PowerCLI Workshop
From Beginner to Advanced

Luc Dekens
Kyle Ruddy

Luc Dekens

PowerCLI Mastermind

Writer @ lucd.info
GitHub @ github.com/lucdekens
Twitter @lucd22
MS MVP Cloud\Datacenter Management

Kyle Ruddy

Senior Architect, Technical Marketing

Writer @ kmruddy.com
GitHub @ github.com/kmruddy
Podcast @ vBrownBag.com
Twitter @kmruddy
MS MVP Cloud\Datacenter Management

Agenda

- What are PowerShell and PowerCLI?
- The Lingo Dictionary
- Setup and Configuration
- Starting to Code
- Writing Logic Statements
- Lab Time

What are PowerShell and PowerCLI?

"Windows PowerShell is a **task automation** and **configuration management framework** from Microsoft, consisting of a command-line shell and associated **scripting language** built on the .NET Framework."

https://en.wikipedia.org/wiki/Windows_PowerShell

PowerShell

A simple and straight-forward path to automation

- Already installed on all modern Windows Operating Systems
- Integrated and rich help system

PowerShell 6.0 (aka. Core) available for Linux and MacOS

- https://github.com/PowerShell/PowerShell

PowerShell

Modular and object-oriented

- The best of a programming language melded with a scripting language
- True portability of code via modules (and snap-ins)
- Objects = Properties + Methods

VMware PowerCLI

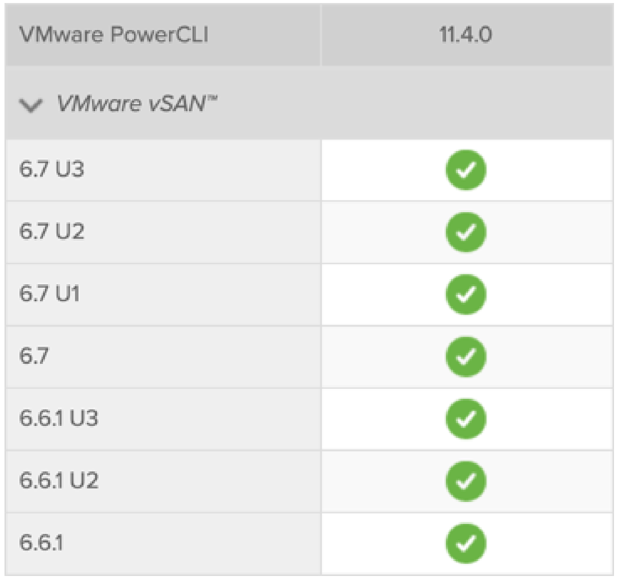
VMware's command-line and scripting tool built on Windows PowerShell

Features more than 700 cmdlets for managing and automating vSphere, vCloud, and Horizon environments

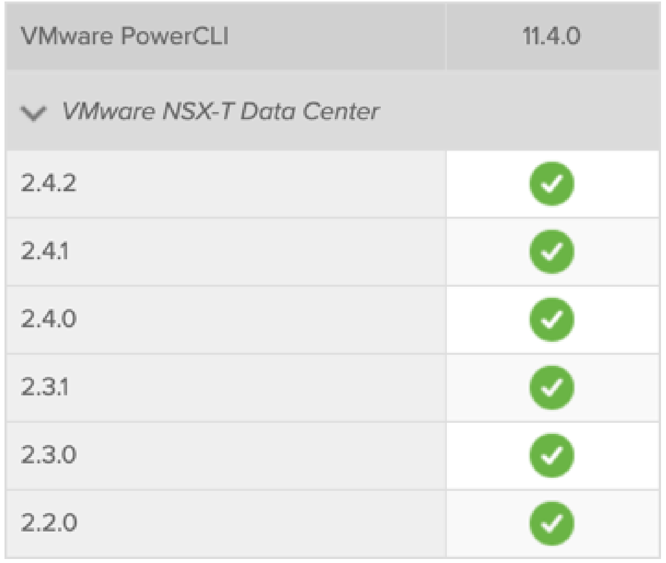
One of the most robust and complete PowerShell deployments in the world

Over 10 years "young"!

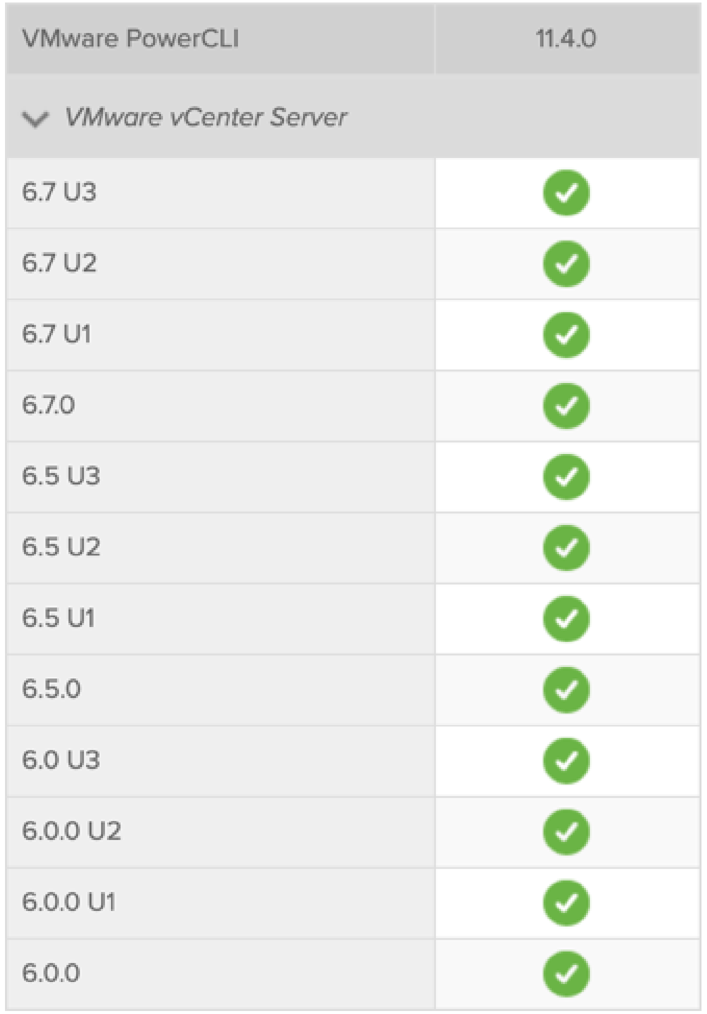
PowerCLI Compatibility



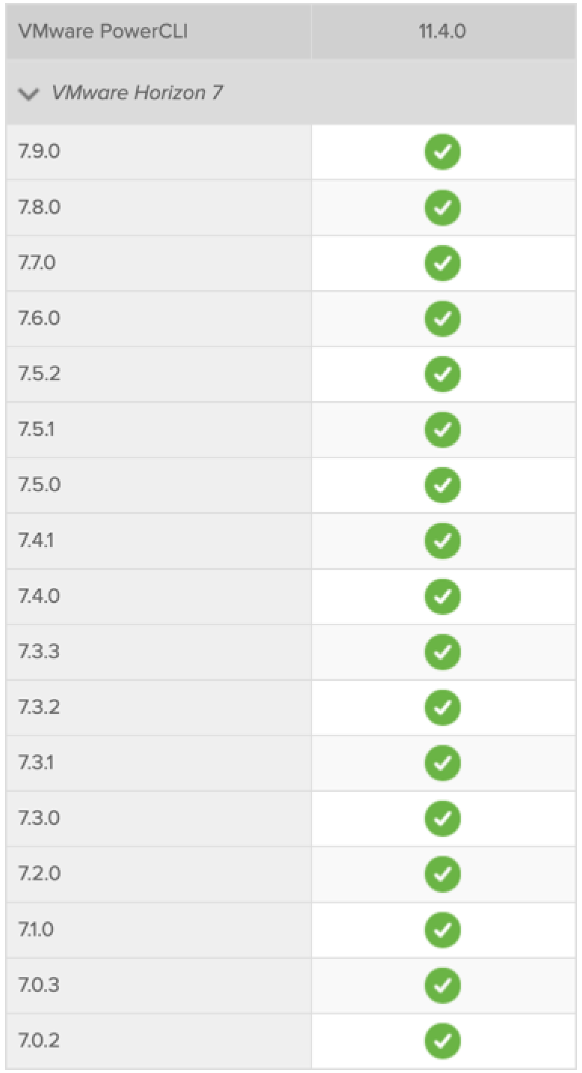
VMware PowerCLI	11.4.0
VMware vSAN™	
6.7 U3	✓
6.7 U2	✓
6.7 U1	✓
6.7	✓
6.6.1 U3	✓
6.6.1 U2	✓
6.6.1	✓



VMware PowerCLI	11.4.0
VMware NSX-T Data Center	
2.4.2	✓
2.4.1	✓
2.4.0	✓
2.3.1	✓
2.3.0	✓
2.2.0	✓



VMware PowerCLI	11.4.0
VMware vCenter Server	
6.7 U3	✓
6.7 U2	✓
6.7 U1	✓
6.7.0	✓
6.5 U3	✓
6.5 U2	✓
6.5 U1	✓
6.5.0	✓
6.0 U3	✓
6.0.0 U2	✓
6.0.0 U1	✓
6.0.0	✓



