

# **Практическая работа № 1**

## **Работа в среде командной оболочки Microsoft PowerShell**

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## (справочный материал)

### 1. Назначение пакета PowerShell

Основной задачей любой операционной системы (ОС) является управление ресурсами компьютерной системы. Именно на основе этих функций управления создается сервис для пользователей. Эффективная профессиональная работа опытного пользователя с ОС компьютера немыслима без овладения интерфейсом, обеспечиваемым командной строкой. Этот вид интерфейса является одним из основных применительно к ОС Unix и Linux. Преимуществом данного интерфейса служит возможность «более углубленного» управления ресурсами системы, чем с помощью графического интерфейса ОС Windows.

Объяснением этого факта, по-видимому, служит следующее: Unix-системы ориентировались на работу профессионально подготовленных пользователей (операторов, программистов, системных инженеров). В них интерфейс командной строки всегда был и остается традиционно богатым и мощным.

Напротив, Windows-ориентированные системы ведут свое развитие от простых персональных компьютеров. Корпорация Microsoft при разработке для них ОС ориентировалась в первую очередь на пользователей-непрофессионалов и закладывала принцип «нулевого администрирования». Согласно этому принципу в различных версиях ОС Windows предусматривалось лишь минимальное участие пользователей в управлении и распределении ресурсов систем. Выполнение этих функций стало прерогативой программ самой ОС. Пользователи же довольствовались в основном сервисом графического интерфейса.

Интерфейс командной строки в ОС Windows присутствует и играет вспомогательную роль. В свое время он формировался как

некое подмножество ОС Unix и особого развития не получил. Однако интерфейс командной строки во многих нестандартных ситуациях остается единственным средством определения расогласований и «тонкой настройки» аппаратно-программных средств. Поэтому в новых версиях ОС Windows на рубеже нового тысячелетия с учетом роста сложности аппаратной и программной частей компьютерных систем добавлен ряд команд, позволяющих выполнять некоторые настройки. Часть команд, заимствованных из MS DOS, получила дополнительные возможности. Например, такие команды, как dir, copy, xcopy, rename и другие в новых редакциях ОС Windows работают с длинными именами файлов.

Начало нового столетия ознаменовалось дальнейшим усложнением структуры компьютеров: появились многоядерные микропроцессоры, повысилась роль распределенных вычислений, что явилось стимулом развития сетевых технологий (сетевые службы, центры обработки данных, «облачные» вычисления). Возможности старых средств стали недостаточными, потребовалась разработка новых средств администрирования и управления ресурсами компьютерных систем.

Windows PowerShell — командная оболочка следующего поколения и язык сценариев от фирмы Microsoft, которые можно использовать вместо устаревшего интерпретатора команд cmd.exe и языка сценариев VBScript. Разработчики Windows PowerShell к разработке нового средства подошли комплексно. Они не просто ввели в интерфейс командной строки набор новых утилит (команд), обеспечивающих возможность достаточно просто выполнять сложные процедуры управления ресурсами компьютерных систем, но и обеспечили совместимость с ранее созданными разработками, в том числе и разработками других платформ, например Unix ориентированными.

Главной задачей разработки PowerShell было создание среды составления сценариев, которая наилучшим образом подходила бы для современных версий ОС Windows и была бы более функциональной, расширяемой и простой в использовании, чем любой какой-либо аналогичный продукт для любой другой ОС. Компания Microsoft в настоящее время позиционирует эту оболочку как основной инструмент управления ОС и рядом разработанных ею приложений. PowerShell официально включен в качестве стандартного компонента во все новые версии ОС Windows, начиная с Vista.

Идея построения и развития принципиально новой командной оболочки основана на нескольких здравых и прозрачных положениях. Она должна стать мощным средством управления и настроек компьютерных систем. Исходными данными для этого является наличие большого количества ресурсов (объектов), необходимых для выполнения автоматических вычислений. Ресурсы могут быть локальными и сетевыми, привлекаемыми для организации вычислений. Каждый ресурс – это некоторый особый набор характеристик (свойств), каждая характеристика – это количественное или качественное выражение. В целях управления необходимо отслеживать состояние ресурсов не только в статике, но и в динамике вычислений. Нужно иметь средства, чтобы просматривать состояния ресурсов, уметь отбирать необходимую информацию о некоторых из них, отсеивать, сортировать, фильтровать сведения, проводить анализ и обработку отобранных данных и предоставлять результаты пользователям (клиентам и администраторам компьютерных систем). Творческий характер процедур анализа и обработки требует включения в оболочку средств разработки программ-функций. Результаты анализа и обработки должны использоваться в управлении. Простейшим видом управления служит выдача справок о состоянии объектов управления. В более сложных случаях результаты

должны использоваться для корректировки и изменения состояний ресурсов, а режим управления становится автоматизированным или даже автоматическим.

По существу новая командная оболочка представляет собой весьма специфичную базу данных, отражающую ресурсы компьютерных систем, а также и систему управления этой базой данных (СУБД) и ресурсами. СУБД включает все необходимые атрибуты: язык описания структуры и данных, язык запросов, язык манипулирования данным, генератор отчетов и т. д. Правда, некоторые из этих категорий представлены в неявном виде. Самым интересным здесь является то, что база данных с ресурсами и СУБД включаются в контур управления реальных компьютерных систем в автоматизированном режиме.

Отличительной особенностью построения PowerShell служит ее ориентация на объектную модель платформы .NET. Именно средства этой модели обеспечивают возможность взаимодействия различных операционных систем друг с другом. Кроме того, объектная .NET является самодокументируемой, т. е. каждый ее объект содержит информацию о своей структуре. Это свойство очень важно при интерактивной работе пользователя, поскольку все необходимые сведения о привлекаемых компонентах находятся «под рукой».

### **1.1. Начало работы в среде PowerShell**

PowerShell включена во все новые версии ОС Microsoft Windows, начиная с Vista. Если на компьютере пользователя PowerShell отсутствует, то необходимо сначала установить платформу .NET. После этого можно установить и собственно оболочку PowerShell с учетом версий и языка представления справок по системе.

Запуск оболочки осуществляется по одному из трех вариантов:

1. Осуществить поиск в «Приложениях» и запустить PowerShell.

2. Нажать кнопку Пуск, открыть Все программы, найти и выбрать Windows PowerShell.
3. Нажать кнопку Пуск, выбрать пункт Выполнить, ввести имя файла PowerShell, нажать кнопку ОК.

После запуска PowerShell открывается командное окно оболочки с приглашением ввода команд (рис. 1.1).

```
Windows PowerShell
(C) Корпорация Майкрософт, 2009. Все права защищены.
PS C:\Users\user> _
```

Рис. 1.1. Командное окно оболочки PowerShell

Следует обратить внимание на вид строки приглашения. Она очень похожа на строку приглашения «cmd.exe», но в ее начале стоят буквы PS, указывающие на принадлежность к оболочке PowerShell.

Для выхода из среды PowerShell автономного компьютера можно набрать команду «exit» и нажать клавишу «Enter» или просто закрыть окно оболочки, но такой способ выхода не является корректным, так как данные проведенного пользователем процесса будут потеряны. При работе в компьютерной сети, с сетевыми ресурсами и с сервером можно завершать сеанс активного пользователя двумя способами:

1. Командой logoff без параметров.
2. Командой shutdown -1, т. е. вызовом утилиты «shutdown.exe» с параметром -1.

Следует ознакомиться со справочной информацией по данным завершения. Справки вызываются по командам logoff /? и shutdown /?.

Разработчики Windows PowerShell предполагали, что большинство пользователей этого средства будут работать с ним в интерактивном режиме. Ввод команд можно выполнять как по

отдельности, так и группировать их в конвейеры или в пакетные файлы.

При разработке новой оболочки командной строки разработчики постарались учесть все лучшее из накопленного опыта в различных ОС. Они пытались сохранить не только преемственность между прежними и новыми системами, но и предоставить возможность работы администраторам вычислительных систем в привычном для них интерфейсе. Поэтому новым средствам управления командной строки можно назначать дополнительные имена (псевдонимы). Это обеспечивает, например администраторам Unix-ориентированных систем, использование новой среды в привычных для них терминах, что облегчает изучение и применение PowerShell.

При первых сеансах работы рекомендуется посмотреть и сравнить результаты выполнения нескольких команд, уже известных пользователю, например по работе с интерпретатором команд «cmd.exe». Практически все команды интерпретатора имеют аналоги с теми же именами (псевдонимами), но представление данных отличается, иногда очень значительно. Прежде всего, следует отметить уровень детализации информации. Посмотрим результат выполнения команды `dir` в среде PowerShell (см. рис. 1.2).

```
Windows PowerShell
(C) Корпорация Майкрософт, 2009. Все права защищены.

PS C:\Users\user> dir

Каталог: C:\Users\user

Mode                LastWriteTime         Length Name
----                -
d-r--              25.04.2014         9:01     Contacts
d-r--              15.03.2019        12:50     Desktop
d-r--              27.02.2019        13:09     Documents
d-r--              04.02.2019        12:29     Downloads
d-r--              07.03.2017        10:36     Favorites
d----              14.01.2014        16:41     Library
d-r--              25.04.2014         9:01     Links
d-r--              25.04.2014         9:01     Music
d-r--              31.01.2019        11:00     Pictures
d-r--              25.04.2014         9:01     Saved Games
d-r--              25.04.2014         9:01     Searches
d-r--              25.11.2015        11:43     Videos

PS C:\Users\user> _
```

Рис. 1.2. Результат выполнения команды «dir»

В среде PowerShell имеется специфичная команда (командлет) `Get-ChildItem`, которая также имеет дополнительное имя (псевдоним) `dir`. Работа этой команды без параметров представлена на рис. 1.2. Приверженцам ОС Unix и Linux более привычным является использование псевдонима этого командлета `ls` (лист) с теми же функциями. В новой редакции команды `dir` появился столбец `Mode`, отражающий возможные режимы использования программных средств, очень похожие на режимы Unix. Режимы определяются отдельно для каталогов и файлов. Полный перечень характеристик, выводимых по различным командам, можно посмотреть с помощью командлета `Get-Member`.

