Создание и запуск сценариев Powershell

Вопро
с №1

Массивы, операторы и управляющие инструкции

Вопро
с №2

Технология PowerShell Desired State Configuration

Вопро
с №3

Разновидность сценариев PowerShell Workflow

Вопрос №1

Массивы, операторы и управляющие инструкции

Массивы

- Типы массивов:
- Текстовые
- Числовые

- Действие над массивом:
- Создание
- Выборка
- Изменение

Для создания числового массива необходимо объявить переменную, которой присваивается через запятую различные числа (элементы массива)

$a1 = 1,2,3,4,5,6,7,8,9,10

или

$a1 = 1..10

$a1.Length - определить количество элементов в массиве

Для создания текстового массива необходимо объявить переменную, которой присваивается через запятую в кавычках различные слова, буквы и др. (элементы массива)

$a2 = “A”, “B”, “C”, “D”, “E”, “F”, “G”

CAST - объявление массива

$a3 = @(1,2,3,4,5)

$a4 = @(get-process; get-date)

- Выборка элементов из одномерного массива
  - Выборка всего массива
  - Выборка одного элемента массива
  - Выборка нескольких элементов из массива
  - Выборка диапазона значений из массива
  - Выборка объединенных диапазонов значений

Выборка всего массива

Вывод в виде строки

Write-host $a1 Write-host $a2

Вывод в виде столба

$a1 $a2

Выбор одного элемента из массива

$a1[0] - первый элемент массива

$a1[-1] - последний элемент массива

Выбор диапазона значений из массива

$a1[0..2] - 1,2,3 элементы массива

$a1[2..-1] - 3,2,1,последний элементы массива

Выбор нескольких элементов из массива

$a1[1,3,5] - 2,4,6 элементы массива

$a1[0,-1] - первый и последний элементы массива

Выбор объединённых диапазонов значений из массива

$a1[0..2+5..7] - 1,2,3,6,7,8 элементы массива

Изменение значения элемента в массиве

$a1[0] = 5 первому элементу массива изменили значение на 5

Добавление элемента в конец массива

$a1 += 5 добавляем к массиву значение 5 и оно становится последним элементом

Добавление элемента в любое место массива

$a1 = $a1[0..2] + 5 + $a1[3..9] добавляем к массиву значение 5 в качестве 4-го элемента нового массива

Исключение любого элемента из массива

$a1 = $a1[0..2] + $a1[4..10] исключаем в массиве 4-ый элемент нового массива

Удаление массива

Remove-Variable –name a1 удаляем массив с именем a1

Арифметические

Оператор	Описание	Пример	Результат
+	Складывает два значения	2+4 “aaa”+”bbb” 1,2,3+4,5	6 “aaabbb” 1,2,3,4,5
*	Перемножает два значения	2*4 “a”*3 1,2,3*2	8 “aaa” 1,2,3,1,2,3
-	Вычитает одно значение из другого	5-3	2
/	Делит одно значение на другое	6/3 7/4	2 1.75
%	Возвращает остаток при целочисленном делении одно значения на другое	7%4	3

Операторы присваивания

Оператор	Пример	Аналог	Описание
=	$a=2		Присваивает переменной указанное значение
+=	$a+=3	$a=$a+3	Прибавляет указанное значение к текущему значению переменной, затем присваивает полученный результат данной переменной
-=	$a-=3	$a=$a-3	Отнимает указанное значение от текущего значения переменной, затем присваивает полученный результат данной переменной
*=	$a*=2	$a=$a*2	Умножает текущее значение переменной на указанное число, затем присваивает полученный результат данной переменной
/=	$a/=2	$a=$a/2	Делит текущее значение переменной на указанное число, затем присваивает полученный результат данной пере менной
%=	$a%=2	$a=$a%2	Находит остаток от деления текущего значения переменной на указанное число, затем присваивает полученный результат данной переменной

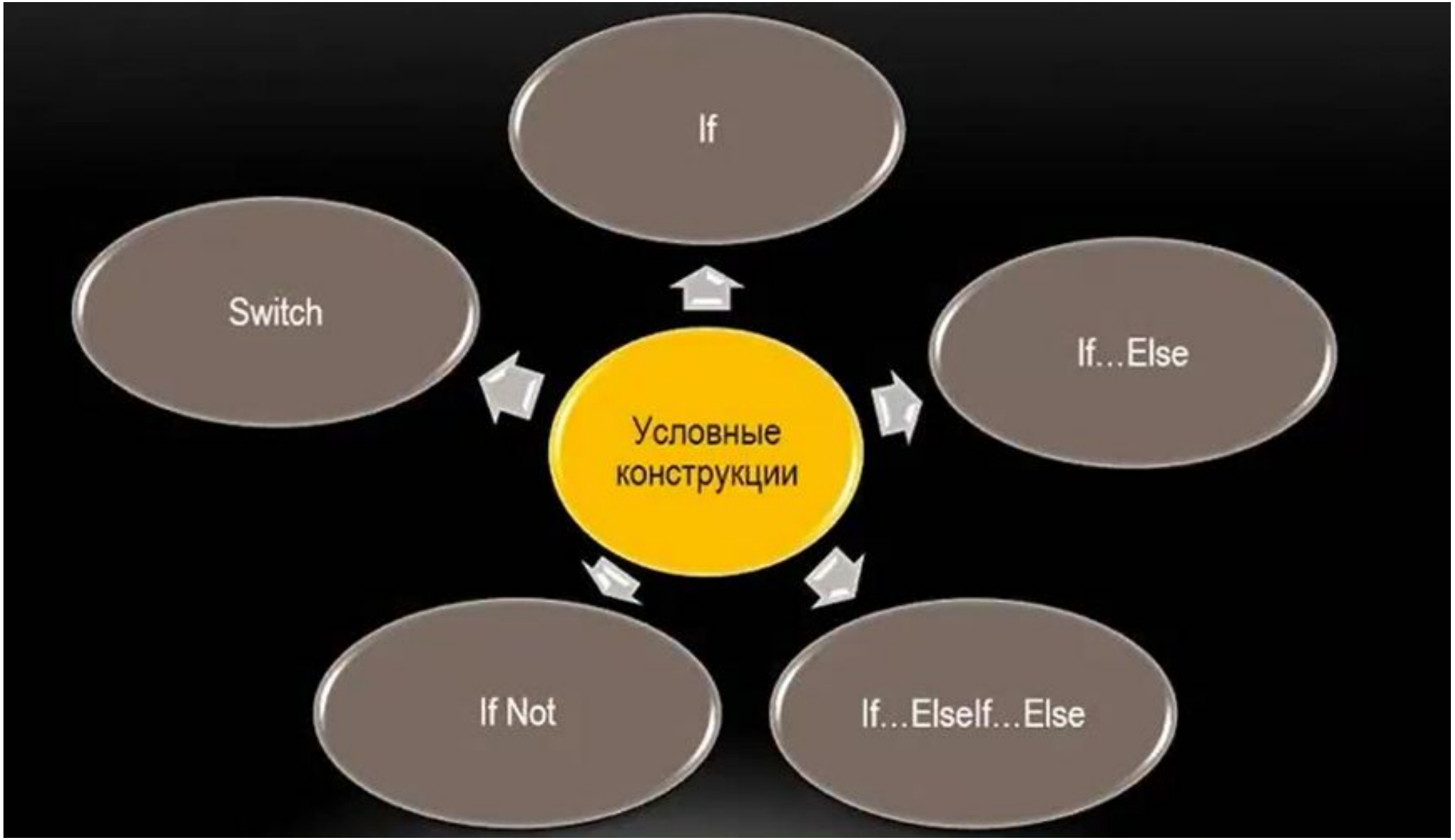
Операторы сравнения

Оператор	Описание	Пример
-eq	Equal / Равно (=)	$var = "619“ $var -eq 123 #$false
-ne	Not equal / Не равно (<>)	$var = "619"$var -ne 123 #$true
-gt	Greater than / Больше (>)	$var = "619"$var -gt 123 #$true
-ge	Greater than or equal / Больше или равно (>=)	$var = "619"$var -ge 123 #$true
-lt	Less than / Меньше (<)	$var = "619"$var -lt 123 #$false
-le	Less than or equal / Меньше или равно (<=)	$var = "619"$var -le 123 #$false
-like	Сравнение с учётом символа подстановки	"Habra" -like "habr*" #true
-notlike	Сравнение с учётом не соответствия символа подстановки	"Habra" -notlike "habr*" #false
-contains	Содержит ли значение слева значение справа	1, 2, 3, 4, 5 -contains 3 #$true
-notcontains	Если значение слева не содержит значение справа, получим истину	1, 2, 3, 4, 5 -notcontains 3 #$false
-match	Использование регулярных выражений для поиска соответствия образцу	$str = "http://habrahabr.ru"$str -match "^http://(\S+)+(.ru)$" #$true
-notmatch	Использование регулярных выражений для поиска несоответствия образцу	$str = "http://habrahabr.ru"$str -notmatch "^http://(\S+)+(.com)$" #true
-replace	Заменяет часть или все значение слева от оператора	"Microhabr" -replace "Micro","Habra" #Habrahabr

Базовый вариант операторов сравнения по умолчанию не учитывает регистр букв. Если оператор начинается с буквы “c” (например –ceq, -cne и т.д.), при сравнении регистр букв будет учитываться.

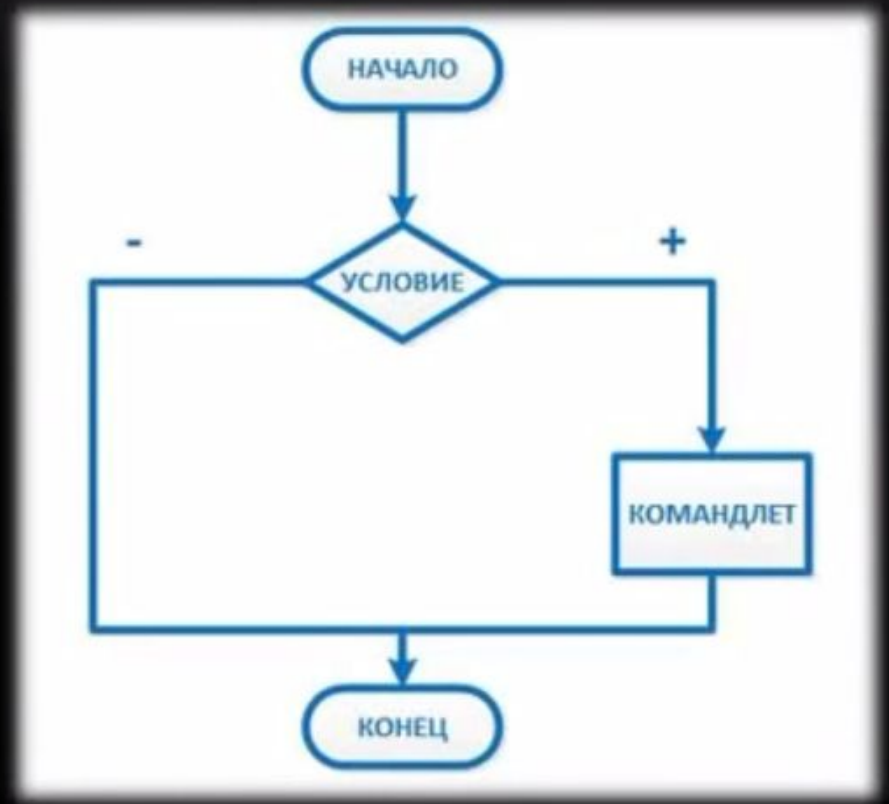
Логические операторы

Оператор	Значение	Пример (возвращается значение True)
-and	логическое И	(10 –eq 10) –and (1 –eq 1)
-or	логическое ИЛИ	(9 –ne 10) –or (3 –eq 4)
-not	логическое НЕ	-not (3 –gt 4)
!	логическое НЕ	! (3 –gt 4)


