

PowerShell: часто задаваемые вопросы



Василий Гусев

Windows PowerShell всё настойчивее входит в жизнь системных администраторов. Если раньше можно было игнорировать «новую забаву Microsoft с пафосным названием», то сейчас это всё сложнее и сложнее.

Поддержка PowerShell стала обязательной (в разумных пределах) для будущих серверных продуктов Microsoft, да и другие компании не удержались от того, чтобы использовать это средство автоматизации в своих продуктах. Например, VMware готовит VI Toolkit – оснастку для автоматизации и управления VMware ESX и VMware VirtualCenter, а Intel выпустила набор командлетов для управ-

ления WebSphere MQ. И это лишь самые громкие имена. Давно стало понятно, что PowerShell останется надолго. И надо сказать – это здорово!

Являясь неимоверно мощным инструментом автоматизации, PowerShell еще и очень удобен и прост в изучении. Множество стандартных псевдонимов и умение прозрачно работать с классическими утилитами командной строки облегчат переход любите-

лям cmd.exe и других оболочек. Но тем не менее это совершенно другой язык и совсем другой подход к работе, поэтому у специалистов, начинающих осваивать PowerShell, неминуемо возникают вопросы. Я собрал вопросы, которые люди мне задавали, которые я встречал на просторах Интернета, ну и еще немножко добавил сам, из своего опыта работы. И в этой статье постараюсь дать на них ответы.

Что такое PowerShell?

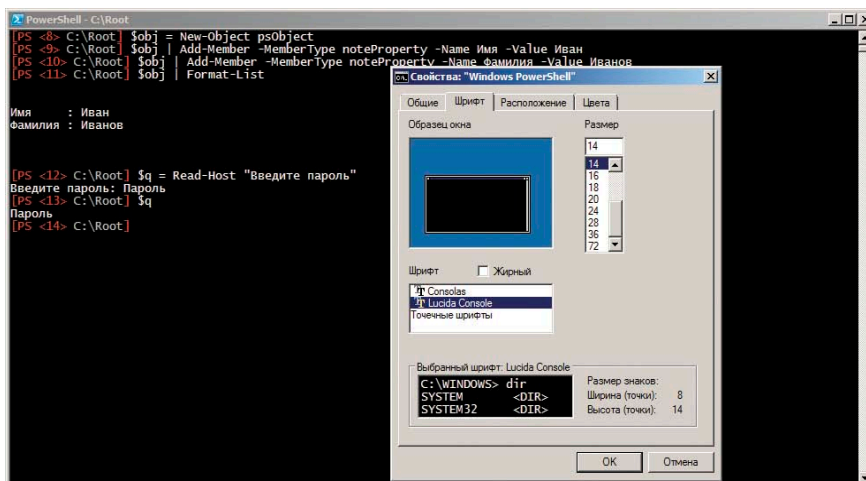
Традиционная формулировка звучит так: «язык сценариев и оболочка командной строки нового поколения». Ранее был известен под кодовым именем «Monad».

Зачем он мне нужен, когда есть VB Script?

Платформа Windows долгое время была отстающей в области командной строки и скриптов. Унаследованный от DOS command.com, превратившийся затем в cmd.exe, Visual Basic Script да изредка JavaScript и Perl. Вот и всё, что было у администраторов. Но решив завоевывать направление серверных операционных систем, специалисты Microsoft поняли, что без командной строки – успеха не видать. Сначала появилось огромное количество утилит, таких как netsh, diskpart, fsutil. Но практически все они обладают собственным синтаксисом, выводят информацию в различном виде, да и для работы с ними всё равно нужна полноценная оболочка. Да и ресурсы Com-объектов тоже хочется использовать с помощью чего-либо более удобного, чем VB Script. И для этого был создан PowerShell. Создавался он с нуля, без оглядок на совместимость (как cmd) и именно для системных администраторов (в отличие от VBS). PowerShell вобрал в себя лучшие элементы из множества языков, например Perl, C# или PHP, но при этом принес в мир командной строки такие вещи, как объектные конвейеры или возможность использования .Net Framework. А учитывая еще огромный интерес производителей программного обеспечения к PowerShell, становится понятно: учить PowerShell – нужно.

