Что такое Windows PowerShell и с чем его едят? Часть 4: Работа с объектами, собственные классы

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

[](https://habr.com/ru/company/ruvds/blog/494978/)  
  
Текстовый вывод команд в окне интерпретатора PowerShell — всего лишь способ отображения информации в пригодном для человеческого восприятия виде. На самом деле среда [ориентирована](https://habr.com/ru/company/ruvds/blog/487876/) на работу с объектами: командлеты и функции получают их на входе и [возвращают на выходе](https://habr.com/ru/company/ruvds/blog/490924/#section13), а доступные в интерактивном режиме и в сценариях типы переменных базируются на классах .NET. В четвертой статье цикла мы изучим работу с объектами более детально.

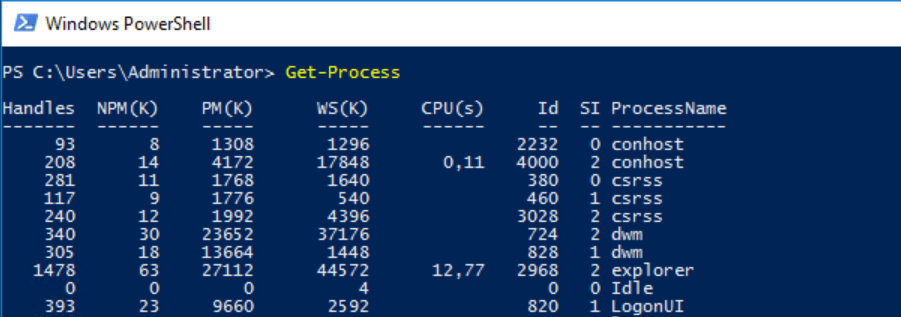
Оглавление:

[Объекты в PowerShell](https://habr.com/ru/company/ruvds/blog/494978/#section1)  
[Просмотр структуры объектов](https://habr.com/ru/company/ruvds/blog/494978/#section2)  
[Фильтрация объектов](https://habr.com/ru/company/ruvds/blog/494978/#section3)  
[Сортировка объектов](https://habr.com/ru/company/ruvds/blog/494978/#section4)  
[Выделение объектов и их частей](https://habr.com/ru/company/ruvds/blog/494978/#section5)  
[ForEach-Object, Group-Object и Measure-Object](https://habr.com/ru/company/ruvds/blog/494978/#section6)  
[Создание объектов .NET и COM (New-Object)](https://habr.com/ru/company/ruvds/blog/494978/#section7)  
[Вызов статических методов](https://habr.com/ru/company/ruvds/blog/494978/#section8)  
[Тип PSCustomObject](https://habr.com/ru/company/ruvds/blog/494978/#section9)  
[Создание собственных классов](https://habr.com/ru/company/ruvds/blog/494978/#section10)

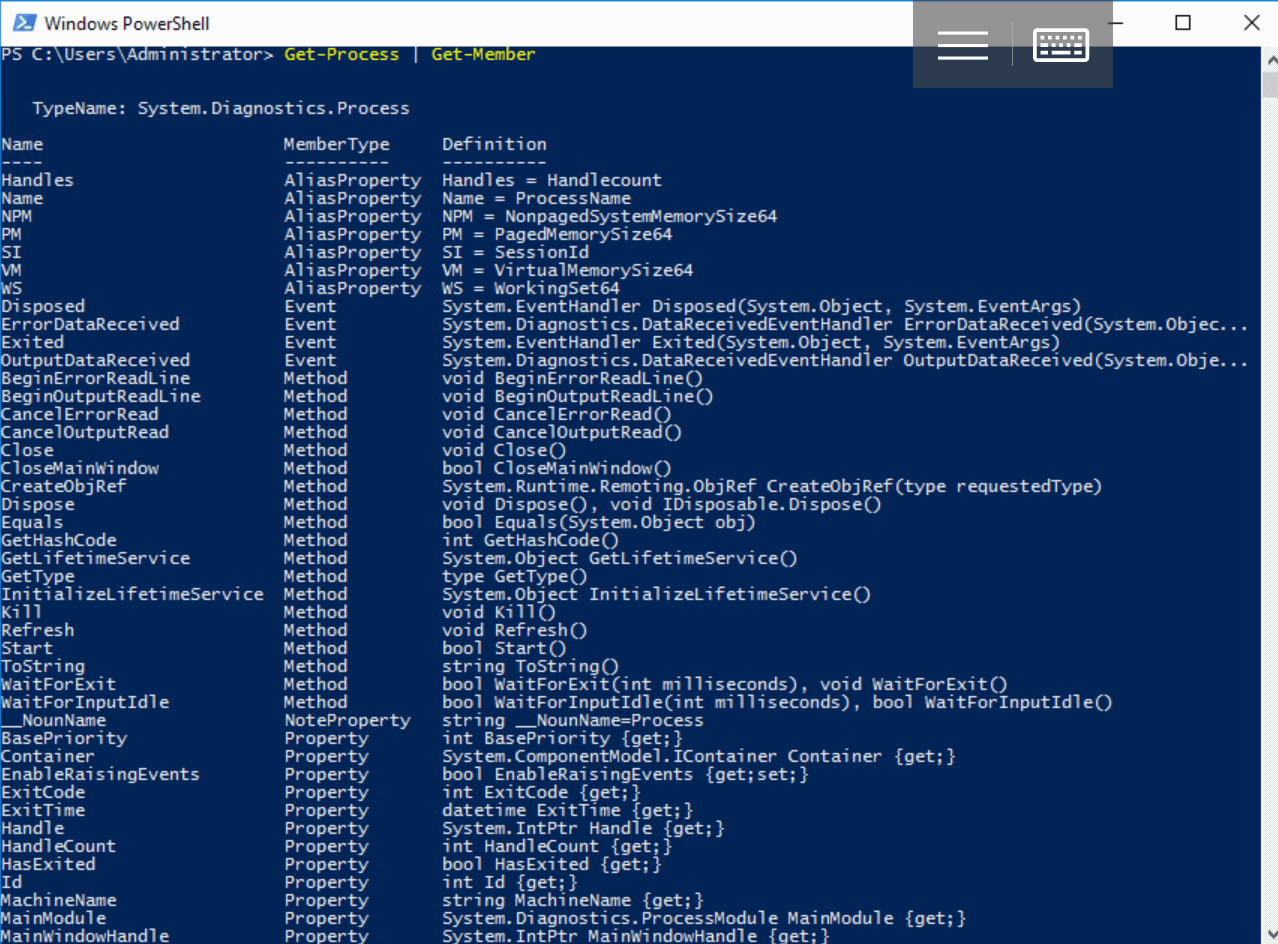
Объекты в PowerShell

Напомним, что объект — это совокупность полей данных (свойств, событий и т.д.) и способов их обработки (методов). Его структура задается типом, который как правило базируется на использующихся в унифицированной платформе .NET Core классах. Также есть возможность работать с объектами COM, CIM (WMI) и ADSI. Свойства и методы нужны для выполнения различных действий над данными, кроме того в PowerShell объекты можно передавать как аргументы в функции и командлеты, присваивать их значения переменным, а также существует [механизм композиции команд](https://habr.com/ru/company/ruvds/blog/493366/#section4) (конвейер или pipeline). Каждая команда в конвейере передает свой вывод следующей поочередно — объект за объектом. Для обработки можно использовать скомпилированные командлеты или создавать собственные [расширенные функции](https://habr.com/ru/company/ruvds/blog/493366/#section6), чтобы производить различные манипуляции с объектами в конвейере: фильтрацию, сортировку, группировку и даже изменение их структуры. Передача данных в таком виде имеет серьезное преимущество: принимающей команде не нужно заниматься синтаксическим разбором потока байтов (текста), вся нужная информация легко извлекается с помощью обращения к соответствующим свойствам и методам.

Просмотр структуры объектов

Для примера запустим командлет Get-Process, позволяющий получить информацию о работающих в системе процессах:  
  
  
  
Он выведет на экран некие отформатированные текстовые данные, не дающие представления о свойствах возвращаемых объектов и их методах. Для тонкого препарирования вывода необходимо научиться исследовать структуру объектов и в этом нам поможет командлет Get-Member:

Get-Process | Get-Member

  
  
Здесь мы уже видим тип и структуру, а с помощью дополнительных параметров можем, например, вывести только свойства попавшего на вход объекта:

Get-Process | Get-Member -MemberType Property

Эти знания понадобятся для решения задач администрирования в интерактивном режиме или для написания собственных скриптов: скажем, чтобы получить сведения о зависших процессах по свойству Responding.