Командная строка PowerShell кроме набора и выполнения команд предоставляет пользователю возможность вычислений арифметических выражений различной сложности. В простейшем случае она обеспечивает вычисления как калькулятор. После записи выражения в командной строке и нажатия клавиши «Enter» результат вычисления отображается на следующей строке. Несколько простых примеров приведено на рис. 1.3.

```
PS C:\Users\user> 150 / 4 - 23
14,5
PS C:\Users\user> (15 - 41) * 5
-130
PS C:\Users\user> 221 / 33
6,6969696969697
PS C:\Users\user> _
```

Рис. 1.3. Результат вычисления арифметических выражений

В более сложных случаях выражения могут включать различные математические функции. Их реализация обеспечивается путем обращения к библиотекам классов платформы .NET, в частности к методам класса `System.Math`.



При сложных вычислениях может потребоваться сохранение промежуточных результатов в каких-то ячейках памяти. Для этого следует простыми средствами определить имя переменной и определить ее значение. Имена переменных должны начинаться знаком \$. Запись только имени переменной после знака доллара означает обращение к выдаче ее значения (рис. 1.4).

```
PS C:\Users\user> $gamma = 25 / 7
PS C:\Users\user> $psi = 6
PS C:\Users\user> $alfa = $gamma + $psi
PS C:\Users\user> $alfa
9,57142857142857
PS C:\Users\user>
```

Рис. 1.4. Результат вычислений, с сохранением промежуточных результатов

На первых сеансах работы пользователей будет полезно использование команд-псевдонимов `cls` (очистка экрана дисплея) и `cd` (изменение каталога), аналогичных по работе с интерпретатором команд `cmd.exe`. Функциональность этих команд остается прежней.

## 1.2. Структура пакета PowerShell и его справочная система

Разносторонний вид ресурсов компьютерных систем и специфический характер управления каждым из них не позволили разработчикам создать единую систему управления ресурсами с четкой и строгой структурой. В связи с постоянным усложнением компьютеров, а также систем и сетей на их основе состав средств управления не может оставаться постоянным, он должен совершенствоваться, пополняться, адаптироваться к новым условиям построения и применения информационных систем. Поэтому разработчики решили сделать новую оболочку предельно простой и документированной. В интерактивном режиме пользователь-администратор всегда может посмотреть, какие средства и в каком режиме он может использовать для достижения определенных целей.

Изучение оболочки лучше начинать с уяснения структуры и возможностей справочной системы. Для этого целесообразно сначала ознакомиться с функциями команды (командлета) Get-Help, обеспечивающей получение справочных данных по всем подсистемам PowerShell с различной детализацией. Именно здесь указываются первые сведения о принципах построения новой оболочки и сведения о делении командлетов на группы. Наберем в командной строке фразу get-help или get-help -?, можно также воспользоваться псевдонимом help без параметров (рис. 1.5).

```
PS C:\Users\user> Get-help
РАЗДЕЛ
    Get-Help

КРАТКОЕ ОПИСАНИЕ
    Отображает справочные сведения о командлетах и концепциях Windows PowerShell.

ПОЛНОЕ ОПИСАНИЕ

СИНТАКСИС
    get-help help <<имя_командлета> ! <название_раздела>>
    help <<имя_командлета> ! <название_раздела>>
    <имя_командлета> -?

    Команды "Get-help" and "-?" отображают справку на одной странице.
    Команда "Help" - на нескольких.
```

Рис. 1.5. Справочные сведения о командлетах

По команде get-help \* выводится внушительный список разделов справочной системы оболочки, где они разделены на четыре большие (Category) группы. Группы имеют обозначения: Alias (псевдоним), Cmdlet (командлеты), Provider (провайдер-программа, обеспечивающая доступ к определенному хранилищу данных) и HelpFile (файл помощи).

Каждая категория может вызываться отдельно, если команду Get-Help набирать с параметром -category и именем группы, например:

```
PS C:\Documents and Settings\user> Get-Help -Category provider
```

Вызов же справки по любому элементу группы производится указанием имени элемента после имени командлета `Get-Help`, например:

```
PS C:\Documents and Settings\user> Get-Help Alias
```

Каждому пользователю необходимо изучить дерево справочной системы, начиная с общих разделов.

### 1.3. Командлеты

Оболочка PowerShell поддерживает команды четырех типов: командлеты, функции, сценарии и внешние исполняемые файлы.

Командлеты — особый вид команд, очень похожих на внутренние традиционные оболочки. Отличительной особенностью командлетов является то, что их имя служит обращением к объектам базового класса `Cmdlet` платформы .NET. В поставку Windows PowerShell включены более 120 командлетов, каждый из которых предназначен для выполнения достаточно простых функций. Организация командлетов такова, что в любое время можно расширить их состав, не изменяя структуры оболочки. Объединение командлетов в одном классе обеспечивает их единый синтаксис и единые принципы построения. Композиции этих функций при составлении конвейеров, в которых результаты действия одного командлета передаются другому, являются мощным средством анализа и управления ресурсами компьютерных систем. Администраторы-профессионалы с помощью пакета `Software Developers Kit (SDK)` могут разрабатывать собственные командлеты, расширяя стандартную поставку PowerShell.

Для всех командлетов принят общий принцип их именования в виде слеша «\», глагола и существительного, например : `Get-Help`, `Set-Service`. Глагол определяет запланированное действие, а существительное определяет объект, над которым это действие выполняется. Приставка `Get` предполагает отображение текущей

информации на экране монитора об объекте, а Set определяет изменение режимов или состояний объекта-ресурса. Командлеты Set-\* способны коренным образом изменять состояния ресурсов и режимы их работы. Поэтому при их использовании возникают опасения, связанные с безопасностью систем. Большинство командлетов поддерживают так называемый прототипный режим, согласно которому сначала просчитывается действие командлета, затем идет уведомление пользователя о предполагаемых изменениях и запрашивается подтверждение (Confirm) на действительное выполнение этих действий.

В общем случае формат командлетов имеет следующую структуру:

*Имя командлета -параметр1 -параметр2 аргумент1 аргумент2*

где *-параметр1* – это параметр, не имеющий значения (подобные параметры часто называют переключателями); *-параметр2* – это параметр, имеющий значение, записанное в поле *аргумент1*;

*аргумент2* – это параметр не имеющий имени (или просто аргумент).

Примеры использования полей параметров и аргументов приведены в подразделе 1.3.2. Из структуры приведенного формата командлета видно, что задание параметров с помощью слеша «\», принятого в оболочке «**cmd.exe**», не используется.

Таблица 1.1

Общие параметры командлетов

Параметр	Тип	Действие
-Verbose	Boolean	Выводит подробные сведения об операциях, таких как результаты мониторинга или журналирование транзакций. Этот параметр эффективен в командлетах, формирующих подробные данные
-Debug	Boolean	Создает подробный отчет об операциях на уровне программирования. Используется в командлетах, создающих данные отладки

-ErrorAction	Enum	Отображает реакцию командлета на возникновение ошибки
-ErrorVariable	String	В дополнение к параметру, \$error определяет переменную, сохраняющую ошибки команды при выполнении
-OutVariable	String	Определяет переменную, сохраняющую выходные данные команды при выполнении
-OutBuffer	Int32	Ограничивает количество хранящихся в буфере объектов перед вызовом следующего командлета в конвейере
-WhatIf	Boolean	Предупреждает об изменениях в состоянии системы, которые неизбежно произойдут при выполнении командлета
-Confirm	Boolean	Запрашивает разрешение на выполнение действий, вносящих изменения в систему

Некоторые параметры поддерживаются практически всеми командлетами (табл. 1.1).

Даже имея только начальные сведения о построении PowerShell, можно убедиться, что заложенные в оболочке возможности значительно превышают возможности и удобства работы командной строки «cmd.exe», а также графической оболочки операционной системы Windows.

### 1.3.1. Работа с дисками

Одним из важнейших ресурсов компьютерных систем является ресурс памяти. С появлением многоядерных микропроцессоров значение этого ресурса еще более возрастает и выходит на передний план.

Фундаментальным положением любой ОС является управление данными, осуществляемое со стороны файловой системы. Файловая система представляет собой дерево вложенных каталогов (папок) и файлов. В командной оболочке PowerShell понятия диска, файла и

папок значительно расширены и практически эквивалентны одноименным понятиям Unix и Linux ОС. В качестве файлов могут выступать не только данные, находящиеся на внешних носителях, но и физические и логические устройства, диски, их разделы и т. п. Это значительно упрощает работу ОС и позволяет средствами файловых систем контролировать работу любых хранилищ данных как локальных, так и сетевых. Кроме того, используя в качестве псевдонимов названия управляющих операторов ОС, отличных от Windows, можно управлять различно организованными данными.

В каждом сеансе работы пользователю необходимо знать, какие ресурсы памяти не только его компьютера, но и сетевых хранилищ доступны. Объем получаемой информации о ресурсах может быть очень большим.

