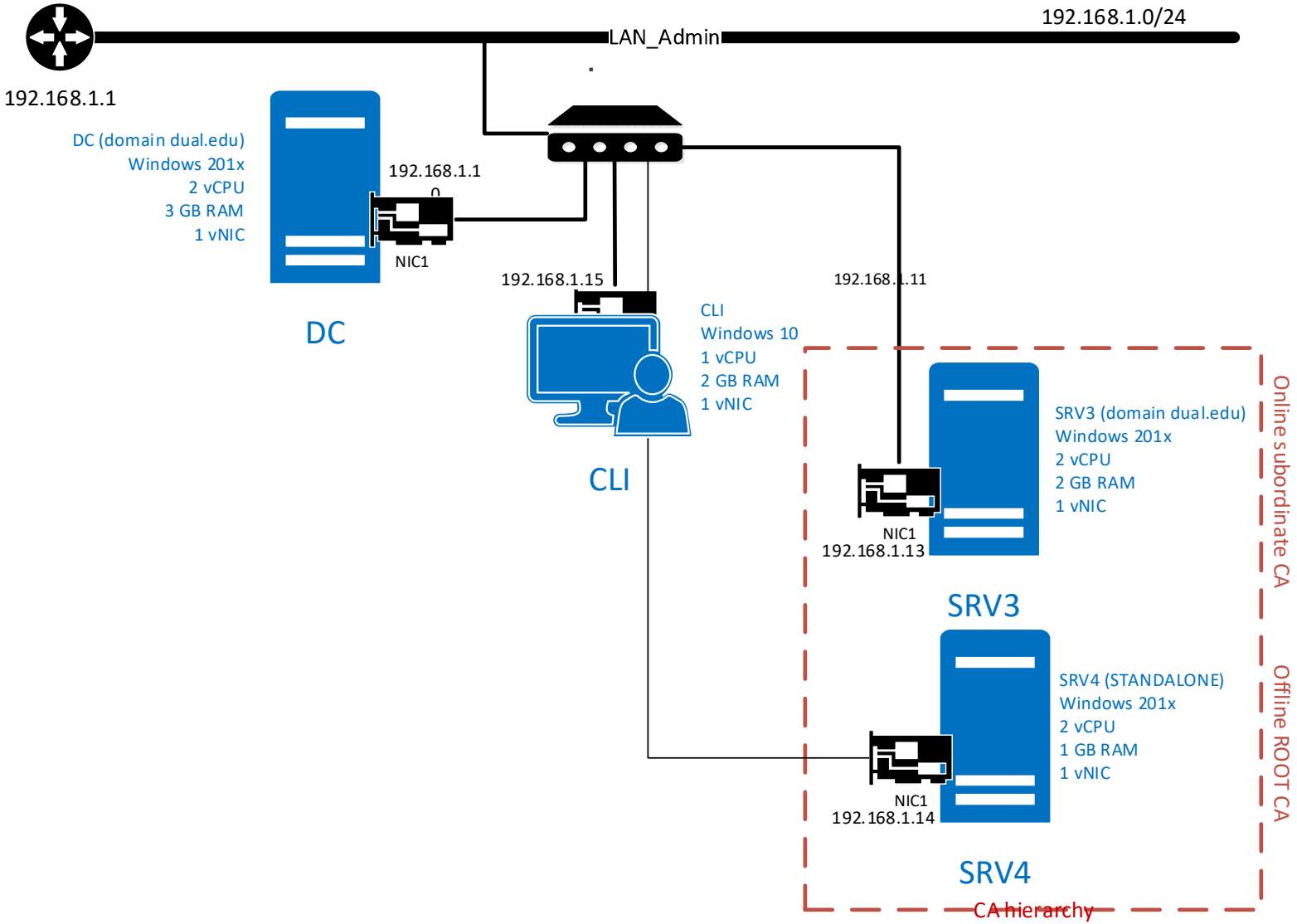
PowerShell pt3

Špecializované IKT systémy Windows

Ing. Stanislav Lukac, PhD.

