PowerShell 7.0 - Quick Reference

https://devblogs.microsoft.com/powershell/announcing-PowerShell-7-0/

www.practicalpowershell.com

Get-Help / Helpful Commands

Update-Help	*Updates local help files.*
Get-Help	*Provides information on a command, it's parameters and available switches.*
Get-Command	*Lists all commands. Can be filtered.*
Get-Module	*Lists modules that are or can be loaded.*
Get-Package	*Lists packages that are or can be loaded.*
Get-PSRepository	Lists available PowerShell Repositories registered to the current user.
Get-Member	*Gets properties and methods of objects.*
Get-PackageProviders	*Lists all loaded package provides. (i.e. NuGet, PowerShellGet, etc.)*
Show-Command	List of available commands (GUI)

Operators

New Operators

<Condition> ? <if-true> : <if-false> *Ternary operator*

```
$Path = 'C:\Scripts'
(Test-Path $path) ? "Path exists" : "Path not found"
# Result is 'Path exists' if the c:\scripts path is present
```

|| , && *Pipeline chain operators*

```
#If process named 'Chrome' is found (left)/stop it (right)
Get-Process Chrome && Stop-Process -Name Chrome
# If the npm install fails, removenode_modules dir.
npm install || Remove-Item -Recurse ./node_modules
```

Null coalescing operators

```
$x = $null          $x = $null
$x ?? 476           $x ??= 456
# Result 476        $x
                    # Result 476, $x is assigned this value
```

Assignment Operators

=	Equal	+=	Increments Value
-=	Decrements value	*=	Multiplies value
/=	Divides value	%=	Divide and assigns remainder
++	Increment value (+1)	--	Decrement Value (-1)

BitWise Operators

*** Only works with integers and works in binary form*

-band	Bitwise AND
-bor	Bitwise OR (inclusive)
-bxor	Bitwise OR (exclusive)
-bnot	Bitwise NOT
-shl	Bit shift left
-shr	Bit shift right

*** https://codesteps.com/2019/03/28/powershell-bitwise-logical-operators/*

Operators

Comparison Operators

-eq	equal	-ne	not equal
-lt	less than	-gt	greater than
-ge	greater than or equal	-le	less than or equal

-replace	Replace string pattern
-like	Returns true when string matches
-notlike	Returns true when string does not match
-match	Returns true when string matches regex
-notmatch	Returns true when string does not match regex
-contains	Returns true when reference value in a collection
-notcontains	Returns true when reference value not in a collection
-in	Returns true when test value contained in a collection
-notin	Returns true when test value not contained in a collection

Logical Operators

-and TRUE when both are TRUE
e.g. '(3 -eq 3) -and (1 -lt 3)' is TRUE

-or TRUE when either is TRUE
e.g. (3 -lt 3) -or (2 -eq 2) is TRUE

-xor TRUE when only one is TRUE
e.g. (1 -eq 1) -xor (2 -eq 2)
FALSE

-not/! When a condition is not TRUE
e.g. -not (1 -eq 1) is FALSE

Other Operators

-split Splits a string

```
'FirstName.LastName' -Split '.'
# Results - 'FirstName' and 'LastName'
```

-join Join's multiples strings

```
'John','Smith','IT','Chicago' -Join ','
# Results - John,Smith,IT,Chicago
```

-replace Replaces a value

```
'Dog.runs.down.street' -Replace ( '.' , ' ')
# Results – 'Dog runs down street
```

Type Operators

-is,-isnot Used to validate a .Net Type

```
(Get-Date) -is [DateTime] #Returns True
(Get-Date) -is [Int32] #Returns False
```

-as Converts input to .Net Type

```
'4/1/2020' -as [DateTime ]
#Returns Wednesday, April 1, 2020 12:00:00 AM
```

-f Format output of string objects

```
"{1} {0} {4}" -f 'runs' , 'dog' , 'fast' , 'yellow' , 'slow'
# Result -  'Dog runs slow'
```

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_operators

[] **Cast operator**. Converts or limits object to type.

```
[DateTime]Today = '2/5/1999'
[Int32]$Counter = 59
```

, **Comma operator**, creates an array.

```
$ThisArray = 1, 2, 5
```

. **Dot sourcing operator** runs a script in the current scope.

```
. C:\Scripts\QA\GetAll.ps1
```

| **Pipeline operator.** Sends output ('pipes') to another cmdlet for processing.

```
Get-Mailbox | Set-Mailbox -RetentionPolicy 'CorpReten'
```

.. **Range Operator**

```
20..33 # Lists numbers 20 through 33, incremented by 1's
```

Redirection Operators

>,>>,&> Sends the output of a stream to a file as well as output of a particular type.

