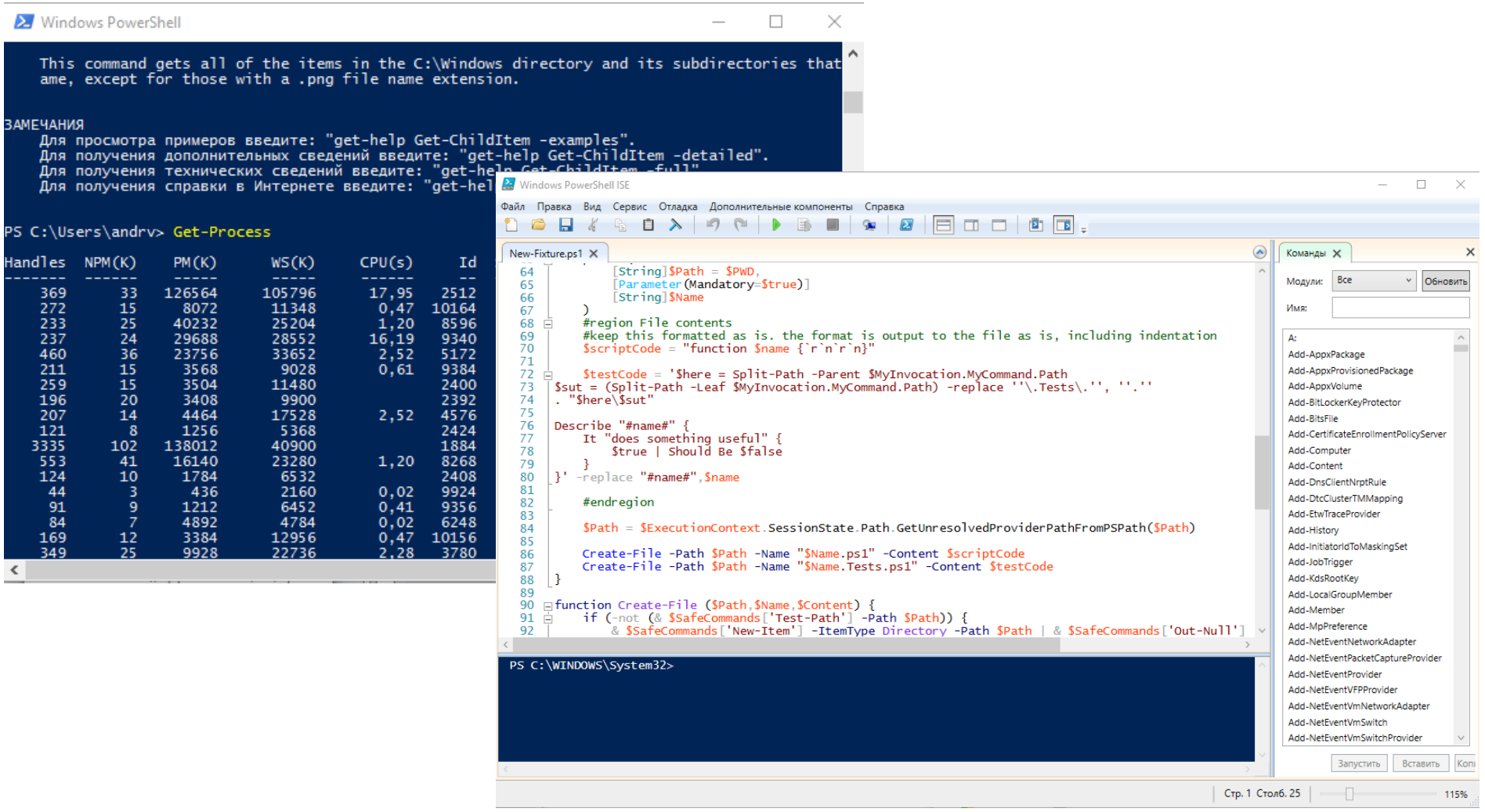
Windows PowerShell – уникальная командная оболочка и среда для создания сценариев Windows



