

Windows PowerShell – уникальная командная оболочка и среда для создания сценариев Windows

The image displays two overlapping windows from Windows PowerShell ISE. The background window shows the output of the `Get-Process` command, listing system processes with columns for Handles, NPM(K), PM(K), WS(K), CPU(s), and Id. The foreground window shows a PowerShell script named `New-Fixture.ps1` designed to create test files and directories. The script includes a `Describe` block for testing, a `function Create-File` definition, and a main execution block that uses `Split-Path` to generate paths and `Create-File` to create files with specific content.

Terminal Output (Background Window):

```
This command gets all of the items in the C:\Windows directory and its subdirectories that are, except for those with a .png file name extension.
```

Terminal Output (Foreground Window):

```
PS C:\Users\andr> Get-Process
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id
369	33	126564	105796	17,95	2512
272	15	8072	11348	0,47	10164
233	25	40232	25204	1,20	8596
237	24	29688	28552	16,19	9340
460	36	23756	33652	2,52	5172
211	15	3568	9028	0,61	9384
259	15	3504	11480	2400	2400
196	20	3408	9900	2392	2392
207	14	4464	17528	2,52	4576
121	8	1256	5368	2424	2424
3335	102	138012	40900	1884	1884
553	41	16140	23280	1,20	8268
124	10	1784	6532	2408	2408
44	3	436	2160	0,02	9924
91	9	1212	6452	0,41	9356
84	7	4892	4784	0,02	6248
169	12	3384	12956	0,47	10156
349	25	9928	22736	2,28	3780

```
PS C:\WINDOWS\System32>
```

Script Content (Foreground Window):

