Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |  |
| --- | --- |
| Институт | математики и компьютерных наук |
| Кафедра | компьютерной безопасности |

ОТЧЕТ

Лабораторная работа №3

по дисциплине «Оболочка командной строки Windows PowerShell»

Выполнил:

Пронин Владимир Иванович,

студент 2 курса

группы КМБ-с-о-23-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

Проверено с оценкой:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

Ставрополь, 2025

Вывести содержимое каталога Windows (для бригад 5 и 10 – и подкаталогов) по указанному в табл. 5 формату на экран и в текстовый файл.

Таблица 5. Варианты заданий

|  |  |  |  |
| --- | --- | --- | --- |
| Номера | Что выводить (имена, размер, дата создания, атрибуты) | Сортировать по | Условие отбора |
| 1, 6 | Только файлы | По размеру | Размер> 10000 |
| 2, 7 | Файлы и подкаталоги | По дате | Первые буквы имени SY |
| 3, 8 | Только подкаталоги | Именам | Последняя буква имени S или T |
| 4, 9 | Только файлы bmp | По размеру | Размер >50000 |
| 5, 10 | Только файлы jpg | Именам | Любые |

dir -Recurse -Directory sy\* | Sort-Object -Property LastWriteTime

Вывести в текстовый файл список свойств процесса, возвращаемый командлетом Get-process и на экран – их общее количество.

Get-Process | Get-Member | Select-Object -ExpandProperty Definition

(Get-Process | Get-Member | Select-Object -ExpandProperty Definition).count

Cоздать текстовый файл, содержащий список выполняемых процессов, упорядоченный по возрастанию указанного в табл.6 параметра. Имена параметров процессов указаны в табл. 6.

Таблица 6. Варианты заданий

|  |  |  |  |
| --- | --- | --- | --- |
| Номера | Список выводимых параметров процессов | Сортировать по значению параметра | Вывести процессы, у которых |
| 7, 8 | Имя процесса, PriorityClass, ProductVersion, Id | Имя процесса | Id > 100 |

Get-Process | Where-Object {$\_.Id -gt 100} | Sort-Object ProcessName

| Select-Object ProcessName, PriorityClass, ProductVersion,Id

Cоздать HTML-файл, содержащий список выполняемых процессов, упорядоченный по возрастанию указанного в табл.5 параметра. Имена параметров процессов указаны в табл. 5

dir -Recurse -Directory sy\* | Sort-Object -Property LastWriteTime

| ConvertTo-Html | Out-File "./FiltredProcess.html"

Найти суммарный объем всех графических файлов (bmp, jpg), находящихся в каталоге Windows и всех его подкаталогах

dir -Include \*.bmp, \*.jpg -Recurse -File

| Measure-Object -Property Length -Sum

| Select-Object -ExpandProperty Sum

Вывести на экран сведения о ЦП компьютера.

Get-WmiObject -Class Win32\_Processor

| Select-Object Name, Manufacturer, NumberOfCores, MaxClockSpeed

1. Нахождения среди выполняющихся процессов имен процессов с наименьшим значением BasePriority

$min = (Get-Process | Sort-Object BasePriority | Select-Object -First 1).BasePriority

Get-Process | Where-Object {$\_.BasePriority -eq $min}

2. Нахождения среди выполняющихся процессов имен процессов, у которых значения параметра WorkingSet одинаковы

Get-Process | Group-Object WorkingSet | Where-Object {$\_.Count -gt 1}

| ForEach-Object { Write-Host $\_.Name, $\_.Group.ProcessName}

Отлично, тогда давайте углубимся в более точные и техничные объяснения:

**1. Типы команд PowerShell (PS).**

* **Cmdlets:** .NET-классы, реализующие интерфейс Cmdlet. Они являются фундаментальными блоками PowerShell. Cmdlets получают входные данные в виде объектов, обрабатывают их и возвращают другие объекты. Имена следуют шаблону Verb-Noun, что способствует обнаружению и пониманию.
* **Functions:** Блоки кода PowerShell, определенные пользователем или встроенные. Функции могут принимать параметры, выполнять произвольную логику и возвращать значения. Они позволяют инкапсулировать сложные задачи и повторно использовать код. Они могут быть определены в скриптах или модулях.
* **Scripts:** Текстовые файлы (.ps1) содержащие последовательность команд PowerShell. Скрипты используются для автоматизации задач, развертывания конфигураций и выполнения пакетных операций. Они могут содержать функции, управляющие структуры и другую логику.
* **Aliases:** Альтернативные имена для командлетов, функций, скриптов или внешних исполняемых файлов. Псевдонимы предназначены для сокращения ввода, улучшения читаемости или обеспечения совместимости с другими оболочками.
* **External Programs:** Исполняемые файлы (.exe, .com, .bat и др.), которые запускаются из PowerShell. PowerShell предоставляет возможность взаимодействия с внешними программами, передавая им аргументы и перенаправляя ввод и вывод.