Стоит ли начинать изучать PowerShell сейчас? Может лучше подождать, когда выйдет версия 2.0?

Смысла ждать второй версии нет никакого. Она не изменит языка, лишь добавит ему новые возможности. Совместимость с первой версией является главным приоритетом разработчиков. Да и доступная на момент написания статьи версия – лишь CTP (Community Technology Preview), то есть даже до Beta-версии PowerShell 2.0 не дорос.



Русские символы в консоли PowerShell

А всё-таки, что будет нового в PowerShell 2.0?

Некоторые нововведения из тех, на которые уже сейчас можно посмотреть в CTP:

- **Удаленное выполнение команд (PowerShell Remoting)** – использование технологии WinRM для выполнения команд PowerShell на одном или множестве удаленных компьютеров, параллельно или последовательно. Ну и конечно, возможность отслеживать их состояние и получать результаты выполнения.
- **Отладчик в консоли** – к командлету Set-PSDebug добавятся новые: Enable-PSBreakPoint, Disable-PSBreakPoint, Get-PSBreakPoint, Invoke-PSBreakPoint, New-PSBreakPoint, Remove-PSBreakPoint, позволяющие работать с отладочными «точками останова» в скриптах. Да так хорошо, что могут позавидовать многие продвинутые языки программирования. И всё это из командной строки.
- **Улучшения работы с WMI** – новые командлеты Invoke-WmiMethod, Remove-WmiObject, Set-WmiInstance, ну и некоторые улучшения в работе Get-WmiObject.
- **Фоновые работы (Background Jobs)** – хотя их можно использовать уже в 1.0, официальная версия не помешает. Это возможность выполнять команды или целые скрипты в фоне, параллельно основной работе.
- **Графический Host** – вариант графической оболочки от авторов языка.

- **Улучшения в командлете Select-String** – новый командлет Out-Grid, позволяющий выводить данные в окне с таблицей. Операторы -Join, -Split для облегчения работы с текстовыми строками.
- **ScriptCmdlets** – возможность создания командлетов только с помощью кода PowerShell, без применения C# или Visual Basic .Net

И многое другое...

На каких операционных системах можно использовать PowerShell?

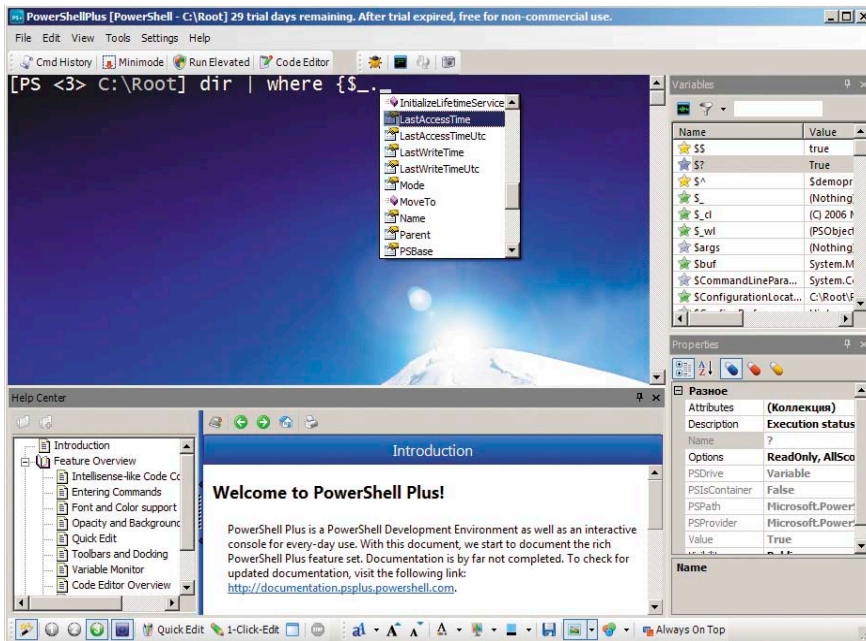
Windows XP SP 2, Windows Server 2003, Windows Vista, Windows Server 2008. Хотя и заявлено, что PowerShell не поддерживается на Home-версиях Windows, он на них прекрасно работает. А вот владельцам Windows 2000 – не повезло, PowerShell на этой системе (и на всех предыдущих) работать не будет.

