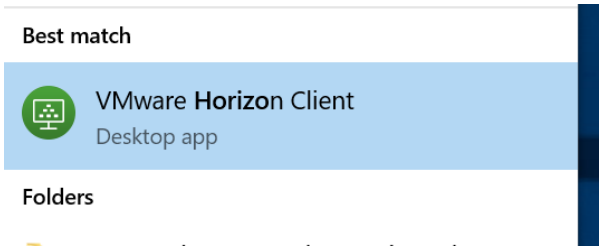
Thanks to the good people at **Arrow**, **IGEL** and **Samsung** for enabling this PowerCLI 101 workshop

Environment

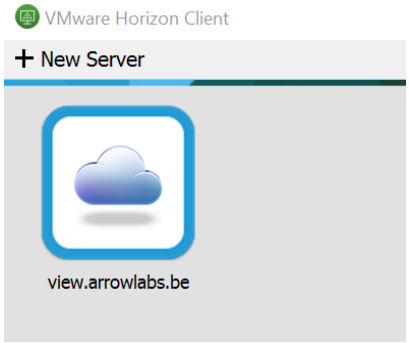
Connect to your station

Horizon Client

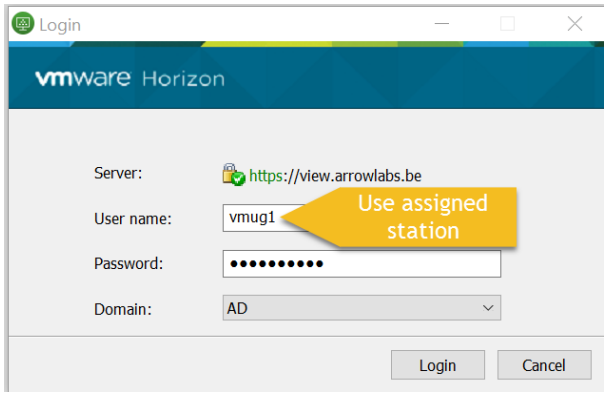
Find and start the Horizon View Client.

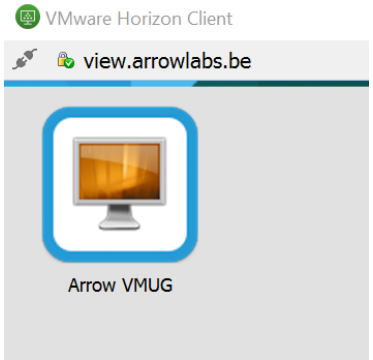


Connect to view.arrowlabs.be



Select a VDI station (allocations will be communicated during session)





Thin Client

Instructions will be communicated on site

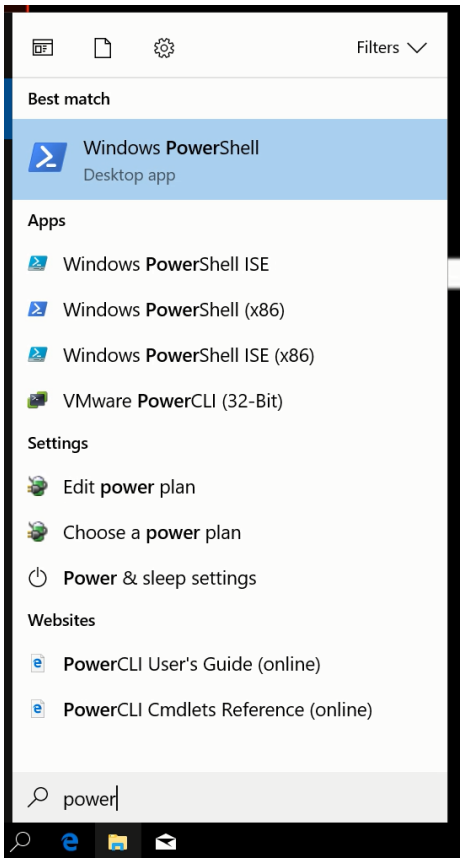
Once connected you'll get the Windows 10 desktop that we will be using during the PowerCLI 101.

Note that these are non-persistent desktops. When you logoff, your changes will be lost!

