# Windows PowerShell

## Introduction to PowerShell

### What is PowerShell?

* Task Automation Framework
* Command-Line Shell **and** Scripting Language for Windows
* Based on the .NET Framework
* Inspired by UNIX shells and scripting languages such as Perl and Python



### Terminology: Shell- vs. Scripting-Languages

#### (Command-Line) Shell

“A shell is a piece of software that lets you access the functionality provided by the operating system.”  
− Bruce Pyette: PowerShell in Action

Common Features:

* Read-Evaluate-Print loop (REP)
* Aliases/Shortcuts for long command names
* Wildcard matching, so you don’t have to type the full names of everything
* Call external programs
* Command History

UNIX Shells

* Bourne-Again Shell (bash)
* Korn Shell (ksh)

Windows Shells

* Command.com
* Cmd.exe



* PowerShell
* Cygwin



Others

* Chrome Console



* Firebug Command Line



#### Scripting Languages

* Scripting languages typically provide more sophisticated features for debugging your scripts and they provide mechanisms for developing larger scripts by letting you break a script into components or modules
* Scripting Language syntax is oriented more towards writing an application than toward interactively issuing commands

### Why PowerShell?

* “When comparing the command-line manageability of a Windows system to a UNIX system, Windows was found to be limited”
* Windows command line is an inferior technology which cannot compete with the known UNIX Shells
* Windows is known for its Management UIs but lacked an automation framework 🡪 “Point and click does not scale”
  + Not across different people
  + Not over time
  + Not across different server environments
  + Not across multiple servers
  + And it is hard to control and to document

### Version History

* Designed from scratch in 2005
* Version 1.0 was released in 2006 for Windows XP, Windows Server 2003 and Windows Vista. Only optional for Windows Server 2008.
* Version 2.0 is the current version and was released in 2009 for Windows 7, Winder Server 2008 R2

### The Future of PowerShell

* PowerShell Version 3.0 is currently a CTP and will be released with Windows Server 8
* The management of Windows Server 8 (Core) will be entirely based on PowerShell
* For the release of Windows Server 8 the number of built-in PowerShell commands (cmdlets) will be increased from 200 to 2300

Links

* Windows Server 8 Developer Preview and PowerShell v3 First Look - <http://www.mikepfeiffer.net/2011/09/windows-server-8-developer-preview-and-powershell-v3-first-look/>
* Microsoft verabschiedet sich vom GUI - <http://www.golem.de/1109/86435.html>
* Powershell dominiert die Server-Verwaltung ab Windows Server 8 - <http://www.nt4admins.de/themen/verwaltungs-tools/artikel/powershell-dominiert-die-server-verwaltung-ab-windows-server-8.html>

### What is special about Windows PowerShell?

* PowerShell is a new class of object-based shell language ­− most other shell-languages are string-based
  + PowerShell preserves the structure of the Windows data types by using the .NET object model 🡪 No tedious string parsing required
* Command-Line Shell AND Scripting Language in one product – PowerShells goal is to be both a good scripting language and a good interactive shell
* PowerShell leverages the full functionality of the .NET framework
  + You can use all the objects and functions that you know and love from the .NET Framework

## Getting Started with PowerShell

### Online Resources about Windows PowerShell

* A list of PowerShell Cmdlets − <http://technet.microsoft.com/en-us/library/hh848794.aspx>
* Scott Hanselman’s PowerShell Posts − <http://www.hanselman.com/blog/archives.aspx#PowerShell>
* PowerGUI: An alternative to the PowerShell ISE − [http://powergui.org](http://powergui.org/)
* Web Administration Cmdlets for Windows PowerShell − <http://technet.microsoft.com/en-us/library/ee790599(WS.10).aspx>
* Microsoft Technet Script Resources − <http://gallery.technet.microsoft.com/scriptcenter/>
* PowerShell Community Extensions − <http://pscx.codeplex.com/>
* Official PowerShell Homepage − <http://technet.microsoft.com/en-us/scriptcenter/dd742419.aspx>
* My Personal PowerShell Bookmark Collection − <http://delicious.com/andyk7/powershell>

### Installation & Startup

#### PowerShell Command-Line

If you are running a recent version of Windows you will have PowerShell already installed:

Start > Accessories > Windows PowerShell



#### Windows PowerShell Integrated Scripting Environment (ISE)

If don’t have the PowerShell debugger installed you can add it via the Windows Features dialog:

Server Management > Features > Add Feature > “Windows Integrated Scripting Environment (ISE)”





#### Installation Directory

The PowerShell executables are located in the folder:

%windir%\System32\WindowsPowerShell\v1.0



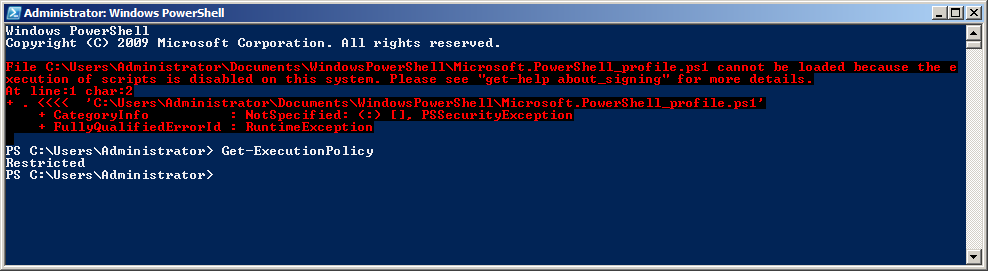
#### PowerShell Security Settings

“The execution policy is part of the security strategy of Windows PowerShell. It determines whether you can load configuration files (including your Windows PowerShell profile) and run scripts, and it determines which scripts, if any, must be digitally signed before they will run.”