Список дисков, доступных пользователю из среды PowerShell, можно получить командой:

PS C:\> Get-PSDrive (см. рис. 1.9).

```
PS C:\Users\user> Get-PSDrive
Name                Used (GB)  Free (GB) Provider      Root          Curr
-----
Alias               49.67      86.47  FileSystem    C:\
cert                100.34     100.34 Certificate    \
D                   100.34     100.34 FileSystem    D:\
E                   100.34     100.34 FileSystem    E:\
Env                 Environment
Function            Function
G                   FileSystem C:\
HKCU                Registry   HKKEY_CURRENT_USER
HKLM                Registry   HKKEY_LOCAL_MACHINE
Variable            Variable
WSMan              WSMan
```

Рис.1.9. Список дисков доступных пользователю

Информация о доступных хранилищах является исходной для работы с ресурсом памяти. По команде

PS C:\>Get-PSDrive

сообщается точное обозначение диска (Root), его имя (Name), имя провайдера (Provider), поддерживающего этот диск, и текущая локализация (CurrentLocation). Кроме этих данных указываются доступные функции, псевдонимы, переменные окружения и т. п. В

PowerShell встроены специфические провайдеры, обеспечивающие доступ к специальным хранилищам: `Alias` – для доступа к псевдонимам PowerShell, `Certificate` – для использования сертификатов X509 цифровой подписи, `Environment` – для переменных среды Windows, `FileSystem` – для обращения к файловой системе, `Function` – для обращения к функциям PowerShell, `Registry` – для обращения к реестру Windows (ветви реестра HKCU – текущего пользователя и HKLM – локальной машины), `Variable` – для переменных PowerShell (см. рис. 1.9).

Провайдер PowerShell — это NET-приложение, предоставляющее пользователям оболочки доступ к хранилищам в едином формате, напоминающем формат обычных дисков файловой системы. Работа с представленными дисками практически ничем не отличается от работы с обычной файловой системой. Навигация по различным дискам, просмотр их содержимого, обращение к элементам данных выполняется с помощью команд:

командлеты `Get-Location` и `Set-Location` с их псевдонимами `pwd`, `cd`, `chdir`, `si`.

Все перемещения по дискам осуществляются командлетом `Get-Location` или с помощью его псевдонима `cd` (`chdir` – полное имя), как и в файловой системе Windows с использованием интерпретатора «`cmd.exe`». Файловая система в PowerShell управляет только физическими и логическими дисками (`C:\`, `D:\`, `E:\`). Все остальные хранилища управляются собственными провайдерами.

Сама файловая система контролирует только часть пространства, именуемого дисками. Очевидно, что возможности PowerShell гораздо обширнее программы «Проводник» ОС Windows. Список всех провайдеров оболочки может быть получен командлетом:

`Get-PSProvider` (рис. 1.10).

```

PS C:\Users\user> get-psprovider

Name                Capabilities                Drives
----                -
MSMan               Credentials                 (MSMan)
Alias               ShouldProcess              (Alias)
Environment        ShouldProcess              (Env)
FileSystem          Filter, ShouldProcess      (C, D, E, G)
Function           ShouldProcess              (Function)
Registry           ShouldProcess, Transactions (HKLM, HKCU)
Variable           ShouldProcess              (Variable)
Certificate        ShouldProcess              (cert)

PS C:\Users\user>

```

Рис. 1.10. Список провайдеров оболочки

Навигация по дискам PowerShell ничем не отличается от типичной работы файловой системы. Здесь также сохраняется понятие рабочего или текущего каталога. Путь к этому каталогу устанавливает командлет

Get-Location (псевдоним cd) без параметров (рис. 1.11).

```

PS C:\Users\user> get-location

Path
----
C:\Users\user

PS C:\Users\user>

```

Рис. 1.11. Установление текущего или рабочего каталога

Аналогом данного командлета является псевдоним pwd, выполняющий те же функции в Unix и Linux оболочках.

Создание новых дисков (хранилищ) не вызывает трудностей. Для примера можно решить следующую задачу: создадим новый диск внутри папки user, который будет содержать каталог с именем Ivan.



После этого достаточно снова набрать команду Get-PSDriver и убедиться, что появился новый диск, доступ к которому обеспечивает файловая система.

### 1.3.2. Работа с файловой системой

При работе с файловой системой пользователь должен уметь создавать каталоги (папки) и файлы, копировать их и перемещать по собственному желанию. В среде PowerShell для этого имеются все необходимые средства.

Изучение любой файловой системы начинают с команд, обеспечивающих получение списка каталогов и файлов. В PowerShell для этих целей предназначен командлет Get-Childitem и его псевдоним dir, более привычный для пользователей персональных компьютеров. Отличительной особенностью новых средств является их расширенная функциональность.

Рассмотрим несколько типовых примеров их применения. Использование параметра -Recurse (рекурсия) позволяет отразить содержание любого диска с его каталогами и подкаталогами. Команда:

```
>dir 'Documents and Settings' -Recurse
```

позволяет просмотреть полное содержимое каталога «Documents and Settings», т. е. все файлы, входящие в каталог.

По умолчанию командлет не отображает скрытые файлы. Если требуется включить в рассмотрение и их, то в команду следует вставить параметр -Force.

В случаях, когда необходимо ограничиться только списком каталогов и подкаталогов, можно задать конвейер, в котором второй командлет Where-Object со свойством PSIsContainer отфильтрует файлы (рис. 1.12):

```
Dir 'd:\BBN' | Where-Object {$_.PSIsContainer}
```

```
PS C:\Users\user> dir 'd:\BBN' | Where-Object {$_.PSIsContainer}
```

Каталог: D:\BBN

Mode	LastWriteTime	Length	Name
d----	19.01.2017 21:45		ARXIV_Постоянный_220210
d----	02.02.2015 20:13		GAD-GAN-GAE
d----	06.01.2016 23:17		ExpJay_Pneep
d----	24.02.2017 20:56		FineReader
d----	19.04.2016 20:10		Internet
d----	09.01.2017 20:36		UGS
d----	24.01.2015 13:23		АвтоТехноПроцессовСС

Рис. 1.12. Задание ограниченного списка каталога или подкаталога

Рассмотрим пример, иллюстрирующий использование большего числа полей в формате командлета. Пусть, например, в каталоге d:\BBN требуется найти все docx-файлы, имена которых начинаются буквами «схе», а оканчиваются буквой «а» (шаблон `схе*a.docx`), Решением этой задачи может служить команда:

```
dir -Recurse -Filter схе*a.docx -Path d:\BBN
```

Учитывая, что синтаксис оболочки PowerShell не критичен в отношении прописных и строчных букв, а также допускает сокращение слов до нескольких символов, обеспечивающих однозначное понимание терминов, запись можно сократить:

```
dir -r -fi схе*a.docx d:\BBN
```

В этом выражении псевдоним `dir` заменяет имя командлета `Get-ChildItem`, переключатель `-r` является сокращением `-Recurse`. Этот переключатель распространяет действие команды не только на указанный каталог, но и на все его подкаталоги. Параметр фильтр `-fi` (`-Filter`) с аргументом `схе*a.docx` задает маску файлов для поиска. Еще один параметр `-Path` с аргументом `d:\BBN` определяет путь к исследуемому каталогу. Имя параметра с ключевым словом `-Path` можно не записывать, если по контексту выражение не имеет других значений (рис. 1.13).

```

PS C:\Users\user> dir -r -fi Схема.docx d:\BBN

Каталог: D:\BBN\Отчеты\Отчеты_2013-14_311213\Информатика_АСВед-12\Мухаметалинов Р.Р\практиче

Mode                LastWriteTime         Length Name
----                -
-a---             25.12.2013      14:01           14087 Схема.docx

Каталог: D:\BBN\Отчеты\Отчеты_2016-17_161124\Семестр_Осень\Информатика\Информатика_АСВед-11
еская_1,2

Mode                LastWriteTime         Length Name
----                -
-a---             22.09.2016      11:01           29363 Схема модулей ядра.docx

Каталог: D:\BBN\Отчеты\Отчеты_2016-17_161204\Семестр_Осень\Информатика\Информатика_АСВед-11
еская_1,2

Mode                LastWriteTime         Length Name
----                -
-a---             22.09.2016      11:01           29363 Схема модулей ядра.docx

PS C:\Users\user>

```

Рис. 1.13. Параметры каталога, с заданием маски файлов для поиска

Результатом выполнения этой команды будет информация о трех файлах: «Схема», «Схема модулей ядра» и «Схема модулей ядра». Файлы расположены в разных каталогах и имеют расширение «.docx».

Другой процедурой файловой системы PowerShell является создание новых каталогов и файлов. Эти функции выполняет командлет New-Item. В качестве параметров требуется указать место размещения создаваемого объекта. Для этого в параметре -Path прописывается полный путь к каталогу, в котором создается объект, а в параметре -Type указывается тип объекта – «directory» или «file». Создадим в каталоге диска «D:\BBN» новый каталог с именем «AIST-21\_OS» (рис. 1.14).

```
PS C:\Users\user> new-item -path d:\BBN -type directory -name AIST-21_OS

Каталог: D:\BBN

Mode                LastWriteTime         Length Name
----                -
d-----           13.04.2017    14:00             AIST-21_OS

PS C:\Users\user> _
```

Рис. 1.14. Создание новых каталогов и файлов

Войдем в созданный каталог «AIST-21\_OS», выполнив команду:

```
cd AIST-21_OS (рис. 1.15)
```

```
PS C:\Users\user> cd d:\BBN\AIST-21_OS
PS D:\BBN\AIST-21_OS> _
```

Рис. 1.15. Вход в созданный каталог

Объявим и создадим текстовый файл Лаб\_15.txt, в который в качестве заголовка внесем текст «Лабораторная» (значение для параметра -Value). Поскольку в параметре -Path путь к файлу не указан (можно опустить и название параметра), то по умолчанию файл создается в текущем каталоге (рис. 1.16).

```
PS C:\Users\user> cd d:\BBN\AIST-21_OS
PS D:\BBN\AIST-21_OS> new-item -path Лаб_15.txt -type file -value "лабораторная"

Каталог: D:\BBN\AIST-21_OS

Mode                LastWriteTime         Length Name
----                -
-a---           13.04.2017    16:39             24 Лаб_15.txt
```

Рис. 1.16. Создание файла в текущем каталоге

Копирование файлов в PowerShell практически ничем не отличается от копирования в других командных оболочках. Копирование осуществляется с помощью командлета Rename-Item, имеющего псевдоним gen.

