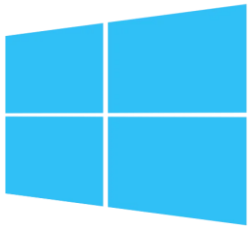
Лекция 10

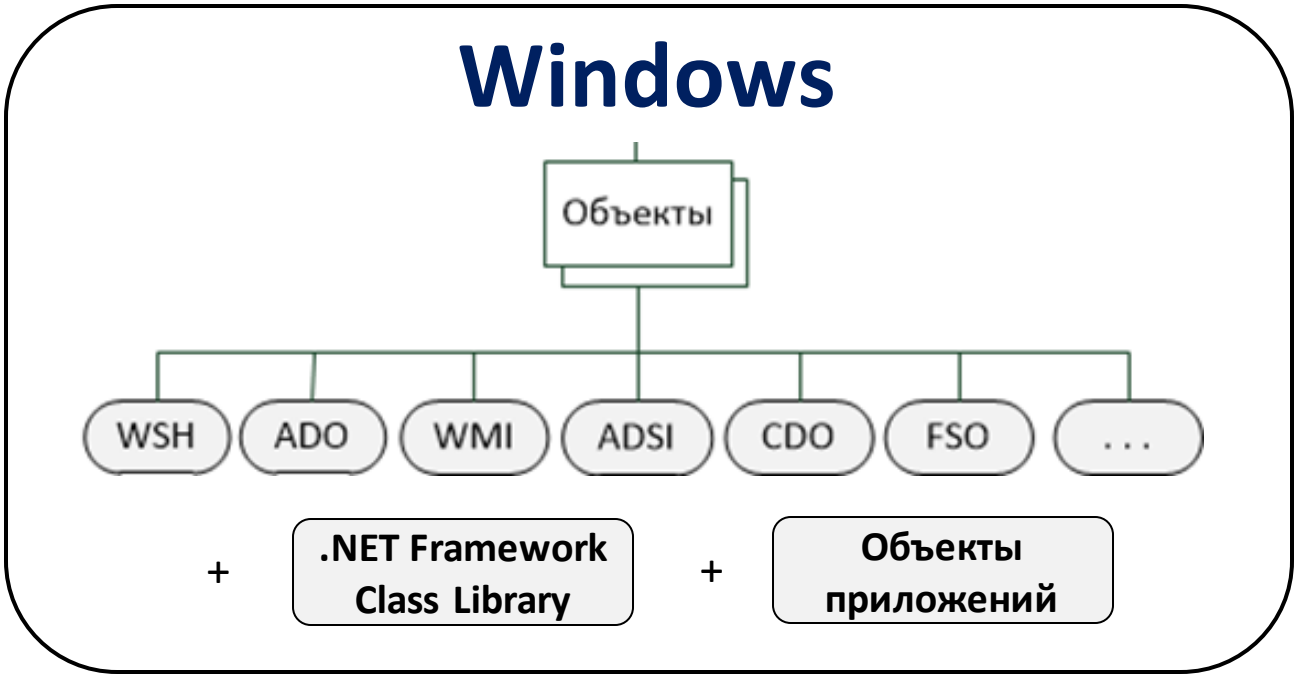
Работа с объектами с помощью PowerShell

Объекты внутри операционки



Windows = API-ориентированная ОС

Управляемые элементы сгруппированы во внутренние объекты.



Регулярно повторяем одни и те же действия?
Можно и нужно автоматизировать!

Объекты внутри Windows - как достать и использовать?

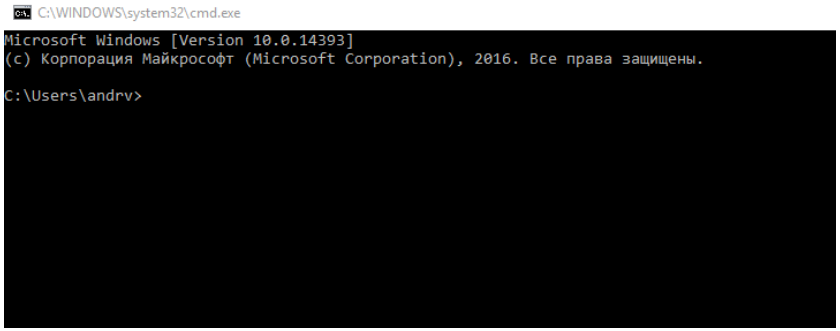
Программы-сценарии (скрипты)