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

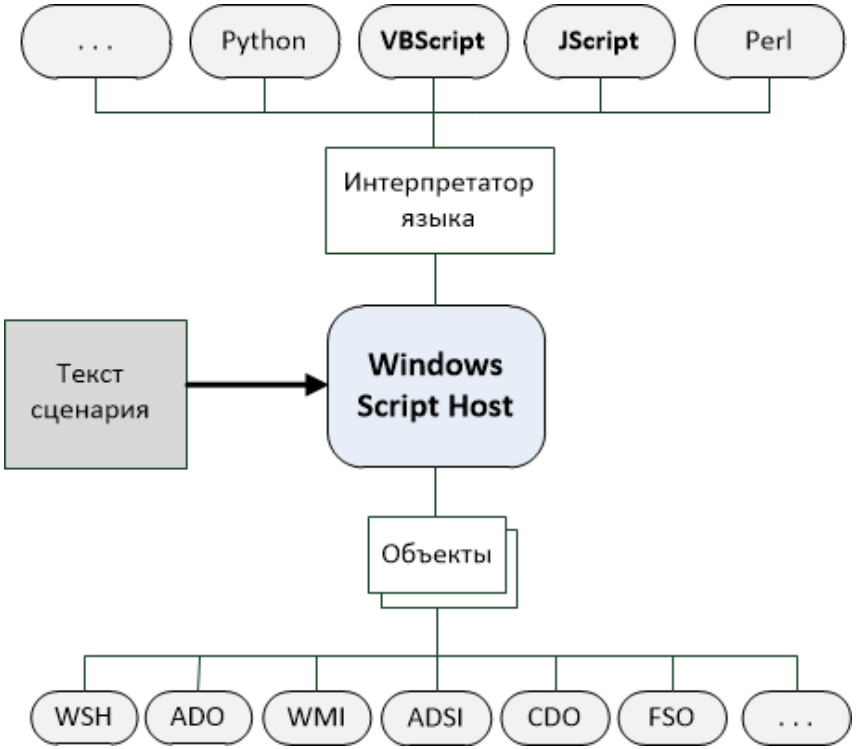
Командный язык для поддержки программных конвейеров

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.

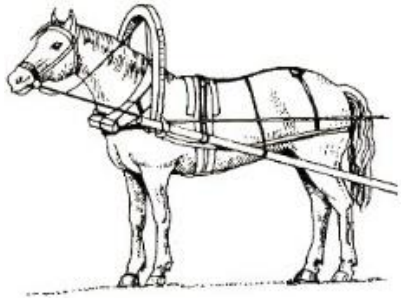


Объекты имеют структуру, их намного легче использовать в программах, чем текстовые данные.

Предыстория

Что было в Windows до PowerShell:

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



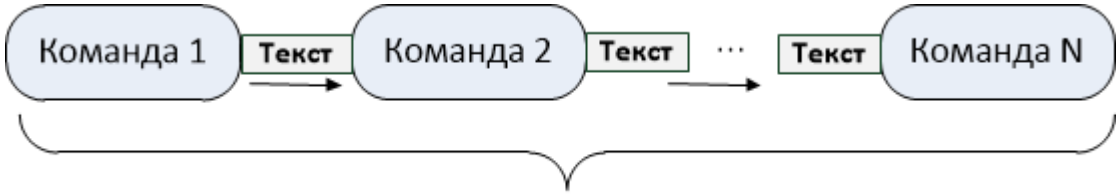
Стиль Unix:
+ конвейеры
- трудно работать с объектами

