

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Брянский государственный технический университет

Утверждаю

Ректор университета

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Н. Федонин

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019

#### ОПЕРАЦИОННЫЕ СИСТЕМЫ

Оболочка и язык сценариев Windows PowerShell

Методические указания

к выполнению лабораторной работы №4

для студентов очной формы обучения

Брянск 2019

УДК 004.43

Оболочка и язык сценариев Windows PowerShell. [Текст] + [Электронный ресурс]: методические указания к выполнению лабораторной работы №4 для студентов очной формы обучения – Брянск: БГТУ, 2019. – 23 с.

Разработали:

К.В.Гулаков

канд. техн. наук, доц.

Е.В. Коптенок

# Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол № 8 от 2.06.2016)

Методические указания публикуются в авторской редакции

# ЦЕЛЬ РАБОТЫ

Целью работы является знакомство с оболочкой Windows PowerShell и изучение основ написания сценариев в ней.

Продолжительность работы – 2 часа.

# ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Windows PowerShell - расширяемое средство автоматизации от Microsoft, состоящее из оболочки с интерфейсом командной строки и сопутствующего языка сценариев. Впервые публично язык был продемонстрирован на PDC (Professional Developers Conference) в сентябре 2003 г. под кодовым названием «Monad». Версия 1.0 выпущена в 2006 году и сейчас доступна для Windows XP SP2/SP3, Windows Server 2003, Windows Vista, и встроена в Windows Server 2008 как опциональный компонент.Windows PowerShell 2.0 был выпущен в составе Windows 7 и Windows Server 2008 R2 как неотъемлемый компонент системы. Кроме того, вторая версия доступна и для других систем, таких как Windows XP SP3, Windows Server 2003 SP2, Windows Vista SP1 и Windows Server 2008.Windows PowerShell построен на базе Microsoft .NET Framework и интегрирован с ним. Дополнительно PowerShell предоставляет удобный доступ к COM, WMI и ADSI, равно как и позволяет выполнять обычные утилиты командной строки, чтобы создать единое окружение, в котором администраторы смогли бы выполнять различные задачи на локальных и удалённых системах.

Эти административные задачи обычно выполняются с помощью командлетов (в оригинале «cmdlets», произносится как «commandlets»), которые являются специализированными классами .Net. Пользователь может комбинировать их в скриптах (сценариях), используя различные конструкции, утилиты командной строки и обращения к обычным классам .NET (или WMI/COM объектам). Кроме того, можно использовать различные хранилища данных, такие как файловая система или реестр, которые предоставляются PowerShell'у посредством «поставщиков» (в оригинале providers).Windows PowerShell также предоставляет механизм встраивания, благодаря которому исполняемые компоненты PowerShell могут быть встроены в другие приложения. Эти приложения затем могут использовать функциональность PowerShell для реализации различных операций, включая предоставляемые через графический интерфейс. Этот подход применен в Microsoft Exchange Server 2007 для реализации управляющей функциональности в виде командлетов PowerShell и графических утилит управления в виде оболочек PowerShell, которые вызывают необходимые командлеты. Таким образом, графический интерфейс управления находится поверх промежуточного слоя - PowerShell. Другие приложения Microsoft, включая Microsoft SQL Server 2008, System Center Operations Manager и System Center Data Protection Manager также предоставляют доступ к своим интерфейсам управления через командлеты PowerShell.

Краткий обзор Powershell

Команды, исполняемые в Windows PowerShell, могут быть в форме командлетов, которые являются специализированными классами .NET, созданными с целью предоставления функциональности в PowerShell в виде сценариев PowerShell (\*.ps1) или являться обычными исполняемыми файлами. Если команда является исполняемым файлом, то PowerShell запускает её в отдельном процессе; если это командлет, то он исполняется внутри процесса PowerShell. PowerShell предоставляет интерфейс командной строки, в котором можно вводить команды и отображать выводимые ими данные в текстовом виде. Этот пользовательский интерфейс, базирущийся на стандартном механизме консоли Windows, предоставляет настраиваемый механизм автозавершения команд, но не обладает возможностью подсветки синтаксиса (хотя при желании её можно обеспечить). В PowerShell также можно создавать псевдонимы (alias) для командлетов, которые при вызове преобразуются в оригинальные команды. Кроме того, поддерживаются позиционные и именованные параметры для командлетов. При выполнении командлета работа по привязке значений аргументов к параметрам выполняется самим PowerShell, но при вызове внешних исполняемых файлов аргументы передаются им для самостоятельного разбора.