Пожелания к языку для сценариев для операционки

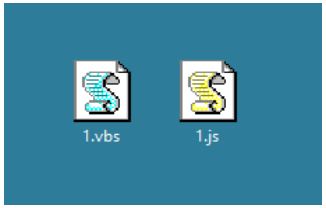
- Доступ к максимально широкому кругу функций и возможностей, поддерживаемому ОС.
- Максимальная простота в изучении языка и написании программ-сценариев.
- В качестве операторов можно использовать исполняемые программы.
- Поддержка как пакетного, так и интерактивного режима работы в командной строке.

Три инструмента автоматизации в Windows

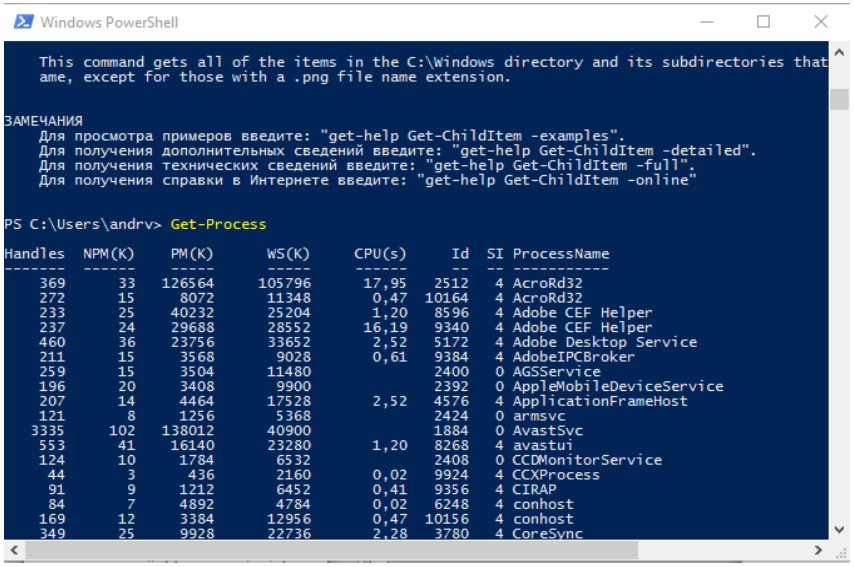
1. Командная строка cmd.exe и пакетные bat-файлы

