**Иркутский национальный исследовательский технический университет**

**Институт информационных технологий и анализа данных**

**Методические указания**

к лабораторным работам по курсу   
«**Администрирование информационных систем**»

Редакция 3.2020

**Лабораторная работа №4**

Язык сценариев ОС Microsoft Windows «Power Shell»

**Цель работы:** Освоить язык сценариев «Power Shell».

**Задание:**

1. Изучить теоретическую информацию о языке сценариев Power Shell.

2. Написать 2-3 простейших сценария, оформив их в виде скрипта. Варианты сценариев студент придумывает сам.

Примеры сценариев:

* Копирование нескольких файлов на сменный носитель в определенное время (учитывать возможность изменения буквы диска).
* Изменения прав доступа по шаблону в папке студента.
* Отобразить список служб на компьютере пользователя.
* Отсортировать все службы на компьютере пользователя на запущенные и остановленные.
* Создать подкаталог текущего каталога и скопировать в него файлы с расширениями PAS, BAS, C и TXT, присвоив им атрибут «только для чтения»;
* В заданном каталоге выдать на экран список имен файлов с расширениями PAS,TXT, BAT.
* Отсортировать указанный файл текущего каталога по убыванию 2-го символа и выдать его на экран.

3. Написать отчет с приложением листинга скрипта.

**Требования к отчету:** Отчетный документ представляется в печатном или электронном виде, в форматах doc(x).

Отчет оформляется согласно действующему нормоконтролю и содержит следующие главы:

1. Введение

2. Краткая информация о языке.

3. Описание всех действий и листинг скрипта с комментариями.

4. Результаты работы скрипта, со скриншотами.

5. Заключение (с анализом возможностей Power Shell).

## 1. Что способствовало появлению Windows PowerShell?

До появления PowerShell существовали (и существуют) следующие инструменты для автоматизации и администрирования сервисов: командная строка Windows и Windows Script Host. Но у этих технологий есть недостатки.

У командной строки Windows есть и возможность выполнять какие-то административные задачи и возможность записать алгоритм действий, сохранив его в виде скрипта (bat-файла), при этом можно использовать некие элементы программирования, например, использовать переменные, условные конструкции и даже что-то вроде циклов.

Большинство программных продуктов имеет консольный интерфейс, т.е. мы можем управлять программой, используя командную строку, при этом экономя ресурсы за счет отсутствия затрат на работу графического интерфейса. Компания Microsoft для серверной операционной системы Windows Server даже выпускает редакции без графического интерфейса, но всего этого недостаточно, так как возможности командной строки ограничены, т.е. написать какую-то сложную логику для автоматизации чего-либо мы не сможем, а если и сможем, то на это нам потребуется время и знания.

Также недостатком командной строки является то, что для полноценного управления сервером мы используем различные консольные утилиты программных продуктов, и поэтому использовать командную строку для администрирования на самом деле сложно, с точки зрения того, что у всех этих утилит разный синтаксис, параметры, методы использования и для того чтобы выполнять типовые задачи, т.е. каждодневные, приходиться держать все это в памяти, а для выполнения нетиповых задач приходиться сначала почитать справку или документацию.

Технология Windows Script Host позволяет выполнять все административные задачи, что и командная строка, включая их автоматизацию путем написания WSH скриптов, но здесь мы уже можем использовать полноценные языки программирования (**VBScript и JScript**), т.е. можно реализовывать сложную логику и алгоритмы. К тому же с помощью WSH мы управляем программными продуктами через объектный интерфейс, другими словами Windows Script Host намного «круче» чем командная строка. Но данная технология также не стала тем идеальным инструментом администрирования и автоматизации этого администрирования для системных администраторов, так как Windows Script Host требовал знаний вышеперечисленных языков программирования, что для системных администраторов на самом деле лишнее. Администраторам нужно всего лишь простой инструмент администрирования с возможностью запрограммировать какие-то действия, а углубляться в объектные модели программных продуктов на языках программирования VBScript и JScript им не хочется.

В итоге компании Microsoft необходимо было разработать такой инструмент администрирования для системных администраторов, который бы на 100 процентов удовлетворял все потребности сисадминов как в плане возможностей администрирования и автоматизации, так и в плане удобства и простоты использования, таким образом, появился Windows PowerShell.