LAB: CA installation



Agenda

1. PowerShell Remoting
2. PowerShell WMI and CIM
3. PowerShell Functions
4. PowerShell Scripts
5. PowerShell Error Handling

PS Remoting

Špecializované IKT systémy Windows

CMDLETs wRemoting

```
Windows PowerShell
PS C:\> Get-Command -ParameterName ComputerName | Format-Wide -Column 4


Add-Computer                Clear-EventLog              Connect-PSSession           Enter-PSSession
Get-EventLog                Get-HotFix                  Get-Process                 Get-PSSession
Get-Service                 Get-WmiObject               Invoke-Command              Invoke-WmiMethod
Limit-EventLog              New-EventLog                New-PSSession               Receive-Job
Receive-PSSession           Register-WmiEvent           Remove-Computer             Remove-EventLog
Remove-PSSession            Remove-WmiObject            Rename-Computer             Restart-Computer
Send-MailMessage            Set-Service                 Set-WmiInstance             Show-EventLog
Stop-Computer               Test-Connection             Write-EventLog
```

```
Windows PowerShell
PS C:\> Get-Command -Noun *Computer* | Format-Wide -Column 4


Get-MpComputerStatus        Add-Computer                Checkpoint-Computer         Disable-ComputerRestore
Enable-ComputerRestore      Get-ComputerInfo            Get-ComputerRestorePoint    Remove-Computer
Rename-Computer             Reset-ComputerMachinePassword Restart-Computer          Restore-Computer
Stop-Computer               Test-ComputerSecureChannel
```

Remoting via WMI

- MS implementation of industry standard
- It was designed primary to read info
- For remote access is use DCOM protocol
- Is organize into Namespaces, Classes and Instances
- Default namespace is root\cimv2
- Each product has own WMI providers to take info out of product (Windows, SQL, Exchange…)
- By default, only administrators have access to wmi remotelly

DEMO::

Windows PowerShell

```
PS C:\> Get-WmiObject -class win32_* -List | Format-Wide -Column 3


Win32_DeviceChangeEvent              Win32_SystemConfigurationChangeEvent    Win32_VolumeChangeEvent
Win32_SystemTrace                    Win32_ProcessTrace                      Win32_ProcessStartTrace
Win32_ProcessStopTrace               Win32_ThreadTrace                       Win32_ThreadStartTrace
Win32_ThreadStopTrace                Win32_ModuleTrace                       Win32_ModuleLoadTrace
Win32_PowerManagementEvent           Win32_ComputerSystemEvent               Win32_ComputerShutdownEvent
Win32_IP4RouteTableEvent             Win32_LogicalDisk                       Win32_MappedLogicalDisk
Win32_DiskPartition                  Win32_Volume                            Win32_CacheMemory
Win32_SMBIOSMemory                   Win32_MemoryArray                       Win32_MemoryDevice
Win32_DiskDrive                      Win32_TapeDrive                         Win32_CDROMDrive
Win32_PnPEntity                      Win32_1394Controller                    Win32_VideoController
```

DEMO::

```
Windows PowerShell
PS C:\> Get-WmiObject -Class win32_Bios


SMBIOSBIOSVersion : F2
Manufacturer      : American Megatrends Inc.
Name              : BIOS Date: 07/04/14 15:16:34 Ver: 04.06.05
SerialNumber      : To be filled by O.E.M.
Version           : ALASKA - 1072009



PS C:\> Get-WmiObject -Class win32_OperatingSystem


SystemDirectory : C:\WINDOWS\system32
Organization    :
BuildNumber     : 16299
RegisteredUser  : Windows User
SerialNumber    : 00330-80000-00000-AA600
Version         : 10.0.16299



PS C:\> Get-WmiObject -Class win32_ComputerSystem


Domain              : WORKGROUP
Manufacturer        : Gigabyte Technology Co., Ltd.
Model               : B85-HD3
Name                : MASTER
PrimaryOwnerName    : Windows User
TotalPhysicalMemory : 25727885312
```

CIM

- MS long term strategy is CIM
- CIM is comming with PS v3
- default name space is root\cimv2
- CIM use protocol HTTP(s) (WinRM) by default
- CIM cmdlet use CIMSessions
- To enable CIM connection to remote computer, both comp must use WSMan 3.0
- Again are using serialization/deserialization methods