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

Стиль Windows:
+ работа с объектами
- нет конвейеров

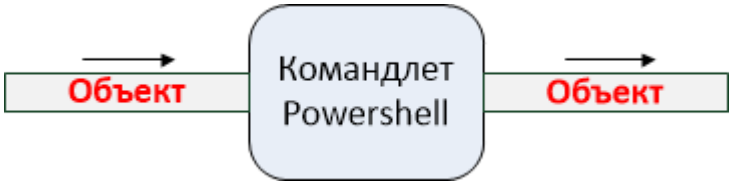
PowerShell: гибрид Unix & Windows

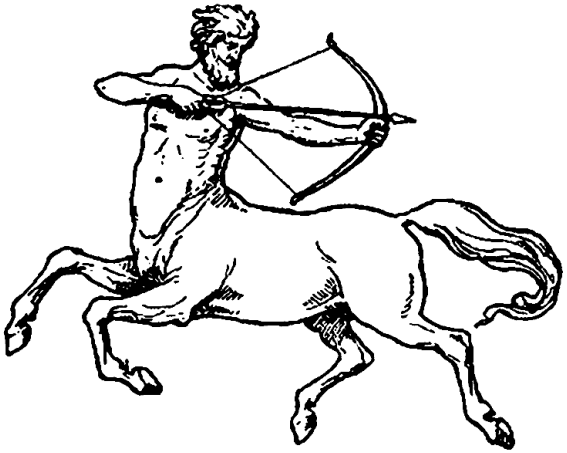
Bash, cmd.exe, …



Решение задачи

Windows Powershell





Решение задачи

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:

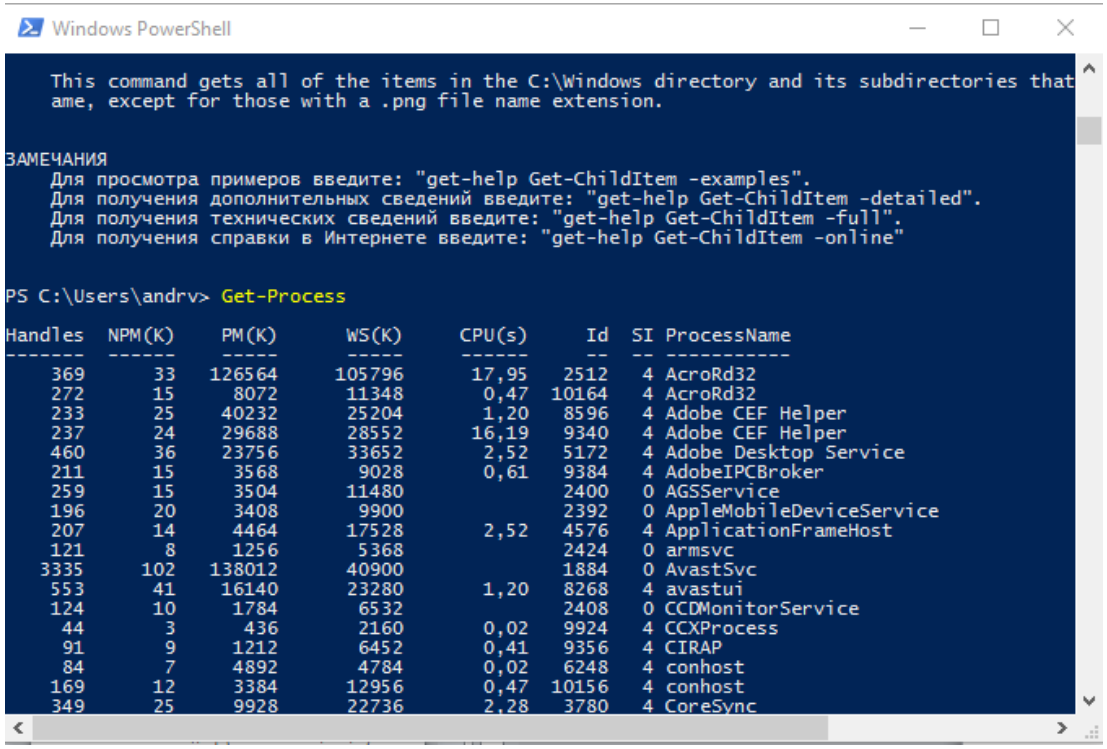
Bash, Ksh – конвейеризация

AS400/VMS – стандартные названия команд

TCL, WSH – встраиваемость и поддержка нескольких языков

PERL, Python – выразительность и стиль

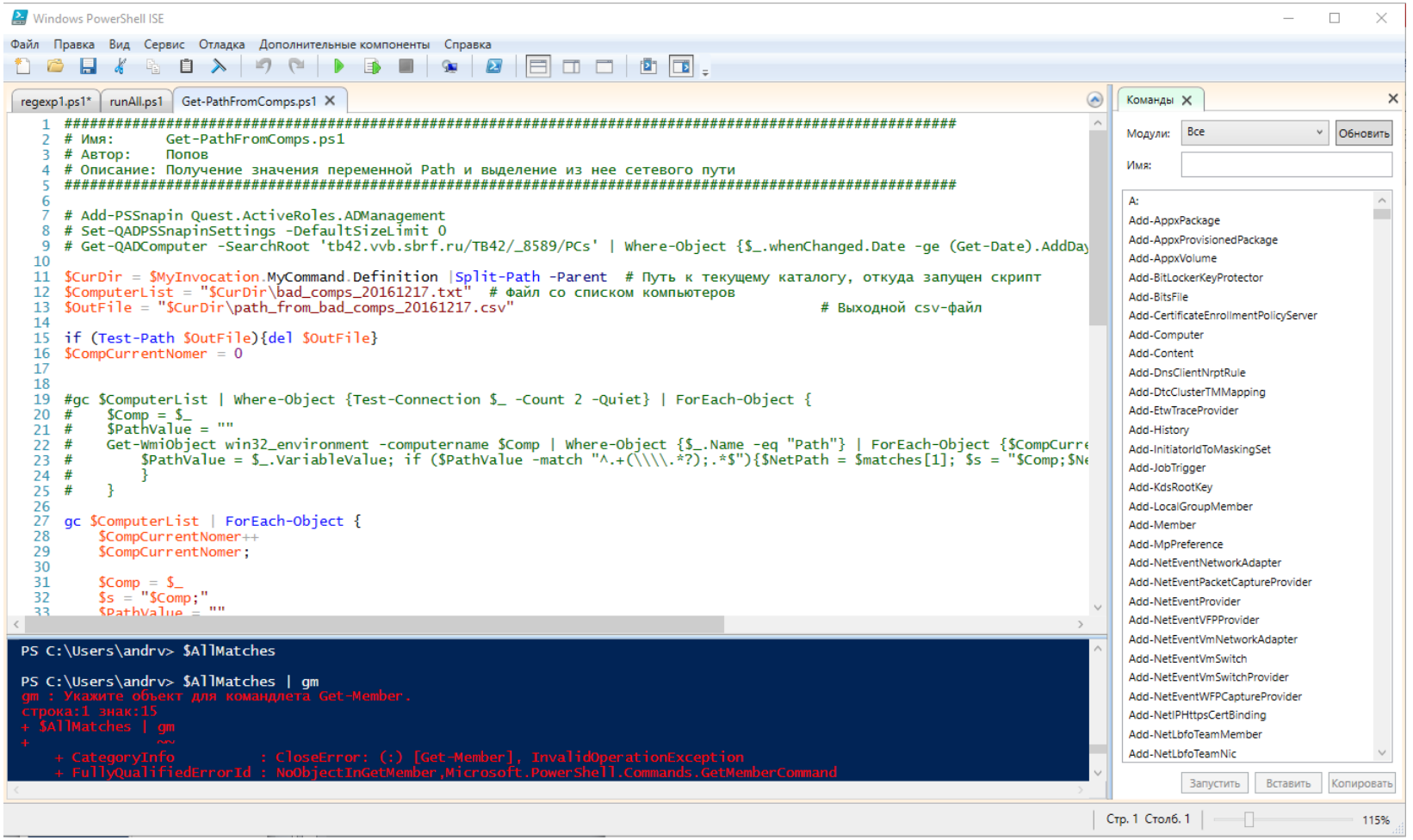
Знакомство с оболочкой



Powershell.exe

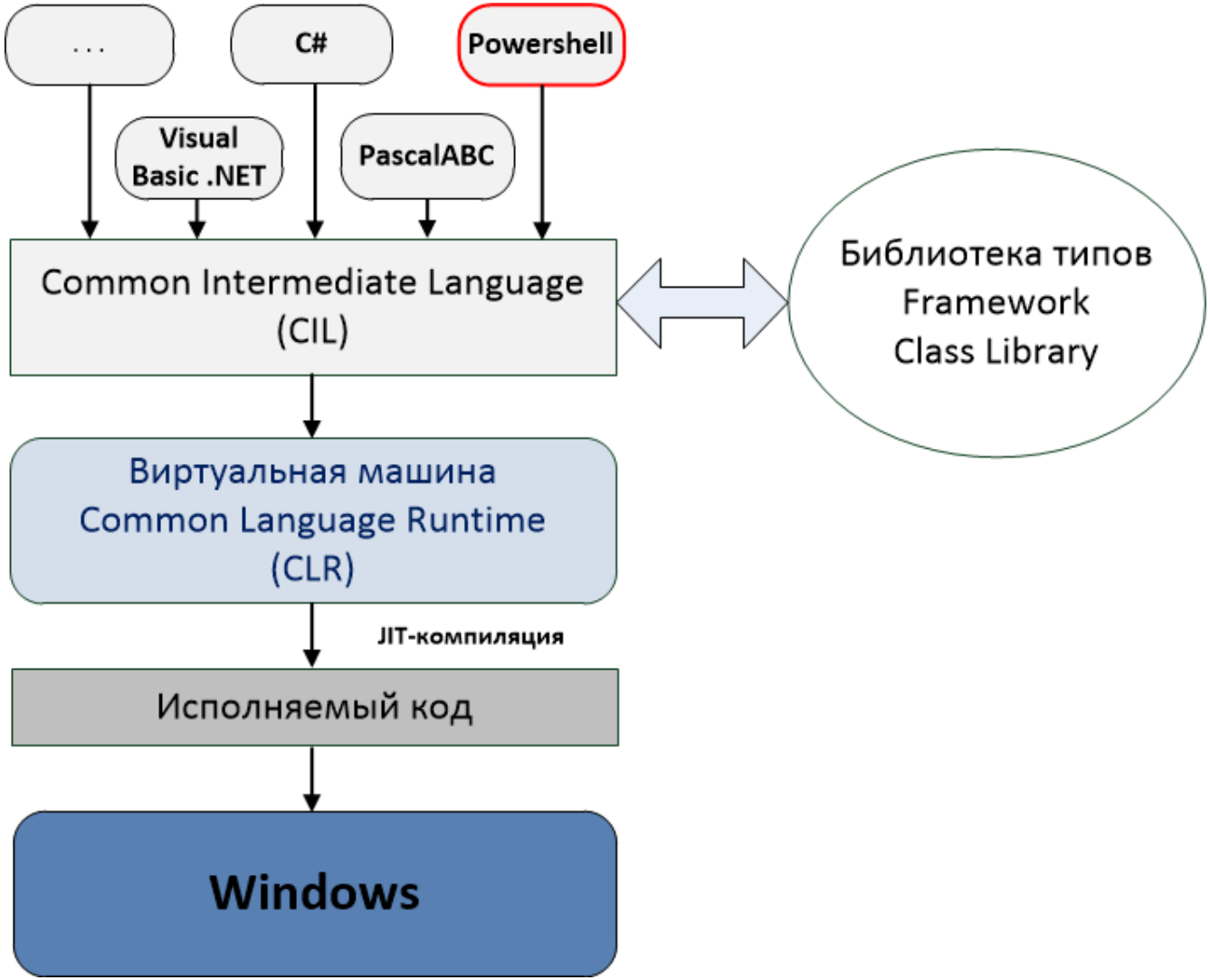
- Псевдонимы для команд
- Встроенная помощь
- Автозавершение ввода
- Стандарт имен команд
- Перенаправление ввода/вывода

Интегрированная среда сценариев



Powershell_ISE.exe

Фундамент PowerShell: .NET Framework



Все в PowerShell – объекты

Объекты .NET Framework – самодокументируемые

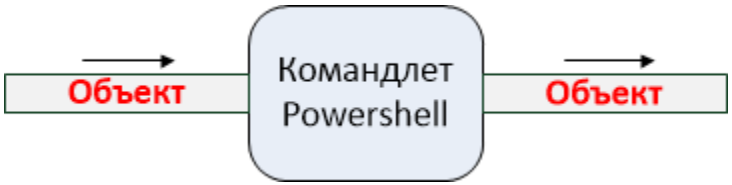
Командлет **Get-Member** – просмотр структуры объектов, поступающих по конвейеру

