



## Useful Commands

Update-Help	Downloads and installs newest help files
Get-Help	Displays information about commands and concepts
Get-Command	Gets all commands
Get-Member	Gets the properties and methods of objects
Get-Module	Gets the modules that have been imported or that can be imported into the current session

## Operators

### Assignment Operators

=, +=, -=, \*=, /=, %=, ++, -- Assigns one or more values to a variable

### Comparison Operators

-eq, -ne	Equal, not equal
-gt, -ge	Greater than, greater than or equal to
-lt, -le	Less than, less than or equal to
-replace	changes the specified elements of a value

**"abcde" -replace "bc", "TEST"**

-match, -notmatch	Regular expression match
-like, -notlike	Wildcard matching
-contains, -notcontains	Returns TRUE if the scalar value on its right is contained in the array on its left

**1,2,3,4,5 -contains 3**

-in, -notin	Returns TRUE only when test value exactly matches at least one of the reference values.
-------------	---

**"Windows" -in "Windows", "PowerShell"**

### Bitwise Operators

-band	Bitwise AND
-bor	Bitwise OR (inclusive)
-bxor	Bitwise OR (exclusive)
-bnot	Bitwise NOT
-shl, -shr	Bitwise shift operators. Bit shift left, bit shift right (arithmetic for signed, logical for unsigned values)

### Other Operators

-Split	Splits a string
<b>"abcdefghi" -split "de"</b>	

-join	Joins multiple strings
<b>"abc","def","ghi" -join ","</b>	

..	Range operator
<b>1..10   foreach {\$_ * 5}</b>	

-is, -isnot	Type evaluator (Boolean). Tells whether an object is an instance of a specified .NET Framework type.
-------------	--

**42 -is [int]**

-as	Type convertor. Tries to convert the input object to the specified .NET Framework type.
-----	---

**\$a = 42 -as [String]**

-f	Formats strings by using the format method of string objects
----	--

**1..10 | foreach { "{0:N2}" -f \$\_ }**

[ ]	Cast operator. Converts or limits objects to the specified type
-----	---

**[datetime]\$birthday = "1/10/66"**

,	Comma operator (Array constructor)
.	Dot-sourcing operator runs a script in the current scope
<b>. c:\scripts\sample.ps1</b>	
\$ ( )	Subexpression operator
@ ( )	Array subexpression operator
&	The call operator, also known as the "invocation operator," lets you run commands that are stored in variables and represented by strings.

**\$a = "Get-Process"**

**& \$a**

**\$sb = { Get-Process | Select -First 2 }**

**& \$sb**

### Logical Operators

-and, -or, -xor, -not, !	Connect expressions and statements, allowing you to test for multiple conditions
--------------------------	--

### Redirection Operators

>, >>	The redirection operators enable you to send particular types of output (success, error, warning, verbose, and debug) to files and to the success output stream.
-------	--

Output streams	* All output
	1 Success output
	2 Errors
	3 Warning messages
	4 Verbose output
	5 Debug messages

**# Writes warning output to warning.txt**

**Do-Something 3> warning.txt**

**# Appends verbose.txt with the verbose output**

**Do-Something 4>> verbose.txt**

**# Writes debug output to the output stream**

**Do-Something 5>&1**

**# Redirects all streams to out.txt**

**Do-Something \*> out.txt**



### Arrays

"a", "b", "c"	Array of strings
1,2,3	Array of integers
@()	Empty array
@(2)	Array of one element
1,(2,3),4	Array within array
,"hi"	Array of one element
\$arr[5]	Sixth element of array*
\$arr[2..20]	Returns elements 3 thru 21
\$arr[-1]	Returns the last array element
\$arr[-3..-1]	Displays the last three elements of the array
\$arr[1,4+6..9]	Displays the elements at index positions 1,4, and 6 through 9
@(Get-Process)	Forces the result to an array using the array sub-expression operator
\$arr=1..10	
\$arr[((\$arr.length-1)..0]	Reverses an array
\$arr[1] += 200	Adds to an existing value of the second array item (increases the value of the element)
\$b = \$arr[0,1 + 3..6]	Creates a new array based on selected elements of an existing array
\$z = \$arr + \$b	Combines two arrays into a single array, use the plus operator (+)
*Arrays are zero-based	

### Associative Arrays (Hash tables)

\$hash = @{} @{foo=1; bar='value2'}	Creates empty hash table Creates and initialize a hash table
[ordered]@{a=1; b=2; c=3}	Creates an ordered dictionary
\$hash.key1 = 1	Assigns 1 to key key1

\$hash.key1	Returns value of key1
\$hash["key1"]	Returns value of key1
\$hash.GetEnumerator   sort Key	Sorts a hash table by the Key property
[pscustomobject]@{x=1; y=2}	Creates a custom object

### Comments

# This is a comment because # is the first character of a token

\$a = "#This is not a comment..."  
\$a = "something" # ...but this is.