Бесплатен ли PowerShell? Как его можно получить?

PowerShell бесплатен. Скачать его можно на страничке <http://www.microsoft.com/powershell/download>. Хочу обратить внимание, что PowerShell (вместе со всей встроенной и прилагающейся документацией) доступен на русском языке. Получить его можно, скачав Localized Package.

На этой страничке нет версии для Windows 2008. Что делать?

PowerShell входит в стандартную поставку Windows 2008, то есть скачивать его не нужно. Установить его можно с помощью Server Manager, выбрав «Add Features».



Альтернативная оболочка для PowerShell – PowerShell Plus

А можно ли использовать PowerShell в режиме установки Windows 2008 Server Core?

К сожалению, пока нет. Это связано с отсутствием на Server Core .Net Framework необходимого PowerShell для работы. Появится он там не раньше следующего релиза. Будет ли это Windows 2008 Release 2 или вообще следующая версия – неизвестно.

В каких продуктах используется или будет использоваться PowerShell?

Exchange Server 2007, SC Operations Manager, SC Virtual Machine Manager, SQL Server 2008, SC Data Protection Manager 2007, Windows Compute Cluster Server 2007, Quest Management Shell for Active Directory, Special Operations Software Specops Command, IBM WebSphere MQ, VMWare VI Toolkit, Quest ActiveRoles Server и множество других.

У меня в консоли PowerShell проблемы с отображением русских символов, что делать?

Сам по себе PowerShell прекрасно поддерживает Unicode, чего не скажешь об оболочке командной строки Windows, доставшейся от cmd.exe. Но и это можно поправить. Для начала удостоверьтесь, что в региональных настройках, в опции «Язык для программ, не поддерживающих Unicode», установлен русский язык. Если и это не помогает – попробуйте сменить

шрифт, использующийся в консоли, например на Lucida Console.

Не хочу работать в этом ужасном черном окне. Хочу прозрачное окно и прочие «блага цивилизации». Куда податься?

На самом деле PowerShell.exe – это лишь простенький интерфейс для языка, так называемый Хост. И его легко можно сменить на альтернативу, которых уже немало написано. Например, ошарашивающий своими возможностями PowerShell Plus (бесплатен для некоммерческого использования) или PowerShell Analyzer, предоставляющий интерфейс, аналогичный SQL Analyzer. Оба доступны для скачивания на странице <http://www.powershell.com>. PoshConsole – хост, основанный на Windows Presentation

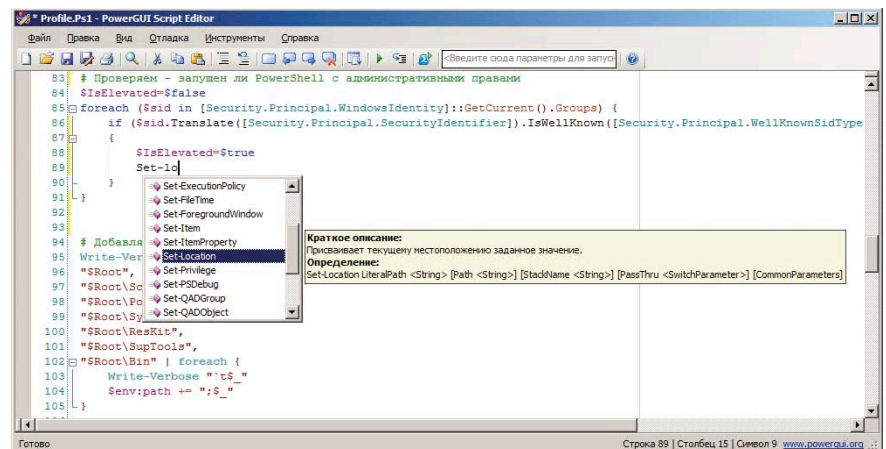
Foundation, можно скачать тут – <http://www.codeplex.com/PoshConsole>. А фанаты FAR наверняка найдут интересной возможность использовать PowerShell в любимой программе – <http://code.google.com/p/farnet/downloads/list>. Ну и программисты тоже не останутся обделенными – VS Command Shell позволяет выполнять команды PowerShell из Visual Studio – <http://www.codeplex.com/VSCmdShell>.

