Что такое Windows PowerShell и с чем его едят? Часть 1: основные возможности

* [Блог компании RUVDS.com](https://habr.com/ru/company/ruvds/),
* [Системное администрирование](https://habr.com/ru/hub/sys_admin/),
* [PowerShell](https://habr.com/ru/hub/powershell/)

*Исторически утилиты командной строки в Unix-системах развиты лучше чем в Windows, однако с появлением нового решения ситуация изменилась.*  
  
Windows PowerShell позволяет системным администраторам автоматизировать большинство рутинных задач. С ее помощью можно менять настройки, останавливать и запускать сервисы, а также производить обслуживание большинства установленных приложений. Воспринимать синее окошко как еще один интерпретатор команд было бы неправильно. Такой подход не отражает сути предложенных корпорацией Microsoft инноваций. На самом деле возможности Windows PowerShell гораздо шире: в небольшом цикле статей мы попробуем разобраться, чем решение Microsoft отличается от более привычных нам средств.  
  
[](https://habr.com/ru/company/ruvds/blog/487876/)

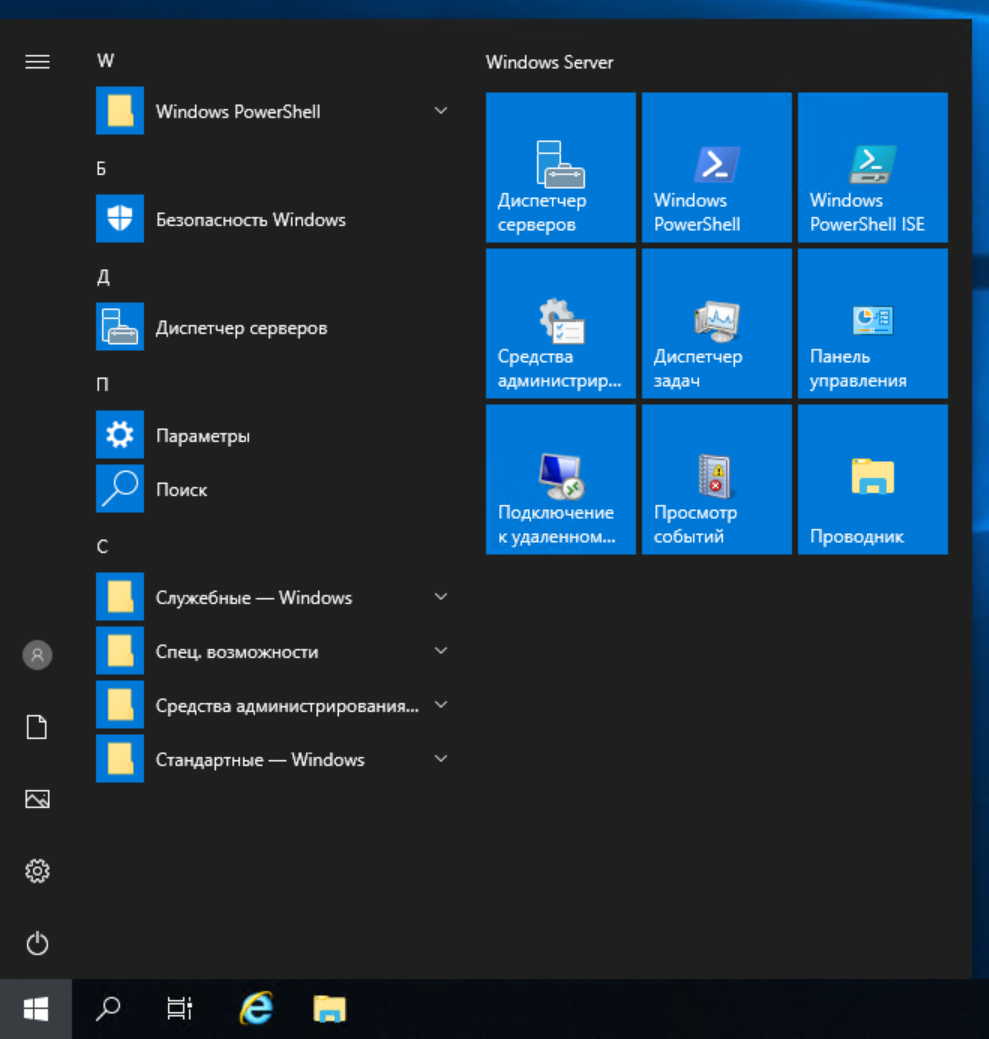
Основные возможности

Разумеется Windows PowerShell — это в первую очередь командная оболочка с языком сценариев, изначально созданная на основе платформы .NET Framework, а позднее — на .NET Core. В отличие от принимающих и возвращающих текстовые данные оболочек, Windows PowerShell работает с классами .NET, у которых есть свойства и методы. PowerShell позволяет выполнять обычные команды, а также дает доступ к объектам COM, WMI и ADSI. В ней используются различные хранилища, вроде файловой системы или реестра Windows, для доступа к которым созданы т.н. поставщики (providers). Стоит отметить возможность встраивания исполняемых компонентов PowerShell в другие приложения для реализации различных операций, в т.ч. через графический интерфейс. Верно и обратное: многие приложения для Windows предоставляют доступ к своим интерфейсам управления через PowerShell.   
  
Windows PowerShell позволяет:

* Менять настройки операционной системы;
* Управлять службами и процессами;
* Настраивать роли и компоненты сервера;
* Устанавливать программное обеспечение;
* Управлять установленным ПО через специальные интерфейсы;
* Встраивать исполняемые компоненты в сторонние программы;
* Создавать сценарии для автоматизации задач администрирования;
* Работать с файловой системой, реестром Windows, хранилищем сертификатов и т.д.

