

PowerShell Quick Reference

v3.03 by Dimitri Koens - www.dimensionit.tv

Please contact me with questions, remarks, etc regarding this document at [dimitrikoens at gmail.com](mailto:dimitrikoens@gmail.com). Join me on Linked in and Twitter to receive valuable information regarding PowerShell, SCOM, SQL Server and Virtualization.



Quick Start

`Get-Process` # displays a list of running processes
`Get-Process | Select-Object Name, Company` # selects several columns
`Get-Process | Select-Object Name, Company | Format-Table -AutoSize` # uses minimal column width
`Get-Process | Select-Object Name, Company | Format-List` # displays a list instead of a table
`Get-Process | Sort-Object ID -Descending` # sorts on process id instead of name
`Get-Process | Where-Object { $_.vm -gt 150MB }` # selects processes where virtual memory is greater than 150MB
`Get-Process | Select-Object Name, @{Name="Virtual Memory"; Expression={$_.vm}}` # changes the name of a column
`Get-Process | Select-Object Name, VM, WS @ {Label="TotalMemory"; Expression={$_.vm + $_.ws}}` # introduces a calculated column
`Get-Process | Select-Object Name, VM, WS @ {Label="Total Memory in MB"; Expression={(int)((($_.vm + $_.ws)/1MB)}}` # calculated column and rounded to integer

Built-in Help functionality

`Get-Help`
`Get-Help Get-Process -full`
`Get-Help Get-Process -examples` # view the example commands
`Get-Help about*` # lists all the about-articles, use the full article name to view its contents, e.g. `about_scripts`

`Get-Command` # display all commands
`Get-Command *process*` # display all commands containing the word "process"

`Get-Command -CommandType Cmdlet` # display all native PowerShell cmdlets
`Get-Process | Get-Member` # display the properties and methods of the output

The following two examples help you to protect you from ... you!
`Get-Process PowerShell | Stop-Process -whatif` # displays the command without actually executing it
`Get-Process PowerShell | Stop-Process -confirm` # asks for confirmation

Aliases (and functions denoted by *)

All default cmdlets with "Object" as a noun are aliased to their verb without "Object", e.g. `Sort` is an alias of `Sort-Object`, `Select` is an alias of `Select-Object`

`Get-Alias | Group-Object Definition` # display all possible aliases per command

Command	Alias	Command	Alias	Command	Alias
<code>Clear-Host</code>	<code>cls, clear</code>	<code>Get-Help</code>	<code>help *, man</code>	<code>Rename-Item</code>	<code>rni, ren</code>
<code>Copy-Item</code>	<code>copy, cpi, cp</code>	<code>Get-Member</code>	<code>gm</code>	<code>Select-String</code>	<code>sls (Psv3+)</code>
<code>ForEach-Object</code>	<code>foreach, %</code>	<code>Get-Process</code>	<code>gps, ps</code>	<code>Set-Location</code>	<code>sl, cd, chdir</code>
<code>Format-List</code>	<code>FL</code>	<code>Get-WmiObject</code>	<code>gwmi</code>	<code>Start-Sleep</code>	<code>sleep</code>
<code>Format-Table</code>	<code>FT</code>	<code>Move-Item</code>	<code>mi, move, mv</code>	<code>Stop-Process</code>	<code>spps, kill</code>
<code>Get-ChildItem</code>	<code>gci, dir, ls</code>	<code>Out-Host</code>	<code>oh</code>	<code>Where-Object</code>	<code>where, ?</code>
<code>Get-Command</code>	<code>gcm</code>	<code>Powershell_ise.exe</code>	<code>ise (Psv2+)</code>	<code>Write-Output</code>	<code>echo, write</code>
<code>Get-Content</code>	<code>gc, type, cat</code>	<code>Remove-Item</code>	<code>ri, del, erase, rmdir, rd, rm</code>		

New alias expected in Windows 2015: `Set-Alias Ping Test-Connection`. `Select-String` has an alias in Psv3: `sls`. Shouldn't this have been 'grep'? ;)

Operators (most of them), Get-Help about_Operators

Operator	Meaning	Operator	Meaning
<code>+, -, *, /, %</code>	add, subtract, multiply, divide, remainder	<code>=, +=, -=, *=, /=, %=</code>	assign/change/append one or more values to variables
<code>-eq, -ne</code>	Equal, not equal: <code>5 -eq 5</code>	<code>-match, -notmatch, -cmatch</code>	regular expression match: "Rick" -match "[DMNR]ick"
<code>-gt, -ge</code>	greater than, greater than or equals: <code>6 -gt 5</code>	<code>-contains, -notcontains</code>	Array contains specific value: "red", "blue" -contains "blue"
<code>-lt, -le</code>	less than, less than or equals: <code>5 -lt 6</code>	<code>-and, -or, -xor, -not, !</code>	Logical operators
<code>-like, -notlike, -clike</code>	wildcard comparison: "Samantha" -like "sam*"	<code>-f</code>	Formatting: <code>\$a=2987654</code> ; "free space: {0:N0} bytes" -f \$a

