга

Эта книга, «Windows PowerShell», является введением в Windows PowerShell, а также содержит практические примеры, позволяющие дать краткое введение в тему, даже если у читателя нет серьезного опыта в написании сценариев. Эта книга совершенно точно не предназначена для профессиональных программистов сценариев – существует обширная справка по Windows PowerShell, множество форумов в Интернете и дополнительная литература, которые обеспечат экспертов всей необходимой информацией. Однако новичок сможет найти в этой книге все, что нужно, чтобы подробнее узнать о сценариях, и будем надеяться, научиться получать удовольствие от работы с компьютером даже без мыши.

Источниками для этой книги в основном послужили публикации Microsoft по Windows PowerShell. Информация в книге представлена по-новому, удобно для целевой аудитории. Мы не стали излагать в начале книги теорию, ее место занимают различные примеры и небольшие практические задания, которые будут надежно удерживать ваше внимание.

Раз вы прочли этот текст и решили сделать Windows PowerShell постоянной частью вашей обычной работы с компьютером, не стесняйтесь читать оригинальную документацию по Windows PowerShell, которая будет автоматически установлена при установке Windows PowerShell:

- Первые шаги с Windows PowerShell
- Базовые принципы Windows PowerShell

Чтобы получить реальную пользу от этой книги, вы должны иметь доступ к ПК, на котором будете делать упражнения во время чтения. Единственное предварительное условие состоит в том, что это должен быть ПК с установленным Windows PowerShell 1.0, который поставляется бесплатно в составе Windows XP SP2. Сведения о загрузке и установке PowerShell см. на указанных ниже веб-сайтах.

Другие источники информации в сети Интернет

Вводная страница Windows PowerShell, содержащая ссылку на загрузку: www.microsoft.com/PowerShell. Здесь вы также сможете найти другие ссылки на очень полезные Интернет-трансляции, книги и другие форумы поддержки.

Полезные блоги о Windows PowerShell можно найти по адресу <http://blogs.msdn.com/PowerShell/>. Здесь можно прочитать о методах программирования сценариев и найти практические примеры. Абсолютно любые.

В Швейцарии вы можете также найти сведения на немецком языке в блоке команды ITPro по адресу <http://blogs.technet.com/chITPro-DE>. В нем можно найти ссылки на немецкие Интернет-трансляции о Windows PowerShell и загрузить примеры к книге из архива за март/апрель 2007 г..

Много полезной информации о Windows PowerShell на русском языке можно найти в блогах Андрея Бешкова <http://blogs.technet.com/abeshkov/default.aspx>, Василия Гусева <http://xaegr.wordpress.com> и Дмитрия Сотникова <http://www.itcommunity.ru/blogs/dmitrysotnikov/default.aspx>. Переводчик этой книги тоже иногда пишет о PowerShell в своем блоге <http://pwrshell.blogspot.com/>.

Содержание

| | |
|---|----|
| Зачем написана эта книга | 2 |
| Другие источники информации в сети Интернет | 2 |
| Первое впечатление о Windows PowerShell | 5 |
| Расширенные возможности вывода: конвейеры | 7 |
| Вводные упражнения с объектами Windows PowerShell..... | 8 |
| Работа с процессами | 8 |
| Вывод в файлы формата TXT, CSV или XML..... | 9 |
| Вывод в цвете | 10 |
| Проверка условий с помощью командлета if | 11 |
| Вывод в виде HTML | 12 |
| Работа с файлами | 15 |
| Поиск информации об объектах с помощью Get-Member..... | 16 |
| Удаление файлов | 18 |
| Создание папок..... | 19 |
| Если у вас есть время | 21 |
| Windows PowerShell как машина обработки произвольных объектов | 23 |
| Объекты WMI..... | 23 |
| Работа с объектами .NET и XML..... | 25 |
| Работа с COM-объектами | 26 |
| Работа с журналами сообщений..... | 29 |
| Сценарии-решения к упражнениям в этой книге | 30 |
| Примеры к Windows PowerShell – от простых к сложным | 33 |
| Теоретические принципы Windows PowerShell | 35 |
| Windows PowerShell – краткое введение..... | 35 |
| Цели разработки Windows PowerShell..... | 35 |
| О тексте, разборе текста и объектах..... | 35 |
| Новый язык сценариев | 36 |
| Команды Windows и служебные программы..... | 37 |
| Интерактивная среда..... | 37 |
| Поддержка сценариев | 37 |
| CMD, WScript или PowerShell? Что выбрать? | 37 |
| Windows PowerShell 1.0 | 38 |
| Безопасность при использовании сценариев | 39 |

Windows PowerShell в работе

Windows PowerShell является свободно распространяемым приложением к семейству операционных систем Windows XP и выше, его можно загрузить с веб-сайта Microsoft по адресу <http://www.microsoft.com/powershell>. Необходимым предварительным условием является наличие среды .NET Framework 2.0, если она еще не установлена, ее следует загрузить и установить отдельно. Сам по себе пакет Windows PowerShell имеет относительно малый объем, около 1,5 МБайт, и легко устанавливается автоматически через стандартные каналы распространения программного обеспечения. Следует лишь учитывать, что для каждой версии и архитектуры Windows используется своя версия Windows PowerShell.

После установки Windows PowerShell помещает себя в меню Start, доступ к ней можно получить, щелкнув по пиктограмме быстрого доступа или введя «PowerShell» в окно для запуска команд Windows.

Первое впечатление о Windows PowerShell

Чтобы получить первое впечатление, запустите одновременно Windows PowerShell и классическую командную строку CMD из меню Start. На первый взгляд обе командные оболочки выглядят очень похоже – за исключением разного цвета:



Рисунок 1: Классическая командная строка CMD



Рисунок 2: Windows PowerShell

Ничего неожиданного в этом нет, поскольку обе командные оболочки используют один и тот же механизм командной строки. К сожалению, это также означает, что Windows PowerShell страдает такой же плохой реализацией операций копирования/вставки, как и CMD. Поэтому будут небесполезны следующие советы:

- Выберите нужный текст с помощью мыши
- Нажмите правую кнопку мыши (= копирование)
- Поместите курсор в нужное место
- Нажмите правую кнопку мыши (= вставка)

Попробуйте сделать это сами. Скопируйте первую строку текста в каждой из оболочек (то есть «Copyright (c) 2006 Microsoft Corporation») и вставьте ее в качестве командной строки. Не беспокойтесь о сообщениях об ошибках, которые появятся после того, как вы нажмете Enter. Привыкайте к этим упражнениям с кнопками мыши, позднее вам придется часто их использовать.

Несмотря на то, что внешне эти оболочки командной строки одинаковы, содержимое и функции каждой из них сильно различаются. Самый простой способ понять это – взглянуть на интерактивную справку. Первый же взгляд ясно покажет, что Windows PowerShell предоставляет значительно больше функций, чем CMD; более 100 команд, также называемых командлетами (пишется «cmdlets»). Сама CMD содержит всего несколько команд, поэтому для нее было разработано множество вспомогательных программ. Поскольку каждая из этих программ CMD имеет собственный синтаксис, опытные эксперты по CMD наизусть знают множество разных команд и их логику. Для командлетов синтаксис и логика единообразны.

Команды Windows PowerShell следуют определенным правилам именования:

- Команды Windows PowerShell состоят из глагола и существительного (всегда в единственном числе), разделенных тире. Команды записываются на английском языке. Пример:
`Get-HeIp` вызывает интерактивную справку по синтаксису Windows PowerShell
- Перед параметрами ставится символ «-»:
`Get-HeIp -Detailed`
- В Windows PowerShell также включены псевдонимы многих известных команд. Это упростит вам знакомство и использование Windows PowerShell. Пример:
Команды `heIp` (классический стиль Windows) и `man` (классический стиль Unix) работают так же, как и `Get-HeIp`.

Выведите на экран различные тексты справки для каждой командной оболочки. Введите команды `heIp` в каждой из оболочек. Вы увидите, что в каждой справке приводится разное количество документированных команд.

Вместо `heIp` или `man` в Windows PowerShell можно также использовать команду `Get-HeIp`. Ее синтаксис описан ниже:

- `Get-HeIp` выводит на экран справку об использовании справки
- `Get-HeIp *` перечисляет все команды Windows PowerShell
- `Get-HeIp` команда выводит справку по соответствующей команде
- `Get-HeIp` команда `-Detailed` выводит подробную справку с примерами команды

Использование команды `heIp` для получения подробных сведений о команде `help`:
`Get-HeIp Get-HeIp -Detailed`.

Подсказка: Используйте клавишу TAB для автоматического завершения ввода команды. Это поможет вам избежать опечаток.

Расширенные возможности вывода: конвейеры

Как было сказано выше, Windows PowerShell является объектно-ориентированной командной оболочкой. Это означает, что вводимые и выводимые данные команд как правило являются объектами. Поскольку человек не может читать объекты, Windows PowerShell «транслирует» объекты для вывода на экран в текст (профессионалы могут даже найти в Windows PowerShell команды, которые позволят настроить вывод в соответствии с их нуждами). На связывание команд указывает команда конвейера: |

Эту связку можно использовать также для создания собственной книги Windows PowerShell. `Get-Helper * | Get-Helper -Detailed` сделает это для вас: команда `Get-Helper *` создаст список известных команд, который будет подан на вход команды `Get-Helper -Detailed`. Она выведет очень много информации, вывод можно будет прервать комбинацией клавиш CTRL-C.

Чтобы иметь возможность использовать результат «справочника» позднее, было бы разумно перенаправить выход в файл, а не выводить данные на экран. Windows PowerShell имеет для этого специальную команду `Out-File`, более известную в варианте символа «>».

Теперь создайте собственный «файл книги». Введите соответствующую команду: `Get-Helper * | Get-Helper -Detailed | Out-File c:\Powershell-Help.txt` или даже `Get-Helper * | Get-Helper -Detailed > C:\PowerShell-Help.txt`. Учтите, что вы должны иметь права на запись в соответствующий каталог (в данном случае `C:\`).

Откройте свой первый файл справки в Блокноте и используйте его как интерактивную справку в последующей работе. Если вы когда-либо будете искать команду, `Get-Helper` поможет вам и в этом. Если вы хотите что-то отсортировать, попробуйте найти что-то подходящее, используя команду `Get-Helper Sort*`. `Get-Helper` будет искать соответствующую команду в репозитории команд Windows PowerShell. Поскольку все команды начинаются с глагола, мы легко можем структурировать поиск, используя форму `Get-Helper` соответствующий английский глагол `*`. Если вы еще не знаете, символ «*» означает поиск по шаблону. Он используется, если после текста для поиска может идти все, что угодно, а мы хотим найти все, что начинается с нашего текста для поиска.

После того, как вы найдете команду (например, пусть это будет `Sort-Object`), просто вызовите еще раз `Get-Helper`, теперь с соответствующей командой и параметром `-Detailed`, чтобы найти примеры использования этой команды:

```
Get-Helper Sort-Object -Detailed.
```

Теперь вы должны быть в состоянии решить свою проблему.

Вводные упражнения с объектами Windows PowerShell

Если вы никогда раньше не работали с объектами, примеры ниже позволят вам понять многочисленные возможности этого мира. Объекты не являются чем-то новым в программировании, но в области сценариев ничего подобного еще не было. Если вы заинтересовались работой с объектами, подробную дополнительную литературу вы сможете найти в MSDN по адресам <http://msdn.microsoft.com> и <http://www.microsoft.com/switzerland/msdn/ru/default.mspx>. Давайте рассмотрим объекты на примере объекта процесса «process». Если слово «процесс» для вас ничего не значит, подумайте о том, что вы видите на экране, когда вызываете Диспетчер задач (Task Manager). Если вы заинтересовались объектом «process», на страницах MSDN вы найдете информацию о нем.

Работа с процессами

Команда `Get-Process` выводит список всех процессов в вашей системе. Этот список может быть очень длинным. Для сортировки списка вы можете воспользоваться другим командлетом: `Sort-Object`. По умолчанию `Sort-Object` имеет фиксированный порядок сортировки (по возрастанию), который можно изменить с помощью параметра `-Descending`. В качестве аргумента можно указать свойство объекта, например, используемое время процессора (CPU).

A1: Сейчас ваша задача состоит в создании списка всех процессов и его сортировке в порядке убывания в соответствии с используемым временем процессора. Вы уже знаете, как это сделать: `Get-Process`, `Sort-Object` и конвейер (`|`).