А как насчет альтернативы блокноту для написания скриптов?

Для начала – неплохие редакторы встроены в вышеупомянутые PowerShell Plus и PowerShell Analyzer. Известная скриптерам компания Sapien добавила поддержку PowerShell в свою среду разработки Primal Script (<http://www.primalscript.com>), и выпустила PrimalScope – <http://www.primalscope.com>. Ну а сам я пользуюсь шустреньким редактором PowerGUI с прекрасным автоматическим дополнением команд и параметров PowerShell и объектов .Net. Кроме того, он обладает превосходным отладчиком и множеством других преимуществ. Кстати, большинство его разработчиков – наши соотечественники.

Как выполнить команду на другом компьютере?

К сожалению, PowerShell 1.0 не поддерживает возможность выполнения команд на другом компьютере. Хотя в большинстве случаев дистанционная работа и возможна благодаря WMI, .Net, иногда хочется именно выполнить команду PowerShell на другой системе. А еще здорово бы получить результаты выполнения в виде объектов... На се-



Редактирование профиля PowerShell с помощью PowerGUI Script Editor

годняшний день существует 3 выхода из этого положения: первый это подождать PowerShell 2.0. Там «Remoting» будет на высоте.

Но зачем ждать, если у нас есть возможности PowerShell? Уже сейчас доступна функция Invoke-RemoteCommand, использующая WinRS (как и PowerShell 2.0) для выполнения команд: <http://blogs.msdn.com/powershell/archive/2008/02/29/remoting-using-powershell-v1.aspx>.

Ну и, кроме того, можно сделать скрипт Invoke-RemoteExpression использующий утилиту PsExec.exe от SysInternals: <http://www.leeholmes.com/blog/UsingPowerShellAndPsExecToInvokeExpressionsOnRemoteComputers.aspx>.

Всё здорово, но какие минусы у PowerShell?

Их, к счастью, немного, но они всё же есть.

■ **PowerShell не работает под Windows 2000.** Конечно, он не работает и под Windows 3.11 или 95, но только Windows 2000 пока еще достаточно популярен. Впрочем, конечно, можно использовать WMI для удаленной работы с этой версией.

■ **PowerShell не отличается большой скоростью выполнения.** С этим трудно спорить, даже разработчики это признают. Но для скриптового языка это не является таким уж большим приоритетом – никто не будет писать на PowerShell системы обработки огромных массивов данных. Для этого есть SQL и множество специализированных программ. Зато PowerShell позволяет очень быстро создавать код для автоматизации задач и вследствие этого выигрывать огромное количество времени.

■ **PowerShell требует установки.** Во всяком случае на Windows XP, Vista и 2003. В Windows 2008 и последующих версиях он будет встроенным компонентом системы, но пока это является значимым недостатком. Хотя стоит отметить, что все стандартные методы автоматизации установки работают. Дистрибутив PowerShell для Windows XP и Windows 2003 понимает ключи

/quiet /passive и /norestart, а пакет .msu для Windows Vista вообще можно легко интегрировать в дистрибутив как компонент операционной системы.

■ **Учитывая предыдущие два пункта, можно сказать, что PowerShell пока не слишком хорош для login-или startup-скриптов.** Во-первых, его сначала необходимо установить на все системы, на которых предполагается выполнение, и, во-вторых, пользователям, возможно, придется подождать на несколько секунд больше, чем если бы сценарий входа был сделан на VBScript. Но если очень хочется, то всё возможно: вышеупомянутый SpecOps Command позволяет использовать PowerShell в сценариях входа и загрузки, автоматически устанавливая его на компьютеры при необходимости, а также проводить предварительное тестирование скриптов и получать отчеты о результатах их выполнения. Подробнее можно узнать тут – <http://www.specopssoft.com/powershell>.

Итак теоретическая часть вопросов позади. Самые нетерпеливые читатели уже установили PowerShell, прочитали вводную статью Андрея Бирюкова в №11 за 2007 год, и, выполнив команду «Set-ExecutionPolicy RemoteSigned» (для разрешения выполнения скриптов), приступили к изучению. Теперь начинают возникать другие вопросы – практические.