```
New-Fixture.ps1 X
64 [String]$Path = $PWD,
65 [Parameter(Mandatory=$true)]
66 [String]$Name
67 )
68 #region File contents
69 #keep this formatted as is. the format is output to the file as is, including indentation
70 $ScriptCode = "function $Name {r`n`r`n}"
71
72 $TestCode = 'Here = Split-Path -Parent $MyInvocation.MyCommand.Path
73 $Ssut = (Split-Path -Leaf $MyInvocation.MyCommand.Path) -replace '\\.Tests\\.','.'
74 . "$Here$Ssut"
75
76 Describe "#name#" {
77     It "does something useful" {
78         $true | Should Be $false
79     }
80 } -replace "#name#", $Name
81
82 #endregion
83
84 $Path = $ExecutionContext.SessionState.Path.GetUnresolvedProviderPathFromPSPath($Path)
85
86 Create-File -Path $Path -Name "$Name.ps1" -Content $ScriptCode
87 Create-File -Path $Path -Name "$Name.Tests.ps1" -Content $TestCode
88
89
90 function Create-File ($Path,$Name,$Content) {
91     if (-not (&$SafeCommands['Test-Path'] -Path $Path)) {
92         &$SafeCommands['New-Item'] -ItemType Directory -Path $Path | &$SafeCommands['Out-Null']
93     }
94 }
```

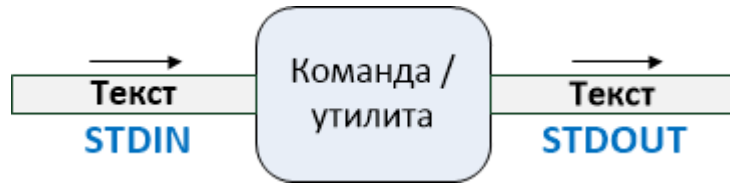
Зачем еще один язык?

Цель: создать оболочку и среду для написания и выполнения сценариев, которая:

- а) идеально подходила бы для Windows;
- б) превосходила бы все остальные оболочки для других операционных систем по функциональности, расширяемости и простоте в использовании.

Командный язык (оболочка) vs Язык сценариев

Командный язык (shell language) – для программирования на базе инструментальных средств (softwaretools).



Одна команда = одна функция



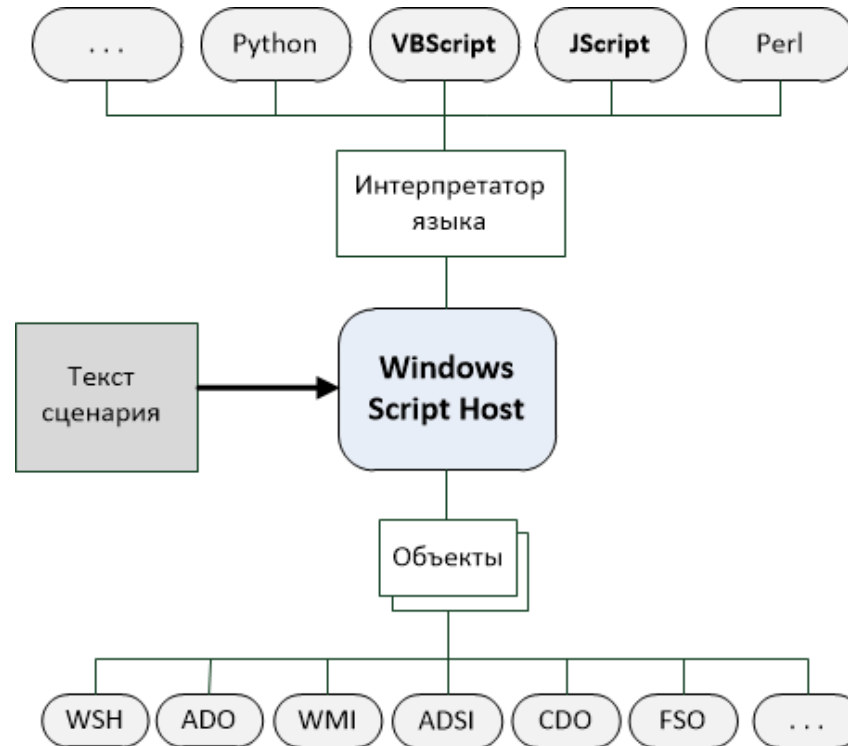
Командный язык для поддержки программных конвейеров

Программный конвейер – пример проектирования программ из независимых модулей. Модульное построение – мощная стратегия управления сложностью.