Подсказка: CPU не является параметром `Sort-Object`, это аргумент, который вы можете использовать при сортировке. Поэтому он не имеет символа «-».

В следующем упражнении мы хотим немного ограничить список, чтобы с ним было проще работать. Мы используем команду `Select-Object`. `Select-Object` понимает несколько параметров (чтобы их узнать, воспользуйтесь командой `Get-Help`), но нам понадобятся только `-First x` и `-Last y`, с помощью которых можно узнать первые `x` или последние `y` объектов списка, например `Select-Object -First 5`. Командлет `Select-Object` не работает сам по себе, он ожидает исходных данных, передаваемых через конвейер.

A2: Создадим список первых 10 процессов по используемому времени процессора. Для этого возьмем результаты упражнения A1 и добавим к ним команду `Select-Object`. Существует два пути получить идеальное решение, в зависимости от того, как вы хотите отсортировать список. Давайте рассмотрим оба.

Подсказка: в одном из путей используется параметр `-First`, в другом `-Last`.

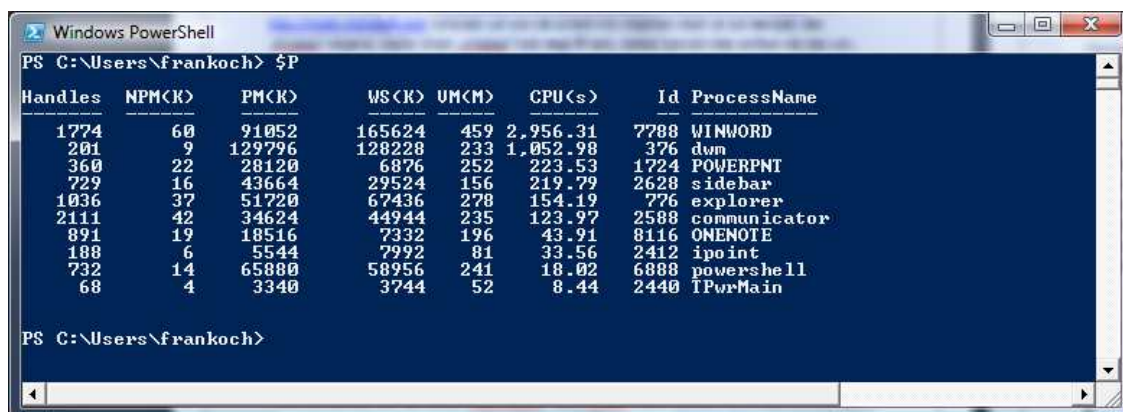
Мы используем это упражнение в качестве краткого введения в переменные. Говоря простым языком, в переменных хранятся все возможные значения, даже если они являются объектами. Здесь мы тоже сошлемся на дополнительную литературу, которую следует использовать для более глубокого изучения переменных. Сейчас нам нужно только знать, что переменные в PowerShell всегда должны начинаться с символа «\$». Вы можете сохранить результат упражнения A2 в переменной, это позволит вам в любое время получать доступ к списку из 10 процессов. Получив текущий список и сравнив его со значением переменной, можно оценить изменения в системе. Присвоить значение переменной можно легко:

```
$a = get-process | sort-object CPU -de...
```


Многоточие в конце строки указывает, что выражение не дописано до конца. Если попытаться выполнить его в таком виде, PowerShell сообщит об ошибке. Закончите его самостоятельно.

A3: Назначьте переменной \$P сокращенный список процессов из упражнения A2.

Подсказка: С помощью клавиши курсора «Стрелка вверх» можно вызвать последнюю использованную команду, а с помощью клавиши «Home» переместить курсор в начало строки, а затем ввести данные. Вывести содержимое переменной можно, просто напечатав в командной строке \$P.



```
PS C:\Users\frankoch> $P
```

| Handles | NPM(K) | PM(K) | WS(K) | UM(M) | CPU(s) | Id | ProcessName |
|---------|--------|--------|--------|-------|----------|------|--------------|
| 1774 | 60 | 91052 | 165624 | 459 | 2.956.31 | 7788 | WINWORD |
| 201 | 9 | 129796 | 128228 | 233 | 1.052.98 | 376 | dwm |
| 360 | 22 | 28120 | 6876 | 252 | 223.53 | 1724 | POWERPNT |
| 729 | 16 | 43664 | 29524 | 156 | 219.79 | 2628 | sidebar |
| 1036 | 37 | 51720 | 67436 | 278 | 154.19 | 776 | explorer |
| 2111 | 42 | 34624 | 44944 | 235 | 123.97 | 2588 | communicator |
| 891 | 19 | 18516 | 7332 | 196 | 43.91 | 8116 | ONENOTE |
| 188 | 6 | 5544 | 7992 | 81 | 33.56 | 2412 | ipoint |
| 732 | 14 | 65880 | 58956 | 241 | 18.02 | 6888 | powershell |
| 68 | 4 | 3340 | 3744 | 52 | 8.44 | 2440 | TPwrMain |

```
PS C:\Users\frankoch>
```

Рисунок 3: Содержимое переменной \$P

Вывод в файлы формата TXT, CSV или XML

По умолчанию Windows PowerShell выводит результаты работы цепочки команд на экран. Все объекты преобразуются в текст, чтобы человек мог прочитать содержащиеся в них данные. Для этого используется команда `Out-Host`. Однако поскольку Windows PowerShell построена с расчетом на максимальную эффективность, эта команда добавляется автоматически и будет невидима, если вы не добавите ее явно. Существуют альтернативы команды `Out-Host`; их можно найти с помощью команды `Get-Helper Out*`.

Вывести результаты в текстовый файл можно легко и быстро: `Out-File имяфайла`. Многие командные оболочки используют команду «>», которая поддерживается и в Windows PowerShell. Выводимые данные можно преобразовать не только в текстовый файл, но и в CSV или XML. После работы команды `Out-Host` специальные командлеты выполнят для вас и эту задачу. Эти командлеты имеют имена `Export-Csv` и `Export-CliXML`, оба они требуют в качестве аргумента имя файла. И да, конечно – если вы можете экспортировать, вы можете и импортировать. Для импорта файлов с целью просмотра используются команды `Import-Csv` и `Import-CliXML`.

A4: Возьмите переменную \$P из упражнения A3 и сохраните ее содержимое в текстовый файл с именем «A4.txt». Затем сохраните содержимое \$P в файл CSV с именем «A4.CSV», и наконец в файл XML с именем «A4.XML».

Подсказка: при использовании > не нужен символ |, который требуется только для командлетов, таких как `Out-File`, `Export-Csv` и т.д. Просмотрите результат, для этого можно воспользоваться Блокнотом (Notepad).

Вывод в цвете

Иногда бывает необходимо выделить результаты, сделав их удобнее для чтения. Это можно сделать, например, с помощью выделения цветом. Команда `write-host` распознает некоторые параметры, такие как `-ForegroundColor` и `-BackgroundColor`. Как вы думаете, какой результат получится после выполнения этой команды?

```
write-host "Red on blue" -ForegroundColor red -BackgroundColor blue
```

Вы можете угадать. `Get-help write-host -Detailed` даст вам полный список возможных цветов. Также существуют определенные заранее сочетания: вы можете также привлечь внимание пользователя с помощью команды `write-warning "error"`. Попробуйте сделать это прямо сейчас. С помощью этой команды вы можете окрасить все данные, выводимые процессами. Однако будет лучше, если вы раскрасите список в соответствии с дополнительными условиями. Давайте подробнее рассмотрим эту возможность.

Для простоты мы используем службы ПК, а не процессы. Если вы не знаете, что такое службы, посмотрите сведения о них, например, в MSDN. Просто скажем, что службы – это то, что показывается в окне Control Panel / Administrative Tools / Services. Что нам важно знать о службах, это то, что они имеют статус «запущена» или «остановлена» («started» или «stopped»), которое можно использовать для окрашивания выводимых данных. Но для начала посмотрим службы с помощью командлета `Get-Service`.

A5: Создайте список всех служб и отсортируйте их по статусу.

Подсказка: Используйте тот же метод, что и для сортировки процессов по используемому времени процессора, но применяйте команду `get-service` и «status» в качестве аргумента командлета `Sort-Object`.

Мы хотим вывести весь список красным цветом. В этом нам поможет командлет `write-host`. К сожалению, команда `Get-Service | write-host -ForegroundColor red` не будет работать так, как мы ожидаем. `write-host` не дружит с другими командлетами и не принимает список объектов, который следует затем вывести нужным цветом. Команда `write-host` должна знать, какой атрибут каждого объекта следует выводить таким образом. Мы можем немного помочь ей с этим. Будем работать со списком объектов последовательно, используя цикл. Существует множество циклов, каждый из которых имеет свои особенности и области использования. Для наших нужд мы воспользуемся циклом `ForEach-Object`, который проходит по списку объектов и передает каждый объект по отдельности следующему за ним командлету. В цикле мы выбираем нужный объект с помощью сокращения `$_`, особого «служебного имени» PowerShell. Затем мы выбираем атрибут объекта, вот так: `$_.имя-свойства`. Давайте рассмотрим пример:

```
Get-Process | ForEach-Object { write-host $_.ProcessName $_.CPU }
```

Хотя `Get-Process` дает нам список процессов, в этом примере мы можем просмотреть только имя и используемое время процессора, так как сначала выводимые данные уходят в канал (`()`), а затем `write-host` извлекает из него только два атрибута каждого объекта. Если вы вызовете пример еще раз, не удивляйтесь результату: не все процессы используют время процессора, так что в определенных условиях строка может содержать только имя.

A6: Создайте список служб и выведите на экран только атрибуты имени и статуса. Воспользуйтесь ранее описанным циклом `ForEach`, но можете подумать и о других возможных решениях.

Теперь мы можем использовать параметры `Write-Host` для вывода в цвете.

A7: Создайте список служб и выведите на экран только атрибуты имени и статуса, используя цвета по желанию. Покажите цвета своему соседу и решите, у кого получилось самое красивое сочетание.

Подсказка: Воспользуйтесь решением из A6 и добавьте параметры `-ForegroundColor` и `-BackgroundColor`.

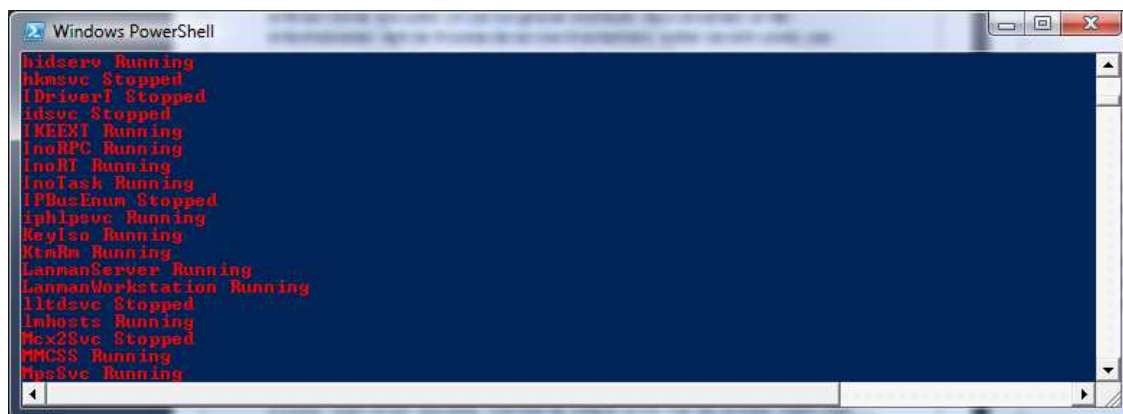


Рисунок 4: Вывод имен и статусов служб красным цветом

Проверка условий с помощью командлета `if`

Мы пропустили еще одну вещь – проверку условий. Для этого существует множество вариантов, подходящих для различных сценариев. Однако сейчас, в этом введении, мы ограничимся командой `If`. Вы, возможно, уже знаете ее синтаксис из других вариантов использования, он довольно прост:

```
If (условие) {выполняемые команды}
```

```
elseif (условие2) {выполняемые команды}
```

```
else {выполняемые команды}
```

`elseif` – необязательная часть, необходимости в ней нет. Если вы хотите выполнить в области `{}` более одной команды, их можно разделять точкой с запятой или использовать для каждой команды новую строку. Windows PowerShell будет ожидать завершающей `}` в конце.