## 2. Что такое Windows PowerShell?

**Windows PowerShell** – это язык сценариев и командная оболочка Windows, которые разработаны для администрирования и конфигурирования операционных систем Windows. PowerShell разработан на основе среды CRL и платформы .NET Framework и в отличие от командной строки, которая принимает и возвращает текст, Windows PowerShell работает с объектами. У каждого объекта в PowerShell есть свойства и методы, которые можно использовать для управления этими объектами.

В Windows PowerShell Microsoft разработала концепцию командлетов (*cmdlets*), которая представляет собой систему именования команд «*Глагол-Существительное*». Данная система позволяет системным администраторам быстрей освоить и упростить работу с PowerShell.

**С помощью Windows PowerShell можно:**

* Получать доступ к файловой системе;
* Управлять реестром;
* Управлять службами;
* Управлять процессами;
* Настраивать операционную систему;
* Устанавливать программное обеспечение;
* Устанавливать роли и компоненты сервера;
* Осуществлять администрирование и конфигурирование ролей и компонентов сервера;
* Писать и использовать сценарии для автоматизации управления и администрирования;
* Выполнять другие задачи системных администраторов.

Windows PowerShell содержит многие часто используемые утилиты и команды, запускаемые из командной строки, например ipconfig, ping и другие. Сделано это для того, чтобы облегчить переход системных администраторов с командной строки на PowerShell.

Также для удобства многие часто используемые команды и утилиты в PowerShell имеют синонимы (*Alias*), например cls - это синоним командлета Clear-Host, dir синоним Get-ChildItem (*полный список синонимов можно посмотреть путем запуска командлета Get-Alias*).

Для упрощения поиска нужной команды в PowerShell есть специальный командлет Get-Command, с помощью которого можно осуществлять поиск, как по глаголу, так и по существительному. Все команды в Windows PowerShell сгруппированы в модули (*например, Hyper-V, NetTCPIP*), что также упрощает поиск нужной команды.

После того как нужная команда найдена, можно посмотреть инструкцию по работе с этой командой, т.е. справку, для этих целей есть специальный командлет Get-Help, например следующая команда покажет справку по командлету Get-Command:

Get-Help Get-Command

Справка в Windows PowerShell может быть краткой, детальной (*параметр -Detailed*), полной (*параметр -Full*), а также можно выводить только примеры (*параметр - Examples*). Следующая команда покажет только примеры использования командлета Get-Command:

Get-Help Get-Command -Examples

Справка PowerShell обновляемая, т.е. ее можно обновить командой Update-Help.

## 3. Версии Windows PowerShell

Первая версия PowerShell 1.0 появилась 14 ноября 2006 года и выпускалась в виде отдельного дистрибутива, который можно было установить на следующие версии операционных систем Windows: Windows XP Service Pack 2, Windows Server 2003 Service Pack 1 и Windows Vista.

В Windows Server 2008 PowerShell 1.0 поставлялся в виде компонента, который также нужно было устанавливать.

Начиная с Windows 7 и Windows Server 2008 R2, PowerShell поставляется как неотъемлемый компонент системы (*т.е. предустановленный, устанавливать его не надо*). Ниже представлена таблица соответствия версии PowerShell и версии операционной системы Windows (*т.е. какая версия PowerShell по умолчанию установлена в той или иной версии Windows*):

|  |  |
| --- | --- |
| **Версия PowerShell** | **Версии Windows** |
| PowerShell 2.0 | Windows 7, Windows Server 2008 R2 |
| PowerShell 3.0 | Windows 8, Windows Server 2012 |
| PowerShell 4.0 | Windows 8.1, Windows Server 2012 R2 |
| PowerShell 5.0 | Windows 10, Windows Server 2016 |

С каждой новой версией PowerShell становится все более мощным инструментом администрирования, для сравнения в первой PowerShell было около 130 командлетов, а в PowerShell 5.0 их уже более 600!

Узнать текущую версию PowerShell можно с помощью свойства PSVersion встроенной переменной $PSVersionTable, например, выполните следующую команду:

$PSVersionTable.PSVersion

Или запустите командлет

Get-Variable -Name PSVersionTable –ValueOnly

где, значение PSVersion и будет версией PowerShell.

## 4. Язык PowerShell

**PowerShell** – это объектно-ориентированный скриптовой язык программирования. Он используется для написания команд управления всеми компонентами операционной системы Windows в оболочке Windows PowerShell, а также для написания сценариев автоматизации задач администрирования в интегрированной среде сценариев Windows PowerShell (ISE). Язык PowerShell хоть и создан для задач администрирования, он является полноценным скриптовым языком программирования, так как имеет программные конструкции, которые присутствуют в каждом языке программирования, такие как: условия, циклы, обработка ошибок, работа с переменными, объектами, массивами.