Windows PowerShell

```
PS C:\Users\andrv> 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.DataReceive
Exited                     Event          System.EventHandler Exited(Sys
OutputDataReceived         Event          System.Diagnostics.DataReceive
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 millisecc
WaitForInputIdle           Method         bool WaitForInputIdle(int mill
__NounName                 NoteProperty   string __NounName=Process
BasePriority               Property       int BasePriority {get;}
Container                  Property       System.ComponentModel.IContain
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 групу

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

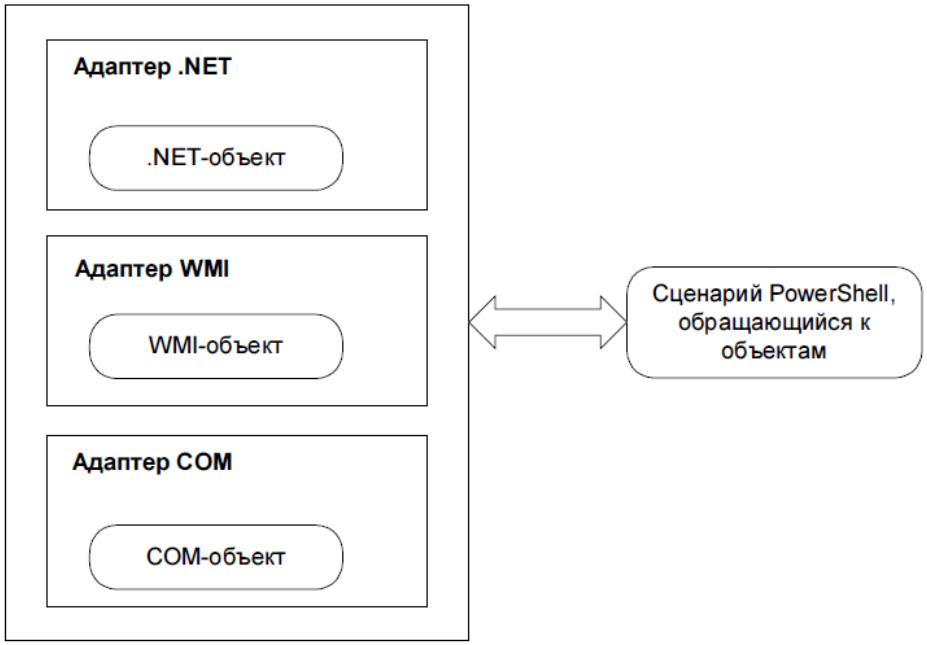
COM-объекты

```
$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*; ...}

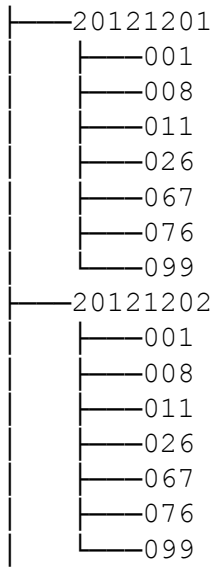
```
PS C:\Users\andrv> $user=@{Фамилия="Попов"; Имя="Андрей"; Телефон="55-55-55"}
PS C:\Users\andrv> $user

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>
```

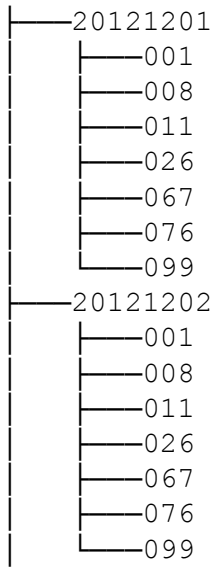
Эффективность: PowerShell vs cmd.exe

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.

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.

```
├───20121201
│   ├───001
│   ├───008
│   ├───011
│   ├───026
│   ├───067
│   ├───076
│   └───099
├───20121202
│   ├───001
│   ├───008
│   ├───011
│   ├───026
│   ├───067
│   ├───076
│   └───099
```

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.