Для сравнения Windows PowerShell использует несколько операторов сравнения. Они все начинаются с «-» и обычно содержат аббревиатуру из двух латинских букв: `-eq` означает «equals», «равно». Наиболее важные операторы перечислены ниже:

| | |
|------------------------|--------------------------------------|
| <code>-Eq</code> | Равно |
| <code>-Match</code> | Сравнение по регулярному выражению |
| <code>-Ne</code> | Не равно |
| <code>-Notmatch</code> | Не совпадает с регулярным выражением |
| <code>-Gt -Ge</code> | Больше / Больше или равно |
| <code>-Lt -Le</code> | Меньше / Меньше или равно |

Последнее упражнение в этом блоке содержит одновременно все варианты отслеживания статуса системы с использованием PowerShell. Все службы в системе

сначала сортируются по статусу, а затем выводятся с выделением цветом: службы со статусом «stopped» красным, службы со статусом «running» зеленым.

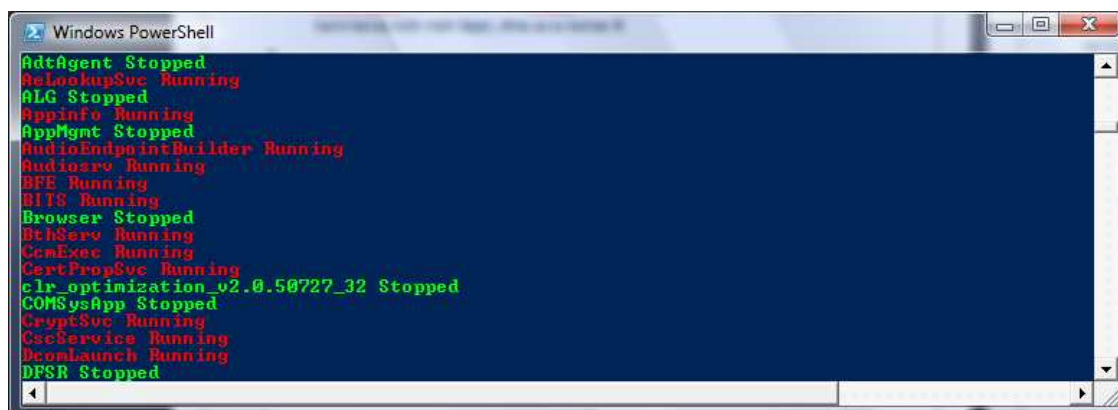
A8: Вызовите список служб. Отсортируйте список по статусу и окрасьте выводимые данные в красный или зеленый цвет в зависимости от статуса службы «stopped» или «running».

Подсказка: Сначала используйте Sort-Object, как в предыдущих упражнениях. Затем воспользуйтесь циклом Foreach, но вместо того, чтобы просто использовать Write-Host, добавьте запрос If. Вы можете просмотреть статусы служб, используя как обычно \$_.status; возможные значения «stopped» или «running».

О синтаксисе: Оператор If помещается в скобки (), а команда вывода в скобки {}. Игнорируйте теоретические параметры и для простоты не пользуйтесь запросом ElseIf. Не забудьте конечную фигурную скобку в Foreach! Когда вы дойдете до конца строки >>, закройте ее, дважды нажав Return для выполнения введенных строк.

Еще раз выполните выражение, но на этот раз без командлета Sort-Object. Чтобы не вводить все данные второй раз, воспользуйтесь клавишами курсора.

Командлет Foreach-Object можно сократить до Foreach. Его можно сделать еще короче, но тогда вы не сможете прочитать его, не зная смысла значка. Поэтому здесь мы будем использовать варианты Foreach-Object или Foreach.



```
Windows PowerShell
AddAgent Stopped
BeLookupSvc Running
ALG Stopped
Appinfo Running
AppMgmt Stopped
AudioEndpointBuilder Running
Audiosrv Running
BFE Running
BITS Running
Browser Stopped
BthServ Running
CcmExec Running
CertPropSvc Running
clr_optimization_v2.0.50727_32 Stopped
COMSysApp Stopped
CryptSvc Running
CseService Running
DcomLaunch Running
DFSR Stopped
```

Рисунок 5: Цветные сведения о службах

Вывод в виде HTML

Пример A8 можно использовать для мониторинга серверов. Теперь было бы полезно упростить повторное использование выводимых данных. Вы уже знаете, как выводить данные в виде CSV и XML. Однако существует и еще одна возможность, иногда более полезная: HTML. Для этого используется командлет Convertto-HTML. При этом данные выводятся не в виде файла, как в случае других командлетов, а в форме, позволяющей осуществлять редактирование непосредственно в канале. В конце можно перенести текст в файл, чтобы, например, вам было легче просматривать его в веб-браузере. С помощью серии мини-примеров мы покажем различные возможности, доступные с помощью Convertto-HTML.

A9: Преобразуем выходные данные Get-Service в HTML. Используем командлет Convertto-HTML, который может работать непосредственно со списком объектов.

Подсказка: Если список слишком длинный, его можно оборвать, нажав CTRL-C.

A10: В конце воспользуемся командами, которые, как мы знаем, помещают выходные данные в файл «.\A10.html». Посмотрим этот файл.

Подсказка: Можно использовать команду `Invoke-Item .\a10.html` для запуска веб-браузера по умолчанию и вывода в него файла прямо из PowerShell. Не забудьте правильно указать путь к A10.html. Если хотите, можете открыть файл с помощью Проводника.

`Convertto-Html` позволяет ограничить выводимые данные, чтобы список не стал нечитаемым. На вход `Convertto-Html` следует подавать список выводимых объектов, т.е. `... | Convertto-Html -Property name, status`.

A11: Продолжение A10: Создайте более привлекательную веб-страницу и список с именами и статусами всех служб. Можно также перед преобразованием отсортировать выводимые данные по статусу.

Подсказка: Ваша командная строка теперь будет состоять из 4 команд: вывести список всех служб, отсортировать их по статусу, преобразовать в HTML, вывести как файл. Поскольку `Convertto-Html` создает текст HTML, результат можно легко модифицировать, если у вас есть опыт работы с HTML. Это не задача Windows PowerShell, но PowerShell может вам помочь. Попробуйте понять, что делает этот код, прежде чем скопировать, вставить и выполнить его в Windows PowerShell:

```
Get-Service | Convertto-Html -Property name,status | Foreach {  
  If ($_ -Like "*<td>Running</td>*") {$_ -Replace "<tr>", "<tr  
bgcolor=green>"}  
  else {$_ -Replace "<tr>", "<tr bgcolor=red>"} } > .\get-service.html
```

Выходной файл должен выглядеть похоже на то, что приведено ниже. В принципе, этот пример работает так же, как `write-host`, но здесь отдельные строки файла HTML переформированы: команда HTML для столбца таблицы задает зеленый или красный фоновый цвет `bgcolor green` или `bgcolor red`. Поскольку не все знают HTML, этот пример приведен, как исключение, с полным кодом решения.

HTML TABLE - Windows Internet Explorer

C:\Users\frankoch\ Live Search

web durchsl

HTML TABLE

| Name | Status |
|--------------------------------|---------|
| AdtAgent | Stopped |
| AeLookupSvc | Running |
| ALG | Stopped |
| Appinfo | Running |
| AppMgmt | Stopped |
| AudioEndpointBuilder | Running |
| Audiosrv | Running |
| BFE | Running |
| BITS | Running |
| Browser | Stopped |
| BthServ | Running |
| CcmExec | Running |
| CertPropSvc | Running |
| clr_optimization_v2.0.50727_32 | Stopped |
| COMSysApp | Stopped |

Computer | Protected Mode: Off 100%

Рисунок 6: Цветной вывод HTML с использованием преобразования HTML

Работа с файлами

В следующих нескольких упражнениях мы будем работать с файлами. Если эта книга используется как часть курса, попросите тестовые файлы у преподавателя. Если вы работаете с упражнениями самостоятельно, просто создайте отдельную папку для упражнений. Для этого скопируйте в папку несколько разных файлов (например, 40). Если вы не можете их найти, воспользуйтесь файлами из своего Интернет-кэша. Вы должны убедиться, что используете файлы как минимум двух разных типов, но их может быть и больше.

Работа с файлами в Windows PowerShell по-настоящему проста. Можно использовать псевдонимы популярных команд, таких как `dir` или `ls`. Для команды `cd` следует учитывать, что между командой и путем должен стоять пробел: «`cd ..`», а не «`cd..`»!

Windows PowerShell превращает все файлы в объекты. Размер файла можно прямо запросить, его не придется выделять из строки. Кроме того, Windows PowerShell может работать не только в классической файловой системе. С помощью командлета `Get-PSdrive` вы можете вывести все диски, к которым Windows PowerShell позволяет осуществлять доступ. Диски выделяются двоеточием после их имени (:).

Выведите список всех дисков Windows PowerShell. Переключитесь (`cd`) на диск **HKLM:**. Введите команду `cd software`. Введите команду `dir`. Где вы теперь находитесь? Переключитесь на диск **ENV:**. Выведите его содержимое командой `ls`, как если бы это была обычная папка Unix. В конце переключитесь на диск **CERT:** и выведите список содержимого, используя командлет `Get-Childitem`.

Вы увидите, что от Windows PowerShell невозможно скрыться почти нигде на вашем компьютере. Команды `dir`, `ls` и `Get-Childitem` всюду имеют одинаковые возможности. Это означает, что вы можете использовать те псевдонимы, которые вам больше нравятся. Чтобы оставаться в рамках синтаксиса Windows PowerShell, я буду говорить в основном о `Get-Childitem`. Еще одно замечание о реестре. Если вы когда-нибудь разбирались с реестром, то отметили, что по нему можно перемещаться при помощи команд `dir` и `cd`, но они не позволяют просмотреть значения ключей реестра. Причина в том, что значения реестра являются свойствами объектов реестра, в то время как размер (`Size`) или дата последнего изменения (`Date Last accessed`) – это свойства файла. Чтобы увидеть значения реестра, необходима команда `Get-ItemProperty`. Эта команда выводит все свойства элемента реестра и их значения. Дополнительные сведения можно найти, как и ранее, в справке по Windows PowerShell.

Чтобы упростить работу с тестовыми файлами, мы создадим в Windows PowerShell новый диск, который будет указывать на реальную тестовую папку. Для этого необходимо выполнить команду `New-PSdrive`.

Создайте новое устройство, введя команду. Измените конец пути так, чтобы он указывал на вашу папку с тестовыми файлами:

```
New-PSdrive -Name FK -PSprovider FileSystem -Root c:\путькпапке
```

Затем воспользуйтесь командой `cd FK:`, чтобы перейти в эту папку и проверьте, в нужной ли папке вы находитесь. Если это не так, используйте команду `Remove-PSDrive FK` для удаления устройства и попробуйте еще раз.

Следующие упражнения продемонстрируют вам обширные возможности Windows PowerShell.

Переключитесь на свой учебный диск: `cd fk:` (В синтаксисе PowerShell, кстати, это будет выглядеть так: `Set-Location fk:`). Выведите список содержимого с помощью команды `Get-Childitem`. Скройте все временные файлы: `Get-Childitem * -Exclude *.tmp, *.temp`

V1: Выведите только имена и длину файлов, игнорируя временные файлы с расширениями `temp` или `tmp`.

Подсказка: Используйте те же методы, что и для вывода процессов и служб.