VMware PowerCLI	11.4.0
VMware Horizon 7	
7.9.0	✓
7.8.0	✓
7.7.0	✓
7.6.0	✓
7.5.2	✓
7.5.1	✓
7.5.0	✓
7.4.1	✓
7.4.0	✓
7.3.3	✓
7.3.2	✓
7.3.1	✓
7.3.0	✓
7.2.0	✓
7.1.0	✓
7.0.3	✓
7.0.2	✓

The Lingo Dictionary

Common Terms

- Cmdlet ("Command let")
  - A single command
  - Usually written and compiled in .NET
- Function
  - A single command
  - Usually written in PowerShell
- Script
  - Series of commands stored in a PS1 file
- Module
  - Package of related commands

Command Structure

- Cmdlets and Functions use properly formatted verbs
  - Use Get-Verb to see the available options
- Most features follow a very simple pattern
  - Get = Gather data
  - Set = Change data
  - New = Create data
  - Remove = Delete data

Object Management

- Variable
  - –Saves objects for later reference
  - –Uses ‘$’ as the initial character
- Example: $Name = “Kyle Ruddy”

- Pipeline
  - –Passes objects from one command to the next
  - –Declared with ‘|’ character
  - –Example: Get-User –Name $Name | Get-Beard

PowerCLI Object Management

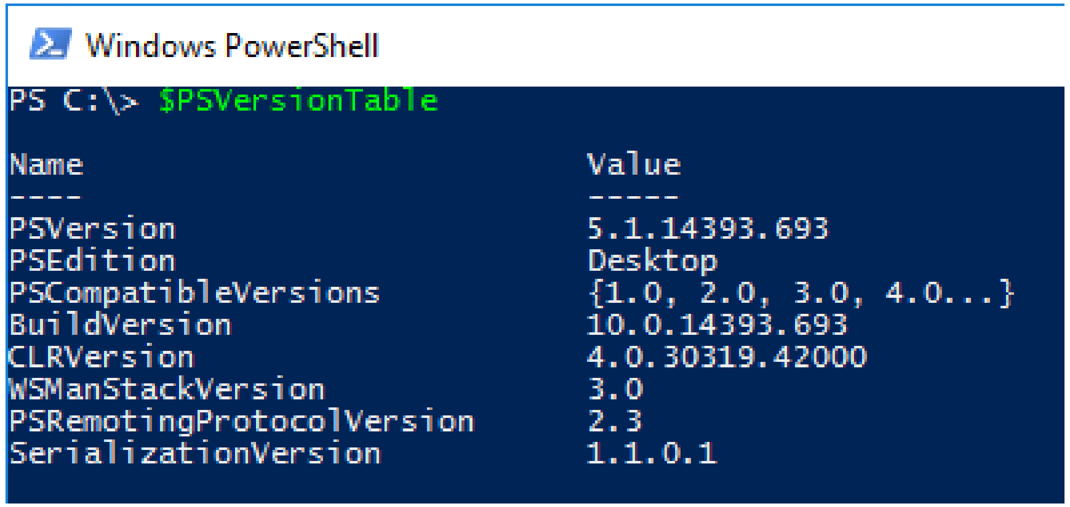
- PowerShell and PowerCLI work with objects
- Not all objects are equal
  - .Net objects
  - vSphere objects

.Net objects	vSphere objects
PowerCLI cmdlet	PowerCLI cmdlet – indirect
Selection of properties	All vSphere properties
PowerCLI methods	All vSphere methods
Get-Help/Get-Command	API Reference
Starter + Intermediate	Intermediate + Advanced

Setup and Configuration

Setting Up PowerShell

- Native to all modern Windows deployments
- Latest version is 5.1
  - –Use $PSVersionTable to see what you're running

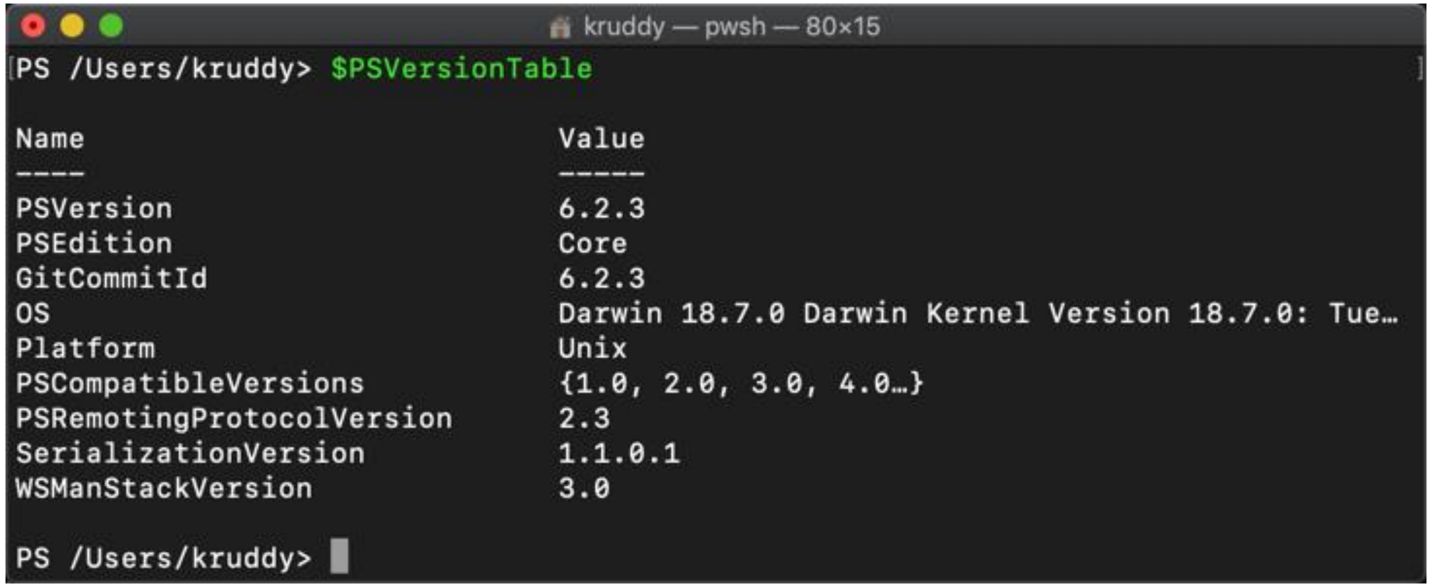


Windows PowerShell

```
PS C:\> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.1.14393.693
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.14393.693
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

Setting Up PowerShell Core

- Available through most package managers
- Latest version is 6.2.3
  - –Use $PSVersionTable to see what you're running



```
PS /Users/kruddy> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      6.2.3
PSEdition                      Core
GitCommitId                    6.2.3
OS                             Darwin 18.7.0 Darwin Kernel Version 18.7.0: Tue…
Platform                       Unix
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0…}
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
WSManStackVersion              3.0

PS /Users/kruddy>
```

Installation of PowerCLI is from the PowerShell Gallery and is done directly from within PowerShell

Prerequisites:

1. PowerShell 5.x (OR manually install PowerShellGet)
2. Uninstall PowerCLI 6.5 R1 (OR any other previous versions)
3. Internet Connectivity

Installation:

1. Open a PowerShell Session
2. Run: Install-Module VMware.PowerCLI -Scope CurrentUser

Modules are deployed to: $home\Documents\WindowsPowerShell\Modules

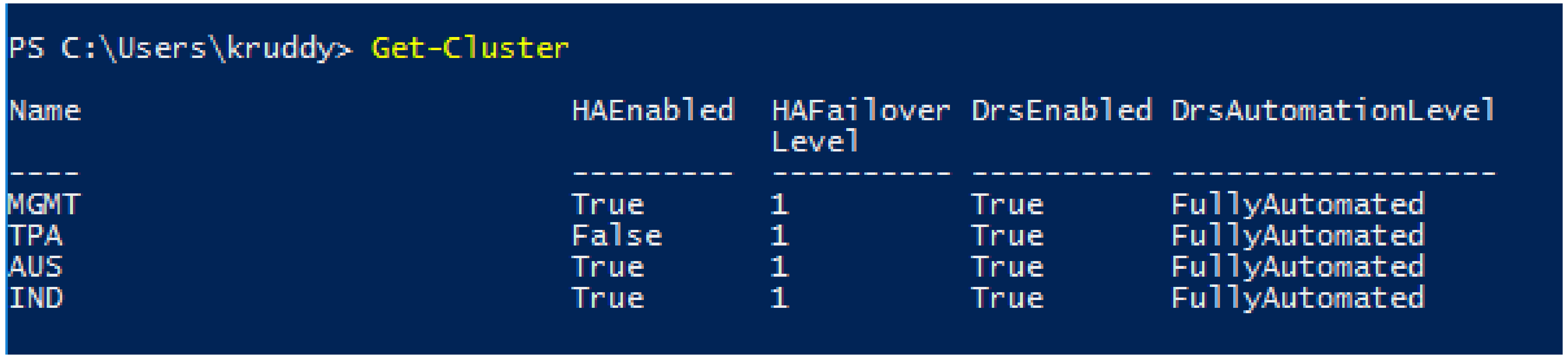
Starting to Code

Safe vs Non-Safe Commands

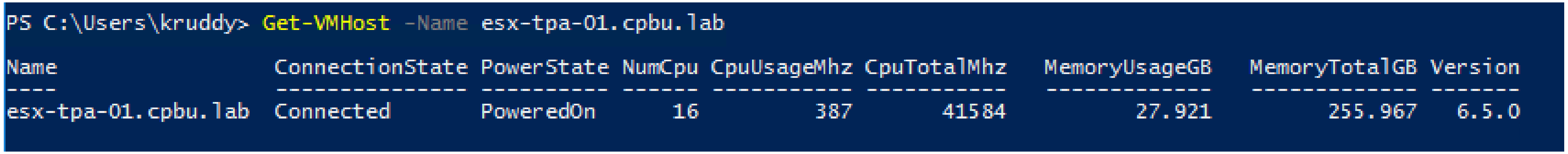
- Suggest starting with cmdlets that pull or display data
  - These are “Safe” in that they are unable to modify data
- This is helpful for learning the PowerShell syntax
  - Which… is a never ending journey!

Identify Safe Commands

Safe means that there is no data modification