Прохождение по конвейеру потока символов хорошо подходит для Unix, где все настройки хранятся в виде текста.

Командный язык (оболочка) vs Язык сценариев

Язык сценариев (scripting language) – программирование в традиционном стиле (без конвейеров), но с использованием внешних объектов.

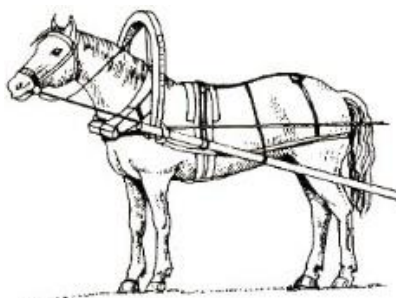


Объекты имеют структуру, их намного легче использовать в программах, чем текстовые данные.

Предыстория

Что было в Windows до PowerShell:

1. **Командная оболочка + пакетные файлы (батники) = интерпретатор cmd.exe**



Стиль Unix:

+ конвейеры

- трудно работать с объектами

2. **Сценарии на языках VBScript, JScript (Python, Perl, ...) = Windows Script Host**



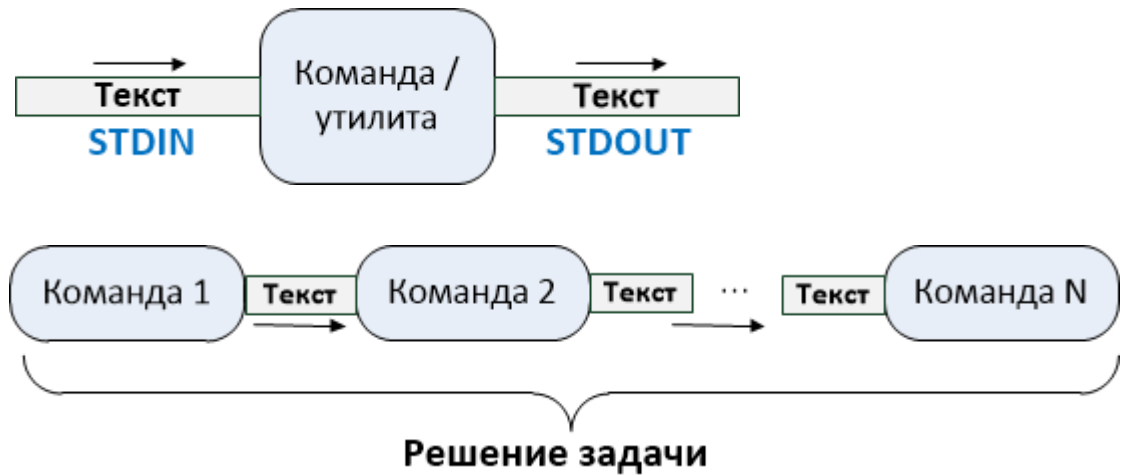
Стиль Windows:

+ работа с объектами

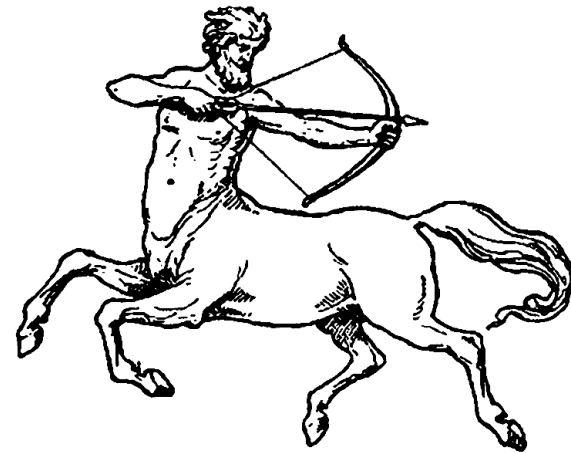
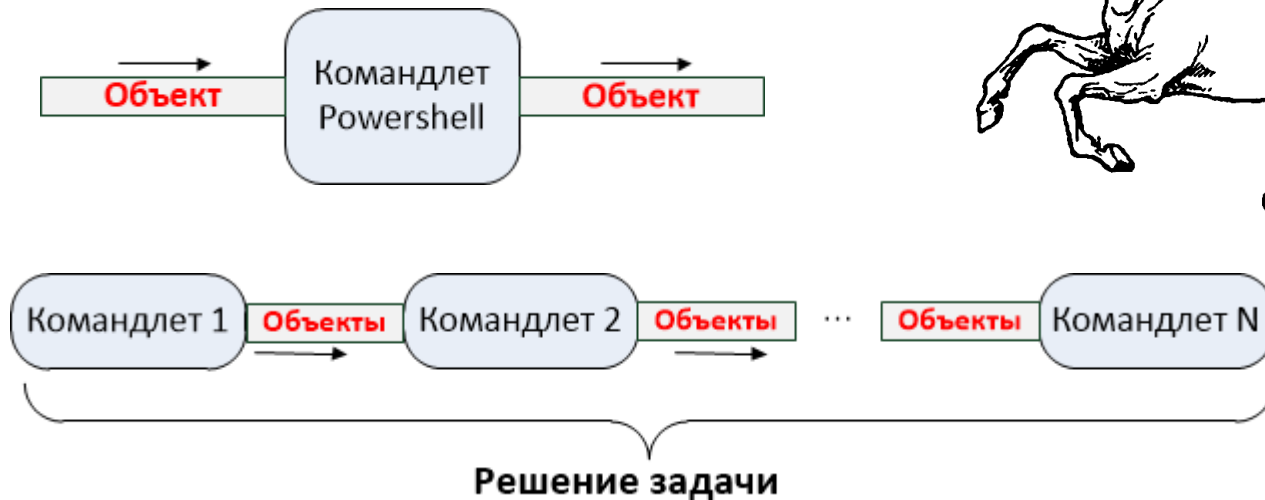
- нет конвейеров

PowerShell: гибрид Unix & Windows

Bash, cmd.exe, ...



Windows Powershell



PowerShell: гибрид Unix & Windows

Возможности

- Конвейерная обработка объектов (а не текста)
- Прямой доступ из командной строки к внешним объектам (COM, WMI, ...)
- Работа с разнородными источниками данных (реестр, сертификаты, ...) по принципу файловой системы
- Унифицированная структура встроенных команд (*Действие-Объект*)
- Возможность расширения встроенного набора команд
- Поддержка знакомых команд из других оболочек через псевдонимы (ls, dir, ...)

Взято лучшее из других языков:

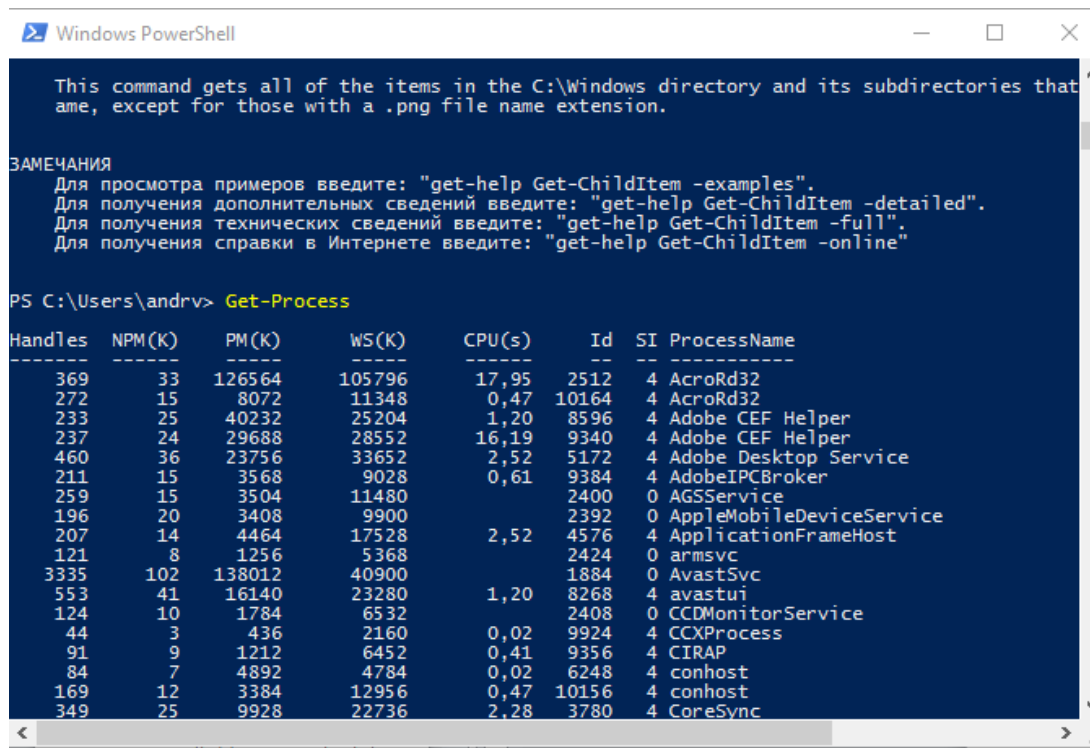
Bash, Ksh – конвейеризация

AS400/VMS – стандартные названия команд

TCL, WSH – встраиваемость и поддержка нескольких языков

PERL, Python – выразительность и стиль

Знакомство с оболочкой