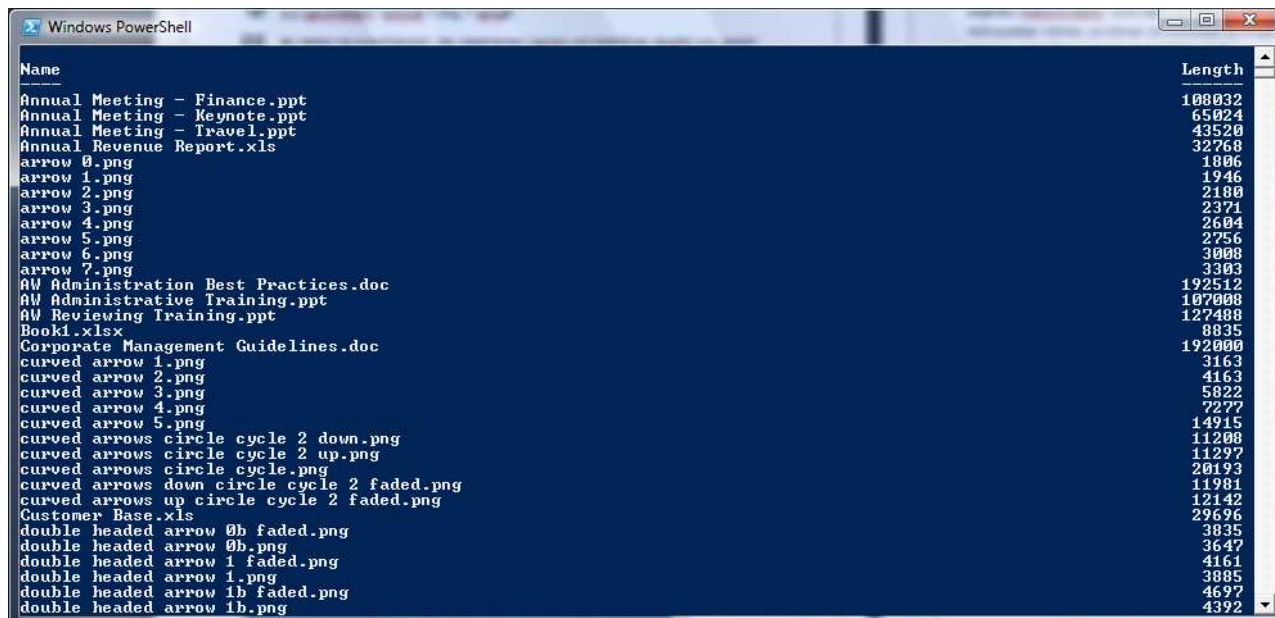


Рисунок 7: Вывод имен и длины файлов без файлов TMP

Чтобы свести к минимуму количество вводимой информации, Windows PowerShell предлагает различные способы сокращения команд. Введите `Get-Alias | Sort-Object Definition` и увидите список всех возможных псевдонимов команд. Однако для автоматического дополнения параметров необходимо точно ввести псевдоним, не ошибаясь в количестве букв, в том числе не добавляя лишних букв. Итак, `Get-Childitem * -Exclude *.tmp | Select-Object name, length` превратится в `ls * -ex *.tmp | select n*, le*`

V2: Отсортируйте файлы в порядке возрастания по размеру (длине), затем по имени.

Подсказка: Используйте те же методы, что и для вывода процессов в предыдущих примерах.

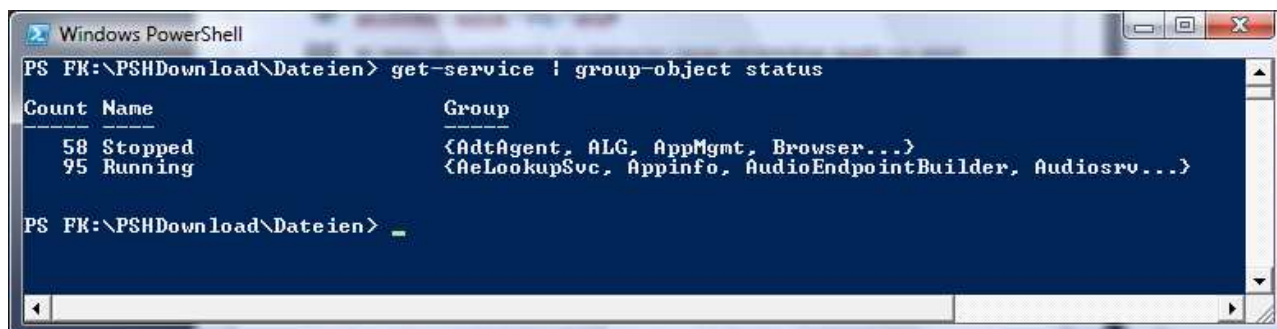
Поиск информации об объектах с помощью `Get-Member`

Используйте командлет `Get-Member`, чтобы просмотреть все атрибуты и функции объекта. Для использования этой команды следует передать объект `Get-Member` через конвейер. Вы можете даже передать список одинаковых объектов `Get-Member`, который сможет правильно с ним разобраться.

V3: Создайте список всех возможных атрибутов файла с помощью командлета `Get-Member`. Отсортируйте все файлы по дате последнего доступа.

Подсказка: Используйте результат работы функции `Get-Member` и «угадайте» нужные атрибуты из списка свойств.

Команда Group-Object может разделить список объектов на группы. Для этого необходимо использовать в качестве аргумента один из атрибутов объекта. Get-Service | Group-Object status затем создаст новый список, содержащий две (или более) записей. Удобно, что будет также показано число служб и их статус:



```
PS FK:\PSHDownload\Dateien> get-service | group-object status
```

| Count | Name | Group |
|-------|---------|---|
| 58 | Stopped | {AdtAgent, ALG, AppMgmt, Browser...} |
| 95 | Running | {AeLookupSvc, Appinfo, AudioEndpointBuilder, Audiosrv...} |

```
PS FK:\PSHDownload\Dateien> _
```

Рисунок 8: Результаты работы командлета Group-Object

B4: Сгруппируйте полученные файлы по расширению. Затем отсортируйте их, используя число файлов с каждым из расширений.

Подсказка: Выведите файлы, сгруппируйте их и отсортируйте новый список по числам. Для этого используйте аргумент count.

Кроме Get-Member для получения информации об объектах можно использовать еще один командлет, Measure-Object. Даже если вы не полностью знаете список параметров, используемых в Measure-Object, можно как минимум оценить его возможности на следующих примерах. Попробуйте понять, какие результаты даст следующая цепочка команд:

```
Get-Childitem | Measure-Object length -Average -Sum -Maximum -Minimum
```

Возможно, после нескольких прочтений вы захотите проверить свои догадки. Все будет отлично работать и с использованием символов подстановки, и для обычных конвейеров командлетов Windows PowerShell.

B5: Определите общий размер всех файлов TMP. На втором шаге выведите ТОЛЬКО общий размер.

Подсказка: После первой попытки поместите всю строку в (). После запуска цепочки повторите команду с добавлением Get-Member для вывода всех атрибутов для получения результата (Вы помните? Windows PowerShell работает с объектами и переводит их в текст, так что их можно прочитать на экране!). Найдите свойство, которое соответствует вашим результатам в (), которое может также соответствовать атрибуту «Total». Помните цикл ForEach и как вы находили свойство «Status»? Да, именно: «object.status». А здесь нам нужен «total», а не «status». Измените текст примера соответствующим образом.

Удаление файлов

Windows PowerShell также содержит все необходимые команды для удаления файлов. Используя командлет Remove-Item можно удалять не только файлы. Он работает аналогично Get-Childitem. Может быть будет разумно создать резервную копию папки с упражнениями. Если вы случайно удалите слишком многое, вы сможете по крайней мере начать сначала.

B6: Удалите все файлы TMP с помощью `Remove-Item` с нужными аргументами!

Иногда возникает необходимость удалить файлы, значения параметров которых выходят за нужные нам границы. В этом случае можно использовать командлет `Where-Object`. Как и для команды `if`, мы можем определить состояние, которому должны удовлетворять выбираемые объекты из списка. Давайте рассмотрим пример со службами. Используя `Get-Service | Where-Object {$_.status -Eq "stopped"}` можно просмотреть только остановленные сервисы.

B7: Теперь удалите все файлы более 2 МБайт. 2 МБайт приблизительно соответствуют 2000000 байт.

Подсказка: Создавайте свой финальный сценарий шаг за шагом. Сначала создайте список всех файлов и отфильтруйте их по размеру (`...length -gt 2000000`). Вы получите новый список, который можно обрабатывать в цикле. Затем выведите только имена файлов (`$_ .fullname`). Эти имена можно использовать для запуска `Remove-Item`. Каждый раз, когда вам надоест слишком длинная строка команд, вводите переменные.

Кстати, вы не можете ввести 2 МБ как 2000000 (кроме всего прочего, это лишь приблизительное значение). Лучше прямо ввести в качестве размера 2MB, Windows PowerShell отлично понимает такой формат. Можно также потребовать сосчитать сумму 512KB + 512KB. Для вычислений требуется просто ввести числа непосредственно в командную оболочку, использовать особые командлеты не требуется.

Создание папок

Теперь попробуем внести в хаос наших файлов некоторый порядок. Мы создадим отдельные подпапки для файлов всех типов, а затем переместим соответствующие файлы в папки. Для этого нам нужен командлет для создания нового «элемента»¹ в файловой системе: `New-Item`. Он использует имя как аргумент, а тип как параметр, например, `directory` для каталога. Вы можете создать новый каталог «Test» следующим образом:

```
New-Item .\test -Type Directory
```

Чтобы упростить вам жизнь, мы еще раз рассмотрим команду сортировки: `Get-Service | Sort-Object status` вы уже знаете, а теперь попробуем

```
Get-Service | Sort-Object status -Unique
```

Эта последовательность команд возвращает только один элемент для каждого статуса. Попробуйте ее выполнить. Теперь у вас есть все, что нужно для создания папок и каталогов.

B8: Создайте в папке с упражнениями отдельные подпапки для файлов с разными расширениями.

Подсказка: Создайте список файлов и выберите их только по атрибуту «Extension». Теперь рассортируйте их с параметром `-Unique`. Вы увидите список расширений файлов. Когда это будет сделано, вы сможете присвоить этот список переменной и перейти к следующему шагу – с помощью цикла пройти по списку объектов и создать подпапки с именами расширений (`.extension`). Помните, что для этого следует использовать полный путь с как минимум одним символом

¹ Такие файловые системы, как FAT или NTFS, не являются на самом деле объектно-ориентированными. Поэтому мы используем команду `New-Item`, а не `New-Object`. В будущем это может измениться.

«\»). Если с путем возникают проблемы, попробуйте использовать в качестве аргумента (".\New"+\$_.Extension). Не забывайте указать тип объекта (directory) для создания каталога.

Для перемещения файлов в нужное место мы используем команду Move-Item. В качестве аргументов этот командлет использует полное имя исходного объекта и путь, указывающий на конечное положение, например, Move-Item .\test.txt .\новаяпапка

V9: Переместите все файлы из тестовой папки в созданные подпапки.

Подсказка: Список Get-Childitem из оригинальной папки теперь содержит новые подпапки, которые следует фильтровать. Создайте новый список всех элементов (сначала внимательно просмотрите список).

Отфильтруйте список с помощью оператора сравнения по регулярному выражению (...type -notmatch "d"). Затем следует применить к списку, который теперь содержит только файлы, цикл. Последний шаг прост: для каждого объекта найдите соответствующую расширению файла папку и переместите файл в эту папку. Для сохранения промежуточных результатов всегда можно использовать переменные.

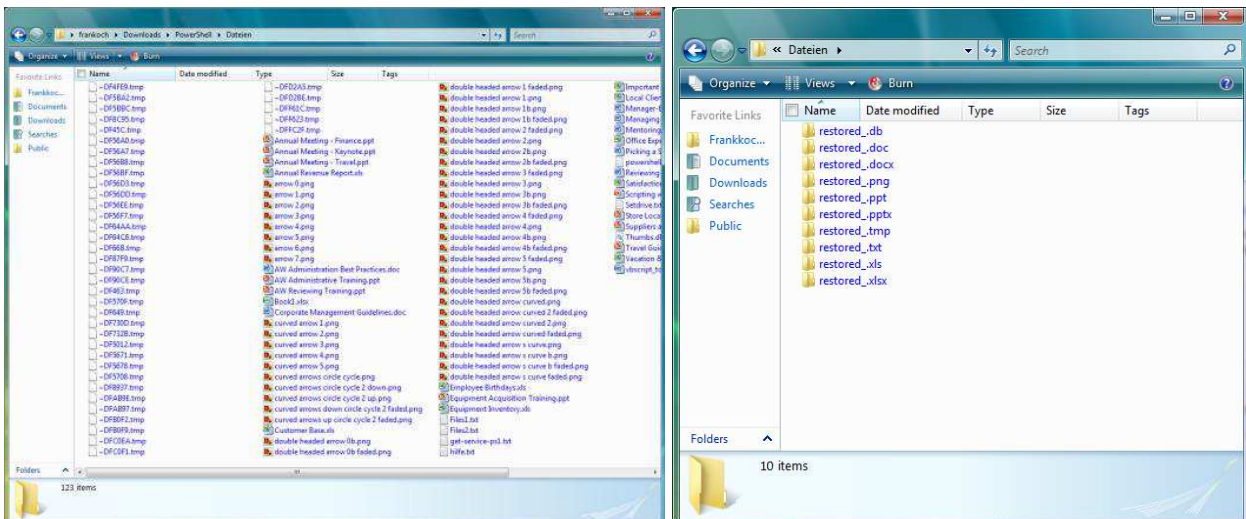


Рисунок 9: папка с файлами до сортировки

Рисунок 10: папка с файлами после сортировки

И наконец, мы снова выводим все эти файлы. Исходная папка теперь будет пуста, а подпапки заполнены. Get-Childitem -Recurse покажет это в деталях. Давайте сохраним эти результаты в файл TXT, чтобы вы могли изучить его в Блокноте (Notepad).

