On the way to Rapid Recovery, part 2: Creating a comprehensive PowerCLI script

In the previous blog post, I tempted your imagination with an omnibus PowerCLI script with Swiss Army knife abilities. Either successful or not, I am determined to continue. As such, the time has come to see how this script can be addressed in real life.

For the record, the goal is to create a script that would be smart enough to find and load the PowerCLI objects into a regular PowerShell console or, even better, to work with the Windows PowerShell Integrated Scripting Environment (ISE). This would make scripting PowerCLI snippets an easy task. If you aren't familiar with ISE, you can find the bare (and I mean bare) basics at https://technet.microsoft.com/en-us/library/dd315244.aspx.

Once you get access to PowerCLI via ISE, we can plan the next steps — how to insert, keep and edit encrypted connection information for your VMware environment and, finally, how to add different tasks, snippets or sub-scripts able to perform, with minimal administrative effort, the variety of VMware-related tasks needed when working with Dell Data Protection | Rapid Recovery 6.0.x.

Without further ado, let's begin!

A common sense question to ask is, "What prerequisites are needed?" The common sense answer is, obviously, PowerCLI. However, like any common sense statement, it means that a lot of groundwork was previously done.

First, you need to determine if PowerCLI is installed. For the sake of this post, we'll suppose that you are working with PowerCLI 6.3 release 1, which is the recommended PowerCLI release and is available for download at https://developercenter.vmware.com/tool/vsphere_powercli/6.3. This is the newest version of PowerCLI and, unlike older versions, it hosts VMware objects in modules as opposed to snap-ins.

Once the PowerCLI installation is finished, let's see if the PowerCLI modules are available to PowerShell.

```
PS C:\tps> Get-Module -ListAvailable
```

It goes without saying that no PowerCLI module is listed. Alas, there's no direct way to access these modules straight from PowerShell.

Assuming that we intend to run PowerCLI tasks on an unfamiliar machine — a customer's Rapid Recovery core, for instance — what is to be done to determine programmatically if PowerCLI is even installed?

The simplest way I could to think of is to check the existence of the corresponding registry key (some research was needed, but I will skip the procedure as there's a lot more important ground to cover). From the registry key, we can determine the installation path and go from there.

The important thing is to know that, for a 64-bit machine, the PowerCLI registry path is:

```
"HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\VMware, Inc.\VMware vSphere PowerCLI"
```

Insert this registry path in PowerShell.

```
$powercliregpath = "REGISTRY::HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\VMware, Inc.\VMware vSphere
PowerCLI"
$haspowercli = test-path $powercliregpath
```

If the registry path is present, the next step is to determine the installation path.

```
$InstallPath = (Get-ItemProperty $powercliregpath -ErrorAction SilentlyContinue -name
InstallPath).InstallPath
```

If PowerCLI was installed at the default location, you'll get the following installation path:
C:\Program Files (x86)\VMware\Infrastructure\vSphere PowerCLI\

After a little digging through the folders available at that path, it looks like the PowerCLI modules are located in C:\Program Files (x86)\VMware\Infrastructure\vSphere PowerCLI\Modules.

There are quite a few of them. And attempting to load the main module (VMware.VimAutomation.Core) results in an error.

```
PS C:\Program Files (x86)\VMware\Infrastructure\vSphere PowerCLI\modules> Import-Module -Name
VMware.VimAutomation.Core
Import-Module : The specified module 'VMware.VimAutomation.Core' was not loaded because no valid
module file was found in any module directory.
At line:1 char:1
+ Import-Module -Name VMware.VimAutomation.Core
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ResourceUnavailable: (VMware.VimAutomation.Core:String) [Import-
Module], FileNotFoundException
    + FullyQualifiedErrorId :
Modules_ModuleNotFound,Microsoft.PowerShell.Commands.ImportModuleCommand
```

Trying a few others at random gets the same result. Looks like we're out of luck.

Now comes the cool part. The most likely reason for the error above is that there are a few dependent sub-modules that need to be loaded, as well. You could spend the day attempting various permutations of which module to load first and you may succeed (and hope that none will change names or dependencies in the next release), but there is a better way.