Другое понятие, используемое в PowerShell, - это «конвейер» (pipeline). Подобно конвейерам в Unix, они предназначены для объединения нескольких команд путём передачи выходных данных одной команды во входные данные второй команды, используя оператор. Но в отличие от аналога в Unix, конвейер PowerShell является полностью объектным, то есть данные между командлетами передаются в виде полноценных объектов соответствующих типов, а не как поток байтов. Когда данные передаются как объекты, содержащиеся в них элементы сохраняют свою структуру и типы между командлетами, без необходимости использования какой либо сериализации или посимвольного разбора (parsing) данных, как в случаях когда доступны лишь байтовые потоки. Объект также может содержать некоторые функции, предназначенные для работы с данными. Они также становятся доступными для получающего их командлета. Вывод последнего командлета в конвейере PowerShell автоматически передаёт на командлет Write-Host, который создаёт текстовое представление объектов и содержащихся в них данных и выводит его на экран.

Так как все объекты PowerShell являются объектами .NET, они содержат метод .ToString(), возвращающий текстовое представление данных объекта. PowerShell использует этот метод для преобразования объекта в текст. Кроме того, он позволяет указать правила форматирования, так что текстовое представление объектов может быть настроено. Однако, с целью поддержания совместимости, если в конвейере используется внешний исполняемый файл, то он получает текстовый поток, представляющий объект, и не интегрируется с системой типов PowerShell.

Расширяемая система типов (Extended Type System, ETS) PowerShell базируется на системе типов .NET, но реализует некоторые дополнения. Например, она позволяет создавать различные представления объектов, отображая лишь некоторые из их свойств и методов, а также применять специальное форматирование и механизмы сортировки. Эти представления привязываются к оригинальным объектам с помощью конфигурационных файлов в формате XML.

1.1 Командлеты

Командлеты (Cmdlets) - это специализированные команды PowerShell'а, которые реализуют различную функциональность. Это родные для PowerShell'а команды. Командлеты именуются по правилу Глагол-Существительное (Verb-Noun), например Get-ChildItem, благодаря чему их предназначение понятно из названия. Командлеты выводят результаты в виде объектов или их коллекций. Опционально командлеты могут получать входные данные в такой же форме и, соответственно, использоваться как получатели в конвейере. Хотя PowerShell позволяет передавать по конвейеру массивы и другие коллекции, командлеты всегда обрабатывают объекты поочередно. Для коллекции объектов обработчик командлета вызывается для каждого объекта в коллекции по очереди.

Командлеты являются специализированными классами .NET, экземпляры которых создаются в PowerShell и запускаются им при вызове. Командлеты наследуются от Cmdlet или от PSCmdlet, причём последний используется тогда, когда командлету необходимо взаимодействовать с исполняемой частью PowerShell (PowerShell runtime). В этих базовых классах оговорены некоторые методы - BeginProcessing(), ProcessRecord() и EndProcessing(), как минимум один из которых реализация командлета должна перезаписать для предоставления своей функциональности. Каждый раз при запуске командлета эти методы вызываются PowerShell'ом по очереди. Сначала вызывается BeginProcessing(), затем, если командлету передаются данные по конвейеру, - ProcessRecord() для каждого элемента, и в самом конце - EndProcessing(). Класс, реализующий Cmdlet, должен иметь один атрибут .NET - CmdletAttribute, в котором указываются глагол (Verb) и существительное (Noun), составляющие имя командлета. Популярные глаголы (рекомендуется использовать только их) представлены в виде перечисления (Enum).

