PowerShell & PowerCLI

Starting from Scratch

Indy VMUG

July 2023

Mike

mikenelsonio
nelmedia
mikenelson-io

- Almost 40 years in tech
- Principal Technical Evangelist @ Pure Storage
- Experience from Helpdesk to Architect
- Scripter, not a coder
- Passion for community, teaching, learning
- Beer, BBQ, & Gadgets

Microsoft Most Valuable Professional

Citrix Technology Advocate CTA

vmware vEXPERT

mikenelson-io

/MyPresentations

Why use PowerCLI?

PowerCLI does not exist without PowerShell

PowerShell

aka PoSH

Started as a scripting framework for automation & evolved into a command line interface (CLI) and a scripting language.

Native executables, cmdlets, scripts, functions, aliases, modules, help, profiles, parameters, and more

Versions

.Net Framework

<=5.1

Windows

.NET Core

7.x

Windows
Linux
MacOS

Profiles

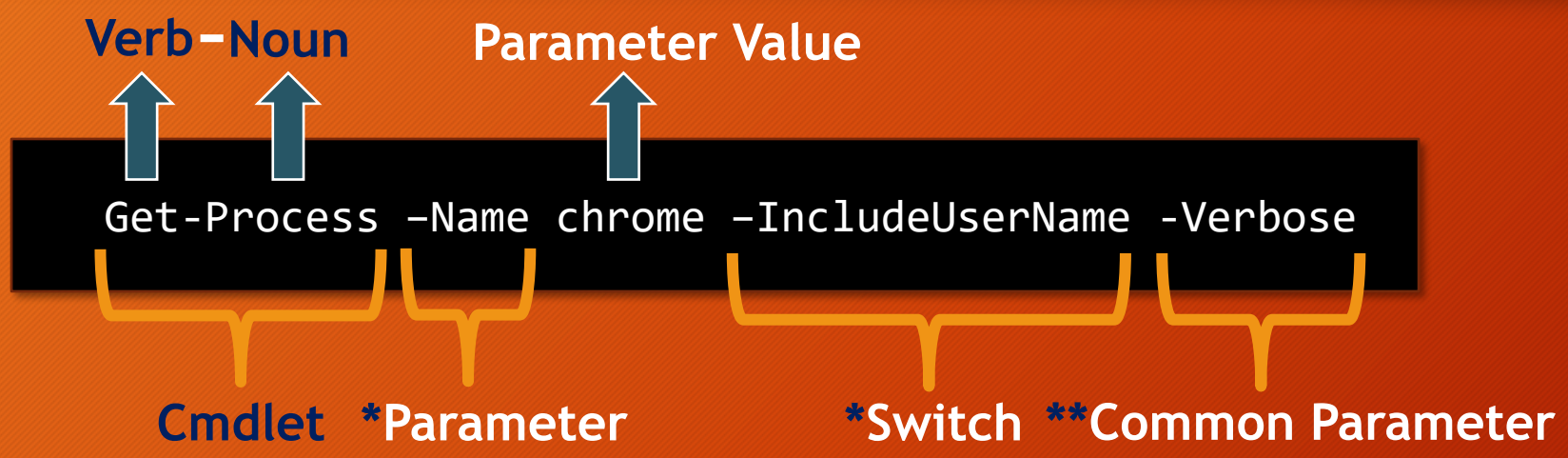
- The PowerShell profile is a script that runs when a PowerShell session is started (unless the –noprofile switch is used).
- Basically, it is a logon script for PowerShell containing commands, aliases, variables, drives, functions, modules, etc.
- You can have different profiles for different user scopes, and there is no default profile.
- Your current user profile is stored in the $profile variable. To edit your current user profile with VSCode, type code $profile at a PowerShell prompt.

Cmdlets

“command-lets”

- The “soul” of PowerShell
- A type of command in PowerShell
- Common syntax & options
- Almost always takes objects as input & return objects as output
- Used at the command line or placed in a .ps1 file. .ps1 files are PowerShell (ie. PowerCLI) script files.