**2. Имена и структура командлетов.**

* **Verb-Noun Naming Convention:** Командлеты PowerShell следуют строгой схеме именования Verb-Noun. Это обеспечивает согласованность и упрощает обнаружение команд.
  + **Verb:** Указывает на действие, которое выполняет командлет (например, Get, Set, New, Remove, Invoke, Out). Microsoft предоставляет список утвержденных глаголов.
  + **Noun:** Указывает на объект, над которым выполняется действие (например, Process, Service, Item, Content, EventLog).
* **Examples:**
  + Get-Process: Возвращает объекты Process.
  + Set-Content: Изменяет содержимое объекта типа Item (обычно файл).
  + New-Item: Создает новый объект типа Item (например, файл или каталог).

**3. Псевдонимы команд.**

* **Purpose:**
  + **Conciseness:** Сокращение длины команд для ускорения ввода.
  + **Familiarity:** Предоставление знакомых имен команд для пользователей, переходящих из других оболочек (например, Linux bash).
  + **Compatibility:** Сохранение совместимости со старыми скриптами, использующими псевдонимы.
* **Management:**
  + Get-Alias: Отображает список доступных псевдонимов.
  + Set-Alias: Создает новые псевдонимы или изменяет существующие.
  + Remove-Alias: Удаляет псевдонимы.

**4. Просмотр структуры объектов.**

* **Get-Member:** Командлет, позволяющий исследовать типы объектов, которые PowerShell обрабатывает. Он предоставляет информацию о свойствах, методах и событиях объектов.
* **Object Structure:**
  + **Properties:** Данные, хранящиеся в объекте. Они описывают характеристики объекта (например, ProcessName, Id, CPU для объекта Process). Доступ к свойствам осуществляется через оператор ., например $process.ProcessName.
  + **Methods:** Действия, которые можно выполнить с объектом. Они представляют собой функции, связанные с объектом (например, Kill() для объекта Process). Методы вызываются с помощью оператора (), например $process.Kill().

**5. Фильтрация объектов в конвейере. Блок сценария.**

* **Where-Object:** Командлет, фильтрующий объекты, проходящие по конвейеру, на основе логического условия.
* **Script Block ({}):** Блок кода PowerShell, который выполняется для каждого входного объекта. Блок сценария должен возвращать $true для объектов, которые должны быть включены в выходной поток, и $false для объектов, которые должны быть исключены.
* **Automatic Variable $\_:** Внутри блока сценария $\_ (или $PSItem) представляет текущий объект, обрабатываемый Where-Object. Свойства этого объекта доступны через оператор ..
* **Example:** Get-Process | Where-Object {$\_.CPU -gt 1 -and $\_.WorkingSet -gt 10MB} (фильтрует процессы с использованием CPU более 1 секунды и рабочим набором более 10MB).

**6. Какую информацию выводит команда Get-Help \* ?**

* Get-Help \*: Выводит краткую справку по *каждому* командлету, функции, поставщику и исполняемому файлу в текущей среде PowerShell. Это очень широкий запрос, который может занять значительное время и вывести огромное количество информации. Он полезен для обзора доступных команд, но обычно лучше использовать Get-Help с конкретным именем команды.

**7. Командлеты для форматирования выводимой информации.**

* **Format-Table:** Отображает выходные данные в виде таблицы, позволяя указать свойства, которые должны быть показаны в столбцах. -AutoSize автоматически подгоняет ширину столбцов.
* **Format-List:** Отображает выходные данные в виде списка свойств и их значений. Полезен для отображения всех свойств объекта или для объектов со многими свойствами.
* **Format-Wide:** Отображает только одно свойство каждого объекта в широком формате, занимающем всю доступную ширину терминала. Полезен для компактного отображения простых списков.
* **Purpose:** Управление внешним видом выходных данных для улучшения читаемости и представления информации.