Язык PowerShell имеет единый синтаксис написания команд и структуру именования этих команд по принципу «Глагол-Существительное», что делает данный язык интуитивно понятным как для программистов, так и для системных администраторов.

Более подробно о программировании на данном языке можете посмотреть в материале - [Программирование на языке PowerShell](http://info-comp.ru/programmirovanie/547-programming-in-powershell.html).

http://info-comp.ru/programmirovanie/547-programming-in-powershell.html

## 5. Оболочка Windows PowerShell

**Оболочка Windows PowerShell** – это среда выполнения команд и сценариев на языке PowerShell. Данная оболочка имеет те же возможности что и командная строка такие как: хранение истории выполнения команд, настройка внешнего вида оболочки, завершение выполнения команд сочетанием клавиш Ctrl+C, а также много других возможностей, которых нет в оболочке командной строки, например такая замечательная возможность как «*подсветка синтаксиса*» (*появилась в PowerShell 5.0*).

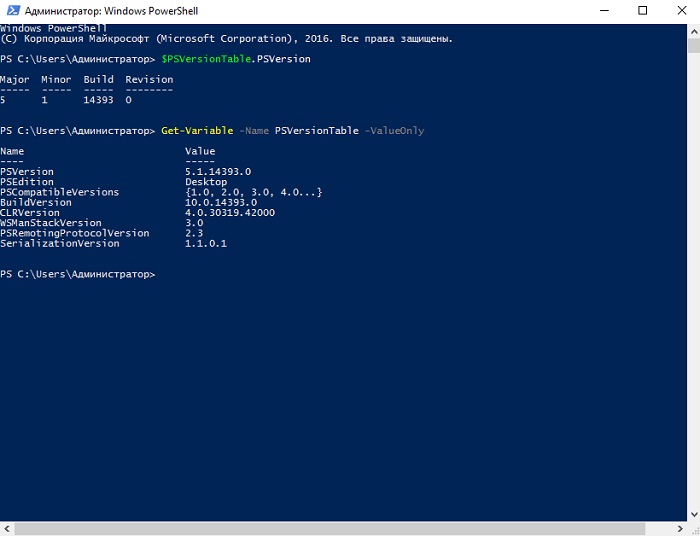
Запустить оболочку PowerShell можно несколькими способами, например:

* Из командной строки, набрав PowerShell;
* Через диалоговое окно «*Выполнить*» (*сочетание клавиш Win+R*), также набрав PowerShell;
* В Windows 7 - Пуск->Все программы ->Стандартные ->Windows PowerShell -> Windows PowerShell;
* В Windows 8.1 или Windows Server 2012 R2 - Пуск->Все программы ->Служебные ->Windows PowerShell;
* В Windows 10 или Windows Server 2016 - Пуск->Все программы -> Каталог Windows PowerShell (*в группе W*) -> Windows PowerShell.

*Пример запуска PowerShell в Windows Server 2016*

## Скриншот 2

Скриншот оболочки PowerShell в Windows Server 2016



## 6. Командлеты в PowerShell

**Командлет** (*cmdlet*) – это команда Windows PowerShell, с помощью которой можно осуществлять взаимодействие с объектами операционной системы с целью их управления. Данные команды являются частью языка PowerShell. Командлеты построены по принципу «*Глагол-Существительное*», разделенные дефисом (-); другими словами, мы сначала указываем, что делать, а через дефис - над чем. Например, командлет Get-Help, где Get - это глагол, означающий «*Получить*», а Help - существительное «*Помощь*» в контексте PowerShell «*Показать – Справку*». Командлеты PowerShell возвращают результаты в виде объектов, что является одним из главных отличий от командной строки Windows, в которой команды возвращают только текст на экран.

Кроме командлетов на получение данных (Get), существуют и такие типы командлетов как:

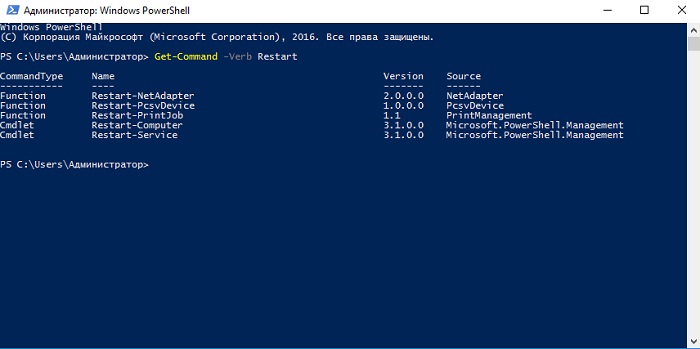
* Add – добавление данных;
* Clear – очистить;
* Enable – включить;
* Disable – выключить;
* New – создать;
* Remove – удалить;
* Set – задать;
* Start - запустить;
* Stop – остановить;
* Export – экспортировать;
* Import – импортировать;
* И еще много других.

