# 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

* Windows PowerShell Documentation on the MSDN − <http://msdn.microsoft.com/en-us/library/windows/desktop/dd835506(v=vs.85).aspx>
* 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



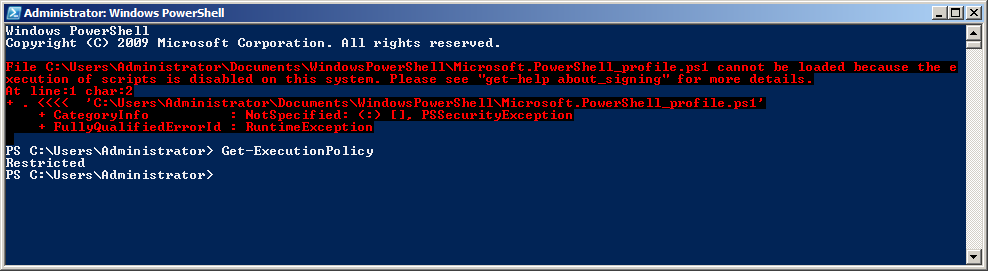
#### PowerShell Security Settings

“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:

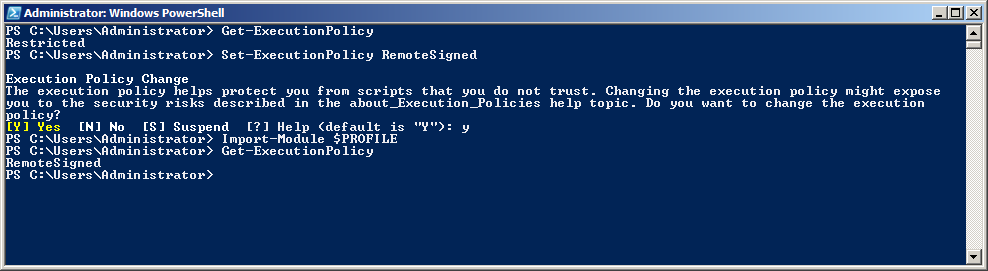
> Get-ExecutionPolicy

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

### Temporary Outline

* Command Interpreter
* Syntax
  + Command Syntax: <verb>-<noun>
  + Variables start with a $
  + & is the function call operator (Invoke Expression)
  + Parameters start with a dash –
  + Output Redirection is done with a |
  + Lambda Variables are $\_
  + Strings a usually wrapped in double quotes “
  + Multi-Line Strings are done like this @”…”@
  + Escape character is the back tick `
* Basics
  + Command Types
    - Cmdlets
    - PowerShell Scripts
    - PowerShell Functions
    - Executable applications / Standalone programs
  + Objects
  + Pipelining
  + Aliases
  + PowerShell Drives
  + Important Commands
    - Get-Help
    - Get-Member
    - Get-item
    - Get-ChildItem
    - New-Item
    - Remove-item
    - Copy-Item
    - Format-List
    - Format-Table
    - Write-Host
    - Sort
    - Select-Object
    - Where-Object
    - ForEach-Object
* Wildcards
* Regex
* Types
* Operators and expressions
* Flow Control
  + If / Else
  + Switch
* Functions
  + Parameters
  + Switches
  + Streaming
* Scripts
* Errors and Exceptions
* Cmdlets
* Naming Conventions for scripts and modules (ps1, …)
* Usefull (environment) variables
  + $Home
  + $Profile
  + $env:windir

### General Stuff about PowerShell

* PowerShell is case insensitive by default
* PowerShell is object-based − not object-oriented. Everything is an object.
* Provider Based
* Streaming
* You can use all the CMD.exe commands you like
* You don’t have to type the full name of every command. Only as much as is required in order to make it distinct.

### Features of the PowerShell console

The PowerShell console is based on cmd.exe and has basically the same features.