Write-Host Hello#world

**Block Comments**

<# This is  
A multi-line comment #>

### Object Properties

An object's properties can be referenced directly with the "." operator.

**\$a = Get-Date**  
**\$a | Get-Member -MemberType Property**  
**\$a.Date**  
**\$a.TimeOfDay.Hours**  
**\$a | Get-Member -MemberType Property -Static**

Static properties can be referenced with the "::" operator.

**[DateTime]::Now**

### Methods

Methods can be called on objects.

**\$a = "This is a string"**  
**\$a | Get-Member -MemberType Method**  
**\$a.ToUpper()**

### \$a.Substring(0,3)

### \$a | Get-Member -MemberType Method -Static

Static methods are callable with the "::" operator.

### [DateTime]::IsLeapYear(2012)

### Strings

"This is a string, this \$variable is expanded as is \$(2+2)"  
'This is a string, this \$variable is not expanded'

@  
This is a here-string can contain anything including carriage returns and quotes. Expressions are evaluated: \$(2+2\*5). Note that the end marker of the here-string must be at the beginning of a line!  
"@

@'  
Here-strings with single quotes do not evaluate expressions: \$(2+2\*5)  
'@

### Variables

Format: \${scope:]name or \${anyname} or \${any path}

\$path = "C:\Windows\System32"  
Get-ChildItem \${env:ProgramFiles(x86)}  
\$processes = Get-Process

\$global:a = 1 # visible everywhere  
\$local:a = 1 # defined in this scope and visible to children  
\$private:a = 1 # same as local but invisible to child scopes  
\$script:a = 1 # visible to everything is this script  
# Using scope indicates a local variable in remote commands and with Start-Job  
\$localVar = Read-Host "Directory, please"  
Invoke-Command -ComputerName localhost -ScriptBlock {  
dir \$using:localVar }  
Start-Job { dir \$using:localVar -Recurse}  
\$env:Path += ";D:\Scripts"

# WINDOWS POWERSHELL 4.0 LANGUAGE QUICK REFERENCE

Created by <http://powershellmagazine.com>



PowerShellMagazine

Get-Command -Noun Variable # the Variable Cmdlets  
Get-ChildItem variable: # listing all variables using the variable drive

# strongly-typed variable (can contain only integers)  
[int]\$number=8

# attributes can be used on variables  
[ValidateRange(1,10)][int]\$number = 1  
\$number = 11 #returns an error

# flip variables  
\$a=1;\$b=2; \$a,\$b = \$b,\$a

# multi assignment  
\$a,\$b,\$c = 0  
\$a,\$b,\$c = 'a','b','c'  
\$a,\$b,\$c = 'a b c'.split()

# create read only variable (can be overwritten with -Force)  
Set-Variable -Name ReadOnlyVar -Value 3 -Option ReadOnly

# create Constant variable (cannot be overwritten)  
Set-Variable -Name Pi -Value 3.14 -Option Constant

## Windows PowerShell Automatic Variables (not exhaustive)

\$\$_	Last token of the previous command line
\$?	Boolean status of last command
\$^	First token of the previous command line
\$_, \$PSItem	Current pipeline object
\$Args	Arguments to a script or function
\$Error	Array of errors from previous commands
\$ForEach	Reference to the enumerator in a foreach loop
\$Home	The user's home directory
\$Host	Reference to the application hosting the PowerShell language

\$Input	Enumerator of objects piped to a script
\$LastExitCode	Exit code of last program or script
\$Matches	Stores a hash table of string values matched by the -match or -notmatch comparison operators.
\$MyInvocation	An object with information about the current command
\$PSHome	The installation location of Windows PowerShell
\$profile	The standard profile (may not be present)
\$Switch	Enumerator in a switch statement
\$True	Boolean value for TRUE
\$False	Boolean value for FALSE
\$PSCulture	Current culture
\$PSUICulture	Current UI culture
\$PsVersionTable	Details about the version of Windows PowerShell
\$Pwd	The full path of the current directory

## Windows PowerShell Preference Variables (not exhaustive)

\$ConfirmPreference	Determines whether Windows PowerShell automatically prompts you for confirmation before running a cmdlet or function
\$DebugPreference	Determines how Windows PowerShell responds to debugging
\$ErrorActionPreference	Determines how Windows PowerShell responds to a non-terminating error
\$FormatEnumerationLimit	Determines how many enumerated items are included in a display
\$MaximumHistoryCount	Determines how many commands are saved in the command history for the current session
\$PSEmailServer	Specifies the default e-mail server that is used to send e-mail messages