Полный список командлетов в Windows PowerShell можно посмотреть с помощью специального командлета Get-Command. Например, запустите его с параметром -CommandType cmdlet, в итоге на экране у Вас отобразится список командлетов.

Get-Command -CommandType cmdlet

Как Вы уже поняли, у командлетов есть параметры, с помощью которых мы можем конкретизировать действия командлета. Параметры бывают обязательные и необязательные, например, у командлета Get-Command обязательных параметров нет.

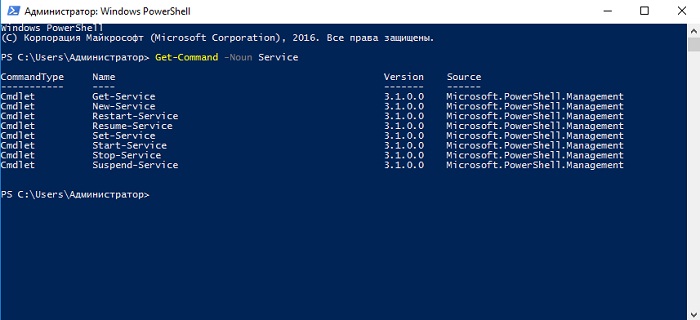
Ниже на картинке представлен способ поиска командлета по глаголу (*параметр Verb*). В данном случае у нас отобразился список командлетов, которые умеют что-то перезапускать.



*Команда*

Get-Command -Verb Restart

Для поиска командлета по существительному необходимо использовать параметр Noun. Например, ниже мы получили список командлетов, которые работают со службами.



*Команда*

Get-Command -Noun Service

Если Вы не нашли нужный командлет по полному названию можете использовать маску в формате \*Текст\*.

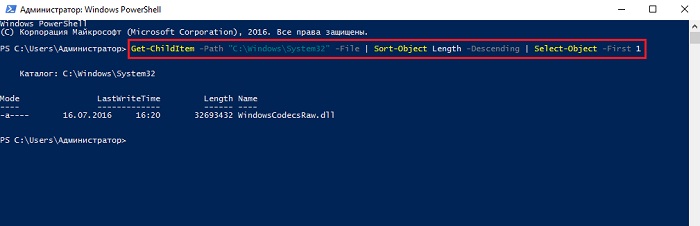
## 7. Конвейер в PowerShell

Одной из главных возможностей Windows PowerShell является возможность использования конвейера при выполнении команд.

**Конвейер** – это передача результата работы командлета через вертикальную черту (|) другому командлету. При этом, как Вы помните, в PowerShell командлеты работают с объектами и возвращают объекты, соответственно по конвейеру передаются также объекты.

С помощью конвейера можно выполнять сложные задачи простым и удобным способом без необходимости написания сложных алгоритмов и сценариев.

Например, давайте узнаем название самого большого файла в каталоге «C:\Windows\System32» (простой пример конвейера).



Команда

Get-ChildItem -Path "C:\Windows\System32" -File | Sort-Object length -Descending | Select-Object -First 1

Где,

* Get-ChildItem – командлет получения объектов в указанном каталоге;
* Sort-Object – командлет для сортировки объектов, в нашем случае мы сортируем по размеру файла (length -Descending);
* Select-Object – командлет выбора нужных свойств объекта, в нашем случае мы выводим стандартные поля и только самый первый объект, т.е. большой файл (параметр -First 1).

Все командлеты отделены друг от друга вертикальной чертой. Результат работы каждого командлета передается на вход другому командлету, т.е. сначала мы получаем все объекты в указанном каталоге, затем сортируем полученный результат и в заключение выбираем самый первый объект.