* Tab Completion

Enter a command partially and then hit <Tab> in order to let PowerShell complete the command

* + For commands
  + For wildcards
  + For variables
  + For Properties
  + For Functions
  + PowerShell Providers (Filesystem, IIS, Certificates, Registry, …)

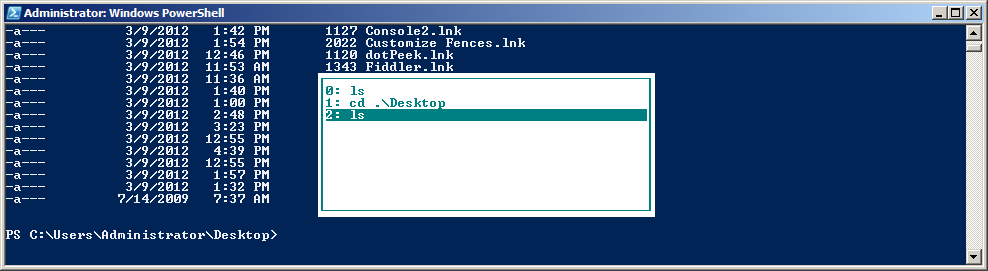
And you can build your own tab-completion modules.

* Evaluation of basic expressions



* F7 Command History

As in cmd.exe the <F7> key pops up a command history window.



### PowerShell Syntax

#### PowerShell Command Syntax

PowerShell commands, called cmdlets [speak: command-lets], use verb\*-noun pairs in order to be self-descriptive.

\* Windows PowerShell uses the term verb to describe a word that implies an action even if that word is not a standard verb in the English language. For example, the term New is a valid Windows PowerShell verb name because it implies an action even though it is not a verb in the English language.

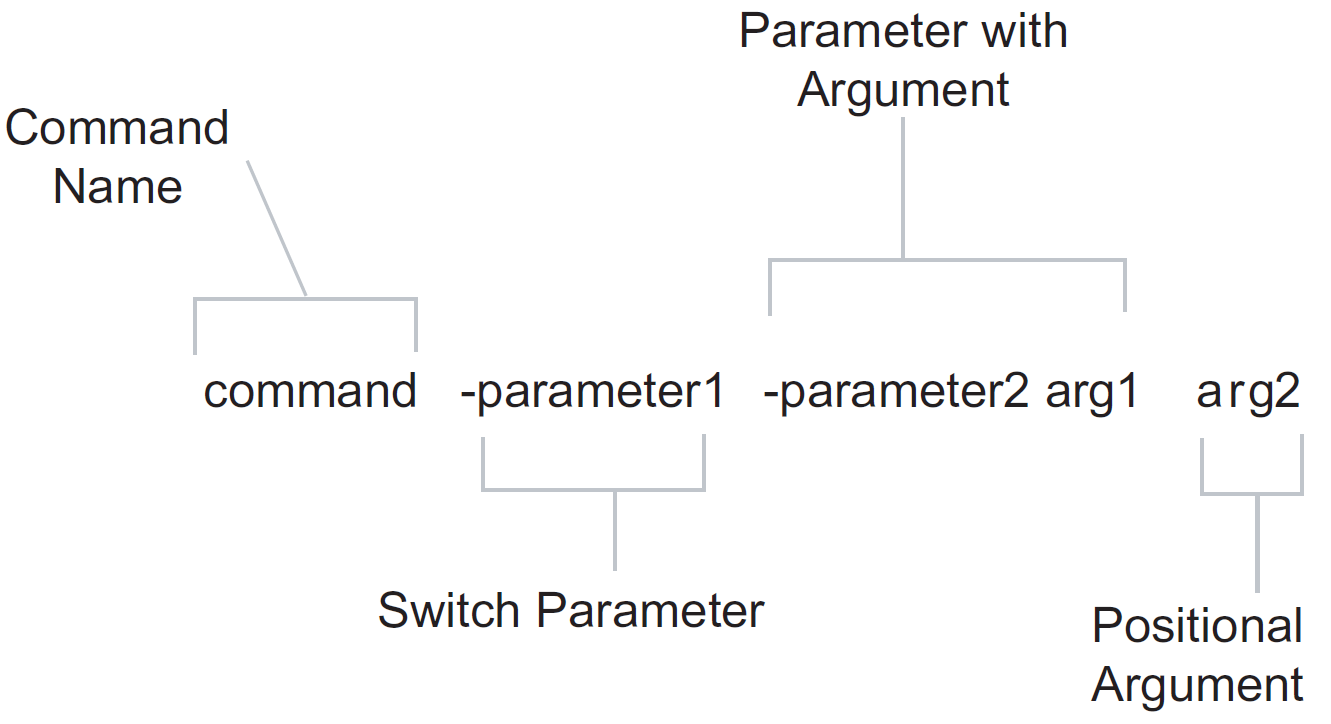
Links:

* Verb Naming Rules − <http://msdn.microsoft.com/en-us/library/windows/desktop/ms714428(v=vs.85).aspx>

#### PowerShell Command Structure

A PowerShell command consists of

* Command Name
* Switch Parameter(s)
* Command Parameter(s)
* And Command Argument(s)



#### Common PowerShell Verbs and their meaning

Here is a list of the most common verbs used for PowerShell commands:

|  |  |  |
| --- | --- | --- |
| Verb | Description | Counter Part |
| Add | Adds a resource to a container, or attaches an item to another item. | Remove |
| Clear | Removes all resources from a container but does not delete the container. |  |
| Close | Changes the state of a resource to make it inaccessible, unavailable, or unusable. |  |
| Copy | Copies a resource to another container. |  |
| Enter | Specifies an action that allows the user to move into a resource. | Exit |
| Exit | Sets the current environment or context to most recently used context. | Enter |
| Find | Looks for an object in a container that is unknown, implied, optional, or specified. |  |
| Format | Arranges objects in a specified form or layout. |  |
| Get | Specifies an action that retrieves a resource. | Set |
| Hide | Makes a resource undetectable. | Show |
| Join | Combines resources into one resource. | Split |
| Lock | Secures a resources. | Unlock |
| New | Creates a resource. |  |
| Open | Changes the state of a resource to make it accessible, available, or usable. | Close |
| Pop | Removes an item from the top of a stack. | Push |
| Push | Adds an item to the top of a stack. | Pop |
| Redo | Resets a resource to the state that was undone. |  |
| Remove | Deletes a resource from a container. | Add |
| Rename | Changes the name of a resource. |  |
| Reset | Sets a resource back to its original state. |  |
| Search | Creates a reference to a resource in a container. |  |
| Select | Locates a resouces in a container. |  |
| Set | Replaces data on an existing resource or creates a resources that contains some data. | Get |
| Show | Makes a resource visible to the user. | Hide |
| Skip | Bypasses one or more resources or points in a sequence. |  |
| Split | Seperates parts of a resource. | Join |
| Step | Moves to the next point or resources in a sequence. |  |
| Switch | Specifies an action that alternates between two resources, such as to change between two locations, responsibilities, or states. |  |
| Undo | Sets a resource to its previous state. |  |
| Unlock | Releases a resource that was locked. | Lock |
| Watch | Continually inspects or monitors a resource for changes. |  |

For more examples please refer to the MSDN documentation (<http://msdn.microsoft.com/en-us/library/windows/desktop/ms714428(v=vs.85).aspx)>

#### Examples for PowerShell Commands

* Get-Help
* Get-Command
* Get-Item
* Get-Member
* Remove-Item
* New-Item
* Move-Item
* Format-List
* Write-Host

#### Variables

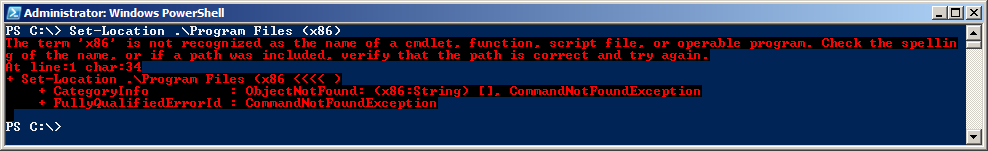
Variables defined by a leading dollar sign ($).

> $someVariable = “Some Content”

#### Quoting

Text in PowerShell is marked by ‘single’- or “double”-quotes. If a command name or argument contains whitespace or other special characters, you must wrap the value in single or double quotes. Otherwise the command-interpreter cannot bind the parameters and arguments correctly.