If
Switch
If…Else
Условные конструкции
If Not
If…ElseIf…Else

КОНСТРУКЦИЯ IF В WINDOWS POWERSHELL 2.0

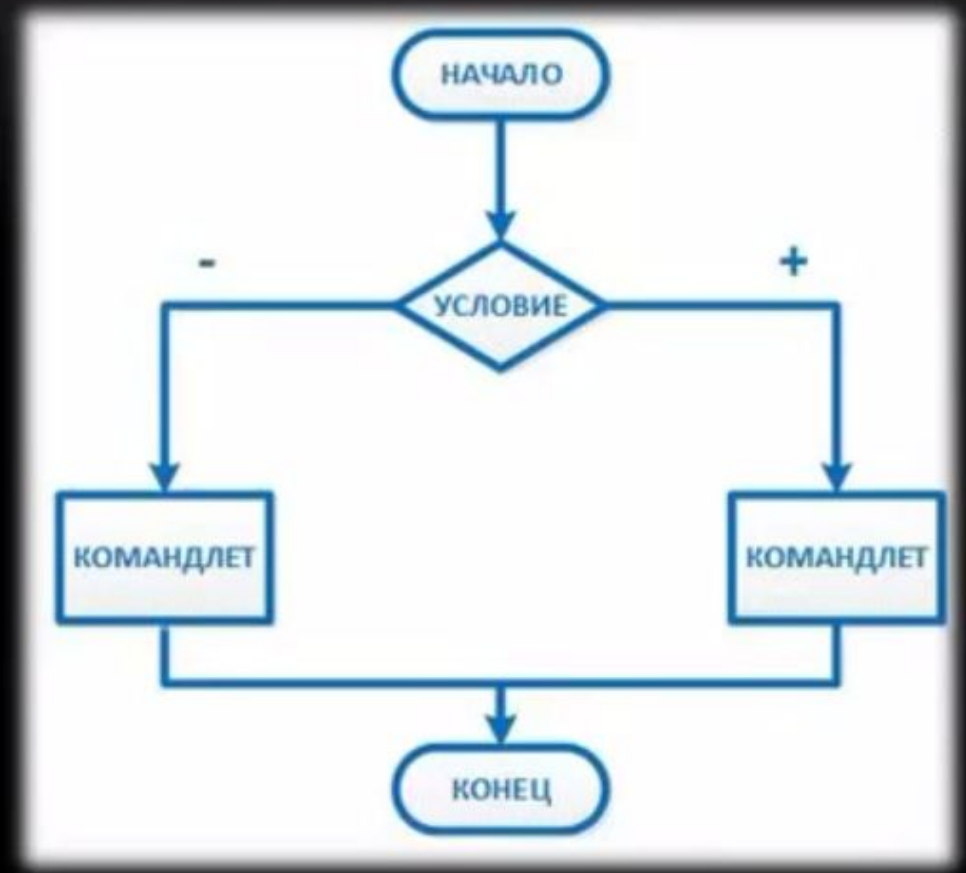
```
If (УСЛОВИЕ) {ДЕЙСТВИЕ}
```



```
 Clear-Host
$a= Read-host ‘Введите
значение переменной $a'
Write-Host '##IF##'
  if ($a -eq 5)
{Write-Host '$a = 5'}
```

КОНСТРУКЦИЯ IF…ELSE В WINDOWS POWERSHELL 2.0

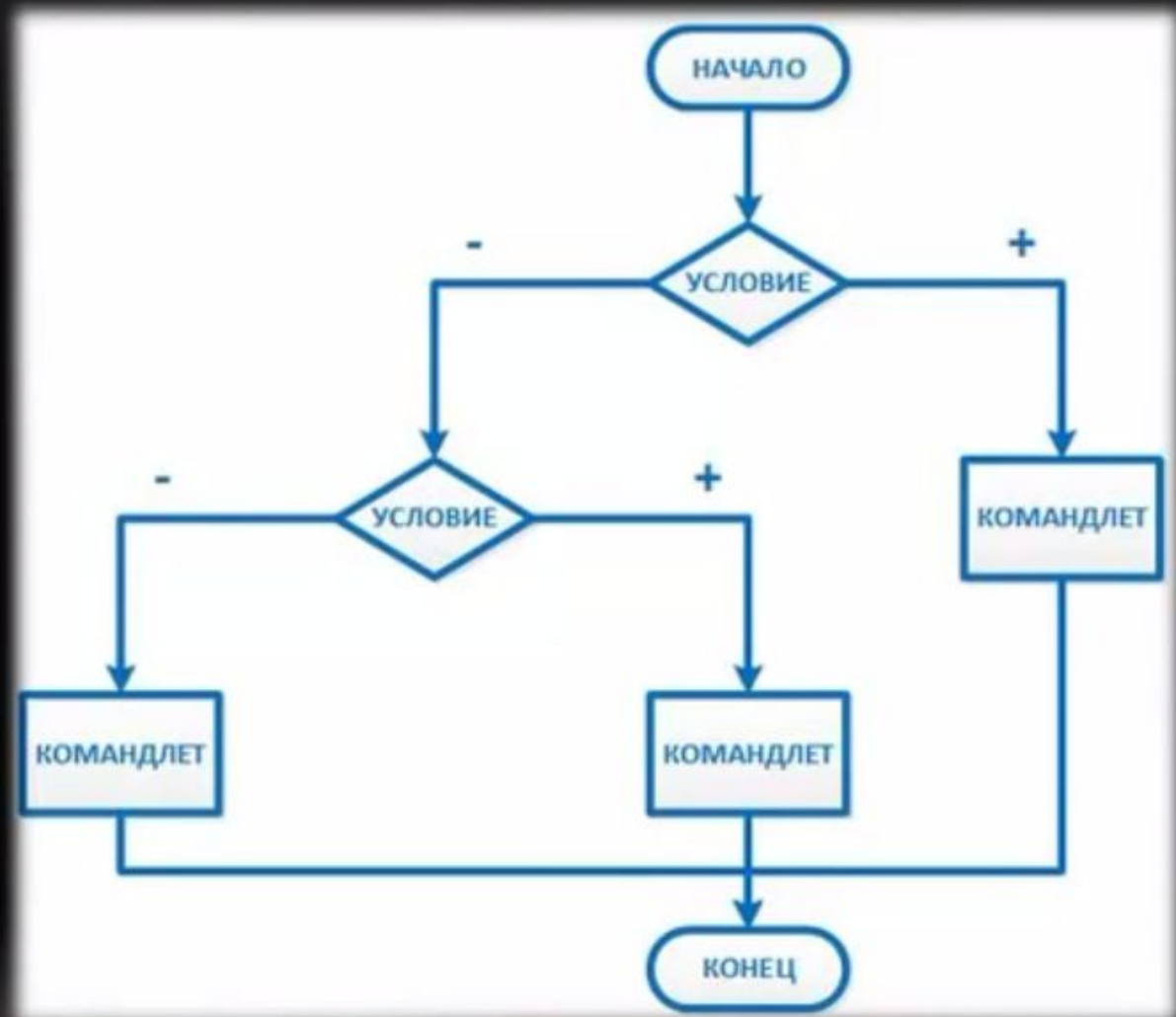
```
If (УСЛОВИЕ) {ДЕЙСТВИЕ} `
 Else {ДЕЙСТВИЕ}
```



```
 Clear-Host
$a= Read-host 'Введите значение
переменной $a'
Write-Host '##IF...ELSE##'
  if ($a -eq 5)
{Write-Host '$a=5'}
  else
{Write-Host '$a <> 5'}
```

КОНСТРУКЦИЯ IF…ELSEIF…ELSE В WINDOWS POWERSHELL 2.0

```
If (УСЛОВИЕ) `
        {ДЕЙСТВИЕ} `
ElseIf (УСЛОВИЕ) `
  {ДЕЙСТВИЕ} `
  Else {ДЕЙСТВИЕ}
```



```
Clear-Host
$a= Read-host ' Введите значение
переменной $a'
Write-Host '##IF...ELSEIF…ELSE##'
  if ($a -eq 5)
{Write-Host '$a = 5'}
   elseif ($a -lt 5)
{Write-Host '$a < 5'}
    else
{Write-Host '$a > 5'}
```

Практическое задание №1

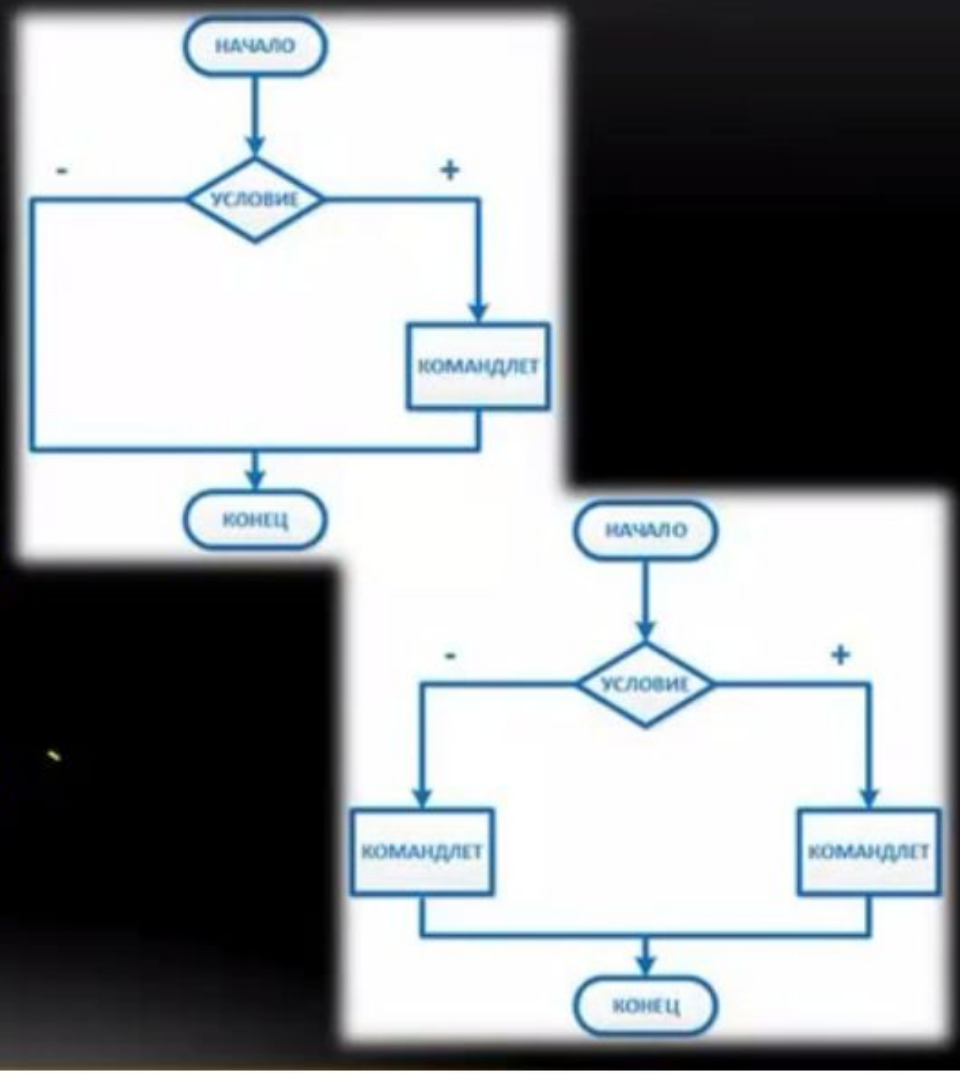
Определите какой сейчас месяц с помощью Powershell и по его номеру определите время года

```
Clear-Host
$zima=1,2,12
$osen=9,10,11
$vesna=3,4,5
$leto=6,7,8
$mesyaz=get-date


if (($mesyaz.Month -eq $zima[0]) -or ($mesyaz.Month -eq $zima[1]) -or
($mesyaz.Month -eq $zima[2]))
{ write-host `Сейчас зима` }
elseif ($osen[0..2] -eq $mesyaz.Month)
  { write-host `Сейчас осень` }
 elseif (($mesyaz.Month -ge $vesna[0]) -and ($mesyaz.Month -le $vesna[2]))
   { write-host `Сейчас весна` }
   elseif (($mesyaz.Month -ge 6) -or ($mesyaz.Month -le 8))
     { write-host `Сейчас лето` }
```