```
PS C:\Users\kruddy> Get-Cluster

Name                           HAEnabled  HAFailover DrsEnabled DrsAutomationLevel
                                          Level
----                           ---------  ---------- ---------- ------------------
MGMT                           True       1          True       FullyAutomated
TPA                            False      1          True       FullyAutomated
AUS                            True       1          True       FullyAutomated
IND                            True       1          True       FullyAutomated
```

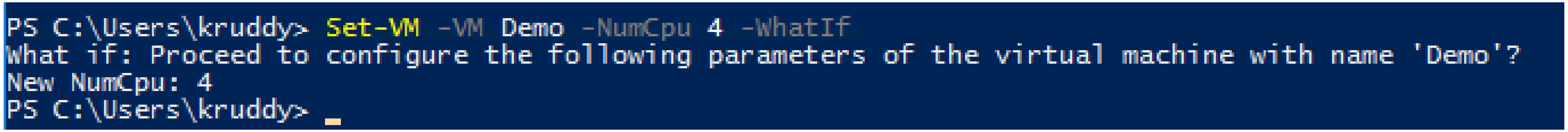


```
PS C:\Users\kruddy> Get-VMHost -Name esx-tpa-01.cpbu.lab

Name                 ConnectionState PowerState NumCpu CpuUsageMhz CpuTotalMhz   MemoryUsageGB   MemoryTotalGB Version
----                 --------------- ---------- ------ ----------- -----------   -------------   ------------- -------
esx-tpa-01.cpbu.lab  Connected       PoweredOn      16         387       41584          27.921         255.967   6.5.0
```

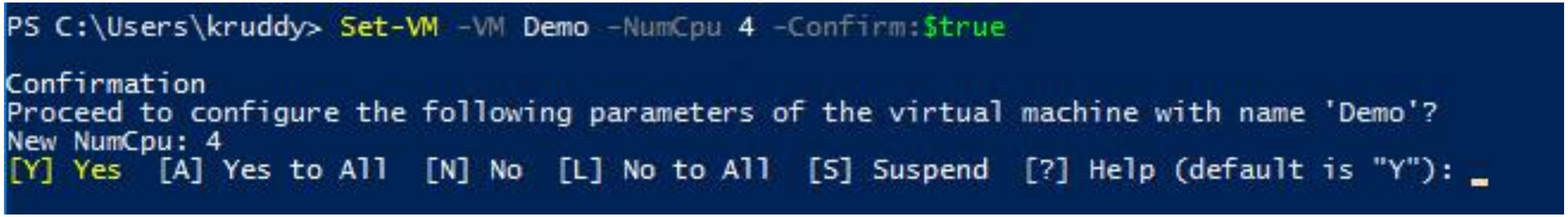
Additional Safeguards

- Whatif
  - Shows you what WOULD happen without actually modifying data
  - Switch
- Confirm
  - Asks you to confirm before any changes are made
  - Boolean ($true or $false)

Additional Safeguards



```
PS C:\Users\kruddy> Set-VM -VM Demo -NumCpu 4 -WhatIf
What if: Proceed to configure the following parameters of the virtual machine with name 'Demo'?
New NumCpu: 4
PS C:\Users\kruddy> _
```



```
PS C:\Users\kruddy> Set-VM -VM Demo -NumCpu 4 -Confirm:$true

Confirmation
Proceed to configure the following parameters of the virtual machine with name 'Demo'?
New NumCpu: 4
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): _
```

Writing Logic Statements

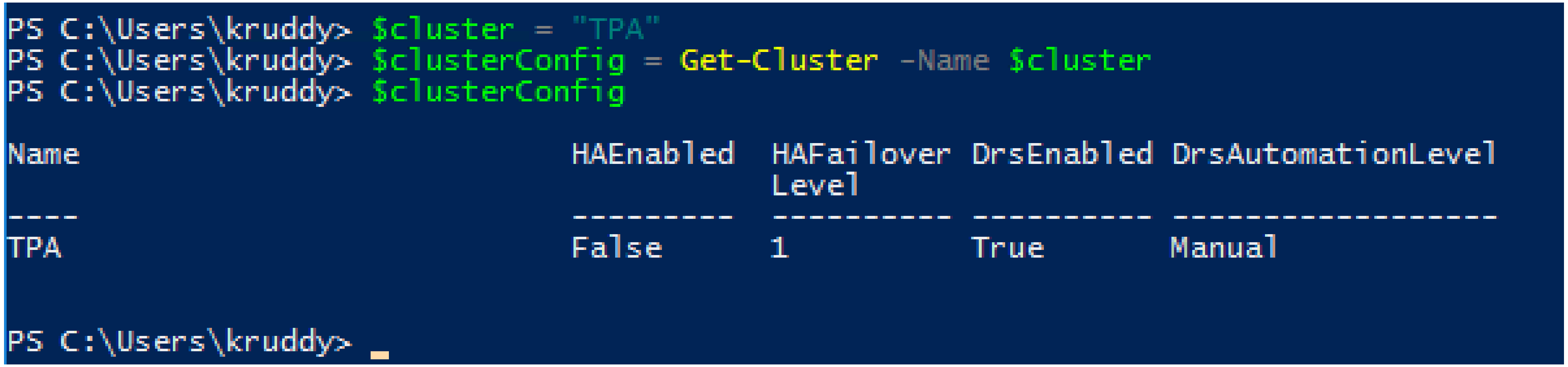
Sample Use Case

- Cluster configuration
  - What values are currently configured?
  - Changing a few of them to match our desired state
- DRS Settings
  - HA is Enabled
  - DRS Automation Level is Fully Automated

Gathering Cluster Data

```
# Variables
$cluster = 'Cluster Name'

# Gather Cluster Data
$clusterConfig = Get-Cluster –Name $cluster
```

Gathering Cluster Data

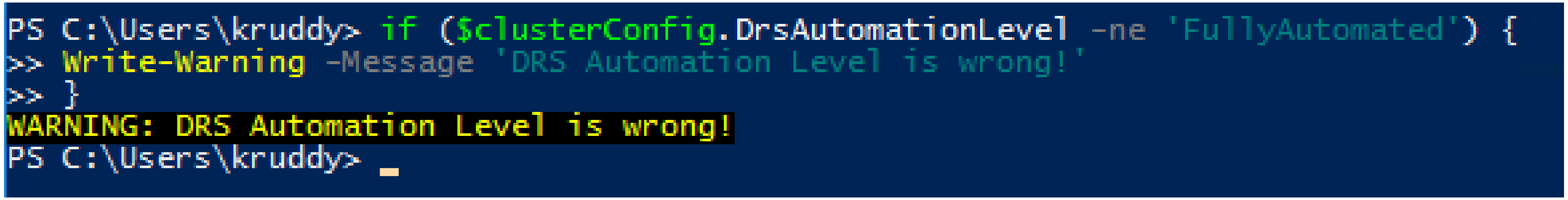


```
PS C:\Users\kruddy> $cluster = "TPA"
PS C:\Users\kruddy> $clusterConfig = Get-Cluster -Name $cluster
PS C:\Users\kruddy> $clusterConfig

Name                           HAEnabled  HAFailover DrsEnabled DrsAutomationLevel
                                          Level
----                           ---------  ---------- ---------- ------------------
TPA                            False      1          True       Manual


PS C:\Users\kruddy> _
```

Decision Making Logic

```
#Adding logic to test values
if ($clusterConfig.DrsAutomationLevel –ne ‘FullyAutomated’) {

    # Display a warning message indicating the DRS Automation level is incorrect
    Write-Warning –Message ‘DRS Automation Level is wrong!’