PowerShell knows the location of the available modules from an environment variable called $PSModulePath, which I stumbled upon in another occasion, but it comes in handy now. To find it, just access the Env: provider.

```
PS C:\tps> cd env:


PS Env:\> ls


Name                           Value
----                           -----
ALLUSERSPROFILE                C:\ProgramData
APPDATA                        C:\Users\tpopescu\AppData\Roaming
CommonProgramFiles             C:\Program Files\Common Files
CommonProgramFiles(x86)        C:\Program Files (x86)\Common Files
CommonProgramW6432             C:\Program Files\Common Files
COMPUTERNAME                   DSG9ZCQXW1
…
SMODULEPATH                   C:\Users\tpopescu\Documents\WindowsPowerShell\Modules;C:\Program
Files\WindowsPowerShell\Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules\;c:\Program
Files\Microsoft Security Client\Mp...
PUBLIC                         C:\Users\Public
```

The next question comes naturally and holds the key to the PowerCLI kingdom: What if we add the PowerCLI module path to the $PSModulePath (and remove it when it's not needed anymore)?

Let's do it.

Get the $env:PsModulePath value and store it in a variable.

```
$p=$env:PSModulepath
```

Check if PowerCLI is already present in the install path (it isn't now, but some other time it may be for a variety of reasons), and if it is not, add it to the PSModulePath.

```
if($p.indexof($installpath) -eq -1){
$p += ";$($installpath)Modules";
$env:PSModulePath =$p}
```

Don't forget to complete the path by adding the Modules folder!

Alternatively — this will be necessary at a later time, so it's good to think of it now so you don't have to worry later — to remove PowerCLI from the install path, run:

```
$p=$p.replace(";$($installpath)Modules","");
$env:PSModulePath =$p;
```

Very simple indeed — just basic string manipulation.

Now, since the PSModulePath points to the location of the PowerCLI modules, we can try loading the main PowerCLI module again.

```
PS Env:\> Import-Module -Name VMware.VimAutomation.Core


PS Env:\> Get-Module -name *vmware* | ft name,version -AutoSize


Name                             Version
----                             -------
Initialize-VMware_VimAutomation_Cis 0.0
VMware.VimAutomation.Cis.Core       6.0.0.0
VMware.VimAutomation.Common         6.3.0.0
VMware.VimAutomation.Core           6.3.0.0
VMware.VimAutomation.Sdk            6.3.0.0
```

It worked. All is well now. And ISE works, as well.

Now that the groundwork has been put in place, you can take care of a few secondary, albeit important, issues.

It would be nice to access the desktop of virtual machines (VMs) without having to use the VMware client. PowerCLI introduced a commandlet, named Open-VMConsoleWindow, that is supposed to open VM desktop objects straight in your browser. However, if you try it, you are in for disappointment — this functionality is achieved via NPAPI, which is deprecated in all modern browsers.

I tried using the commandlet in Internet Explorer and Chrome to no avail. (To see what happens in Chrome, visit https://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=2114800.) And while I was able to use it in Firefox by using an add-on, there are little chances that will last when you take into account that Firefox is updated very often.

Even so, this feature is too cool to lose. There is a work-around, but another prerequisite — the VMware Remote Console (VMRC) — is needed. Like PowerCLI, VMRC is available for free. It allows running VM consoles without having to use the VMware client application or web GUI. Documentation is available at https://www.vmware.com/support/pubs/vmrc_pubs.html.

The logic for approaching VMRC is the same as for PowerCLI.

Returning to the script, if any of these prerequisites (PowerCLI or VMRC) are not present, you will see a menu that opens a browser at the download location. For instance, to download PowerCLI, the following code is needed:

```
$powerclidownload = "https://developercenter.vmware.com/web/dp/tool/vsphere_powercli/6.3"
 [System.Diagnostics.Process]::Start($powerclidownload) | Out-Null
```

At the end of a session, all connections should be dropped, PowerCLI modules unloaded and the PSModulePath adjusted by removing the PowerCLI modules path.

Suffice to say, all this functionality has been encapsulated into a function called `start-powercli`. It features a single switch parameter called unloadmodules. When used, it resets all settings as stated above.

