

PowerShell & PowerCLI

Starting from Scratch

Nashville VMUG UserCon
October 2022

Mike

 mikenelsonio

 nelmedia

 mikenelson-io

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





/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



Windows
Linux
MacOS



7.x

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.
- ▶ Profiles can be for all users, the current user, all hosts, and the current host. You can have a mix of none, some, or all of these and there is a precedence order.
- ▶ 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

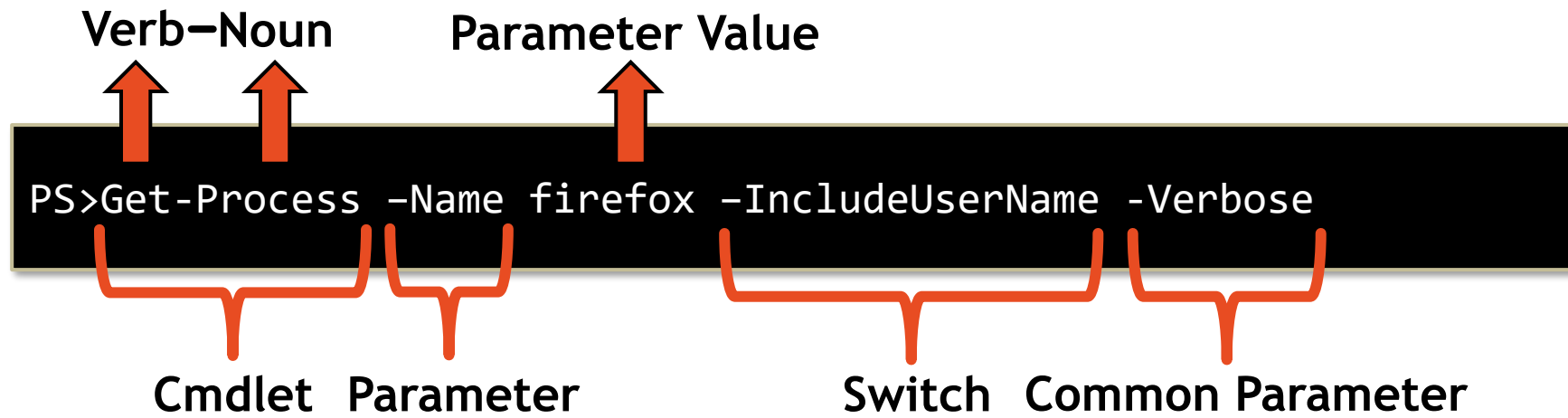
“command-lets”

- ▶ The “soul” of PowerShell
- ▶ A type of command in PowerShell
- ▶ Common syntax & options
- ▶ Usually take object input & return objects
- ▶ Stored in **.ps1** file for scripts

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

Syntax



Parameters

Named

Positional

Dynamic

Common

Sets

- ▶ Allow for users to provide input or options
- ▶ A pre-hyphen (“-”) is not always necessary (ie. positional)
- ▶ 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 firefox -IncludeUserName | Stop-Process
```

Object returned by first cmdlet sent to second cmdlet

“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
```