V10: Выведите содержимое папки с упражнениями, включая все подпапки, в текстовый файл, и сохраните его под именем FinalOutput.txt.

V11: Если у вас имеются исходные файлы упражнений, вы можете сделать еще одну вещь: Чтобы еще более упростить себе жизнь, можно сбросить атрибут read-only для каждого файла Word. Для этого перейдите в подпапку для файлов .doc и вызовите все объекты. Атрибут объекта, который следует установить, называется IsReadOnly, ему следует присвоить значение 0 (числовой ноль).

Подсказка: Используйте две команды: создайте список всех объектов, а затем с помощью цикла пройдите по объектам, как делали это раньше в других упражнениях.

Windows PowerShell также поможет вам с ACL, вашими спецификациями защиты. Используя командлеты `Get-Acl` и `Set-Acl`, вы легко можете переносить их от одного объекта к другому и даже создавать новые. Однако это выходит за рамки нашего курса. Дополнительные сведения по ACL можно найти в интерактивной справке.



```
dirbt - Notepad
File Edit Format View Help

Directory: Microsoft.PowerShell.Core\FileSystem::c:\users\frankoch\downloads\powershell\dateien\restored_db

Mode                LastWriteTime         Length Name
----                -
-a----           12.11.2006    06:06         84992 Thumbs.db

Directory: Microsoft.PowerShell.Core\FileSystem::c:\users\frankoch\downloads\powershell\dateien\restored_doc

Mode                LastWriteTime         Length Name
----                -
-a----           24.08.2005    18:51    192512 Aw Administration Best Practices.doc
-a----           08.07.2005    17:42    192000 Corporate Management Guidelines.doc
-a----           08.07.2005    17:42    194560 Manager-Employee Conduct.doc
-a----           08.07.2005    17:42    193024 Managing Your Store.doc
-a----           08.07.2005    17:42    185344 Mentoring New Managers.doc
-a----           08.07.2005    17:42    194560 Picking a Store Location.doc
-a----           08.07.2005    17:42    193024 Reviewing Employees.doc

Directory: Microsoft.PowerShell.Core\FileSystem::c:\users\frankoch\downloads\powershell\dateien\restored_docx

Mode                LastWriteTime         Length Name
----                -
-a----           18.03.2007    12:37    134718 vbscript_to_powershell.docx

Directory: Microsoft.PowerShell.Core\FileSystem::c:\users\frankoch\downloads\powershell\dateien\restored_png

Mode                LastWriteTime         Length Name
----                -
-a----           20.08.2005     00:13     1806 arrow 0.png
-a----           20.08.2005     00:13     1946 arrow 1.png
-a----           20.08.2005     00:13     2180 arrow 2.png
-a----           20.08.2005     00:13     2371 arrow 3.png
-a----           20.08.2005     00:12     2604 arrow 4.png
-a----           20.08.2005     00:12     2746 arrow 5.png
-a----           20.08.2005     00:12     3008 arrow 6.png
-a----           20.08.2005     00:12     3303 arrow 7.png
-a----           20.08.2005     00:13     3181 curved arrow 1.png
-a----           20.08.2005     00:13     4163 curved arrow 2.png
-a----           20.08.2005     00:13     5822 curved arrow 3.png
-a----           20.08.2005     00:13     7277 curved arrow 4.png
-a----           20.08.2005     00:13    14915 curved arrow 5.png
-a----           20.08.2005     00:13    11208 curved arrows circle cycle 2 down.png
-a----           20.08.2005     00:13    11297 curved arrows circle cycle 2 up.png
-a----           20.08.2005     00:13    20193 curved arrows circle cycle.png
-a----           20.08.2005     00:13    11981 curved arrows down circle cycle 2 faded.png
-a----           20.08.2005     00:13    12142 curved arrows up circle cycle 2 faded.png
-a----           20.08.2005     00:12     3835 double headed arrow 0b faded.png
-a----           20.08.2005     00:12     3647 double headed arrow 0b.png
-a----           20.08.2005     00:12     4161 double headed arrow 1 faded.png
-a----           20.08.2005     00:12     3885 double headed arrow 1.png
-a----           20.08.2005     00:12     4697 double headed arrow 1b faded.png
-a----           20.08.2005     00:12     4392 double headed arrow 1b.png
-a----           20.08.2005     00:12     4312 double headed arrow 2 faded.png
-a----           20.08.2005     00:12     4031 double headed arrow 2.png
```

Рисунок 11: Вывод содержимого подпапок

Если у вас есть время ...

Давайте вернемся к началу, к командлетам `Export-Csv` и `Export-CliXML`. Используйте свою переменную `$P`, если она еще доступна, а если нет, введите в Windows PowerShell следующую строку:

```
$p = Get-Process
```

Теперь сохраните переменную в файл CSV и файл CliXML:

```
$p | Export-Csv .\test.csv
```

```
$p | Export-CliXML .\test.xml
```

Далее импортируйте эти значения в две новые переменные:

```
$p1 = Import-Csv .\test.csv
```

```
$p2 = Import-CliXML .\test.xml
```

C1: Вычислите среднее использование процессорного времени, максимальное и минимальное значения.

Подсказка: Используйте команду `Measure-Object`, как в примерах выше. Делайте это по отдельности для трех переменных `$p`, `$p1` и `$p2`.

C2: Теперь отсортируйте переменные, которые показаны в списке, по использованию процессорного времени, и выберите первые пять. Снова сделайте это для каждой из трех переменных \$p, \$p1 и \$p2. Все ли результаты остались теми же? Какая переменная отличается от других в разных случаях? Что здесь неправильно?

Разгадка относительно проста. При экспорте в файл CSV и последующем импорте Windows PowerShell теряет информацию о типе значений. Это означает, что при использовании списка чисел, которые теперь имеют форму строк, 8,0 внезапно оказывается больше, чем 800. В файлах XML эта информация сохраняется и может использоваться для правильной сортировки. Вычислить среднее значение и т.п. легко, так, $8,0 + 800$ равно $808,0$ а значит выражение для `Measure-Object` правильно, но для сортировки оно не подходит. Убедитесь, что вы приняли это во внимание при сохранении переменных. XML может оказаться более безопасным форматом.

РАБОТА С ДРУГИМИ ОБЪЕКТАМИ

Windows PowerShell как машина обработки произвольных объектов

С помощью Windows PowerShell можно работать не только с собственными объектами; можно получить доступ ко всему миру объектов, включая WMI, .NET и даже COM. Далее идут уроки по объектам, требующим отдельного обучения. По указанным вопросам можно найти большое количество дополнительной литературы. Сейчас мы даже не будем начинать рассматривать эти объекты, ограничившись небольшим примером для каждого. Надеюсь, это вызовет у вас желание узнать о них больше.

Объекты WMI

Вы, вероятно, знаете о существовании объектов WMI из Windows Scripting Host WSH и VBScript. Если нет, добро пожаловать в тему, но не ожидайте, пожалуйста, глубокого рассмотрения WMI. Мы сосредоточимся исключительно на контексте Windows PowerShell.

Мы создаем объекты WMI в Windows PowerShell с помощью специального командлета `Get-WmiObject`. Уже это показывает, как важен для нас WMI. Войдите в Windows PowerShell и введите следующую команду:

```
Get-WmiObject -Class win32_computersystem
```

Вы увидите на экране базовую информацию о своей системе. В противоположность VBScript или другим языкам, Windows PowerShell убергает вас от сложного синтаксиса, сводя количество вводимых данных до абсолютного минимума.

Нам необходимы только:

- Командлет `Get-WmiObject` для определения того факта, что мы хотим работать с WMI
- Соответствующий класс WMI, с которым мы хотим работать, т.е. `-Class win32_computersystem`

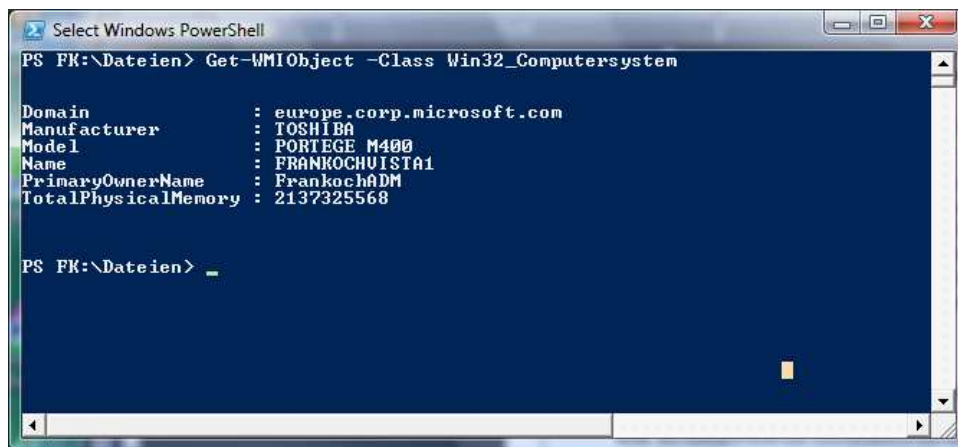


Рисунок 12: вывод для объекта WMI WIN32_COMPUTERSYSTEM

Выводимые данные, разумеется, составляют малую часть данных этого объекта. Для вывода списка атрибутов используйте команду Windows PowerShell `Get-Member`.

D1: Выведите атрибут «Имя пользователя» («User name») своей системы.
Подсказка: Используйте исходный пример WMI и укажите соответствующий атрибут для имени пользователя.

WMI, как мы говорили, является собственным миром. Мы можем не только читать информацию, но и изменять ее.

Эту операцию можно проделывать даже по сети с внешними системами, в том случае, если мы можем идентифицировать себя. Чтобы войти в мир WMI, можно использовать браузеры графических объектов, такие, как CIM Studio. Литература по этой теме даст вам всю необходимую информацию. Однако я буду рад показать вам несколько небольших примеров. Полезную помощь в повседневной работе с ИТ могут предоставить следующие классы объектов WMI:

Список информации о настройках рабочего стола

```
Get-wmiobject -Class win32_desktop -Computersname .
```

Точка является частью команды и показывает, что вы имеете в виду свой локальный компьютер. В других случаях используйте другое имя компьютера(server1, server4.mycompany.ch, и т.д.).

Информация о BIOS вашей системы:

```
Get-wmiobject -Class win32_bios
```

Если вы хотите обратиться к своему компьютеру, параметр «-Computersname .» необязателен.

Список всех установленных исправлений

```
Get-wmiobject -Class win32_quickfixengineering или:
```

```
Get-wmiobject -Class win32_quickfixengineering -Property hotfixid |
```

```
Select-Object -Property hotfixid
```

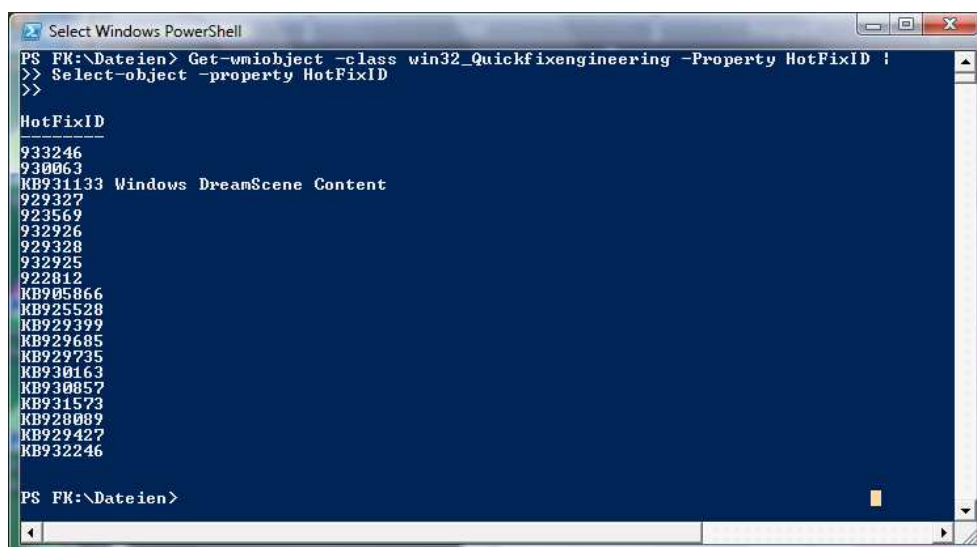


Рисунок 13: Вывод исправлений в виде номеров KB

Запись данных в объекты WMI производится так же, как в объекты Windows PowerShell, однако после установки новых значений следует вызвать метод Put объекта, чтобы применить сделанные изменения. Вызовите объект командой Get-Member, чтобы просмотреть его атрибуты и методы. Вы увидите, есть ли у вас возможность записывать соответствующие значения (set), или их можно только читать (get). Атрибут «Visible» в последующих примерах COM является одним из тех, которые можно как читать, так и записывать.