Изменять дату: **Set-Date -Date 1.XX.15**

КОНСТРУКЦИЯ IF NOT В WINDOWS POWERSHELL 2.0

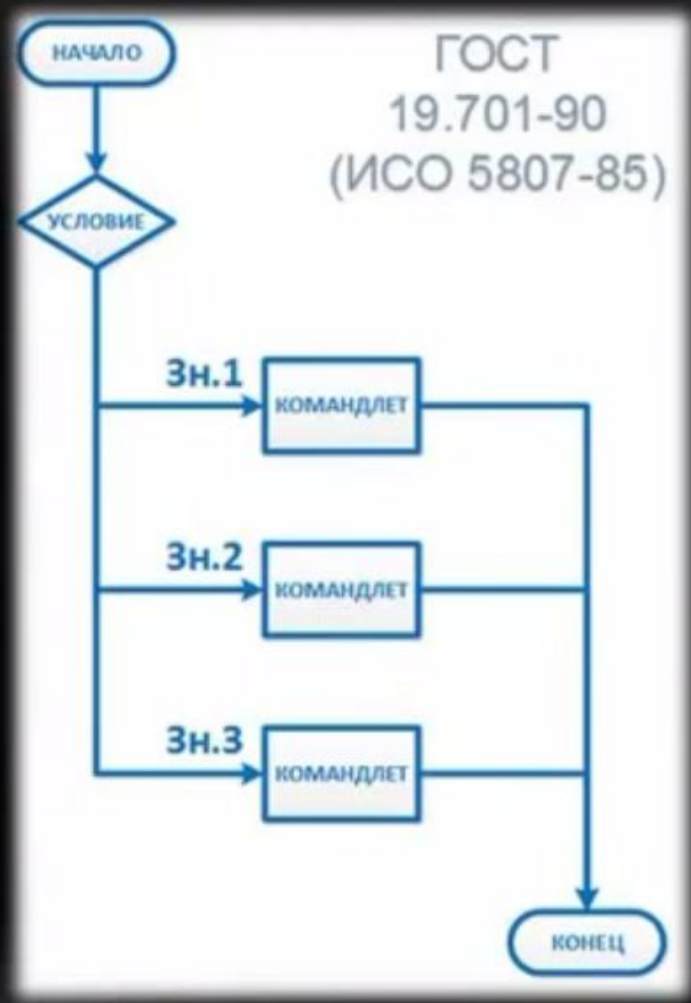
```
If (!(УСЛОВИЕ)) {ДЕЙСТВИЕ}

If (!(УСЛОВИЕ)) {ДЕЙСТВИЕ} `
Else {ДЕЙСТВИЕ}

If (-not(УСЛОВИЕ)) {ДЕЙСТВИЕ}

If (-not(УСЛОВИЕ)) {ДЕЙСТВИЕ} `
Else {ДЕЙСТВИЕ}
```



```
 Clear-Host
$a= Read-host 'Введите значение
переменной $a'
Write-Host '##IF NOT##'
  if (-not($a -eq 5))
{Write-Host '$a не равно 5'}
  else
{Write-Host '$a = 5'}
```

КОНСТРУКЦИЯ SWITCH В WINDOWS POWERSHELL 2.0

```
Switch (УСЛОВИЕ) {
        Значение1 {ДЕЙСТВИЕ}
        Значение2 {ДЕЙСТВИЕ}
        …
        ЗначениеN {ДЕЙСТВИЕ}
}
```

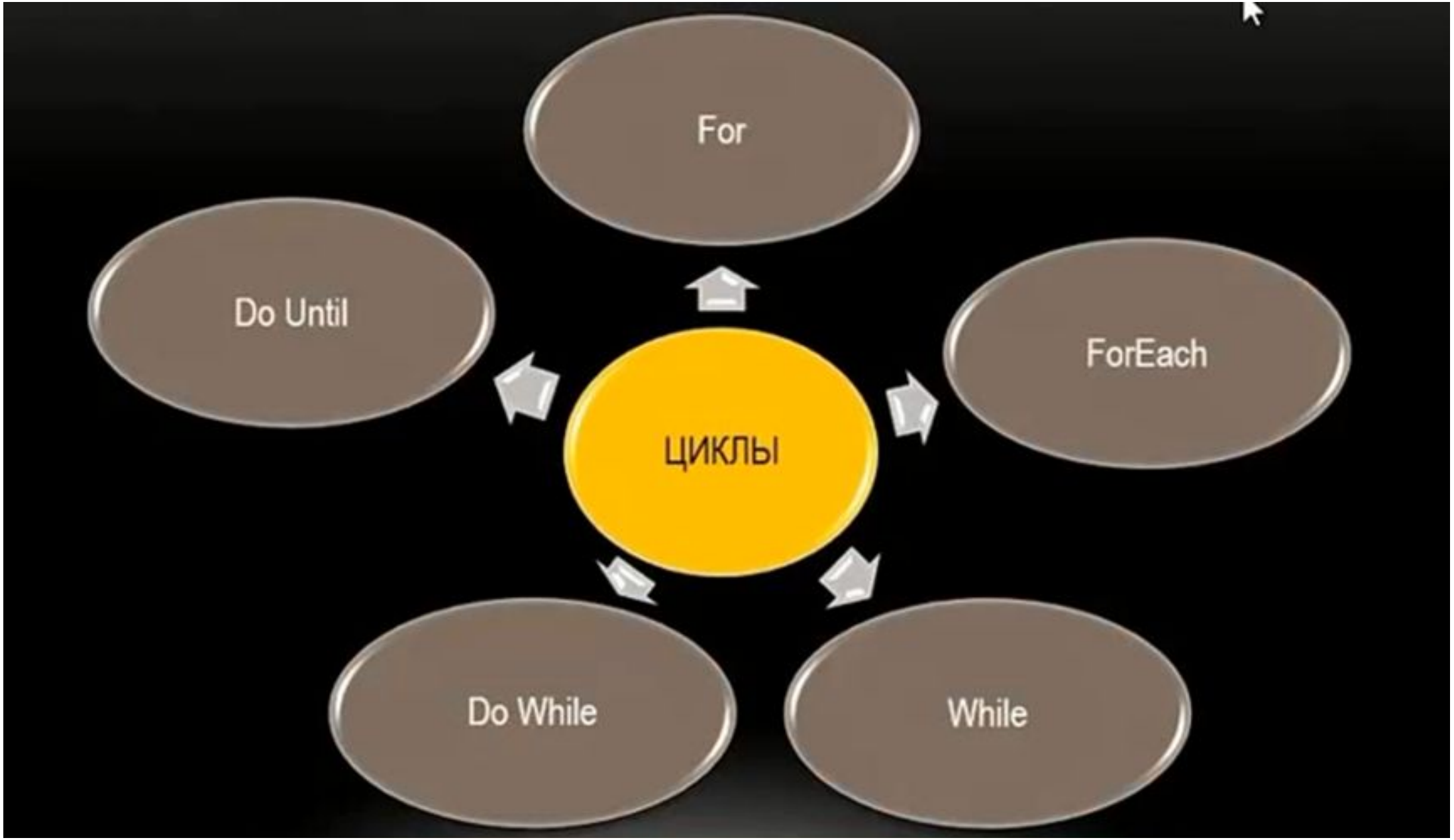


```
Clear-Host
$x = Read-host 'Введите значение переменной $x от
0 до 5'
Write-Host '##SWITCH##'
switch ($x)
{ 0 {Write-Host 'Вы ввели 0'}
  1 {Write-Host 'Вы ввели 1'}
  2 {Write-Host 'Вы ввели 2'}
  3 {Write-Host 'Вы ввели 3'}
  4 {Write-Host 'Вы ввели 4'}
  5 {Write-Host 'Вы ввели 5'}
}
```