**8. Перенаправление выводимой информации.**

* **> (Output Redirection Operator):** Перенаправляет стандартный вывод команды в указанный файл, перезаписывая содержимое файла.
* **>> (Append Redirection Operator):** Перенаправляет стандартный вывод команды в указанный файл, добавляя содержимое в конец файла.
* **Out-File Cmdlet:** Более гибкий командлет для записи вывода в файл. Предоставляет параметры для указания кодировки (-Encoding), добавления в файл (-Append), ширины строк (-Width) и других опций.
* **Error Handling:** По умолчанию, операторы > и >> не перенаправляют ошибки. Для перенаправления ошибок используйте 2> (для перенаправления стандартного потока ошибок) или \*> (для перенаправления всех потоков).

**9. Управляющие инструкции PS.**

* **if, elseif, else:** Условные операторы для выполнения кода в зависимости от истинности или ложности заданных условий.
* **for:** Цикл, выполняющий блок кода заданное количество раз, обычно с использованием счетчика.
* **foreach:** Цикл, перебирающий элементы коллекции (массива, списка, и т.д.) и выполняющий блок кода для каждого элемента.
* **while:** Цикл, выполняющий блок кода до тех пор, пока заданное условие остается истинным.
* **do...while и do...until:** Циклы, аналогичные while, но гарантированно выполняющиеся хотя бы один раз.
* **switch:** Оператор многозначного выбора, позволяющий выполнять разные блоки кода в зависимости от значения выражения.

**10. Назначение регулярных выражений.**

* **Purpose:** Регулярные выражения (regex) - это мощный инструмент для сопоставления, поиска, замены и извлечения текстовых шаблонов. Они используются для:
  + **Data Validation:** Проверка формата входных данных (например, адресов электронной почты, номеров телефонов).
  + **Text Parsing:** Извлечение конкретной информации из текста.
  + **String Manipulation:** Замена или удаление текста на основе шаблонов.
  + **Log Analysis:** Поиск определенных событий или ошибок в файлах журналов.
* **PowerShell Integration:** PowerShell имеет встроенную поддержку регулярных выражений через операторы -match, -replace, -split и другие.

**11. Сохранение данных в текстовом файле и html-файле.**

* **Text File:**
  + Out-File: Для записи данных в простой текстовый файл. Укажите кодировку (-Encoding) для правильной обработки символов.
* **HTML File:**
  + ConvertTo-Html: Преобразует объекты PowerShell в HTML-таблицу. Можно добавить CSS стили для улучшения внешнего вида.
  + Out-File: Используется для сохранения сгенерированного HTML-кода в файл с расширением .html.

**12. Получение справочной информации в PS.**

* **Get-Help <Cmdlet-Name>:** Отображает справку для указанного командлета.
* **Parameters:**
  + -Examples: Показывает примеры использования.
  + -Detailed: Отображает подробное описание командлета, его параметров и входных/выходных данных.
  + -Full: Отображает всю доступную информацию, включая описание, параметры, примеры, связанные ссылки и технические детали.
  + -Online: Открывает справку в браузере на сайте Microsoft Docs.
* **Update-Help:** Загружает последние версии файлов справки с сайта Microsoft, чтобы ваша локальная справка была актуальной.

**13. Как создать массив в PS?**

* **Array Literals:**
  + $myArray = @(): Создает пустой массив.
  + $myArray = 1, 2, 3: Создает массив с целыми числами.
  + $myArray = "one", "two", "three": Создает массив строк.
* **Array Constructor:**
  + $myArray = @(1, 2, 3): Эквивалентно записи с литералами.
* **New-Object:** Используется для создания массивов с определенным типом и размером (менее распространенный способ).

**14. Как объединить два массива?**

* **+ Operator:** Используйте оператор сложения (+) для объединения двух массивов.
* **Example:** $array1 = 1, 2, 3; $array2 = 4, 5, 6; $combinedArray = $array1 + $array2
* **Important:** При объединении массивов создается новый массив.

**15. Как увеличить размер созданного в PS массива?**

* **Fixed Size Arrays:** Массивы в PowerShell по умолчанию имеют фиксированный размер. Нельзя просто добавить элемент в существующий массив, увеличив его размер.
* **Alternatives:**
  + **Create a New Array:** Создайте новый массив большего размера и скопируйте в него элементы из старого массива.
  + **Use ArrayList:** ArrayList - это динамический массив, который может автоматически изменять свой размер.
  + **Example (ArrayList):**
* $arrayList = New-Object System.Collections.ArrayList
* $arrayList.Add(1)
* $arrayList.Add(2)

[array]$myArray = $arrayList.ToArray([int]) # Convert ArrayList to a fixed-size array (optional)

powershell

* + **Use List:** Более современный и типобезопасный способ работы с динамическими массивами.
  + $list = New-Object System.Collections.Generic.List[int]
  + $list.Add(1)
  + $list.Add(2)