Оболочка и среда разработки

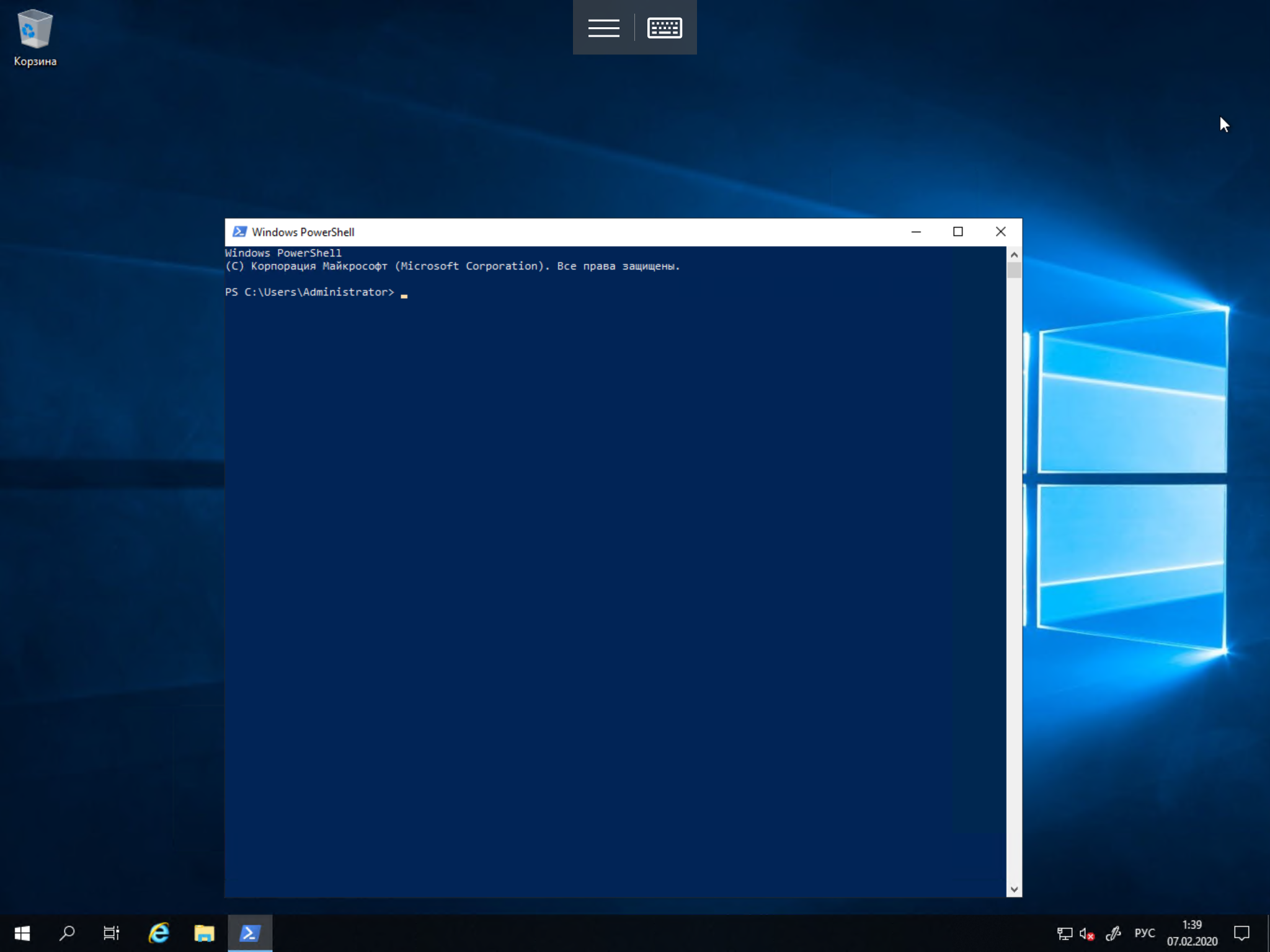
Существует Windows PowerShell в двух ипостасях: помимо эмулятора консоли с командной оболочкой есть интегрированная среда сценариев (Integrated Scripting Environment — ISE). Чтобы получить доступ к интерфейсу командной строки достаточно выбрать соответствующий ярлык в меню Windows или запустить powershell.exe из меню «Выполнить». На экране появится синее окошко, заметно отличающееся по возможностям от допотопного cmd.exe. Там есть автодополнение и другие фишки, привычные пользователям командных оболочек для Unix-систем.