Работа с объектами .NET и XML

Возможности использования .NET и XML впечатляют не меньше, чем возможности WMI. В приложении вы найдете два примера. Один вызывает веб-сайт в сети, подключается к ленте RSS и считывает из нее все темы, а также их полные URL, в точности как ваша собственная персональная программа для чтения RSS. Первый пример впечатляюще краток и приведен здесь только в демонстрационных целях. Мы вынуждены исключить подробное обсуждение, поскольку оно не входит в круг вопросов, рассматриваемых в этой книге. В замечательной книге о Windows PowerShell «Windows PowerShell in Action» Брюса Пайетта (Bruce Payette), родителя Windows PowerShell подробно рассмотрены оба сценария.

```
([xml](new-object net.webclient).DownloadString(  
http://blogs.msdn.com/powershell/rss.aspx)).rss.channel.item |  
Format-Table title,link
```

Да, вы все правильно понимаете, все что нужно для этого – две строки сценария. Синтаксис также схож с WMI. Теперь давайте заглянем внутрь скобок [XML]. Внутри имеется простой для понимания командлет new-object. Вы можете использовать его для создания нового объекта. Поскольку мы явно вводим тип (net.webclient), Windows PowerShell немедленно понимает, что это объект .NET типа webclient.

Для этого объекта мы используем метод DownloadString(url), который принимает URL-адрес (нужного веб-сайта).

Все остальное – просто «обрамление»: поскольку мы знаем, что скрывается за этим адресом, мы не проверяем его; вместо этого мы считаем, что это лента RSS и вызываем объект с атрибутами, которые, как мы знаем, имеются у лент RSS. Если ввести другой URL-адрес, сценарий скорее всего не будет работать, если только вам не посчастливится наткнуться на другую ленту RSS.

format-table также является обычным командлетом и просто выводит список объектов в виде таблицы. Вы можете прямо ввести столбцы, которые хотите видеть в виде заголовка (title), и URL-адрес (link) ленты RSS.

В приложении вы можете найти второй пример, обозначенный «Пример 4». Этот сценарий также использует возможности работы с объектами .NET в сценариях Windows PowerShell. В нем используется компонент Windows WinForms.

Скопируйте пример в Windows PowerShell и запустите его. Результат будет достаточно впечатляющим, чтобы вам захотелось узнать больше. Сведения по программированию для .NET и объектах .NET можно найти как в Microsoft MSDN, так и в одной из множества опубликованных книг по .NET.

Работа с COM-объектами

В нашем последнем примере я буду рад кратко рассмотреть возможности COM-объектов. Я использую пример для Excel. Если на вашем ПК не установлен Excel, вы можете просмотреть альтернативную версию с Internet Explorer.

Однако я не собираюсь приводить здесь обширное введение в COM. Этот вопрос рассмотрен в таком количестве книг, что я действительно не смогу добавить ничего нового.

Итак, мы говорим Windows PowerShell, что хотим работать с объектом COM. Чтобы гарантировать, что позднее мы ничего не забудем, немедленно назначим объекту переменную. Это означает, что мы получим собственный объект COM определенного типа. Специального командлета для использования COM-объектов, в отличие от объектов WMI, не существует. Вместо этого мы используем стандартный командлет для новых объектов: `new-object`.

```
$a = new-object -comobject excel.application
```

В качестве аргумента мы укажем, какой COM-объект хотим использовать. В данном случае это Excel, и мы используем аргумент `excel.application`. Как мы увидим, в вашей системе имеются COM-объекты, для которых полные имена не очевидны. Я советую вам посмотреть информацию об этом в Microsoft MSDN или одной из множества книг о COM. Кроме использования Excel в качестве набора данных для создания отчетов, в повседневных задачах ИТ-администрирования вы можете применять Visio для графического отображения системных значений, автоматизировав эту задачу с помощью Windows PowerShell. В каталоге издательства MS Press вы найдете несколько книг, действительно хорошо описывающих COM-объекты Visio. Однако давайте вернемся к нашему примеру с Excel.

Для добавления данных в Excel нам нужна рабочая книга. Поскольку мы вошли в мир COM, нам следует пользоваться синтаксисом COM. Windows PowerShell поможет нам сохранить все, что мы делаем, достаточно простым. Мы можем посмотреть синтаксис, используя `$a | Get-Member`. Команда `Get-Member` выводит полный список атрибутов и методов и для COM-объектов. Просто от таких приложений, как Excel, следует ожидать большего, чем маленькие объекты, которые мы видели раньше. Метод `workbooks.Add()` создаст для нас новую рабочую книгу. Нам следует также иметь возможность загрузить существующую рабочую книгу. Для этого необходимо указать путь. Но мы просто создадим новую рабочую книгу из ничего:

```
$b = $a.workbooks.Add()
```

Возможно, при создании книги Excel вы получите сообщение об ошибке вида «`Error: 0x80028018 (-2147647512) Description: Old Format or Invalid Type Library`». Эта проблема описана в статье № 320369 из базы знаний Microsoft. Обычно эта ошибка возникает, если Excel установлен на языке (например, `English (US)`), отличающемся от региональных настроек Windows (например, `English (UK)`). Это ошибка в объекте Excel. Во время написания этой книги Microsoft еще не выпустила исправления этой ошибки. В качестве временного варианта в этом случае для тестирования измените региональные настройки в параметрах системы на `English (US)` и перезапустите Windows PowerShell. После этого пример для Excel должен работать без ошибок.

Листы, входящие в рабочую книгу, создаются автоматически. Мы выберем первый:

```
$c = $b.worksheets.Item(1)
```

Если мы хотим записать что-то на лист, это следует делать не в лист, а в ячейку на листе Excel. Это означает, что нам необходимо ввести строку:

```
$c.Cells.Item(1,1) = "windows PowerShell rocks!"
```

И это все. Наша запись в Excel готова. Вы не верите? Хорошо, давайте выведем Excel на экран, можете посмотреть сами. Команда

```
$a.Visible = $true
```

задает атрибуту «Visible» объекта Excel значение \$true. И как по волшебству Excel показывает нам результат.

Так же, как атрибут «Visible», мы можем вызывать методы (функции) Excel и использовать их в наших целях. Вы можете, разумеется, догадаться, что делает следующая строка:

```
$b.SaveAs(".\Test.xls")
```

Если вас это удивляет: нет, мы здесь сохраняем не Excel, а соответствующую рабочую книгу Excel. Нас интересует файл XLS, а не программа.

Разумеется, эта упрощенная демонстрация. Мы не показали в подробностях, как получить информацию при работе с Excel. Знаете ли вы, что наборы таблиц Excel называются рабочими книгами? И как следует использовать команду «SaveAs» и ее синтаксис? Все это доступно через COM-объекты, и для изучения этого материала вы можете использовать любые источники информации о COM, чтобы использовать эти знания потом в Windows PowerShell.

Get-Member всегда будет для вас главным источником подсказок. Поскольку мы говорим только об инструменте Windows PowerShell, я ограничусь использованием информации, не раскрывая ее источник. В конце мы должны очистить и закрыть Excel, если не собираемся ничего делать вручную:

```
$a.Quit()
```

Если хотите, попытайтесь выполнить следующее небольшое упражнение:

D2: Создайте список всех служб и введите имена и статусы служб в таблицу Excel. Подсказка: Используйте приведенный выше пример для создания объекта Excel и присвоения его переменной. Чтобы указать строку в таблице Excel, мы используем отдельную переменную \$i. Используйте сценарий для вывода служб разными цветами и замените строку с выводом цветом на запись \$c.Cells.Item(\$i,1).

Не забывайте увеличивать \$i после каждой строки, например с помощью выражения \$i = \$i + 1. Вы можете вводить в одну строку несколько команд, используя точку с запятой «;». Сохраните результаты из Excel в файле XLS, но пожалуйста, автоматически, с помощью своего сценария, а не вручную, через меню Excel.

В результате вы обнаружите, что статус служб выводится в Excel в виде числа. Дружественный Windows PowerShell заменяет эти числа текстом «running» или «stopped», который более удобен для пользователя.

| Service Name | Service Status | |
|--------------------------------|----------------|---|
| AdtAgent | ✓ | 1 |
| AeLookupSvc | ✗ | 4 |
| ALG | ✓ | 1 |
| Appinfo | ✓ | 1 |
| AppMgmt | ✓ | 1 |
| AudioEndpointBuilder | ✗ | 4 |
| Audiosrv | ✗ | 4 |
| BFE | ✗ | 4 |
| BITS | ✗ | 4 |
| Browser | ✓ | 1 |
| BthServ | ✗ | 4 |
| CcmExec | ✗ | 4 |
| CertPropSvc | ✗ | 4 |
| clr_optimization_v2.0.50727_32 | ✓ | 1 |
| COMSysApp | ✓ | 1 |
| CryptSvc | ✗ | 4 |
| CscService | ✗ | 4 |
| DcomLaunch | ✗ | 4 |
| DFSR | ✓ | 1 |
| Dhcp | ✗ | 4 |
| Dnscache | ✗ | 4 |
| dot3svc | ✓ | 1 |

Рисунок 14: Вывод служб и их статуса в Excel 2007. Также используется специальный формат Excel 2007 для вывода значений статуса в виде значков, а не чисел.

Если на вашем ПК не установлен Excel, вы можете использовать другое упражнение, использующее Internet Explorer. Вместо ввода данных в ячейку мы перейдем на веб-сайт. По сравнению с ранее показанным сценарием для чтения RSS нам потребуется только один дополнительный шаг: при автоматическом чтении ленты RSS будем искать в заголовке ключевые слова и автоматически вызывать веб-страницы. Навигация с дивана никогда еще не была такой простой.

Сценарий начинается, как и сценарий для Excel, с вызова new-object:

```
$ie = New-Object -Comobject InternetExplorer.application
```

В этом случае новый объект также изначально невидим, и как и для сценария Excel, для этой проблемы имеется очень простое решение:

```
$ie.Visible = $True
```

Чтобы просмотреть возможности, предоставляемые объектом Internet Explorer, мы также используем командлет Get-Member:

```
$ie | Get-Member
```

Чтобы упростить работу, мы будем искать по популярному веб-сайту:

```
$ie.Navigate(http://www.microsoft.com/powershell)
```

Если хотите, можно объединить этот сценарий со сценарием чтения RSS. Вы увидите множество интересных ссылок и заголовков. Возьмите список заголовков, проведите поиск ключевых слов, и при их обнаружении перейдите на нужную страницу. В нашем курсе мы еще не рассматривали поиск ключевых слов в списке заголовков; воспользуйтесь для получения дополнительных сведений справкой Windows PowerShell. В этой книге я не буду приводить решения этой задачи.

Работа с журналами сообщений

В завершение наших практических упражнений в этой книге мы кратко рассмотрим журналы сообщений. Windows PowerShell обеспечивает доступ к журналам сообщений с помощью довольно сложных методов. Объекты WMI, .NET и COM имеют для этого множество команд. И даже в Windows PowerShell имеется несколько командлетов, которые всегда могут вам помочь. Наиболее важный из них – это `Get-Eventlog`.