DEMO::

```
Windows PowerShell

PS C:\> Get-CimInstance -ComputerName S -ClassName C -Filter F -Namespace N ^C
PS C:\>
PS C:\> Get-CimInstance -ClassName win32_Bios


SMBIOSBIOSVersion : F2
Manufacturer      : American Megatrends Inc.
Name              : BIOS Date: 07/04/14 15:16:34 Ver: 04.06.05
SerialNumber      : To be filled by O.E.M.
Version           : ALASKA - 1072009



PS C:\> Get-CimInstance -ClassName win32_OperatingSystem

SystemDirectory     Organization BuildNumber RegisteredUser SerialNumber           Version
---------------     ------------ ----------- -------------- ------------           -------
C:\WINDOWS\system32              16299       Windows User   00330-80000-00000-AA600 10.0.16299


PS C:\> Get-CimInstance -ClassName win32_ComputerSystem

Name            PrimaryOwnerName     Domain            TotalPhysicalMemory  Model            Manufacturer
----            ----------------     ------            -------------------  -----            ------------
MASTER          Windows User         WORKGROUP         25727885312          B85-HD3          Gigabyte Technol.
```

DEMO::

```
Administrator: Windows PowerShell
PS C:\>
PS C:\>  Get-WmiObject -Class win32_OperatingSystem | Select-Object -Property InstallDate,LastBootUpTime,LocalDateTime

InstallDate               LastBootUpTime            LocalDateTime
-----------               --------------            -------------
20171021090425.000000+060 20180217174600.563863+060 20180226183429.442000+060


PS C:\>  Get-CimInstance -Class win32_OperatingSystem | Select-Object -Property InstallDate,LastBootUpTime,LocalDateTime


InstallDate           LastBootUpTime       LocalDateTime
-----------           --------------       -------------
21-Oct-17 10:04:25 AM 17-Feb-18 5:46:00 PM 26-Feb-18 6:34:37 PM
```

```
Administrator: Windows PowerShell
PS C:\>
PS C:\> Invoke-WmiMethod -Class Win32_Process -Name Create -ArgumentList "Notepad.exe"


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 2
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ProcessId        : 548
ReturnValue      : 0
PSComputerName   :
```

PowerShell remoting

- Feature from PS v2.0
- Enabled by default from Windows server 2012
- Authentication by default done by Kerberos
- Built on top WinRM (WInRM is MS impelementation of WS-Management protocol)
- default TCP 5985,5986

- To configure you need to run Enable-PSRemoting cmdlet

- Test-WSMan -ComputerName <computerName>
- Get-Service WinRM -*ComputerName* SRV1

DEMO::

Administrator: Windows PowerShell

```
PS C:\> Set-WSManQuickConfig

WinRM Quick Configuration
Running the Set-WSManQuickConfig command has significant security implications, as it enables
remote management through the WinRM service on this computer.
This command:
 1. Checks whether the WinRM service is running. If the WinRM service is not running, the service
is started.
 2. Sets the WinRM service startup type to automatic.
 3. Creates a listener to accept requests on any IP address. By default, the transport is HTTP.
 4. Enables a firewall exception for WS-Management traffic.
 5. Enables Kerberos and Negotiate service authentication.
Do you want to enable remote management through the WinRM service on this computer?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"):
WinRM is already set up to receive requests on this computer.
WinRM is already set up for remote management on this computer.
PS C:\>
```

DEMO::