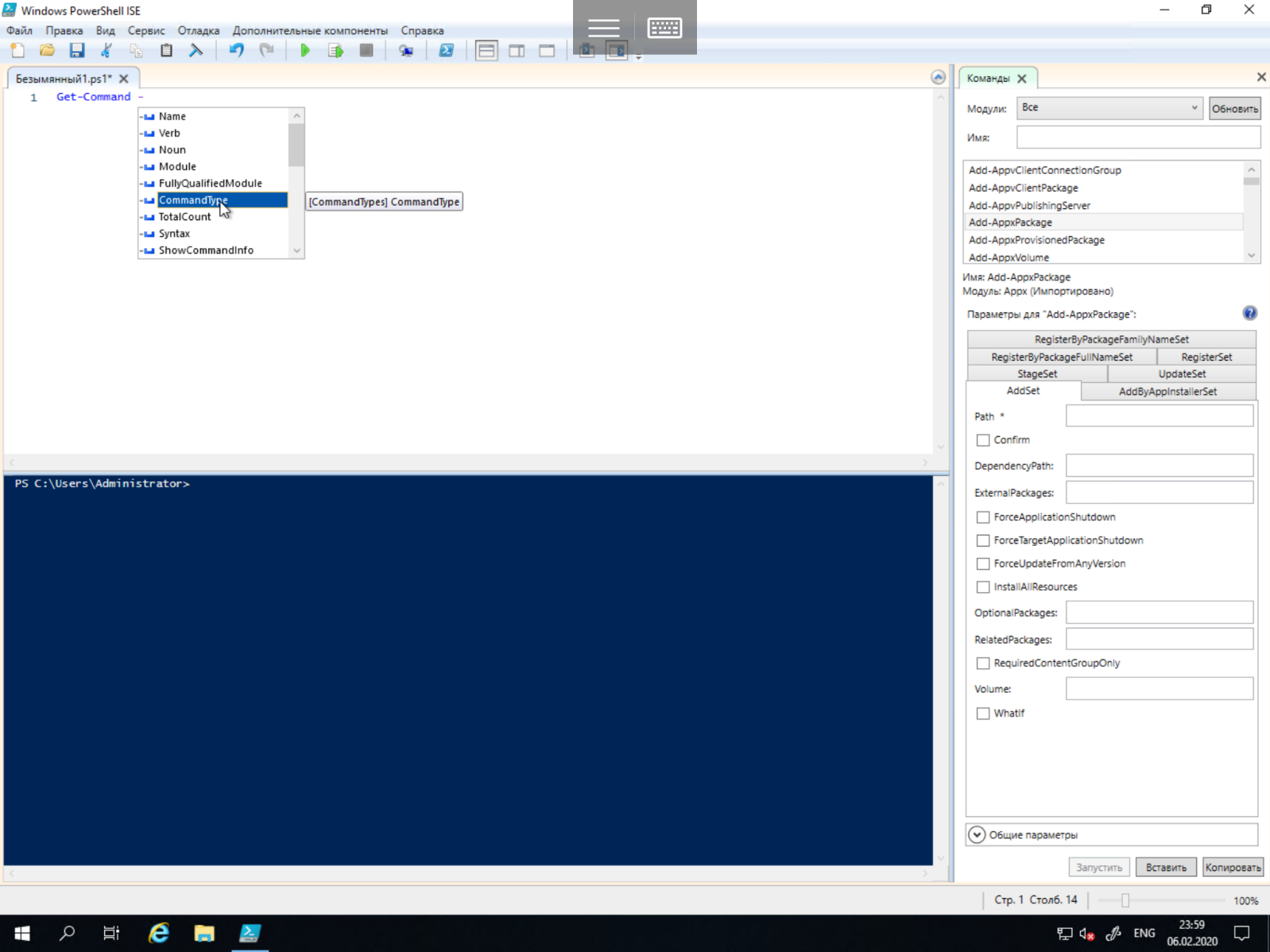
Для работы с оболочкой нужно запомнить некоторые сочетания клавиш:

* Стрелки вверх и вниз прокручивают историю для повтора ранее набранных команд;
* Стрелка вправо в конце строки перенабирает предыдущую команду посимвольно;
* Ctrl+Home удаляет набранный текст от позиции курсора до начала строки;
* Ctrl+End удаляет текст от курсора до конца строки.

F7 показывает окно с набранными командами и позволяет выбрать одну из них. В консоли также работает выделение текста мышью, копипаст, позиционирование курсора, удаление, backspace — все как мы любим.



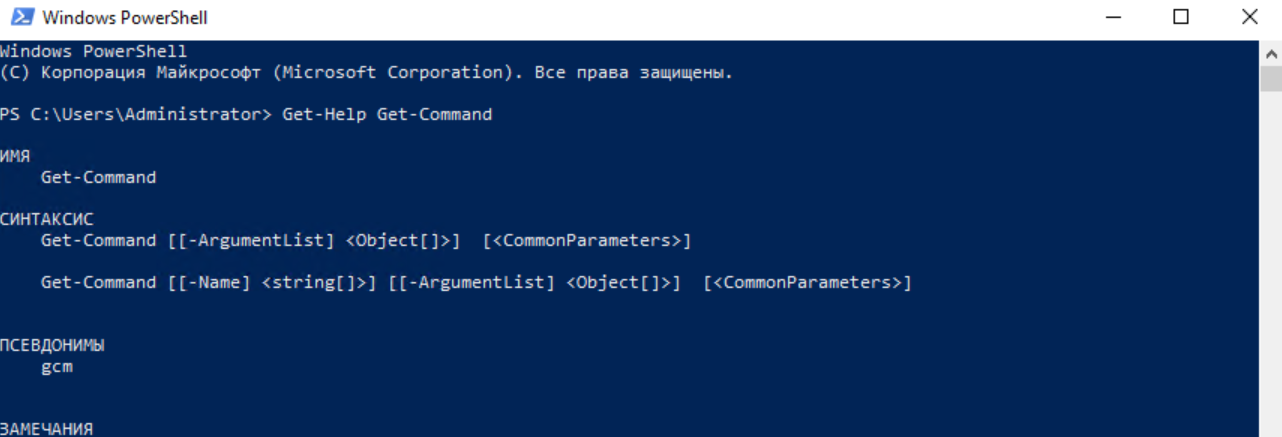
Windows PowerShell ISE является полноценной средой разработки с поддерживающим вкладки и подсветку синтаксиса редактором кода, конструктором команд, встроенным отладчиком и другими программистскими радостями. Если в редакторе среды разработки после имени команды написать знак дефис, вы получите в выпадающем списке все доступные параметры с указанием типа. Запустить PowerShell ISE можно либо через ярлык из системного меню, либо с помощью исполняемого файла powershell\_ise.exe.