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
```

Functions

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

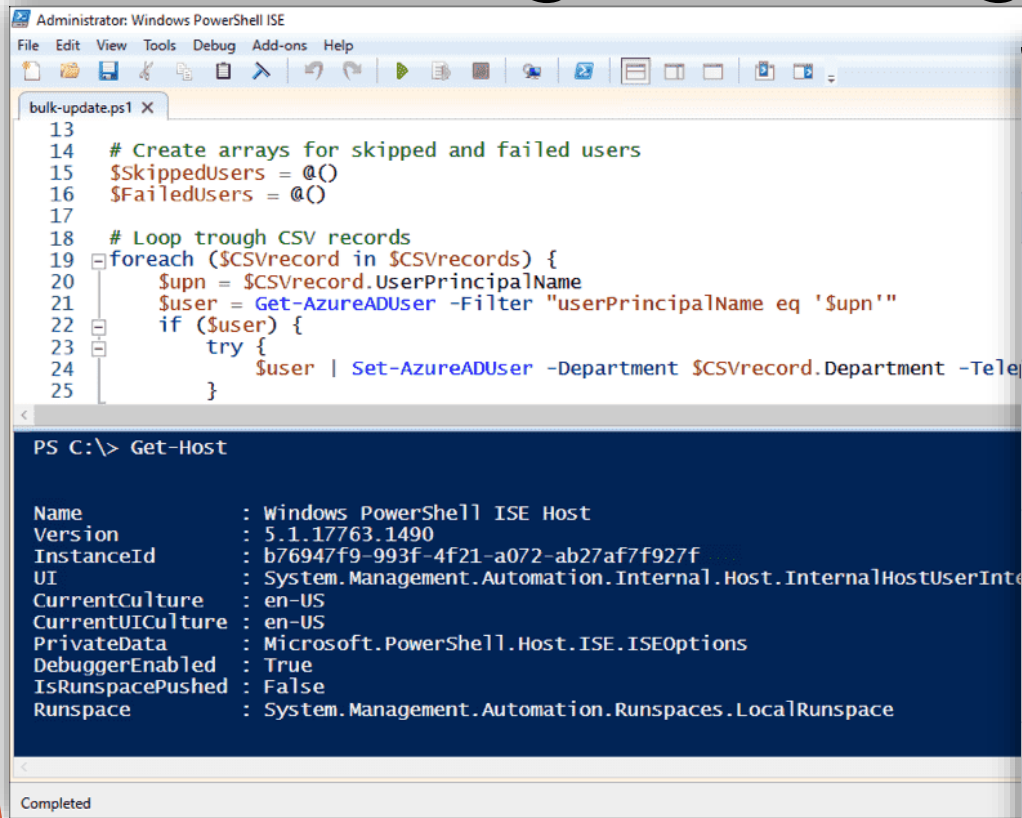
```
function Get-FirefoxProcess { Get-Process firefox }  
  
function Get-FirefoxProcess {  
    $a = Get-Process firefox  
    if ($a -eq $null) {  
        Write-Host "No Firefox process present"}  
    return $a  
}  
  
Get-FirefoxProcess
```

To run a function, simply “call” it.

Command Line, Scripts, & Modules

- Command line execution - simple, as-is, possibly pipelined
- Scripts are **.ps1** files
- Modules are **.psm1** files
 - Collection of commands, providers, variables, functions, help context, aliases, workflows, etc.
 - Can be imported from a **.psd1** manifest file. Basically a definition file.

Creating / Editing / Running



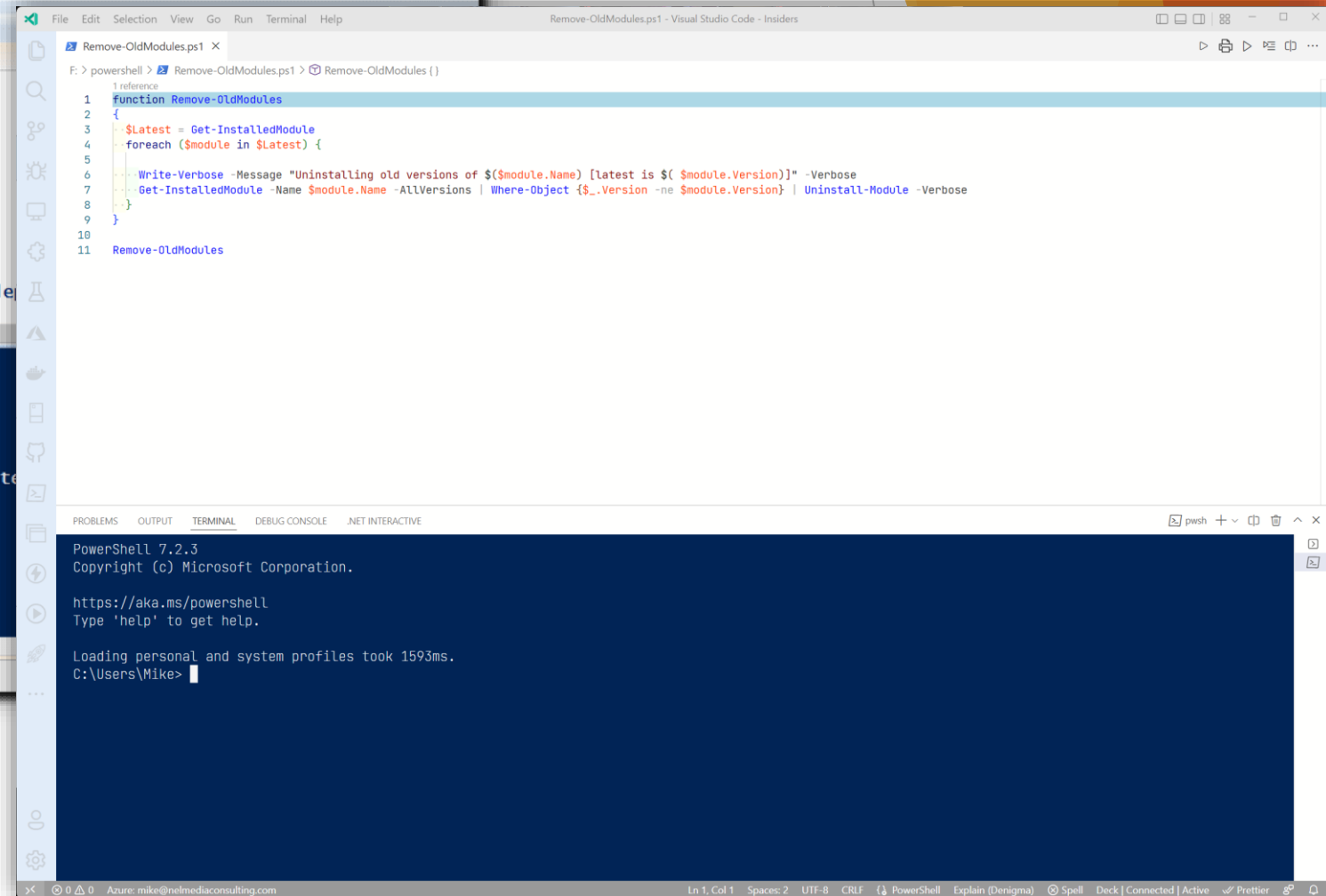
The screenshot shows the Windows PowerShell ISE interface. The top pane displays a PowerShell script named `bulk-update.ps1`. The script includes comments and logic for processing CSV records, creating arrays for skipped and failed users, and looping through CSV records to update Azure AD users. The bottom pane shows the output of the `Get-Host` command, displaying host information such as Name, Version, InstanceId, UI, CurrentCulture, CurrentUICulture, PrivateData, DebuggerEnabled, IsRunspacePushed, and Runspace.