}
```

Decision Making Logic



```
PS C:\Users\kruddy> if ($clusterConfig.DrsAutomationLevel -ne 'FullyAutomated') {
>> Write-Warning -Message 'DRS Automation Level is wrong!'
>> }
WARNING: DRS Automation Level is wrong!
PS C:\Users\kruddy> _
```

PSCONF.ASIA

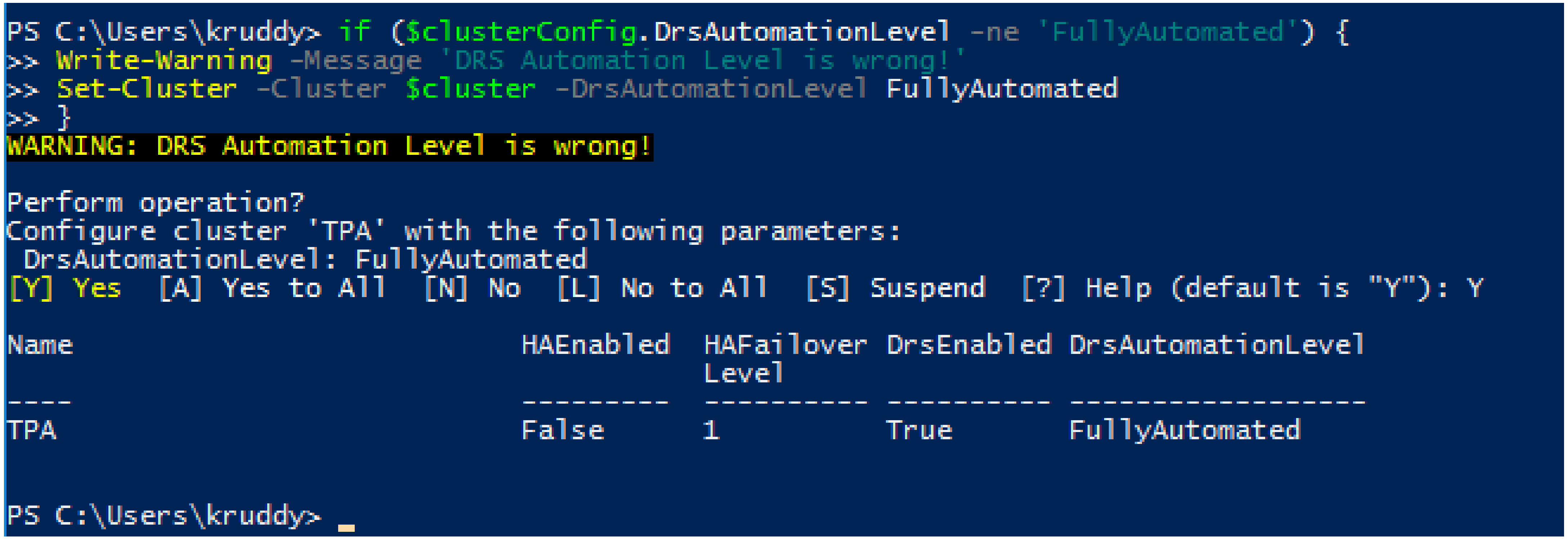
Decision Making Logic

```
# Add logic to modify these values
if ($clusterConfig.DrsAutomationLevel –ne ‘FullyAutomated’) {

    # Display a warning message that the DRS Automation level is incorrect
    Write-Warning –Message ‘DRS Automation Level is wrong!’
    # Change the cluster configuration to be the proper level for DRS
    Set-Cluster –Cluster $cluster –DrsAutomationlevel FullyAutomated

}
```

master* 0↓ 1↑ |Ln 1, Col 1 |Spaces: 4 |UTF-8 |CRLF |5.1

PSCONF.ASIA

Gathering Cluster Data



```
PS C:\Users\kruddy> if ($clusterConfig.DrsAutomationLevel -ne 'FullyAutomated') {
>> Write-Warning -Message 'DRS Automation Level is wrong!'
>> Set-Cluster -Cluster $cluster -DrsAutomationLevel FullyAutomated
>> }
WARNING: DRS Automation Level is wrong!

Perform operation?
Configure cluster 'TPA' with the following parameters:
 DrsAutomationLevel: FullyAutomated
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

Name                           HAEnabled  HAFailover DrsEnabled DrsAutomationLevel
                                          Level
----                           ---------  ---------- ---------- ------------------
TPA                            False      1          True       FullyAutomated


PS C:\Users\kruddy> _
```

master* 0↓ 1↑ |Ln 1, Col 1 |Spaces: 4 |UTF-8 |CRLF |5.1

Lab Time!

VMware Hands On Labs

The lab for today's workshop:

HOL-2012-01-SDC - VMware vSphere Automation – PowerCLI

Link: https://labs.hol.vmware.com/HOL/

Even after today, if you don't have your own lab - Use ours!

More Advanced Topics

Try-Catch-Finally

- The “forced” terminating error
- Use Try-Catch(-Finally)
  - Part of your repertoire!

```
$vNic = Get-VM -Name $vmName | Get-NetworkAdapter -Name $nicName
if (-not $vNic)
{
  $vNic = New-NetworkAdapter -VM $vmName -NetworkName $pgName -Type $nicType
}
elseif ($vNic.Type -ne $nicType)
{
  $vNic = Set-NetworkAdapter -NetworkAdapter $vNic -Type $nicType -Confirm:$false
}
```

Try-Catch-Finally

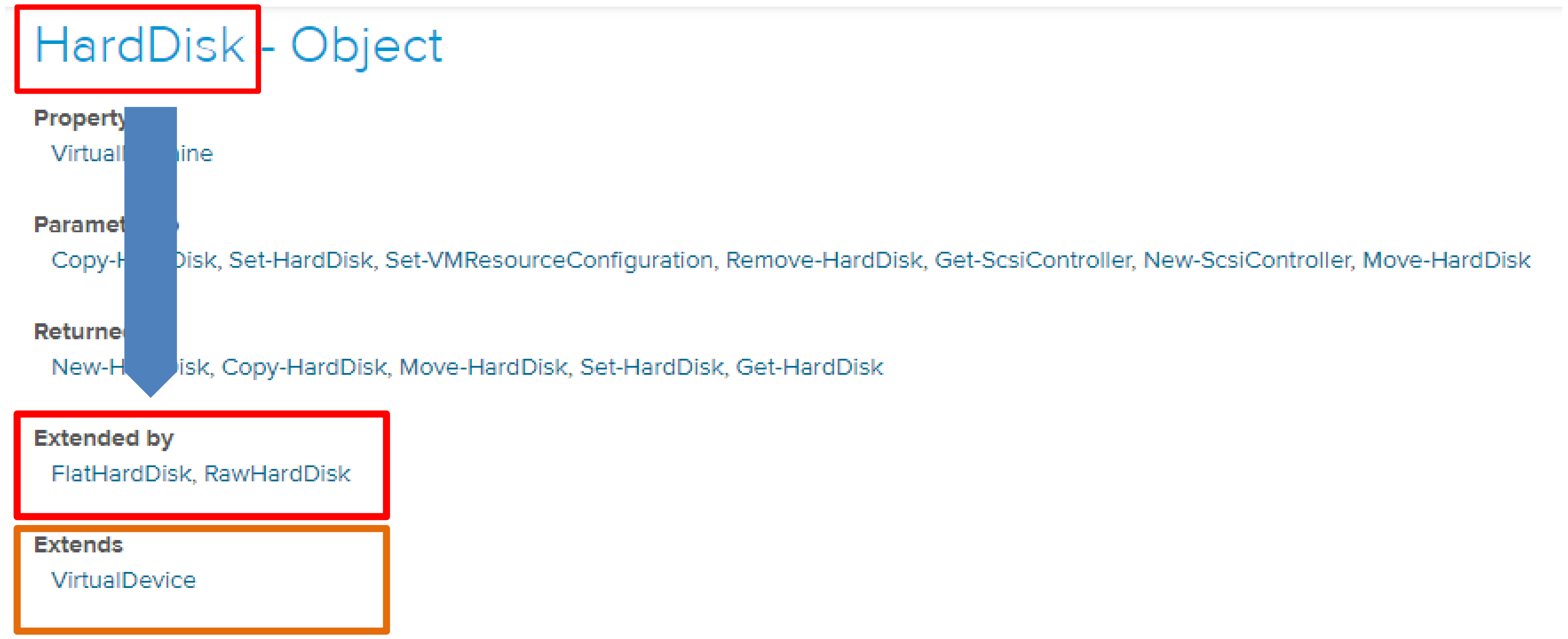
```
try
{
  $vNic = Get-VM -Name $vmName |
  Get-NetworkAdapter -Name $nicName -ErrorAction Stop |
  Set-NetworkAdapter -NetworkName $pgName -Type $nicType
}
catch
{
  $vNic = Get-VM -Name $vmName |
  New-NetworkAdapter -NetworkName $pgName -Type $nicType
}
```

Try-Catch-Finally