Output Streams		
	*	All Output
	1	Success
	2	Error
	3	Warning
	4	Verbose infor
	5	Debug messages
	6	Information

Redirection Operator examples:

```
# Writes warning output to warning.txt
Get-Mailbox 3> warning.txt

# Appends verbose.txt with the verbose output
Set-Computer  4>> verbose.txt

# Writes debug output to the output stream
Remove-AzVM 5>&1

# Redirects output to ADDCs.txt file
Get-ADDomainContrller > ADDCs.txt
```

PowerShell 7.0 - Quick Reference

www.practicalpowershell.com

Automatic Variables (not exhaustive)

Variables that store state information, created/maintained by PowerShell and should be treated as Read-Only.

$$	Last token in the last line received by the session
$?	Contains the execution status of the last command.
$^	Contains the first token in the last line received by the session.
$_,$PSItem	Current object in the pipeline object.
$args	Contains an array of values for undeclared parameters that are passed to a function, script, or script block.
$ConsoleFileName	Contains the path of the console file (.psc1) that was most recently used in the session.
$Error	Array of errors from previous commands.
$ExecutionContext	Contains an EngineIntrinsics object that represents the execution context of the PowerShell host.
$foreach	Contains the enumerator of a ForEach loop.
$HOME	Full path of the user's home directory.
$Host	Represents the current host application for PowerShell.
$input	Enumerates all input passed to a function.
$IsCoreCLR	.NET Core Runtime check. $True/$False
$IsLinux	$True if Operating system is Linux.
$IsMacOS	$True if Operating system is Mac.
$IsWindows	$True if Operating system is Windows.
$LastExitCode	Exit code of the last Windows-based program that was run.
$Matches	Hash table of any string values matched with the -match and -notmatch operators.
$MyInvocation	Contains information about the current command, such as the name, parameters, parameter values, and more.
$null	Represents an empty or null value.
$PID	Process identifier (PID) of PowerShell session.
$PROFILE	Full path of the PowerShell profile for the current user and the current host application.
$PSCulture	Reflects the culture of the current session.
$PSDebugContext	This variable contains information about the debugging environment.
$PSHome	Full path of the installation directory for PowerShell
$PSItem, $_.	Contains the current object in the pipeline object.
$PSScriptRoot	Directory from which a script is being run.
$PSSenderInfo	Contains the directory from which a script is being run.
$PSUICulture	Name of the user interface (UI) culture for OS.
$PSVersionTable	Read-only hash table that displays details about the version of PowerShell that is running in the current session.
$PWD	Path Object - full path of the current directory.
$ShellID	Identifier of the current shell.
$StackTrace	Stack trace for the most recent error.
$Switch	Contains the enumerator not the resulting values of a Switch statement.

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_automatic_variables

Variables

Examples:

```
$Path = 'C:\Scripts\TestScript'
$Date = Get-Date
$Processes = Get-Process
```

Change value of variable

```
$Path = 'C:\Windows\System32'
$Date = ($Date).AddDays(-90)
$Processes = (Get-Process).Name
```

Clear Variable of values

```
Clear-Variable -Name $Path
Clear-Variable -Name $Date
Clear-Variable -Name $Processes
```

Scoped

```
$Global:Server='Ex01'     Global variable, visible everywhere
$Local:Count=1            Visible in local scope and child scopes
$Private:State='Test'     Visible in local scope, but not child scopes
```

Multi-Assignment

```
$State,$Count,$PC = 'Enabled', '1', 'Windows10'
```

Flip Variables

```
$Count1=3 ; $Count2=5 ; $Count1,$Count2 = $Count2,$Count1
```

Read-Only Variable (can be overwritten with -Force)

```
Set-Variable 'PermRef' -Value '1973' -Option ReadOnly
```

Constant Variable Cannot be overwritten

```
Set-Variable 'Important' -Value '1973' -Option Constant
```

Variable Acceptable Values:

```
[ValidateRange(90,150)][int]$Tolerance = 99
$Tolerance = 151 #Returns error – not valid for the variable
```