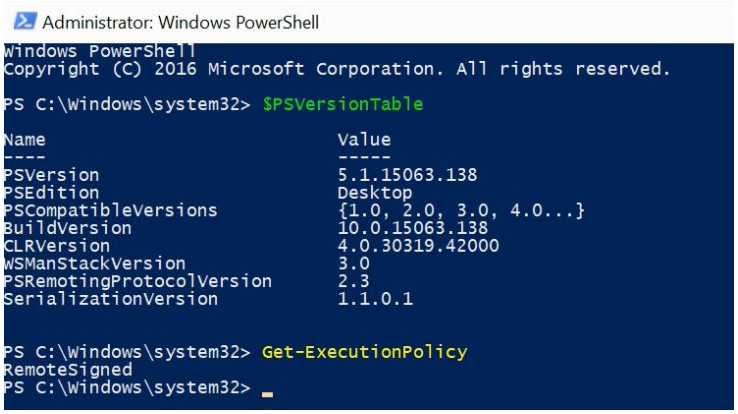
PowerShell

Let's explore the environment.

Start PowerShell



You should see PS v5.1 and RemoteSigned.



PowerCLI 6.5.1

Check the PowerCLI version.
One way of doing that is through the WMI class Win32_Product.

```
Get-WmiObject -Class Win32_Product | Where{$_.Caption -match "PowerCLI"}
```

```
PS C:\Users\vmug1> Get-WmiObject -Class Win32_Product | Where{$_.Caption -match "PowerCLI"}


IdentifyingNumber : {2E4FAF13-B720-4385-A23C-5C38D742D6C6}
Name              : VMware PowerCLI
Vendor            : VMware, Inc.
Version           : 6.5.0.234
Caption           : VMware PowerCLI
```

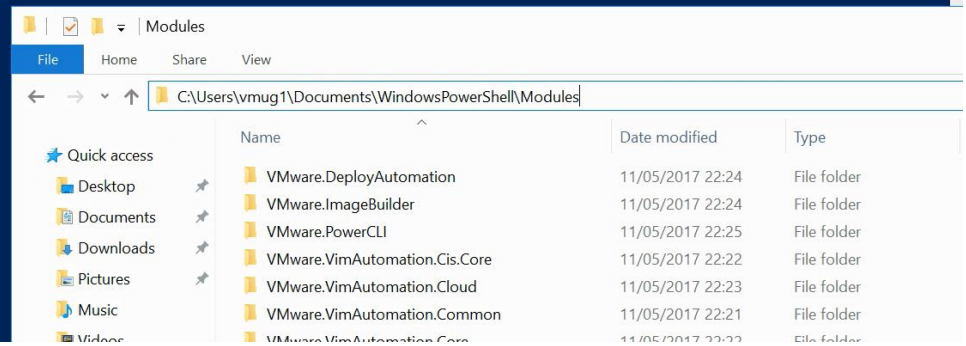
We are going to use the latest 6.5.1 version, which is available in the PowerShell Gallery.

- Uninstall the current PowerCLI installation (see SW-WMI-Uninstall.ps1)

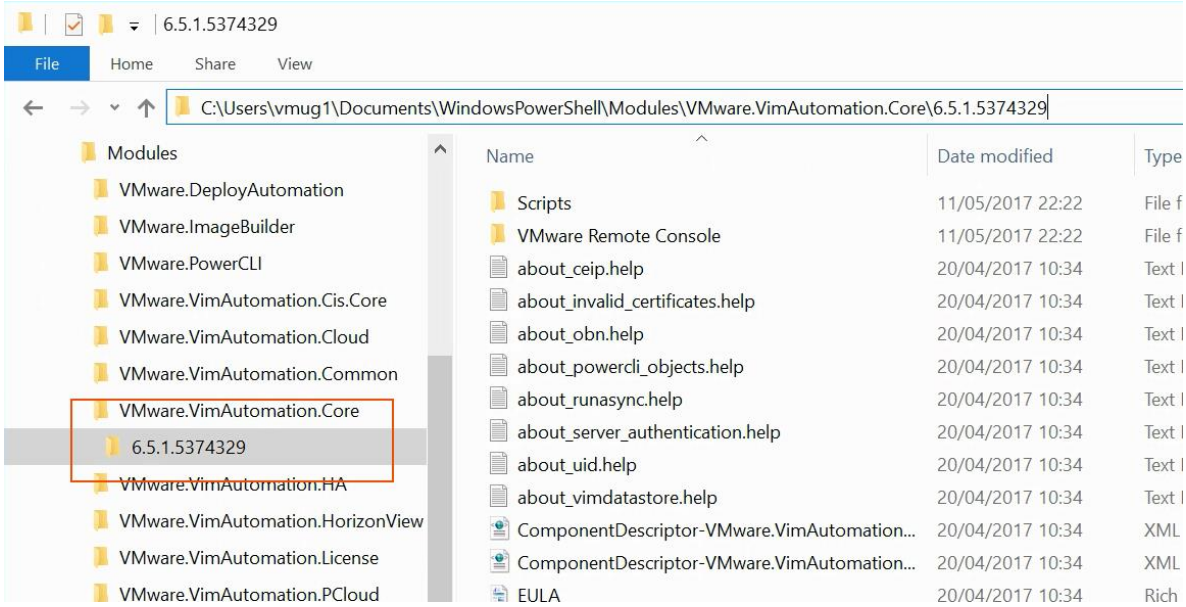
```
PS C:\Users\vmug1> $sw = Get-WmiObject -Class Win32_Product | Where{$_.Caption -match "PowerCLI"}
PS C:\Users\vmug1> $sw | gm -MemberType Method


   TypeName: System.Management.ManagementObject#root\cimv2\Win32_Product

Name      MemberType Definition
----      ---------- ----------
Configure Method     System.Management.ManagementBaseObject Configure(System.UInt16 InstallState, System.UInt16 Inst...
Reinstall Method     System.Management.ManagementBaseObject Reinstall(System.UInt16 ReinstallMode)
Uninstall Method     System.Management.ManagementBaseObject Uninstall()
Upgrade   Method     System.Management.ManagementBaseObject Upgrade(System.String PackageLocation, System.String Opt...


PS C:\Users\vmug1> $sw.Uninstall()


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 1603
PSComputerName   :


PS C:\Users\vmug1> $sw = Get-WmiObject -Class Win32_Product | Where{$_.Caption -match "PowerCLI"}
PS C:\Users\vmug1>
```