```
try
{
  $vNic = Get-VM -Name $vmName |
     Get-NetworkAdapter -Name $nicName -ErrorAction Stop
}
catch
{
  $vNic = Get-VM -Name $vmName |
     New-NetworkAdapter -NetworkName $pgName
}
finally
{
  Set-NetworkAdapter -NetworkAdapter $vNic -WakeOnLan:$true -Confirm:$false
}
```

Inheritance

- No, not that kind!
- Most objects use an inheritance scheme



HardDisk - Object

Property

VirtualI ine

Paramet

Copy-H Disk, Set-HardDisk, Set-VMResourceConfiguration, Remove-HardDisk, Get-ScsiController, New-ScsiController, Move-HardDisk

Returne

New-H isk, Copy-HardDisk, Move-HardDisk, Set-HardDisk, Get-HardDisk

Extended by

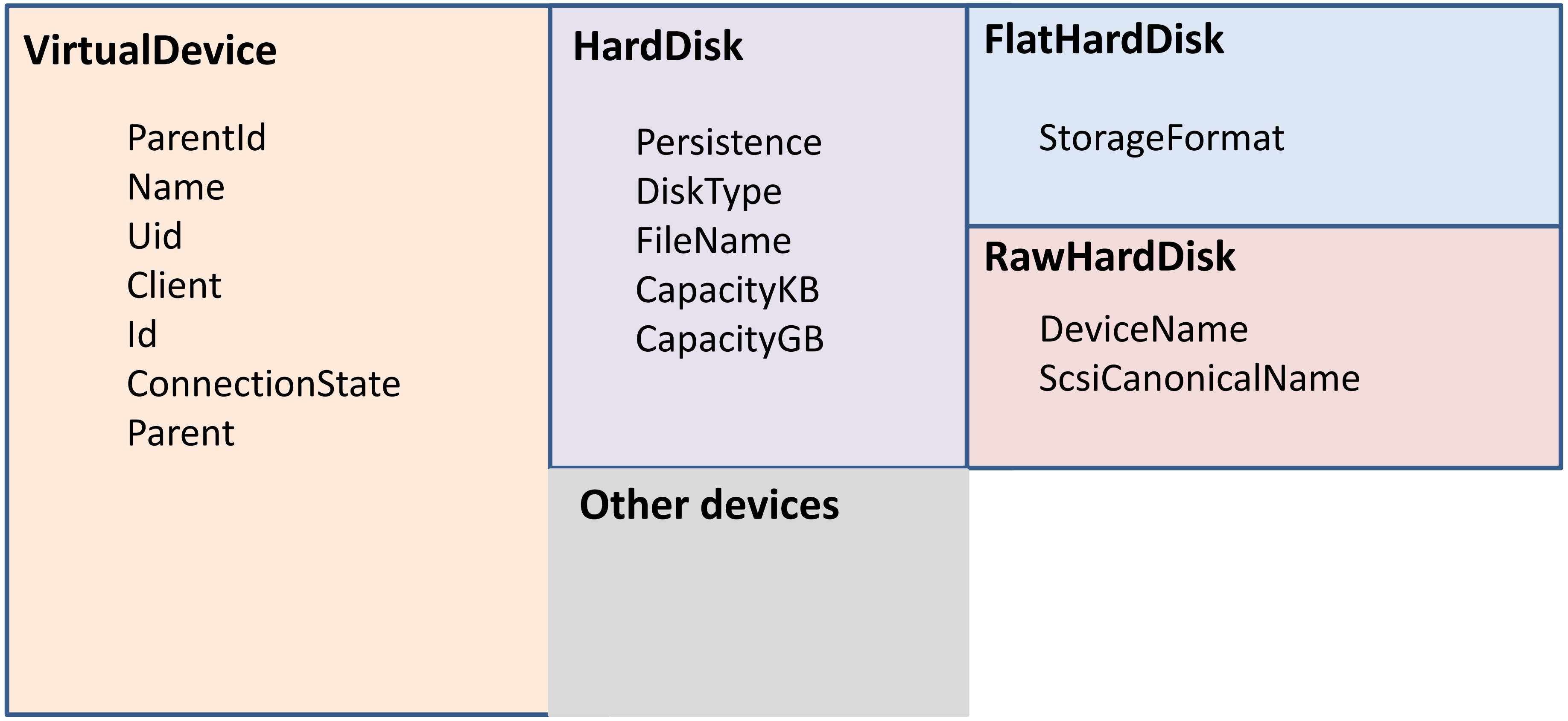
FlatHardDisk, RawHardDisk

Extends

VirtualDevice

Interitance

- Chain of objects, sum of properties



Inheritance

- What is in it for me?
- Simplify your code
- Understand why some code will not work
  - … and how to fix it

Inheritance

```
Get-VM -Name VM1 | Get-HardDisk |
Where-Object {
  $_ -is
   [VMware.VimAutomation.ViCore.Types.V1.VirtualDevice.FlatHardDisk] -or
  $_ -is
   [VMware.VimAutomation.ViCore.Types.V1.VirtualDevice.RawHardDisk]
}
```

```
Get-VM -Name VM1 | Get-HardDisk |
Where-Object {
  $_ -is
   [VMware.VimAutomation.ViCore.Types.V1.VirtualDevice.VirtualDisk]
}
```

PSCONF.ASIA

Inheritance

```
Get-VM -Name VM1 | Get-HardDisk | `
     Select-Object -Property Name, FileName, ScsiCanonicalName
```

```
Name        Filename                                                 ScsiCanonicalName
----        --------                                                 -----------------
Hard disk 1 [vsanDatastore] 6b44515d-2237-11e1-2646-0050569cd2ee/VM1.vmdk
Hard disk 2 [vsanDatastore] 6b44515d-2237-11e1-2646-0050569cd2ee/VM1_1.vmdk naa.60003ff44dc75adc8a8545aa4f696c11
```

```
Get-VM -Name VM1 | Get-HardDisk | Export-Csv -Path .\report.csv
Import-csv -Path .\report.csv | `
          Select-Object -Property Name, FileName, CanonicalName
```

```
Name        Filename                                                   CanonicalName
----        --------                                                   -------------
Hard disk 1 [vsanDatastore] 6b44515d-2237-11e1-2646-0050569cd2ee/VM1.vmdk
                                                                             ?
Hard disk 2 [vsanDatastore] 6b44515d-2237-11e1-2646-0050569cd2ee/VM1_1.vmdk
```

master* 0↓ 1↑ |Ln 1, Col 1 |Spaces: 4 |UTF-8 |CRLF | 5.1

Inheritance

- Export-Csv takes the 1st object as template, if no explicit properties given

```
$hd = Get-VM -Name VM1 | Get-HardDisk
$props = $hd | % { $_.psobject.properties.name } | Sort-Object -Unique
$hd | Select-Object -Property $props | Export-Csv -Path .\report.csv

Import-csv -Path .\report.csv | `
      Select-Object -Property Name, FileName, ScsiCanonicalName
```

```
Name        Filename                                                    ScsiCanonicalName
----        --------                                                    -----------------
Hard disk 1 [vsanDatastore] 6b44515d-2237-11e1-2646-0050569cd2ee/VM1.vmdk
Hard disk 2 [vsanDatastore] 6b44515d-2237-11e1-2646-0050569cd2ee/VM1_1.vmdk naa.60003ff44dc75adc8a8545aa4f696c11
```

Wait For It

- Template + OSCustomizationSpec ➔ VM
- But when is it really available?
  – Use the Events (Luke)!
- Simple VM deployment

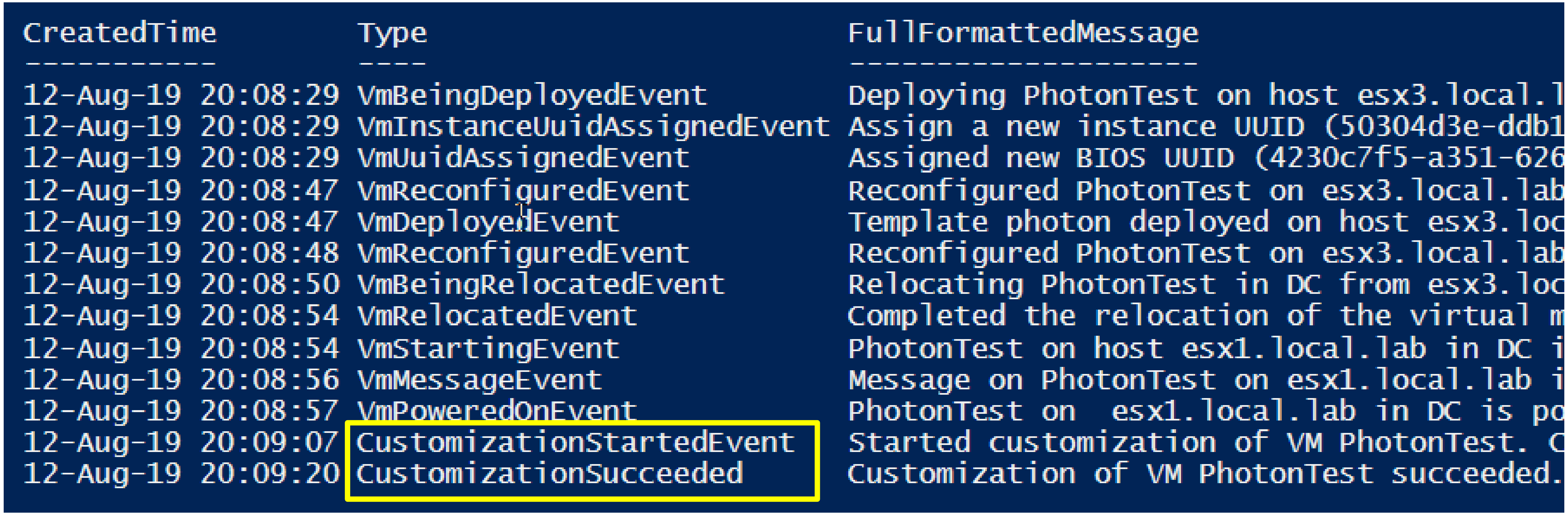
```
$sVM = @{
  Name = $vmName
  Template = Get-Template -Name $templateName
  ResourcePool = Get-Cluster -Name $clusterName
  Datastore = Get-Datastore -Name $dsName
  OSCustomizationSpec = Get-OSCustomizationSpec -Name $custName
}
New-VM @sVM | Start-VM -Confirm:$false
```

Wait For It

- Analyze the timeline