Preference Variables

$ConfirmPreference	Determines whether PowerShell automatically prompts you for confirmation before running a cmdlet or function.
$DebugPreference	Determines how PowerShell responds to debugging.
$ErrorActionPreference	Determines how PowerShell responds to a non-terminating error.
$ErrorView	Determines the display format of error messages in PowerShell.
$FormatEnumerationLimit	Determines how many enumerated items are included in a display.
$InformationPreference	Lets you set information stream preferences that you want displayed to users.
$MaximumHistoryCount	Determines how many commands are saved in the command history for the current session.
$OFS	The Output Field Separator specifies the character that separates the elements of an array that is converted to a string. Default (" ")
$OutputEncoding	Determines the character encoding method that PowerShell uses when it sends text to other applications.
$ProgressPreference	Determines how PowerShell responds to progress updates.
$PSEmailServer	Specifies the default e-mail server that is used to send email messages.
$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 PowerShell responds to verbose messages generated.
$WarningPreference	Determines how PowerShell responds to warning messages generated.
$WhatIfPreference	Determines whether WhatIf is automatically enabled for every command that supports it.

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_preference_variables

PowerShell 7.0 - Quick Reference

www.practicalpowershell.com

Arrays

'bob','r','smith'	Array of strings
10,45,100	Array of integers
@()	Empty array (initiate)
@(3)	Array of 1 element
@(3,4,5)	Array of 3 elements
2,(5,7),10	Array within an array
$Process[0]	First element in an array
$Computer[2]	Third element in an array
$User[5..14]	Elements 6 through 15
$Server[-1]	Returns last element
$Num[-4..-1]	Returns last 4 elements
@(Get-AzVM)	Stores results in an array

Reverse an Array

```
$a = 1,2,3,4,5
[array]::Reverse($a)
# $a would then store the values as 5,4,3,2,1
```

Combine Arrays (+)

```
$A = 1,2,3 ; $B = 4,5,6 ; $C = $A+$B
```

Create new array based on existing array

```
$SomePCs = $AllPCs[1,3,5,7+9..13]
```

Hash Tables

$Hash = @ { }	Creates an empty hash table
$Hash =@{ColorOne = 'Red'}	Creates hash table with data
$Hash.ColorOne	Display ColorOne key
$Hash.ColorTwo = 'Green'	Assigns 'Green' to this key

Add values to hash

```
$Color = 'ColorThree' ; $Value = 'White'
$Hash.Add($Color,$Value)
```

Remove value from hash

```
$Hash.Remove('ColorTwo')
```

Sort table by Key values

```
$Hash =@{ColorOne = 'Red'}
$Hash.ColorTwo = 'Green'
$Color = 'ColorThree' ; $Value = 'Blue'
$Hash.Add($Color,$Value)
$Hash.Remove('ColorTwo')
$Hash.GetEnumerator() | Sort-Object -Property Value
```

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_hash_tables

Comments

Starting a line with a '#' makes the line a comment

```
# Load PowerShell Modules
$Var = '#Not a comment example'
# Write-Host 'But this is an example'
$State = 'Enabled' # Set the State variable
```

Multi-Line Comments

```
<#
Synopsis: This is a section of comments.
Purpose: To enclose a large section of text.  Possibly to be
used as a header for a script.
Version: 1.0
Parameters: None
#>
```

Helpful Tips

Use tab to autocomplete cmdlets
Tab through parameters to see all available
Check for latest module versions
Read latest Microsoft Docs for PowerShell
Read PowerShell MVP blogs for more tips
Remove line wrapping from PowerShell session

TAB	Autocomplete or cycle through all options
Ctrl+Space	Display all available parameters/switches
Ctrl+V	Copy data to session

Object Properties

Properties for an object can be accessed with '.' followed by the property name. For example:

```
$Process = Get-Process 'Chrome'
$Process.ID
$DC =  get-adcomputer dc01 -Properties *
$DC.dSCorePropagationData
```

If there are sub-properties, add with the '.' separator:

```
$DC.dSCorePropagationData.Date
```

For Static Properties use ::

```
[datetime]::Now
```

Strings

```
'String – this is an example'
"Contains a $Variable that displays its value"
'Single quotes $Variable whose content is not displayed'
@"
This is a more versatile string that can store quotes, returns
and can also evaluate variables.  For example.  Today's date:
$Date
Then we can close it off like we started this string.
"@
@'
This one is less versatile as it will not evaluate variables:
$Date
Then we can close it off like we started this string.
'@
```