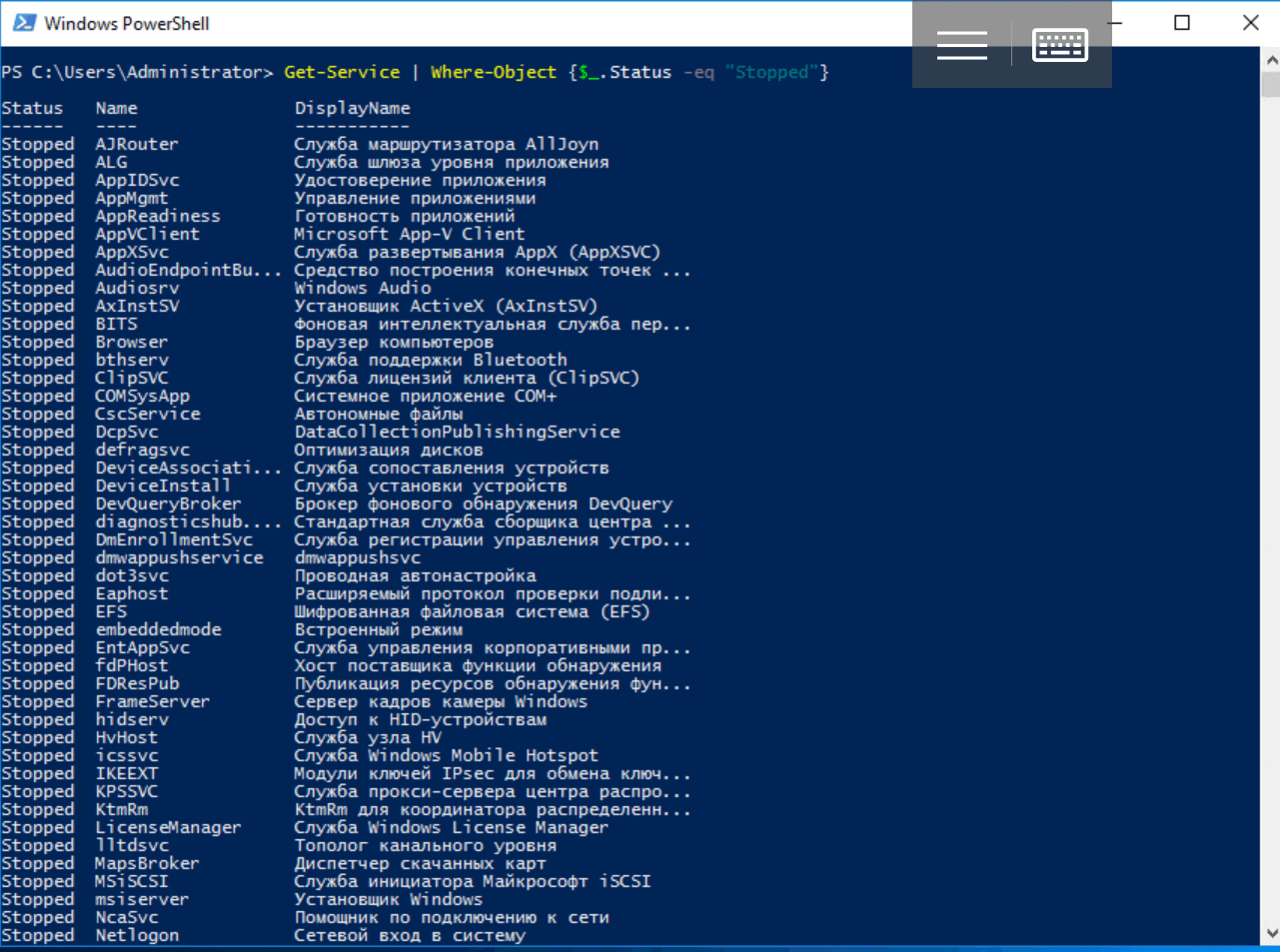
Фильтрация объектов

PowerShell позволяет пропускать по конвейеру объекты, удовлетворяющие определенному условию:

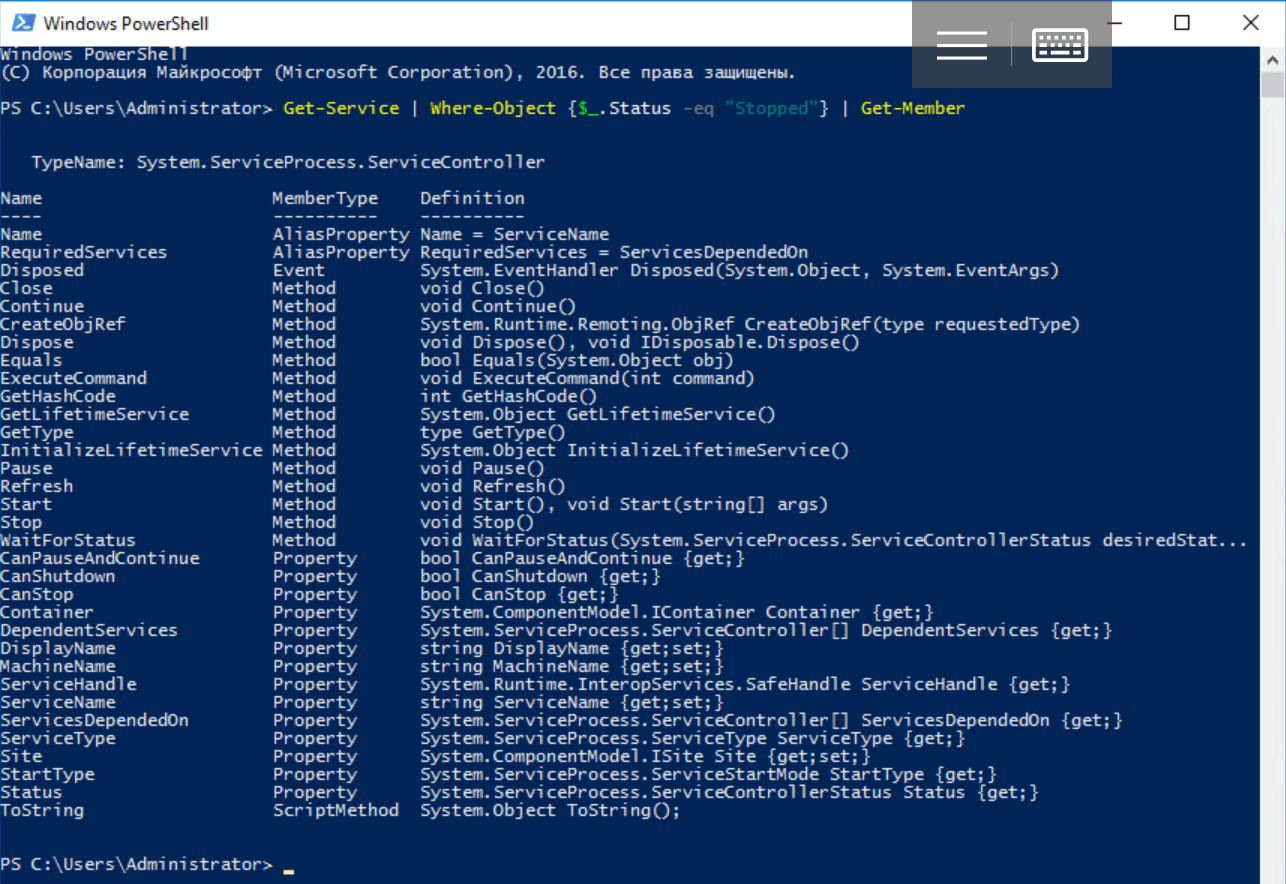
Where-Object { блок сценария }

Результатом выполнения блока сценария в операторных скобках должно быть логические значение. Если оно истинно ($true) попавший на вход командлету Where-Object объект будет передан по конвейеру дальше, в противном случае (значение $false) он будет удален. Для примера выведем список остановленных служб Windows Server, т.е. таких, у которых свойство Status имеет значение «Stopped»:

Get-Service | Where-Object {$\_.Status -eq "Stopped"}

  
  
Здесь мы снова видим текстовое представление, но при желании понять тип и внутреннее устройство проходящих через конвейер объектов нетрудно:

Get-Service | Where-Object {$\_.Status -eq "Stopped"} | Get-Member



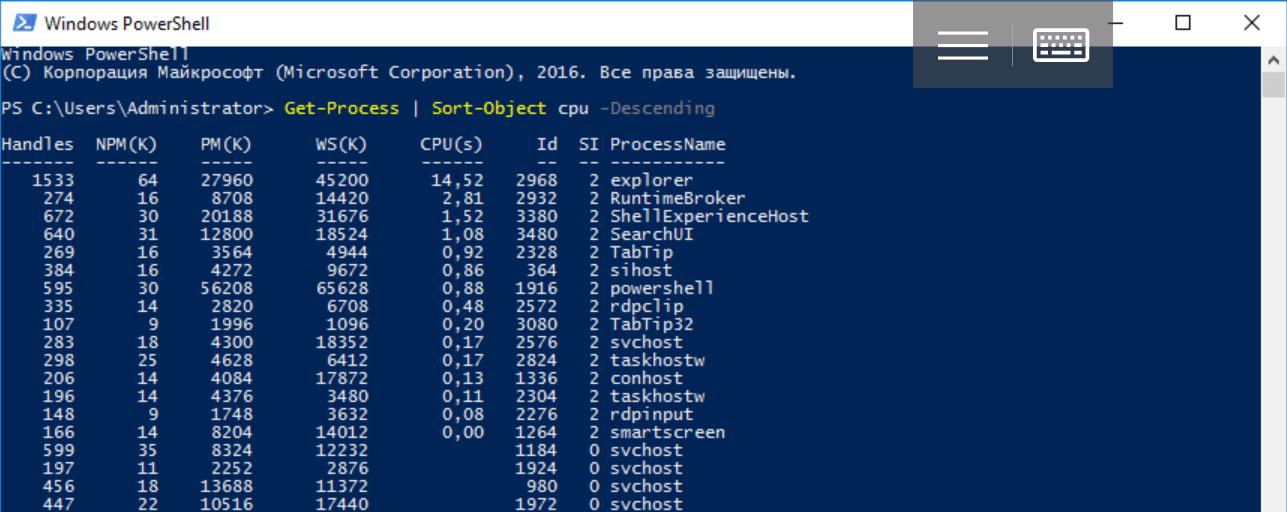
Сортировка объектов

При конвейерной обработке объектов часто возникает необходимость их сортировки. В командлет Sort-Object передаются имена свойств (ключей сортировки), а он возвращает упорядоченные по их значениям объекты. Вывод запущенных процессов несложно отсортировать по затраченному процессорному времени (свойство cpu):

Get-Process | Sort-Object –Property cpu

Параметр -Property при вызове командлета Sort-Object можно не указывать — он используется по умолчанию. Для обратной сортировки применяется параметр -Descending:

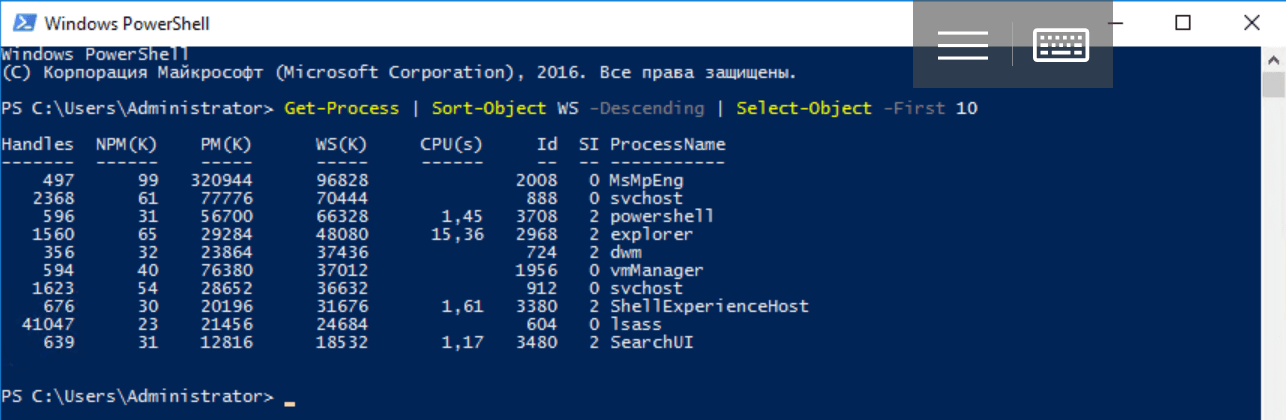
Get-Process | Sort-Object cpu -Descending



Выделение объектов и их частей

Командлет Select-Object позволяет выделить определенное количество объектов в начале или в конце конвейера с помощью параметров -First или -Last. С его помощью можно выбрать единичные объекты или определенные свойства, а также создать на их основе новые объекты. Разберем работу командлета на простых примерах.  
  
Следующая команда выводит информацию о 10 процессах, потребляющих максимальный объем оперативной памяти (свойство WS):

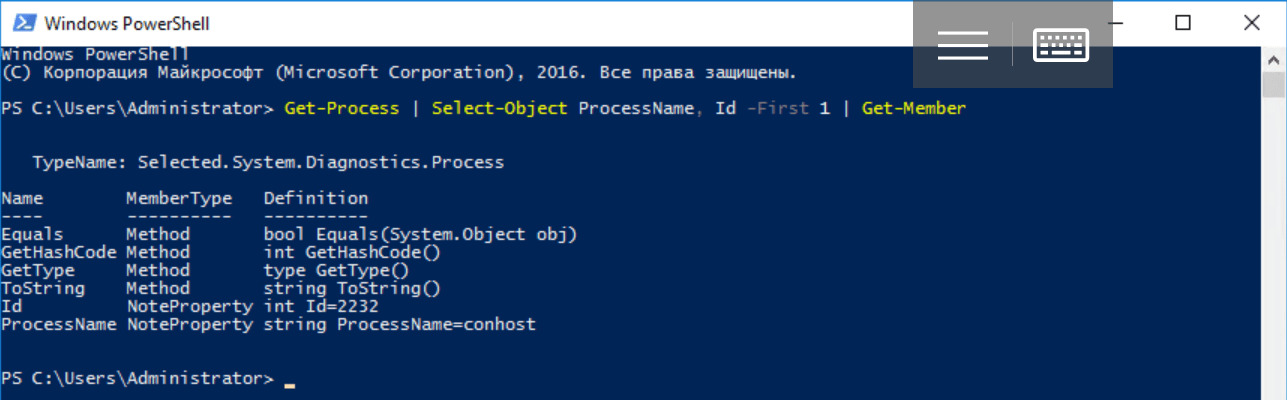
Get-Process | Sort-Object WS -Descending | Select-Object -First 10

  
  
Можно выделить только определенные свойства проходящих через конвейер объектов и создать на их основе новые:

Get-Process | Select-Object ProcessName, Id -First 1

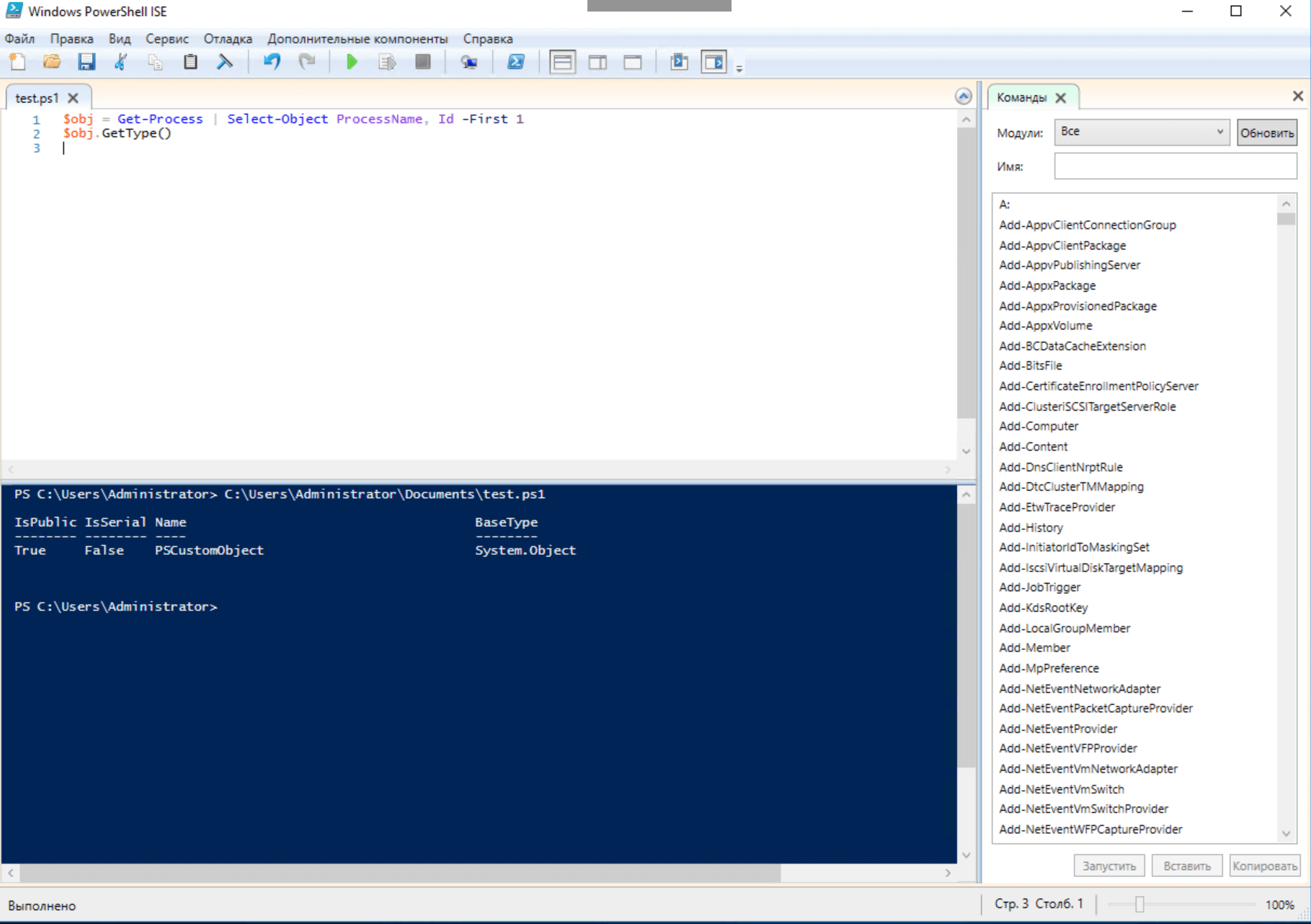
В результате работы конвейера мы получим новый объект, структура которого будет отличаться от структуры возвращаемых командлетом Get-Process. Убедимся в этом при помощи Get-Member:

Get-Process | Select-Object ProcessName, Id -First 1 | Get-Member

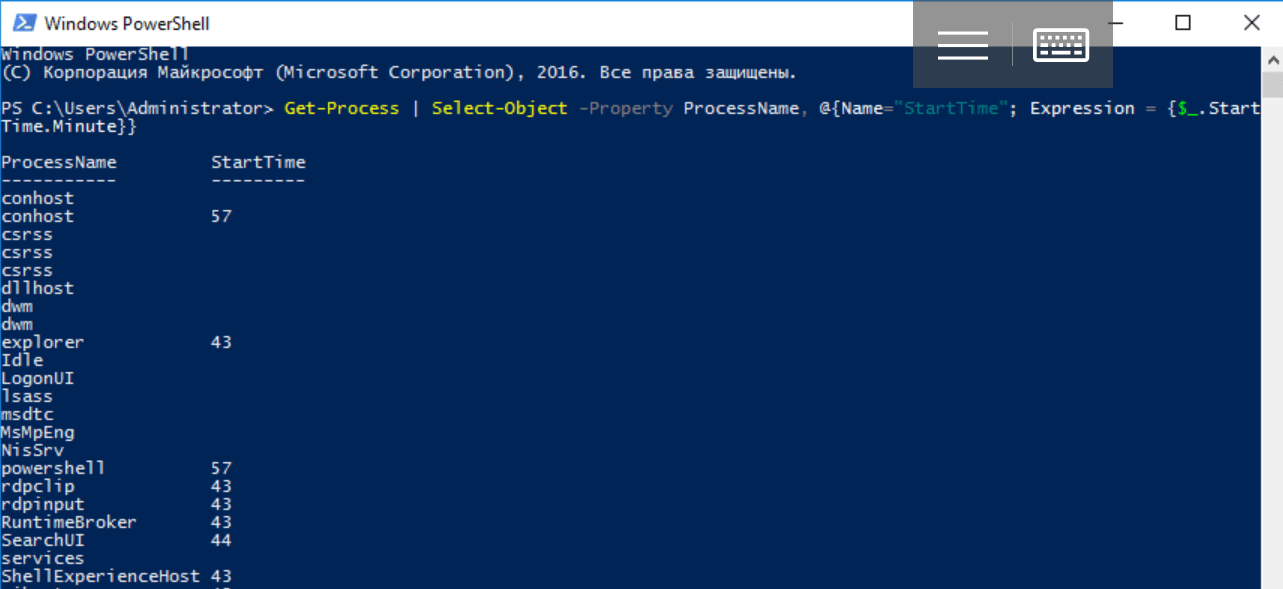
  
  
Обратите внимание, что Select-Object возвращает единичный объект (-First 1), у которого всего два указанных нами поля: их значения были скопированы из первого переданного в конвейер командлетом Get-Process объекта. На использовании Select-Object основан один из способов создания объектов в сценариях PowerShell:

$obj = Get-Process | Select-Object ProcessName, Id -First 1

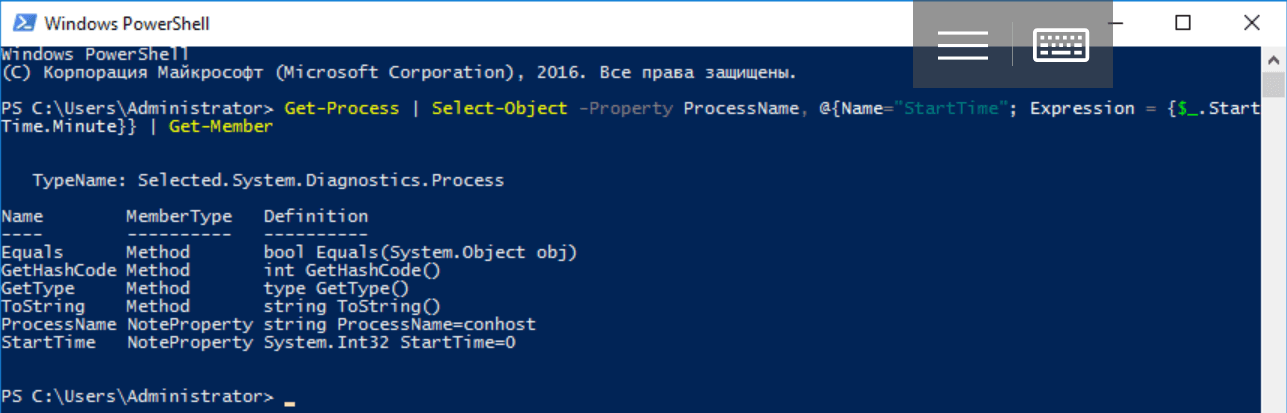
$obj.GetType()

  
  
С помощью Select-Object можно добавлять объектам вычисляемые свойства, которые необходимо представить в виде [хэш-таблицы](https://habr.com/ru/company/ruvds/blog/490924/#section12). При этом значение ее первого ключа соответствует имени свойства, а значение второго — значению свойства для текущего элемента конвейера:

Get-Process | Select-Object -Property ProcessName, @{Name="StartTime"; Expression = {$\_.StartTime.Minute}}

  
  
Посмотрим на структуру проходящих через конвейер объектов:

Get-Process | Select-Object -Property ProcessName, @{Name="StartTime"; Expression = {$\_.StartTime.Minute}} | Get-Member



ForEach-Object, Group-Object и Measure-Object

Для работы с объектами существуют и другие командлеты. Для примера расскажем о трех наиболее полезных:  
  
**ForEach-Object** позволяет выполнить код на языке PowerShell для каждого объекта в конвейере:

ForEach-Object { блок сценария }

**Group-Object** группирует объекты по значению свойства:

Group-Object PropertyName

Если запустить его с параметром -NoElement, можно узнать количество элементов в группах.  
  
**Measure-Object** агрегирует различные сводные параметры по значениям полей объектов в конвейере (вычисляет сумму, а также находит минимальное, максимальное или среднее значение):

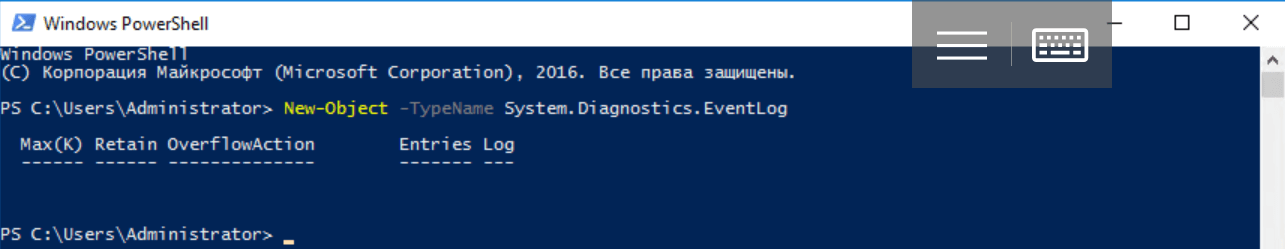
Measure-Object -Property PropertyName -Minimum -Maximum -Average -Sum

Обычно рассмотренные командлеты используются в интерактивном режиме, а в скриптах чаще создаются [функции](https://habr.com/ru/company/ruvds/blog/493366/#section5) с блоками Begin, Process и End.