To check how it works, load it in ISE and then run it. Enter a few VMware PowerCLI commandlets to prove that it works. Run it again (in the command line ISE console) with the unloadmodules parameter to undo all that was done!

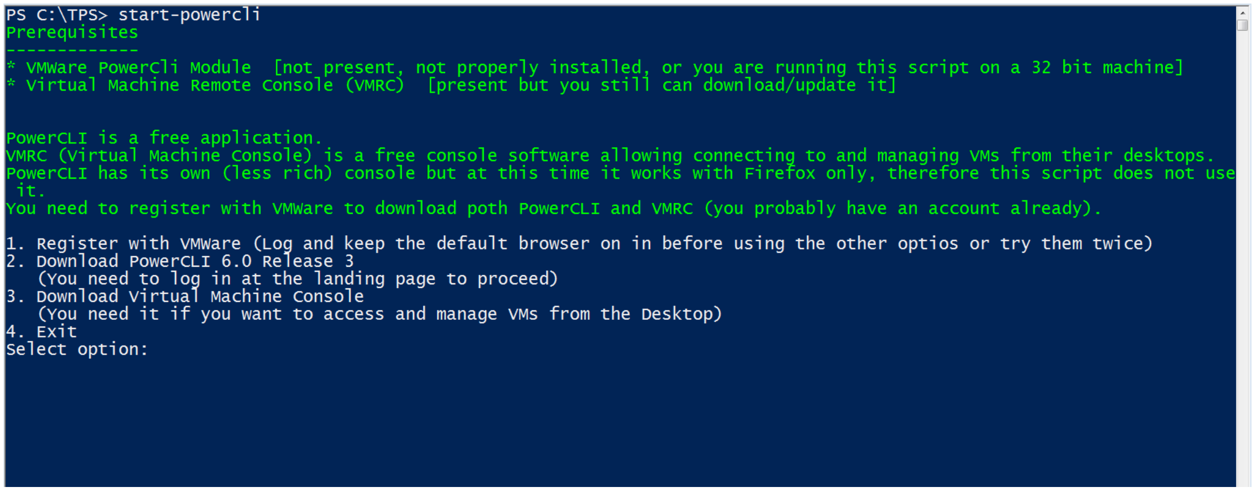
Here’s the code for the `start-powercli` function:

```
function start-powercli{
param ([switch]$unloadmodules)
$powercliregpath = "REGISTRY::HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\VMware, Inc.\VMware vSphere 
PowerCLI"
$vmrcpath = "C:\Program Files (x86)\VMware\VMware Remote Console\vmrc.exe"
$haspowercli = test-path $powercliregpath
$hasvmrcpath = test-path $vmrcpath
if($prerequisites.IsPresent){$haspowercli=$false;$hasvmrcpath=$false}

if(!($haspowercli -and $hasvmrcpath)){

$haspowercli = test-path $powercliregpath
$hasvmrcpath = test-path $vmrcpath

$powerclimessage = " `[not present, not properly installed, or you are running this script on a 32 
bit machine`]"
$vmrcmessage=" `[not present`]"

if($haspowercli){$powerclimessage =  " `[present but you still can download`/update it`]"}
if($hasvmrcpath){$vmrcmessage =  " `[present but you still can download`/update it`]"}

Write-Host @"

Prerequisites
-------------

* VMWare PowerCli Module $powerclimessage
* Virtual Machine Remote Console (VMRC) $vmrcmessage
`n
```

```
PowerCLI is a free application.
VMRC (Virtual Machine Console) is a free console software allowing connecting to and managing VMs
from their desktops.
PowerCLI has its own (less rich) console but at this time it works with Firefox only, therefore
this script does not use it.


You need to register with VMware to download both PowerCLI and VMRC (you probably have an account
already).


"@ -ForegroundColor Green


$VMwareRegistration = "https://my.vmware.com/web/vmware/registration"
$powerclidownload = "https://developercenter.vmware.com/web/dp/tool/vsphere_powercli/6.3"
$vmrcdownload = "https://my.vmware.com/en/group/vmware/details?downloadGroup=VMRC800&productId=491"
$opt=0
do{
$opt = Read-Host @"
`n1. Register with VMware (Log and keep the default browser on in before using the other options or
try them twice)


2. Download PowerCLI 6.0 Release 3
   (You need to log in at the landing page to proceed)


3. Download Virtual Machine Console
   (You need it if you want to access and manage VMs from the Desktop)


4. Exit


Select option
"@
do{



}until ("1","2","3","4" -contains $opt)
if($opt -eq 1){[System.Diagnostics.Process]::Start($VMwareRegistration) | out-null}
if($opt -eq 2){[System.Diagnostics.Process]::Start($powerclidownload) | Out-Null}
if($opt -eq 3){[System.Diagnostics.Process]::Start($vmrcdownload) | Out-Null}
}until ($opt -eq 4)
exit


}
$InstallPath = (Get-ItemProperty $powercliregpath -ErrorAction SilentlyContinue -name
InstallPath).InstallPath
$p=$env:PSModulepath
if($unloadmodules.IsPresent){Write-Host "`nClosing all Connections and Removing VMware Modules" -
ForegroundColor Yellow;
try{disconnect-viserver -Server * -Force -confirm:$false -ErrorAction stop}catch{Write-Host "No
ESXi or Vcenter connections found"};
Remove-module -name *VMware*
$p=$p.replace(";$($installpath)Modules","");$env:PSModulePath =$p; return
```

```
}
if($p.indexof($installpath) -eq -1){$p += ";$($installpath)Modules";$env:PSModulePath =$p}
Import-Module -Name VMware.VimAutomation.Core
Get-Module -name *vmware* | ft name,version -AutoSize
return
}
```