- Find and install the latest PowerCLI distribution

```
Windows PowerShell
PS C:\Users\vmug1>
PS C:\Users\vmug1> Find-Module -Name VMware.PowerCLI

NuGet provider is required to continue
PowerShellGet requires NuGet provider version '2.8.5.201' or newer to interact with NuGet-based repositories. The NuGet
 provider must be available in 'C:\Program Files\PackageManagement\ProviderAssemblies' or
'C:\Users\vmug1\AppData\Local\PackageManagement\ProviderAssemblies'. You can also install the NuGet provider by running
 'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do you want PowerShellGet to install and
import the NuGet provider now?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y

Version    Name                                Repository           Description
-------    ----                                ----------           -----------
6.5.1.5... VMware.PowerCLI                     PSGallery            This Windows PowerShell module contains VMware.P...


PS C:\Users\vmug1> Find-Module -Name VMware.PowerCLI | Install-Module -AllowClobber -Scope CurrentUser

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from
'https://www.powershellgallery.com/api/v2/'?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): y
PS C:\Users\vmug1>
```



- Module folders installed in Users (see Scope **CurrentUser**).
- If you need to install for all users
    - Start PowerShell as **Administrator**
    - Change Scope value to **AllUsers**
- **AllowClobber** avoids errors on duplicate cmdlets, i.e. Get-VM
- Since PowerShell v5, the module version is in the path. **Not** compatible with Powershell v4



Editor

ISE

With PowerShell we have been getting the ISE for free.