Punctuation Marks

`(expression)` { code block } [item in array] "string with automatic variable expansion"
``` backtick is the escape character, mostly found on the key combined with tilde-sign ~ 'string without automatic variable expansion'

## Keyboard shortcuts

|                                                   |                                                                                                                                        |                                                                                                                                                         |
|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Tab</code> : command completion             | <code>F7</code> : display history popup, <code>Alt-F7</code> : clears command buffer                                                   | <code>Ctrl ←, Ctrl →</code> : jump one word left or right                                                                                               |
| <code>Esc</code> : clear the command line         | <code>F8</code> : lookup last command that starts with current input. Try this: <code>Get-Process; &lt;enter&gt;; Get&lt;F8&gt;</code> | More: <code>&lt;Ctrl-C&gt;</code> quit, <code>&lt;q&gt;</code> quit, <code>&lt;space&gt;</code> scroll page, <code>&lt;enter&gt;</code> scroll one line |
| Use arrow up and down to browse previous commands | <code>Home, End</code> : jump to start or end of current command line                                                                  | Within ISE: <code>F5</code> = Run, <code>F8</code> = Run Selection                                                                                      |

## Security

| The .ps1 extension                                                                                                                                                                               | Execution Policy (Set- and Get-ExecutionPolicy)                                                                                                                                                                        | To prevent command hijacking                                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Associated with Notepad. When a user receives a PowerShell script through e-mail and doubleclicks it then the script just opens in notepad instead of executing (like the i-love-you virus did). | Restricted (default), AllSigned, RemoteSigned, Unrestricted (not recommended)<br>Remote scripts: not on local fixed disks, like CD's/DVD's, drive mappings to network shares, attachments in e-mail and chat-programs. | You can only run commands from the current location by specifying the path to the script.<br>Example: <code>.\script.ps1</code> instead of <code>script.ps1</code> . |

## Variables

`$_` # Current object in the pipeline  
`$Home` # Full path to the user's home directory  
`$PSHome` # Full path to the installation directory  
  
`$Host` # Displays the PowerShell version  
`$i = 1` # storing value 1 in variable \$i  
`$i++` # incrementing \$i with 1, resulting in 2

## Working with files

`Get-Process | Out-File p.txt -append` | `Get-Process | Export-CSV p.csv` | `Get-Process | Export-CliXML p.xml`  
`Import-CSV p.csv` # displays using `Format-List` because there are more than four columns and object type is not recognized  
`Import-CliXML p.xml` # displays using `Format-Table` because object type is recognized (try to add: | `Get-Member`)  
`Compare-Object (Import-Clixml p.xml) (Get-Process) -Property name` # compare processes stored in XML with current situation

`Dir -Recurse | Where { $_.length -gt 100MB }` | `Group Length` | `Where { $_.count -gt 1 }` # displays large files with exact same size, might be duplicate

## What's New in PowerShell 3

Requirements: W2008 R2 SP1, W7 SP1, .NET 4.0, download and install KB2506146 or KB2506143.

Note: install ISE first before upgrading to PowerShell 3 on Windows 2008 (R2) or Windows 7!

Automatic module loading: no need to use e.g. "Import-Module ActiveDirectory" anymore.

ISE changes: IntelliSense, brace matching, error indication, start snippets, rich copy, block select (Alt+Mouse), context sensitive help (F1), many more!

New modules: BranchCache, DirectAccess, Dism, DHCP, DNS, iSCSI, NetAdapter, NetTCPIP, NetworkSecurity, PKI, Printing, Scheduled Tasks, SMB, Storage, many more!

`$PSItem` # same function as `$_`: current item in pipeline

`Show-Command` # shows a GUI for a specific Cmdlet

`Get-Process` | `Out-GridView -OutputMode Multiple` | `Stop-Process` # `Out-GridView` has an `OutputMode` parameter: select several items and press OK

`Install-WindowsFeature WindowsPowerShellWebAccess -IncludeManagementTools -Restart` # Installs PowerShell Web Access feature

`Install-PswaWebApplication -UseTestCertificate` # creates the virtual directory and application pool using a test certificate

`Add-PswaAuthorizationRule * * *` # Creates an authorization rule. Now browse with any supported browser (IE, Chrome!, Firefox!, Safari!) to `http://<server>/pswa`

## Looping

`for ($i = 1; $i -le 10; $i++) { $i }` # displays numbers 1 through 10. See the Active Directory section for a practical example

| While loop only executes when condition is true | Do ... While loop, always executes, at least once | Do ... Until loop, always executes, at least once |
|-------------------------------------------------|---------------------------------------------------|---------------------------------------------------|
| <code>\$i = 1</code>                            | <code>\$a = 1</code>                              | <code>\$a = 1</code>                              |
| <code>While (\$i -le 10) { \$i; \$i++ }</code>  | <code>Do {\$a; \$a++} While (\$a -lt 10)</code>   | <code>Do {\$a; \$a++} Until (\$a -gt 10)</code>   |

## Typical example of a Do ... Until loop

`$RequiredLength = 12`