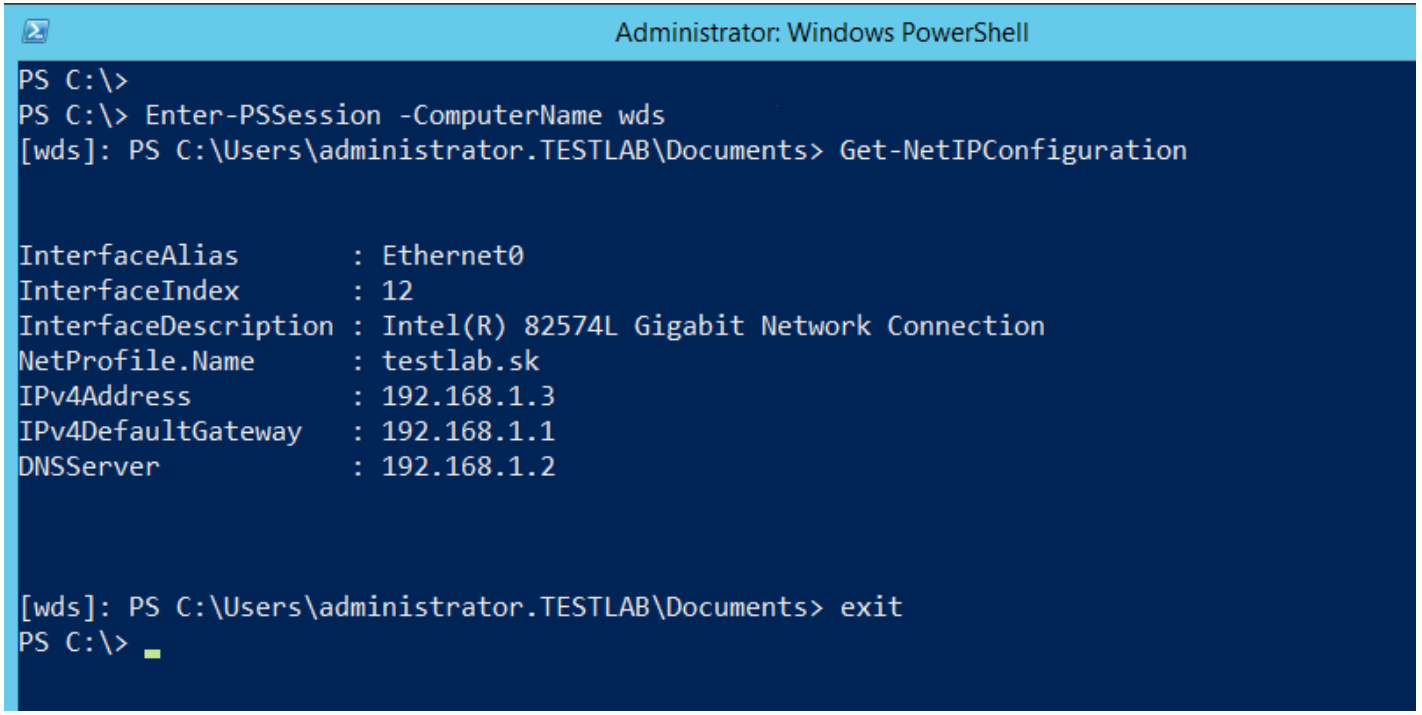
```
Examples:
winrs -r:https://myserver.com command
winrs -r:myserver.com -usessl command
winrs -r:myserver command
winrs -r:http://127.0.0.1 command
winrs -r:http://169.51.2.101:80 -unencrypted command
winrs -r:https://[::FFFF:129.144.52.38] command
winrs -r:http://[1080:0:0:0:8:800:200C:417A]:80 command
winrs -r:https://myserver.com -t:600 -u:administrator -p:$%fgh7 ipconfig
winrs -r:myserver -env:PATH=^%PATH^%;c:\tools -env:TEMP=d:\temp config.cmd
winrs -r:myserver netdom join myserver /domain:testdomain /userd:johns /passwordd:$%fgh789
winrs -r:myserver -ad -u:administrator -p:$%fgh7 dir \\anotherserver\share
PS C:\> winrs.exe -r:wds
The filename, directory name, or volume label syntax is incorrect.

PS C:\> winrs.exe -r:wds hostname
The filename, directory name, or volume label syntax is incorrect.

PS C:\> winrs.exe -r:wds
```