Example Syntax



* = Optional or required
** = the cmdlet may or may not support

Variables

- A unit of *memory* in which a value is stored
- PowerShell variables are text strings represented by the dollar sign "$"prefix (ex. $a, $my_var, $var1, etc.)
- Although special characters and spaces allowed, variable names should be kept simple
- Types of variables:
  - User – user defined and deleted on exit (add to your PowerShell Profile to sustain)
  - Automatic – defined by Posh & not editable (ex. $PSHOME)
  - Preference – defaults defined & are user editable
- Type `Get-Variable` to show all variables defined in a session

Parameters

- Allow for users to provide input or options
- A pre-hyphen ("-") is not always necessary (ie. a positional parameter)
- Some parameters have default values (dev decision)
- Different Types:
  - Named -> default full name of parameter
  - Positional -> typed in a relative order (caution)
  - Dynamic -> only available under special conditions
  - Common -> built-in parameters
  - Sets -> expose different parameters & return different information

Pipelines

Pipeline operator

```
PS>Get-Process –Name chrome –IncludeUserName | Stop-Process
```

Object(s) returned by first cmdlet are sent (piped) to the second cmdlet

“One-liner”

Pipelines

“One-liner”

Get all Windows VMs that need updated tools, then update all the tools at once

```
PS> Get-VM -Location 'MyDatacenter' | Where-Object { $_.ExtensionData.Guest.ToolsVersionStatus
 -eq 'guestToolsNeedUpgrade' -and $_.PowerState -like 'PoweredOn' } |
Get-VMGuest | Where-Object { $_.GuestFamily -like 'WindowsGuest'} |
Update-Tools -NoReboot -RunAsync
```

cmd.exe max character limit? 8,191

PowerShell max character limit? 32,764

PowerShell command separator? “||” Ex. Get-Process || Get-Disk

Functions

A list of PowerShell statements that run like you had entered them on the command line.

```
function Get-ChromeProcess { Get-Process chrome }

function Get-ChromeProcess {
  $a = Get-Process chrome
    if ($a –eq $null) {
      Write-Host “No Chrome process present”}
    return $a
}

Get-ChromeProcess
```

To run a function, simply “call” it.

Scripts, & Modules

- Review: Scripts are .ps1 files
- Modules are .psm1 files, which can contain commands, providers, variables, functions, help context, aliases, workflows, etc., all bundled into a single file.
- A .psd1 file is a module manifest file, which is basically a definition file for a module.
- Modules can be autoloaded by PowerShell
  - Uses the Abstract Syntax Tree (AST) to determine module from cmdlet

Example Manifest & Modules

VMware-vCD-TenantReport.psd1

Get-NicDetails.psm1

Get-NewAndRemovedVMs.psm1

Creating / Editing / Running

Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

bulk-update.ps1

```
13
14  # Create arrays for skipped and failed users
15  $SkippedUsers = @()
16  $FailedUsers = @()
17
18  # Loop trough CSV records
19  foreach ($CSVrecord in $CSVrecords) {
20      $upn = $CSVrecord.UserPrincipalName
21      $user = Get-AzureADUser -Filter "userPrincipalName eq '$upn'"
22      if ($user) {
23          try {
24              $user | Set-AzureADUser -Department $CSVrecord.Department -Tele
25          }
```

```
PS C:\> Get-Host


Name             : Windows PowerShell ISE Host
Version          : 5.1.17763.1490
InstanceId       : b76947f9-993f-4f21-a072-ab27af7f927f
UI               : System.Management.Automation.Internal.Host.InternalHostUserInte
CurrentCulture   : en-US
CurrentUICulture : en-US
PrivateData      : Microsoft.PowerShell.Host.ISE.ISEOptions
DebuggerEnabled  : True
IsRunspacePushed : False
Runspace         : System.Management.Automation.Runspaces.LocalRunspace
```

Completed

Integrated Scripting Environment (ISE)

File Edit Selection View Go Run Terminal Help — Remove-OldModules.ps1 - Visual Studio Code - Insiders