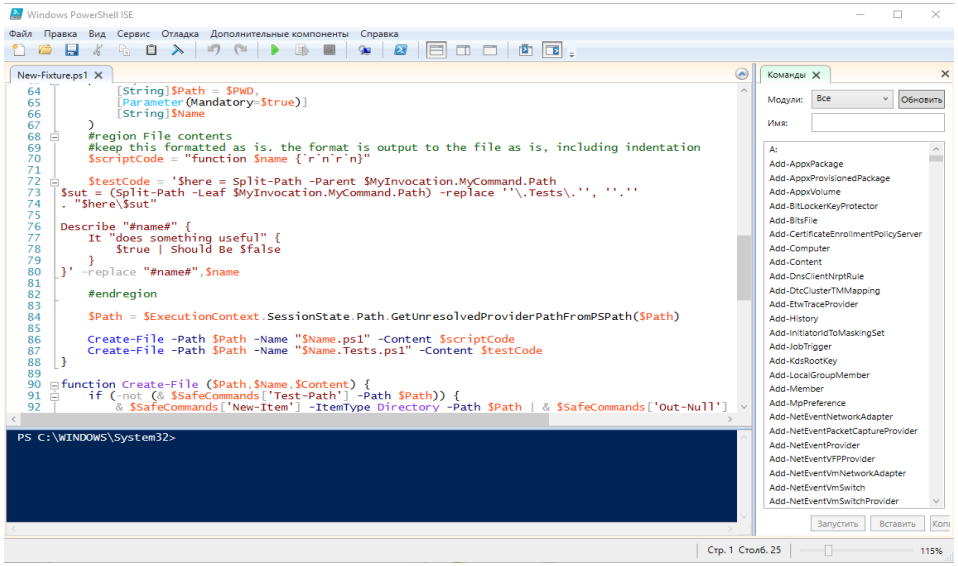


2. Windows Script Host (VBScript и JScript)



3. Оболочка и среда выполнения сценариев PowerShell





Эволюция сценариев Windows

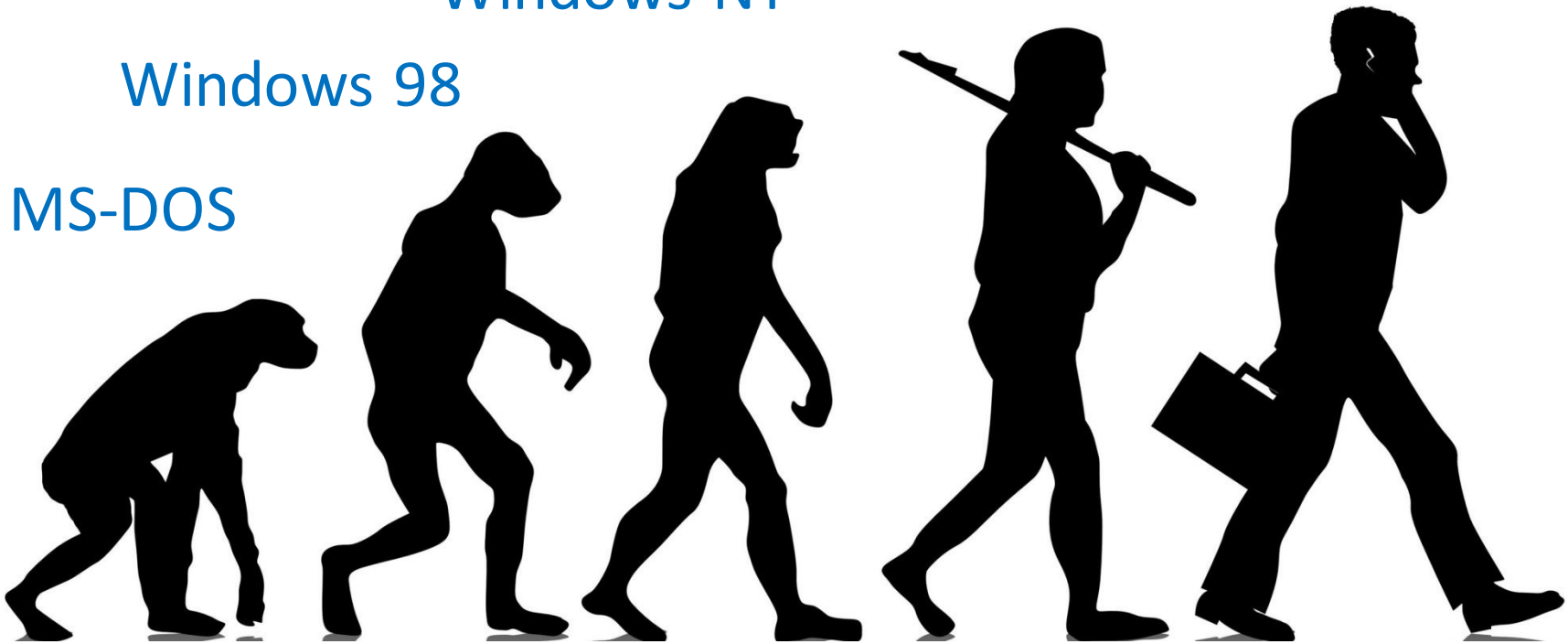
Windows 10

Windows XP

Windows NT

Windows 98

MS-DOS



Сценарии Windows Script Host

Командные файлы
command.com/cmd.exe

Сценарии
Windows PowerShell

Скрипты на VBScript/JScript через WSH

Windows Script Host

- Позволяет запускать в контексте безопасности пользователя файлы *.vbs и *.js в консольном (cscript.exe) или графическом (wscript.exe) режиме.
- Обеспечивает доступ из сценариев к внешним объектам.
- Регистрирует в системе объекты, обеспечивающие функциональность для работы с Windows.