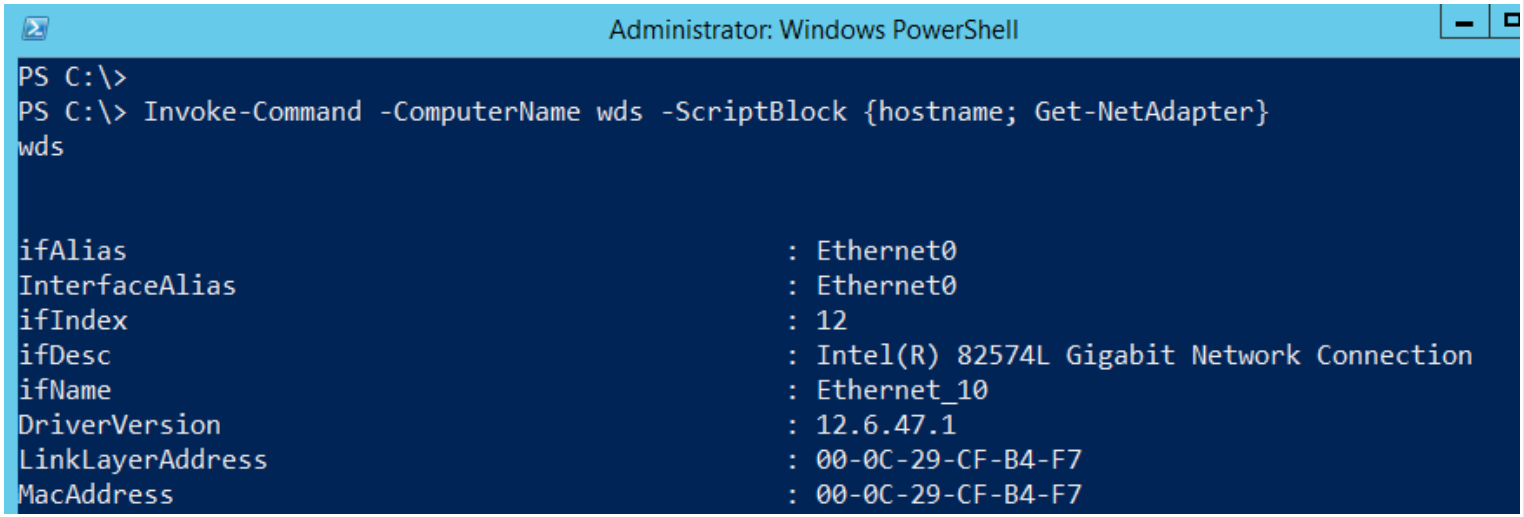
DEMO::

1:1 Remoting



1:many Remoting

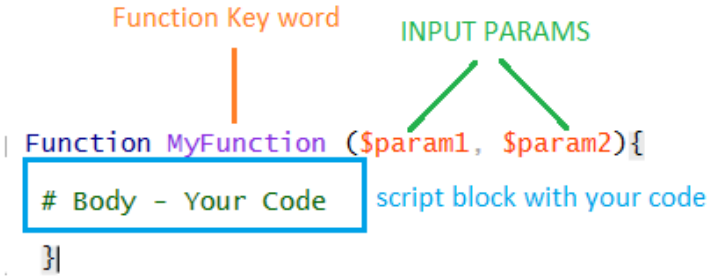
DEMO::



PS Functions

Špecializované IKT systémy Windows

Simple Function



Function is script code for which we can assign name

Function declaration via key word FUNCTION

Syntax

```
Function [SCOPE:]MyFunction ([Type]$param1, [Type]$param2)
{
 # Body - Your Code
 }
```

DEMO::

```powershell
#Simple function Example1
Function Get-Calc ([int]$a,[int]$b) {

    Clear-Host

    Write-Host "`t`tSIMPLE CALC"
    Write-Host "`t`t###########"
    Write-Host "`n"

    $a = Read-Host -Prompt "Please enter first number"
    $b = Read-Host -Prompt "Please enter second number"

    $out = $a * $b

    Write-Host "`t RESULT: $a * $b = $out" -ForegroundColor Cyan
}
```

```powershell
#Simple function Example2
function Get-LargFile($Path, $MinSize){

  Clear-Host

  Write-Host "`nTHis files in path $path are larger then $MinSize" -Foreground Yellow

  $files = Get-ChildItem -path $path | where-Object {$_.length -gt $minsize} |
  Select-Object LastWriteTime,@{n="Size{MB]";e={$_.Length / 1Mb -as [int]}},Name

  Write-Output $files

}
```