The following image shows what it looks like when the function is running.

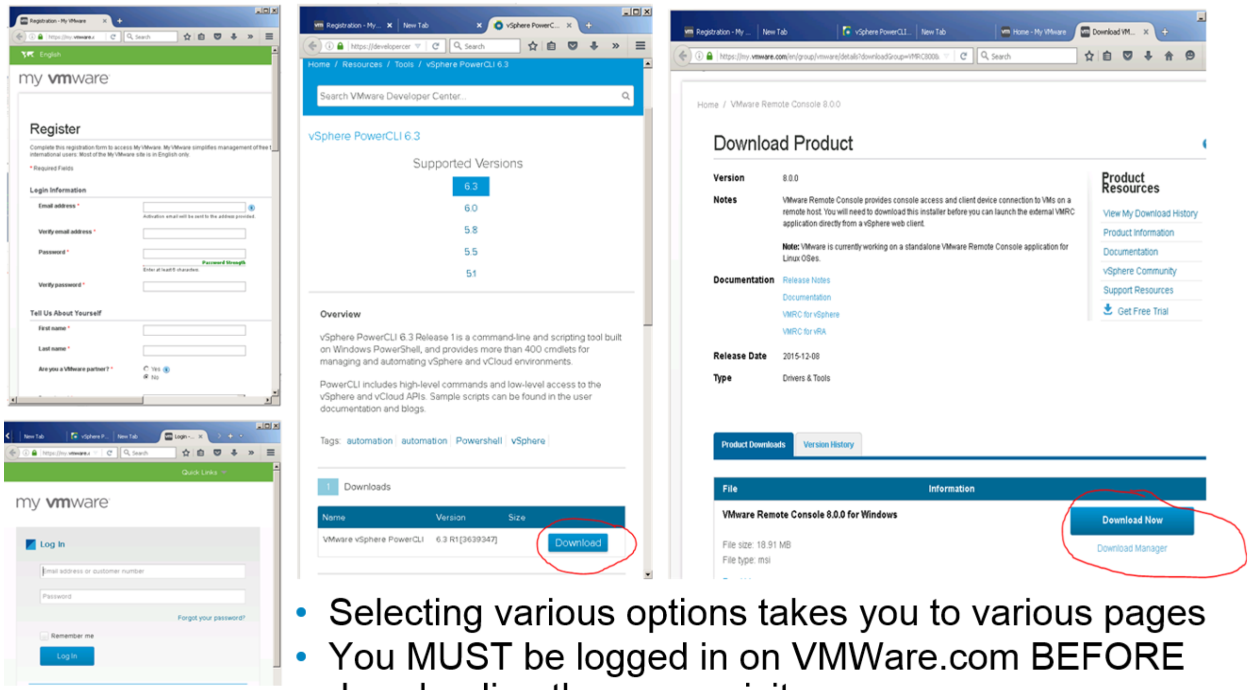


```
PS C:\TPS> start-powercli
Prerequisites
-------------
* VMware PowerCli Module  [not present, not properly installed, or you are running this script on a 32 bit machine]
* Virtual Machine Remote Console (VMRC)  [present but you still can download/update it]


PowerCLI is a free application.
VMRC (Virtual Machine Console) is a free console software allowing connecting to and managing VMs from their desktops.
PowerCLI has its own (less rich) console but at this time it works with Firefox only, therefore this script does not use
 it.
You need to register with VMware to download poth PowerCLI and VMRC (you probably have an account already).

1. Register with VMware (Log and keep the default browser on in before using the other optios or try them twice)
2. Download PowerCLI 6.0 Release 3
   (You need to log in at the landing page to proceed)
3. Download Virtual Machine Console
   (You need it if you want to access and manage VMs from the Desktop)
4. Exit
Select option:
```