\$OFS	Output Field Separator. Specifies the character that separates the elements of an array when the array is converted to a string. The default value is Space.
\$PSDefaultParameterValues	Specifies default values for the parameters of cmdlets and advanced functions
\$PSModuleAutoLoadingPreference	Enables and disables automatic importing of modules in the session. "All" is the default.
\$PSSessionApplicationName	Specifies the default application name for a remote command that uses WS-Management technology
\$PSSessionConfigurationName	Specifies the default session configuration that is used for PSSessions created in the current session
\$PSSessionOption	Establishes the default values for advanced user options in a remote session
\$VerbosePreference	Determines how Windows PowerShell responds to verbose messages generated by a script, cmdlet or provider
\$WarningPreference	Determines how Windows PowerShell responds to warning messages generated by a script, cmdlet or provider
\$WhatIfPreference	Determines whether WhatIf is automatically enabled for every command that supports it

## Collection Filtering

Collection filtering by using a method syntax is supported.  
.Where({ expression } [, mode [, numberToReturn]])  
.ForEach({ expression } [, arguments...])  
  
\$Services = Get-Service  
\$Services.Where({\$\_.Status -eq 'Stopped'}, 'First', 3)  
\$Services.ForEach({\$\_.Name.ToUpper()})  
  
(1..5).ForEach({\$args[0] + \$\_, 'Server'})



## WINDOWS POWERSHELL LEARNING RESOURCES

### Microsoft Resources

#### Scripting with Windows PowerShell

<http://technet.microsoft.com/library/bb978526.aspx>

#### Windows PowerShell Team Blog

<http://blogs.msdn.com/PowerShell>

#### Microsoft Script Center

<http://technet.microsoft.com/scriptcenter/default>

#### Windows PowerShell Forum

<http://social.technet.microsoft.com/Forums/en-US/winserverpowershell/>

#### Hey, Scripting Guy! Blog

<http://blogs.technet.com/b/heyscriptingguy/>

#### Windows PowerShell Survival Guide

<http://social.technet.microsoft.com/wiki/contents/articles/183.windows-powershell-survival-guide-en-us.aspx>

#### Windows PowerShell ISE Add-on Tools

<http://social.technet.microsoft.com/wiki/contents/articles/2969.windows-powershell-ise-add-on-tools.aspx>

#### Windows PowerShell Customer Connection: Submit bugs and feature requests

<https://connect.microsoft.com/powershell>

#### Report Windows PowerShell documentation bugs by email

[write-help@microsoft.com](mailto:write-help@microsoft.com)

### Community Resources

**PowerShell Code Repository:** <http://poshcode.org>

**PowerShell.com Community:** <http://powershell.com>

**PowerShell.org Community:** <http://powershell.org>

**PowerGUI Community:** <http://en.community.dell.com/techcenter/powergui/>

**The PowerShell Community Toolbar:** <http://powershell.ourtoolbar.com/>

**PowerScripting Podcast:** <http://powerscripting.net>

**PowerShell Magazine:** <http://powershellmagazine.com>

[irc.freenode.net #PowerShell](http://irc.freenode.net/#PowerShell)

### Free eBooks and Guides

#### Mastering PowerShell, Second Edition - Dr. Tobias Weltner

<http://powershell.com/cs/blogs/ebookv2/default.aspx>

#### PowerShell.org Free eBooks

<http://powershell.org/wp/ebooks/>

#### Effective Windows PowerShell - Keith Hill

<http://rkeithhill.wordpress.com/2009/03/08/effective-windows-powershell-the-free-ebook/>

### Popular Community Projects

#### PowerShell Community Extensions (PSCX)

<http://pscx.codeplex.com/>

#### PSReadLine - A bash inspired readline implementation for PowerShell

<https://github.com/lzybkr/PSReadLine>

#### TabExpansionPlusPlus - PowerShell module to improve tab expansion and Intellisense

<https://github.com/lzybkr/TabExpansionPlusPlus>

#### PowerSploit - A PowerShell Post-Exploitation Framework

<https://github.com/mattifestation/PowerSploit>

#### PoshSec - PowerShell module focused on security in the Windows environment

<https://github.com/PoshSec/PoshSec>

#### Posh-SecMod - PowerShell module with security-related cmdlets

<https://github.com/darkoperator/Posh-SecMod>

#### Pester - PowerShell BDD-style testing framework

<https://github.com/pester/Pester>