Advanced function

- Accept parameter with validation
- Run script block via one command
- Use Verb-Noun name format
- WhatIf and Confirm support
- **Syntax**

```
Function Verb-Noun{
    [cmdletbinding()]
    Param ()
    Begin{Script block}
    Process {script block}
    End {script block}
}
```

Advanced function

PARAM BLOCK

```
Param(
    # Param1 help description
    [Parameter(Mandatory=$true,
               ValueFromPipeline=$true,
               ValueFromPipelineByPropertyName=$true,
               Position=0)]
    [ValidatePattern("[a-z]*")]
    [ValidateLength(0,15)]
    [ValidateRange(0,5)]
    [ValidateCount(0,5)]
    [ValidateScript({$true})]
    [ValidateSet("aaa", "bbb", "ccc")]
    [Alias("p1")]
    [DataType[]]$ParamName = 'DefaultValue'
```

about_Functions_Advanced_Parameters

Advanced function

BEGIN BLOCK

Optional block

Contains code, which will be executed only once per function instance at the BEGINNING

Typical example of usage:

- Information about start of execution
- Get user inputs
- Param declaration
- Test INPUT/OUTPUT path
- Name convention or regex definition
- Test function requirements

about_Functions_Advanced_Parameters

Advanced function

PROCESS BLOCK

- Mandatory part which contains script which can be executed several times or not at all
- Key part of function definition
- Can contains ANY supported PowerShell constructs and elements

about_Functions_Advanced_Parameters

Advanced function

END BLOCK

Optional block

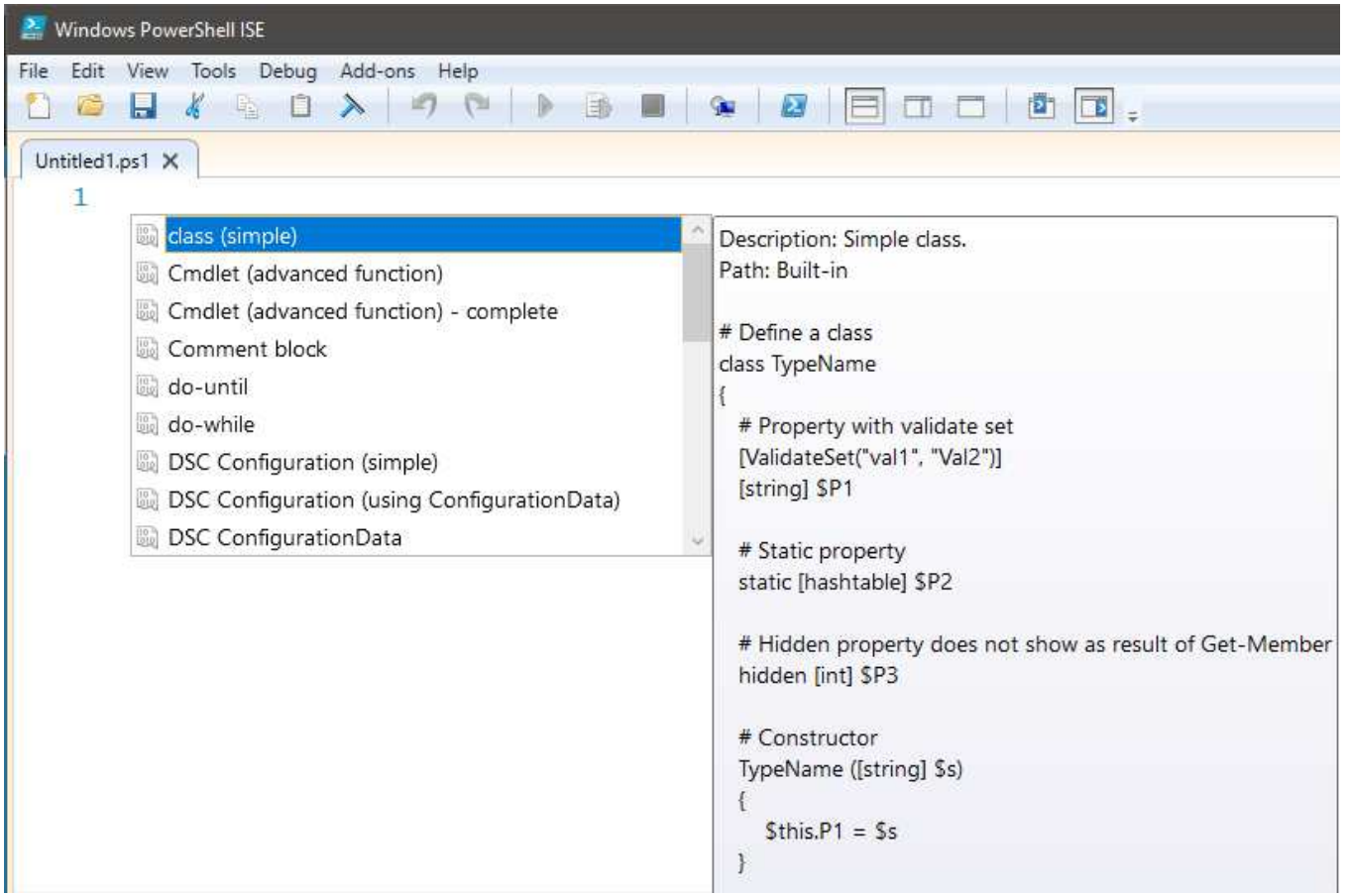
Contains code, which will be executed only once per function instance at the END