> Set-Location .\Program Files (x86)



> Set-Location ‘.\Program Files (x86)’



> Set-Location “.\Program Files (x86)”

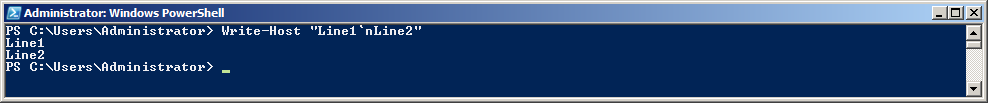


#### Escape Character

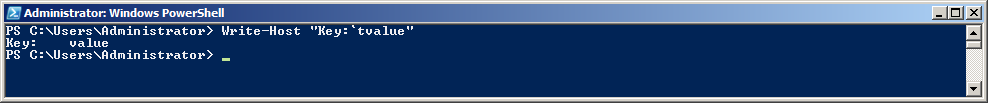
Since PowerShell is a Shell that needs to support Windows-style file systems, it cannot use the usual backslash (\) for escaping characters with a special meaning, but uses the backtick character instead (`):

|  |  |
| --- | --- |
| Escape Sequence | Corresponding Special Character |
| `n | Newline |
| `r | Return |
| `t | Tab |
| `a | Alert |
| `b | <Backspace> |
| `' | Single Quote (') |
| `" | Double Quote (") |
| `0 | Null |
| `` | A single backtick (`) |

> Write-Host "Line1`nLine2"

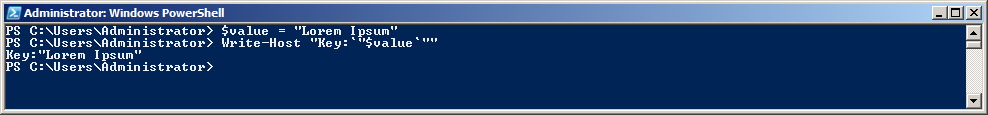


> Write-Host "Key:`tvalue"



> $value = "Lorem Ipsum"

> Write-Host "Key:`"$value`""



#### Statement Termination

PowerShell has two statement termination characters: the semicolon (;) and sometimes the newline (\r\n). If the previous text is a syntactically complete statement, a newline is considered to be a statement termination. If it isn’t complete, the newline is simply treated like any other whitespace.

> $someArrayContainingNumbers = @(1,2,

> 3,

> 4,

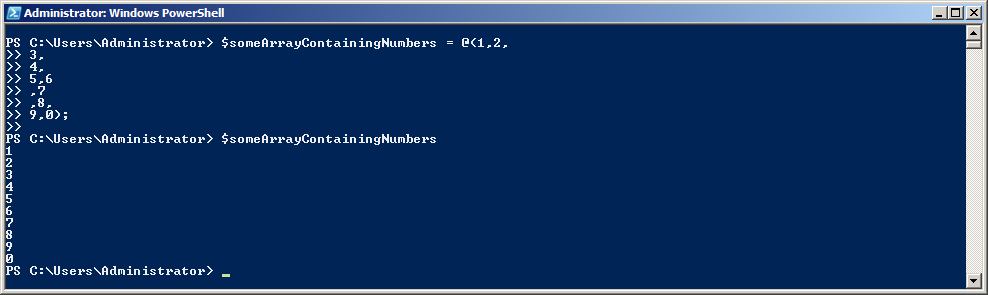
> 5,6

> ,7

> ,8,

> 9,0)

>

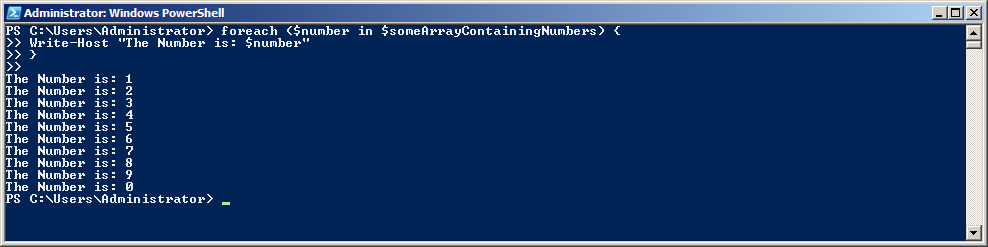


> foreach ($number in $someArrayContainingNumbers) {

> Write-Host "The Number is: $number"