```
13
14 # Create arrays for skipped and failed users
15 $SkippedUsers = @()
16 $FailedUsers = @()
17
18 # Loop through CSV records
19 foreach ($CSVrecord in $CSVrecords) {
20     $supn = $CSVrecord.UserPrincipalName
21     $user = Get-AzureADUser -Filter "userPrincipalName eq '$supn'"
22     if ($user) {
23         try {
24             $user | Set-AzureADUser -Department $CSVrecord.Department -Tele
25         }
26     }
27 }
```

```
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
```

Integrated Scripting Environment (ISE)



The screenshot shows the Visual Studio Code interface. The top pane displays a PowerShell script named `Remove-OldModules.ps1`. The script defines a function `Remove-OldModules` that iterates through installed modules and removes older versions. The bottom pane shows the output of the `Remove-OldModules` function, displaying the PowerShell version, copyright information, and the current directory.

```
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
```

```
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>
```

Visual Studio Code (VSCode)

* Use VSCodium for security-minded folks

Core Commands to Know

Get-Command

Get-Help

Get-Member

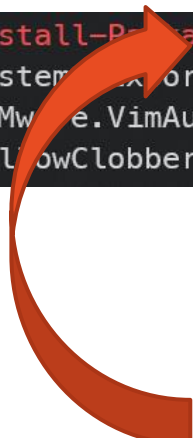
Show-Command

Update-Help

Update-Module

PowerCLI

- Install-Module VMware.PowerCLI
 - -Allow-Clobber, -Force & -SkipPublisherCheck may be necessary



```
Install-Module: 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.
```

- If certificate error, Update-Module will not work
- Cmdlet collisions such as Get-VM
- 800+ cmdlets in 15+ modules