Командлеты

В Windows PowerShell появились т.н. командлеты (cmdlets). Это специализированные классы .NET, в которые заложена разнообразная функциональность. Именуются они по принципу «Действие-Объект» (или «Глагол-Существительное, если вам так больше нравится), а разделенная дефисом связка напоминает сказуемое и подлежащее в предложениях естественных языков. Например, Get-Help буквально означает «Получить-Помощь» или в контексте PowerShell: «Показать-Справку». По сути это аналог команды man в Unix-системах и мануалы в PowerShell нужно запрашивать именно так, а не вызывая командлеты с ключом --help или /?.. Не стоит забывать и об онлайн-документации по PowerShell: у Microsoft она достаточно подробная.  
  
Помимо Get в командлетах для обозначения действий используются и другие глаголы (и не только глаголы, строго говоря). В списке ниже мы приведем несколько примеров:  
  
Add — добавить;  
Clear — очистить;  
Enable — включить;  
Disable — выключить;  
New — создать;  
Remove — удалить;  
Set — задать;  
Start — запустить;  
Stop — остановить;  
Export — экспортировать;  
Import — импортировать.  
  
Есть системные, пользовательские и опциональные командлеты: в результате выполнения все они возвращают объект или массив объектов. К регистру они не чувствительны, т.е. с точки зрения интерпретатора команд нет разницы между Get-Help и get-help. Для разделения используется символ ‘;’, но ставить его обязательно только если в одной строке выполняется несколько командлетов.   
  
Командлеты Windows PowerShell группируются в модули (NetTCPIP, Hyper-V и т.д.), а для поиска по объекту и действию существует командлет Get-Command. Показать справку по нему можно так:

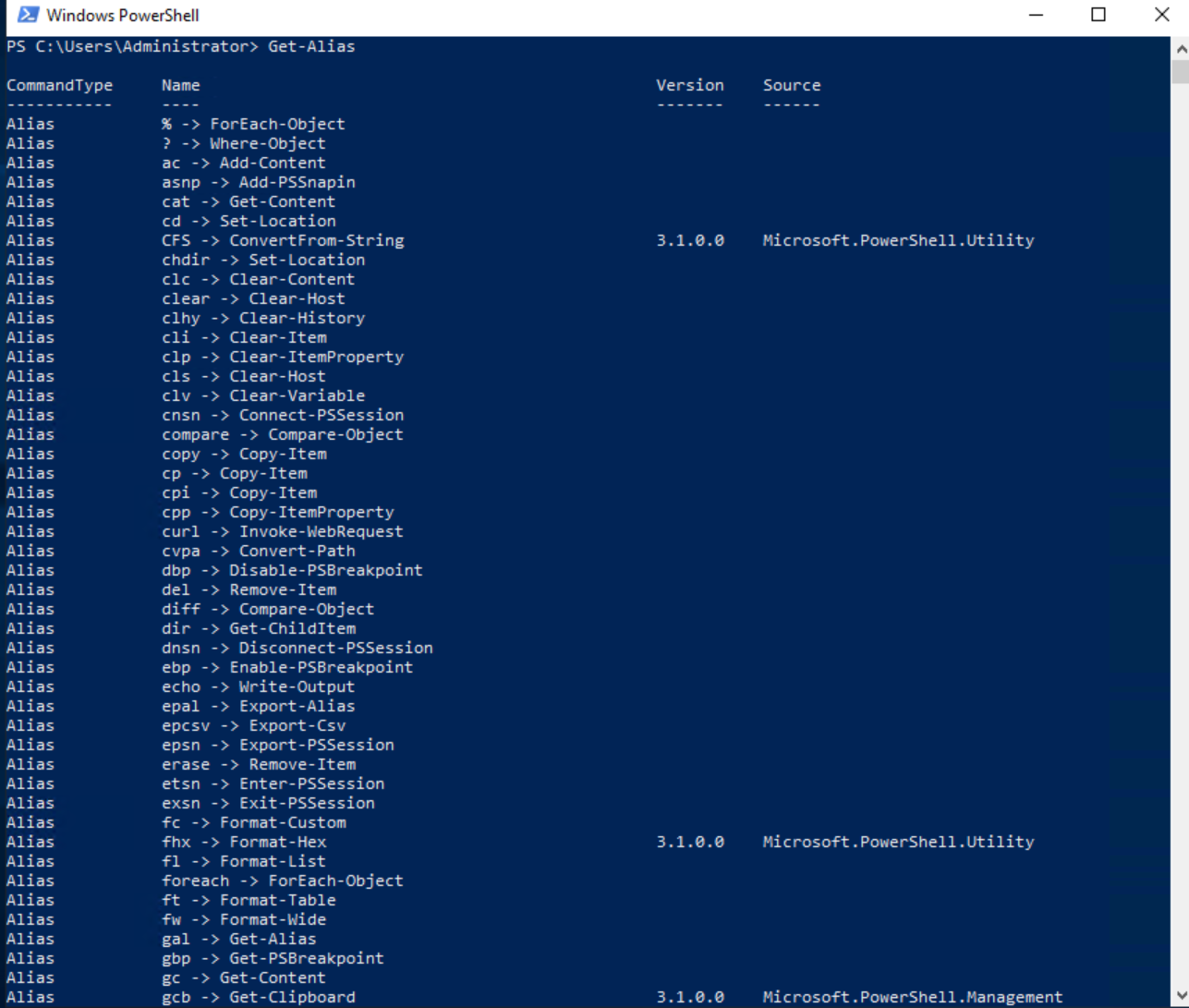
Get-Help Get-Command



По умолчанию команда отображает краткую справку, но в командлеты при необходимости передаются параметры (аргументы). С их помощью можно, например, получить детальную (параметр -Detailed) или полную (параметр -Full) справку, а также вывести на экран примеры (параметр -Examples):