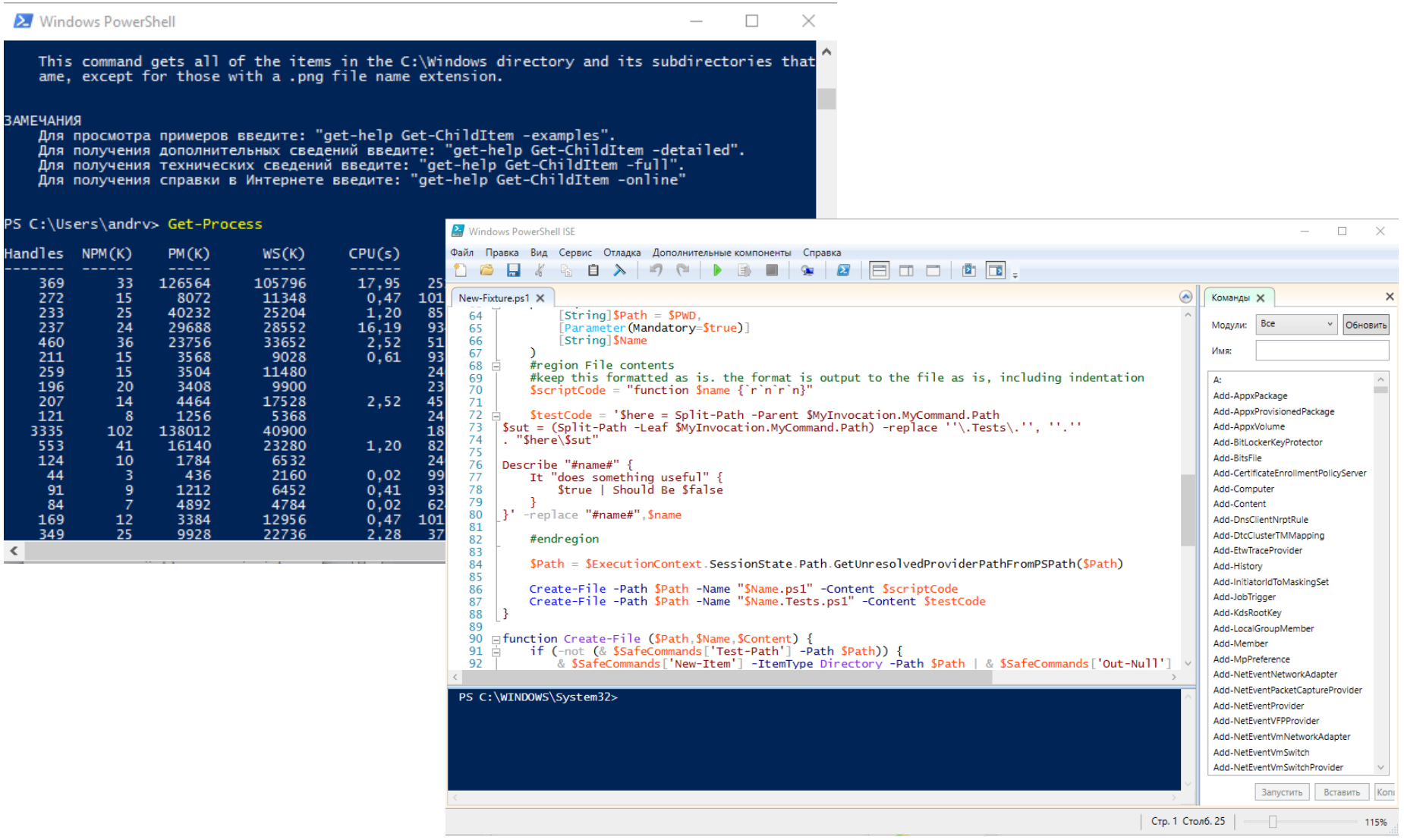
Для полного счастья не хватает…

- Интерактивности
- Выполнения однострочных команд
- Перенаправления ввода/вывода внутри сценариев
- Подключения в скриптах внешних файлов
- Нет простого и быстрого способа узнать, какие свойства и методы содержит объект
- Работать с WMI слишком сложно

Объекты внутри Windows - как достать?