Typical example of usage:

- Information about end of funtion
- Information about errors, warnings

about_Functions_Advanced_Parameters

Advanced function



about_Functions_Advanced_Parameters

Advanced function - Help

Comment based help

```
<#
.Synopsis
   Short description
.DESCRIPTION
   Long description
.EXAMPLE
   Example of how to use this cmdlet
.EXAMPLE
   Another example of how to use this cmdlet
.INPUTS
   Inputs to this cmdlet (if any)
.OUTPUTS
   Output from this cmdlet (if any)
.NOTES
   General notes
.COMPONENT
   The component this cmdlet belongs to
.ROLE
   The role this cmdlet belongs to
.FUNCTIONALITY
   The functionality that best describes this cmdlet
#>
```

```
#.Synopsis
#   Short description
#.DESCRIPTION
#   Long description
#.EXAMPLE
#   Example of how to use this cmdlet
#.EXAMPLE
#   Another example of how to use this cmdlet
#.INPUTS
#   Inputs to this cmdlet (if any)
#.OUTPUTS
#   Output from this cmdlet (if any)
#.NOTES
#   General notes
#.COMPONENT
#   The component this cmdlet belongs to
#.ROLE
#   The role this cmdlet belongs to
#.FUNCTIONALITY
#   The functionality that best describes this cmdlet
```

PS Scripts

Špecializované IKT systémy Windows

Scripts

Scripts Execution policy

- One of security mechanism
- Default value is Restricted
- Can be handle with Get-Executionpolicy and Set-Executionpolicy

- **Policy**
  - **Restricted** - Prevents running of all script files, including formatting and configuration files (.ps1xml), module script files (.psm1), and Windows PowerShell
  - **AllSigned -** Requires that all scripts and configuration files be signed by a trusted **publisher,** including scripts that you write on the local computer
  - RemoteSigned **-** Requires a digital signature from a trusted publisher on scripts and configuration files that are downloaded from the Internet but not locally created
  - **Unrestricted** – all scripts will run only display warning for script downloaded from net
  - **Bypass -** Nothing is blocked and there are no warnings or prompts.

Scripts

Other security mechanism

- Default association ps1 files
- Full path required to run script from shell
- Add signature to script

Best practices

- RemoteSigned policy
- Do not change default association ps1 files
- Script properly documented
- Downloaded script fully understand and test before
- Use confirmation methods (shouldprocess, shouldcontinue)

Scripts

How to run PowerShell script

- **.\script.ps1**
- **<Full path>\script.ps1**

- **cmd powershell <full path>\script.ps1**
- Cmd powershell –command "&{gc script.ps1}"

PowerShell.exe switches:

- -noLogo
- -noProfile
- -noInteractive
- -noExit

DEMO::

PS Error handling

Špecializované IKT systémy Windows

Error handling

- Error is msg display when something go wrong
- Errors can be terminating and non terminating
- By default are stored into variable $Error

Default is 256 exception define by $MaximumErrorCount

- ErrorActionPeference control behavior of error
  - Continue(3) is default behavior
  - SilentlyContinue(0)
  - Stop(1)
  - Inquire(3)
  - ignore(4)

Error variable

$Error[o]

$?