```
Windows PowerShell
This command gets all of the items in the C:\Windows directory and its subdirectories that
ame, except for those with a .png file name extension.

ЗАМЕЧАНИЯ
Для просмотра примеров введите: "get-help Get-ChildItem -examples".
Для получения дополнительных сведений введите: "get-help Get-ChildItem -detailed".
Для получения технических сведений введите: "get-help Get-ChildItem -full".
Для получения справки в Интернете введите: "get-help Get-ChildItem -online"

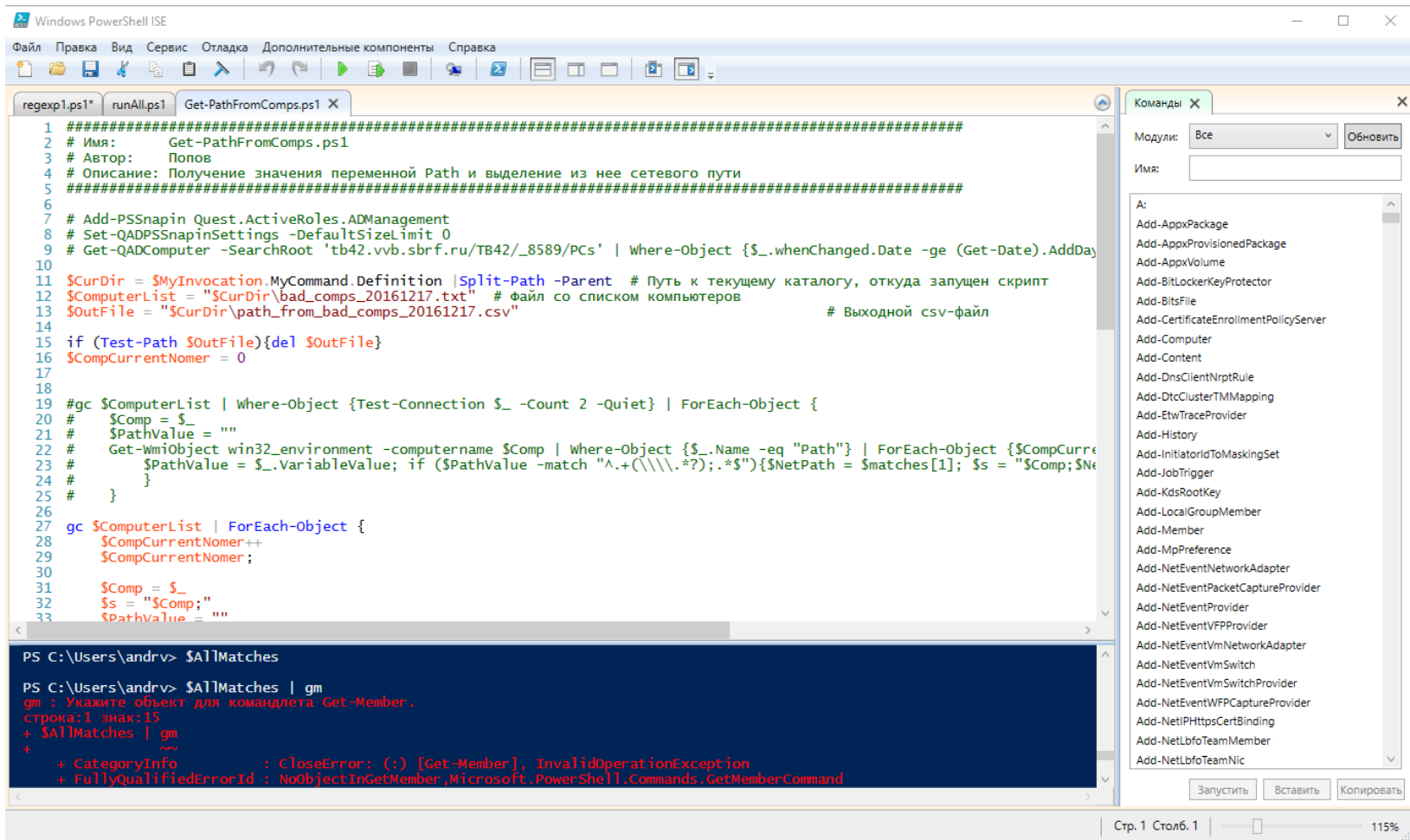
PS C:\Users\andr> Get-Process

Handles  NPM(K)  PM(K)  WS(K)  CPU(s)  Id  SI ProcessName
-----  -
369      33     126564 105796 17,95   2512  4 AcroRd32
272      15      8072   11348  0,47   10164  4 AcroRd32
233      25     40232  25204  1,20   8596  4 Adobe CEF Helper
237      24     29688  28552  16,19  9340  4 Adobe CEF Helper
460      36     23756  33652  2,52   5172  4 Adobe Desktop Service
211      15      3568   9028  0,61   9384  4 AdobeIPCBroker
259      15      3504   11480  2400  0 AGSService