Реализации командлетов могут вызывать любые доступные .NET API и могут быть написаны на любом языке .NET. PowerShell также предоставляет некоторые дополнительные API, такие как WriteObject(), которые необходимы для доступа к специфичной для PowerShell функциональности, например для вывода результирующих объектов в конвейер. Командлеты могут использовать API для доступа к данным напрямую или воспользоваться инфраструктурой поставщиков (Provider) PowerShell, которые позволяют обращаться к хранилищам данных через уникальные пути. Хранилища данных представляются через буквы дисков и иерархическую структуру внутри них (директории). Windows PowerShell поставляется с поставщиками для файловой системы, реестра Windows, хранилища сертификатов, а также для псевдонимов команд (alias:), переменных (variable:) и функций (functions:). Другие приложения тоже могут добавлять свои командлеты и поставщики для доступа к своим хранилищам данных.

В PowerShell V2 была добавлена возможность создания командлетов на самом PowerShell, без использования .NET языков.

Конвейер

В PowerShell, как и в оболочках Unix/Linux, присутствует конвейер. Этот конвейер служит для передачи выходных данных одного командлета во входные данные другого командлета. В частности, пользователь может вывести результаты командлета Get-Process в командлет Sort-Object (например, для сортировки процессов по дескрипторам (handles) ), затем в Where-Object, чтобы отфильтровать процессы, которые, например, занимают меньше 1 МБ страничной памяти, и в конце концов передать результаты в командлет Select-Object, чтобы выбрать только первые 10 процессов (по количеству дескрипторов).

Концепция конвейера изначально используется в Unix/Linux системах (см. Конвейер (UNIX)) , PowerShell отличается от них. В Unix/Linux вывод одной команды передаётся на следующий этап конвейера в бинарной форме, т.е. являет собой фактически поток данных. Рассмотрим пример Unix/Linux shell:

dd if=/dev/zero bs=1M count=1M | bzip2 -z9 -c > ex.bz2

поток "нулей" блоками по 1 МБ в количестве 1Миллиона раз (из устройства /dev/zero) командой dd (копирования специальных файлов) передаётся на ввод команды Bzip2, которая их сжимает максимально возможно (с точки зрения алгоритма сжатия bzip2, опция -z9) и результирующий поток передаёт на stdout (опция -с), который в свою очередь перенаправляется в файл ex.bz2. Результатом выполнения такой относительно короткой команды станет создание архива, внутри которого будет 1Т поток нулевых байтов. Сам процесс создания такого архива применяет в данном случае 2-а последовательных конвейера.

Реализовать подобную функциональность и гибкость в Windows средствами самой ОС Windows долгое время было практически невозможным. Заделать данную брешь в средах Windows и был фактически призван PowerShell являющийся неким подобием Unix shell например как Bourne shell.

# ХОД РАБОТЫ

Запуск PowerShell в Windows 10

Если у вас нет плитки PowerShell на начальном экране или ярлыка на рабочем столе, тогда вы можете открыть его с помощью поиска – это один из самых простых способов. Нажмите http://windowstips.ru/wp-content/uploads/2014/12/W8K7.png + Q, введите «powershell», а затем нажмите Enter.

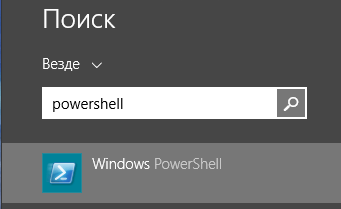


Рис.1. Запуск PowerShell

Также вы можете открыть PowerShell в разделе «Приложения» на начальном экране: перейдите на начальный экран и переместите курсор в левый нижний угол, чуть ниже последней плитки в первой колонке. Нажмите на значок со стрелкой, указывающей вниз. Затем нажмите на иконку PowerShell в разделе «Служебные».

Чтобы открыть PowerShell с помощью диалога «Выполнить», нажмите клавиши http://windowstips.ru/wp-content/uploads/2014/12/W8K8.png + R, введите «powershell» и нажмите Enter.

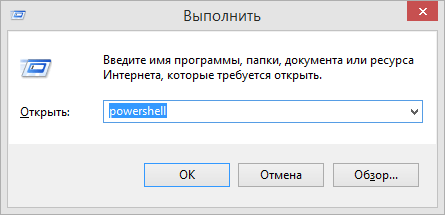


Рис.2. запуск PowerShellЗапуск PowerShell в Windows 7

В Windows 7 вы можете найти инструмент по ключевому слову «powershell» в меню «Пуск».