Создание объектов .NET и COM (New-Object)

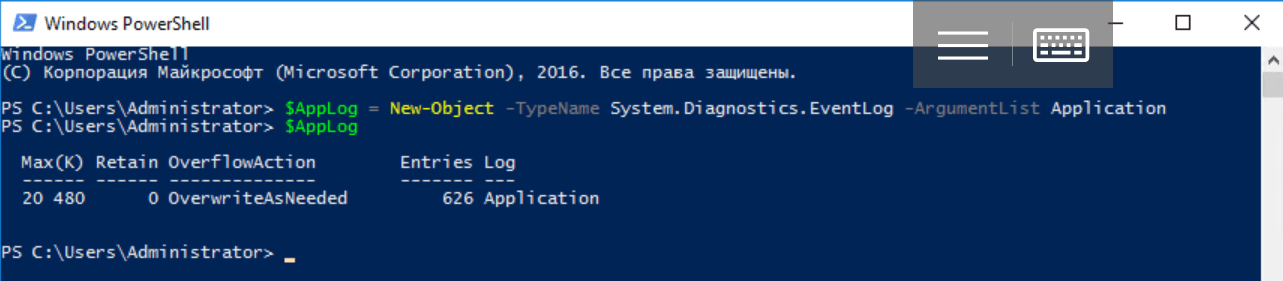
Есть множество программных компонентов с интерфейсами .NET Core и COM, которые пригодятся системным администраторам. С помощью класса System.Diagnostics.EventLog можно управлять системными журналами непосредственно из Windows PowerShell. Разберем пример создания экземпляра этого класса при помощи командлета New-Object с параметром -TypeName:

New-Object -TypeName System.Diagnostics.EventLog

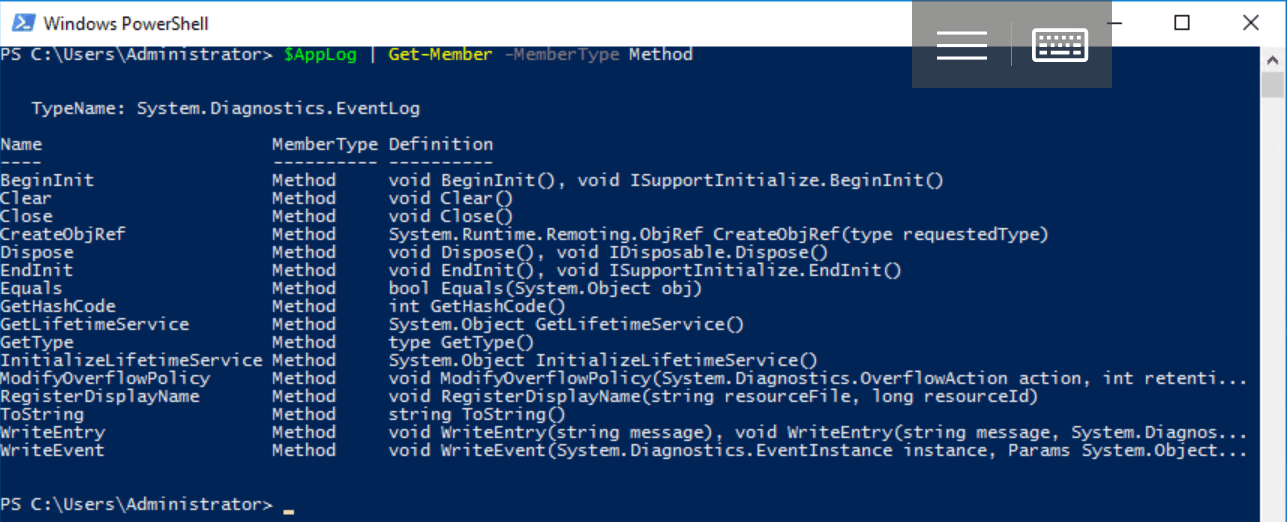
  
  
Поскольку мы не указали определенный журнал событий, полученный экземпляр класса не содержит данных. Чтобы это изменить, необходимо во время его создания вызвать специальный метод-конструктор при помощи параметра -ArgumentList. Если мы хотим получить доступ к журналу приложений, в конструктор следует передать строку «Application» в качестве аргумента:

$AppLog = New-Object -TypeName System.Diagnostics.EventLog -ArgumentList Application

$AppLog

  
  
Обратите внимание: выходные данные команды мы сохранили в переменной $AppLog. Хотя в интерактивном режиме обычно используются конвейеры, написание сценариев часто требует сохранения ссылки на объект. Кроме того основные классы .NET Core содержатся в пространстве имен System: PowerShell по умолчанию ищет в нем указанные типы, поэтому написание Diagnostics.EventLog вместо System.Diagnostics.EventLog вполне корректно.  
  
Для работы с журналом можно обращаться к соответствующим методам:

$AppLog | Get-Member -MemberType Method

  
  
Скажем очищается он методом Clear() при наличии прав доступа:

$AppLog.Clear()

Командлет New-Object применяется и для работы с СОМ-компонентами. Их довольно много — от поставляемых с сервером сценариев Windows библиотек до приложений ActiveX, таких, например, как Internet Explorer. Чтобы создать СОМ-объект, требуется задать параметр -ComObject с программным идентификатора ProgId нужного класса:

New-Object -ComObject WScript.Shell

New-Object -ComObject WScript.Network

New-Object -ComObject Scripting.Dictionary

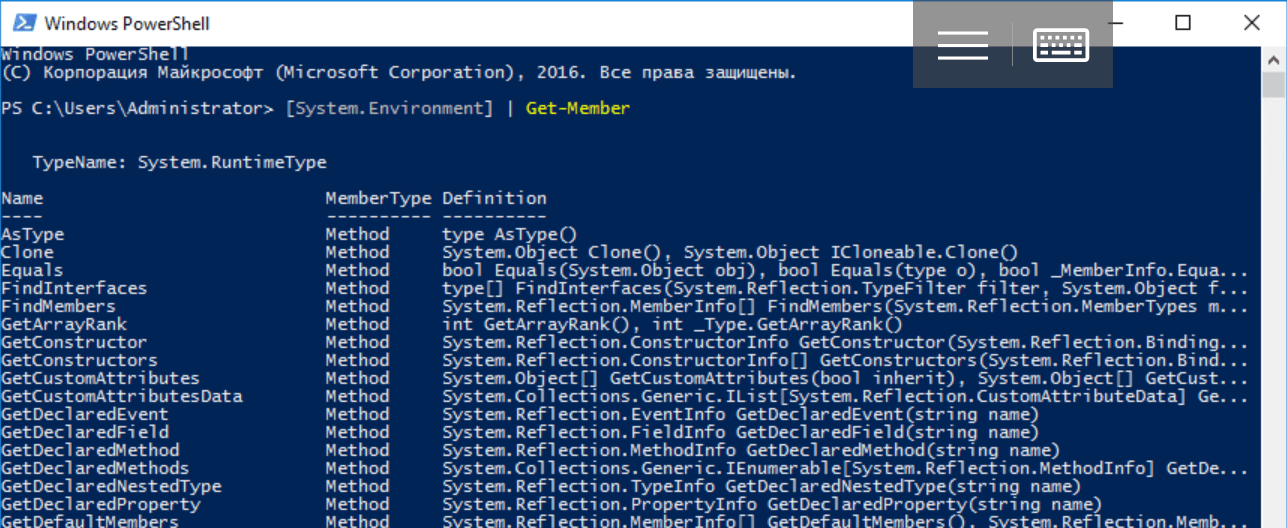
New-Object -ComObject Scripting.FileSystemObject

Для создания собственных объектов с произвольной структурой использование New-Object выглядит слишком архаичным и громоздким, этот командлет используется для работы с внешними по отношению к PowerShell программными компонентами. В следующих статьях этот вопрос будет разобран более подробно. Помимо объектов .NET и COM мы также изучим объекты CIM (WMI) и ADSI.