Microsoft PowerShell

PowerShell – уникальная командная оболочка и среда выполнения сценариев операционной системы



Командный язык (оболочка) vs Язык сценариев

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

Одна команда = одна функция
find, cat, sed, grep, awk, …

Командный язык для поддержки программных конвейеров

Командный язык (оболочка) vs Язык сценариев

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

Одна команда = одна функция
find, cat, sed, grep, awk, …

Командный язык для поддержки программных конвейеров

```
find /tmp -type f -name '*' -mtime +7 -print0 | xargs -0 rm -f
```

Командный язык (оболочка) vs Язык сценариев

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

Одна команда = одна функция
find, cat, sed, grep, awk, …

Командный язык для поддержки программных конвейеров

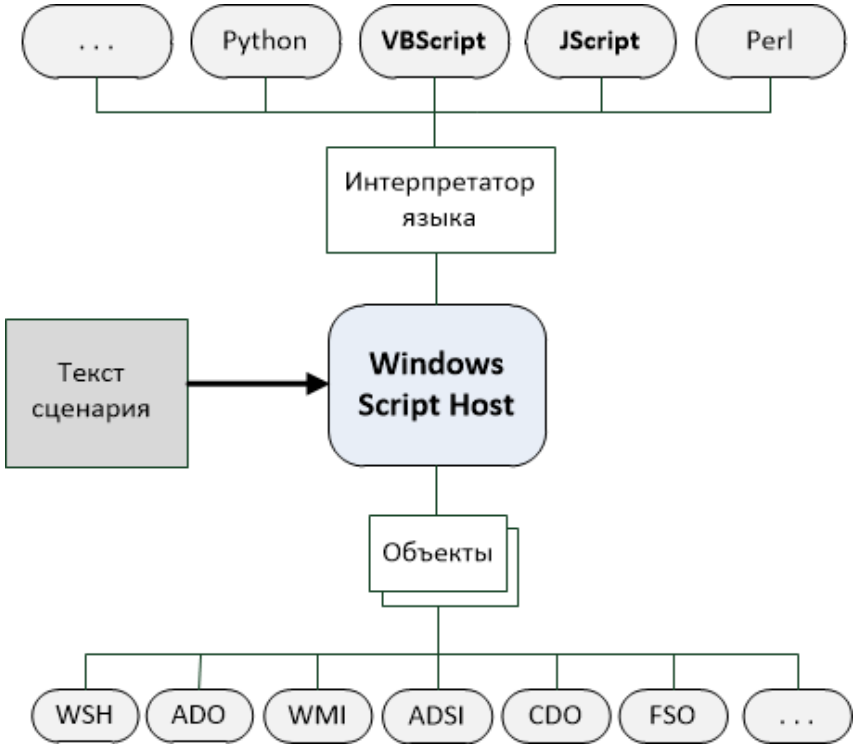
```
find /tmp -type f -name '*' -mtime +7 -print0 | xargs -0 rm -f
```

Программный конвейер – пример модульного проектирования (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

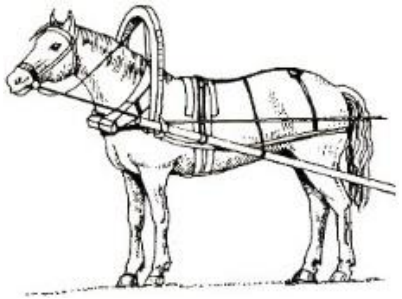
Язык сценариев (scripting language) – программирование в традиционном стиле (без конвейеров), но с использованием внешних объектов.



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

Что было в Windows до PowerShell:

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



Стиль Unix:

+ конвейеры

- нельзя работать с объектами

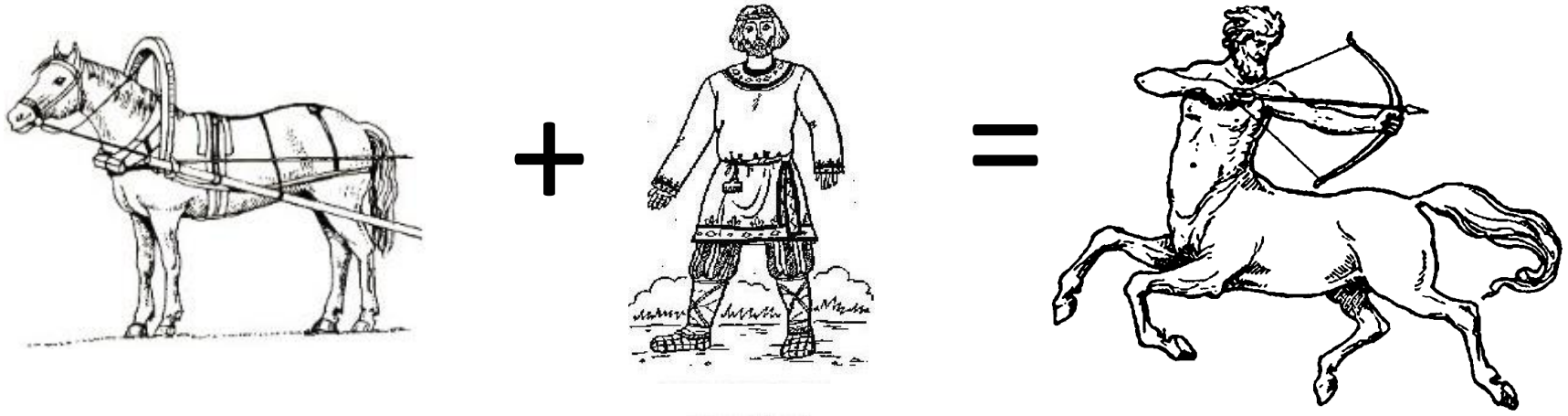
2. **Сценарии** на языках VBScript, JScript (Python, Perl, …) = Windows Script Host