Please note: Registering with VMware is necessary to download VMRC, and you need to be logged in prior to attempting it. But, there is an option to download VMRC without registering at https://my.vmware.com/en/web/vmware/details?downloadGroup=VMRC800&productId=491.

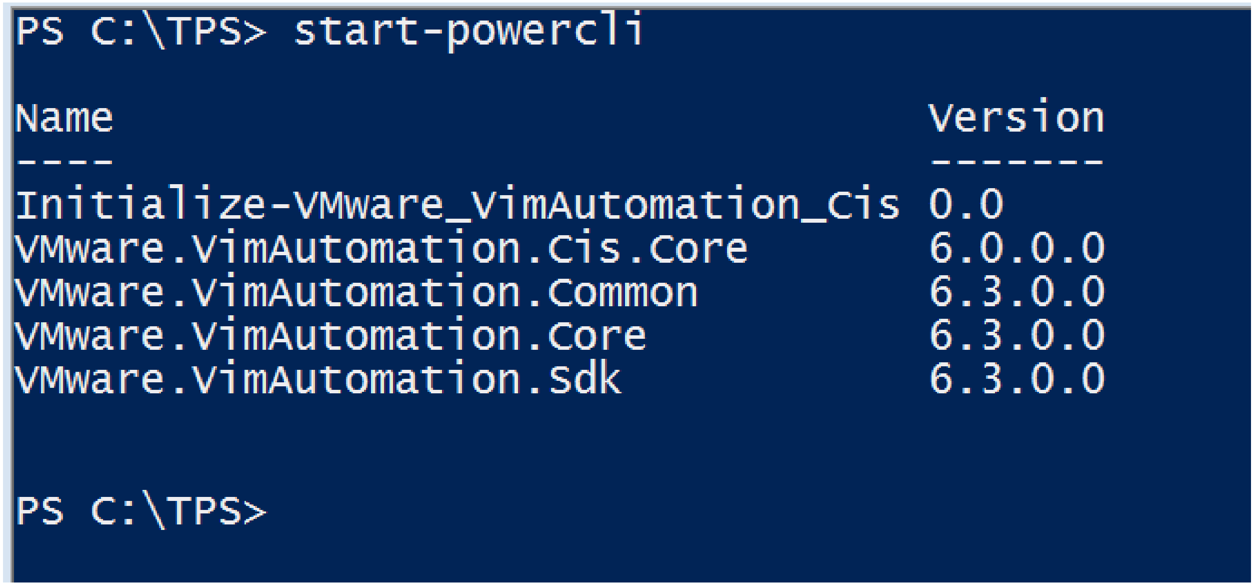
While this option exists, I would argue that, if you aren't registered with VMware but are using their products, it is courteous to register. As such, I didn't remove the "Register" option from the script.

Below are the various VMware web pages necessary to get the prerequisites.