Я нашел в Интернете скрипт PowerShell, который начинается со слова function, как мне его применить?

Обычно можно просто скопировать тело скрипта в буфер обмена и вставить в окно консоли, нажав затем пару раз <Enter>. После этого можно будет применять эту функцию почти как командлет. Но если вы собираетесь применять его неоднократно – лучше поместить его к себе в файл профиля, чтобы он подгружался автоматически. Либо сделать отдельный файл, например MyFunctions.ps1, и уже его подгружать в профиле, например такой командой:

```
c:\Scripts\MyFunctions.ps1
```

Обратите внимание на точку перед вызовом скрипта. Она указывает PowerShell, что его необходимо выполнить в текущем окружении и сохранить все объявленные в нём функции и переменные.

Я нашел скрипт, который начинается со слова Param, что делать в этом случае?

Предполагается, что вы поместите этот скрипт в файл .ps1 и будете вызывать следующим образом:

```
MyScript.ps1
```

Если скрипт находится в каталоге, который не содержится в переменной окружения Path, то необходимо указать полный путь к нему. Но если он расположен в текущем каталоге PowerShell, то можно сделать так:

```
.\MyScript.ps1
```

Точка в данном случае обозначает текущий каталог.

Как использовать переменные окружения?

В PowerShell переменные окружения представлены на специальном виртуальном диске – Env:. То есть теперь можно просто перейти на этот диск с помощью cd Env: и посмотреть переменные окружения, используя команду Dir (ну или Ls, кому как нравится – всё равно это псевдонимы для Get-ChildItem). Если же нужно использовать переменную окружения, применяется следующая конструкция:

```
# Выводим имя компьютера на экран
$env:computername
#переходим в каталог Windows
cd $env:windir
"Welcome to " + $env:computername + "!"
```

Более подробно можно прочитать во встроенной справке выполнив команду:

```
Get-Help About_Environment_variable
```

Как из результатов выполнения команды выбрать только объекты, удовлетворяющие определенным условиям?

Так как в PowerShell используются объекты, уже не нужно разбирать строки,

Скрипт PowerTab дополняет свойства объектов не хуже Visual Studio

выдаваемые командой, на части. Достаточно просто работать со свойствами объектов.

Например, для отбора объектов можно использовать командлет Where-Object или его псевдонимы – Where и вопросительный знак.

Выбираем из текущего каталога только файлы с размером более двух килобайт:

```
dir | where {$_.length -ge 2kb}
```

В качестве первого аргумента команды Where-Object используется скриптовый блок, если выражение в нём истинно, то объект передается по конвейеру дальше. Внутри блока, переменная \$_ представляет текущий объект, и мы можем сравнить его по какому-то критерию с чем-то еще, или произвести любые другие вычисления. В данном случае свойство файла Length (размер файла в байтах) с помощью оператора -ge (больше или равно) сравнивается с значением 2kb (kb – встроенный множитель, равный 1024).

```
Get-WmiObject win32_share | where {$_.path -like "?:\\"}
```

А здесь, получив из WMI список объектов win32_share, мы отбираем лишь те из них, у которых свойство Path подпадает под указанную маску.

Аналогичным образом получаем все процессы, запущенные не из папки Windows и её подпапок, и будто бы завершаем их. Если убрать ключ – whatif, то они будут завершены на самом деле.

```
ps | where {$_.path -notlike "c:\windows*"} | kill -whatif
```

В этом случае мы используем псевдоним ps для Get-Process и kill для Stop-Process. Но стоп, если просто вы-

полнить команду Get-Process, то мы не увидим свойства Path! И возникает следующий вопрос:

Как узнать какие свойства есть у объекта?