Get-Help Get-Command -Examples

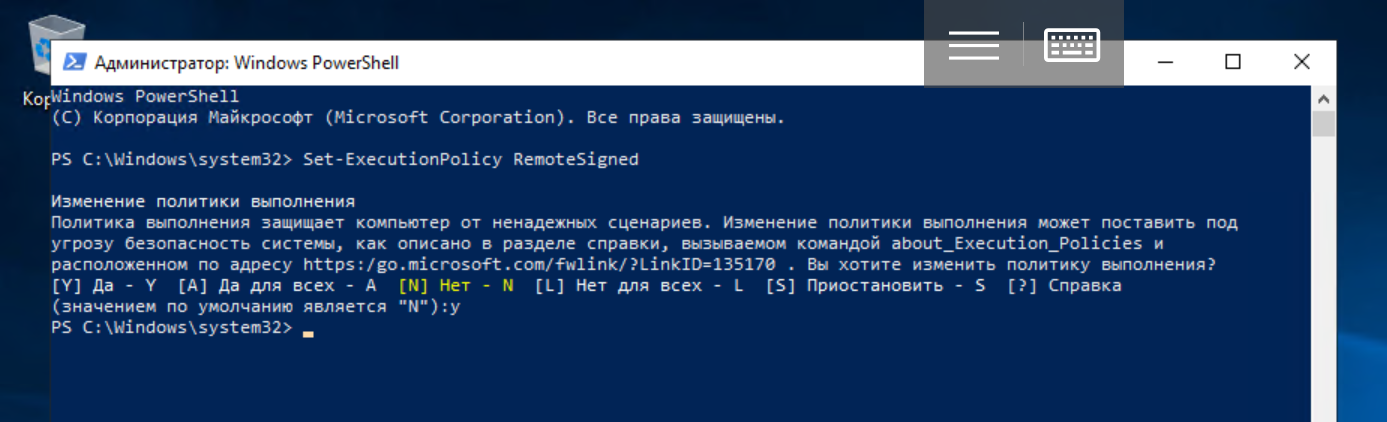
Справка в Windows PowerShell обновляется командлетом Update-Help. Если строка команд получается слишком длинной, аргументы командлета можно перенести на следующую, написав служебный символ ‘`’ и нажав Enter — просто закончить писать команду на одной строке и продолжить на другой не получится.  
  
Ниже приведем несколько примеров распространенных командлетов:   
  
Get-Process — показать запущенные в системе процессы;  
Get-Service — показать службы и их статус;  
Get-Content — вывести содержимое файла.  
  
Для часто используемых командлетов и внешних утилит в Windows PowerShell есть короткие синонимы — алиасы (от англ. Alias). Например, dir — алиас Get-ChildItem. Есть в списке синонимов и аналоги команд из Unix-систем (ls, ps и т.д.), а командлет Get-Help вызывается командой help. Полный список синонимов можно посмотреть с помощью командлета Get-Alias:



Сценарии, функции, модули и язык PowerShell

Скрипты Windows PowerShell хранятся в виде обычных текстовых файлов с расширением .ps1. Запустить их двойным кликом нельзя: нужно правой кнопкой мыши вызвать контекстное меню и выбрать пункт «Запустить в PowerShell». Из консоли придется либо указать полный путь к скрипту, либо перейти в соответствующий каталог и написать имя файла. Запуск сценариев также ограничен системной политикой, а для проверки текущих настроек можно использовать командлет Get-ExecutionPolicy, который выдаст одно из следующих значений:  
  
Restricted — запуск сценариев запрещен (по умолчанию);  
AllSigned — разрешен только запуск подписанных доверенным разработчиком сценариев;  
RemoteSigned — разрешен запуск подписанных и собственных сценариев;  
Unrestricted — разрешен запуск любых сценариев.  
  
У администратора есть два варианта действий. Наиболее безопасный предполагает подписание скриптов, но это довольно серьезное колдунство — мы будем разбираться с ним в следующих статьях. Сейчас пойдем по пути наименьшего сопротивления и поменяем политику:

Set-ExecutionPolicy RemoteSigned



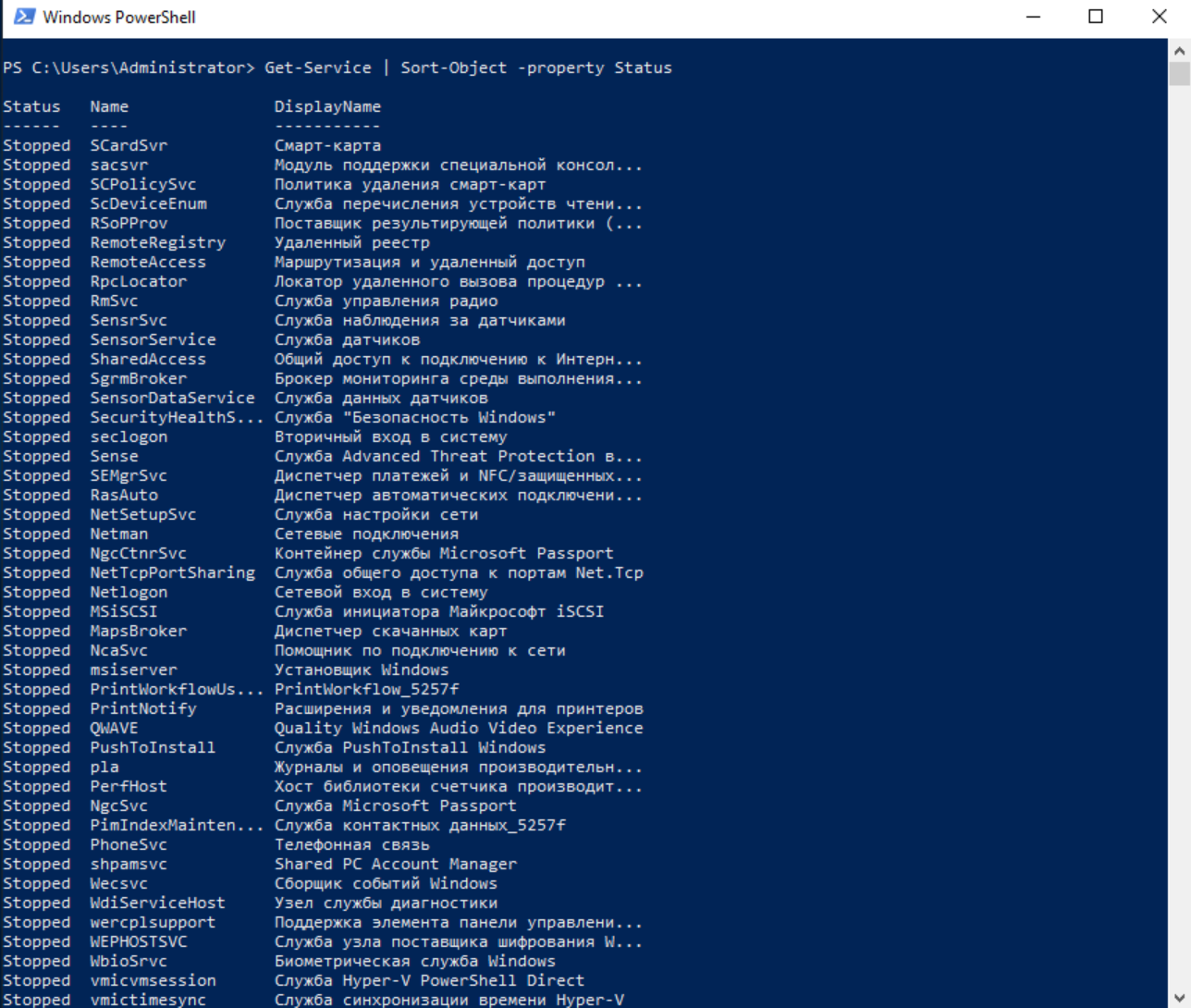
PowerShell для этого придется запустить от имени администратора, хотя с помощью специального параметра можно изменить политику и для текущего пользователя.  
  
Пишутся скрипты на объектно-ориентированном языке программирования, команды которого именуются по тому же принципу, что и рассмотренные ранее командлеты: «Действие-Объект» («Глагол-Существительное»). Основное его предназначение — автоматизация задач администрирования, но это полноценный интерпретируемый язык, в котором есть все необходимые конструкции: условный переход, циклы, переменные, массивы, объекты, обработка ошибок и т.д. Для написания сценариев годится любой текстовый редактор, но удобнее всего запустить Windows PowerShell ISE.  
  
В скрипт можно передавать параметры, делать их обязательными, а также задавать значения по умолчанию. Кроме того Windows PowerShell позволяет создать функции и вызывать их точно так же, как и командлеты: для этого используется конструкция Function и фигурные скобки. Сценарий с функциями называется модулем и имеет расширение .psm1. Модули должны храниться в каталогах, определенных в переменных окружения PowerShell. Посмотреть их можно при помощи следующей команды:

Get-ChildItem Env:\PSModulePath | Format-Table -AutoSize

Конвейеры

В последнем примере мы применили знакомую пользователям оболочек для Unix-систем конструкцию. В Windows PowerShell вертикальная черта также позволяет передать выход одной команды на вход другой, но в реализации конвейера есть и существенная разница: речь здесь идет уже не о наборе символов или каком-то тексте. Встроенные командлеты или пользовательские функции возвращают объекты или массивы объектов, а также могут получать их на входе. Как в Bourne shell и его многочисленных последователях, в PowerShell с помощью конвейера упрощается выполнение сложных задач.  
  
Простейший пример конвейера выглядит так:

Get-Service | Sort-Object -property Status



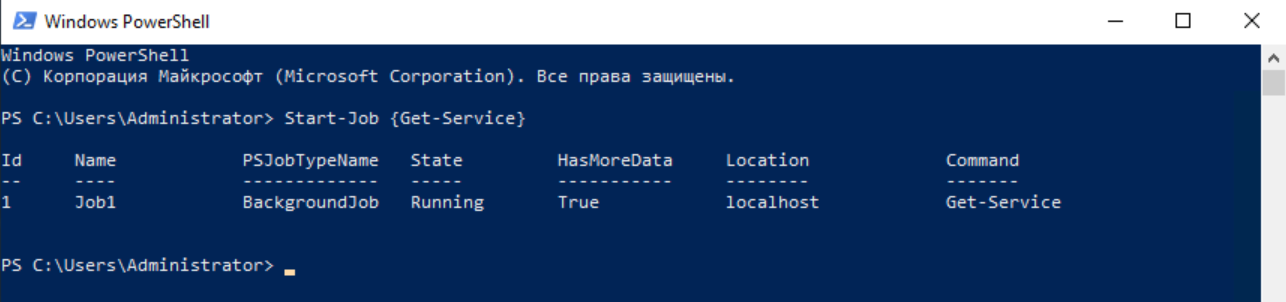
Сначала выполняется командлет Get-Service, а потом все полученные им службы передаются на сортировку по свойству Status командлету Sort-Object. В какой именно аргумент передается результат работы предыдущего участка конвейера, зависит от его типа — обычно это InputObject. Подробнее этот вопрос будет рассматриваться в посвященной языку программирования PowerShell статье.   
  