Remove-OldModules.ps1

F: > powershell > Remove-OldModules.ps1 > Remove-OldModules { }

```
1 reference
1  function Remove-OldModules
2  {
3      $Latest = Get-InstalledModule
4      foreach ($module in $Latest) {
5
6          Write-Verbose -Message "Uninstalling old versions of $($module.Name) [latest is $( $module.Version)]" -Verbose
7          Get-InstalledModule -Name $module.Name -AllVersions | Where-Object {$_.Version -ne $module.Version} | Uninstall-Module -Verbose
8      }
9  }
10
11 Remove-OldModules
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE .NET INTERACTIVE

```
PowerShell 7.2.3
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

Loading personal and system profiles took 1593ms.
C:\Users\Mike>
```

Azure: mike@nelmediaconsulting.com — Ln 1, Col 1 — Spaces: 2 — UTF-8 — CRLF — PowerShell — Explain (Denigma) — Spell — Deck | Connected | Active — Prettier

Visual Studio Code (VSCode)

** Use VSCodium for security-minded folks*

Core Commands to Know

Get-Command

Show-Command

Get-Help

-ShowWindow

Get-Member

Update-Help

Update-Module

PowerCLI

- **800+** cmdlets in 28+ modules
- Install-Module VMware.PowerCLI
  - -Allow-Clobber, -Force & -SkipPublisherCheck may be necessary

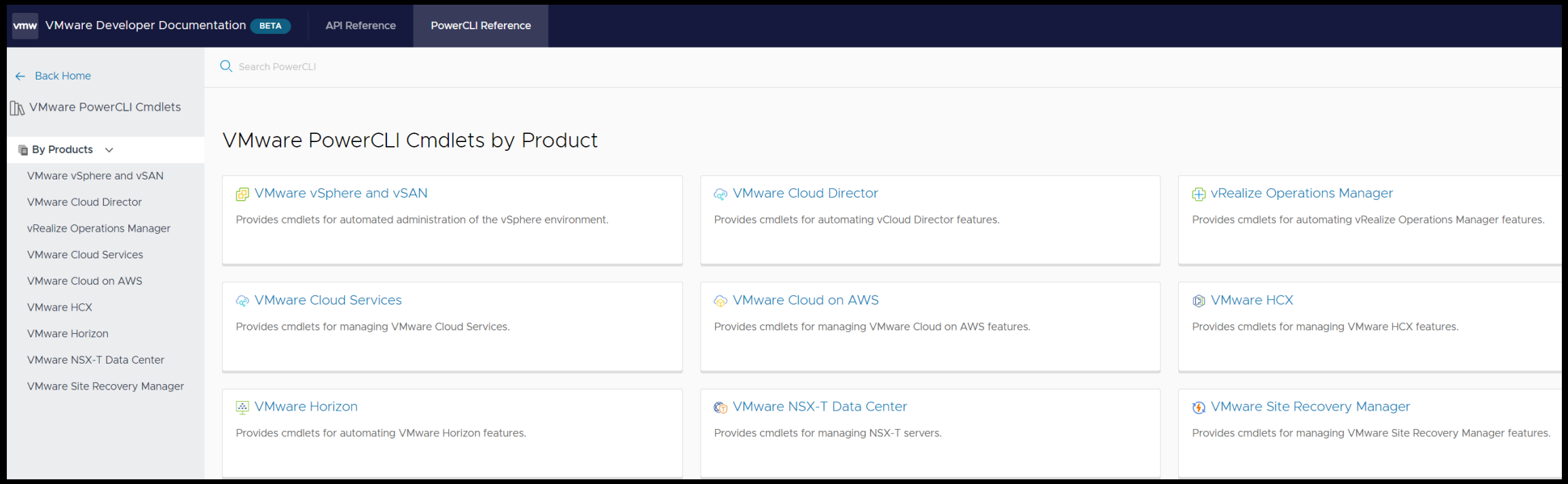
```
Install-Package: The following commands are already available on this
system:'Export-VM,Get-VM,Get-VMHost,Move-VM,New-VM,Remove-VM,Restart-VM,Set-VM,Set-VMHost,Start-VM,Stop-VM,Suspend-VM'. This module
'VMware.VimAutomation.Core' may override the existing commands. If you still want to install this module 'VMware.VimAutomation.Core', use
-AllowClobber parameter.
```

- Cmdlet collisions such as “Get-VM”
  - Import-Module VMware.PowerCLI –Prefix “Vmx”
    - Get-VM becomes Get-VmxVM
  - Use –NoClobber parameter

28 modules!