Создадим в папке AIST-21\_OS файл Лаб\_15-1.tmp (рис. 1.17)

```
PS D:\BBN\AIST-21_OS> new-item -path Лаб_15-1.tmp -type file

Каталог: D:\BBN\AIST-21_OS

Mode                LastWriteTime         Length Name
----                -
-a---             18.04.2017   17:22             0 Лаб_15-1.tmp

PS D:\BBN\AIST-21_OS> _
```

Рис. 1.17. Создание файла в заданной папке

Сделаем ему копию (файл Лаб\_15-7.tmp). Для того чтобы сразу видеть результат, укажем параметр -PassThru (рис. 1.18).

```
PS D:\BBN\AIST-21_OS> ren Лаб_15-1.tmp Лаб_15-7.tmp -PassThru

Каталог: D:\BBN\AIST-21_OS

Mode                LastWriteTime         Length Name
----                -
-a---             18.04.2017   17:22             0 Лаб_15-7.tmp

PS D:\BBN\AIST-21_OS> _
```

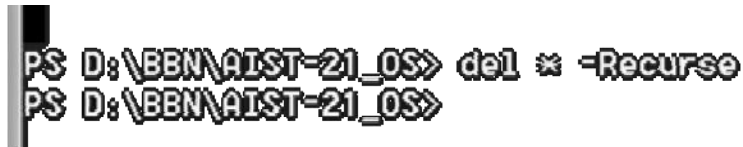
Рис. 1.18. Создание копии файла в текущей папке

Для контроля проведенных работ можно использовать команды-псевдонимы dir и type.

Удаление каталогов (псевдоним rd) и файлов (псевдоним del) выполняется с помощью командлета Remove-Item. После имени командлета или псевдонима, его заменяющего, прописывается путь удаляемому объекту и параметры: —Include, значение которого задает шаблон удаляемых файлов, например \*.txt (удаляются все файлы с расширением .txt). Параметр -Exclude, указывающий, на какие файлы команда не должна действовать, например \*.ps1 (файлы с расширением .ps1 не удаляются).

Команда удаления файлов в созданной папке AIST-21\_OS может иметь вид

>del \* -Recurse (рис. 1.19).



```
PS D:\BBN\AIST-21_OS> del * -Recurse
PS D:\BBN\AIST-21_OS>
```

Рис. 1.19. Из папки «AIST-21\_OS» удалены все файлы

**Никаких подтверждений на удаление файлов команда не запрашивает.**

### 1.3.3. Работа с конфигурацией оболочки

В качестве примера работы в среде PowerShell рассмотрим, каким образом можно изменять ее параметры, обеспечивая комфортность работы пользователя. По своему желанию пользователь может устанавливать размеры и расположение окна PowerShell, характеристики используемых шрифтов, выбор цвета и другие параметры.

Проще всего эти установки выполняются с помощью диалогового окна оболочки. Для его вызова нужно установить курсор мыши на заголовке окна, кликнуть правой кнопкой (ПКМ) и выбрать пункт «Свойства» в появившемся контекстном меню. Должно появиться диалоговое окно (рис. 1.20). Окно имеет несколько вкладок (Общие, Шрифт, Расположение, Цвета), каждая из которых позволяет настраивать определенную группу параметров. Здесь возможно буферирование и запоминание интерактивно выполняемых команд с целью их последующего использования отдельным блоком в сценарии.

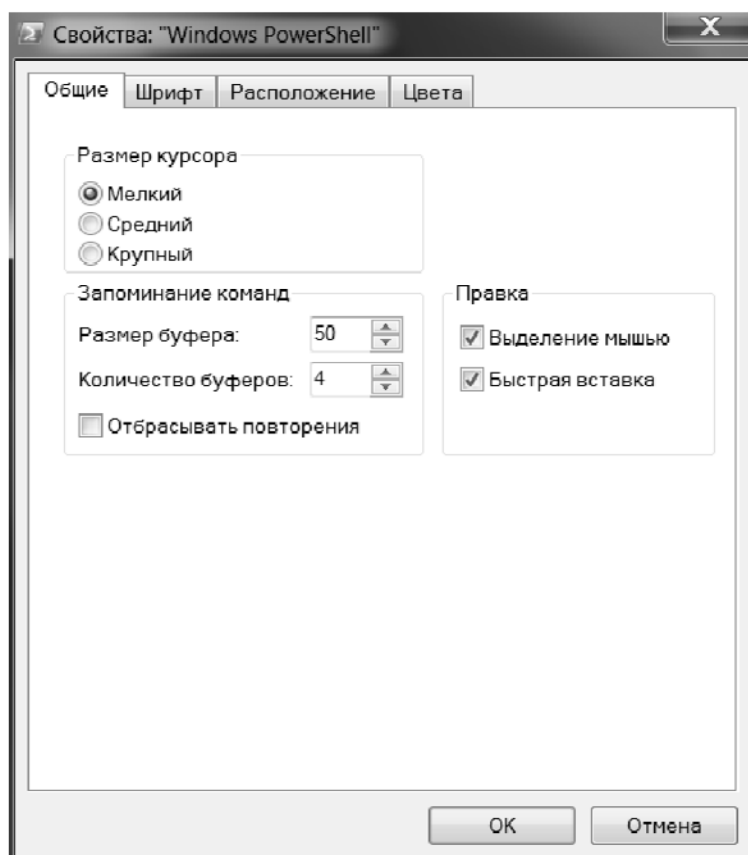


Рис. 1.20. Диалоговое окно оболочки PowerShell

После выбора и установки всех нужных параметров следует нажать кнопку «ОК». Система тут же потребует указаний, к какому объекту применить эти изменения. Если изменения свойств должны действовать постоянно, то следует выбрать переключатель «Сохранить свойства для других окон с тем же именем», если изменения предусмотрены как разовые, то выбирается «Изменить свойства только текущего окна».

Кроме инструментальных средств настройки командного окна имеется возможность применения чисто программных средств, являющихся неотъемлемой частью самой оболочки.

По умолчанию командлет «Get-Host» (рис. 1.21) без параметров отображает информацию о самой оболочке (региональные настройки, версия и т. п.).

```
Windows PowerShell
(©) Корпорация Майкрософт, 2009. Все права защищены.

PS C:\Users\user> Get-Host

Name                : ConsoleHost
Version             : 2.0
InstanceId          : f2668edd-4aa5-4d67-9589-0fddd39902fc
UI                  : System.Management.Automation.Internal.Host.InternalHostUserInter
CurrentCulture      : ru-RU
CurrentUICulture    : ru-RU
PrivateData         : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
IsRunspacePushed   : False
Runspace            : System.Management.Automation.Runspace.LocalRunspace

PS C:\Users\user>
```

Рис.1.21. Отображение информации о самой оболочке PowerShell

В команде (Get-Host).UI имя командлета взято в круглые скобки. Это обозначает, что требуется выполнить данный командлет и сформировать выходной объект. Только после этого извлекается свойство объекта «UI». Пройдя эту цепочку, получаем доступ к параметрам командного окна (рис. 1.22):

```
PS C:\Users\user> (Get-Host).UI.RawUI

ForegroundColor      : DarkYellow
BackgroundColor      : DarkGreen
CursorPosition       : 0,19
WindowPosition       : 0,0
CursorSize           : 50
BufferSize           : 120,3000
WindowSize           : 89,42
MaxWindowSize        : 106,47
MaxPhysicalWindowSize : 106,47
KeyAvailable         : False
WindowTitle          : Windows PowerShell

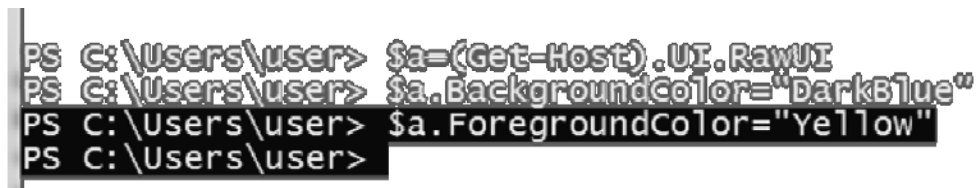
PS C:\Users\user>
```

Рис. 1.22. Выполнение командлета и формирование выходного объекта

Значение каждого из перечисленных параметров можно изменить, настраивая внешний вид окна по желанию. При изменениях параметров целесообразно объект «RawUI» сохранить в качестве



значения отдельной переменной. Например, изменение цвета фона и текста. Свойство «BackgroundColor» отвечает за цвет фона, а «ForegroundColor» — за цвет текста. Можно использовать 16 цветов: «Black», «Gray», «Red», «Magenta», «Yellow», «Blue», «Green», «Cyan», «White», «DarkGreen», «DarkCyan», «DarkRed», «DarkMagenta», «DarkYellow», «Dark-Gray», «DarkBlue». Установим желтый цвет текста на темно-синем фоне, Желаемый эффект обеспечивается выполнением трех команд, приведенных на рис. 1.23.



```
PS C:\Users\user> $a=(Get-Host).UI.RawUI
PS C:\Users\user> $a.BackgroundColor="DarkBlue"
PS C:\Users\user> $a.ForegroundColor="Yellow"
PS C:\Users\user>
```

Рис. 1.23. Изменение цвета фона и текста оболочки PowerShell

Действие этих команд обеспечивается сразу после их выполнения и не затрагивает строки, предшествующие этим командам.