```
$sEvent = @{
  Entity = Get-VM -Name $vmName
  MaxSamples = [int]::MaxValue
  Start = (Get-Date).AddMinutes(-30)
}

Get-VIEvent @sEvent |
where{$_ -is [VMware.Vim.VmEvent]} |
Sort-Object -Property CreatedTime |
Select CreatedTime,@{N='Type';E={$_.GetType().Name}},FullFormattedMessage
```

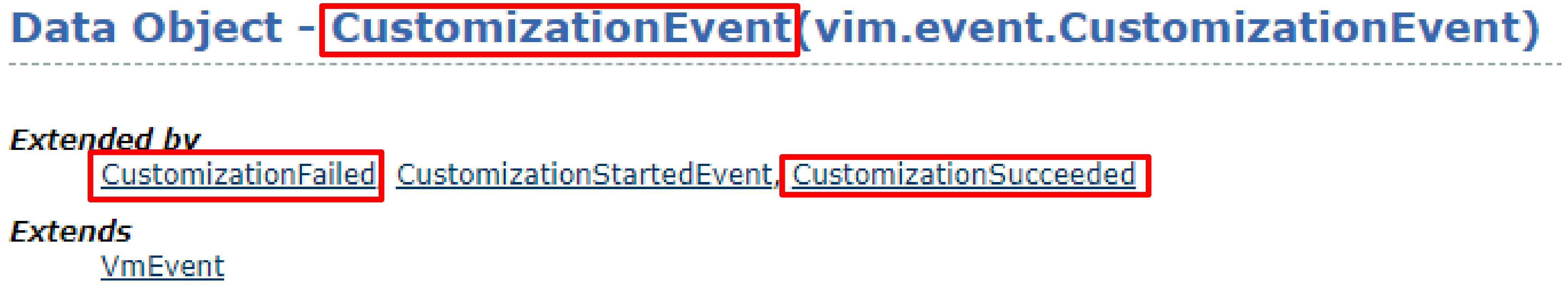
Wait For It



```
CreatedTime        Type                          FullFormattedMessage
-----------        ----                          --------------------
12-Aug-19 20:08:29 VmBeingDeployedEvent          Deploying PhotonTest on host esx3.local.l
12-Aug-19 20:08:29 VmInstanceUuidAssignedEvent   Assign a new instance UUID (50304d3e-ddb1
12-Aug-19 20:08:29 VmUuidAssignedEvent           Assigned new BIOS UUID (4230c7f5-a351-626
12-Aug-19 20:08:47 VmReconfiguredEvent           Reconfigured PhotonTest on esx3.local.lab
12-Aug-19 20:08:47 VmDeployedEvent               Template photon deployed on host esx3.loc
12-Aug-19 20:08:48 VmReconfiguredEvent           Reconfigured PhotonTest on esx3.local.lab
12-Aug-19 20:08:50 VmBeingRelocatedEvent         Relocating PhotonTest in DC from esx3.loc
12-Aug-19 20:08:54 VmRelocatedEvent              Completed the relocation of the virtual m
12-Aug-19 20:08:54 VmStartingEvent               PhotonTest on host esx1.local.lab in DC i
12-Aug-19 20:08:56 VmMessageEvent                Message on PhotonTest on esx1.local.lab i
12-Aug-19 20:08:57 VmPoweredOnEvent              PhotonTest on  esx1.local.lab in DC is po
12-Aug-19 20:09:07 CustomizationStartedEvent     Started customization of VM PhotonTest. C
12-Aug-19 20:09:20 CustomizationSucceeded        Customization of VM PhotonTest succeeded.
```

Wait For It

- We wait for: Succeeded or Failed
- We capture the CustomizationEvent object



Data Object - CustomizationEvent(vim.event.CustomizationEvent)

Extended by
CustomizationFailed, CustomizationStartedEvent, CustomizationSucceeded

Extends
VmEvent

Wait For It

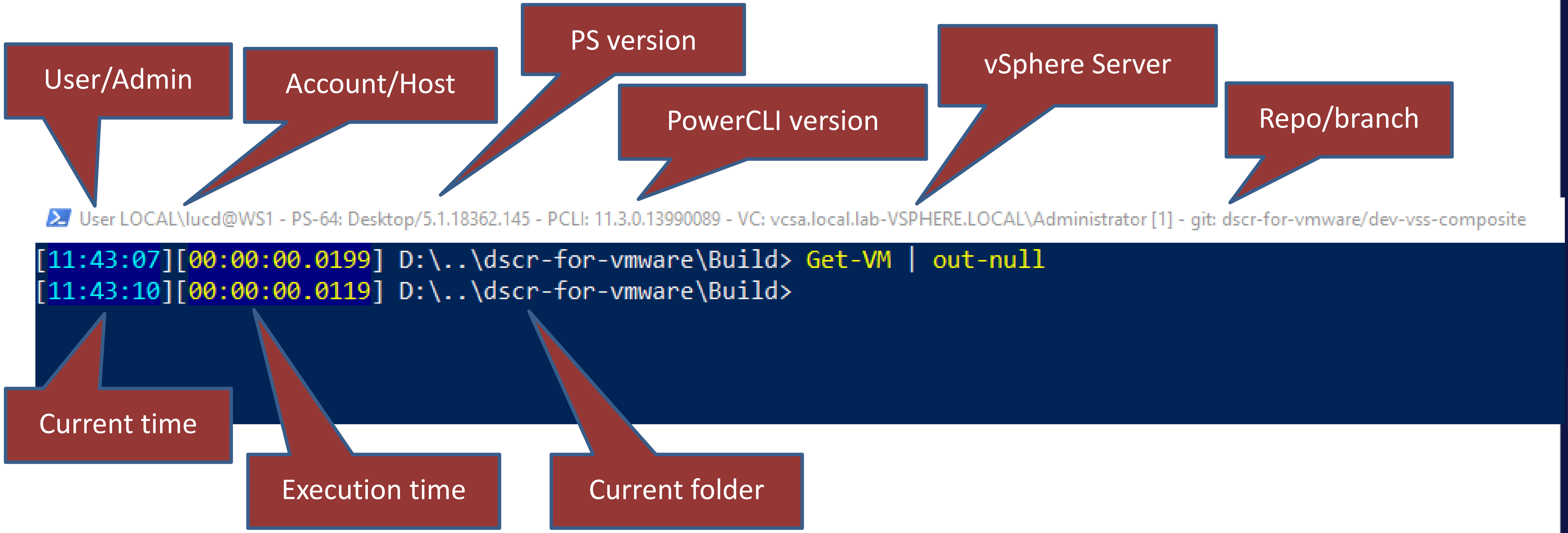
1
```
$vm = New-VM @sVM | Start-VM -Confirm:$false
```
2
```
$sEvent = @{
    Entity = $vm
    MaxSamples = [int]::MaxValue
    Start = (Get-Date).AddSeconds(-5)
  }
```
3
```
$condition = {
    $_ -is [VMware.Vim.CustomizationSucceeded] -or
    $_ -is [VMware.Vim.CustomizationFailed]
  }
```
4
```
Do {
    Start-Sleep -Seconds 5
    $custEvents = (Get-VIEvent @sEvent).Where($condition)
  } until ($custEvents)
```

At Your Fingertips

- How often did/do you execute any of these?
  - `$global:DefaultViServer`
  - `$PSVersionTable`
  - `pwd`
  - `Get-Module -Name VMware.PowerCLI -ListAvailable`
  - …
- Useful while coding and when asking for guidance
- Can be automated

At Your Fingertips

- Meet your “power(cli)” profile
- A lot of info at your fingertips