Практическое задание №2

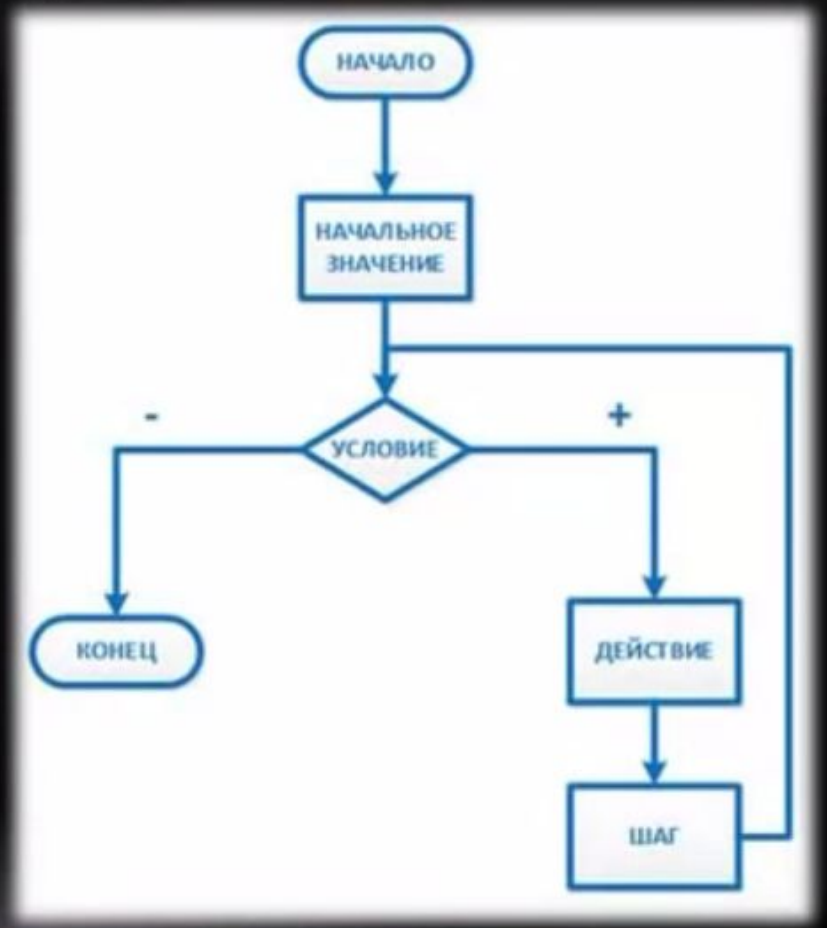
Создайте сценарий с использованием конструкции Switch, который позволяет быстро переключать сетевые настройки операционной системы (например, локальная сеть или интернет). При этом, при включении настроек Интернет должен запускаться браузер IE, а в случае локального доступа браузер должен закрываться. После применения настроек конфигурация TCP/IP должна быть отображена на экране. В случае неправильного ввода данных должно выдаваться сообщение.

```
clear-host

$network = read-host "Введите 1 для выбора локальной сети или 2 для интернета"

switch ($network)
{
    '1'
    {$NetAdapter = Get-NetAdapter -Name Ethernet0;
     $NetAdapter | remove-NetIPAddress;
     $NetAdapter | Remove-NetRoute -DestinationPrefix 0.0.0.0/0;
     $NetAdapter | Set-DnsClientServerAddress -ResetServerAddresses
     $NetAdapter | New-NetIPAddress -IPAddress 10.10.10.20 -PrefixLength 24

     stop-process -Name Iexplore -Force;
     gip;
     Write-Host "Локальная сеть"}

    '2'
   {$NetAdapter = Get-NetAdapter -Name Ethernet0;
    $NetAdapter | remove-NetIPAddress;
    $NetAdapter | Remove-NetRoute -DestinationPrefix 0.0.0.0/0;
    $NetAdapter | Set-DnsClientServerAddress -ResetServerAddresses
    $NetAdapter | New-NetIPAddress -IPAddress 192.168.0.2 -PrefixLength 24 -DefaultGateway 192.168.0.1
    $NetAdapter | Set-DnsClientServerAddress -ServerAddresses 192.168.0.1

    Start-Process "C:\Program Files (x86)\Internet Explorer\Iexplore.exe" -WindowStyle Maximized;
    gip;
    Write-Host “Вы в Интернете" }
    Default {write-host "Не правильный ввод"}
}
```


For
ForEach
While
Do While
Do Until
ЦИКЛЫ

ЦИКЛ FOR
WINDOWS POWERSHELL 2.0

```
for (НАЧАЛЬНОЕ_ЗНАЧЕНИЕ;УСЛОВИЕ;ШАГ)
        {
                ДЕЙСТВИЕ
        }
```

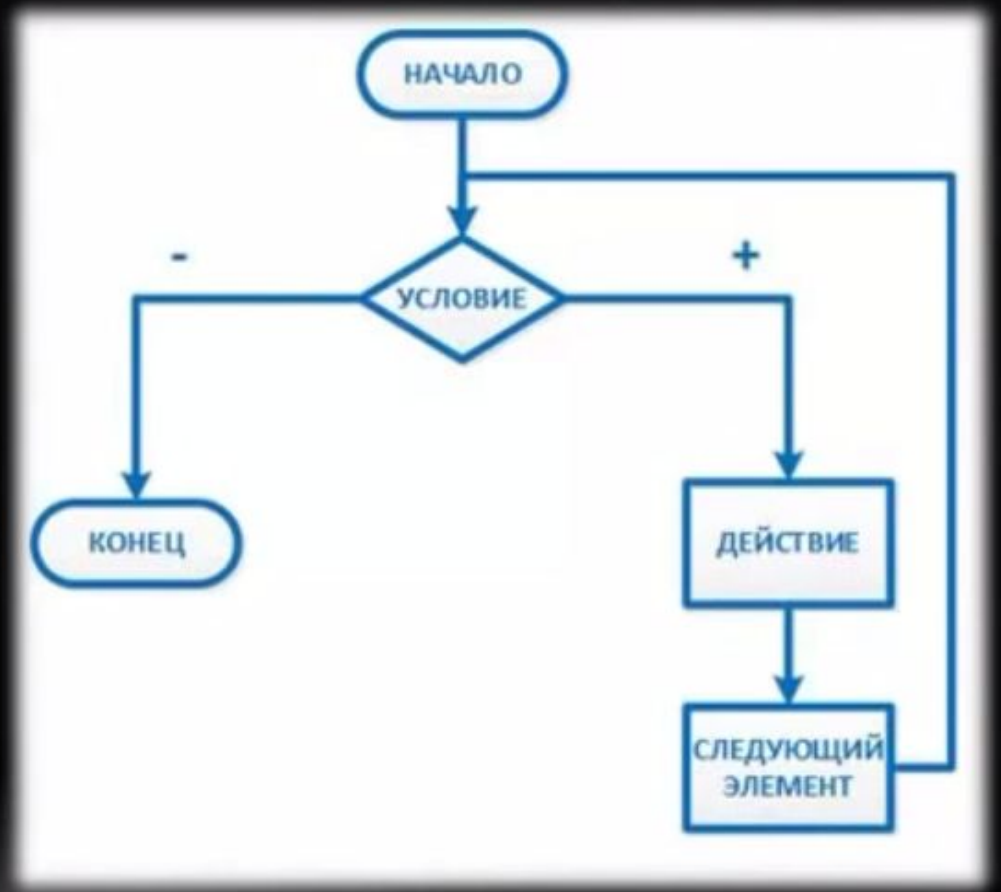


Цикл со счетчиком

```
#for#
Clear-Host
for ($a = 1; $a -lt 6; $a++)
{ Write-Host $a }

Write-Host '###############'

for ($i = 10; $i -gt 0; $i=$i-2)
{ Write-Host $i  }
```

ЦИКЛ FOREACH
WINDOWS POWERSHELL 2.0

```
foreach (ЭЛЕМЕНТ in МАССИВ)
        {
                ДЕЙСТВИЕ
        }
```



```powershell
Clear-Host
# Посчитать количество служб #
$tab = "`t"
$a = 1
foreach ($service  in Get-Service)
{
    Write-Host $a -NoNewline
    Write-Host $tab -NoNewline
    Write-Host $service.Status -NoNewline
    Write-Host $tab -NoNewline
    Write-Host $service.Name
    $a = $a + 1
}
```

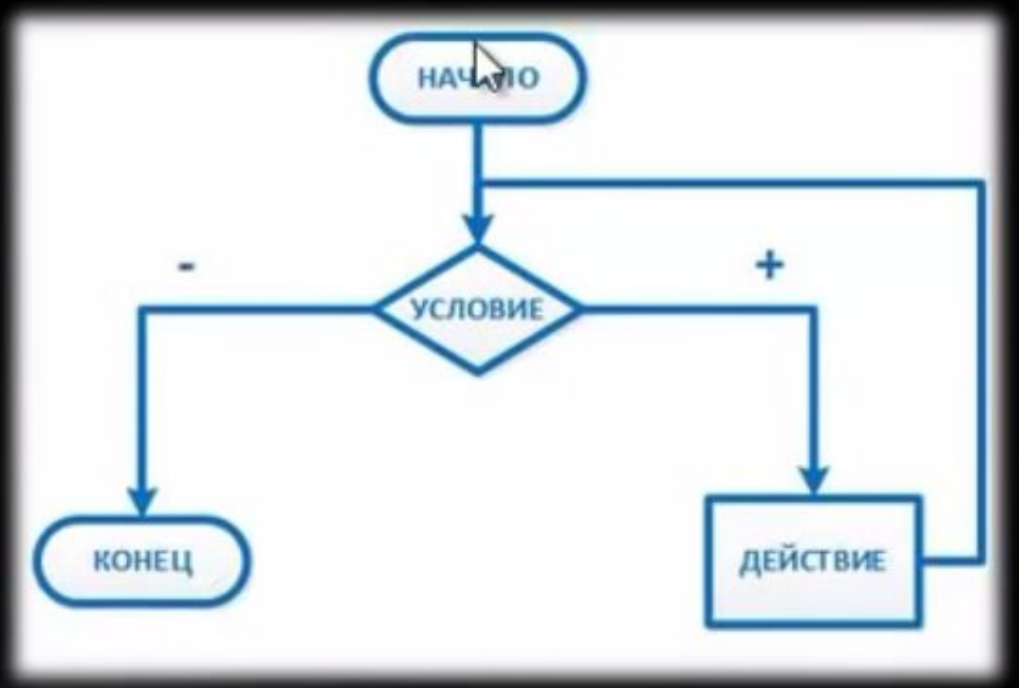
Практическое задание №3

Определить количество служб, находящихся в состоянии выполнения

```
Clear-Host
$tab = "`t"
$a = 0

foreach ($service in Get-Service)
{ if ($service | Where-Object -Property Status -like
"Running")
   {   $a = $a + 1
       Write-Host $a -NoNewline
       Write-Host $tab -NoNewline
       Write-Host $service.Name
   }
}
```

ЦИКЛ WHILE
WINDOWS POWERSHELL 2.0

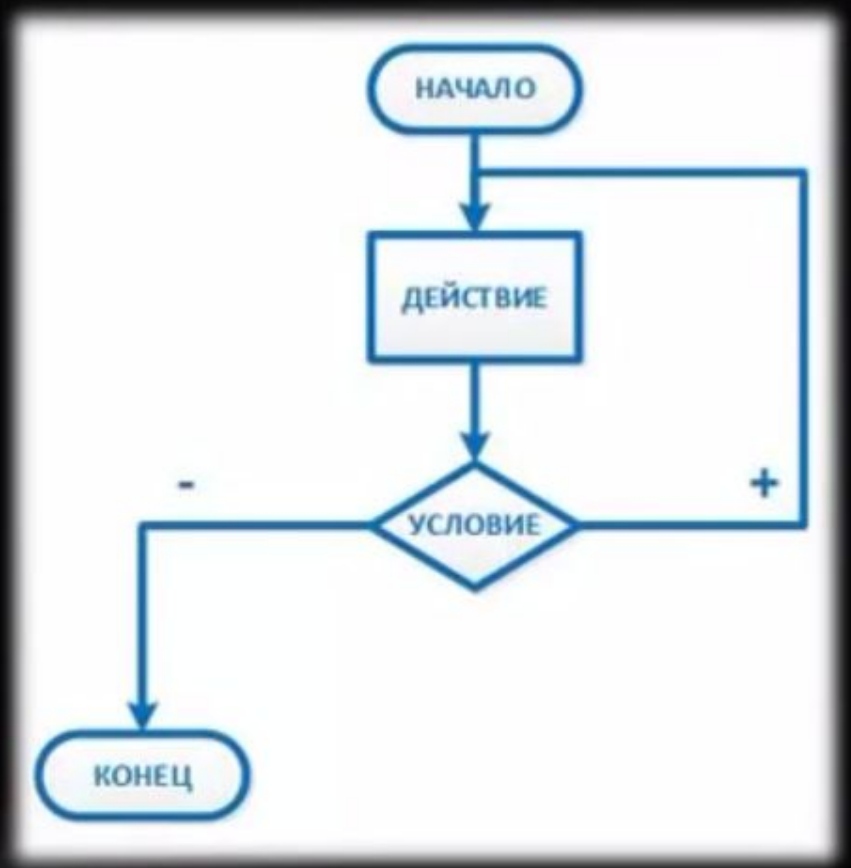
```
WHILE (УСЛОВИЕ)
        {
                ДЕЙСТВИЕ
        }
```



```
Clear-Host
$tab= “`t"
$a=0
while ($a -le 5)
{ Write-Host $a $tab -NoNewline
    $a++ }
```