Некоторые параметры оболочки, например «WindowSize» (см. рис. 1.22), содержат по две координаты. Для изменения их значений объявляем новую переменную

```
$b=$a.WindowSize
```

Переопределим значения ширины и высоты командами:

```
$b.Width=80 и $b.Height=25 ,
```

а затем изменим содержимое объекта «WindowSize» переменной \$a, т. е. выполним команду

```
$a.WindowSize=$b.
```

Последняя строка, изображенная на рис. 1.22, отражает заголовок командного окна «PowerShell». Оно достаточно длинное. Выполнив команду

```
$a.WindowTitle="Командное окно VVN" ,
```

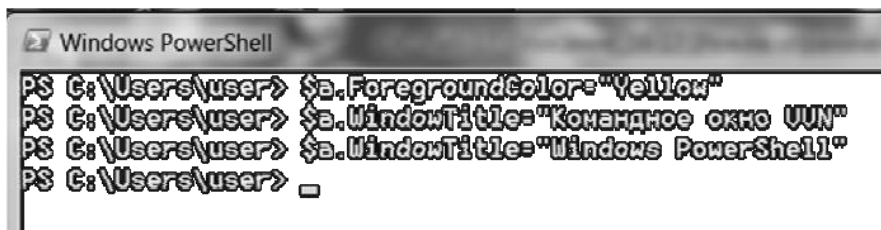
получаем планируемое название (рис. 1.24).



```
Командное окно VVN
PS C:\Users\user> $a.ForegroundColor="Yellow"
PS C:\Users\user> $a.WindowTitle="Командное окно VVN"
PS C:\Users\user> _
```

Рис. 1.24. Изменение названия командного окна

Далее меняем название командного окна на предыдущее (рис. 1.24.1):



```
Windows PowerShell
PS C:\Users\user> $a.ForegroundColor="Yellow"
PS C:\Users\user> $a.WindowTitle="Командное окно VVN"
PS C:\Users\user> $a.WindowTitle="Windows PowerShell"
PS C:\Users\user> _
```

Рис. 1.24.1. Восстановление предыдущего названия командного окна

В некоторых случаях необходимо изменить приглашение. Мигающему курсору после букв «PS» предшествует запись полного пути к текущему каталогу. Вид приглашения командной строки в PowerShell определяется функцией Prompt, которая имеет формат, отражаемый следующей командой:

```
PS C:\Users\user> <Set-Item Function:Prompt> .Definition 'PS' +
$<Get-Location> +
$<if <$nestedprompt level -ge 1> <'>> '>>' >> + '>' >
```

Используя справочную систему, покажем, как создать приглашение, эквивалентное командной строке «cmd.exe» (рис. 1.25).



```
PS C:\Users\user> Function Prompt {"$(Get-Location) > " }
C:\Users\user >
```

Рис. 1.25. Изменение вида курсора

В данной команде используется конструкция “\$(Get-Location) > “.., называемая подвыражением (subexpression).

Подвыражение – это блок кода на языке PowerShell, который в строке заменяется значением полученным в результате выполнения этого кода.

Все приведенные выше настройки выполняются в интерактивном режиме, и их действие распространяется лишь на время текущего сеанса работы. После окончания сеанса работы в оболочке PowerShell они утрачивают силу.

Для сохранения настроек с целью их регулярного, а может быть, и повседневного использования необходимо создать файл с соответствующим набором команд-настроек. Этот файл текстового типа получил название профиль.

Профиль – это сценарий, который будет загружаться и активизировать необходимые настройки при каждом запуске оболочки PowerShell. Значение профиля очень близко значениям файлов autoexec.bat – для ранних и autoexec.nt – для современных версий ОС Windows. Все они предназначены для автоматического выполнения требуемых подготовительных работ. Корректно составленный профиль обеспечивает не только комфортные условия работы пользователя, но и создать удобства для администрирования. Разработка и распространение профилей позволяют создать единые условия работы пользователей на группе компьютеров в распределенной среде, например, в локальных компьютерных сетях.

В зависимости от уровня выполняемых настроек и значимости администрируемых ресурсов формируются профили четырех видов:

1. Действующие на всех пользователей сети и на все их оболочки PowerShell (хосты).
2. Действующие на всех пользователей сети с использованием единой оболочки PowerShell.

3. Действующие только на текущего пользователя и на все оболочки.

4. Действие, которых распространяется только на текущего пользователя и только на хост powershell.exe.

Поэтому в инструментальных средствах оболочки (см. рис. 1.20): Свойства; Вкладка; Общие) предусмотрена возможность работы с несколькими буферами, в которых могут одновременно подготавливаться блоки команд сразу для нескольких профилей.

```
PS C:\Users\user> $profile
C:\Users\user\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1
PS C:\Users\user>
```

Рис. 1.26. Задание места хранения пользовательского профиля

Каждый тип профиля имеет свое место хранения в особой зоне ОС Windows. При работе с оболочкой на автономном компьютере используется только пользовательский профиль, относящийся к последнему типу. Место его расположения и имя файла можно определить по значению специальной переменной «\$profile» (рис. 1.26).

#### 1.3.4. Работа с объектами

Оболочка PowerShell относится к объектно-ориентированной среде, в которой все действия совершаются над объектами – ресурсам компьютерных систем. Каждый объект в общем случае включает совокупность данных, отражающих свойства объекта, и модули обработки этих данных (методы объекта).

Фундаментом оболочки следует считать платформу Microsoft.NET, так как она изначально предназначалась для разработки различных информационных систем. В составе этой платформы уже заложен набор сетевых служб и серверов, что

позволяет создать программный образ функционирования любой системы. Часть это платформы Microsoft.NET Framework предназначена для разработки приложений. Она дает большие преимущества для разработчиков программ в части использования различных систем программирования.

Самым мощным достоинством платформы Microsoft.NET служит наличие в ней обширной библиотеки классов (тысячи классов) содержащих готовые, отлаженные и постоянно пополняемые методы обработки. Мощным средством обработки данных в оболочке PowerShell является использование конвейеров.

Конвейер – это последовательность команд, разделенных вертикальной чертой « | », в которой результаты обработки одной команды передаются другой команде. В оболочке PowerShell по конвейеру передаются целые объекты, наборы свойств отформатированных данных с требуемой детализацией.

Последовательный и пошаговый принцип обработки конвейера позволяет решать как задачи анализа сложных систем, так и на его базе строить новые информационные системы любой сложности.

Проанализируем на примерах, какой арсенал средств может применять пользователь в своей работе. Очень часто при анализе многопрограммной работы компьютера возникает задача определения, какие процессы и как используют ресурсы системы.

Начальную информацию можно получить, включив командлет «Get-Process», который отражает часть данных об активных процессах. Результатом работы командлет «Get-Process» является список запущенных в системе процессах. Каждая запись содержит набор параметров-характеристик процессов, Некоторые из них интуитивно понятны (ProcessName – имя процесса, CPU(s) – время, затрачиваемое на работу процесса в секундах), другие имеют не всегда понятные сокращения (рис. 1.27).

```

Windows PowerShell
(C) Корпорация Майкрософт, 2009. Все права защищены.

PS C:\Users\user> Get-Process

Handles      NPM(K)      PM(K)      WS(K) VM(M)      CPU(s)      Id ProcessName
-----
1309         42      29208      58596    240         9,00      2416 amigo
146          20      26240      17948    179         0,22      2828 amigo
166          23      37696      51700    238        16,68      3148 amigo
152          22      28272      40416    227         1,75      3156 amigo
147          21      27524      38056    218         0,45      3168 amigo
145          21      24964      37012    223         0,58      3176 amigo
149          21      24876      37840    220         0,51      3192 amigo
146          22      28504      39160    226         0,64      3216 amigo
148          21      24872      35780    209         0,50      3248 amigo
71           8       1176       3924     42          0,00      1584 armsvc
130          11      15892      15712    53          0,00      6924 audiodg
2508        413     250276     108784   663         9,92      1756 avp
748         88      85986      4680     361         0,00      5664 avpui
34          7       2016       3428     28          0,00      2456 cmd
35          6       980        2924     29          0,00      1960 conhost
48          6       996        2964     29          0,00      2248 conhost

```

Рис. 1.27. Список запущенных в системе процессах

Для понимания смысла сокращений и выяснения полной структуры объекта целесообразно использовать конвейер двух командлетов «Get-Process | Get-Member» (рис. 1.28).

```

PS C:\Users\user> Get-Process | Get-Member

TypeName: System.Diagnostics.Process

Name      MemberType Definition
-----
Handles  AliasProperty Handles = Handl...
Name      AliasProperty Name = ProcessName
NPM      AliasProperty NPM = NonpagedSystemMemorySize
PM       AliasProperty PM = PagedMemorySize
VM       AliasProperty VM = VirtualMemorySize
WS       AliasProperty WS = WorkingSet
Disposed Event System.EventHandler Disposed(System.Object, S...
ErrorDataReceived Event System.Diagnostics.DataReceivedEventHandler (...
Exited Event System.EventHandler Exited(System.Object, Sys...
OutputDataReceived Event System.Diagnostics.DataReceivedEventHandler (...
BeginErrorReadLine Method System.Void BeginErrorReadLine()
BeginOutputReadLine Method System.Void BeginOutputReadLine()
CancelErrorReadLine Method System.Void CancelErrorReadLine()

```

Рис. 1.28. Формирование полной структуры объекта

Для сокращения длины команды и уменьшения трудоемкости набора можно записать конвейер из псевдонимов этих командлетов – «gps | gm» (рис. 1.28.1).

```
PS C:\Users\user> gps | gm

TypeName: System.Diagnostics.Process

Name                MemberType          Definition
-----                -
Handles             AliasProperty      Handles = HandleCount
Name                AliasProperty      Name = ProcessName
NPM                 AliasProperty      NPM = NonpagedSystemMemorySize
PM                  AliasProperty      PM = PagedMemorySize
VM                  AliasProperty      VM = VirtualMemorySize
WS                  AliasProperty      WS = WorkingSet
Disposed            Event               System.EventHandler Disposed(System.Object, Sys
ErrorDataReceived  Event              System.Diagnostics.DataReceivedEventHandler Err
Exited              Event              System.EventHandler Exited(System.Object, Syste
outputDataReceived Event              System.Diagnostics.DataReceivedEventHandler Out
BeginErrorReadLine Method              System.Void BeginErrorReadLine()
BeginOutputReadLine Method              System.Void BeginOutputReadLine()
CancelErrorRead    Method              System.Void CancelErrorRead()
```