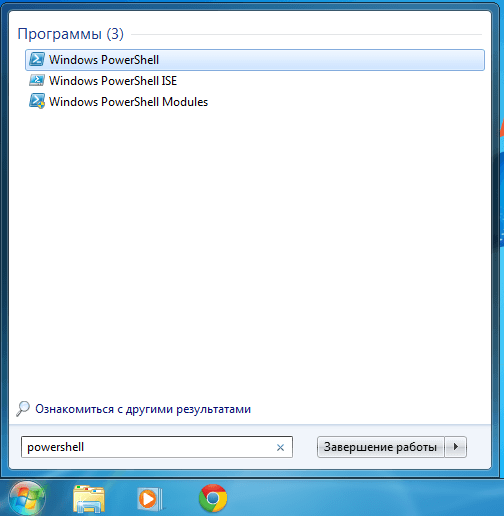


Рис.3. Открытие через поиск

Кроме того, открыть PowerShell через меню «Пуск» можно в разделе «Стандартные».

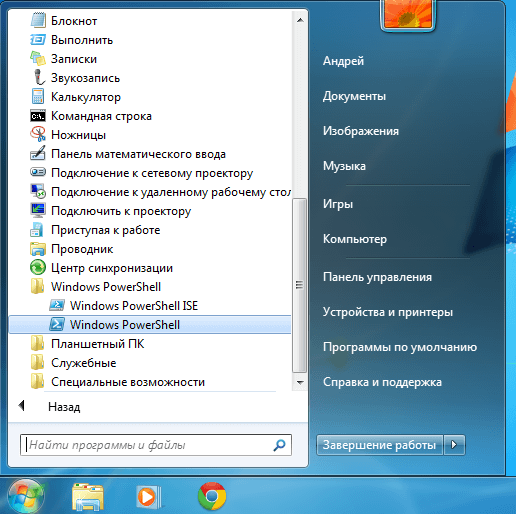


Рис.4. Открытие через меню «Пуск»

Очевидно, что упомянутая выше команда в окне «Выполнить» работает и в Windows 7.

Пользователям **других версий Windows** необходимо скачать и установить:

* Windows Management Framework 4.0;
* Windows PowerShell 4.0;
* Интегрированная среда сценариев Windows PowerShell;
* Веб-службы Windows PowerShell (расширение IIS OData для управления);
* Средство удаленного управления Windows (WinRM);
* Инструментарий управления Windows (WMI);
* Поставщик WMI диспетчера серверов;
* Новая функция, представленная только в версии 4.0, — настройка требуемого состояния (DSC) Windows PowerShell/

Синтаксис языка сценариев

## Переменные:

$var = 'hello'

$number = 1

$numbers = 1,2,3,4,5,6,7,8,9

Чтобы получить первый элемент, нужно использовать выражение $numbers[0], второй — $numbers[1], а последний элемент изображается так: $numbers[-1], предпоследний — $numbers[-2] и так далее.

## Кавычки:

Вместо знака доллара с именем переменной будет вставлено ее содержимое.

$name = 'Don'

$prompt = "My name is $name"

Также в тексте, ограниченном двойными кавычками, Windows PowerShell будет искать управляющий символ, или обратную одинарную кавычку, и действовать соответствующим образом. Вот несколько примеров:

$debug = "`$computer contains $computer"

$head = "Column`tColumn`tColumn"

В первом примере первый знак доллара $ отменяется. Обратная кавычка отменяет его значение как аксессора переменной. Если переменная $computer содержит строку «SERVER», тогда в $debug будет содержаться такая строка: «$computer contains SERVER».

Во втором примере «`t» представляет собой символ табуляции, поэтому Windows PowerShell разместит между словами Column знаки табуляции. Подробнее об управляющих символах см. веб-страницу.