ЦИКЛ DO WHILE
WINDOWS POWERSHELL 2.0

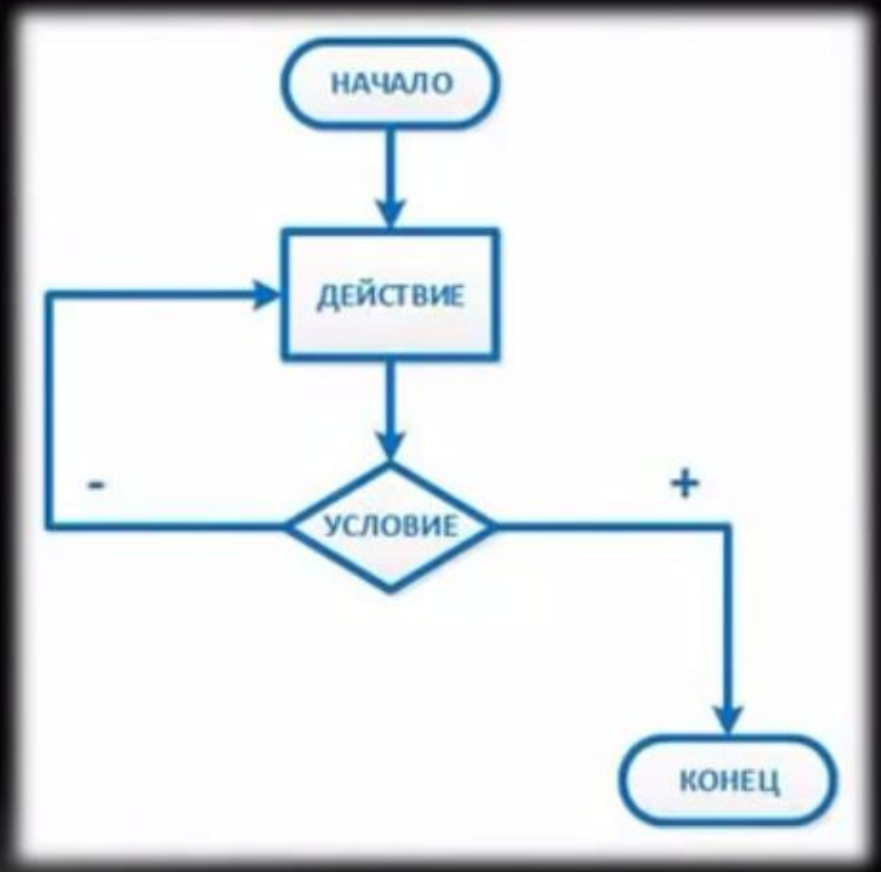
```
do      {
                ДЕЙСТВИЕ
        }
WHILE  (УСЛОВИЕ)

# УСЛОВИЕ ВЫХОДА $false
```



```
//Делать до тех пор, пока условие верно

Clear-Host
$a=10
    do
{ $a= Read-Host 'Введите значение
переменной $a'
  Write-Host $a }
    while ($a -eq 10)
```

ЦИКЛ DO UNTIL
WINDOWS POWERSHELL 2.0

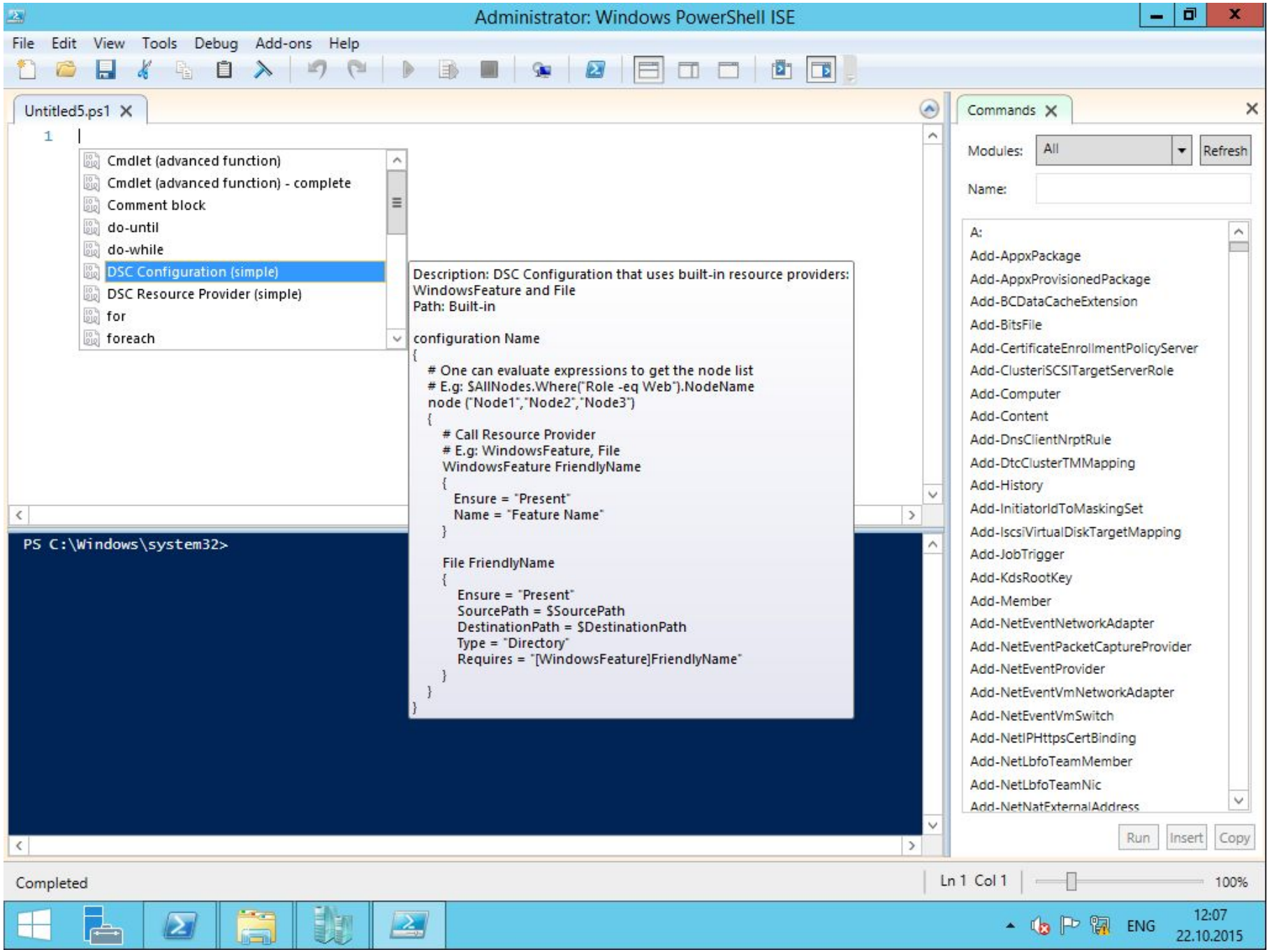
```
do      {
                ДЕЙСТВИЕ
        }
WHILE (УСЛОВИЕ)


# УСЛОВИЕ ВЫХОДА $true
```



```
 //Цикл, который повторяет набор команд,
пока выполняется условие

Clear-Host
$a=10
    do
{ $a= Read-Host 'Enter $a value'
  Write-Host $a }
    until ($a -eq 10)
```