При желании цепочку можно продолжить и передать результат работы Sort-Object еще одному командлету (выполняться они будут слева направо). Кстати, пользователям Windows доступна и привычная всем юниксоидам конструкция для постраничного вывода: 

Get-Service | Sort-Object -property Status | more

Запуск задач в фоновом режиме

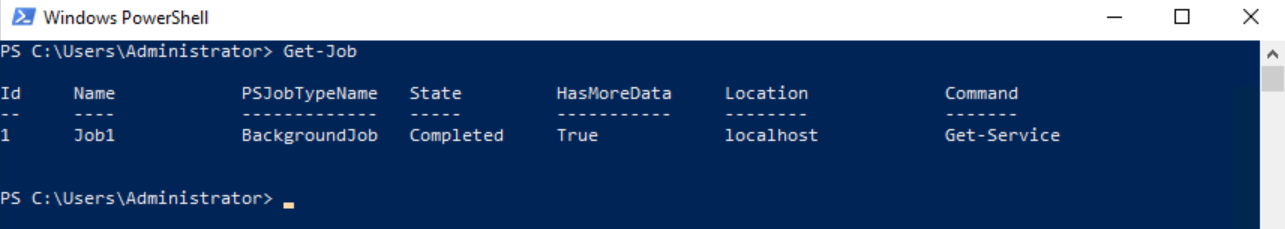
Довольно часто бывает нужно запустить некую команду в фоне, чтобы не дожидаться результата ее выполнения в сессии оболочки. В Windows PowerShell есть несколько командлетов на такой случай:  
  
Start-Job — запуск фоновой задачи;  
Stop-Job — остановка фоновой задачи;  
Get-Job — просмотр списка фоновых задач;  
Receive-Job — просмотр результата выполнения фоновой задачи;  
Remove-Job — удаление фоновой задачи;  
Wait-Job — перевод фоновой задачи обратно в консоль.  
  
Для запуска фоновой задачи мы используем командлет Start-Job и в фигурных скобках указываем команду или набор команд:

Start-Job {Get-Service}



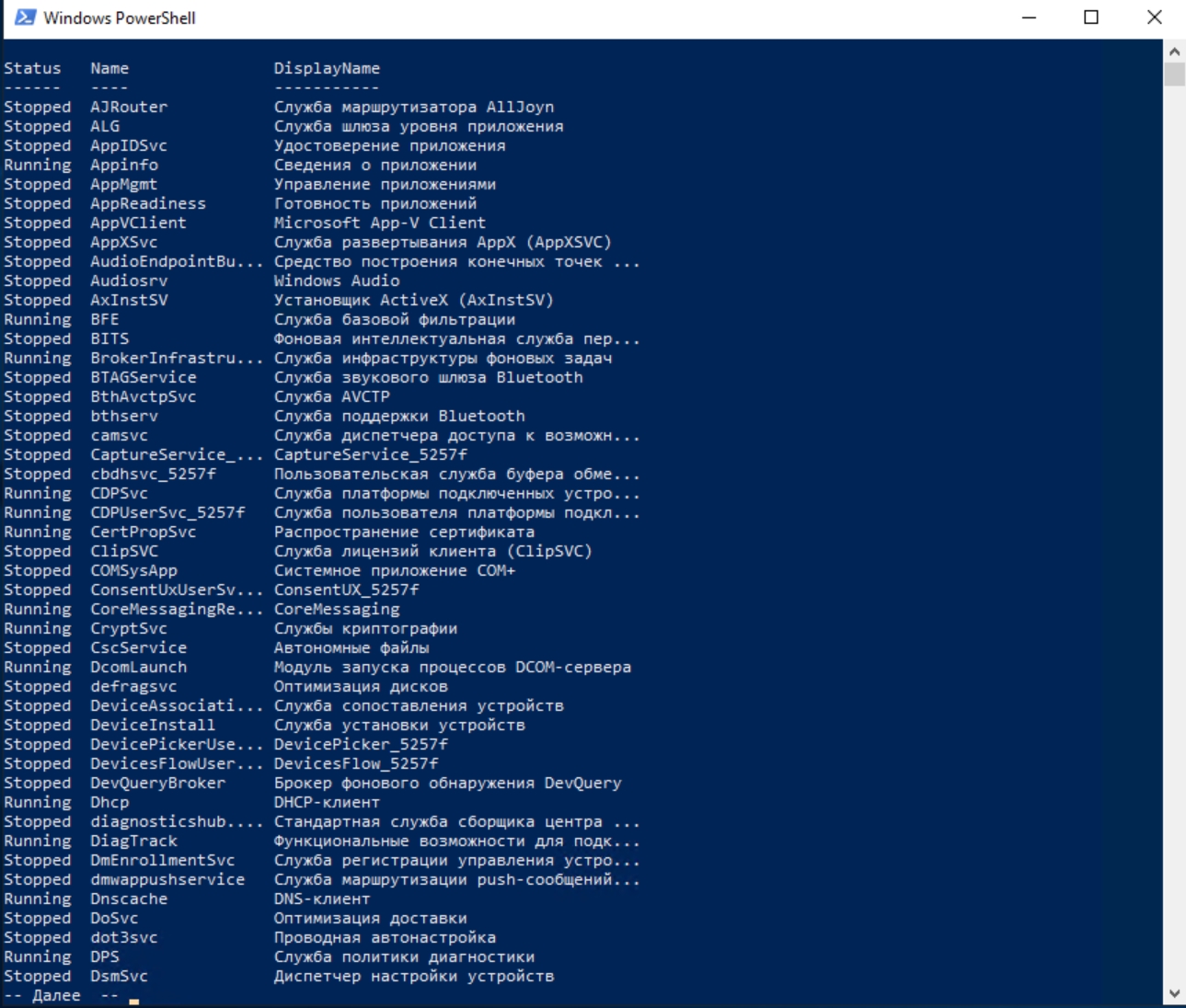
Фоновыми задачами в Windows PowerShell можно манипулировать, зная их имена. Для начала научимся их отображать:

Get-Job



Теперь покажем результат работы задания Job1:

Receive-Job Job1 | more



Всё довольно просто.