```
Do {
 $password = read-host -prompt "Password, please"
 if ($password.length -lt $RequiredLength) { "password is too short!" }
} Until ($password.length -ge $RequiredLength)
```

## Functions

Function `Get-NewestEventlog` { Param (\$log="system", \$newest=5) `Get-Eventlog $log -newest $newest` }

`Get-NewestEventlog` # try with parameters like `-log application -newest 10`

## WMI

`Get-WmiObject -list` # lists all WMI classes

# inspecting shares through WMI

`Get-WmiObject Win32_Share`

`$share = Get-WmiObject Win32_Share | Where { $_.Name -eq "C:" }`

`$share | Get-Member` # check name and caption

# we'll need the wmiclass type to create objects through WMI

`$share=[WMICLASS]"Win32_Share"`

`$share.create("C:\", "mynewshare", 0)` # creating a new share

# automating defragmentation (please check with your SAN administrator!)

`$Cvolume = Get-WmiObject Win32_Volume | Where { $_.name -eq "C:" }`

`$df = $Cvolume.DefragAnalysis()` # can take several minutes or even hours

`$df` # inspecting the result

`if ($df.DefragRecommended) { $Cvolume.defrag($true) }`

`Get-WmiObject Win32_OperatingSystem -computername (Get-Content servers.txt) | Format-Table __SERVER,Version,ServicePackMajorVersion,ServicePackMinorVersion`

`Get-WmiObject Win32_LogicalDisk -Filter 'DriveType=3' -ComputerName (Get-Content computers.txt) |`

`Format-Table __SERVER,DeviceID,FreeSpace,@{Label='PercentFree';Expression={$_.FreeSpace / $_.Size};FormatString='{0:#0.00%}'}`

## Active Directory

**Requirements:** PowerShell v2, Active Directory Module for Windows PowerShell (on a Domain Controller, also part of RSAT). Open port **TCP/9389**.

**Requirements:** Windows Server 2008 R2 Domain Controller or install ADMGS on a W2003/2008 Domain Controller.

`Import-Module ActiveDirectory` # imports the Active Directory module for PowerShell

`Get-Command -module ActiveDirectory` # displays all 76 commands in PowerShell v2

`New-ADOrganizationalUnit "Employees" -Path "DC=Contoso,DC=com"` # creates a new OU

`Get-ADOrganizationalUnit -Filter "*" | FT Name, DistinguishedName -AutoSize`

`New-ADUser TestUserA` # creates a disabled user in the Users container

# The next script takes a plain text password as input and creates an enabled user account in the Employees OU

`$userpwd = ConvertTo-SecureString -AsPlainText "Pa$$w0rd" -Force` # converts plaintext to secure string

`New-ADUser TestUserB -AccountPassword $userpwd -Enabled $true -Path 'OU=Employees,DC=Contoso,DC=com'`

`For ($i=1; $i -le 10; $i++) { New-ADUser -name Testuser$i }` # creates ten new testusers

## Background Jobs

`Start-Job { Get-Process PowerShell }`

`Get-Job`

`Get-Job -Id 1 | Receive-Job` # use the `-Keep` parameter to keep the data in memory

`Start-Job { Sleep 60 }` # starts a new job which just waits for 60 seconds

`Wait-Job -Id 3` # wait for a job to complete, or use: `Stop-Job -Id 3` # stops job

`Remove-Job -Id 3` # remove a completed job

## Remoting

**Requirements:** PowerShell v2 on local and remote systems. Enable Remoting on remote system. Open port **TCP/5985**.

`Enable-PSRemoting` # Run this on the remote system. Use `-Force` optionally.

`Enter-PSSession -ComputerName Server2`

# use your PowerShell skills now on the remote machine

`Exit-PSSession`

Free ebook about remoting: [http://www.ravichaganti.com/blog/?page\\_id=1301](http://www.ravichaganti.com/blog/?page_id=1301)

## Error handling and Debugging

`$ErrorActionPreference` # displays the default action when an error occurs

`Dir c:, x:, c:` # should result in a file listing of the c-drive, followed by an error, followed by a listing of the c-drive

`$ErrorActionPreference = 'SilentlyContinue'` # or 'Continue' or 'Inquire' or 'Stop'

`Dir c:, x:, c:` # should result in two file listings of the c-drive, no more error

# Use the `-ErrorAction` parameter to use a other error action for the current command only

`$Error[0] | Get-Member` # displays information about the last error

## More information on the Internet (and the previous page...)

<http://blogs.msdn.com/b/powershell/>

<http://technet.microsoft.com/en-us/scriptcenter/dd742419.aspx>

<http://poshoholic.com/>

<http://thepowershellguy.com/>

<http://www.powershellmagazine.com/>

<http://www.computerperformance.co.uk/powershell/>

<http://powerwf.com/products.aspx>

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

<http://blogs.msdn.com/b/kathykam/archive/2006/03/29/564426.asp>