196      20      3408   9900  2392  0 AppleMobileDeviceService
207      14      4464   17528  2,52  4576  4 ApplicationFrameHost
121      8       1256   5368  2424  0 armsvc
3335     102    138012 40900  1884  0 AvastSvc
553      41     16140  23280  1,20  8268  4 avastui
124      10     1784   6532  2408  0 CCDMonitorService
44       3       436    2160  0,02  9924  4 CCXProcess
91       9      1212   6452  0,41  9356  4 CIRAP
84       7      4892   4784  0,02  6248  4 conhost
169      12     3384   12956  0,47  10156  4 conhost
349      25     9928   22736  2,28  3780  4 CoreSync
```

Powershell.exe

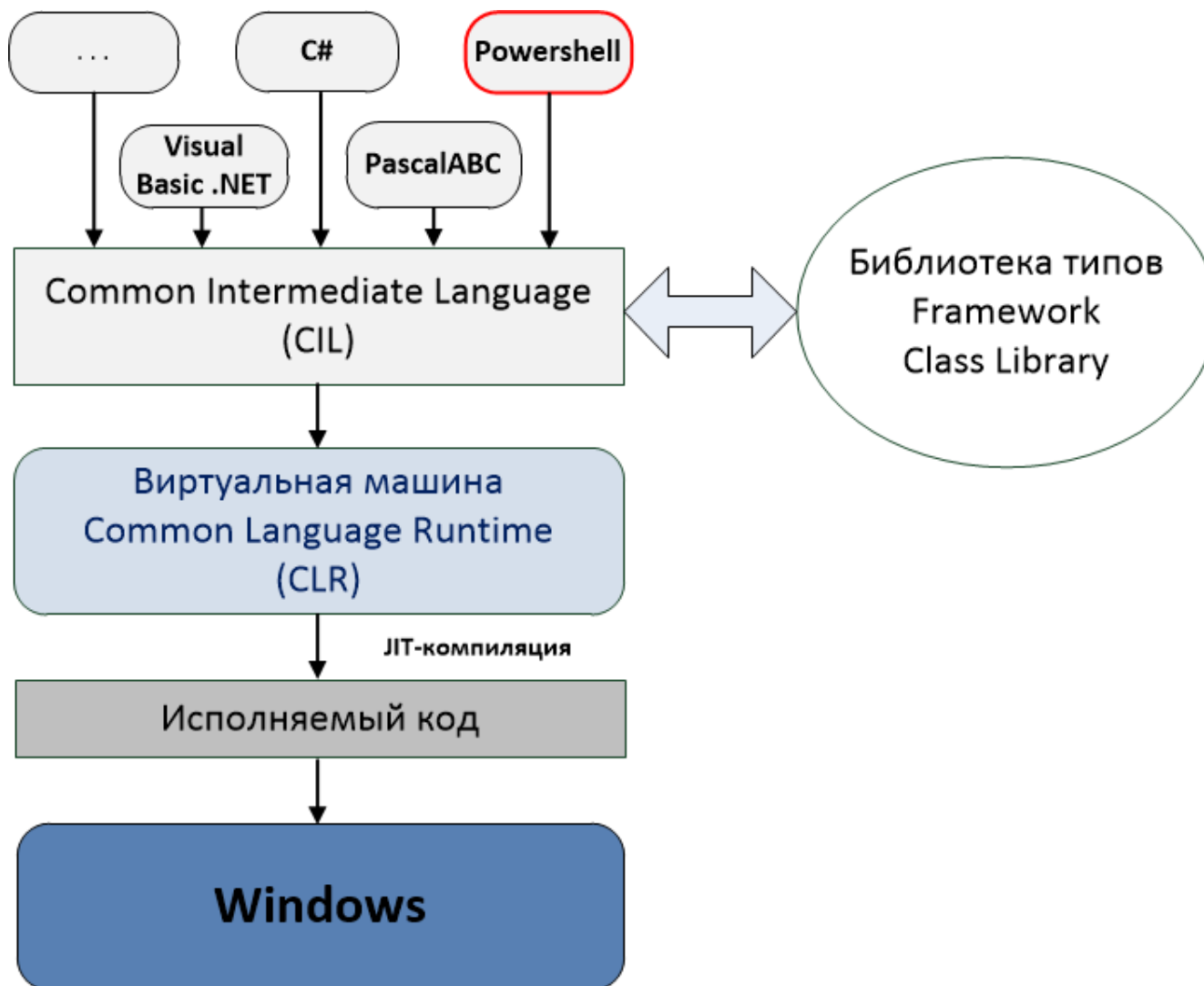
- Псевдонимы для команд
- Встроенная помощь
- Автозавершение ввода
- Стандарт имен команд
- Перенаправление ввода/вывода

Интегрированная среда сценариев



Powershell_ISE.exe

Фундамент PowerShell: .NET Framework



Все в PowerShell – объекты

Объекты .NET Framework – самодокументируемые

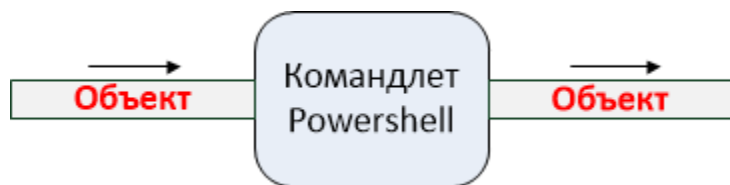
Командлет **Get-Member** – просмотр структуры объектов, поступающих по конвейеру