Третья, самая полезная команда в PowerShell после Get-Command и Get-Help, это Get-Member. Перенаправив в неё объект (или их коллекцию), можно увидеть свойства и методы, которыми эти объекты обладают. Например, выполнив следующую команду, можно увидеть все свойства объектов типа System.Diagnostics.Process (именно такие нам возвращает команда Get-Process).

```
Get-Process | Get-Member -MemberType *property
```

Есть и более наглядный способ посмотреть свойства объекта:

```
$e = Get-Process explorer
$e | Format-List -Property *
```

Тут мы сначала помещаем в переменную \$e объект, представляющий процесс Explorer, а затем выводим его на экран, форматируя в виде списка с помощью командлета Format-List. Для Format-List можно указать список свойств, которые нужно получить, и в нашем случае это простая маска – *. Короче говоря, все свойства. Можно применять псевдоним и просто опустить имя параметра – так будет значительно короче.

```
$e | fl *
```

Упомяну еще о третьем способе. Когда вы наберёте \$e и нажмёте клавишу табуляции, – будут перебираться все свойства и методы этого объекта. Листать их в обратную сторону можно сочетанием <Alt> + <Tab>.

Разумеется, таким же образом

можно исследовать и другие объекты: .Net, WMI, COM, ADSI или любые другие.

Как из текстового файла выбрать строки, содержащие определенный текст? И как насчет регулярных выражений?

Уж не знаю, почему разработчики не сделали для командлета Select-String псевдоним grep, но ничего не мешает сделать его самостоятельно:

```
New-Alias grep Select-String
```

Впрочем, точной копией grep это всё равно не будет, так что я предпочитаю еще более короткий вариант:

```
New-Alias ss Select-String
```

Ну и дальше анализируем файлы:

```
Select-String 3389 C:\Windows\System32\drivers\etc\services
```

Эта команда вернёт две строки, в следующем формате: «имя_файла: номер_строки:текст».

Если же у вас задача получить только текст, – не надо пытаться парсить эти строки! Командлет Select-String тоже возвращает объекты, и достаточно лишь выбрать нужное свойство, в данном случае это Line.

```
ss 3389 C:\Windows\System32\drivers\etc\services |%{$_.Line}
```

Тут я использовал % – это псевдоним для командлета ForEach-Object. И для каждого объекта получаем его свойство Line.

Я показал лишь маленькую толику возможностей Select-String. Этот командлет работает с регулярными выражениями и может обрабатывать несколько файлов сразу. Можно, например, перенаправить на него вывод команды Dir:

```
dir c:\windows\*.log | Select-String "Error"
```

Раз уж заговорили о регулярных выражениях, как еще можно с ними работать?

Следующий абзац для тех, кто пока не знаком с регулярными выражениями. Вы наверняка не раз использовали маски «*» или «?» для обозначения нес-

колько похожих имен файлов и других подобных задач в командной строке. Регулярные выражения – это, можно сказать, продолжение идеи. Хотя их синтаксис гораздо сложнее, взамен они позволяют описывать практически любые условия, вытаскивать из строк необходимые подстроки и еще много полезных вещей. Подробно на синтаксисе я останавливаться не буду, даже его краткий обзор – это тема для отдельной статьи.

Кроме командлета Select-String, нам доступен еще оператор -match. Он позволяет проверить – подпадает ли строка под регулярное выражение и возвращает соответственно \$True или \$False в качестве результата.

Например, следующая команда вернёт \$True.

```
"xaegr@yandex.ru" -match "\S@\S+"
```

На всякий случай поясню: «\S+» означает «любой символ, кроме пробела («\S») в количестве от 1 и более («+»)». Ну а символ @ подразумевает сам себя.

Конечно, можно использовать еще и «группы захвата», чтобы разбивать строку на компоненты. Эти группы создаются путем помещения отдельных элементов регулярного выражения в круглые скобки. После выполнения оператора -match группы помещаются в специальную переменную \$matches.

```
if ("xaegr@yandex.ru" -match "\S+@\S+") {$matches[2]}
```

Здесь будет возвращено содержимое второй группы – домен почтового адреса.

Поддерживаются практически все возможности регулярных выражений .Net.

Так, например, используются именованные группы:

```
"E-mail: xaegr@yandex.ru" -match "(?<Имя>\S+) (?<Домен>\S+)"
```

Теперь можно получить значения так:

```
$matches["Домен"]
```

Или даже так:

```
$matches.Имя
```

Еще один оператор, работающий с регулярными выражениями, это -replace. С его помощью легко заменить текст, используя все те же регулярные выражения. В качестве первого операнда указывается обрабатываемая строка (или массив строк), а в качестве второго – массив из двух элементов.

Первый из них – искомое выражение, а второй – то, на что будет произведена замена.