By default, PowerShell does not allow the execution of unsigned/untrusted scripts, including your own PowerShell Profile:

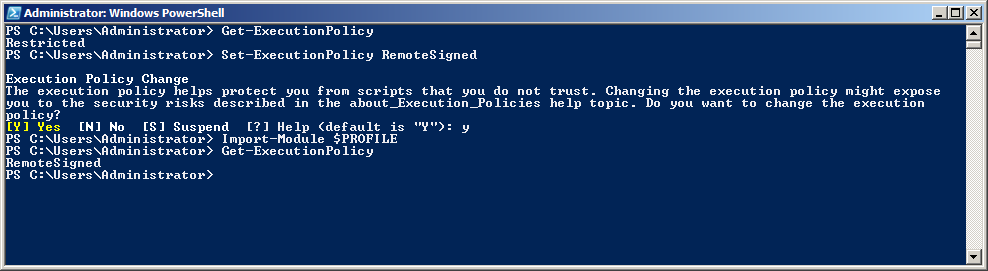
> Get-ExecutionPolicy

Restricted



In order to be able to run (unsigned) PowerShell scripts and configuration files you need to set the ExecutionPolicy to “RemoteSigned”:

> Set-ExecutionPolicy RemoteSigned



Links

* Set-ExecutionPolicy − <http://technet.microsoft.com/en-US/library/dd347628.aspx>

### Configuring Mercurial

Start PowerShell

> notepad $Home\Mercurial.ini

Enter the following text and then save the changes:

[ui]  
username = Your Name <Your.Name@Domain.Com>  
verbose = True

[Extensions]

purge =



### Fetching the workshop material from my Mercurial repository

In order to get a copy of the PowerShell workshop material please clone my mercurial repository and create a branch that is named after you.

Start PowerShell

> mkdir $Home\Desktop\dev | Set-Location  
> hg clone <http://andyk.dyndns-server.com:8000> PowerShell-Workshop  
> Set-Location PowerShell-Workshop  
> hg branch “YourName”  
> hg push --new-branch

## Core Elements of PowerShell

PowerShell is case insensitive by default

## Lessons

### Installing the PowerShell Community Extensions (PSCX)

The PowerShell Community extensions (PSCX) are a PowerShell module that adds a lot of useful cmdlets to PowerShell:

* **Edit-File**: Opens the specified text file in a text editor
* **Show-Tree**: Shows the specified path as a tree
* **Out-Speech**: Outputs text as spoken words
* **Enable-OpenPowerShellHere**: Creates the registry entries required to create Windows Explorer context menu "Open PowerShell Here" for both Directories and Drives
* **Format-Xml**: Pretty print XML
* **Write-Zip**: Creates an ZIP Archive from the pipeline input
* **New-Junction**: Creates an NTFS directory junction link (just like mklink.exe)
* **Out-Clipboard**: Formats text via Out-String before placing in clipboard. Can also place string in clipboard as a file.
* …

Instructions

1. Create a PowerShell Profile if you don’t have one yet

if ((Test-Path $Profile) -eq $false)

{

New-Item $Profile -Type file -Force | Out-Null

}

explorer (Split-Path $profile -parent)

1. Extract the PSCX archive (“Pscx-2.0.0.1.zip”) into your PowerShell Profile Directory

C:\Users\Administrator\Documents\WindowsPowerShell\Modules

1. Add an import statement to your PowerShell profile

Import-Module Pscx -arg "$(Split-Path $profile -parent)\Modules\Pscx\Pscx.UserPreferences.ps1"

1. Restart PowerShell

Links

* PowerShell Community Extensions (PSCX) − <http://pscx.codeplex.com/documentation>

### Reading and Writing XML with PowerShell

PowerShell supports XML documents as a primitive data type.