Loops

Foreach

The Foreach statement steps (iterates) through a series of values in a collection of items.

```
$CSVFileData = Import-CSV "C:\Data.csv"
Foreach ($Line in $CSVFileData) {
   $DisplayName = $Line.DisplayName
   $Size = $Line.MailboxSizeMB
   Write-host "$DisplayName mailbox = $Size MB ."
}
```

ForEach-Object (Parallel – New Feature)

```
$Logs | Foreach-Object -Parallel {$File = $_+'.txt';get-winevent -
LogName $_ -MaxEvents 5000 > $File } -ThrottleLimit 10
```

Do While

Traverses list one or more times, subject to a While condition.

```
$Counter = 1
Do {
   Write-Host "This is pass # $counter for this loop."
   $Counter++
} While ($Counter -ne 1000)
```

Do Until

Traverses list one or more times, subject to a Until condition.

```
$Users = Get-ADUser
Do {
   Foreach ($User in $Users) {
      $State = $Users.Enabled
      $FirstDisabledUserAccount = $User
   }
} Until ($State -eq 'Disabled)
```

PowerShell 7.0 - Quick Reference

www.practicalpowershell.com

Compatibility

Reference: https://github.com/powershell/powershell#get-powershell

Windows Server 2008 R2, 2012, 2012 R2, 2016, and 2019	
Windows 7, 8.1, and 10	macOS 10.13+
Red Hat Enterprise Linux (RHEL) / CentOS 7+	Fedora 29+
Debian 9+	Ubuntu 16.04+
openSUSE 15+	Alpine Linux 3.8+
ARM32 and ARM64 flavors of Debian and Ubuntu	ARM64 Alpine Linux.

Experimental Features

New to PowerShell 7.0 is the concept of *Experimental Features*. These features are testing newly developed modules, in test and **not** production.

List any Experimental Features available to PowerShell 7.0's shell:
Get-ExperimentalFeature

Disable an Experimental Feature (if further development is needed, for example):
Disable-ExperimentalFeature

Enable a new experimental feature or to enable an existing disabled feature (Microsoft's examples):
Enable-ExperimentalFeature

Other Topics

List all executed commands for the current session:

Get-History	List all previous commands
Get-History -Id 17 | Fl	List the 17th executed command
Clear-History	Remove all entries from the history
Add-History	Add additional entries to the history
Invoke-History -Id 12	Re-runs item 12 from the history

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_history?view=powershell-7

Dates can be important in PowerShell

Get-Date	Displays the current date and time
(Get-Date).AddDays(-30)	Displays the date from 30 days ago
(Get-Date).AddHours(4)	Displays the time 4 hours from now

Format date examples:
Get-Date -Format yyyymmdd-hhmmss
Get-Date -Format "MM.dd.yyyy-hh.mm-tt"

List items in a graphic format

$Processes | Out-GridView	Displays running process in a grid

List items in a grid, allows selection and pass back to session
$Processes | Out-GridView -PassThru

Measure how long a function takes to execute:

```
$StopWatch = [Diagnostics.Stopwatch]::StartNew()
& $FunctionToExecute
$StopWatch.Stop()
$StopWatch.Elapsed
```

Or

```
Measure-Command {$FunctionToExecute}
```

File Output
Get-AzVM | Export-CSV AzureVirtualMachines.csv
Get-AdComputer -Filter * | Out-File AllDomainComputers.txt
Get-Process | Out-File AllProcesses.txt -Append -NoClobber

Supported Modules

* All modules supported by PowerShell 6

Incompatible modules
Import-Module -UseWindowsPowerShell <Module Name>
Uses local WindowsPowerShell for this module

Working with Modules

PowerShell cmdlets are grouped my modules. We can work with supported cmdlets from any module. We can also load and unload modules as needed depending on if we need more cmdlets.

