Module 1

Scripts
Execution Policy
Script Logic
Parameters
Functions

DIST IT SERVICES & TRAINING

1

Scripts

Requirements to start a script

> A **script** is a plain text file that contains one or more Windows PowerShell commands. Windows PowerShell scripts have a .ps1 file name extension. (get-help about_scripts)

- **Execution Policy**
  - Restricted
    - Prevents all scripts (*.ps1, *.psm1, *.ps1xml) from running
    - Default in Win8, Win2012, Win8.1
  - Unrestricted
    - All scripts run, downloaded scripts bring a warning
  - RemoteSigned
    - All local written scripts run, downloaded scripts must be signed or unblocked
    - Default in Win2012R2
  - AllSigned
    - All scripts run, if they are signed, even local scripts
    - Scripts from untrusted publishers will not start, from unclassified publishers bring prompt

DIST IT SERVICES & TRAINING

2

Scripts

Requirements to start a script

- **Execution Policy**
  - Bypass
    - Nothing is blocked and no warnings
  - Undefined
    - no execution policy set in a scope

```
No Accumulation !
```

- **Scope of Execution Policy and Precedence**
  - MachinePolicy
  - UserPolicy
  - Process (current powershell.exe, saved in $env:PSExecutionPolicyPreference)
  - CurrentUser (current user, saved in Registry)
  - LocalMachine (~, saved in Registry)
- **Change Execution Policy**
  - GPO
  - cmdlet

```
Set-ExecutionPolicy -ExecutionPolicy Restricted -Scope CurrentUser
Powershell.exe -ExecutionPolicy Unrestricted

Get-ExecutionPolicy
Get-ExecutionPolicy -List
```


3

Script Logic


```
If (Condition)
{
   # code, if $true
}
Else
{
   # code, if $false
}
```

```
If (Condition)
{
   # code, if $true
}
Elseif (Condition)
{
   # code, if $true
}
Elseif (Condition)
{
   # code, if $true
}
Else
{
   # code, if $false
}
```

```
Switch ($x)
{
   '1'
   { # code, if $x is 1 }
   '2'
   { # code, if $x is 2 }

   {$_ -in 'a', 'b', 'c'}
   { # code, if $x is a, b or c }

   {$_ -in 3..5}
   { # code, if $x is 3, 4 or 5 }

   default
   { # code, if no case applies }
}

[-regex|-wildcard|-exact][-casesensitive]

#HINT: Use break !!!
```

DIST IT SERVICES & TRAINING

4

Script Logic


```
For (Start; Condition; Step)
{
   # repeating code to execute
}
```

```
ForEach ($temp in $collection|$array)
{
   # code to execute for each element
   # in the collection/array
}
```

```
Do
{
   # repeating code to execute
   # if condition is true
} While (Condition)


Do
{
   # repeating code to execute
   # if condition is false
} Until (Condition)


While (Condition)
{
   # repeating code to execute
   # if condition is true
}
```


5

Parameter of Scripts

How to pass parameters to scripts

- **$args[]**
  - Intrinsic variable/array
  - First parameter in $args[0], second in $args[1], …
  - Characteristics: no named parameter, no validation, no additional arguments, …
- **Param - Section**
  - First section in script
  - All parameters are enclosed in parenthesis (…)
  - Definitions separated by comma
  - Characteristics: Naming, TypeDefinition, validation, etc. possible

6

Parameter of Scripts

Arguments of parameters

- **Arguments declared in Param section**
  - [Parameter()][TypeDef]$variable
  - Parameter() can have more than one attribute
  - Attribute has a name and value (Attribute=value)
  - Attributes are separated by comma
- **Parameter Attributes**
  - Mandatory = $true | $false
  - HelpMessage="…"
    - with Mandatory attribute
  - Position=x
  - ParameterSetName = "SetName"
  - ValueFromPipeline = $true | $false
  - ValueFromPipelineByPropertyName = $true | $false
  - ValueFromRemainingArguments = $true | $false

DIST IT SERVICES & TRAINING

7

Parameter of Scripts

Other attributes and validation

- **Other Attributes**
  - Alias Attribute: [Alias("othername")]
- **Validation**
  - [AllowNull()]
  - [AllowEmptyString()]
  - [AllowEmptyCollection()]
  - [ValidateCount(2,5)]
  - [ValidateLength(7,10)]
  - [ValidatePattern("…")]
  - [ValidateRange(0,5)]
  - [ValidateSet("Sun","Moon","Earth")]
  - [ValidateScript({ $_ -le 9999 })]
  - [ValidateNotNull()]
  - [ValidateNotNullOrEmpty()]

DIST IT SERVICES & TRAINING

8

Parameter of Scripts

Examples

```
Param
(
   [Int]$value1,
   [Parameter(Mandatory=$true)][Int]$value2
)
```

```
Param
(
   [Parameter(Position=1)][String]$value3,
   [Parameter(Mandatory=$true, Position=0)][Char]$value4
)
```

```
Param
(
   [parameter(Mandatory=$true, ParameterSetName="Computer")][String[]]$ComputerName,
   [parameter(Mandatory=$true, ParameterSetName="User")][String[]]$UserName,
   [parameter(Mandatory=$false)][Switch]$Summary
)
```

```
.\Script.ps1 -Summary
.\Script.ps1 -Summary:$false
.\Script.ps1 -Summary:$true
```


Scripts and Pipeline

How to use piped values in a script

- **Solution 1**
  - Use intrinsic $input variable
- **Solution 2**
  - Use sections BEGIN, PROCESS, END
  - In PROCESS use $_
  - Similar to ForEach
- **Solution 3**
  - Use *ValueFromPipeline* in param section

```
BEGIN
{
   # this code is executed once,
   # before the PROCESS section.
}

PROCESS
{
   # this code is executed as often as
   # elements come through the pipeline
}

END
{
   # this code is executed once,
   # after the PROCESS section.
}
```

Function

How to write and use functions

```
A function is a list of Windows PowerShell statement
that has a name that you assign. (get-help about_functions)
```

- **Used for**
  - Unregularly repeating code in script (reuse)
  - Separating code (maintenance)
  - Modules
- **Enclosed in 'function functionname { … }'**
  - Can have all section a script has (including Help)
  - 'Filter' also possible, not common
- **Started**
  - by functionname with/without parameters
  - by functionname in a pipeline
- **Return value**
  - Send to the calling scope with return

DIST IT SERVICES & TRAINING

Do You Have Any Questions?

DIST IT SERVICES & TRAINING