Члены и переменные объектов

В Windows PowerShell все состоит из объектов. Даже простая строка, такая как «name», является объектом типа System.String. Чтобы узнать тип объекта (то есть, что представляет собой объект), а также его членов, к которым относятся его свойства и методы, достаточно передать объект в Get-Member по конвейеру:

$var = 'Hello'

$var | Get-Member

$svc = Get-Service

$svc[0].name

$name = $svc[1].name

$name.length

$name.ToUpper()

Параметры в сценарии

Параметры определяются особым образом в начале сценария.

param (

[string]$computername,

[string]$logfile,

[int]$attemptcount = 5

)

Несколько способов запуска сценария (предполагается, что сценарий содержится в файле Test.ps1):

./test -computername SERVER

./test -comp SERVER -log err.txt -attempt 2

./test SERVER err.txt 2

./test SERVER 2

./test -log err.txt -attempt 2 -comp SERVER

# Примеры команд

1. Выводит список файлов из папки C:\Windows\temp, измененных за последние 30 минут:

Get-ChildItem C:\Windows\temp\\*.\* | where {$\_.LastWriteTime -ge (Get-Date).AddMinutes(-30)}

2. Выводит список файлов из папки C:\Windows с расширением dll:

Get-ChildItem -Path C:\Windows | Where {$\_.extension -eq ".dll"}

3. Выполняется поиск архивов в папке D:\Backup, созданных после 1 мая 2011 года, размер которых находится в диапазоне 10-100 Мб.

Get-ChildItem -Path D:\Backup -Recurse -Include \*.zip | Where-Object -FilterScript {($\_.LastWriteTime -gt "2011-05-01") -and ($\_.Length -ge 10mb) -and ($\_.Length -le 100mb)}

4. Аналогом команды copy стал командлет Copy-Item. Для перезаписи целевого файла используется параметр Force.

Copy-Item -Path D:\Script\script-01.ps1 -Destination E:\Backup\27-09-2011\script-01.ps1 –Force

5. Копирование дерева папок выполняется той же командой, но с указанием ключа Recurse.

Copy-Item -Path D:\Script -Recurse E:\Backup\27-09-2011

6. При необходимости копирования определенных объектов, например только скриптов PowerShell-а вы можете задать фильтр.

Copy-Item -Filter \*.txt -Path D:\Script -Recurse E:\Backup\27-09-2011

7. Создание файлов и папок средствами PowerShell производится с помощью командлета New-Item.

Для создания папки придется указать тип элемента “Directory”, а для создания файла “File”

New-Item -Path ‘D:\Script\New Folder’ -ItemType "directory"

New-Item -Path ‘D:\Script\New Folder\script-02.ps1′ -ItemType “file”

8. Переименование элементов производится с помощью команды Rename-Item.

Rename-Item .\File-01.ps1 .\File-01.ps1.bak

9. Удаление производится с помощью Remove-Item.

Remove-Item d:\Backup\20-09-2011 -Recurse

10. Задача поиска в лог файлах определенного выражения по маске ip-адреса “192.168.100.253″ с последующим копированием найденного в папку D:\TEMP:

$current = Get-Date

$Days = "-30"

$start = $current.AddDays($days)

Get-ChildItem D:\Logs -Filter \*.log -Recurse | Where-Object {($\_.LastWriteTime.Date -ge $Start.Date) -and ($\_.LastWriteTime.Date -le $End.Date)} | Select-String "192.168.100.253" | Copy-Item -Destination D:\TEMP

11. Сценарий, когда требуется из определенной папки удалять все файлы и папки старше 30 дней. Удаляется все, что есть в папке и в подпапках т.е. удаление еще и самих папок.