Стиль Windows:

+ работа с объектами

- нет конвейеров

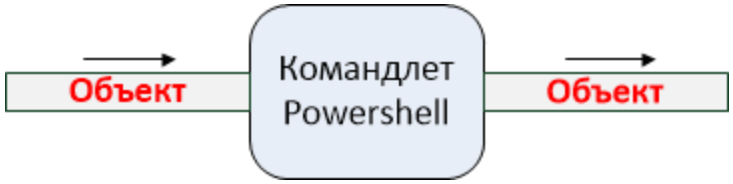
PowerShell: гибрид Unix и Windows



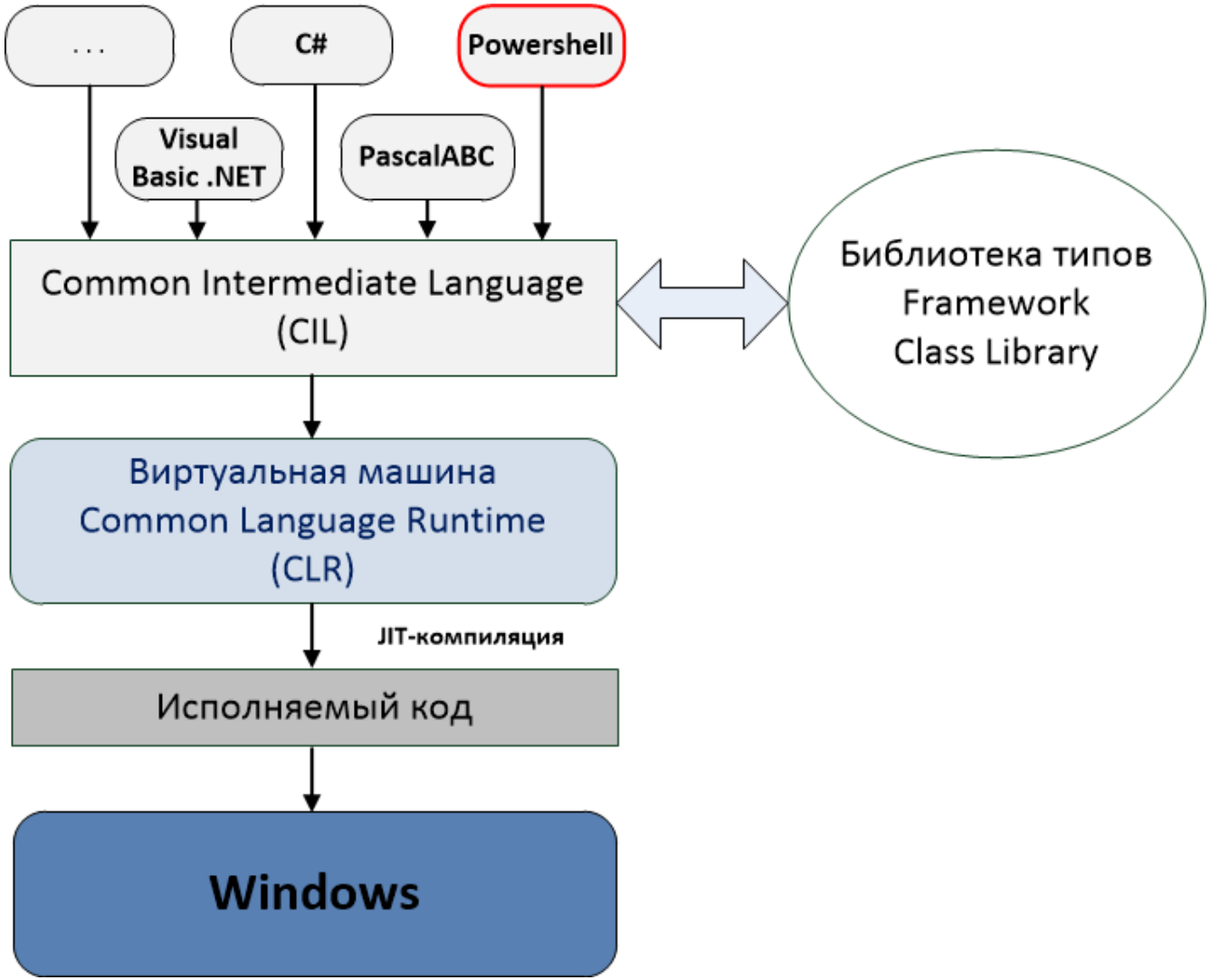
PowerShell: гибрид Unix и Windows

Bash, cmd.exe, …

PowerShell



Фундамент PowerShell - платформа .NET Framework



Все в PowerShell – объекты

Объекты .NET Framework – самодокументируемые

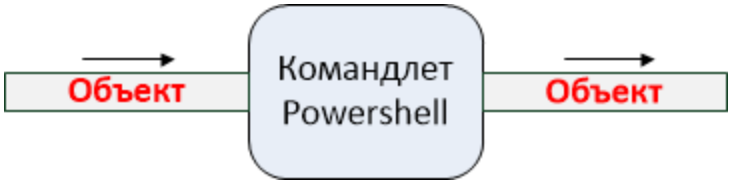
Get-Member – просмотр структуры объектов, поступающих по конвейеру

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;}
```



PowerShell: гибрид Unix & Windows