Команда `Get-Eventlog -List` выводит на экран все журналы системы. Чтобы получить возможность доступа к конкретному журналу, мы можем воспользоваться командлетом `Where-Object`. Например, доступ к системному журналу производится следующим образом¹:

```
Get-Eventlog -List | where-Object {$_.logdisplayname -eq "System"}
```

И хотя это самый простой путь, получившаяся строка кода все равно будет слишком длинной для того, чтобы постоянно ее использовать.

Таким образом, если вы хотите работать с записями из журнала сообщений, просто запустите `Get-Eventlog`, добавив имя нужного журнала. Результат может оказаться слишком объемным, но его вывод можно в любой момент остановить, нажав CTRL-C. Если вы хотите увидеть только последние 20 записей, используйте параметр `-newest 20`. Разумеется, вы можете заменить 20 на любое другое число.

```
Get-Eventlog system -Newest 3
```

```
Get-Eventlog system -Newest 3 | Format-List
```

События можно обрабатывать как обычно в Windows PowerShell, сортировать и группировать.

E1: Найдите имя журнала событий Windows PowerShell. Сгруппируйте сообщения по коду ID сообщения, а затем отсортируйте по имени. Вторым шагом выведите список событий с ID 403.

Подсказка: Если имя журнала событий содержит пробелы, вводите полное имя в кавычках "мой журнал сообщений".

E2: Отсортируйте последние 15 записей в системном журнале событий по коду ID в нисходящем порядке.

Подсказка: Если вы не можете написать ответ немедленно, прочтите книгу еще раз.

¹ В русской версии Windows XP этот журнал называется «Система». Если вы используете эту версию ОС, измените имя в строке сценария.

ОТВЕТЫ К УПРАЖНЕНИЯМ

Сценарии-решения к упражнениям в этой книге

A1

```
get-process | sort-object CPU
```

A2a

```
get-process | sort-object CPU -descending | select-object -first 10
```

```
get-process | sort-object CPU | select-object -last 10
```

A3

```
$P = get-process | sort-object CPU -descending | select-object -first 10
```

A4

```
$P > .\a4.txt
```

```
$P | export-csv .\a4.csv
```

```
$P | export-CliXML .\a4.xml
```

A5

```
get-service | sort-object status
```

A6

```
get-service | foreach-object{ write-host $_.name $_.status}
```

A7

```
get-service | foreach-object{ write-host -f yellow -b red $_.name $_.status}
```

вместо -f можно написать также -foregroundcolor, а вместо -b – -backgroundcolor

A8

```
get-service | foreach-object{
```

```
if ($_.status -eq "stopped") {write-host -f green $_.name $_.status}`
```

```
else{ write-host -f red $_.name $_.status}}
```

A9

```
get-service | convertto-html
```

A10

```
get-service | convertto-html > .\a10.html
```

A11

```
get-service | sort-object status | convertto-html name, status > .\a10.html
```

B1

```
get-childitem * -exclude *.tmp | select-object name, length
```

B2

```
get-childitem * -exclude *.tmp | select-object name, length | sort-object length, name
```

B3

```
get-childitem | get-member
```

B4

```
get-childitem | group-object extension | sort-object count
```

B5

```
(get-childitem .\*.tmp | measure-object length -sum).sum
```

B6

```
remove-item .\*.tmp
```

B7

```
get-childitem | where-object {$_.length -gt 2000000}
```

```
| foreach-object {remove-item $_.fullname}
```

B8

```
get-childitem | select-object extension | sort-object extension -unique |
```

```
foreach-object {new-item (".\New" + $_.extension) -type directory}
```

B9

```
get-childitem | where-object {$_.mode -notmatch "d"} |
```

```
foreach-object {$b= ".\New" + $_.extension; move-item $_.fullname $b}
```

B10

```
get-childitem -recurse > .\finaloutput.txt
```

B11

```
get-childitem *.doc | foreach-object {$_.Isreadonly = 0}
```

C1

```
$p | measure-object CPU -min -max -average
```

C2

```
$p | sort-object CPU -Descending | Select-Object -first 5
```

D1

```
(get-wmiobject -class win32_computersystem).username
```

D2

```
$a = new-object -comobject excel.application
```

```
$a.Visible = $True
```

```
$b = $a.Workbooks.Add()
```

```
$c = $b.Worksheets.Item(1)
```

```
$c.Cells.Item(1,1) = "Service Name"
```

```
$c.Cells.Item(1,2) = "Service Status"
```

```
$i = 2
```

```
get-service | foreach-object{ $c.cells.item($i,1) = $_.name
```

```
$c.cells.item($i,2) = $_.status; $i=$i+1}
```

```
$b.SaveAs("C:\Users\frankoch\Downloads\Test.xls")
```

```
$a.Quit()
```

E1

```
get-eventlog "Windows PowerShell" | group-object eventid | sort-object name
```

```
get-eventlog "Windows PowerShell" | where-object {$_.eventid -eq 403}
```

E2

```
get-eventlog system -newest 15 | sort-object eventid -descending
```


ПРИЛОЖЕНИЕ

ПРИМЕРЫ СЦЕНАРИЕВ

Примеры к Windows PowerShell – от простых к сложным

Вы можете просто скопировать следующие сценарии и запустить их в Windows PowerShell. Они демонстрируют теоретические возможности PowerShell, а также то, что краткое введение не в состоянии описать все возможности Windows PowerShell. Чтобы скопировать сценарии, выделите текстовые строки сразу после первого символа «>» до комментариев (не включая комментарии). Эти упражнения в основном взяты из горячо рекомендуемой книги «Windows PowerShell in Action» Брюса Пайетта, создателя Windows PowerShell. В этой книге примеры снабжены очень подробными пояснениями, на случай если вам потребуется дополнительная информация.

Пример 1: Прямой вывод строки

Самая короткая программа "Hello world":

```
"Hello world"
```

Комментарии:

Windows PowerShell может непосредственно распознавать введенную строку и выводить ее на экран.

Пример 2: Анализ файла журнала

Создает список всех файлов журналов в папке «Windir»; находит файлы, содержащие слово «Error», выводит имя файла журнала и строку с ошибкой:

```
dir $env:windir\*.log | Select-String -List error | Format-Table path,linenumber -AutoSize
```

Комментарии:

Может возникнуть сообщение об ошибке, связанной с недостатком полномочий и т.д. Игнорируйте подобные сообщения.

Пример 3: Ваша программа для чтения RSS

Вызывает веб-страницу, читает ленту RSS, выводит сообщения из ленты RSS и их URL-адреса:

```
([xml](New-Object net.webclient).DownloadString(
```

```
"http://blogs.msdn.com/powershell/rss.aspx")).rss.channel.item | Format-Table title,link
```

Комментарии:

В самом начале соединение с веб-сайтом может устанавливаться достаточно медленно.

Пример 4: Добавление окон в сценарий

Создает отдельную форму WinForm для графического вывода информации:

```
[void][reflection.assembly]::LoadWithPartialName("System.Windows.Forms")
```

```
$form = New-Object Windows.Forms.Form
```

```
$form.Text = "My First Form"
```

```
$button = New-Object Windows.Forms.Button
```

```
$button.text="Push Me!"
```

```
$button.Dock="fill"
```

```
$button.add_click({$form.close()})
```

```
$form.controls.add($button)
```

```
$form.Add_Shown({$form.Activate()})
```

```
$form.ShowDialog()
```

Комментарии:

Для завершения просто щелкните по новому окну (может находиться за окном Windows PowerShell).

ПРИЛОЖЕНИЕ

ТЕОРИЯ WINDOWS POWERSHELL

Теоретические принципы Windows PowerShell

Windows PowerShell – краткое введение

Преыдущие попытки Microsoft по созданию оболочек командной строки были не слишком успешны. Старый `command.com`, вероятно, соответствовал уровню первых версий MS DOS, но растущее число функций операционной системы вскоре превысило его возможности. Оболочка `cmd.exe`, появившаяся в Windows NT, предоставляла пользователям дополнительные средства. Но по сравнению с популярными оболочками Unix, такими как Bash, командная строка Microsoft, несомненно, имела множество недостатков.

Теперь Microsoft полностью изменила ситуацию. Создавая Windows PowerShell (ранее называлась Monad Shell, MSH), разработчики средств администрирования хотели дать нам оболочку для Windows, которую можно будет использовать для написания любых сценариев для управления системой. Windows PowerShell следует совершенно новой концепции, в отличии от тексто-ориентированных оболочек, таких как `bash`.

Цели разработки Windows PowerShell

Windows PowerShell – это новая оболочка командной строки Windows, разработанная специально для системных администраторов. Эта оболочка содержит интерактивную командную строку и среду сценариев, которые можно использовать как по одиночке, так и вместе. В противоположность большинству оболочек командной строки, которые принимают и возвращают текст, Windows PowerShell базируется на объектной модели, появившейся в .NET Framework 2.0. Это фундаментальное отличие в среде позволяет использовать для управления и настройки Windows совершенно новые инструменты и методики.

Windows PowerShell вводит идею командлетов (пишется «`cmdlet`»). Командлет – это простой инструмент командной строки, интегрированный в оболочку и выполняющий единственную функцию. Хотя вы можете использовать командлеты поодиночке, их мощь становится более очевидной при использовании комбинаций командлетов для выполнения сложных задач. Windows PowerShell содержит несколько сотен базовых командлетов, а также дает возможность создавать собственные командлеты и сценарии и передавать их другим пользователям.

Как многие другие командные оболочки, Windows PowerShell дает вам доступ к файловой системе компьютера. Благодаря поставщикам Windows PowerShell вы получаете легкий доступ к другим хранилищам данных, таким как реестр и хранилища сертификатов.

О тексте, разборе текста и объектах

PowerShell полностью объектно-ориентирована. По сравнению с обычными оболочками она работает иначе, результатом выполнения команды будет не текст, а объект (далее этот момент будет рассмотрен подробнее). Но аналогично более ранним популярным оболочкам, она имеет конвейер, в который передаются и в котором

обрабатываются результаты отдельных команд. Единственная разница состоит в том, что исходные и промежуточные значения и результаты являются объектами, а не текстом.

Объекты PowerShell не отличаются от объектов в программах C++ или C#. Вы можете представить объект как блок данных с атрибутами и методами. Методы представляют собой действия, которые можно совершить с объектом.

Если, например, вы обращаетесь к службе из Windows PowerShell, вы на самом деле используете объект, соответствующий этой службе. Если вы выводите на экран информацию о службе, вы выводите атрибуты соответствующего объекта службы. И если вы запускаете службу, то есть меняете атрибут статуса службы на «запущен», вы используете метод объекта службы. По мере увеличения опыта вы лучше поймете преимущества обработки объектов и будете работать с объектами осознанно.

Объектно-ориентированная концепция PowerShell делает стандартный разбор текста командных оболочек Unix (анализ/оценка) и текстовую информацию со всеми ее проблемами и склонностью к ошибкам, полностью излишними. Чтобы пояснить это, рассмотрим следующий пример:

Предположим, вы хотите получить список всех процессов, которые расходуют более 100 дескрипторов. В традиционной командной оболочке Linux мы должны вызвать команду для просмотра процессов (ps -A). Эта команда возвращает текстовый список. Каждая строка будет содержать сведения о процессе, разделенные пробелами. Вы должны разобрать эти строки с помощью отдельного инструмента, отфильтровать коды ID процессов, после чего с помощью другой программы запросить число используемых процессом дескрипторов. Затем вы должны разобрать полученный в виде текста результат, отфильтровать нужные строки и вывести на экран соответствующий текст.

В зависимости от того, насколько хорошо вы обрезаете и фильтруете информацию из текста, выдаваемого функциями, этот подход заслуживает доверия в большей или меньшей степени. Однако, например, если заголовок столбца в выходных данных изменился, и имена процессов стали слишком длинными, у вас несомненно возникнут проблемы.

PowerShell использует принципиально другой подход. Вы также начинаете с команды get-process, которая возвращает все запущенные процессы в операционной системе. Только в этом случае вы получаете список объектов, состоящий из объектов процессов. Эти объекты можно исследовать на предмет их атрибутов и запросить значения прямо у них – таким образом, вам не нужно исследовать текстовые строки и разделять их на столбцы. Мы еще поговорим об этом более подробно.

Новый язык сценариев

В Windows PowerShell не используется какой-либо существующий язык, для нее был создан собственный. Причины этого таковы:

- . Windows PowerShell необходим язык для управления объектами .NET.
- . Язык должен поддерживать сложные задачи, не делая простые задачи сложными.
- . Язык должен соответствовать соглашениям других языков, используемых для программирования в .NET, таким как C#.