$Path = "C:\temp"

$Days = "-30"

$CurrentDate = Get-Date

$OldDate = $CurrentDate.AddDays($Days)

Get-ChildItem $Path -Recurse | Where-Object { $\_.LastWriteTime -lt $OldDate } | Remove-Item

# Примеры скриптов

12. Функция разбора log-файла. Apache Log Parser for PowerShell

function Read-ApacheLog

{

param

(

[Parameter(Mandatory=$true)]

[string]

$Path,

[string]

$TimeStart,

[string]

$TimeFinish

)

Get-Content -Path $Path | Foreach {

# combined format

if ($\_ -notmatch "^(?<Host>.\*?) - (?<User>.\*?) \[(?<TimeString>.\*?)\] `"(?<Request>.\*?)`" (?<Status>.\*?) (?<BytesSent>.\*)")

{

throw "Invalidline: $\_"

}

$entry = $matches

$entry.Time = [DateTime]::ParseExact($entry.TimeString, "dd/MMM/yyyy:HH:mm:ss zzz", [System.Globalization.CultureInfo]::InvariantCulture)

if ($entry.Request -match "^(?<Method>.\*?) (?<Path>.\*?) (?<Version>.\*)$")

{

$entry.Method = $matches.Method

$entry.Path = $matches.Path

$entry.Version = $matches.Version

}

$TimeFrom = [DateTime]::Parse($TimeStart)

$TimeTo = [DateTime]::Parse($TimeFinish)

if(($entry.Time -gt $TimeFrom) -and ($entry.Time -lt $TimeTo))

{

return New-Object PSObject -Property $entry

}

}

}

Read-ApacheLog .\iipo.log1 "1.10.2014 09:06:44" "1.10.2014 10:06:45"