- 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

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
```

Работа с внешними объектами

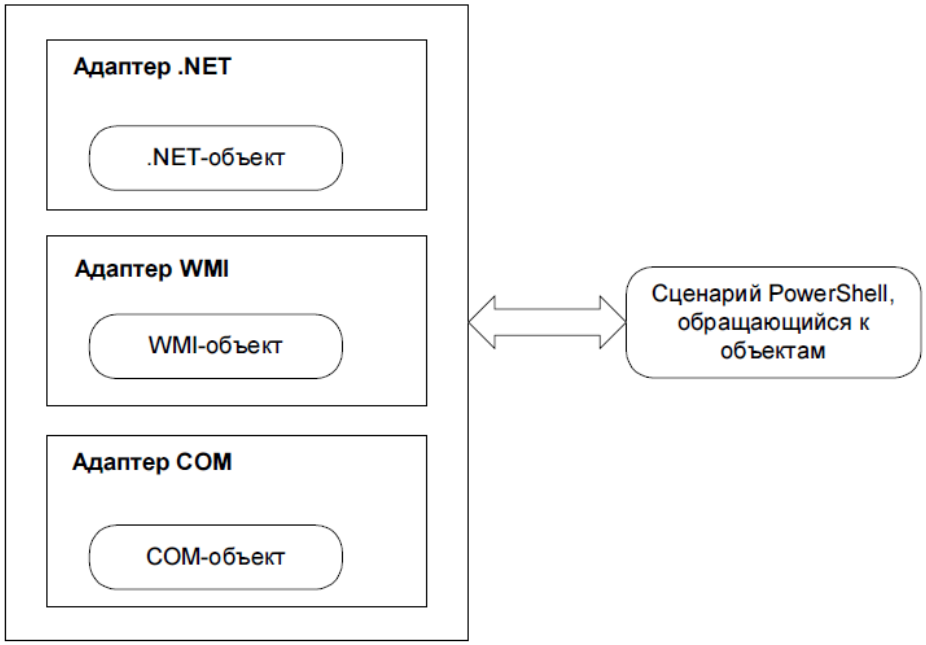
COM-объекты

```
$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.



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
```

PowerShell как язык программирования

- Упрощает работу с внешними объектами (простое создание экземпляров объектов, возможность получить список свойств и методов этих объектов).

- Конвейеры объектов позволяют уменьшить объем кода, сделать его более понятным и выразительным.