Рис. 1.28.1. Использование псевдонимов при записи командлетов

На экран выводится весь перечень свойств процессов. Перед списком указывается, к какому .NET-типу относятся все названия свойств «Diagnostics.Process». Если требуется вывести только определенные категории свойств, то следует задать значение типа параметра (рис. 1.29).

```
PS C:\Users\user> gps | gm -MemberType Property

TypeName: System.Diagnostics.Process

Name                MemberType          Definition
-----                -
BasePriority         Property            System.UInt32 BasePriority {get;}
Container            Property            System.ComponentModel.IContainer Container {g
EnableRaisingEvents Property            System.Boolean EnableRaisingEvents {get;set;}
ExitCode             Property            System.UInt32 ExitCode {get;}
ExitTime             Property            System.DateTime ExitTime {get;}
Handle               Property            System.IntPtr Handle {get;}
HandleCount          Property            System.UInt32 HandleCount {get;}
HasExited            Property            System.Boolean HasExited {get;}
Id                   Property            System.UInt32 Id {get;}
MachineName          Property            System.String MachineName {get;}
```

Рис. 1.29. Определенные категории свойств объекта

В оболочке PowerShell имеется несколько конфигурационных файлов, необходимых для отображения объектов различных типов. Они находятся в том же каталоге, что и «powershell.exe», и имеют названия, заканчивающиеся «\*format.pslxml» (рис. 1.30).

```

PS C:\WINDOWS\system32\WindowsPowerShell\v1.0>
PS C:\WINDOWS\system32\WindowsPowerShell\v1.0> dir

Каталог: C:\WINDOWS\system32\WindowsPowerShell\v1.0

Mode                LastWriteTime         Length Name
----                -
d-----          12.04.2011        17:26      en-US
d-----          14.07.2009         9:52      Examples
d-----          12.04.2011        17:37      Modules
d-----          12.04.2011        17:26      ru-RU
-a-----          11.06.2009         0:41      27338 Certificate.format.ps1xml
-a-----          14.07.2009         5:29      126976 CompiledComposition.Micros
-a-----          11.06.2009         0:41      27106 Diagnostics.Format.ps1xml
-a-----          11.06.2009         0:41      72654 DotNetTypes.format.ps1xml
-a-----          11.06.2009         0:41      24857 FileSystem.format.ps1xml
-a-----          11.06.2009         0:42      15603 getevent.types.ps1xml

```

Рис. 1.30. Формирование файлов различных типов по заданному окончанию

Файл (рис. 1.30) в этом списке «DotNetTypes.format.ps1xml» предназначен для формирования объектов System.Diagnostics.Process.

Примеры показывают, что количество выводимых данных может быть очень большим, и требуется иметь средства, позволяющие отсеивать ненужную информацию и выделять требуемую. Обычно для этого используются процедуры сортировки и фильтрации. Создание рабочих массивов данных, упорядоченных по определенному параметру, позволяет использовать методы дихотомии согласно зависимости

$$n = \log_2 N,$$

где  $n$  – число проб, в результате которых находятся данные в отсортированном массиве;  $N$  – количество элементов в исходном анализируемом массиве.

Дихотомия – раздвоенность, последовательное деление на две части, не связанные между собой. Способ логического деления класса на подклассы, который состоит в том, что делимое понятие полностью делится на два взаимодополняющих понятия. Дихотомическое деление в математике является способом образования подразделов одного понятия или термина и служит для образования классификации элементов.



Существует теорема: «Если непрерывная функция на концах некоторого интервала имеет значения разных знаков, то внутри этого интервала у нее есть корень (как минимум, один, но может быть и несколько)». На базе этой теоремы построено численное нахождение приближенного значения корня функции. Обобщенно этот метод называется дихотомией, т. е. делением отрезка на две части.

За операции сортировки данных обычно отвечает командлет «Sort-Object». В качестве его параметра указываются имена свойств, по которым упорядочиваются объекты. Выведем список процессов, упорядоченный по процессорному времени (рис. 1.31).

```
PS C:\> gps|Sort-Object -property cpu
```

Handles	NPW(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
1064	10	4512	7356	198		2728	postgres
1101	11	7080	14228	204		2788	postgres
1063	10	4484	7884	198		2712	postgres
37	6	960	3560	54	0,02	2888	hpwusdhd2
80	8	2180	6376	69	0,02	2516	hkcmd
75	9	1368	4972	68	0,02	2880	jusched
90	10	1448	5256	66	0,08	2900	HpqSRmon
105	10	2568	7476	78	0,08	2868	tgfxpers
85	9	1448	5388	64	0,08	2908	VcdDaemon
70	8	1708	5152	52	0,08	6208	sp1wow64
178	18	6416	2424	100	0,14	2620	ksderrt

Рис. 1.31. Сортировка объектов по указанному имени свойства (по возрастанию)

Для получения списка, упорядоченного в порядке убывания процессорного времени, должен быть включен параметр `-descending` (рис. 1.32).

```
PS C:\> gps|Sort-Object -property cpu -descending
```

Handles	NPW(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
564	75	98172	144296	404	172,85	6784	WINWORD
166	23	40886	55504	247	57,21	3076	amigo
115	15	52564	54776	184	29,13	1848	dwm
1418	42	30028	55536	237	21,87	2400	amigo
722	89	88672	4872	362	12,56	5786	avpu1
874	74	42800	59664	277	11,65	1928	explorer
486	47	15000	28020	179	9,10	2448	utorrent
384	24	52064	51252	574	1,54	1940	powershell
92	9	3124	9952	89	1,56	2120	conhost

Рис. 1.32. Сортировка объектов по указанному имени свойства (по убыванию)

Часто в отсортированном списке наибольший интерес представляют записи, имеющие максимальные или минимальные значения некоторых параметров. Для выявления подобных объектов в списке командлетов имеется «Select-Object», который позволяет в отсортированном списке отобразить несколько первых (-First) или последних (-Last) записей. Например, для выявления пяти процессов, использующих наибольшие объемы памяти (свойство WS), можно сформировать команду-конвейер, состоящую из трех командлетов (рис.1.33).

```
PS C:\> gps | sort-object WS | select-object -Last 5
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
166	23	43916	56100	250	70,12	3076	amigo
876	75	42732	59820	277	13,63	1928	explorer
544	30	2780	59892	172		676	csrss
455	30	122088	126928	250		576	svchost
558	77	106588	151908	416	374,32	6784	WINWORD

```
PS C:\> █
```

Рис. 1.33. Пять процессов, использующие наибольших объем памяти

Для фильтрации данных обычно используют командлет Where-Object. Например, для определения данных об остановленных службах (свойство «Status» равно «Stopped») следует сформировать команду, приведенную на рис. 1.34.

```

PS C:\> Get-Service|Where-Object {$_.Status -eq "Stopped"}
Status Name DisplayName
-----
Stopped AdobeFlashPlaye... Adobe Flash Player Update Service
Stopped AeLookupSvc Информация о совместимости приложений
Stopped ALG Служба шлюза уровня приложения
Stopped AppIDSvc Удостоверение приложения
Stopped AppInfo Сведения о приложениях
Stopped AppMgmt Управление приложениями
Stopped AxInstSV Установщик ActiveX (AxInstSV)
Stopped BDESVC Служба шифрования дисков BitLocker
Stopped Browser Браузер компьютеров
Stopped bthserv Служба поддержки Bluetooth
Stopped CertPropSvc Распространение сертификата
Stopped clr_optimizatio... Microsoft .NET Framework NGEN v2.0....
Stopped clr_optimizatio... Microsoft .NET Framework NGEN v2.0....

```

Рис. 1.34. Список остановленных процессов операционной системы

Появляется информация об остановленных службах. Следует обратить внимание, что для сравнения свойств в фигурных скобках командлета не используются знаки =, <, >, а операторы сравнения задаются мнемоническими сокращениями (табл. 1.2).

Таблица 1.2

Операторы сравнения в PowerShell

Оператор	Значение
-eq	Равно
-ne	Не равно
-lt	Меньше
-le	Меньше или равно
-gt	Больше
-ge	Больше или равно
-like	Сравнение на совпадение с учетом подстановочного знака во втором операнде
-notlike	Сравнение на несовпадение с учетом подстановочного знака во втором операнде
-contains	Содержит
-notcontains	Не содержит

После выделения требуемой информации и отсева ненужной возникают задачи определения характеристик выделенных объектов. Применение некоторых командлетов позволяет решать часть из них.

Одной из типовых задач является определение суммарного объема некоторой группы файлов. Подсчитаем объем памяти с точностью до байта, занимаемый каталогом, например \BBN. Решение этой задачи можно обеспечить последовательностью команд, изображенных на рис. 1.35.

Первая строка формирует переменную \$TotalLength со значением нуль, вторая строка представлена конвейером из двух командлетов, подсчитывающих суммарный объем памяти. Командлет ForEach-Object обеспечивает циклическое накопление суммы. Третья строка считывает полученный итог (рис. 1.35).

```
PS D:\BBN> $TotalLength=0
PS D:\BBN> dir |ForEach-Object {$TotalLength+=$_ .Length}
PS D:\BBN> $TotalLength
0
PS D:\BBN> dir

Каталог: D:\BBN

Mode                LastWriteTime         Length Name
----                -
d-----          18.04.2017         18:00     AIST-21_OS
d-----          18.04.2017         15:30     AIST-21_OS_
d-----          19.01.2017          21:45     ARXIV_Постоянный_220210
d-----          02.02.2015          20:13     CAD-CAM-CAE
d-----          06.01.2016          23:17     Exp'lay_Imcep
d-----          24.02.2017          20:56     FineReader
d-----          19.04.2016          20:10     Internet
```

Рис. 1.35. Определение суммарного объема группы файлов

Эту же задачу в более расширенном функциональном формате можно решить, используя командлет «Measure-Object» (рис. 1.36).