Sample XML

<Settings>

<Entry id="1">Value 1</Entry>

<Entry id="2">Value 2</Entry>

<Entry id="3">Value 3</Entry>

</Settings>

Reading XML

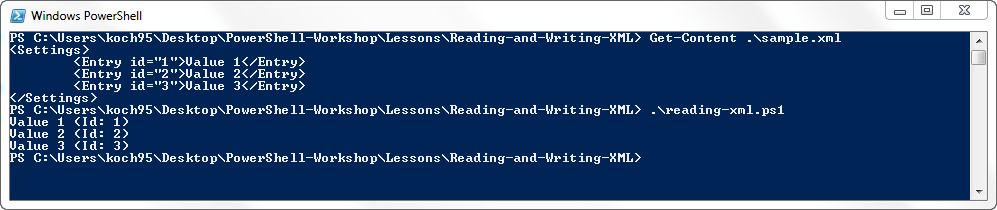
[xml] $xml = Get-Content sample.xml

foreach ($setting in $xml.Settings.Entry)

{

Write-Host "$($setting.psbase.InnerText) (Id: $($setting.Id))"

}



Writing XML

# Read XML

[xml] $xml = Get-Content sample.xml

# Create new node

$newEntry = $xml.CreateElement("Entry")

$newEntry.psbase.InnerText = "Value 4"

# Create new node attribute

$idAttribute = $xml.CreateAttribute("id")

$idAttribute.psbase.Value = "4"

# Assign new attribute to new node

$newEntry.SetAttributeNode($idAttribute)

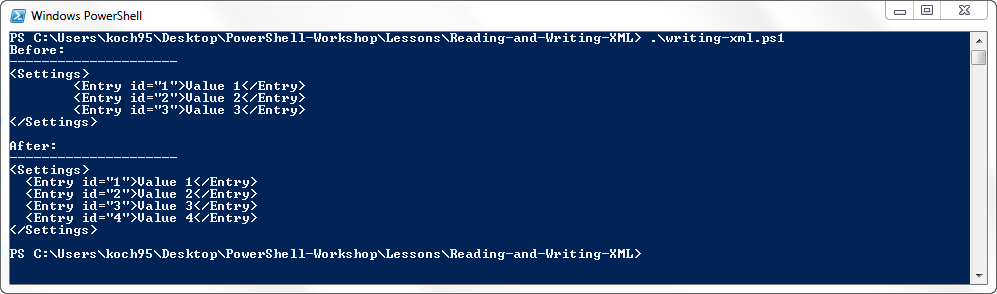
# Append new node

$xml.Settings.AppendChild($newEntry)

# Save XML (Note: Use absolute paths for XML.Save())

$targetFile = Join-Path "$(Get-Location)" "sample-extended.xml"

$xml.save($targetFile)



### Signing PowerShell Scripts

### Automating Visual Studio with PowerConsole

### Extending PowerShell with Clipboard Access (Lesson: Profile-Extension-Clipboard)

#### What it the PowerShell Profile

The Windows PowerShell profile is simply a script file that runs when you start Windows PowerShell. All functions defined in this script will be automatically available in your new PowerShell session.

The location of your PowerShell profile is stored in the $PROFILE variable:

> $profile

C:\Users\Administrator\Documents\WindowsPowerShell\Microsoft.PowerShell\_profile.ps1



You can edit your profile script by entering:

> notepad $profile

In case you don’t have a profile script you will get a message like this:



In order to create the profile script and open it with notepad type this command:

> notepad (New-Item $PROFILE -Type file -force)



Everything that you add to this script will be executed when you launch a new instance of PowerShell.

In order to open the directory of the profile script you can enter:

> explorer (Get-Item $PROFILE).Directory



Now you can add the functions for reading and writing to and from the Windows Clipboard to your personal PowerShell profile:

new-alias Out-Clipboard $env:SystemRoot\system32\clip.exe

function Get-ClipboardText()

{

Add-Type -AssemblyName System.Windows.Forms

$tb = New-Object System.Windows.Forms.TextBox

$tb.Multiline = $true

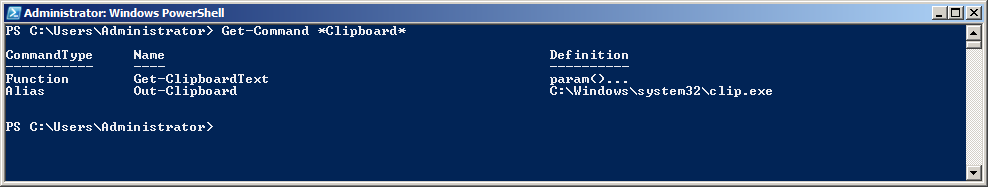
$tb.Paste()

$tb.Text

}

**Start a new instance of PowerShell** and verify that your functions have been applied correctly:

> Get-Command \*Clipboard\*



And now you can take your new functions “Get-ClipboardText” and “Out-Clipboard” out for a spin:

> Get-Content $Profile | Out-Clipboard

> dir | Out-Clipboard

> Get-Process | Out-Clipboard

Links

* The Windows PowerShell Profile − <http://technet.microsoft.com/en-us/library/ee692764.aspx>
* Copy and Paste with Clipboard from PowerShell − <http://brianreiter.org/2010/09/03/copy-and-paste-with-clipboard-from-powershell/>
* Using the New-Alias Cmdlet − <http://technet.microsoft.com/en-us/library/ee176913.aspx>
* Using the Add-Type Cmdlet − <http://technet.microsoft.com/en-US/library/dd315241.aspx>
* Using the New-Object Cmdled − <http://technet.microsoft.com/en-US/library/dd315334.aspx>
* The .NET TextBox Class − <http://msdn.microsoft.com/en-us/library/system.windows.forms.textbox.aspx>