## 8. Фоновое исполнение заданий

В Windows PowerShell есть возможность фонового исполнения заданий, она представляет собой механизм, с помощью которого можно запустить на выполнение команду (например, которая долго выполняется) в фоновом режиме, т.е. после запуска Вы возвращаетесь в текущую сессию и можете продолжить работу, не дожидаясь окончания работы команды. Данная возможность Вам понадобится тогда, когда у Вас возникнет необходимость запустить команду, работа которой занимает достаточно продолжительное время, а как Вы знаете, в течение этого времени сессия оболочки PowerShell блокируется до завершения работы команды, а Вам нужно продолжать работать.

Всеми заданиями, которые запущенны в фоновом режиме, Вы можете управлять, например, посмотреть список заданий, остановить работу задания, удалить задание, и, конечно же, посмотреть результат работы задания.

В Windows PowerShell для работы с фоновыми заданиями есть следующие командлеты:

* Start-Job – запустить фоновую задачу;
* Stop-Job – остановить фоновую задачу
* Get-Job – посмотреть список фоновых задач;
* Receive-Job – посмотреть результат выполнения фоновой задачи;
* Remove-Job – удалить фоновую задачу;
* Wait-Job – перевести фоновую задачу на передний план, для того чтобы дожидаться ее окончания.

Для запуска в фоновом режиме необходимо написать команду Start-Job, а в фигурных скобках {} команду или набор команд, которые необходимо выполнить в фоновом режиме.

Например, давайте запустим какую-нибудь задачу (показать список служб) в фоновом режиме, затем посмотрим на список фоновых заданий и выведем на экран результат работы нашего задания (т.е. список служб).

Запуск задачи в фоновом режиме

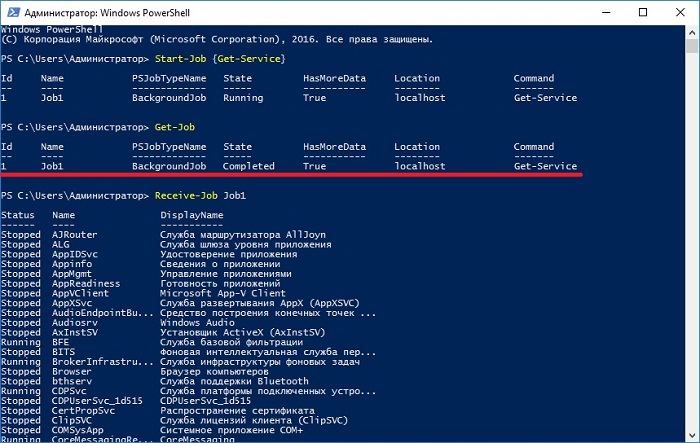
Start-Job {Get-Service}

Смотрим на список задач запущенных в фоновом режиме

Get-Job

Отображаем результат работы задания Job1

Receive-Job Job1



Как видим, у нас появилась задача со статусом «Completed», т.е. она уже выполнилась (просто Get-Service отрабатывает быстро).

Для того чтобы посмотреть результат работы фоновой задачи, т.е. командлета Get-Service, мы выполнили команду Receive-Job и передали ей имя задания (можно и значение идентификатора). В результате у нас отобразился список служб.

## 9. Удаленное управление на PowerShell

Windows PowerShell рассчитан не только на локальное использование, но и на удаленное выполнение команд. Данная возможность необходима, чтобы Вы могли со своего рабочего места управлять удаленными компьютерами, т.е. выполнять команды PowerShell.

Существует несколько способов удаленного управления:

* С помощью параметра **–ComputerName** (есть у многих команд). Другими словами Вы передаете имя компьютера, на котором необходимо выполнить команду, в качестве параметра. Способ обладает недостатком, так как ограничивается выполнением одной команды;
* С помощью сессий. Командлет **Enter-PSSession** (интерактивный сеанс). Таким способом Вы подключаетесь к удаленному компьютеру и все команды, которые Вы будете набирать в оболочке PowerShell, будут выполняться на удаленном компьютере так же, как если бы Вы набирали команды непосредственно на удаленном компьютере. Способ также обладает недостатком, так как сеанс ограничивается одним компьютером;
* С помощью командлета **Invoke-Command**. С помощью данного способа можно выполнять команды или сценарии как на одном компьютере, так и на нескольких.

Например, чтобы подключиться к удаленному компьютеру (в примере ниже ServerName) интерактивным сеансом выполните следующую команду:

Enter-PSSession ServerName

## 10. Сценарии, функции и модули в Windows PowerShell

Если Вы написали некий алгоритм, скорей всего Вы захотите сохранить его, для того чтобы в дальнейшем им пользоваться. В Windows PowerShell для этих целей существует механизм использования сценариев.