```
PS D:\BBN>
PS D:\BBN>
PS D:\BBN> dir |Measure-Object -Property Length -Sum
```

Рис. 1.36. Определение суммарного объема группы файлов

Этот конвейер может дать больше расчетных данных об объекте. Достаточно знать, какие характеристики интересуют пользователя (рис. 1.37).

```
PS D:\BBN>  
PS D:\BBN>  
PS D:\BBN>  
PS D:\BBN> dir|Measure-object -Property Length -Sum -Minimum -Maximum -Average
```

Рис. 1.37. Расчетные данные об объектах с заданными характеристиками

Следует помнить, что все командлеты имеют строго ограниченную функциональность, поэтому используются только для решения узких, типовых задач. Творческий характер анализа и обработки характеристик объектов в основном переносится в функции.

#### 1.4. Функции командной оболочки MS PowerShell

Набор командлетов оболочки PowerShell можно отнести к языку запросов. В терминах СУБД Microsoft Access они обеспечивают запросы-выборки, в которых имя командлета указывает на объект, с которым работает пользователь, а вариация параметров является инструкцией к тому, какие данные и в каком виде должны быть представлены в результате выполнения запроса. Комбинированное действие командлетов в виде конвейеров позволяет получить более сложные виды запросов: запросы с группировкой, перекрестные запросы, запросы с параметрами, запросы-действия и т. п.

Функциональность каждого командлета изменить нельзя, так как их программный код из оболочки не доступен. Только функции и сценарии позволяют формировать программный код, который пользователь создает по своему желанию. Обработка данных в зависимости от контекста работ и специфики аппаратных и программных ресурсов может осуществляться с различной степенью детализации.

Функция в PowerShell – блок кода, имеющий уникальное имя. Этот блок активизируется при первом к нему обращении и остается действительным до завершения текущего сеанса работы с оболочкой. Функции могут быть очень простыми – без параметров и очень сложными – с формальными и замещаемыми параметрами. Они могут включаться в конвейеры и возвращать значения не только некоторых переменных, а даже целых массивов переменных различного типа данных.

Следует отметить, что функциональность языковых средств программирования очень высокая, и это позволяет создавать очень эффективные программы. Язык программирования оболочки PowerShell требует отдельного рассмотрения. Ниже будут приведены примеры построения функций различной сложности.

Для определения функции используется формат :

Function Имя\_функции {тело функции} [аргументы],

где Function – ключевое слово, которым объявляется новая функция; Имя\_функции – присваиваемое уникальное имя;

{тело функции} – набор операторов встроенного языка программирования, обеспечивающих обработку данных. Тело функции обязательно заключается в фигурные скобки «{ }»;

[аргументы] – набор аргументов и параметров функции. Квадратные скобки указывают, что аргументы и параметры не являются обязательным элементом, они могут отсутствовать.

На рисунке 1.38 приведен пример формирования простейшей функции без аргументов. Создадим функцию «MyFunc», формирующую текстовое сообщение:

```
Windows PowerShell
(C) Корпорация Майкрософт, 2009. Все права защищены.

PS C:\Users\user> Function MyFunc ("Привет, привет")
PS C:\Users\user>
PS C:\Users\user> MyFunc
Привет, привет
PS C:\Users\user>
```

Рис. 1.38. Формирование простой функции без аргументов

Для активизации функции достаточно в командную строку записать ее имя.

Аналогично можно создавать функции, извещающие пользователя о начале и прекращении каких-либо работ в системе.

В более сложных случаях функции могут использовать аргументы, которые передаются ей при запуске. Имеется два вида обработки аргументов: с помощью переменной «\$Args» и путем задания формальных параметров. Рассмотрим оба варианта обработки аргументов.

В оболочке PowerShell имеется переменная «\$Args», которая в общем является массивом. Элементы массива могут быть параметрами функции, заданными при ее запуске. Переопределим предыдущую функцию таким образом, чтобы она могла принимать переменные значения (рис. 1.39).

```
PS C:\Users\user>
PS C:\Users\user>
PS C:\Users\user> Function MyFunc ("Привет, привет! $Args!")
PS C:\Users\user> MyFunc Александр Михаил Павел
Привет, привет! Александр Михаил Павел!
PS C:\Users\user>
```

Рис. 1.39. Переопределение предыдущей функции для задания переменных значений

Переменная «\$Args» помещена в тело функции, заключенное в фигурные скобки. Это обозначает, что при запуске функции она

примет значения аргументов и вставит их в строку результата (см. рис. 1.39).

Внутри переменной «\$Args», являющейся массивом, можно обращаться к элементам массива по их порядковому номеру. Например, если требуется подсчитать сумму нескольких чисел и знать их количество, то функцию «SumArgs» можно определить, как показано на рис. 1.40.

У этой функции может быть переменное число аргументов. Требуется сложить пять слагаемых: 50, 14, 4, 7, 33 (рис. 1.40).

```
PS C:\Users\user> Function SumArgs ("Количество аргументов: $($Args.Count)"
>> $n=0
>> For ($i=0; $i -lt $Args.Count; $i++) {$n+=$Args[$i] }
>> "Сумма аргументов: $n"}
>>
PS C:\Users\user> SumArgs 50 14 4 7 33
Количество аргументов: 5
Сумма аргументов: 108
PS C:\Users\user>
```

Рис. 1.40. Определение суммы пяти слагаемых из массива

Подобную функцию вполне можно использовать для обработки чисел массивов, образующихся при выполнении некоторых командлетов и их конвейеров. Другим методом учета аргументов является задание формальных параметров функции, значения которых замещаются значениями аргументов. Это типовой прием во многих системах программирования.

Рассмотрим несколько примеров. Определим функцию, обеспечивающую сложение двух аргументов (рис. 1.41).

```
PS C:\Users\user>
PS C:\Users\user> Function Add ($x, $y) {$x+$y}
PS C:\Users\user> ■
```

Рис. 1.41. Определение функции для сложения двух аргументов



При определении функции выражение в круглых скобках устанавливает порядок ввода и анализа переменных, а выражение в фигурных скобках формирует тело функции. По умолчанию эта функция, как и другая функция PowerShell, ведет себя полиморфным образом. Она учитывает и «приспосабливается» к желаниям пользователей. Рассмотрим несколько вариантов работы функции с различными типами данных (рис. 1.42).

```
PS C:\Users\user> Add 5 7
12
PS C:\Users\user> Add "10" "201"
10201
PS C:\Users\user> Add 1.5 4
5.5
PS C:\Users\user> Add 4 1.5
6
PS C:\Users\user> Add 5 1.4
6
PS C:\Users\user>
```

Рис. 1.42. Варианты работы функции с различными типами данных

Примеры показывают, что программа по-разному себя ведет, принимая данные различных типов. Контекст выполнения функции строится в зависимости от типа данных. Она может складывать целые, числа с плавающей точкой. Если аргументы имеют данные различных типов, то программа их приводит к типу первого слагаемого. Вторая строка показывает, что при «сложении» строковых данных включается конкатенация (соединение) строк. Последние две строки демонстрируют правила округления и перевода вещественных чисел в целочисленную форму (рис. 1.42).

Основные достоинства PowerShell заключаются в реализации различных конвейеров. Функции как средства обработки играют здесь важную роль. С их помощью можно перебирать, анализировать, фильтровать и обсчитывать элементы потоковой информации, а также

разрабатывать новые командлеты. Рассмотрим работу функции в конвейере при поступлении потока данных. Для передачи потоковых данных в PowerShell служит переменная `$Input`, которая предназначена для хранения коллекции входящих объектов.

Создадим функцию «Sum», обеспечивающую суммирование элементов входящего потока (рис. 1.43).

```
PS C:\Users\user>  
PS C:\Users\user> Function Sum {  
>> $n=0  
>> ForEach ($i in $Input) { $n+=$i }  
>> $n  
>> }  
>>
```

Рис. 1.43. Функция для суммирования элементов входящего потока

Создадим входной поток из целых чисел от 1 до 10. В этом случае функция должна подсчитать сумму членов арифметической прогрессии (рис. 1.44).

```
PS C:\Users\user> 1..10|Sum  
55  
PS C:\Users\user>
```

Рис. 1.44. Запуск конвейера с ответом «55»

Запуск конвейера формирует ответ: подсчитанная сумма равна 55.

## 1.5. Примеры работ в Windows PowerShell

Квалифицированное использование PowerShell требует знаний аппаратного, программного и информационного обеспечения компьютерных систем. Рассмотрим несколько примеров определения некоторых характеристик компьютерных систем. Нужно отметить, что отдельные управляющие конструкции-конвейеры – громоздки, требуют внимательности и терпения. Ошибки даже в одном символе недопустимы.

**Получение информации о BIOS.** Вывод всех характеристик BIOS можно получить выполнением команды, приведенной на рис. 1.45.

```
Windows PowerShell
(С) Корпорация Майкрософт, 2009. Все права защищены.