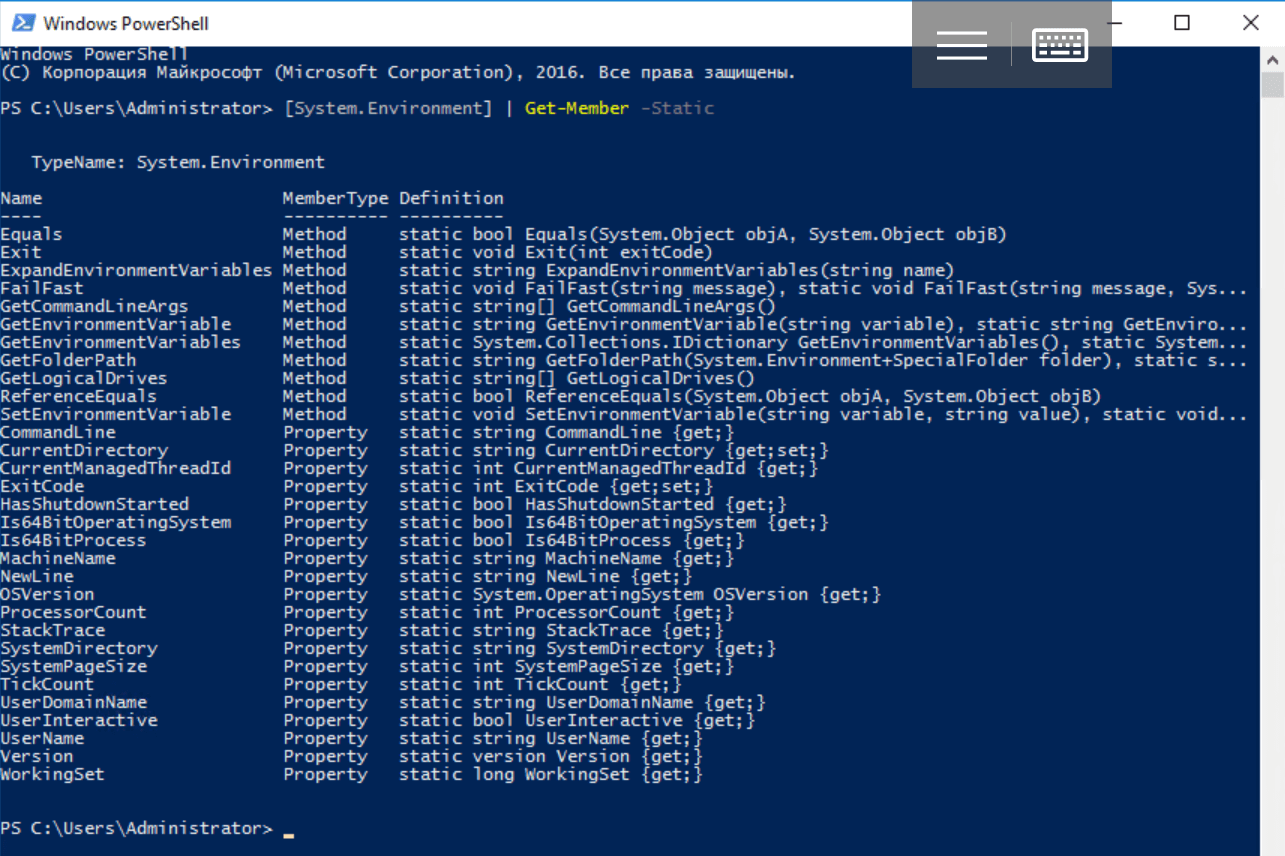
Вызов статических методов

Экземпляры некоторых классов .NET Core создать невозможно: к их числу относятся System.Environment и System.Math. Они являются [статическими](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members) и содержат только статические свойства и методы. По сути это справочные библиотеки, которые используются без создания объектов. Сослаться на статический класс можно через литерал, заключив имя типа в квадратные скобки. При этом если посмотреть на структуру объекта с помощью Get-Member, мы увидим тип System.RuntimeType вместо System.Environment:

[System.Environment] | Get-Member

  
  
Для просмотра только статических элементов нужно вызвать Get-Member с параметром -Static (обратите внимание на тип объекта):

[System.Environment] | Get-Member -Static

  
  
Для доступа к статическим свойствам и методам используются два идущих подряд двоеточия вместо точки после литерала:

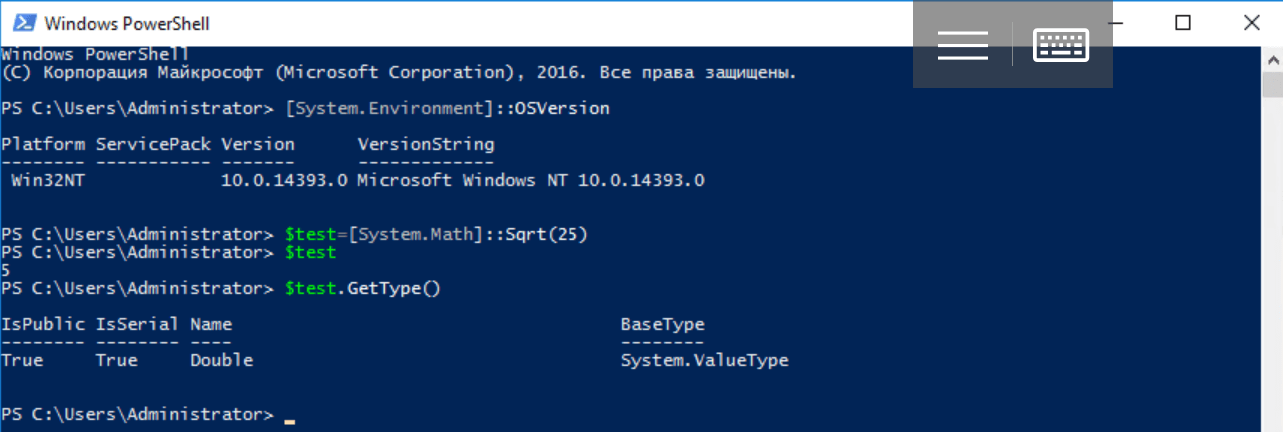
[System.Environment]::OSVersion

Или

$test=[System.Math]::Sqrt(25)

$test

$test.GetType()



Тип PSCustomObject

Среди многочисленных доступных в PowerShell типов данных отдельно стоит упомянуть PSCustomObject, предназначенный для хранения объектов с произвольной структурой. Создание такого объекта с помощью командлета New-Object считается классическим, но громоздким и устаревшим способом:

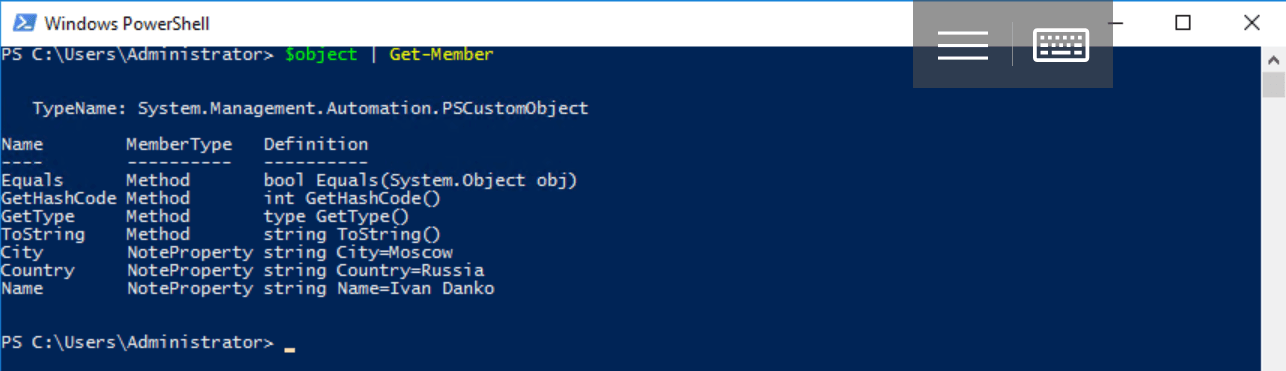
$object = New-Object –TypeName PSCustomObject -Property @{Name = 'Ivan Danko';

City = 'Moscow';

Country = 'Russia'}

Посмотрим на структуру объекта:

$object | Get-Member

  
  
Начиная с PowerShell 3.0 доступен и другой синтаксис:

$object = [PSCustomObject]@{Name = 'Ivan Danko';

City = 'Moscow';

Country = 'Russia'