Return true or false based on last command8/script execution

- **$LastExitCode**

- **$ErrorView**

DEMO::

```
Windows PowerShell
PS C:\>
PS C:\> $Error
PS C:\> Get-Service -Name 'Not exist'
Get-Service : Cannot find any service with service name 'Not exist'.
At line:1 char:1
+ Get-Service -Name 'Not exist'
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (Not exist:String) [Get-Service], ServiceCommandException
    + FullyQualifiedErrorId : NoServiceFoundForGivenName,Microsoft.PowerShell.Commands.GetServiceCommand

PS C:\> $?
False
PS C:\> $Error[0]
Get-Service : Cannot find any service with service name 'Not exist'.
At line:1 char:1
+ Get-Service -Name 'Not exist'
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (Not exist:String) [Get-Service], ServiceCommandException
    + FullyQualifiedErrorId : NoServiceFoundForGivenName,Microsoft.PowerShell.Commands.GetServiceCommand
```

DEMO::

```
Windows PowerShell
PS C:\>
PS C:\> $LASTEXITCODE = $null
PS C:\>
PS C:\> Get-Service -Name 'Not exist'
Get-Service : Cannot find any service with service name 'Not exist'.
At line:1 char:1
+ Get-Service -Name 'Not exist'
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (Not exist:String) [Get-Service], ServiceCommandException
    + FullyQualifiedErrorId : NoServiceFoundForGivenName,Microsoft.PowerShell.Commands.GetServiceCommand

PS C:\> $LASTEXITCODE
PS C:\>
PS C:\> sc.exe query notexist
[SC] EnumQueryServicesStatus:OpenService FAILED 1060:

The specified service does not exist as an installed service.

PS C:\> $LASTEXITCODE
1060
PS C:\>
PS C:\> net start notexist; if ($LastExitCode  -ne 0) {Write-host "ERROR" -ForegroundColor Red}
The service name is invalid.

More help is available by typing NET HELPMSG 2185.

ERROR
PS C:\>
```

TRY CATCH FINALLY

- They always come as pair
- The try-block marks the area of your code where you want to handle errors.
- The catch-block defines the code that is executed when an error in the try-block occurs.
- You can use multiple catch blocks where each catch represent different type of exception
- Required $ErrorActionPreference= "Stop" (-ea "Stop")

About_Try_Catch_Finally

DEMO::

```
Windows PowerShell
PS C:\>
PS C:\> #TRY-CATCH
PS C:\> $computers = 'localhost','NotExist','127.0.0.1','srv100'
PS C:\> ForEach ($comp in $computers) {
>>     try {
>>       $ErrorActionPreference = 'Stop'
>>       $currentcomputer = $_
>>       Get-WmiObject -class Win32_OperatingSystem -computername $comp -ErrorAction Stop |
>>         Select-Object __Server, Caption
>>     }
>>     catch {
>>       $global:FailedComputers = @()
>>       $FailedComputers += "Failed to access computer $Comp"
>>       Write-Warning ('Failed to access "{0}" : {1} in "{2}"' -f $comp, `
>>       $_.Exception.Message, $_.InvocationInfo.ScriptName)
>>     }
>>   }

WARNING: Failed to access "NotExist" : The RPC server is unavailable. (Exception from HRESULT: 0x800706BA) in ""
__SERVER Caption
-------- -------
MASTER   Microsoft Windows 10 Pro
MASTER   Microsoft Windows 10 Pro
WARNING: Failed to access "srv100" : The RPC server is unavailable. (Exception from HRESULT: 0x800706BA) in ""


PS C:\>
```

NEXT

::Webs

https://devblogs.microsoft.com/powershell/

https://powershell.org/

https://www.planetpowershell.com/

powershellmagazine.com