PS C:\Users\user> Get-WMIObject Win32_BIOS|Select-Object -Property *

Status           : OK
Name             : BIOS Date: 02/05/10 19:18:52 Ver: 08.00.10
Caption          : BIOS Date: 02/05/10 19:18:52 Ver: 08.00.10
SMBIOSPresent   : True
  _GENUS         : 2
  _CLASS        : Win32_BIOS
  SUPERCLASS    : CIM_BIOSElement
  DYNASTY       : CIM_ManagedSystemElement
  RELPATH       : Win32_BIOS.Name="BIOS Date: 02/05/10 19:18:52 Ver: 08.00.10",SoftwareElementID="BIOS
5/10 19:18:52 Ver: 08.00.10",SoftwareElementState=8,TargetOperatingSystem=0,Version="
072009"
  _PROPERTY_COUNT : 27
  DERIVATION      : (CIM_BIOSElement, CIM_SoftwareElement, CIM_LogicalElement, CIM_ManagedSystemElement)
  SERVER         : PC00X0UN
  NAMESPACE      : root\cimv2
  PATH           : \\PC00X0UN\root\cimv2:Win32_BIOS.Name="BIOS Date: 02/05/10 19:18:52 Ver: 08.00.10",S
oftwareElementID="BIOS Date: 02/05/10 19:18:52 Ver: 08.00.10",SoftwareElementState=8,TargetOper
atingSystem=0,Version="ASUS_ - 1072009"
  BIOSCharacteristics : (7, 11, 12, 15...)
  BIOSVersion      : (_ASUS_ - 1072009, BIOS Date: 02/05/10 19:18:52 Ver: 08.00.10, BIOS Date: 02/05/10 19:18:52 Ver: 08.00.10)
```

Рис. 1.45. Вывод всех характеристик BIOS

Служебные характеристики для WMI (Windows Management Instrumentation), имена которых начинаются двумя знаками подчеркивания, можно убрать, если ввести параметр

-ExcludeProperty \_\_\* (рис. 1.46).

```

PS C:\Users\user> Get-WMIObject Win32_BIOS|Select-Object -Property * -ExcludeProperty __*

Status                : OK
Name                  : BIOS Date: 02/05/10 19:13:52 Ver: 03.00.10
Caption               : BIOS Date: 02/05/10 19:13:52 Ver: 03.00.10
SMBIOSPresent        : True
BiosCharacteristics   : (7, 11, 12, 15...)
BIOSVersion           : (_ASUS_ - 1072009, BIOS Date: 02/05/10 19:13:52 Ver: 03.00.10, BIOS Date: 02/05
                      : 03.00.10)
BuildNumber           :
CodeSet               :
CurrentLanguage       : eng
Description           : BIOS Date: 02/05/10 19:13:52 Ver: 03.00.10
IdentificationCode    :
InstallableLanguages : 6
InstallDate           :
LanguageEdition       :
ListOfLanguages       : (eng, fra, ger, che...)
Manufacturer          : American Megatrends Inc.
OtherTargetOS         :
PrimaryBIOS           : True
ReleaseDate           : 20110826000000.000000+000
SerialNumber          : System Serial Number
SMBIOSBIOSVersion     : 0409

```

Рис. 1.46. Вывод характеристик BIOS, с исключением заданных

**Вывод характеристик ОС.** Список основных характеристик (дата установки, загрузочное устройство и т. п.) операционной системы можно получить при обращении к экземпляру класса WMI Win32\_OperatingSystem. Конвейер блокирует выдачу служебных свойств WMI (рис. 1.47).

```

PS C:\Users\user> Get-WMIObject Win32_OperatingSystem|Select-Object -Property * -ExcludeProperty __*

Status                : OK
Name                  : Microsoft Windows 7 Максимальная [C:\Windows\Device\Harddis
FreePhysicalMemory    : 2462136
FreeSpaceInPagingFiles : 3780004
FreeVirtualMemory     : 6049656
BootDevice            : \Device\HarddiskVolume1
BuildNumber           : 7691
BuildType              : Multiprocessor Free
Caption               : Microsoft Windows 7 Максимальная
CodeSet               : 1251
CountryCode           : 7
CreationClassName     : Win32_OperatingSystem
CSCreationClassName   : Win32_ComputerSystem

```

Рис. 1.47. Основные характеристики операционной системы

## Получение информации о физической памяти компьютера

```
PS C:\Users\user> Get-WMIObject Win32_PhysicalMemory | Select-Object -Property * -Exclude
BankLabel           : BANK0
Capacity            : 4294967296
Caption             : Physical Memory
CreationClassName   : Win32_PhysicalMemory
DataWidth           : 64
Description         : Physical Memory
DeviceLocator       : DIMM0
FormFactor          : 3
HotSwappable        :
InstallDate         :
InterleaveDataDepth : 0
InterleavePosition  : 0
Manufacturer        : Samsung
MemoryType          : 0
Model               :
Name                : Physical Memory
OtherIdentifyingInfo :
PartNumber          : M378B5273CH0-CH9
PositionInRow       : 1
PoweredOn           :
Removable           :
Replaceable         :
SerialNumber        : 63FB38E
```

Рис. 1.48. Информация о физической памяти компьютера

Экземпляры класса Win32\_PhysicalMemory позволяют определить характеристики памяти компьютера (рис. 1.48).

## **ПРАКТИЧЕСКАЯ ЧАСТЬ.**

### **Задание 1. Командная оболочка PowerShell.**

#### **Операционная система Windows**

*Задание:* студентам предлагается выполнить задания по темам 1.1 и 1.2 теоретической части практикума.

##### 1.1. Начало работы в среде PowerShell

1. Загрузить командную оболочку PowerShell и запустить `dir`.
2. Просмотреть работу средств PowerShell по указанным псевдонимам: `cd`, `ls`, `copy`, `del`, `dir`, `echo`, `erase`, `more`, `popd`, `pushd`, `ren`.
3. Опробовать работу PowerShell в режиме калькулятора для вычисления простых арифметических выражений: пять арифметических выражений.
4. Опробовать работу PowerShell в режиме калькулятора для вычисления простых переменных: пять переменных и одна переменная итоговая.

##### 1.2. Структура пакета PowerShell и его справочная система

1. Вызвать обобщенную справку по пакету PowerShell, набрав в командной строке `Get-Help` без параметров. Просмотреть справочные данные по команде `help`. Ознакомиться с контекстом команд. Первая команда выдаёт одностраничную справку, а последняя команда дает многостраничную справку.
2. Отобразить все разделы справочной системы, набрав команду `Get-Help*`. Параметр `*` является шаблоном, обозначающим «любое сочетание символов».
3. Ознакомиться со структурой PowerShell по перечню разделов справки, набрав по две команды, указанные в качестве примеров, по каждому из разделов. Посмотреть, как меняется содержание справочных данных, если в команду справки включаются параметры `-detailed` или `-full`.
4. Просмотреть справку по командлету `Get-Process`, отображающая процессы, активизированные в локальном компьютере пользователя. Для этого набираем в командной строке команду

PS C:\ users \student> Get-Help Get-process -full

Ознакомиться с перечнем характеристик процессов, активизированных в компьютере.

5. Просмотреть справку по командлету Get-Process, набрав команду  
PS C:\ users \student >Get-process \ ?

Сравнить полученную справку с предыдущими данными пункта 4.

## **Задание 2. Командлеты командной оболочки PowerShell. Операционная система Windows**

*Задание:* студентам предлагается выполнить задания по темам 1.3, 1.3.1 и 1.3.2 теоретической части практикума.

### 1.3.1. Работа с дисками

1. Получить список дисков, доступных пользователю из среды PowerShell.
2. Изучить командлеты Get-Location и Set-Location и их псевдонимы pwd, cd, chdir.
3. Получить список всех провайдеров оболочки командлетом Get-PSProvider
4. Загрузить в систему текущий или рабочий каталог (командлет Get-Location, псевдоним cd).

### 1.3.2. Работа с файловой системой

1. Получить список подкаталога, одного из выбранного каталога, без файлов, входящих в подкаталог.
2. Осуществить поиск файлов в выбранном каталоге, задав фильтр по имени файла и выбрав расширение файла.
3. Создать два подкаталога в текущем каталоге пользователя.
4. В одном из подкаталогов создать два новых текстовых файла. Текст в одном файле фамилия, в другом имя и отчество.
5. Сделать копию двух файлов в выбранном каталоге.

## **Задание 3. Конфигурация командной оболочки PowerShell.**

## **Операционная система Windows**

*Задание:* студентам предлагается выполнить задания по темам 1.3 и 1.3.3 теоретической части практикума.

### **1.3.3. Работа с конфигурацией оболочки**

1. Вызвать диалоговое окно оболочки PowerShell и установить параметры команд окна: размер курсора, фон, шрифт.

Изучить рубрики: «Общие», «Шрифт», «Расположение», «Цвета».

2. Изменить цвет фона и цвет текста с помощью составных переменных. Перед изменениями параметров составных переменных сохранить значение объекта RawUI.

3. Изменить параметры оболочки WindowSize (см. рис. 1.22) на указанные в описании. Восстановить прежние значения WindowSize.

4. Изменить наименование командного окна PowerShell и восстановить прежнее название.

5. Создать «Приглашение», соответствующее командной строке Windows «cmd.exe».

Сформулировать и запомнить определение «Подвыражение».

6. Определить место расположение и имя файла профиля переменной «Sprofile».

Сформулировать и запомнить определение «Профиль».

### **Отчет по выполненной практической работы:**

1. Выполненная работа оформляется в электронном виде (формат А4)

2. Электронный отчет по выполненной работе состоит:

-Название работы.

- Назначение Windows PowerShell.

-Задание на работу.

-Выполненная работа (пункт задания и копия с экрана: результат работы по пункту задания).



## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бэкон, Д. Операционные системы/ Д. Бэкон, Т. Харрис. – Санкт-Петербург : Питер; Киев; Издательская группа ВНУ, 2004 . – 800 с.
2. Гордеев, А.В. Операционные системы: учебник для вузов / А. В. Гордеев. – 2-е изд. – Санкт-Петербург : Питер, 2007. – 416 с.
3. Назаров, С. В. Операционные системы. Практикум : учебное пособие / С. В. Назаров, Л. П. Гудыно, А. А. Кириченко. – Москва : КНОРУС, 2012. – 376 с.
4. Таненбаум, Э. Современные операционные системы / Э. Таненбаум. – 3-е изд. – Санкт-Петербург : Питер, 2012. – 1120 с.