```
# Modules that must be imported into the global environment prior to importing this module
RequiredModules = @(
@{"ModuleName"="VMware.VimAutomation.Sdk";"ModuleVersion"="12.5.0.19093564"}
@{"ModuleName"="VMware.VimAutomation.Common";"ModuleVersion"="12.6.0.19600917"}
@{"ModuleName"="VMware.Vim";"ModuleVersion"="7.0.3.19601056"}
@{"ModuleName"="VMware.VimAutomation.Core";"ModuleVersion"="12.6.0.19601570"}
@{"ModuleName"="VMware.VimAutomation.Srm";"ModuleVersion"="12.6.0.19609133"}
@{"ModuleName"="VMware.VimAutomation.License";"ModuleVersion"="12.0.0.15939670"}
@{"ModuleName"="VMware.VimAutomation.Vds";"ModuleVersion"="12.5.0.19167830"}
@{"ModuleName"="VMware.CloudServices";"ModuleVersion"="12.6.0.19606210"}
@{"ModuleName"="VMware.VimAutomation.Vmc";"ModuleVersion"="12.6.0.19609014"}
@{"ModuleName"="VMware.VimAutomation.Nsxt";"ModuleVersion"="12.5.0.19168180"}
@{"ModuleName"="VMware.VimAutomation.vROps";"ModuleVersion"="12.5.0.19167825"}
@{"ModuleName"="VMware.VimAutomation.Cis.Core";"ModuleVersion"="12.6.0.19601368"}
@{"ModuleName"="VMware.VimAutomation.HorizonView";"ModuleVersion"="12.5.0.19033914"}
@{"ModuleName"="VMware.VimAutomation.Cloud";"ModuleVersion"="12.0.0.15940183"}
@{"ModuleName"="VMware.DeployAutomation";"ModuleVersion"="7.0.3.19599828"}
@{"ModuleName"="VMware.ImageBuilder";"ModuleVersion"="7.0.3.19599828"}
@{"ModuleName"="VMware.VimAutomation.Storage";"ModuleVersion"="12.6.0.19609013"}
@{"ModuleName"="VMware.VimAutomation.StorageUtility";"ModuleVersion"="1.6.0.0"}
@{"ModuleName"="VMware.VumAutomation";"ModuleVersion"="12.1.0.16941488"}
@{"ModuleName"="VMware.VimAutomation.Security";"ModuleVersion"="12.3.0.17833870"}
@{"ModuleName"="VMware.VimAutomation.Hcx";"ModuleVersion"="12.6.0.19606303"}
@{"ModuleName"="VMware.VimAutomation.WorkloadManagement";"ModuleVersion"="12.4.0.18627055"}
@{"ModuleName"="VMware.Sdk.Runtime";"ModuleVersion"="1.0.106.18628394"}
@{"ModuleName"="VMware.Sdk.vSphere";"ModuleVersion"="1.0.104.18678708"}
@{"ModuleName"="VMware.PowerCLI.VCenter";"ModuleVersion"="12.6.0.19600125"}
@{"ModuleName"="VMware.Sdk.Nsx.Policy";"ModuleVersion"="3.2.0.19610335"}
)
```

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

vmw

VMware Developer Documentation BETA

API Reference

PowerCLI Reference

← Back Home

VMware PowerCLI Cmdlets

By Products ▾

VMware vSphere and vSAN

VMware Cloud Director

vRealize Operations Manager

VMware Cloud Services

VMware Cloud on AWS

VMware HCX

VMware Horizon

VMware NSX-T Data Center

VMware Site Recovery Manager

Search PowerCLI

VMware PowerCLI Cmdlets by Product

VMware vSphere and vSAN

Provides cmdlets for automated administration of the vSphere environment.

VMware Cloud Director

Provides cmdlets for automating vCloud Director features.

vRealize Operations Manager

Provides cmdlets for automating vRealize Operations Manager features.

VMware Cloud Services

Provides cmdlets for managing VMware Cloud Services.

VMware Cloud on AWS

Provides cmdlets for managing VMware Cloud on AWS features.

VMware HCX

Provides cmdlets for managing VMware HCX features.

VMware Horizon

Provides cmdlets for automating VMware Horizon features.


VMware NSX-T Data Center

Provides cmdlets for managing NSX-T servers.

VMware Site Recovery Manager

Provides cmdlets for managing VMware Site Recovery Manager features.

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

 PURE STORAGE

Demos

Thank you!

@mikenelsonio