```
# Modules that must be imported into the global environment prior to importing this module
RequiredModules = @(
@{"ModuleName"="VMware.VimAutomation.Sdk";"ModuleVersion"="13.1.0.21605170"}
@{"ModuleName"="VMware.VimAutomation.Common";"ModuleVersion"="13.1.0.21605386"}
@{"ModuleName"="VMware.Vim";"ModuleVersion"="8.1.0.21605554"}
@{"ModuleName"="VMware.VimAutomation.Core";"ModuleVersion"="13.1.0.21606170"}
@{"ModuleName"="VMware.VimAutomation.Srm";"ModuleVersion"="12.7.0.20091290"}
@{"ModuleName"="VMware.VimAutomation.License";"ModuleVersion"="12.0.0.15939670"}
@{"ModuleName"="VMware.VimAutomation.Vds";"ModuleVersion"="13.1.0.21610933"}
@{"ModuleName"="VMware.CloudServices";"ModuleVersion"="12.6.0.19606210"}
@{"ModuleName"="VMware.VimAutomation.Vmc";"ModuleVersion"="13.0.0.20797723"}
@{"ModuleName"="VMware.VimAutomation.Nsxt";"ModuleVersion"="13.1.0.21606089"}
@{"ModuleName"="VMware.VimAutomation.vROps";"ModuleVersion"="13.1.0.21611158"}
@{"ModuleName"="VMware.VimAutomation.Cis.Core";"ModuleVersion"="13.1.0.21605976"}
@{"ModuleName"="VMware.VimAutomation.HorizonView";"ModuleVersion"="13.1.0.21610272"}
@{"ModuleName"="VMware.VimAutomation.Cloud";"ModuleVersion"="13.1.0.21611174"}
@{"ModuleName"="VMware.DeployAutomation";"ModuleVersion"="8.0.0.21610665"}
@{"ModuleName"="VMware.ImageBuilder";"ModuleVersion"="8.0.0.21610262"}
@{"ModuleName"="VMware.VimAutomation.Storage";"ModuleVersion"="13.1.0.21606282"}
@{"ModuleName"="VMware.VimAutomation.StorageUtility";"ModuleVersion"="1.6.0.0"}
@{"ModuleName"="VMware.VumAutomation";"ModuleVersion"="12.7.0.20091294"}
@{"ModuleName"="VMware.VimAutomation.Security";"ModuleVersion"="13.1.0.21606510"}
@{"ModuleName"="VMware.VimAutomation.Hcx";"ModuleVersion"="13.0.0.20803747"}
@{"ModuleName"="VMware.VimAutomation.WorkloadManagement";"ModuleVersion"="12.4.0.18627055"}
@{"ModuleName"="VMware.Sdk.Runtime";"ModuleVersion"="1.0.1111.21624264"}
@{"ModuleName"="VMware.Sdk.vSphere";"ModuleVersion"="8.0.1111.21624264"}
@{"ModuleName"="VMware.PowerCLI.VCenter";"ModuleVersion"="12.6.0.19600125"}
@{"ModuleName"="VMware.Sdk.Nsx.Policy";"ModuleVersion"="4.1.0.21605558"}
@{"ModuleName"="VMware.Sdk.Srm";"ModuleVersion"="8.7.0.21605564"}
@{"ModuleName"="VMware.Sdk.Vr";"ModuleVersion"="8.7.0.21605566"}
# @{"ModuleName"="VMware.Sdk.Vcf.CloudBuilder";"ModuleVersion"="0.0.0.0"}
# @{"ModuleName"="VMware.Sdk.Vcf.SddcManager";"ModuleVersion"="0.0.0.0"}
```

https://developer.vmware.com/docs/powercli/latest/products/



https://www.powershellgallery.com/packages?q=Tags%3A%22Powercli%22

Thank you!

@mikenelsonio