Суть данного механизма следующая, Вы пишите нужный Вам набор команд и сохраняете его в качестве сценария - это простой текстовый файл с расширением .PS1.

Затем, для того чтобы выполнить этот сценарий, Вы просто запускаете его в оболочке PowerShell. При этом Вам необходимо либо указать полный путь к файлу сценария, либо перейти в каталог со сценарием и вызвать его по имени.

**Важно!**

По умолчанию выполнение сценариев в Windows запрещено! Для того чтобы посмотреть политику выполнения сценариев выполните командлет **Get-ExecutionPolicy**. В результате он вернет действующую политику, например:

* Restricted – блокируется выполнение любых сценариев (значение по умолчанию);
* AllSigned – разрешено выполнение сценариев, которые имеют цифровую подпись;
* RemoteSigned – разрешено выполнение локальных сценариев, все скачанные сценарии должны иметь цифровую подпись;
* Unrestricted - разрешено выполнение любых сценариев (не рекомендуется, так как небезопасно!).

Для разрешения выполнения сценариев необходимо использовать командлет **Set-ExecutionPolicy** с одним из вышеперечисленных параметров.

Например, для разрешения выполнения локальных сценариев выполним следующую команду, и согласимся с внесением изменений, нажав Y.

Set-ExecutionPolicy RemoteSigned

В сценарии можно передавать параметры, делать их обязательными или задавать значение по умолчанию.

В Windows PowerShell предусмотрен механизм создания собственных функций, которые также как и встроенные командлеты можно будет использовать в оболочке PowerShell.

Для этого необходимо указать ключевое слово Function и затем в фигурных скобках {} написать алгоритм работы этой функции, т.е. набор команд (например, какая-нибудь часто используемая процедура: создать пользователя с определенными правами, очистить определенные каталоги и так далее). Потом необходимо сохранить все это в сценарий, но только уже с расширением .psm1, так как этот файл будет являться уже модулем.

Это еще не все, этот файл необходимо поместить в каталог, в котором PowerShell ищет модули. Таких каталогов несколько (специальный каталог в профиле пользователя, каталог, где установлен PowerShell), их можно посмотреть в переменных окружения PowerShell. Для этого выполните следующую команду

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

После того как Вы поместите написанный Вами модуль в специальный каталог, Вы сможете обращаться к своим функциям как к обычным командам PowerShell.

## 11. Интегрированная среда сценариев Windows PowerShell (ISE)

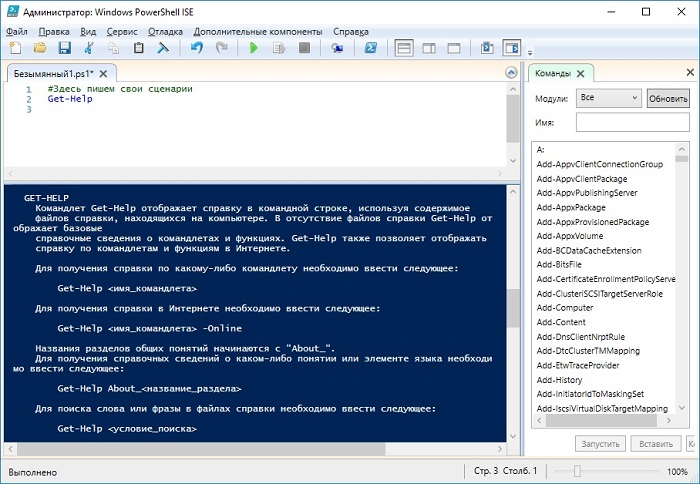
Для того чтобы было удобно писать сценарии, функции и соответственно модули, компания Microsoft разработала специальную графическую программу **Integrated Scripting Environment** (ISE) - интегрированная среда сценариев. Работать в этой программе очень удобно, к тому же она имеет мощный функционал (создание множества вкладок со сценариями, область вывода, встроенный отладчик и другое).

Запустить ее можно следующим образом:

* В Windows 7 - Пуск->Все программы ->Стандартные ->Windows PowerShell -> Windows PowerShell ISE;
* В Windows 10 или Windows Server 2016 - Пуск->Все программы -> Каталог Windows PowerShell (в группе W) -> Windows PowerShell ISE.

**Примечание!** ISE не будет работать на системе Windows Server, установленной в варианте Server Core.

Скриншот интегрированной среды сценариев PowerShell (ISE) в Windows Server 2016



<http://remontcompa.ru/851-scenarii-v-windows-powershell-20.html>