```
Windows PowerShell
PS C:\Users\andr> Get-Process | Get-Member

TypeName: System.Diagnostics.Process

Name      MemberType Definition
-----
Handles   AliasProperty Handles = Handlecount
Name      AliasProperty Name = ProcessName
NPM       AliasProperty NPM = NonpagedSystemMemorySize
PM        AliasProperty PM = PagedMemorySize64
SI        AliasProperty SI = SessionId
VM        AliasProperty VM = VirtualMemorySize64
WS        AliasProperty WS = WorkingSet64
Disposed  Event System.EventHandler Disposed(S
ErrorDataReceived Event System.Diagnostics.DataReceiv
Exited    Event System.EventHandler Exited(Sys
OutputDataReceived Event System.Diagnostics.DataReceiv
BeginErrorReadLine Method void BeginErrorReadLine()
BeginOutputReadLine Method void BeginOutputReadLine()
CancelErrorRead Method void CancelErrorRead()
CancelOutputRead Method void CancelOutputRead()
Close     Method void Close()
CloseMainWindow Method bool CloseMainWindow()
CreateObjRef Method System.Runtime.Remoting.ObjRef
Dispose   Method void Dispose(), void IDisposab
Equals    Method bool Equals(System.Object obj)
GetHashCode Method int GetHashCode()
GetLifetimeService Method System.Object GetLifetimeServi
GetType   Method type GetType()
InitializeLifetimeService Method System.Object InitializeLifeti
Kill      Method void Kill()
Refresh   Method void Refresh()
Start     Method bool Start()
ToString  Method string ToString()
WaitForExit Method bool WaitForExit(int millisec
WaitForInputIdle Method bool WaitForInputIdle(int mill
__NounName NoteProperty string __NounName=Process
BasePriority Property int BasePriority {get;}
Container Property System.ComponentModel.IContainer
EnableRaisingEvents Property bool EnableRaisingEvents {get;
ExitCode Property int ExitCode {get;}
ExitTime Property datetime ExitTime {get;}
Handle Property System.IntPtr Handle {get;}
HandleCount Property int HandleCount {get;}
HasExited Property bool HasExited {get;}
Id Property int Id {get;}
MachineName Property string MachineName {get;}

```



Обработка объектов в конвейере



Стандартные операции на объектами в конвейере:

- Фильтрация
- Сортировка
- Выделение объектов и свойств
- Произвольные действия
- Группировка
- Измерение характеристик

Фильтрация и сортировка объектов

Командлет **Where-Object**

```
Get-Service | Where-Object {$_.Status -eq "Stopped"}  
Get-Process | Where-Object {$_.Id -gt 1000}
```

Командлет **Sort-Object**

```
Get-Process | Sort-Object cpu -Descending
```

Объединение фильтрации и сортировки:

```
Get-Process | Where-Object {$_.Id -gt 1000} | Sort-Object  
cpu -Descending
```

Получим упорядоченный по количеству затраченного процессорного времени список процессов, идентификатор которых больше 1000.

Выделение объектов и свойств

Командлет **Select-Object**

- Выделение нескольких первых/последних объектов
`Get-Process | Sort-Object WS | Select-Object -Last 5`
- Выделение в итоговых объектах нужных свойств
`Get-Process | Select-Object ProcessName, Id`
- Добавление к объектам новых свойств
`Get-Process | Select-Object ProcessName,
@{Name="StartMin"; Expression = {$_.StartTime.Minute}}`

Выполнение произвольных действий

Командлет **ForEach-Object**

```
$TotalLength=0  
Get-ChildItem | ForEach-Object {$TotalLength+=$_.length}
```

Группировка объектов

Командлет **Group-Object**

Группировка проходящих по конвейеру объектов по значению определенных свойств

```
Get-Process | Group-Object Company
```

Поле Name – название группы, Count – количество элементов в группе, Group – коллекция элементов, входящих в группу

Измерение характеристик объектов

Командлет **Measure-Object**

Применение к свойствам элементов, проходящих по конвейеру, функций агрегирования

```
dir | Measure-Object -Property Length -Minimum -Maximum  
-Average -Sum
```

Функции в PowerShell

Особенности:

- Вызываются как команды (аргументы указываются через пробел без дополнительных круглых скобок и выделения символьных строк кавычками).
- Каждое вычисляемое в функции выражение помещает свой результат в выходной поток.

```
function subtract ($from, $count) {$from-$count}
```

Вызов функций

```
subtract 10 2
```

```
subtract -from 10 -count 2
```

```
subtract -count 3 -from 5
```


Функции в PowerShell

Особенности:

- Вызываются как команды (аргументы указываются через пробел без дополнительных круглых скобок и выделения символьных строк кавычками).
- Каждое вычисляемое в функции выражение помещает свой результат в выходной поток.