}

Получить доступ к данным можно одним из эквивалентных способов:

$object.Name

$object.'Name'

$value = 'Name'

$object.$value

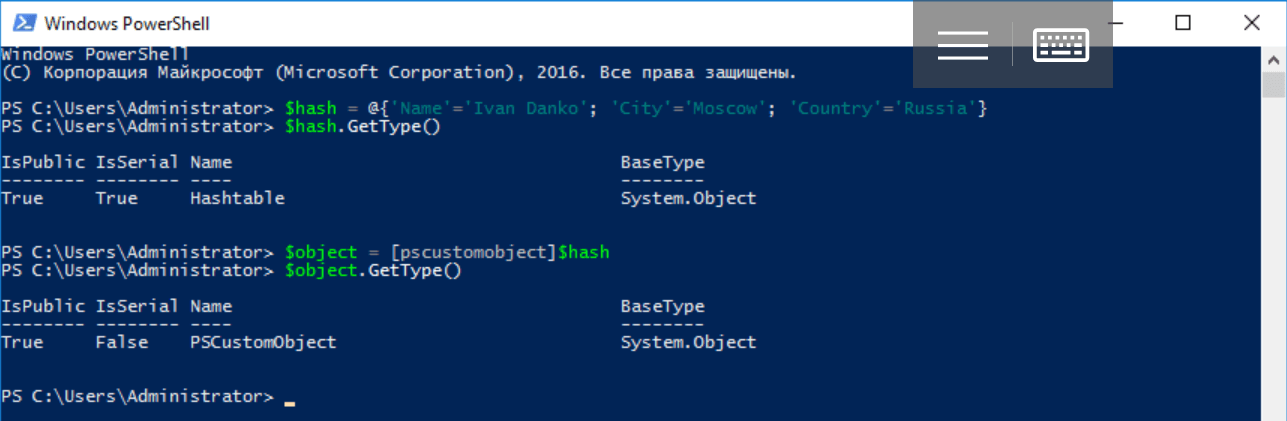
Приведем пример преобразования в объект существующей хэштаблицы:

$hash = @{'Name'='Ivan Danko'; 'City'='Moscow'; 'Country'='Russia'}

$hash.GetType()

$object = [pscustomobject]$hash

$object.GetType()

  
  
Один из недостатков объектов этого типа — порядок их свойств может поменяться. Чтобы этого избежать, необходимо использовать атрибут [ordered]:

$object = [PSCustomObject][ordered]@{Name = 'Ivan Danko';

City = 'Moscow';

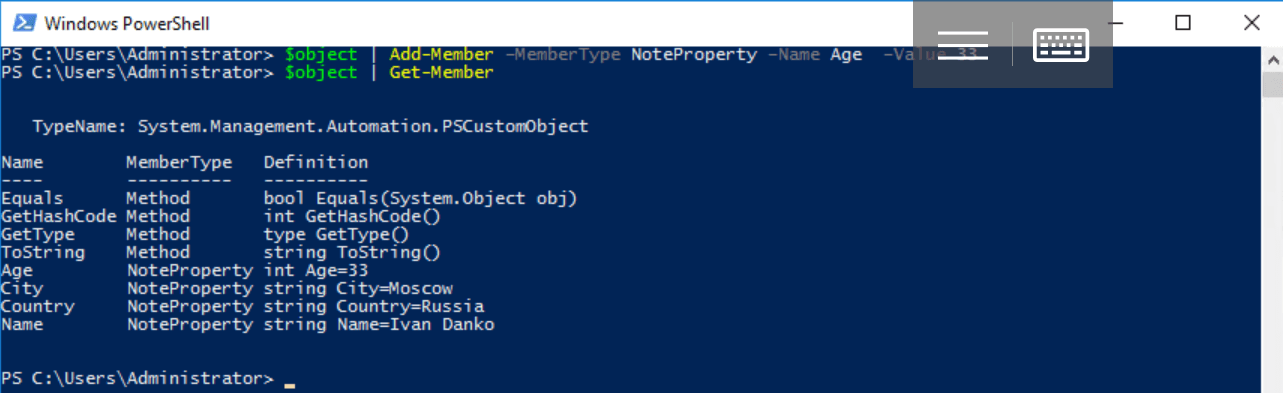
Country = 'Russia'

}

Есть и другие варианты создания объекта: выше мы рассмотрели использование командлета [Select-Object](https://habr.com/ru/company/ruvds/blog/494978/#section5). Осталось разобраться с добавлением и удалением элементов. Сделать это для объекта из предыдущего примера довольно просто:

$object | Add-Member –MemberType NoteProperty –Name Age –Value 33

$object | Get-Member

  
  
Командлет Add-Member позволяет добавлять ранее созданному объекту $object не только свойства, но и методы посредством использования конструкции "-MemberType ScriptMethod":

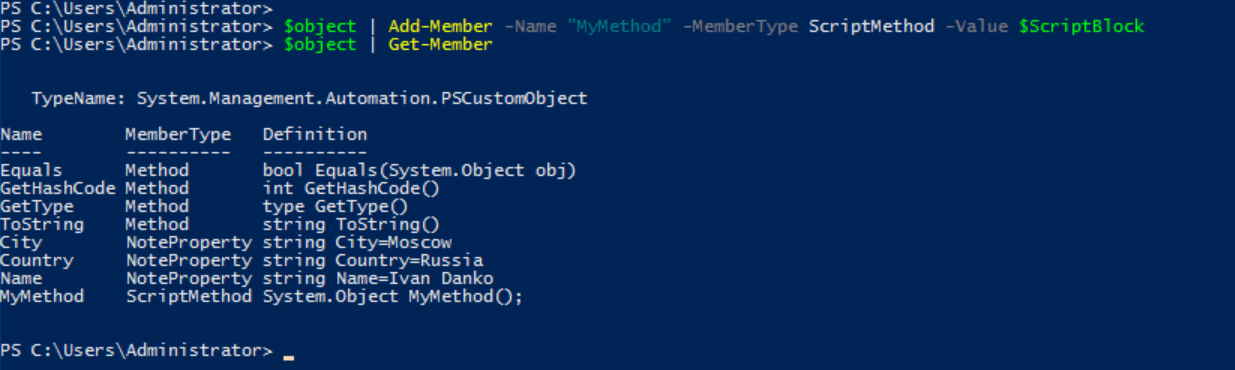
$ScriptBlock = {

*# код*

}

$object | Add-Member -Name "MyMethod" -MemberType ScriptMethod -Value $ScriptBlock

$object | Get-Member

Обратите внимание: для хранения кода нового метода мы использовали переменную $ScriptBlock типа ScriptBlock.  
  
  
  
Для удаления свойств используется соответствующий метод:

$object.psobject.properties.remove('Name')

Создание собственных классов

В PowerShell 5.0 появилась возможность определения [классов](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_classes) с использованием характерного для объектно-ориентированных языков программирования синтаксиса. Для этого предназначено служебное слово Class, после которого следует задать имя класса и описать его тело в операторных скобках:

class MyClass

{

*# тело класса*

}

Это настоящий тип .NET Core, в теле которого описываются его свойства, методы и другие элементы. Рассмотрим пример определения простейшего класса:

class MyClass

{

[string]$Name

[string]$City

[string]$Country

}

Для создания объекта (экземпляра класса) используется командлет [New-Object](https://habr.com/ru/company/ruvds/blog/494978/#section7), либо литерал типа [MyClass] и [псевдостатический метод](https://habr.com/ru/company/ruvds/blog/494978/#section8) new (конструктор по умолчанию):