Так, например, следующая команда возвратит строку «PowerShell»:

```
"SimpleShell" -replace "Simple", "Power"
```

Если второй элемент массива не указывать, то весь текст, подпадающий под выражение, будет удален.

```
"PowerShell" -replace "[wrel]"
```

Эта команда удалит из строки все символы w, r, e и l. В результате мы получим «Posh». В -replace тоже можно применять группы захвата:

```
"PowerShell" -replace "(. {5}) (. {5})", '$2$1'
```

Эта строчка вернёт нам значение «ShellPower».

Кстати, я не просто так поместил последнюю строчку в одинарные кавычки. Дело в том, что символ \$ используется в PowerShell для обозначения переменных, и в строке, окруженной двойными кавычками, он попытается заменить \$1 и \$2 значениями соответствующих переменных.

В случае же, если всех вышеперечисленных методов недостаточно, можно использовать класс System.Text.RegularExpressions.Regex из .Net Framework. Создать его экземпляр можно, например, таким образом:

```
[regex]$r = "[,;]"
```

А затем посмотреть все его методы с помощью команды Get-Member:

```
$r | Get-Member -MemberType method
```

Так только с помощью System.Text.RegularExpressions.Regex можно применить метод Split:

```
$r.Split("1;2,3,4,5;6")
```

Как заставить PowerShell сравнивать строки (-eq, -match, -like) учитывая регистр символов?

Все операторы, работающие со строками, имеют версии, чувствительные к регистру символов. Отличаются они лишь приставленной спереди буквой «с» (от Case sensitive).

Так, команда:

```
"Test" -eq "test"
```

вернёт \$True, а вот:

```
"Test" -ceq "test"
```

уже \$False. Кстати, то же самое правило относится и к операторам -match и -replace.

Как посмотреть что это за команды – «%», «gps» или «ft»?

Достаточно воспользоваться командлетом Get-Command:

```
Get-Command %, gps, ft
```

Выполнив эту команду, мы увидим, что вышеперечисленные символы – псевдонимы (Alias) для командлетов Foreach-Object, Get-Process и Format-Table соответственно.

В качестве аргумента может быть не только командлет или алиас PowerShell. В данном случае мы узнаем полный путь к исполняемому файлу ping.exe.

```
Get-Command ping
```

Как изменить приглашение командной строки?

Для этого достаточно переопределить функцию Prompt, например, получить приглашение в формате пользователь@компьютер:каталог> можно таким образом:

```
# Название функции
function prompt {
# Бонус! Выводим полный путь
# в заголовок окна
$host.UI.RawUI.WindowTitle = "
    Get-Location ;
# Получаем из переменной окружения
# имя пользователя
$env:username + "@" +
# Имя компьютера
$env:computername + ":" +
# И последний элемент текущего
# каталога
(get-location | split-path -leaf) .\
+ "> "}
```

Безусловно, выполнять такой код каждый раз при запуске PowerShell не очень то интересно, поэтому возникает следующий вопрос:

Как создать/изменить профиль?

На всякий случай уточню – профиль PowerShell – это файл скрипта, который выполняется автоматически при каждом запуске консоли PowerShell. Изначально при установке PowerShell не создает его, но это легко исправить следующей командой:

```
New-Item -type file -Path $PROFILE -Force
```

Это создаст пустой файл профиля. Открыть его проще всего так:

```
notepad $PROFILE
```

Ну и дальше всё в ваших руках. Тут да можно поместить команды для загрузки необходимых оснасток, объявления функций и псевдонимов и т. п.

Понял, что деваться некуда, хочу учиться. Какая литература есть по PowerShell?

Литературы множество, но, к сожалению, пока очень мало что доступно на русском языке. Правда, кое-что всё же есть.

- Прилагающаяся документация переведена на несколько языков, включая русский! Её можно получить, либо установив локализованный дистрибутив PowerShell (меню «Пуск → Все программы → Windows PowerShell»), либо скачав отдельно WindowsPowerShell_Localized_DocumentationPack.zip со страницы <http://www.microsoft.com/downloads/details.aspx?FamilyID=b4720b00-9a66-430f-bd56-ec48bfca154f>.
- Если вы установили локализованный PowerShell, то у вас будет не только русская документация, но и встроенная справка. Её можно посмотреть с помощью команды Get-Help или Man – кому как привычнее.
- Статья «Знакомимся с PowerShell – новой командной оболочкой» Андрея Бирюкова в журнале за ноябрь 2007 года. Также она доступна в Интернете по адресу <http://www.samag.ru/cgi-bin/go.pl?q=articles;n=11.2007;a=01>.