```
function subtract ($from, $count) {$from-$count}
```
- Если функция внутри конвейера, то в переменной `$input` хранится входящий поток объектов.

Вызов функций

```
subtract 10 2
subtract -from 10 -count 2
function sum {
    $n=0
    foreach ($i in $input) { $n+=$I }
    $n
}
1..10 | sum
```

Фильтры в PowerShell

Фильтр – это функции особого вида, которые находясь внутри конвейера, запускаются для каждого входящего элемента.

Переменная `$_` соответствует текущему элементу конвейера, проходящему через фильтр.

```
filter Double {$_*2}
1..4 | Double
```

`ForEach-Object` – безымянный фильтр

Функции как командлеты

```
function имя_функции ( параметры ) {  
    begin {  
        блок_кода_инициализация }  
    process {  
        блок_кода_обработка_элемента }  
    end {  
        блок_кода_завершение }  
}
```

ForEach-Object – безымянный фильтр

Работа с внешними объектами

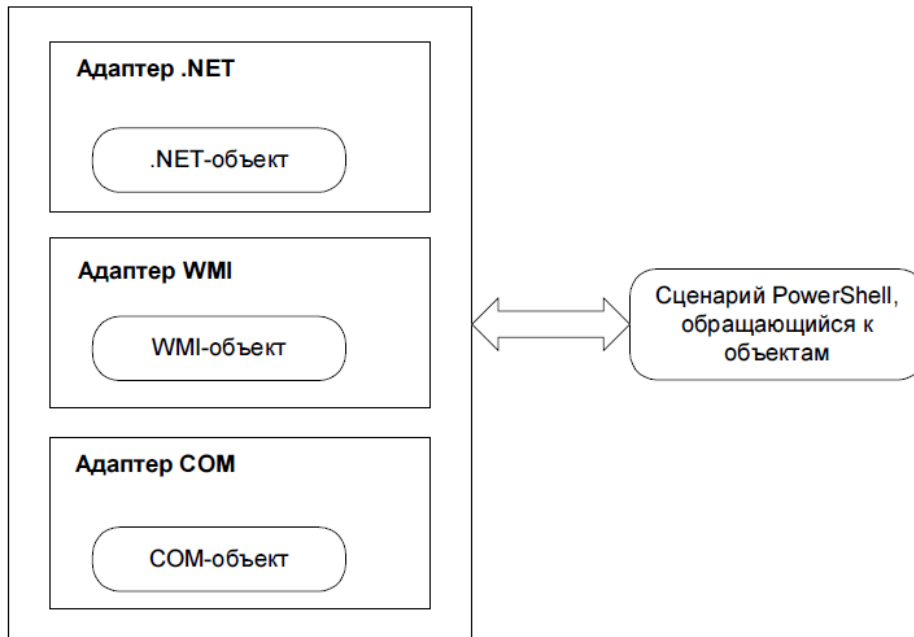
COM-объекты

```
$Shell = New-Object -ComObject WScript.Shell  
$Shell | Get-Member  
(New-Object -Com Sapi.SpVoice).Speak("Hello World")
```

WMI

```
Get-WmiObject -List  
Get-WmiObject Win32_Service
```

Команды PowerShell работают с внешними объектами не напрямую, а через систему адаптации типов и объект PSObject.



Ассоциативные массивы (хэш-таблицы)

Структуры для хранения коллекции ключей и их значений, связанных попарно.

\$имя_массива = @{ключ1 = элемент1; ключ2 = элемент2; ...}

```
PS C:\Users\andrv> $user=@{Фамилия="Попов"; Имя="Андрей"; Телефон="55-55-55"}
PS C:\Users\andrv> $user

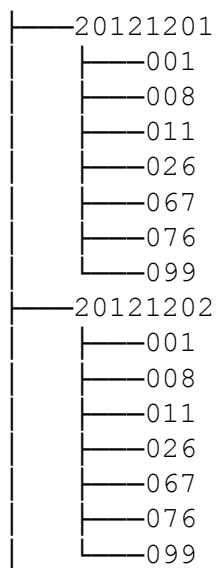
Name                Value
----                -
Фамилия             Попов
Имя                 Андрей
Телефон             55-55-55

PS C:\Users\andrv> $user.Фамилия
Попов
PS C:\Users\andrv> $user["Фамилия"]
Попов
PS C:\Users\andrv>
```

Эффективность: PowerShell vs cmd.exe

Несколько филиалов банка ежедневно формируют операционные дневники в виде файлов `od_*.htm`, которые записываются в папки `ГГГГММДД\NNN`, где `ГГГГММДД` – текущая дата, `NNN` – номер филиала.

Нужно подсчитать количество опердневников для каждого филиала в течение месяца.



Эффективность: PowerShell vs cmd.exe

Несколько филиалов банка ежедневно формируют операционные дневники в виде файлов `od_*.htm`, которые записываются в папки `ГГГГММДД\NNN`, где `ГГГГММДД` – текущая дата, `NNN` – номер филиала.

Нужно подсчитать количество опердневников для каждого филиала в течение месяца.

Батник

1. Формируем файл с путями к папкам, в которых есть дневники. Сортируем этот файл по номеру филиала.