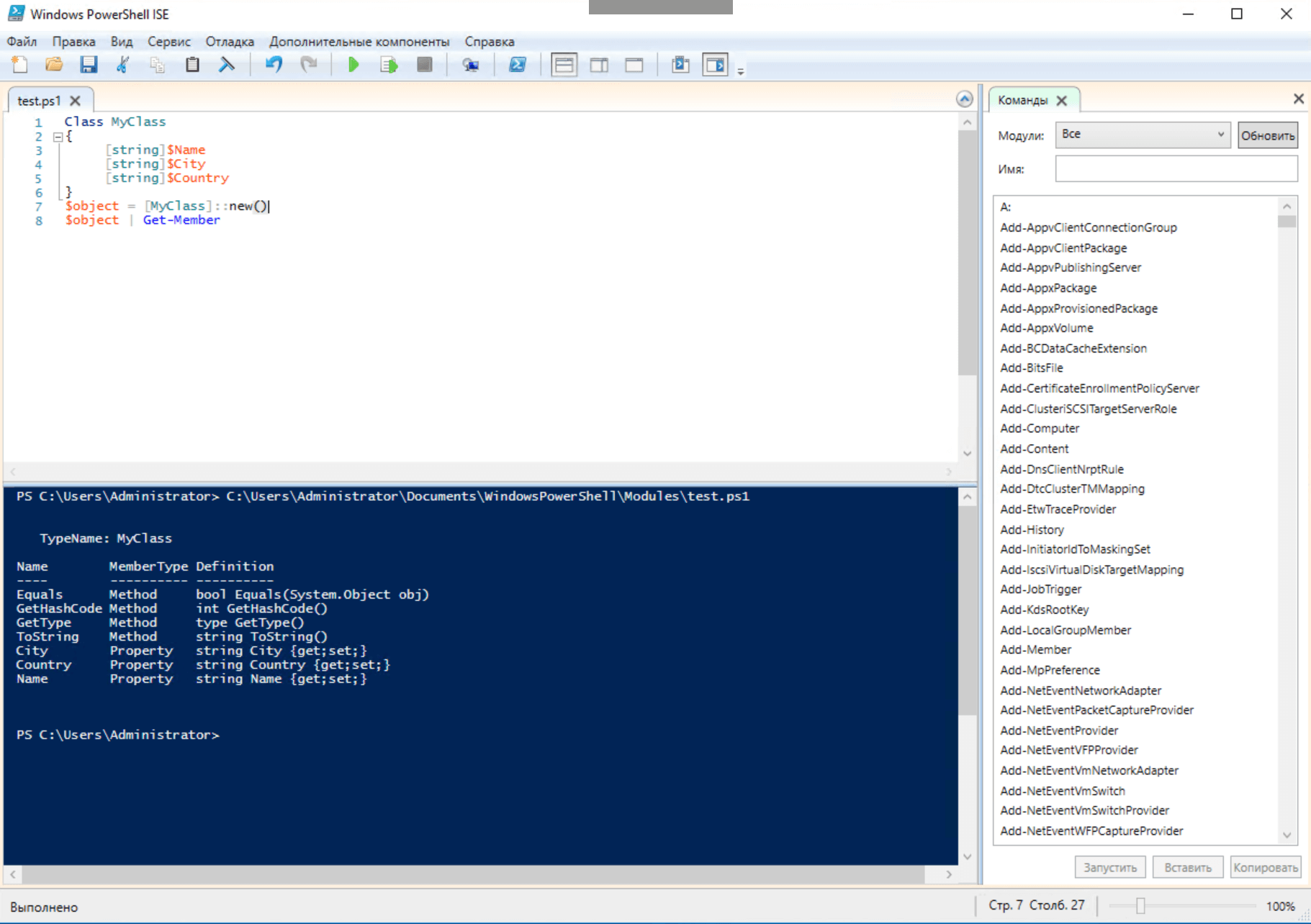
$object = New-Object -TypeName MyClass

или

$object = [MyClass]::new()

Проанализируем структуру объекта:

$object | Get-Member

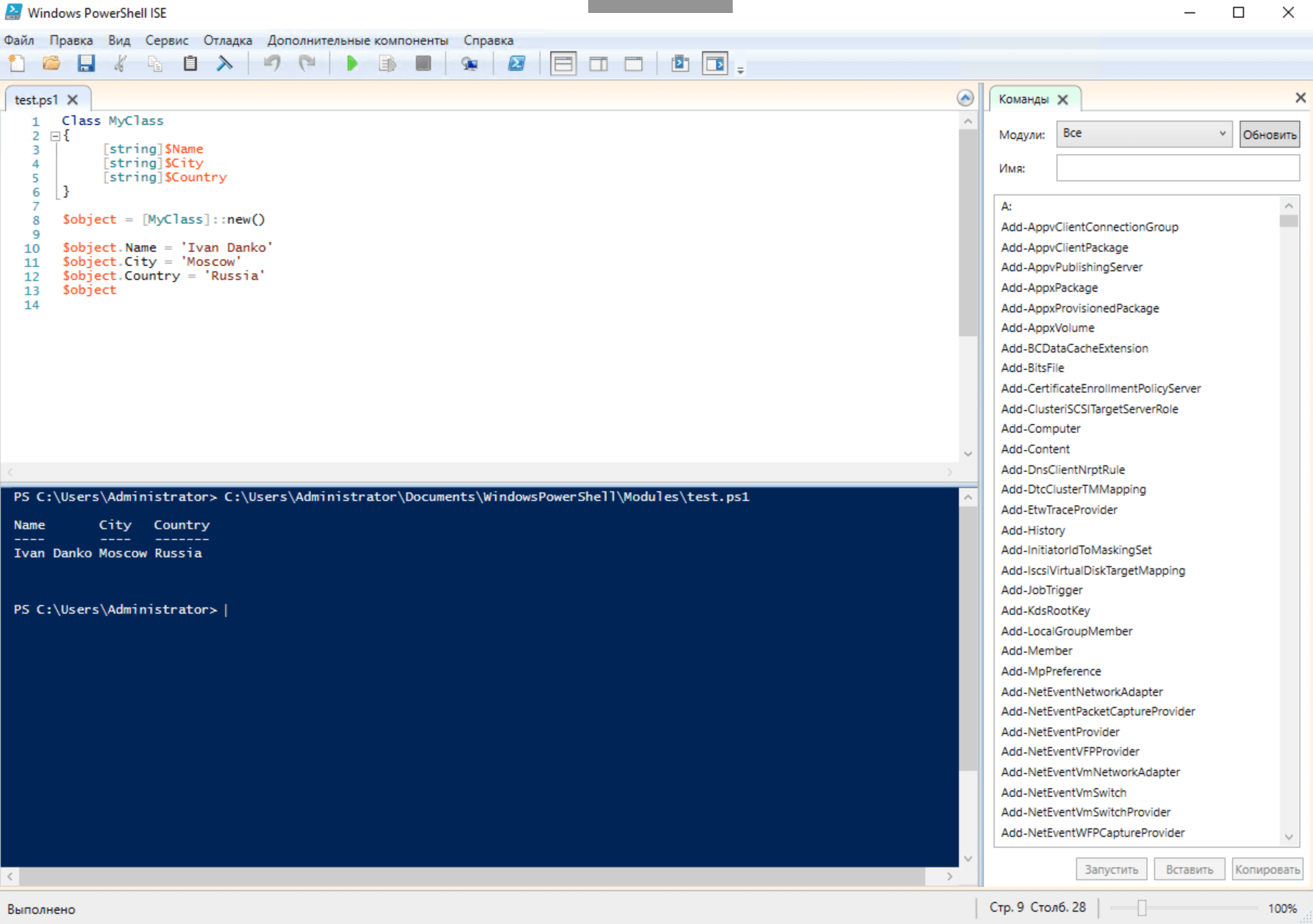
  
  
Не стоит забывать про область видимости: нельзя ссылаться на имя типа в виде строки или использовать литерал типа за пределами скрипта или модуля, в котором определен класс. При этом функции могут возвращать экземпляры класса (объекты), которые будут доступны вне модуля или скрипта.  
  
После создания объекта заполним его свойства:

$object.Name = 'Ivan Danko'

$object.City = 'Moscow'

$object.Country = 'Russia'

$object

  
  
Отметим, что в описании класса задаются не только типы свойств, но и их значения по умолчанию:

class Example

{

[string]$Name = 'John Doe'

}

Описание метода класса напоминает описание функции, но без использования служебного слова function. Как и в функции, в методы при необходимости передаются параметры:

class MyClass

{

[string]$Name

[string]$City

[string]$Country

*#описание метода*

Smile([bool]$param1)

{

If($param1) {

Write-Host ':)'

}

}

}

Теперь представитель нашего класса умеет улыбаться:

$object = [MyClass]::new()

$object.Smile($true)

Методы можно перегружать, кроме того у класса бывают [статические свойства и методы](https://habr.com/ru/company/ruvds/blog/494978/#section8), а также конструкторы, имена которых совпадают с именем самого класса. Определенный в скрипте или модуле PowerShell класс может служить базовым для другого — так реализуется наследование. При этом в качестве базовых допускается использование существующих классов .NET:

class MyClass2 : MyClass

{

*#тело нового класса, базовым для которого является MyClass*

}

[MyClass2]::new().Smile($true)

Наше описание работы с объектами в PowerShell трудно назвать исчерпывающем. В следующих публикациях попробуем его углубить на практических примерах: пятая статья цикла будет посвящена вопросам интеграции PowerShell со сторонними программными компонентами. Прошлые части можно найти по ссылкам ниже.