```
User LOCAL\lucd@WS1 - PS-64: Desktop/5.1.18362.145 - PCLI: 11.3.0.13990089 - VC: vcsa.local.lab-VSPHERE.LOCAL\Administrator [1] - git: dscr-for-vmware/dev-vss-composite
[11:43:07][00:00:00.0199] D:\..\dscr-for-vmware\Build> Get-VM | out-null
[11:43:10][00:00:00.0119] D:\..\dscr-for-vmware\Build>
```

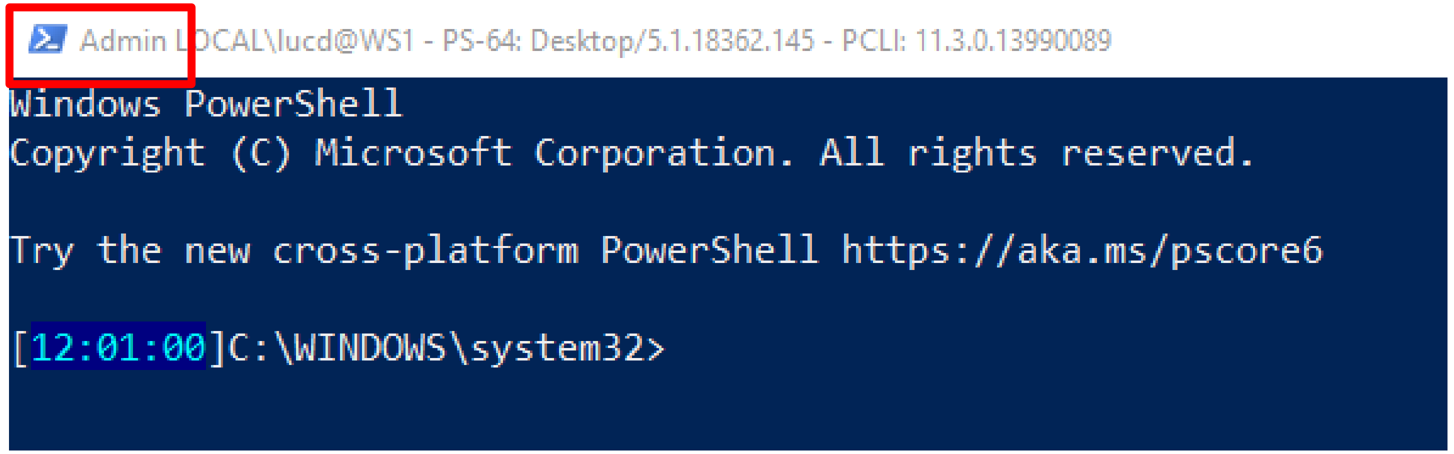
At Your Fingertips

```
function prompt
 {
   # Shorted PWD
   $path = $pwd.Path.Split('\')
   if ($path.Count -gt 3)
   {
     $path = $path[0], '..', $path[-2], $path[-1]
   }
   Write-Host -Object "$($path -join '\')" –NoNewline
 }
```

At Your Fingertips

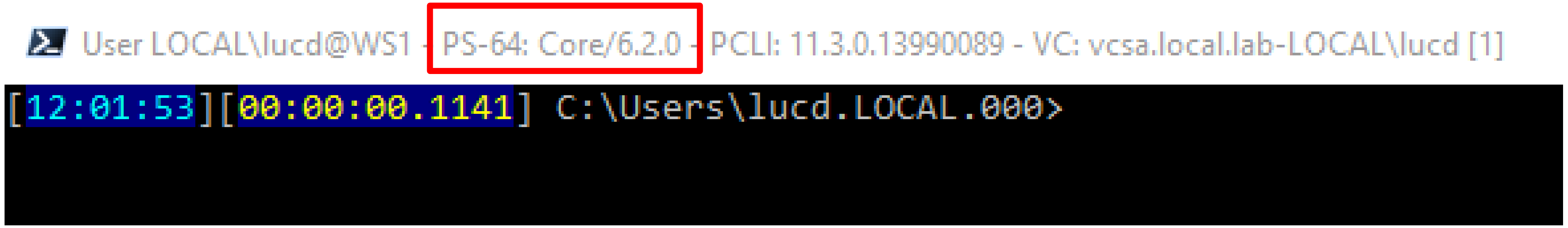
```
if ($global:defaultviserver)
  {
    $vcObj = (Get-Variable -Scope global -Name 'DefaultVIServer').Value
    if ($vcObj.ProductLine -eq 'vpx’){ $vcSrv = 'VC’ }
    else{ $vcSrv = 'ESXi’ }
    $vc = " - $($vcSrv): $($vcObj.Name)-$($vcObj.User)”
    $vc = “[$($global:DefaultVIServers.Count)]”
  }

  # Update the Window's title
  $host.ui.RawUI.WindowTitle = "$user$ps$pcli$vc$gitStr"
```

At Your Fingertips



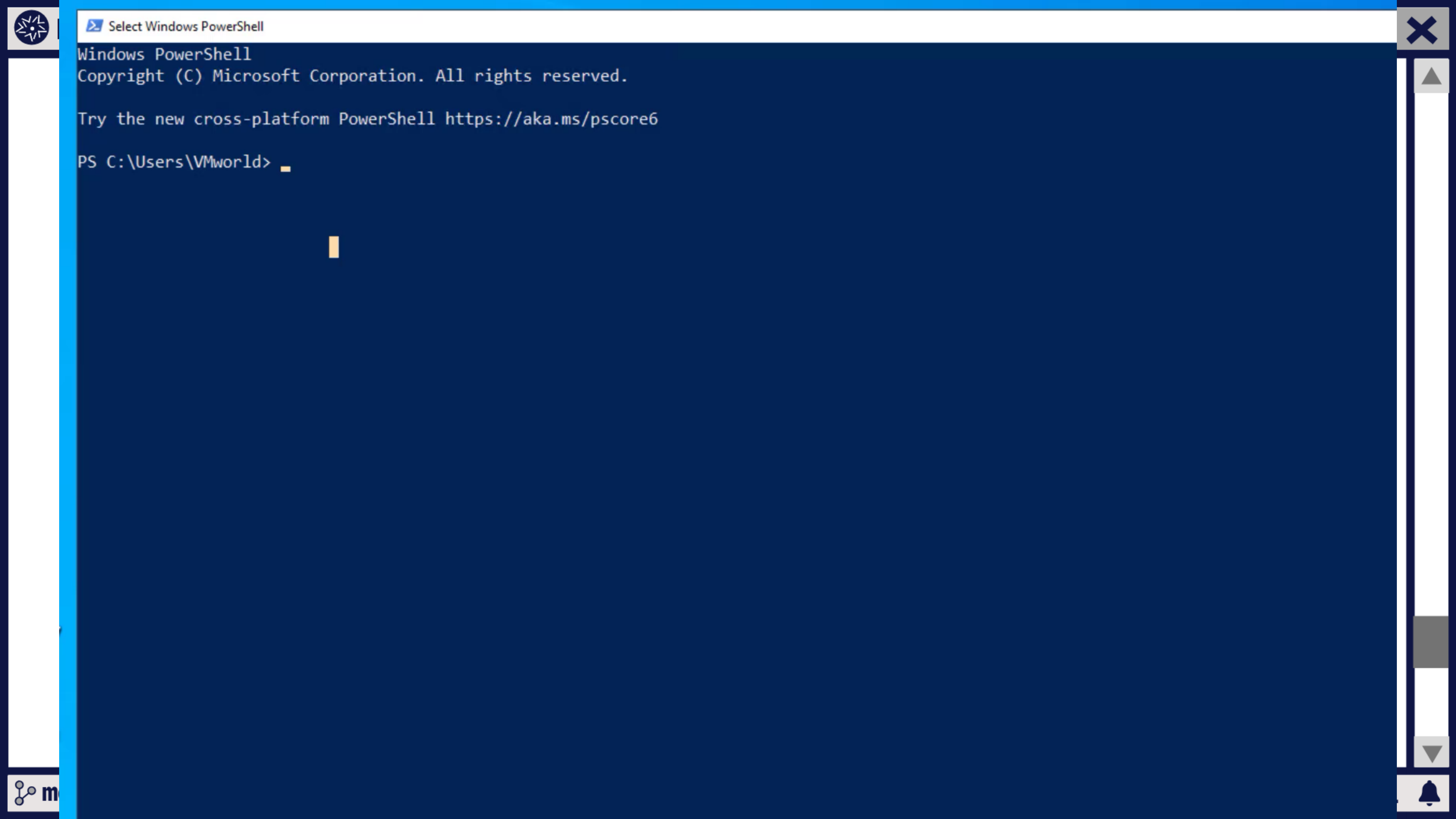
Admin LOCAL\lucd@WS1 - PS-64: Desktop/5.1.18362.145 - PCLI: 11.3.0.13990089

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

[12:01:00]C:\WINDOWS\system32>
```



User LOCAL\lucd@WS1 - PS-64: Core/6.2.0 - PCLI: 11.3.0.13990089 - VC: vcsa.local.lab-LOCAL\lucd [1]

```
[12:01:53][00:00:00.1141] C:\Users\lucd.LOCAL.000>
```