[array]$myArray = $list.ToArray() # Convert List<T> to a fixed-size array (optional)

powershell

**16. Как ввести данные в массив?**

* **Read-Host:** Получает данные от пользователя из консоли.
* **Example:**

$myArray = @()

while ($true) {

$input = Read-Host "Введите значение для массива (или нажмите Enter, чтобы закончить)"

if ($input -eq "") { break }

$myArray += $input

}

powershell

**17. Использование командлета Out-Null.**

* **Purpose:** Out-Null отбрасывает выходные данные команды. Он используется для:
  + **Suppressing Output:** Предотвращения отображения нежелательной информации в консоли.
  + **Performance Optimization:** Иногда, отбрасывание вывода может немного ускорить выполнение команды (хотя это часто незначительно).
* **Example:** Get-Process | Out-Null (получает список процессов, но ничего не отображает в консоли).

**18. Оператор PowerShell –match.**

* **Purpose:** Оператор -match используется для проверки, соответствует ли строка заданному регулярному выражению.
* **Syntax:** $string -match "<regex>"
* **Returns:** Возвращает $true, если строка соответствует шаблону, и $false в противном случае.
* **Automatic Variable $Matches:** Если совпадение найдено, оператор -match заполняет автоматическую переменную $Matches информацией о найденном совпадении. $Matches[0] содержит всю найденную строку, а $Matches[1], $Matches[2] и т.д. содержат результаты захвата групп (если они есть в регулярном выражении).

**19. Использование символа ^ в командлетах.**

* **Regular Expression Anchor:** В регулярных выражениях символ ^ является якорем, обозначающим начало строки.
* **Example:** Get-Content file.txt | Where-Object {$\_ -match "^Error:"} (находит все строки в файле file.txt, начинающиеся с “Error:”).

**20. Использование символа $ в командлетах.**

* **Regular Expression Anchor:** В регулярных выражениях символ $ является якорем, обозначающим конец строки.
* **Example:** Get-Content file.txt | Where-Object {$\_ -match "Success$"} (находит все строки в файле file.txt, заканчивающиеся на “Success”).

**21. Количественные модификаторы (квантификаторы).**

* **Purpose:** Квантификаторы управляют количеством повторений символа, группы или класса символов в регулярном выражении.
* **Common Quantifiers:**
  + \*: Ноль или более раз.
  + +: Один или более раз.
  + ?: Ноль или один раз (необязательный символ).
  + {n}: Ровно n раз.
  + {n,}: Не менее n раз.
  + {n,m}: От n до m раз (включительно).
* **Example:** [0-9]{3}-[0-9]{2}-[0-9]{4} (соответствует формату даты “XXX-XX-XXXX”).

**22. Использование групп захвата.**

* **Purpose:** Группы захвата позволяют извлекать определенные части текста, соответствующие частям регулярного выражения.
* **Syntax:** Заключите часть регулярного выражения в круглые скобки ().
* **Accessing Captured Groups:** После выполнения оператора -match, найденные группы будут доступны в автоматической переменной $Matches. $Matches[0] содержит всю найденную строку, а $Matches[1], $Matches[2] и т.д. содержат результаты захвата групп (в порядке их появления в регулярном выражении).
* **Example:**
* $string = "Имя: Джон, Возраст: 30"
* if ($string -match "Имя: (.\*), Возраст: (.\*)") {
* $name = $Matches[1]
* $age = $Matches[2]
* Write-Host "Имя: $name, Возраст: $age"

}

powershell

**23. Командлеты для измерения свойств объектов.**

* **Measure-Object:** Вычисляет статистические значения для числовых свойств объектов. Может подсчитывать количество объектов (Count), вычислять среднее (Average), минимальное (Minimum), максимальное (Maximum) и сумму (Sum).
* **Example:**
* Get-ChildItem | Measure-Object -Property Length -Sum # Вычисляет общий размер всех файлов в текущем каталоге

Get-Process | Measure-Object -Property CPU -Average # Вычисляет среднее использование CPU всеми процессами

powershell

* **Calculation Without Numerical Property:** You need to get the numerical values of what you’d like to summarize, and send that to Measure-Object

Get-ChildItem | ForEach-Object {$\_.length} | Measure-Object -Sum # Calculates total bytes of files in the directory

powershell

Эти ответы более техничны и содержат больше деталей, чем предыдущие. Они предполагают, что у вас уже есть базовое понимание концепций командной строки и скриптов. Надеюсь, это то, что вам нужно!