```
@echo off
del temp.txt
FOR /D %%f IN (d:\odb\201612*) DO (
    dir /s/b %%f\od_*.htm >> temp.txt)
sort /+17 temp.txt /o odb.txt
del temp.txt
```

2. Открываем этот файл с разделителями в Excel, выделяем столбец с номерами филиалов, вычисляем промежуточные итоги с подсчетом количества элементов в группе.

20121201
001
008
011
026
067
076
099
20121202
001
008
011
026
067
076
099

Эффективность: PowerShell vs cmd.exe

Несколько филиалов банка ежедневно формируют операционные дневники в виде файлов `od_*.htm`, которые записываются в папки `ГГГГММДД\NNN`, где `ГГГГММДД` – текущая дата, `NNN` – номер филиала.

Нужно подсчитать количество опердневников для каждого филиала в течение месяца.

Батник

1. Формируем файл с путями к папкам, в которых есть дневники. Сортируем этот файл по номеру филиала.

```
@echo off
del temp.txt
FOR /D %%f IN (d:\odb\201612*) DO (
    dir /s/b %%f\od_*.htm >> temp.txt)
sort /+17 temp.txt /o odb.txt
del temp.txt
```

2. Открываем этот файл с разделителями в Excel, выделяем столбец с номерами филиалов, вычисляем промежуточные итоги с подсчетом количества элементов в группе.

PowerShell

```
Get-ChildItem -Path d:\odb\201612*\*\od_*.htm | group-object -property {$_.Directory.Name} -noelement
```


Эффективность: PowerShell vs WSH

В текстовый файл записываются имена серверов, на которых возникли ошибки. Нужно во всем файле подсчитать количество записей для каждого сервера.

Эффективность: PowerShell vs WSH

В текстовый файл записываются имена серверов, на которых возникли ошибки. Нужно во всем файле подсчитать количество записей для каждого сервера.

VBScript

```
set fso = CreateObject("Scripting.FileSystemObject")
dim errcount() ' объявление массива с динамическим изменением размера
max_names = 20 ' начальное резервирование места для 20 имен
redim errcount(2, max_names)
n_names = 0 ' пока имен в списке нет
set strm = fso.OpenTextFile("servers.txt")
do while not strm.AtEndOfStream ' сканирование всех записей в журнале
  servername = strm.ReadLine() ' получение имени сервера из записи журнала
  ' с увеличением его счетчика ошибок
  for i = 1 to n_names ' имя уже есть в списке?
    if errcount(1,i) = servername then ' да, тогда просто
      errcount(2,i) = errcount(2,i)+1 ' увеличение его счетчика ошибок
      exit for
    end if
  next
  if i > n_names then ' не найдено, тогда добавить это имя
    if n_names = max_names then ' список полон, увеличит его размер
      max_names = max_names+20
      redim preserve errcount(2,max_names)
    end if
    n_names = n_names+1 ' добавление нового имени к списку
    errcount(1, n_names) = servername
    errcount(2, n_names) = 1 ' установка счетчика ошибок в 1
  end if
loop
strm.Close

' а теперь вывод списка счетчиков
wscript.echo "Сервер" & vbTab & "Количество ошибок "
for i = 1 to n_names
  wscript.echo errcount(1,i) & vbTab & cstr(errcount(2,i))
next
```

Эффективность: PowerShell vs WSH

В текстовый файл записываются имена серверов, на которых возникли ошибки. Нужно во всем файле подсчитать количество записей для каждого сервера.

VBScript

```
set fso = CreateObject("Scripting.FileSystemObject")
dim errcount() ' объявление массива с динамическим изменением размера
max_names = 20 ' начальное резервирование места для 20 имен
redim errcount(2, max_names)
n_names = 0 ' пока имен в списке нет
set strm = fso.OpenTextFile("servers.txt")
do while not strm.AtEndOfStream ' сканирование всех записей в журнале
  servername = strm.ReadLine() ' получение имени сервера из записи журнала
  ' с увеличением его счетчика ошибок
  for i = 1 to n_names ' имя уже есть в списке?
    if errcount(1,i) = servername then ' да, тогда просто
      errcount(2,i) = errcount(2,i)+1 ' увеличение его счетчика ошибок
    exit for
  end if
next
if i > n_names then ' не найдено, тогда добавить это имя
  if n_names = max_names then ' список полон, увеличит его размер
    max_names = max_names+20
    redim preserve errcount(2,max_names)
  end if
  n_names = n_names+1 ' добавление нового имени к списку
  errcount(1, n_names) = servername
  errcount(2, n_names) = 1 ' установка счетчика ошибок в 1
end if
loop
strm.Close

' а теперь вывод списка счетчиков
wscript.echo "Сервер" & vbTab & "Количество ошибок "
for i = 1 to n_names
  wscript.echo errcount(1,i) & vbTab & cstr(errcount(2,i))
next
```

PowerShell

```
$errcount = @{} # Создание пустой хэш-таблицы
# получение строк из файла и подсчет
get-content '.\servers.txt' | foreach {$errcount[$_]++}
$errcount
```

Выводы

Достоинства Windows PowerShell как языка программирования:

- Упрощает работу с внешними объектами (простое создание экземпляров объектов, возможность получить список свойств и методов этих объектов).
- Во многих случаях конвейеры объектов позволяют уменьшить объем кода, сделать его более понятным и выразительным.