В настоящее время каждый язык имеет свои собственные команды. В Windows PowerShell мы обеспечили соответствие всех команд определенной логике с точки зрения конструкций и наименования. Командлет представляет собой специализированную команду, которая работает с объектами в Windows PowerShell. Вы можете узнать командлеты по их именам: глагол и существительное, всегда в единственном числе,

разделенные значком тире (-), например, `get-help`, `get-process` и `start-service`. В Windows PowerShell большинство командлетов очень просты и созданы для использования совместно с другими командлетами. Так, например, командлеты «Get» только извлекают данные, командлеты «Set» создают или изменяют данные, командлеты «Format» форматируют данные, а командлеты «Out» пересылают выводимые данные в указанное место.

Команды Windows и служебные программы

Вы привыкли использовать определенные команды. Новый язык, который не принимает во внимание этот факт, обречен на быстрое забвение. Поэтому в Windows PowerShell вы можете использовать стандартные команды Windows и запускать программы Windows, имеющие графический интерфейс, например Блокнот (Notepad) и Калькулятор (Calculator). Кроме того, как и в `Cmd.exe`, вы можете получать текстовый вывод из других программ и использовать этот текст в командной оболочке. Даже если команды, такие как `dir`, `ls` или `cd`, не следуют официальному синтаксису Windows PowerShell, они будут работать и могут использоваться без каких-либо проблем.

Интерактивная среда

Как и в других командных оболочках, в Windows PowerShell поддерживается полностью интерактивная среда. Если вы в ответ на приглашение вводите команду, она выполняется и результат выводится в окно командной оболочки. Вы можете пересылать результат работы команды в файл или на принтер, а также использовать оператор конвейера (`|`) для передачи его другой команде.

Поддержка сценариев

Если вы уже повторяли одни и те же команды, мы рекомендуем не вводить команды или последовательность команд по отдельности, а сохранить их в файле и выполнять этот файл. Файл, содержащий команды, называется сценарием.

Аналогично поддержке интерактивных интерфейсов, Windows PowerShell обеспечивает полную поддержку сценариев. Вы можете запустить сценарий, введя его имя в строке приглашения. Расширение файлов сценариев Windows PowerShell scripts - `.ps1`; вводить расширение файла необязательно.

Хотя сценарии используются очень часто, они также могут быть использованы для распространения вредоносного кода. Поэтому вы можете определить в Windows PowerShell политики безопасности (также называемые политиками выполнения), чтобы указать, какие сценарии могут запускаться, и должны ли они иметь цифровую подпись. Чтобы избежать ненужного риска, ни в каких политиках выполнения Windows PowerShell не разрешается выполнять сценарии двойным щелчком на их значках, что можно делать, например, со старыми файлами `.bat`, `.cmd` или `.vbs`.

CMD, WScript или PowerShell? Что выбрать?

Для Windows XP имеется три оболочки языков сценариев: старая добрая оболочка CMD, Windows Scripting Host для ваших сценариев VB или Jscript, а теперь и Windows PowerShell. Но не бойтесь, вам не нужно выбирать между оболочками или беспокоиться о том, что одна из них устареет. Даже в новых версиях Windows, таких как Vista или Longhorn Server, вы обнаружите, что все три оболочки остались равноправными. Вы можете использовать ту из них, которую предпочитаете, на ваш вкус. Вы также можете использовать оболочку, которая наилучшим образом подходит для конкретной задачи. Если

вы до сих пор написали немного сценариев, сейчас самое время начать работать с Windows PowerShell, чтобы иметь под рукой новейшую и самую простую в использовании технологию. Решая практические задачи, которые у вас есть или будут, вы увидите, как это просто и какими мощными могут быть простые сценарии, не перерывая гигантские тома и книги из 60-х и 70-х годов о программировании командных файлов.

Windows PowerShell 1.0

Хотя Windows PowerShell имеет версию 1.0, качество продукта весьма впечатляет, и я могу с чистой совестью рекомендовать его использование в практической работе. В основном это вызвано тем, что Windows PowerShell на самом деле является действительно новым способом использования популярной среды .NET Framework 2.0. Однако вам следует учитывать небольшой номер версии продукта.

Уже набор функций должен сказать вам, что не все, необходимое вам, уже имеется в Windows PowerShell. Windows PowerShell сейчас может обеспечить прямой доступ к файловым системам, журналам сообщений, записям реестра, интерфейсам и объектам .NET, WMI и ADSI; однако удаленная работа в нем еще не реализована. Это означает, что доступ к другим компьютерам работает с помощью средств WMI или .NET. В будущем мы сможем, вероятно, использовать для администрирования новые интерфейсы веб-служб, которые были сертифицированы в 2006 году. Сейчас следует удовлетвориться использованием существующих проверенных методов, таких как WMI и .NET.

ПРИЛОЖЕНИЕ

СЦЕНАРИИ И БЕЗОПАСНОСТЬ

Безопасность при использовании сценариев

Со службой Windows Scripting Host (WSH), введенной в Windows 2000, Microsoft заявила о создании новой мощной системы сценариев. Эта система была настолько мощной, что была использована вирусописателями в качестве новой арены. Неопытные пользователи получали несколько первых электронных писем с многообещающими привлекательными картинками, но когда они открывали вложение, там было не на что смотреть, они оборачивались сценариями VBScript, которые приступали к взлому системы. Windows PowerShell делает все возможное, чтобы противостоять этому типу угроз.

Так, базовые настройки Windows PowerShell предполагают запрет запуска любых сценариев.

Эта функция должна быть явным образом активирована системным администратором. Активация обеспечивает различные уровни безопасности, которые используют подписание сценариев. Кроме того, расширение файлов Windows PowerShell (PS1) ассоциировано с блокнотом (Notepad). Даже если ваша среда допускает выполнение сценариев, все неблагоприятные двойные щелчки по почтовым вложениям или файлам будут просто приводить к запуску Блокнота и демонстрации исходного кода. И наконец, Windows PowerShell всегда требует явного ввода полного пути для файлов, которые не находятся в папках обозначенных в переменной окружения %Path%. Это может предотвратить случайное выполнение программы, которую вы не хотели использовать, в Windows PowerShell.

Чтобы быть в состоянии запустить сценарий, вам следует изменить настройки безопасности для Windows PowerShell. Для этой цели используются два командлета: `get-executionpolicy` и `set-executionpolicy`. С помощью `get-executionpolicy` вы получаете существующие настройки. Существует четыре уровня безопасности:

| Значение политики | Описание |
|--------------------------------------|--|
| Restricted (Запрещено, по умолчанию) | Сценарии не запускаются |
| Allsigned (Все подписанные) | Запускаются только подписанные сценарии |
| RemoteSigned (Удаленные подписанные) | Разрешен запуск локальных сценариев, прочие сценарии должны быть подписаны |
| Unrestricted (Без ограничений) | Запускаются все сценарии |

Для изменения этих настроек системный администратор должен вызвать, например, команду

```
set-executionpolicy remotesigned
```

Microsoft предоставляют шаблон групповой политики, чтобы автоматически устанавливать ключ в больших организациях. Дополнительные сведения по этой теме можно найти в документации по Windows PowerShell.

ЗАКЛЮЧЕНИЕ

Я хочу поблагодарить свою жену Петру за ее любовь и участие. Она отказалась от множества воскресных поездок и вечеров со мной, когда я проводил время без нее, со своим компьютером.

Эта книга не была бы написана без разработчиков Windows PowerShell, и я искренне благодарен им. Особые благодарности я, разумеется, выражаю Брюсу Пайетту, который дал мне начальный импульс своей книгой «Windows PowerShell in Action». Очень рекомендую прочесть ее!

Не все части этой книги я писал сам, в частности, теоретические разделы взяты из содержимого, опубликованного в Microsoft MSDN или файла справки Windows PowerShell. Там вы найдете больше полезной информации, которую я рекомендую вам прочитать.

Если у вас имеются комментарии или советы по этой книге, напишите мне по электронной почте. Я могу быть не в состоянии ответить каждому лично, но буду очень признателен за любую конструктивную критику и похвалы: frankoch@microsoft.com

PowerShell – краткая шпаргалка

Отдельные команды

Для справки по любому командлету используйте `get-help`

`Get-Help Get-Service`

Чтобы получить список доступных командлетов, используйте `get-command`

`Get-Command`

Чтобы получить список всех свойств и методов объекта, используйте `get-member`

`Get-Service | Get-Member`

Установка политики безопасности

Просмотр и изменение политики безопасности с помощью `Get-ExecutionPolicy` и `Set-ExecutionPolicy`

`Get-ExecutionPolicy`

`Set-ExecutionPolicy remotesigned`

Выполнение сценария

`powershell.exe -noexit &"c:\myscript.ps1"`

Функции

Параметры разделяются пробелами.
`Return` необязателен.

```
function sum ([int]$a,[int]$b)
{
    return $a + $b
}
sum 4 5
```

Переменные

Должны начинаться с `$`

`$a = 32`

Могут иметь тип

`[int]$a = 32`

Массивы

Инициализация:

`$a = 1,2,4,8`

Запрос элемента:

`$b = $a[3]`

Константы

Создаются без `$`

`Set-Variable -name b -value 3.142 -option constant`

Используются с `$`

`$b`

Создание объектов

Создание экземпляра объекта COM

`New-Object -comobject <ProgID>`

```
$a = New-Object -comobject "wscript.network"
```

```
$a.username
```

Создание экземпляра объекта .Net Framework. При необходимости могут передаваться параметры

`New-Object -type <.Net Object>`

```
$d = New-Object -Type System.DateTime 2006,12,25
```

```
$d.get_DayOfWeek()
```

Пользовательский ввод

Для получения ввода от пользователя используйте `Read-Host`

```
$a = Read-Host "Enter your name"
```

```
Write-Host "Hello" $a
```

Запись в консоль

Имя переменной

```
$a
```

или

```
Write-Host $a -foregroundcolor "green"
```

Передача аргументов командной строки

Передаются в сценарий, разделяясь пробелами

```
myscript.ps1 server1 benp
```

Получаются в сценарии в массив `$args`

```
$servername = $args[0]
```

```
$username = $args[1]
```

Разное

Перенос строки ```

```
Get-Process | Select-Object `
name, ID
```

Комментарии `#`

`#` код отсюда не выполняется **Объединение строк** ;

```
$a=1;$b=3;$c=9
```

Передача выходных данных по каналу другой команде |

```
Get-Service | Get-Member
```

Цикл Do While

Повторяет набор команд, пока выполняется условие

```
$a=1  
Do {$a; $a++}  
While ($a -lt 10)
```

Цикл Do Until

Повторяет набор команд до выполнения условия

```
$a=1  
Do {$a; $a++}  
Until ($a -gt 10)
```

Цикл For

Повторяет одинаковые шаги определенное количество раз

```
For ($a=1; $a -le 10; $a++)  
{ $a }
```

Цикл ForEach

Цикл по коллекции объектов

```
Foreach ($i in Get-Childitem c:\windows)  
{ $i.name; $i.creationtime }
```

Оператор If

В зависимости от заданного условия выполняет тот или иной участок кода

```
$a = "white"  
if ($a -eq "red")  
{ "The colour is red" }  
elseif ($a -eq "white")  
{ "The colour is white" }  
else  
{ "Another colour" }
```

Оператор Switch

Другой способ выполнять определенный участок кода в зависимости от заданных условий

```
$a = "red"  
switch ($a)  
{  
"red" {"The colour is red"}  
"white" {"The colour is white"}  
default {"Another colour"}  
}
```

Запись в простой файл

Для простого текстового файла используйте Out-File или >

```
$a = "Hello world"  
$a | out-file test.txt
```

Или используйте > для вывода результатов работы сценария в простой текстовый файл

```
.\test.ps1 > test.txt
```

Чтение из файла

Используйте Get-Content для создания массива строк. Затем выполните цикл по массиву

```
$a = Get-Content "c:\servers.txt"  
foreach ($i in $a)  
{ $i }
```

Запись в файл HTML

Используйте ConvertTo-Html и >

```
$a = Get-Process  
$a | Convertto-Html -property Name,Path,Company > test.htm
```

Запись в файл CSV

Используйте Export-Csv и Select-Object для фильтрации выводимых данных

```
$a = Get-Process  
$a | Select-Object Name,Path,Company | Export-Csv -path test.csv
```