Использование готовых шаблонов управляющих инструкций, DSC, Workflow

Одновременно нажмите на клавиши CTRL и J



1.Скопирова
установку с

2.Скопирова
машины с за

3.Запустить

4. Перейти
Internet скри

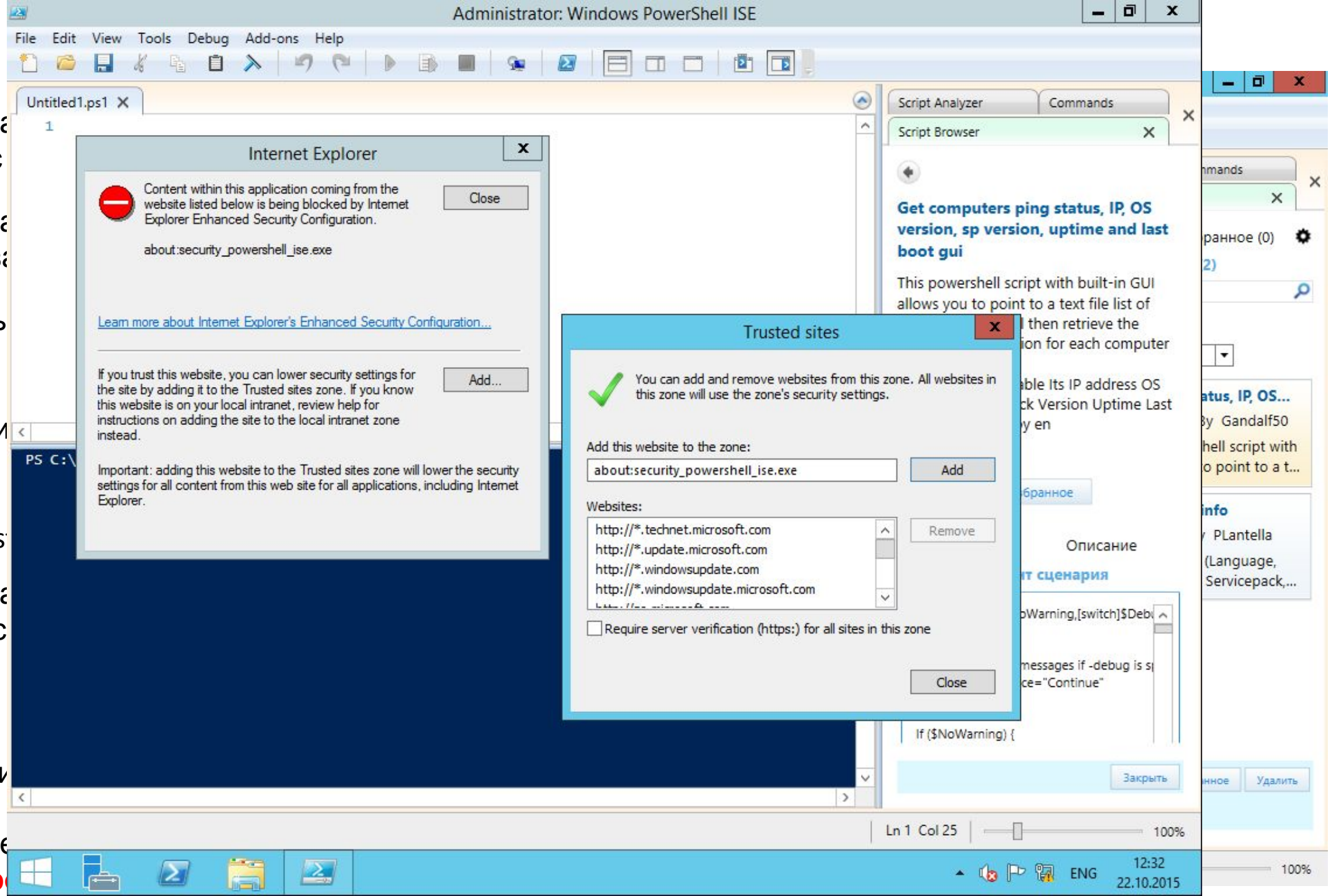
5. Выбрать
operating Sys

6. Два раза
(потребуетс

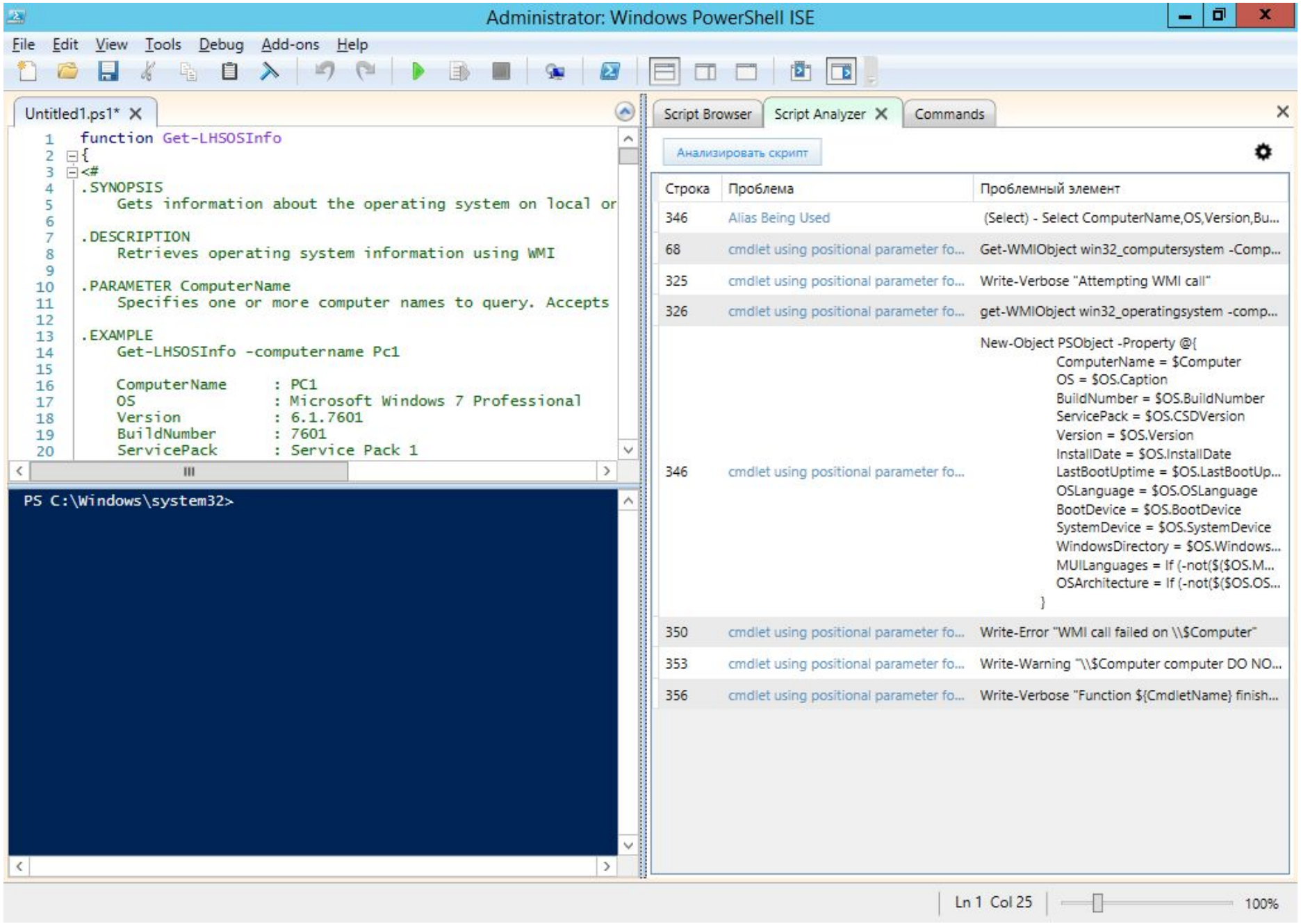
7. Нажмите

8. Теперь и
Analyzer.

9. В случае
«**Анализир**



Результат анализа скрипта при помощи оснастки Script Analyzer

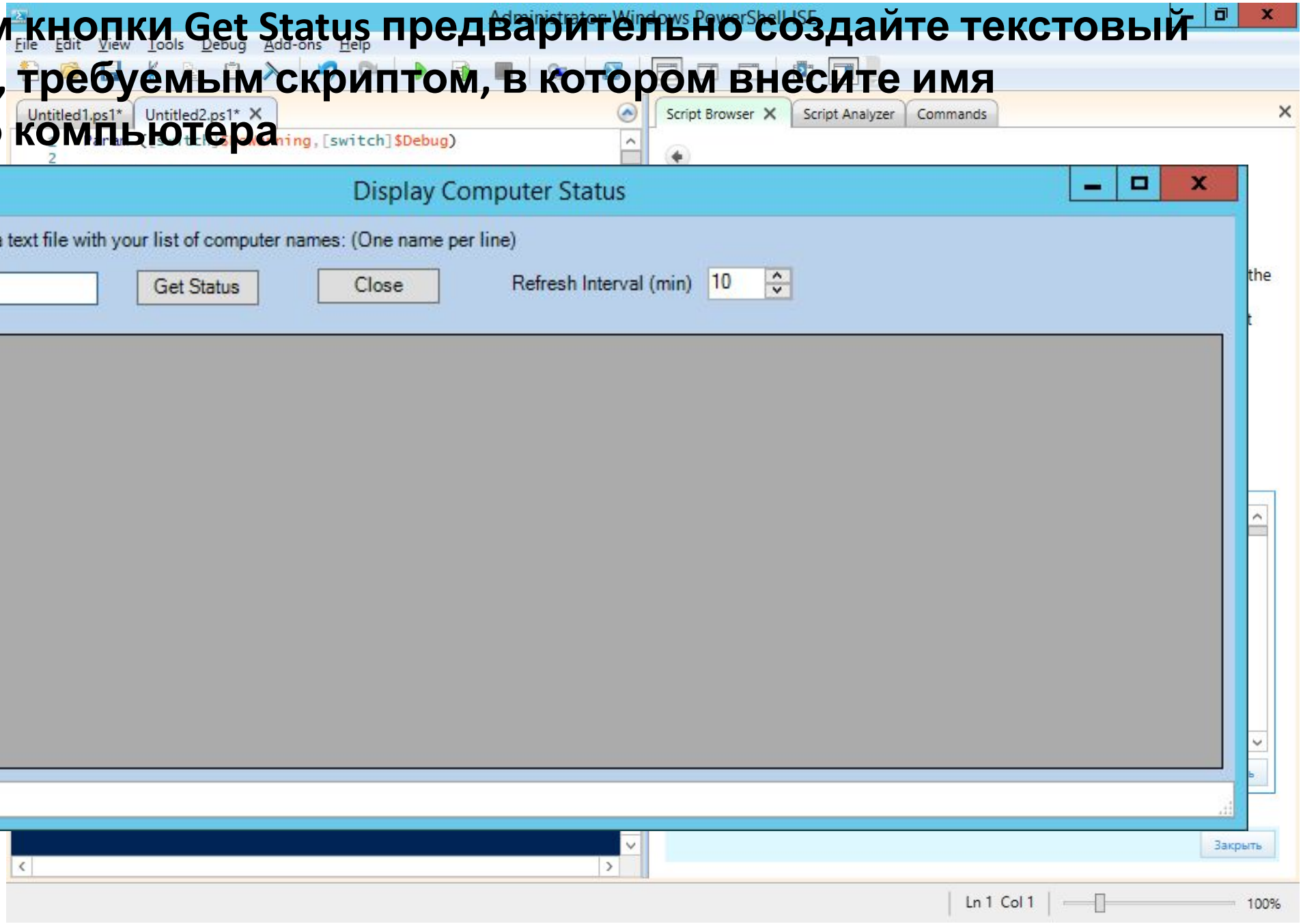


Перед нажатием кнопки Get Status предварительно создайте текстовый файл с именем, требуемым скриптом, в котором внесите имя опрашиваемого компьютера