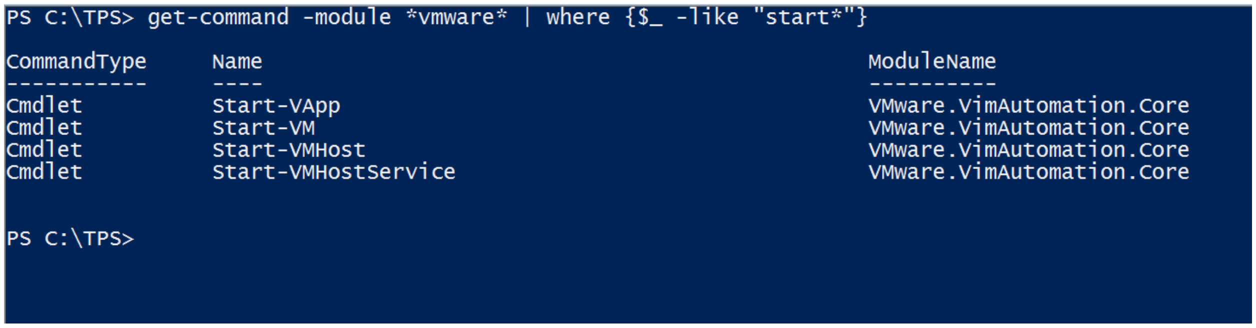
- Selecting various options takes you to various pages
- You MUST be logged in on VMWare.com BEFORE downloading the prerequisites

If all prerequisites are present when running the `start-powercli` function, you'll see the following screen.



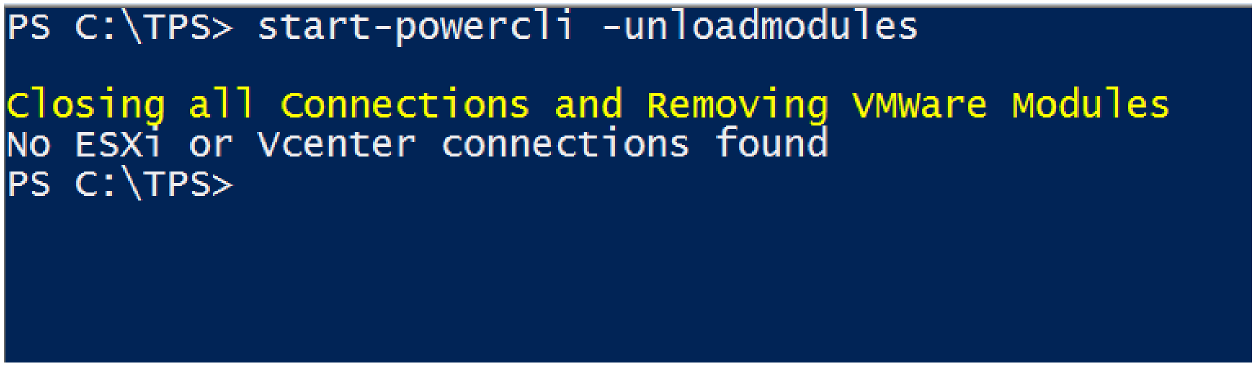
```
PS C:\TPS> start-powercli

Name                                Version
----                                -------
Initialize-VMware_VimAutomation_Cis 0.0
VMware.VimAutomation.Cis.Core       6.0.0.0
VMware.VimAutomation.Common         6.3.0.0
VMware.VimAutomation.Core           6.3.0.0
VMware.VimAutomation.Sdk            6.3.0.0


PS C:\TPS>
```

Here, you can see a list of some VMware commandlets.



```
PS C:\TPS> get-command -module *vmware* | where {$_ -like "start*"}

CommandType     Name                                               ModuleName
-----------     ----                                               ----------
Cmdlet          Start-VApp                                         VMware.VimAutomation.Core
Cmdlet          Start-VM                                           VMware.VimAutomation.Core
Cmdlet          Start-VMHost                                       VMware.VimAutomation.Core
Cmdlet          Start-VMHostService                                VMware.VimAutomation.Core


PS C:\TPS>
```

And here you see modules unloading and all settings reverting.



```
PS C:\TPS> start-powercli -unloadmodules

Closing all Connections and Removing VMWare Modules
No ESXi or Vcenter connections found
PS C:\TPS>
```

Next time, we will discuss how to script the VMware host connection information and use encryption to secure it.