■ Русскоязычные блоги (с PowerShell так сложилось, что дневники разработчиков и просто энтузиастов являются бесценным источником информации о языке. Ну и, кроме того, там можно найти множество примеров скриптов. На русском языке мне пока известны следующие активные блоги:

- ☑ <http://xaegr.wordpress.com> – блог вашего покорного слуги;
- ☑ <http://blogs.technet.com> – журнал Андрея Бешкова, специалиста по инфраструктурным решениям Microsoft;
- ☑ <http://www.itcommunity.ru/blogs/dmitrysotnikov/default.aspx> – русскоязычный дневник Дмитрия Сотникова, работника Quest Software, одного из создателей PowerGUI и AD Cmdlets.

Если же английский язык не является преградой, то объем доступной литературы сильно увеличивается. В частности, по PowerShell уже выпущено огромное количество книг. Перечислять их все не имеет смысла – я уверен, что все читатели журнала сумеют воспользоваться поиском по слову PowerShell. Я лишь отмечу те, что прочитал сам и рекомендую другим:

- PowerShell in Action – это книга от одного из разработчиков PowerShell Брюса Пэйета (Bruce Payette). Превосходная книга не только для начинающих, но и для профи. В первой части постепенно раскрываются основы языка, но при этом разъясняется не только «Как оно работает?» но и «Почему оно работает так?». Во второй же части книги следуют более сложные темы с примерами реальных задач.
- PowerShell Cookbook от O'Reilly. Эта книга прекрасно дополняет PowerShell in Action, используя совершенно другой подход. Практически вся книга составлена из пар задача – решение, что превращает её в превосходный сборник рецептов.

Еще стоит упомянуть пару книг от работника немецкого подразделения Microsoft – Frank Koch. Эти книги доступны на английском и немецком языках для свободного скачивания в Интернете.

■ PowerShell course book – <https://blogs.technet.com/chitpro-de/archive/2007/05/10/english-version-of-windows-powershell-course-book-available-for-download.aspx>.

■ Administrative tasks using Windows PowerShell – <https://blogs.technet.com/chitpro-de/archive/2008/02/28/free-windows-powershell-workbook-server-administration.aspx>.

Англоязычные блоги. Стоит посмотреть, даже если вам не очень легко воспринимать английский язык, ведь PowerShell зачастую выполняет ту задачу, для которой был придуман эсперанто.

- <http://blogs.msdn.com/powershell> – официальный блог PowerShell. Разработчики пишут сюда интересные новости о PowerShell, занимательный код и сообщают об интересных событиях.
- <http://thepowershellguy.com> – блог /\o/\, одного из Гигу PowerShell, автора PowerTab и PowerShell WmiExplorer. Тут можно почитать много интересного о работе с ADSI, WMI и посмотреть на множество классных скриптов.
- <http://dmitrysotnikov.wordpress.com> – англоязычный блог Дмитрия Сотникова.
- <http://www.leeholmes.com/blog/> – дневник Lee Holmes. Один из разработчиков PowerShell, а также автор PowerShell Cookbook и системы автоматического кормления кошки с помощью PowerShell скрипта.

Ну и, конечно, стоит заглянуть на официальную страницу – <http://microsoft.com/powershell> и в Script Center – <http://www.microsoft.com/technet/scriptcenter>, где можно найти, например, руководство по конвертации скриптов из VBScript – <http://www.microsoft.com/technet/scriptcenter/topics/winpshtconvert>.

Кстати, все эти ссылки (и еще много других) доступны на моей страничке, где я их собираю, – <http://windowspowershell.ru> или <http://xaegr.wordpress.com/poshlinks>.

А у меня остались еще вопросы, кому их можно задать?

Можно задавать их мне на e-mail xaegr@yandex.ru, буду рад ответить! 