Выберите первый скрипт (Get-computers …) при осна Brow его

Дан име граф инте кото разр Pow

Зап на и посмотрите его возможности

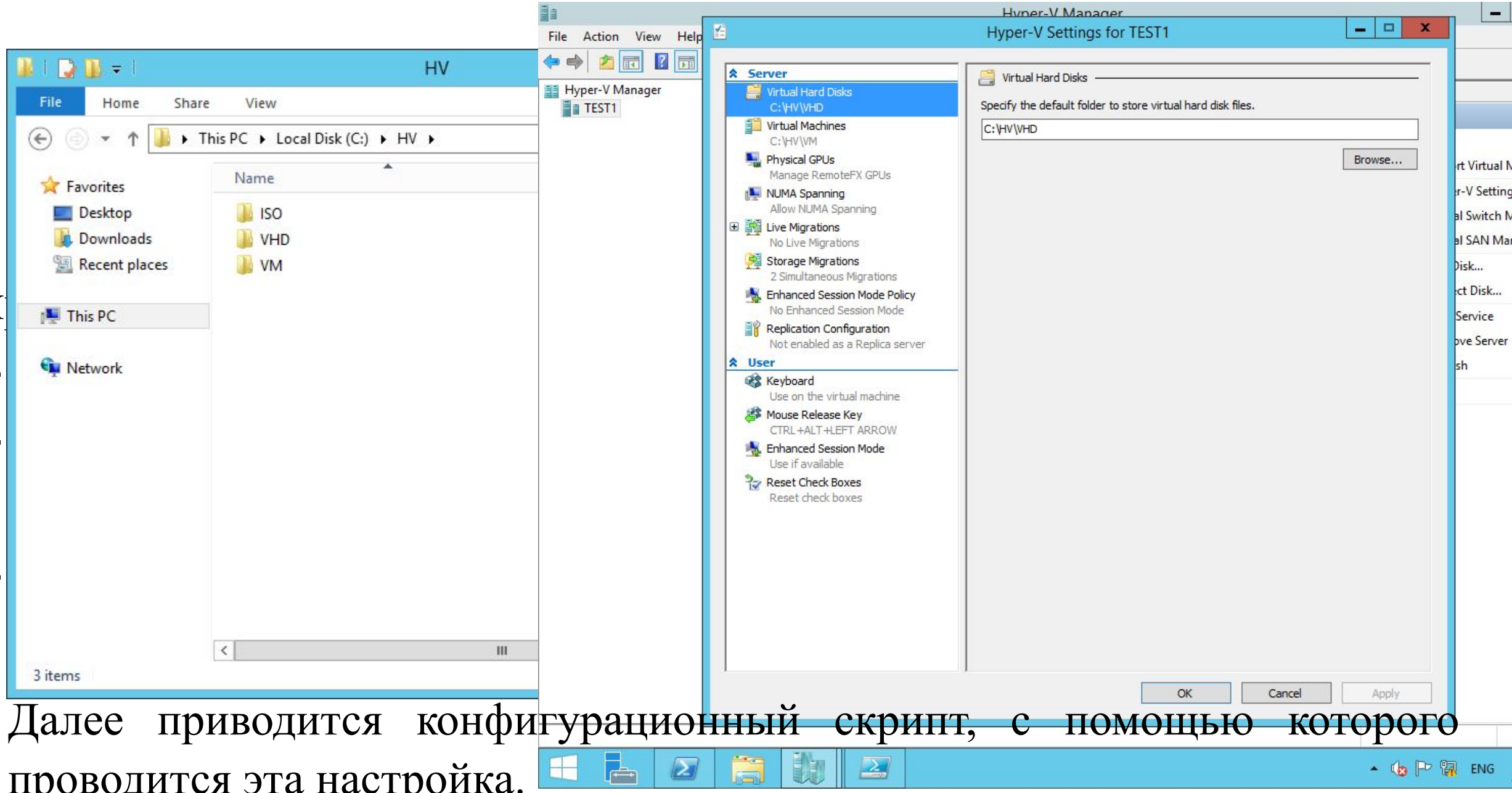


Вопрос №2

Технология PowerShell Desired State Configuration

Во многом работа системного администратора связана с выполнением **повторяющиеся** операций, которые сводятся к созданию, удалению, изменению настроек, установки и настройки ролей и компонентов системы. Абсолютно естественным является желание максимально **автоматизировать** такие задачи. Одним из инструментов, который может помочь администратору, является PowerShell Desired State Configuration (настройка требуемого состояния), которая впервые была представлена в Windows Server 2012 R2.

Используя PowerShell DSC, вы описываете как хотите, чтобы ваша система выглядела в конечном итоге, и далее происходит ее автоматическая настройка в соответствии с заданными требованиями.



Далее приводится конфигурационный скрипт, с помощью которого проводится эта настройка.

Методика использования DSC

Этап №1 Создать скрипт в Powershell_ISE

```
configuration hvconfig
{
node test1
 {
file createdirISO
 {
ensure = "present"
type = "Directory"
DestinationPath = "C:\HV\ISO"
 }
File CreateDirVhd
 {
ensure = "present"
Type = "Directory"
DestinationPath= "C:\HV\VHD"
 }
File CreateDirVM
 {
ensure = "Present"
Type = "Directory"
DestinationPath = "c:\HV\VM"
DependsOn = "[File]CreateDirVhd"
 }
windowsFeature installhvtools
 {
ensure = "Present"
name = "RSAT-Hyper-V-Tools"
IncludeAllSubFeature = $true
DependsOn = "[File]CreateDirVM"
 }
windowsFeature installHV
 {
ensure = "Present"
name = "Hyper-V"
IncludeAllSubFeature = $true
DependsOn = "[WindowsFeature]installhvtools"
 }
Registry HVSettingsVM
 {
ensure = "present"
key = "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Virtualization"
ValueName = "DefaultExternalDataRoot"
ValueData = "C:\HV\VM"
DependsOn = "[WindowsFeature]installHV"
 }
Registry HVSettingsVHD
 {
ensure = "present"
 key = "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Virtualization"
ValueName = "DefaultVirtualHardDiskPath"
ValueData = "C:\HV\VHD"
DependsOn = "[WindowsFeature]installHV"
}
}
}
```

Этап №2. Выполнить скрипт в Powershell_ISE (клавиша на клавиатуре **F5**)

Этап №3. Создать mof - файл в Powershell_ISE выполнив команду: **hvconfig** (название задается параметром **Configuration** на первом этапе)

PowerShell DSC это конфигурационный скрипт и начинается он с ключевого слова ***Configuration***, с помощью которого описывается главный контейнер конфигурации. По сути, этот блок является функцией. Внутри блока *Configuration* вы можете указать желаемые настройки для каждого из компьютеров, которые есть в вашем окружении.

Блок конкретного компьютера начинается с ключевого слова ***Node***. Далее следует имя целевого компьютера, которое может быть, как постоянным, так и задаваться при помощи переменной. Внутри блока *Node* вы уже указываете желаемую настройку для вашего конечного компьютера, с помощью **ресурсов**.

Ресурсы DSC – это специализированные модули PowerShell, с помощью которых и осуществляется финальная настройка целевых узлов. Ресурсы разделяются на встроенные и пользовательские. Встроенных ресурсов всего 12. Тем не менее легко можно дописать недостающие ресурсы, если возникнет такая необходимость.

Встроенные ресурсы	Назначение
DSC Archive Resource	Распаковывает архивы в формате zip с возможностью указания пути
DSC Environment Resource	Управление системным переменным окружением
DSC File Resource	Управление файлами и директориями
DSC Group Resource	Управление локальными группами
DSC Log Resource	Управление Log файлами
DSC Package Resource	Установка и управление пакетами, поиск Windows Installer и Setup.exe пакетов
DSC WindowsProcess Resource	Управление процессами
DSC Registry Resource	Управление реестром
DSC WindowsFeature Resource	Установка и удаление ролей и компонентов Windows
DSC Script Resource	Запуск Windows Powershell скриптов
DSC Service Resource	Управление службами
DSC User Resource	Управление учетными записями пользователей

Для описания того, как наша система должна будет выглядеть, используем:

```
Ensure = "Present"
```

С помощью этой команды мы убеждаемся, что нужные нам компоненты будут присутствовать на конечном компьютере, после того, как конфигурация будет применена.

Применение конфигурации выглядит следующим образом:

После запуска конфигурационного скрипта создается MOF-файл (Managed Object Format файл). Это текстовый файл, в котором содержатся все требования по настройке, которые в дальнейшем применяются на целевом компьютере. Имя MOF-файла будет соответствовать значению *Node*. Сам файл будет находиться в папке, название которой будет совпадать с названием *Configuration*. Здесь важно отметить, что использование MOF-файлов позволяет использовать DSC не только для настройки компьютеров под управлением Windows, но и под управлением Linux.

Далее необходимо MOF-файл передать на целевой компьютер (на котором мы хотим развернуть роль). Конфигурация применяется двумя способами: с помощью метода Push (конфигурационный скрипт должен быть перенесен на конечный компьютер вручную) или метода Pull (создается Pull Server, который периодически проверяет корректность конфигурации, и если клиент сконфигурирован неверно, то Pull Server отправляет на него требуемые настройки).

Применение конфигурации осуществляется с помощью следующих командлетов (в случае необходимости выбора учетной записи):

```
$Session = New-CimSession -ComputerName "test1" -Credential Administrator

Start-DscConfiguration -Path C:\windows\system32\hvconfig -CimSession $Session
-Wait -Verbose
```

или если учетные записи совпадают:

```
Start-DscConfiguration -Path C:\windows\system32\hvconfig -wait -verbose
```

С помощью параметра –Path указываем путь к MOF-файлу. Время применения конфигурации зависит от того, насколько соответствует текущая настройка компьютера тем требованиям, что указаны в MOF-файле.

После применения конфигурации возможность PowerShell DSC не заканчиваются. Ведь часто нам нужно определить произошли ли какие-то изменения в настройках. Сделать это можно с помощью командлета:

```
Test-DscConfiguration –CimSession $session
```

Запустив его, мы запустим проверку: совпадает ли текущая конфигурация системы с той, что прописана в MOF-файле. Если конфигурации совпадают, то будет возвращено значение «True», в противном случае – «False».

Что делать, если мы узнаем об изменении конфигурации? Если мы используем PowerShell DSC достаточно всего лишь снова запустить командлет и все недостающие элементы будут восстановлены:

```
Start-DscConfiguration -Path C:\windows\system32\hvconfig
-wait -verbose
```

Преимущества PowerShell Desired State Configuration является то, что его использование позволяет точно настроить конечный компьютер без каких-либо дополнительный проверок. Мы просто описываем как хотим, чтобы системы выглядела; все остальное делается автоматически, во-вторых PowerShell DSC позволяет отследить возможные изменения настроек и быстро их исправить; в-третьих, с помощью PowerShell DSC можно одновременно и одинаково сконфигурировать несколько компьютеров.

Практическое задание №1

Включите удаленное управление на сервере **test1**, настройте доверие между компьютерами и проверьте возможность удаленного управления сервером **test1** с компьютера **test2**.

Практическое задание №2

Создайте файл скрипта DSC на компьютере **test2** и выполните его. Примените mof – файл на удаленном сервере. Перегрузите удаленный сервер и проверьте применение требуемой конфигурации.

Практическое задание №3

Проверьте состояние исполненной конфигурации. Измените параметры конфигурации (удалите подкаталог ISO) на сервере **test1** и проверьте состояние измененной конфигурации. Исправьте измененную конфигурацию на исходную.

Вопрос №3

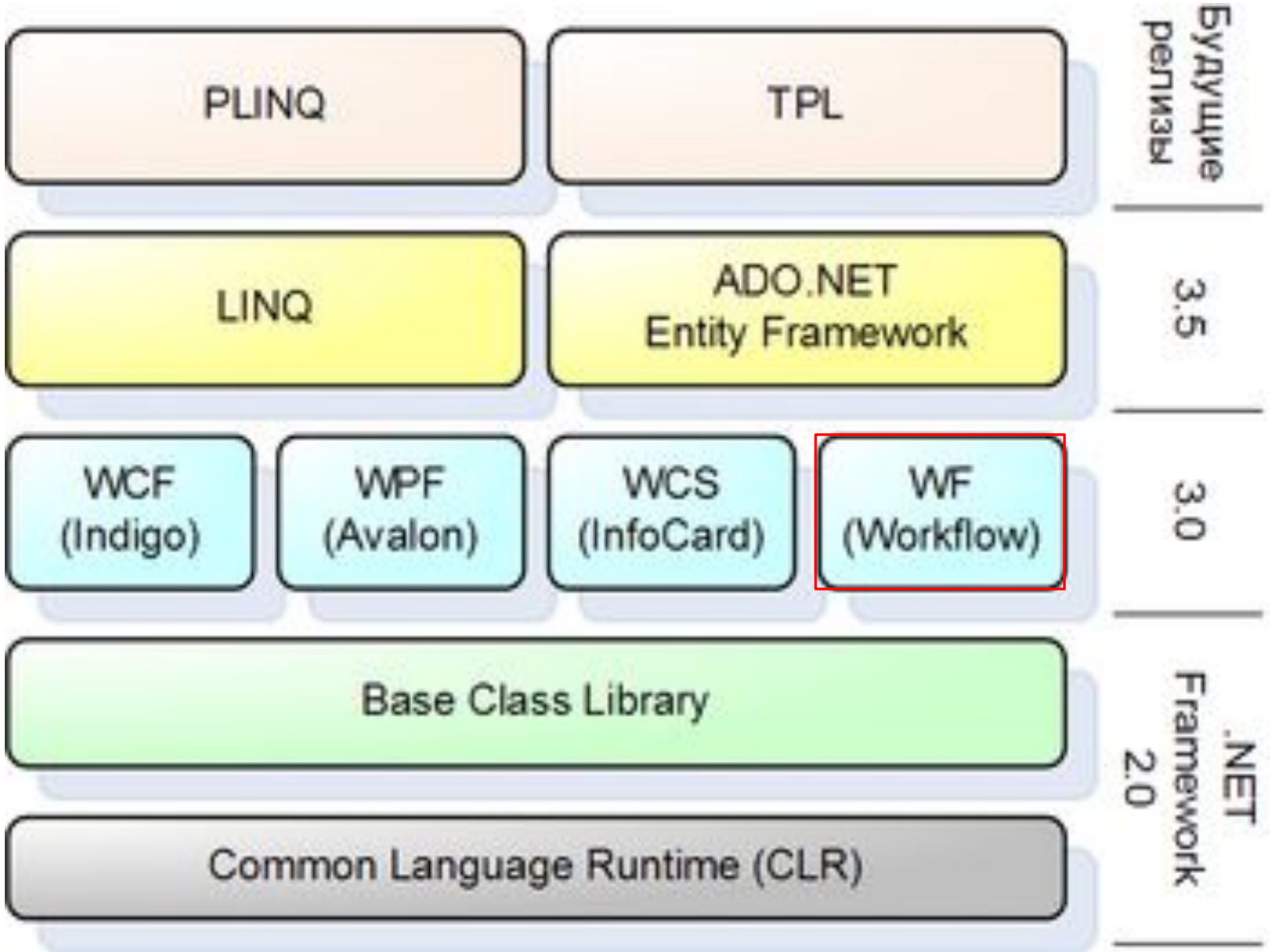
Разновидность сценариев PowerShell Workflow