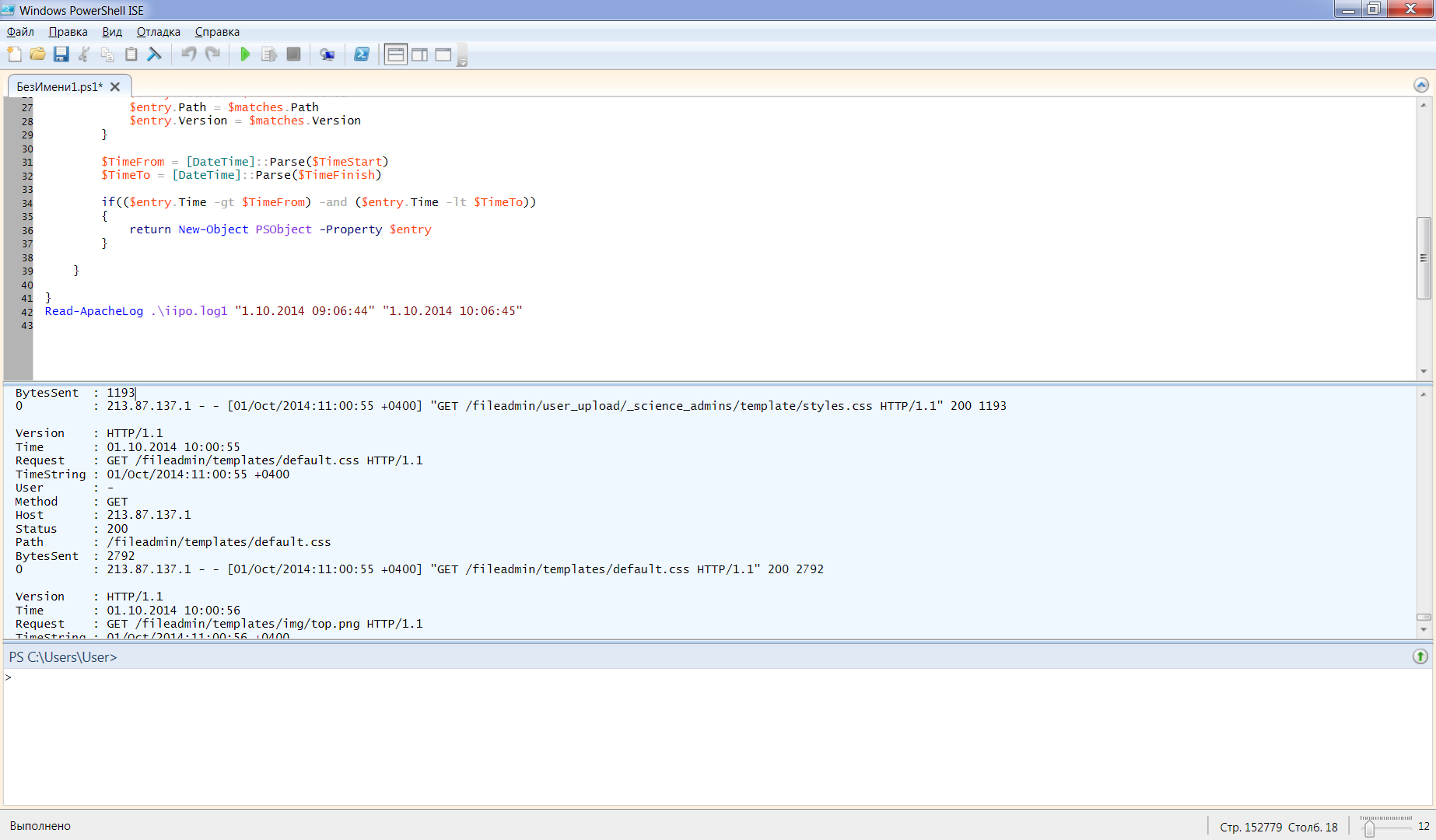


Рис.5. Результат выполнения скрипта

см. также

<https://www.simple-talk.com/sysadmin/powershell/powershell-data-basics-file-based-data/>

13. Создание отчёта об установленном программном обеспечении

Напишем небольшой скрипт создания html файл в c:\users\username, где будет отображаться список недавно установленных программ, сортированных по дате. Программы, установленные недавно будут отображаться в начале списка.

#переменная

$vUserName = (Get-Item env:\username).Value

$vComputerName = (Get-Item env:\Computername).Value

$filepath = (Get-ChildItem env:\userprofile).value

$name = (Get-Item env:\Computername).Value

## Html Style

$a = "<style>"

$a = $a + "BODY{background-color:Lavender ;}"

$a = $a + "TABLE{border-width: 1px;border-style: solid;border-color: black;border-collapse: collapse;}"

$a = $a + "TH{border-width: 1px;padding: 3px;border-style: solid;border-color: black;background-color:thistle}"

$a = $a + "TD{border-width: 1px;padding: 3px;border-style: solid;border-color: black;background-color:PaleGoldenrod}"

$a = $a + "</style>"

# удаленние старого отчёта, если существует.

if (test-Path $filepath\$name.html) { remove-Item $filepath\$name.html;

Write-Host -ForegroundColor white -BackgroundColor Red "Old file removed"

}

# Управление командой

ConvertTo-Html -Title "Software Information for $name" -Body "<h1> Computer Name: $name </h1>" > "$filepath\$name.html"

Get-WmiObject win32\_Product -ComputerName $name | Select Name,Version,PackageName,Installdate,Vendor | Sort Installdate -Descending `

| ConvertTo-html -Head $a -Body "<H2> Software Installed</H2>" >> "$filepath\$name.html"

$Report = "The Report is generated On $(get-date) by $((Get-Item env:\username).Value) on computer $((Get-Item env:\Computername).Value)"

$Report >> "$filepath\$name.html"

write-Host "file is saved in $filepath and the name of file is $name.html" -ForegroundColor Cyan

invoke-Expression "$filepath\$name.html"

## конец срипта

14. Опрос всех машин в заданном ip диапазоне

Второй скрипт будет полезен тем, кто хочет отследить присутствие той или иной машины в сети.

function Get-ComputerNameByIP {

param(

$IPAddress = $null

)

begin { }

process {

if ($IPAddress -and $\_) {

Throw `Please use either pipeline or input parameter'

break

} elseif ($IPAddress) {

([System.Net.Dns]::GetHostbyAddress($IPAddress))

} elseif ($\_) {

trap [Exception] {

write-warning $\_.Exception.Message

continue;

}

[System.Net.Dns]::GetHostbyAddress($\_)

} else {

$IPAddress = Read-Host “Please supply the IP Address”

[System.Net.Dns]::GetHostbyAddress($IPAddress)