Удаленное выполнение команд

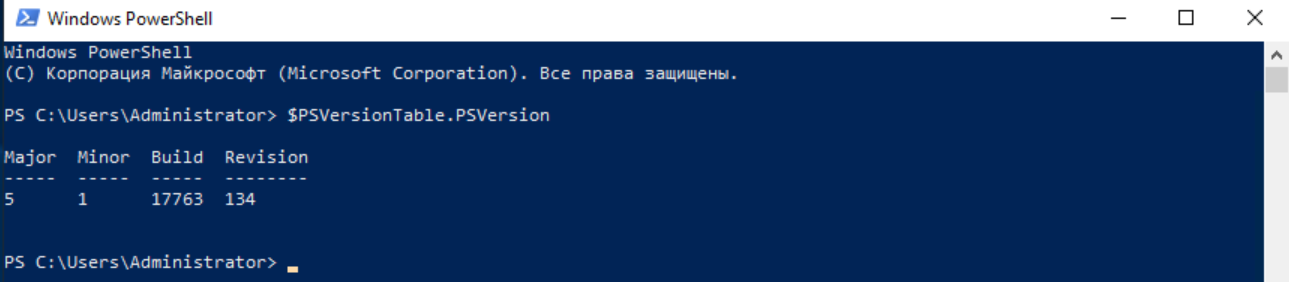
Windows PowerShell позволяет выполнять команды и сценарии не только на локальном, но и на удаленном компьютере и даже на целой группе машин. Для этого существует несколько способов:

* У многих командлетов есть параметр -ComputerName, но таким способом не получится, например, создать конвейер;
* Командлет Enter-PSSession позволяет создать на удаленной машине интерактивный сеанс;
* С помощью командлета Invoke-Command можно выполнять команды или сценарии на одном или нескольких удаленных компьютерах.

Версии PowerShell

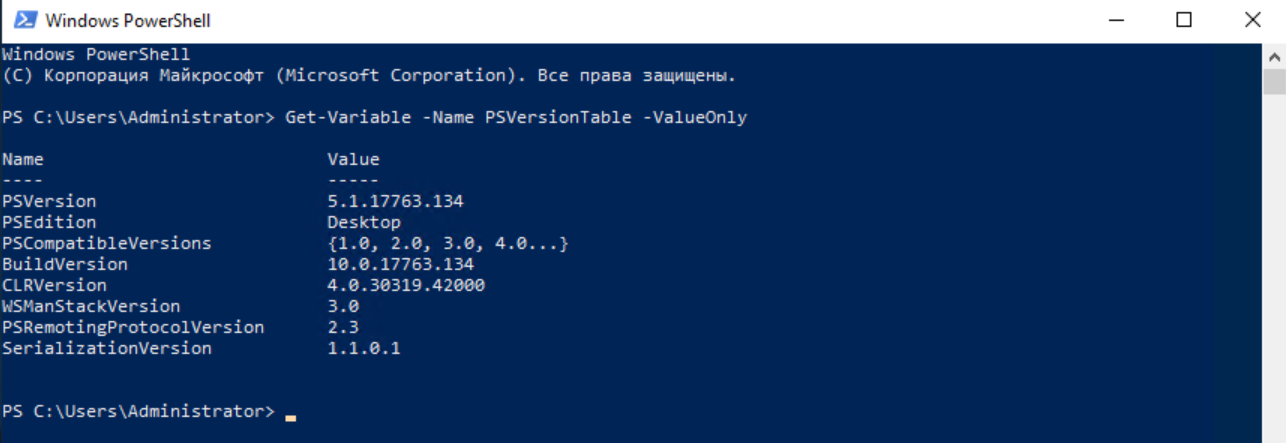
С момента первого релиза в 2006 году PowerShell серьезно изменился. Инструмент доступен для множества систем, работающих на разных аппаратных платформах (x86, x86-64, Itanium, ARM): Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008/2008 R2, Windows 7, Windows 8, Windows 8.1, Windows RT, Windows RT 8.1, Windows Server 2012/2012 R2, Windows 10, Windows Server 2016, GNU/Linux и OS X. Последний релиз 6.2 выпущен 10 января 2018 года. Написанные для более ранних версий скрипты с высокой вероятностью будут работать в более поздних, а вот с обратным переносом могут возникнуть проблемы, поскольку за годы развития в PowerShell появилось большое количество новых командлетов. Узнать версию установленной на компьютере командной оболочки можно с помощью свойства PSVersion встроенной переменной $PSVersionTable:

$PSVersionTable.PSVersion



Также можно воспользоваться командлетом:

Get-Variable -Name PSVersionTable –ValueOnly



То же самое делается и с помощью командлета Get-Host. На самом деле вариантов множество, но для их применения нужно изучить язык программирования PowerShell, чем мы и займемся в [**следующей статье**](https://habr.com/ru/company/ruvds/blog/490924/). 

Итоги

Корпорации Microsoft удалось создать действительно мощную командную оболочку с удобной интегрированной средой для разработки скриптов. От привычных нам по миру Unix средств ее отличает глубокая интеграция с операционными системами семейства Windows, а также с программным обеспечением для них и платформой .NET Core. PowerShell можно назвать объектно ориентированной оболочкой, потому что командлеты и пользовательские функции возвращают объекты или массивы объектов и могут получать их на входе. Думаем, этим инструментом должны владеть все администраторы серверов на Windows: прошло время, когда они могли обойтись без командной строки. Продвинутая консольная оболочка особенно необходима на [наших недорогих VPS, работающих под управлением Windows Server Core](https://ruvds.com/ru-rub/marketplace/servercore#order), но это уже совсем другая история.