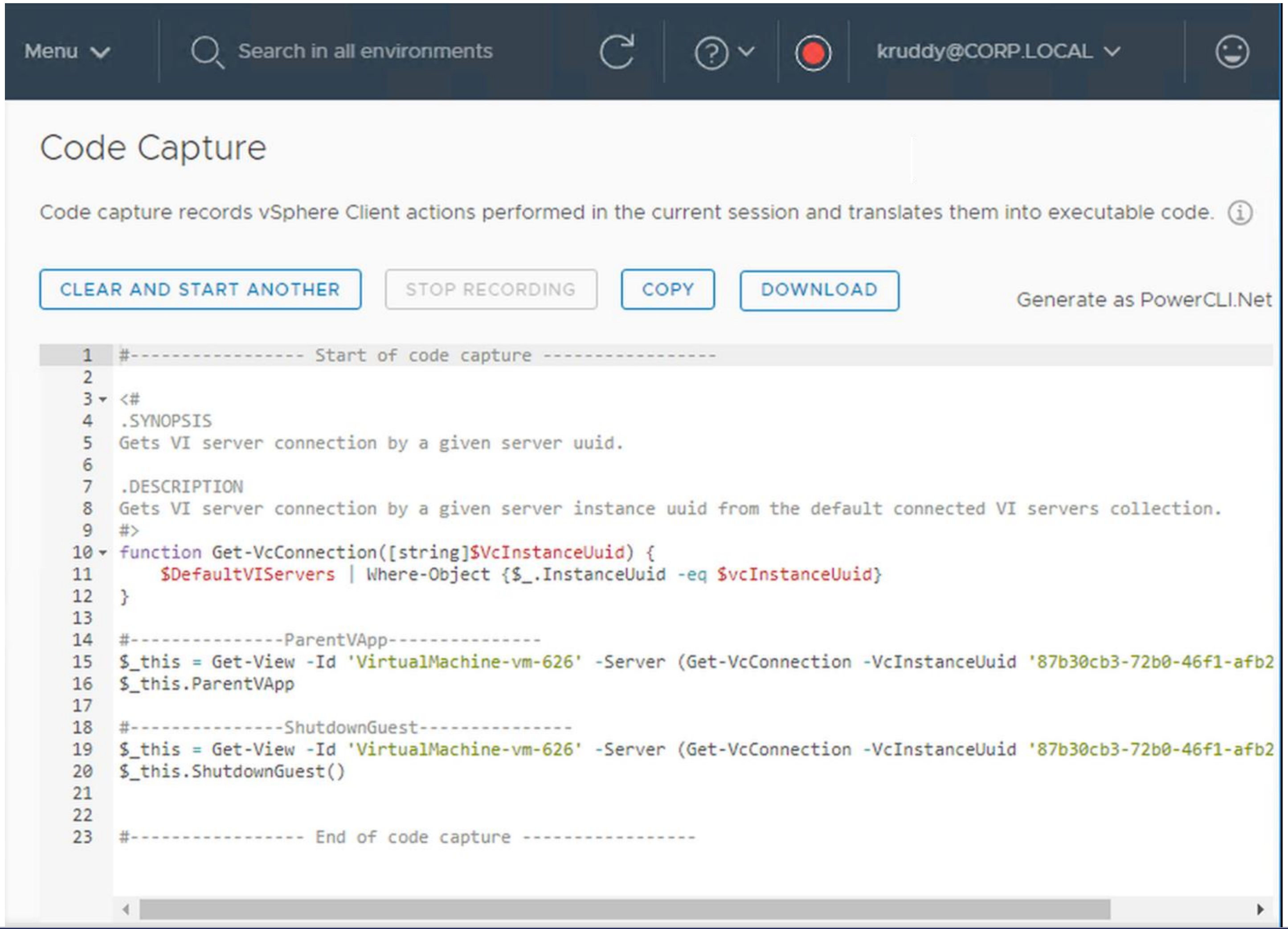
Select Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\VMworld> _
```

Code Capture

Turn UI Actions into Executable Code

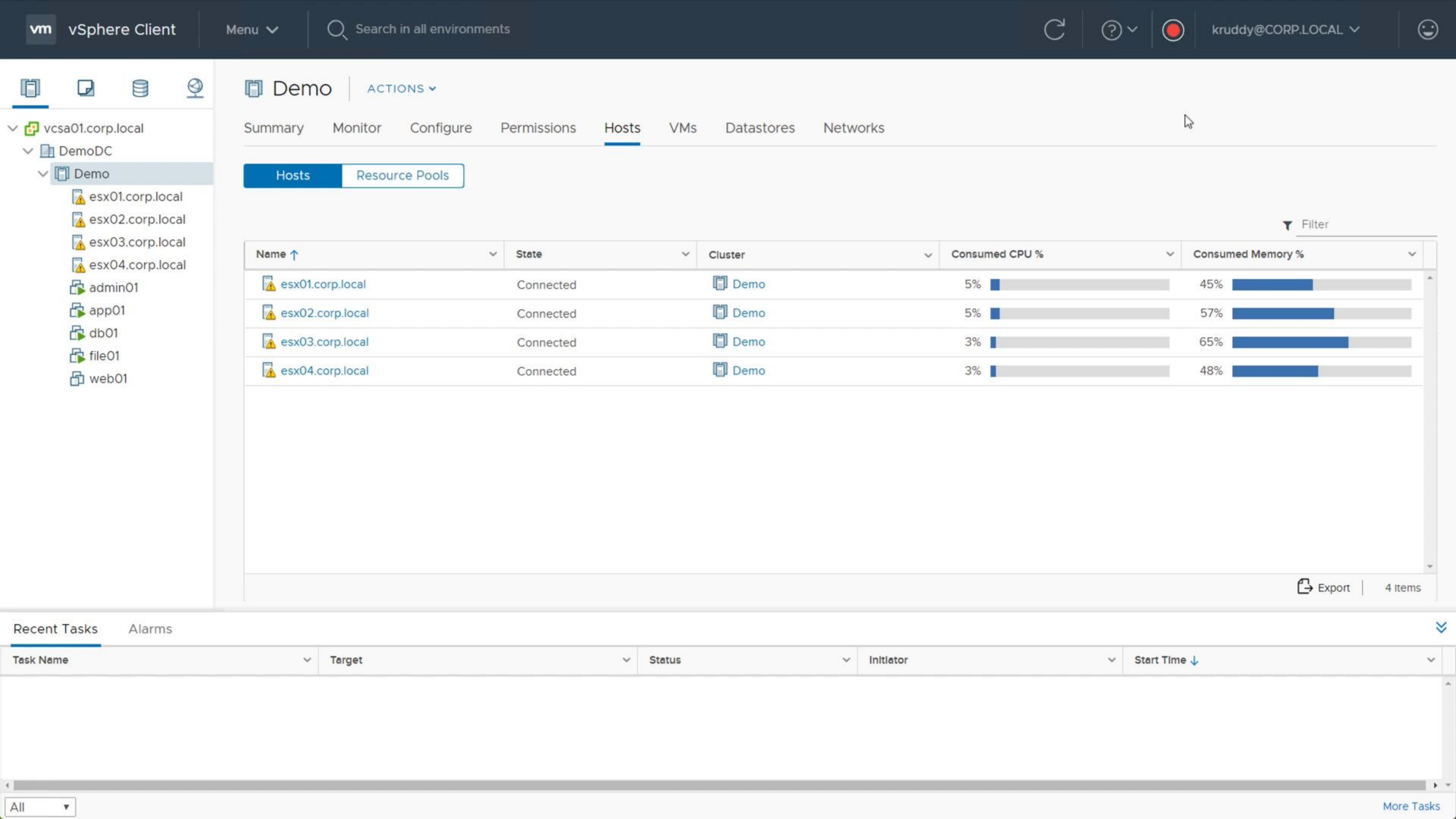
- Available
  - vSphere 6.7 U2
  - vSphere HTML 5 Fling
    - Currently Supports vSphere 6.5
- Output Type: PowerCLI
- Never use as-is!
- Refer to Code Later
  - Copy to Clipboard
  - Download as Script



Code Capture

Code capture records vSphere Client actions performed in the current session and translates them into executable code.

```
#---------------- Start of code capture ----------------

<#
.SYNOPSIS
Gets VI server connection by a given server uuid.

.DESCRIPTION
Gets VI server connection by a given server instance uuid from the default connected VI servers collection.
#>
function Get-VcConnection([string]$VcInstanceUuid) {
    $DefaultVIServers | Where-Object {$_.InstanceUuid -eq $vcInstanceUuid}
}

#--------------ParentVApp--------------
$_this = Get-View -Id 'VirtualMachine-vm-626' -Server (Get-VcConnection -VcInstanceUuid '87b30cb3-72b0-46f1-afb2
$_this.ParentVApp

#--------------ShutdownGuest--------------
$_this = Get-View -Id 'VirtualMachine-vm-626' -Server (Get-VcConnection -VcInstanceUuid '87b30cb3-72b0-46f1-afb2
$_this.ShutdownGuest()


#---------------- End of code capture ----------------
```



Name	State	Cluster	Consumed CPU %	Consumed Memory %
esx01.corp.local	Connected	Demo	5%	45%
esx02.corp.local	Connected	Demo	5%	57%
esx03.corp.local	Connected	Demo	3%	65%
esx04.corp.local	Connected	Demo	3%	48%

Additional Information

Make use of the community:

- VMware PowerCLI Community: https://vmware.com/go/powercli
- VMware Code Slack Group: https://code.vmware.com/web/code/join

Example Scripts:

- Community Repo: https://github.com/vmware/PowerCLI-Example-Scripts
- Sample Exchange: https://code.vmware.com/samples

Additional Information

The Complete Guide to PowerShell Punctuation

- https://www.simple-talk.com/sysadmin/powershell/the-complete-guide-to-powershell-punctuation/

PowerCLI Info Page: https://code.vmware.com/tool/vmware-powercli

–Includes: Cmdlet Reference, User Guide, Change Log, Release Notes

PSCONF.ASIA

The End

QUESTIONS?

OK Cancel

master* 0↓ 1↑ |Ln 1, Col 1 |Spaces: 4 |UTF-8 |CRLF | 5.1