> }

>



Using expressions and semicolons for multi-line statements:

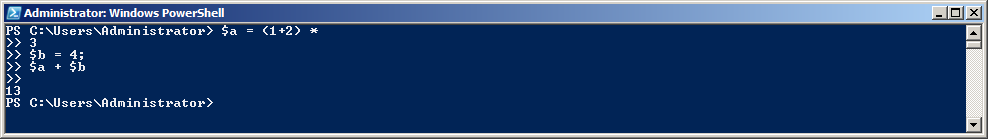
> $a = (1 + 2) \*

> 3

> $b = 4;

> $a + $b

>



Using double quotes to start a multi-line statement:

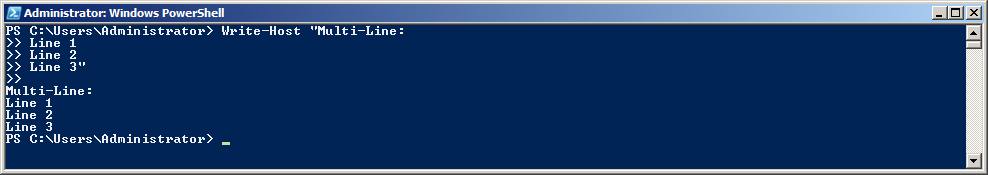
> Write-Host "Multi-Line:

> Line 1

> Line 2

> Line 3"

>



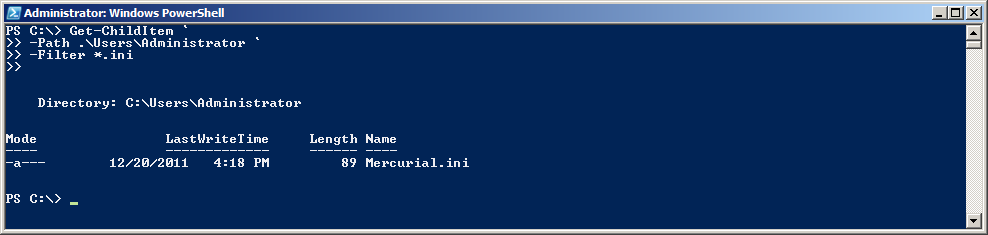
Using the backtick character (`) for bypassing the statement termination:

> Get-ChildItem `

> -Path .\Users\Administrator `

> -Filter \*.ini

>

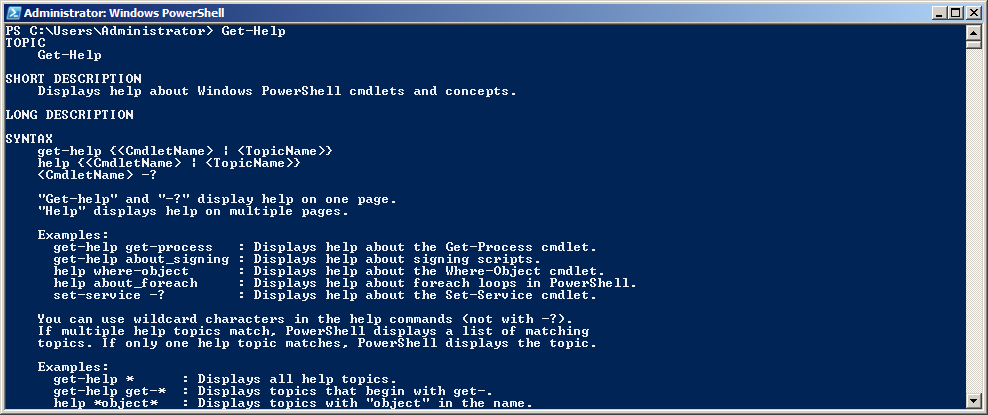


### Using the get-help cmdlet to learn about PowerShell

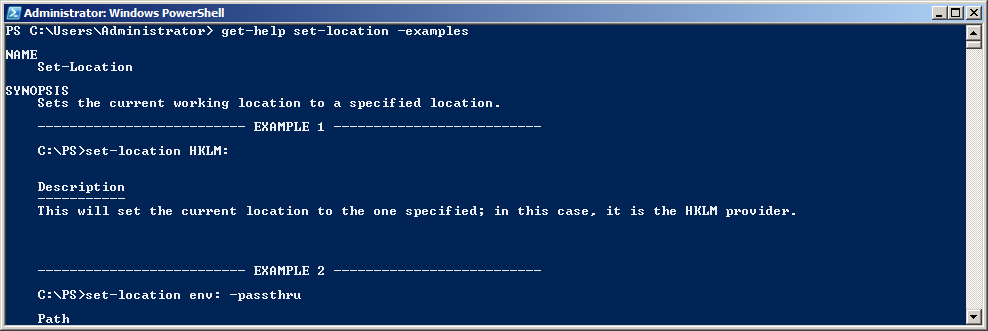
PowerShell has a built-in help command that can provide a lot of information and about the different PowerShell commands and the scripting language itself − just like the Linux man pages.

|  |  |
| --- | --- |
| Command | Description |
| get-help \* | Display all available help topics |
| get-help get-\* | Display all help topics that begin with "get-" |
| get-help \*SomeString\* | Display topics with "SomeString" in the name |
| get-help set-location | Displays help about the "set-location" cmdlet |
| get-help set-location -examples | Displays examples on how to use the "set-location" cmdlet |
| get-help set-location -detailed | Display additional information about a cmdlet, including descriptions of the parameters and examples of using the cmdlet. |
| get-help set-location -full | Display the entire help file for a cmdlet, including technical information about the parameters. |
| get-help about\* | Display conceptual help topics |
| get-help about\_functions | Displays conceptual help about "functions" |
| get-help about\_switch | Display conceptual help about "switch" statements |

> get-help

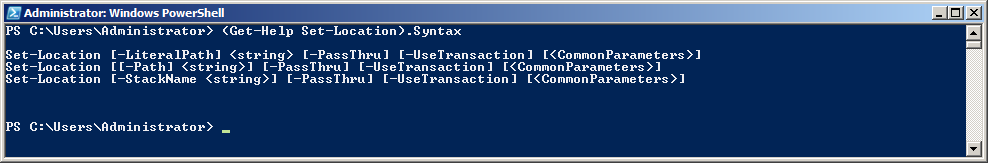


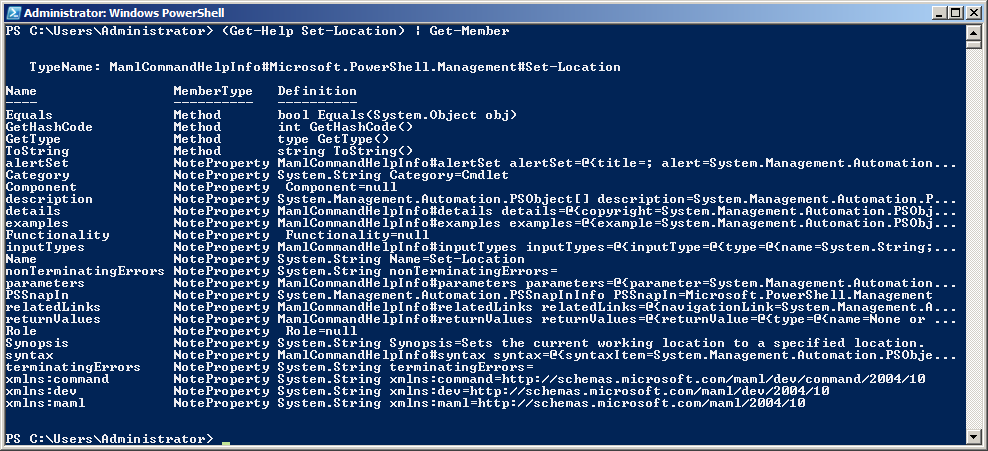
> get-help set-location –examples



Note: The PowerShell Get-Help cmdlets returns objects not just plain text:

> (get-help set-location).Syntax





### Commands

#### Command Types

There are four kinds of command Windows PowerShell can execute:

1. Cmdlets [command-lets]

.NET programs that are designed for PowerShell

1. PowerShell Scripts (\*.ps1)  
   Scripts/Functions that live on the disk
2. PowerShell functions

Scripts/Function that live only in memory

1. Standalone executable programs

All the programs that are available through the classic cmd.exe (e.g. notepad.exe)

#### Common Commands

#### Command Aliases

### PS Drives

* Variables
* Functions
* Environment Variables
* …

### Pipelining

### Output Redirection

>>

### Variables

#### Namespaces

#### Useful environment variables

### Strings, Wildcards and Regular Expressions

#### String Formating

### Flow Control

#### If/Else

#### Switch

#### Loops

### Functions

### Types

#### Using .NET Types

(new-object System.Net.WebClient).DownloadString("http://www.ix.de")

### Scripts

### Streaming

### Errors and Exceptions

## Lessons

### Working with files and folders

### Using the Web Administration Module

* Recycling an App Pool
* Stopping a website
* Enabling Maintenance

### Writing a custom Cmdlet

### Debugging PowerShell Scripts with PowerShell ISE

### Using .NET DLLs

* CacheWarmup Tool
* Commerce Server Connection String Switcher

### 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

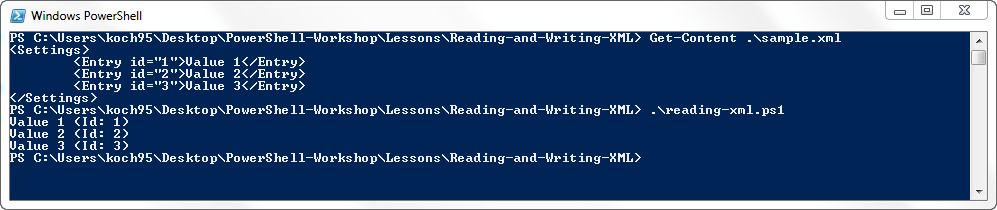
[xml] $xml = Get-Content sample.xml

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)

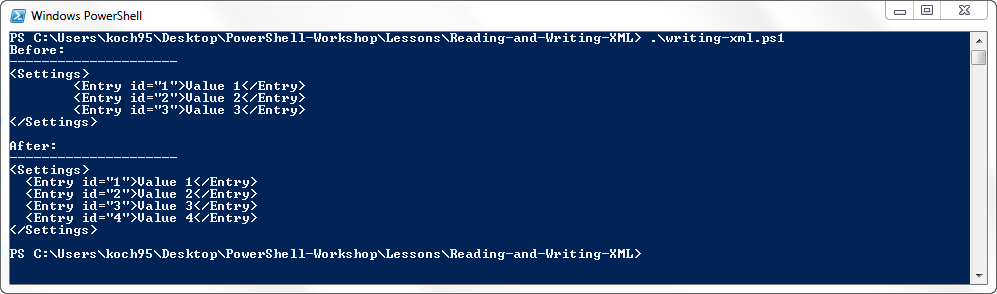
# Append new node

$xml.Settings.AppendChild($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

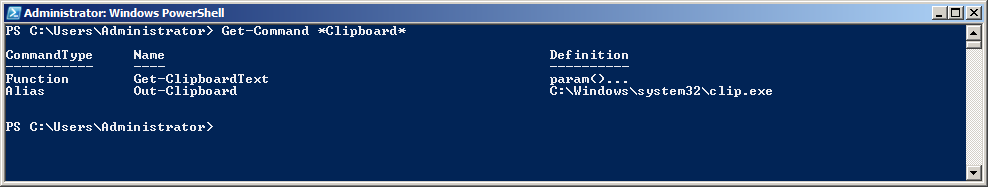
$tb.Paste()

$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>