}

}

end { }

}

function Check-Online {

param(

$computername

)

test-connection -count 1 -ComputerName $computername -TimeToLive 5 -asJob |

Wait-Job |

Receive-Job |

Where-Object { $\_.StatusCode -eq 0 } |

Select-Object -ExpandProperty Address StatusCode

}

$Start = Get-Date

$ips = 1..254 | ForEach-Object { "192.168.1.$\_" }

$online = Check-Online -computername $ips

$online

foreach ($PC in $online) {

Get-ComputerNameByIP $PC

}

$End = Get-Date

Write-Host "`nStarted at: " $Start

Write-Host "Ended at: " $End

# ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

**Задание 1.** Разобрать и запустить все примеры 1-14, приведённые в разделе «Примеры» в Windows PowerShell.

**Задание 2.** Выполнить согласно варианту:

1. Скопировать все имеющиеся в каталоге Windows исполняемые файлы в каталог WinEx на диске С.
2. Скопировать все имеющиеся в каталоге Windows растровые графические файлы в каталог WinGrafika на диске С.
3. Cоздать текстовый файл, содержащий список выполняемых процессов, упорядоченный по возрастанию (Список выводимых параметров процессов - Id, Имя процесса, WorkingSet,CPU; Сортировать по значению параметра-Id; Вывести процессы, у которых - CPU > 5).
4. Вывести в текстовый файл список свойств процесса, возвращаемый командлетом Get-process и на экран – их общее количество.
5. Разработать командлет для проверки наличия в текущем каталоге файлов одинакового размера. Если такие файлы есть – вывести их имена.
6. Разработать командлет для нахождения в каталоге windows и его подкаталогах имен файлов, записанных русскими буквами и имеющих расширение jpg.
7. Разработать командлет для нахождения в каталоге windows\system32 имен трех dll с самой ранней датой создания.
8. Разработать командлет для нахождения в заданном каталоге трех файлов наименьшего размера.
9. Скопировать все имеющиеся в каталоге Windows текстовые файлы в каталог Text на диске С.
10. Cоздать текстовый файл, содержащий список выполняемых процессов, упорядоченный по возрастанию (Список выводимых параметров Имя процесса, Id, PriorityClass, UserprocessorTime, TotalProcessorTime; Сортировать по значению параметра- TotalProcessorTime; Вывести процессы, у которых - BasePriority > 7).

# СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. А.Попов. Введение в Windows PowerShell
2. http://technet.microsoft.com/ru-ru/magazine/hh551144.aspx
3. <http://blog.wadmin.ru/2011/09/powershell-lessons-working-with-files-and-folders/>
4. http://technet.microsoft.com/en-us/library/bb978526.aspx
5. PowerShell как средство автоматического администрирования, автор: И.В. Коробко.
6. Windows PowerShell 2.0. Справочник администратора, автор: Уильям Р. Станек.
7. Microsoft Windows Command-Line: Administrator's Pocket Consultant, автор: Уильям Р. Станек.

Оболочка и язык сценариев Windows PowerShell [Электронный ресурс]: Методические указания к выполнению лабораторной работы №4 для студентов очной формы обучения по направлению подготовки 09.03.04 «Программная инженерия» – Брянск: БГТУ, 2019. – 23 с.

ГУЛАКОВ КОНСТАНТИН ВАСИЛЬЕВИЧ

КОПТЕНОК ЕЛИЗАВЕТА ВИКТОРОВНА

Научный редактор Д.Г. Лагерев

Компьютерный набор Е.В. Коптенок

Иллюстрации К.В. Гулаков

Подписано в печать \_\_.\_\_.\_\_. Усл.печ.л. 0,7 Уч.-изд.л. 0,7

Брянский государственный технический университет

241035, Брянск, бульвар 50 лет Октября, 7 БГТУ

Кафедра «Информатика и программное обеспечение», тел. 56-09-84

**Сопроводительный лист на издание в авторской редакции**

Название работы Оболочка и язык сценариев Windows PowerShell [Электронный ресурс]: Методические указания к выполнению лабораторной работы №4 для студентов очной формы обучения по направлению подготовки 09.03.04 «Программная инженерия»

Актуальность и соответствующий научно-методический уровень подтверждаю \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись научного редактора)

Рукопись сверена и проверена автором \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись автора)

Рекомендуется к изданию \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись заведующего кафедрой)