List Modules

Get-Module	Lists loaded modules
Get-Module -ListAvailable	Lists all available modules

Load and unload modules

Import-Module ActiveDirectory	Loads ActiveDirectory module
Remove-Module AZ	Unloads the AZ module

List cmdlets for a module

```
$Module = 'SharePointPnPPowerShellOnline'
Import-Module $Module
Get-Command |Where {$_.Source -eq 'SharePointPnPPowerShellOnline'}
```

Locate a Module in a repository
Find-module MicrosoftTeams
Find-module ExchangeOnline* #Can use wildcards

Install Module
Install-Module MicrosoftTeams
Find-module ExchangeOnlineManagement

Other Module functions
Uninstall-Module LyncOnlineConnector
Update-Module ExchangeOnlineManagement

Troubleshooting

New cmdlet – Get-Error
Use this cmdlet to retrieve past error messages.
Examples

Get-Error	# Diplays the last error message
Get-Error -Newest 2	# Displays last two error messages

Pause and Sleep
Add a pause or have PowerShell 'Sleep' for a matter of seconds

Pause	# waits for operator to hit the 'Enter' key
Sleep 10	# Waits 10 seconds and then moves on

Write-Host
Can be used to display variable content, known possible errors

```
Write-Host 'Step 1'
Write-Host 'Step 2'
Write-Host 'Step 3'
```

Write a Windows Event

```
New-winevent -ProviderName Microsoft-Windows-PowerShell
  -ID 8196
Get-WinEvent -ProviderName Microsoft-Windows-PowerShell
  -MaxEvents 100
```

List Providers
Get-NetEventProvider -ShowInstalled | Ft Name

Comments
Use comments to remove a one-liner or cmdlet from executing
Set-Mailbox -RetentionPolicies Temp

Try and Catch
Used to catch errors and perform secondary/final actions.

```
Try {
Set-ADForestMode -Identity corp.loc -ForestMode
      Windows2016Forest
} Catch {
Write-Host 'AD cmdlet failed to execute.' -ForegroundColor Red
}
```

PowerShell 7.0 - Quick Reference

www.practicalpowershell.com

PowerShell Reference Links

PowerShell Dev Blog
https://devblogs.microsoft.com/powershell/

Scripting Blog
https://devblogs.microsoft.com/scripting/

PowerShell 7.0
https://docs.microsoft.com/en-us/powershell/scripting/overview

DSC
https://docs.microsoft.com/en-us/powershell/scripting/dsc/overview/overview

Windows PowerShell Forum
https://social.technet.microsoft.com/Forums/windowsserver/en-US/home?forum=winserverpowershell

PowerShell Survival Guide
https://social.technet.microsoft.com/wiki/contents/articles/183.windows-powershell-survival-guide.aspx

Visual Studio Code
https://code.visualstudio.com/

Visual Studio Code Extensions
https://marketplace.visualstudio.com/VSCode

PowerShell Documentation
https://docs.microsoft.com/en-us/powershell/

PowerShell Podcast
https://powershell.org/category/podcast/

PowerShell Magazine
http://powershellmagazine.com

Good Blogs (Community and MVP blogs)
https://powershell.org/
https://www.planetpowershell.com/
https://mikefrobbins.com/
http://jdhitsolutions.com/blog/
https://richardspowershellblog.wordpress.com/
https://www.powershellmagazine.com/
https://evotec.xyz/category/powershell/
https://adamtheautomator.com/
https://learn-powershell.net/
https://blog.netnerds.net/

PowerShell Tips of the Week
www.practicalpowershell.com/blog

PowerShell Tools

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

PowerShell Script Analyzer
https://github.com/PowerShell/PSScriptAnalyzer

PowerShell Editors

Visual Studio Code	Microsoft premier PowerShell editor, replaces ISE. Supports more than just PowerShell editing.
PowerShell ISE	The original Microsoft PowerShell editor ***** ISE does not support PowerShell 7.0 *****
Notepad++	Notepad++ free editor, supports more than PowerShell editing
PowerShell Plus	Free PowerShell editor by Idera
PowerShell Studio	Paid editor by Sapien Technologies
Notepad	OK. It's an editor, but it's not an IDE.

Popular GitHub Repos

PSReadLine
https://github.com/PowerShell/PSReadLine

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

Windows OS Hardening with DSC
https://github.com/NVISO-BE/posh-dsc-windowsserver-hardening

PoSH Git
https://github.com/dahlbyk/posh-git

Ninja
https://github.com/ahmedkhlief/Ninja

Detection Lab
https://github.com/clong/DetectionLab

Atomic Red Team
https://github.com/redcanaryco/atomic-red-team

Free eBooks and Guides

https://leanpub.com/u/devopscollective
https://books.goalkicker.com/PowerShellBook/

PowerShell About Pages (Good read!)

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/