Windows PowerShell Workflow (рабочий процесс) нововведение Windows Management Framework версии 3.0, представляющий собой специализированную разновидность сценариев Windows PowerShell. Он описывает набор операций, одни из которых должны выполняться в определенной последовательности, тогда как другие могут выполняться параллельно.

Windows PowerShell Workflow устанавливается вместе с Windows Server 2012 и Windows 8. Он также доступен для Windows 7, Windows Server 2008 и Windows Server 2008 R2. Для запуска рабочего процесса обязательно нужна одна из этих ОС. Однако адресатом рабочего процесса, т. е. системой, в который запускаются задания, может быть любая версия Windows, в зависимости от того, какое именно задание вы пытаетесь выполнить.

При написании рабочих процессов используется стандартный синтаксис Windows PowerShell. Windows PowerShell Workflow активно использует функции удаленного выполнения (remoting) Windows PowerShell.

Windows PowerShell Workflow не выполняет рабочий процесс, а преобразовывает сценарий в XAML (англ. eXtensible Application Markup Language) — расширяемый язык разметки для приложений, основанный на XML языке разметки для декларативного программирования приложений, разработанный Microsoft, а затем передает его Windows Workflow Foundation (WWF), являющейся частью Microsoft .NET Framework.


PLINQ
TPL
LINQ
ADO.NET
Entity Framework
WCF
(Indigo)
WPF
(Avalon)
WCS
(InfoCard)
WF
(Workflow)
Base Class Library
Common Language Runtime (CLR)
Будущие релизы
3.5
3.0
.NET Framework 2.0

WWF принимает XAML-данные и выполняет рабочий процесс. В WWF предусмотрены **контрольные точки** для отслеживания хода рабочего процесса. Поэтому, если компьютер, на котором выполняется рабочий процесс, по каким-либо причинам завершит работу (например, его выключат), то после повторного запуска компьютера выполнение процесса продолжится с того места, на котором оно остановилось.

Кроме того, можно вручную приостанавливать и возобновлять рабочие процессы. Например, у вас может быть рабочий процесс, который выполняет определенный набор заданий до момента, когда требуется ручное вмешательство. Вы можете сделать так, чтобы в этот момент он приостанавливал работу и автоматически отправлял сообщение об изменении статуса по электронной почте. После выполнения ручных операций можно возобновить рабочий процесс и позволить ему выполняться дальше.

Если нужно, рабочие процессы могут выполнять операции параллельно. Например, если у вас имеется набор независимых заданий, которые можно выполнять в любом порядке, то их можно запустить примерно в одно и то же время. Это сократит общее время выполнения рабочего процесса, а значит, повысит эффективность. Кроме того, WWF отслеживает каждый этап рабочего процесса. Это значит, что можно сгенерировать подробный журнал аудита с данными о том, какие задания выполнялись, очень полезный при устранении неполадок.

Практический пример использования workflow

Задача: Развернуть **50** виртуальных машин одинаковой конфигурации с именами **VM<номер>** от 1 до 50.

Решение № 1. Написать код без использования массивов и циклов: Последовательный запуск 50 раз командлета New-VM с различными именами виртуальных машин.

Решение № 2. Использовать массивы и циклы:

```
1..50 | ForEach-Object { New-VM -Name VM$_ -MemoryStartupBytes 512MB -NewVHDPath C:\HV\VHD\VM$_.vhdx -NewVHDSizeBytes 10GB }
```

В цикле задания выполняются последовательно. Этот тот случай, когда рабочий процесс может серьезно увеличить скорость выполнения вашего скрипта.

Решение №3. Использование workflow. Для этого необходимо добавить параметр –Parallel в командлете `ForEach-Object`, который позволяет выполнять 5 потоков одновременно. Таким образом, если ваше оборудование справится с нагрузкой, то создание виртуальных машин может стать до 5 раз быстрее.

```
 workflow New-VirtualMachines
{
   $VMs = 1..50
   foreach -parallel ($VM in $VMs)
   {
     New-VM -Name VM$VM -MemoryStartupBytes 512MB
-NewVHDPath C:\HV\VHD\VM$VM.vhdx -NewVHDSizeBytes 10GB
   }
}
```

Рабочие процессы могут принимать параметры, так же, как и функции, например следующий рабочий процесс в качестве входных параметров принимает имя виртуальной машины, число создаваемых виртуальных машин и размер виртуального диска:

```
workflow New-VirtualMachines
{
  [CmdletBinding()]
  Param (
  [Parameter(Mandatory=$true,Position=0)]$VMName,
  [Parameter(Mandatory=$true,Position=1)]$VMCount,
  [Parameter(Mandatory=$true,Position=2)]$VHDSize
  )
    $VMs = 1..$VMCount
    foreach -parallel ($VM in $VMs)
    {
    New-VM -Name $VMName$VM -MemoryStartupBytes 512MB -NewVHDPath
C:\HV\VHD\$VMName$VM.vhdx -NewVHDSizeBytes $VHDSize
  }
}
```

Вызываем рабочий процесс так же, как и функцию:

```
New-VirtualMachines -VMName VM -VMCount 50 -VHDSize 10GB
```

Практическое задание № 1

Оценка времени исполнение скрипта с использованием цикла с последовательным выполнением

```
$watch = [System.Diagnostics.Stopwatch]::StartNew()
$watch.Start() #Запуск таймера

1..50 | ForEach-Object { New-VM -Name VM$_ -MemoryStartupBytes
512MB -NewVHDPath C:\HV\VHD\VM$_.vhdx -NewVHDSizeBytes 10GB }

$watch.Stop() #Остановка таймера
Write-Host $watch.Elapsed
```

Удалите созданные виртуальные машины и жесткие диски

//Удалить созданные виртуальные жесткие диски

```
Remove-Item -Path c:\hv\vhd\*
```

//Удалить созданные виртуальные машины

```
Remove-VM -ComputerName test1 -Name vm* -Force
```

Практическое задание № 2

Оценка времени исполнение скрипта с использованием workflow

```
$watch = [System.Diagnostics.Stopwatch]::StartNew()
$watch.Start() #Запуск таймера

workflow New-VirtualMachines
{
  [CmdletBinding()]
  Param (
  [Parameter(Mandatory=$true,Position=0)]$VMName,
  [Parameter(Mandatory=$true,Position=1)]$VMCount,
  [Parameter(Mandatory=$true,Position=2)]$VHDSize
  )
    $VMs = 1..$VMCount
    foreach -parallel ($VM in $VMs)
     {
    New-VM -Name $VMName$VM -MemoryStartupBytes 512MB -NewVHDPath C:\HV\VHD\$VMName$VM.vhdx -NewVHDSizeBytes
$VHDSize
  }
}

New-VirtualMachines -VMName VM -VMCount 50 -VHDSize 10GB

$watch.Stop() #Остановка таймера
Write-Host $watch.Elapsed
```

Удалите созданные виртуальные машины и жесткие диски

//Удалить созданные виртуальные жесткие диски

```
Remove-Item -Path c:\hv\vhd\*
```

//Удалить созданные виртуальные машины

```
Remove-VM -ComputerName test1 -Name vm* -Force
```

Практическое задание №3

Проверка функции Workflow – восстановление при сбое

Подключитесь к удаленному серверу (test1) и запустите сценарий Powershell Workflow (прз.№3 без таймера). Посмотрите на начало развертывания виртуальных машин Hyper-V на удаленном сервере с помощью Hyper-V менеджера. Поставьте на паузу виртуальную машину Vmware (test1), на которой начался рабочий процесс. Подождите 30 секунд и включите оставленную виртуальную машину Vmware. Посмотрите информацию, выдаваемую сценарием workflow (на test2) и процесс развертывания виртуальных машин Hyper-V на удаленном сервере (**test1**). Примечание: разрыв сетевого соединения не должен быть более 4 минут. В случае разрыва соединения более 4 минут сессия прервется и для ее восстановления необходимо будет подключаться заново.

```
WARNING: The network connection to test1 has been interrupted. Attempting to reconnect for up
to 4
minutes...
WARNING: Attempting to reconnect to test1 ...
```

Удалите созданные виртуальные машины и жесткие диски

//Удалить созданные виртуальные жесткие диски

```
Remove-Item -Path c:\hv\vhd\*
```

//Удалить созданные виртуальные машины

```
Remove-VM -ComputerName test1 -Name vm* -Force
```

Практическое задание №4

Приостановка и продолжение сценария Powershell workflow

Обратите внимание: Внутри блока parallel, контрольная точка не создается, пока параллельная обработка не завершится для всех данных! Для создания контрольных точек в тело цикла необходимо добавить Checkpoint-Workflow.

```
workflow New-VirtualMachines
{
  [CmdletBinding()]
  Param (
  [Parameter(Mandatory=$true,Position=0)]$VMName,
  [Parameter(Mandatory=$true,Position=1)]$VMCount,
  [Parameter(Mandatory=$true,Position=2)]$VHDSize
  )
    $VMs = 1..$VMCount
    foreach ($VM in $VMs) # -parallel должен отсутствовать
         {
        New-VM -Name $VMName$VM -MemoryStartupBytes 512MB -NewVHDPath C:\HV\VHD\$VMName$VM.vhdx
-NewVHDSizeBytes $VHDSize
        Checkpoint-Workflow
     }
}
```

// Запуск скрипта в виде работы

New-VirtualMachines -VMName VM -VMCount 50 -VHDSize 10GB -AsJob

//Определение номера или имени работы

Get-Job

// Приостановка работы по имени

Suspend-Job –name Jobx

// Просмотр выполненных заданий на момент остановки работы по ее ID

Receive-Job –Id x

// Возобновить работу по ее имени

Resume-Job –name Jobx

//Остановить службу по ее имени

Stop-Job –name Jobx

// Удалить работу по ее имени

Remove-job –name Jobx

// Добавление модуля PSWorkflow при необходимости.

Import-module PSWorkflow