VSC

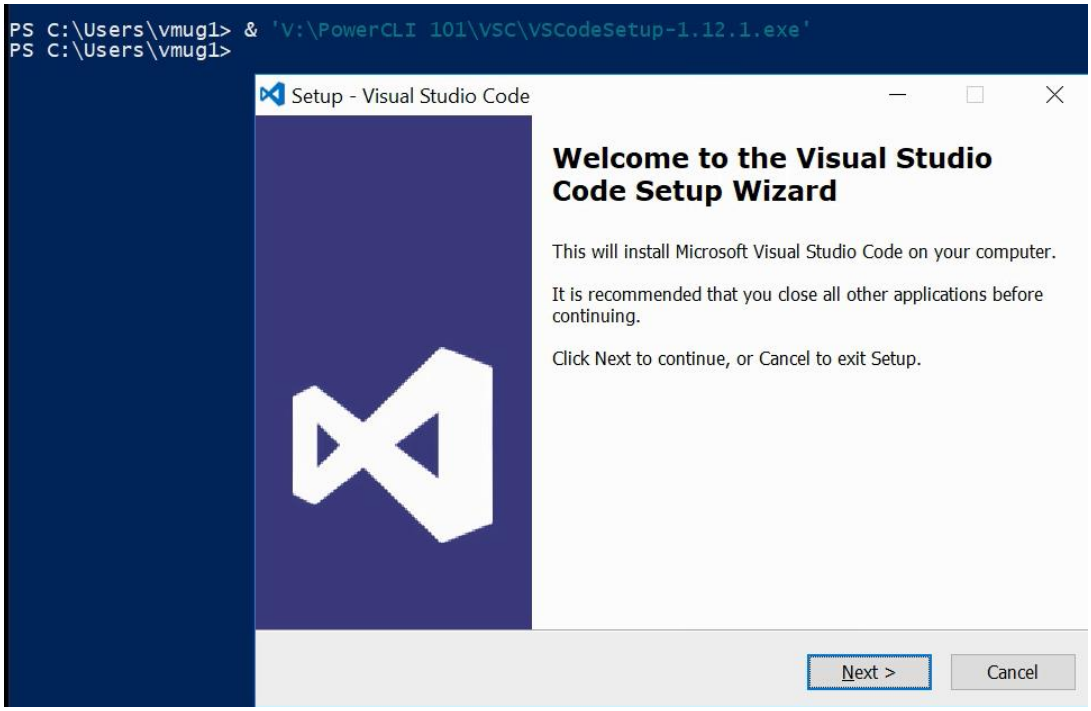
Since MSFT went multi-platform, there was a need for a multi-platform editor.

Editor

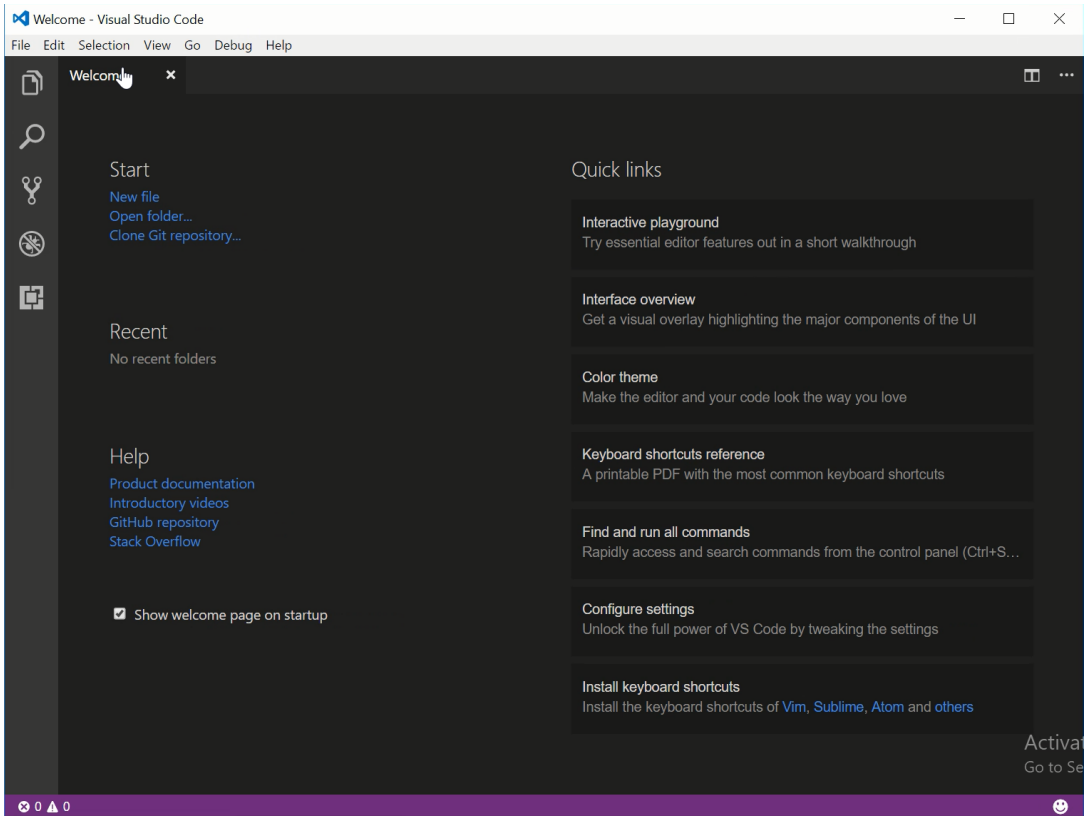
Visual Studio Code editor is an open sourced editor for Windows, Linux and MacOS.

We're installing the VSC

& 'V:\PowerCLI 101\VSC\VSCodeSetup-1.12.1.exe'



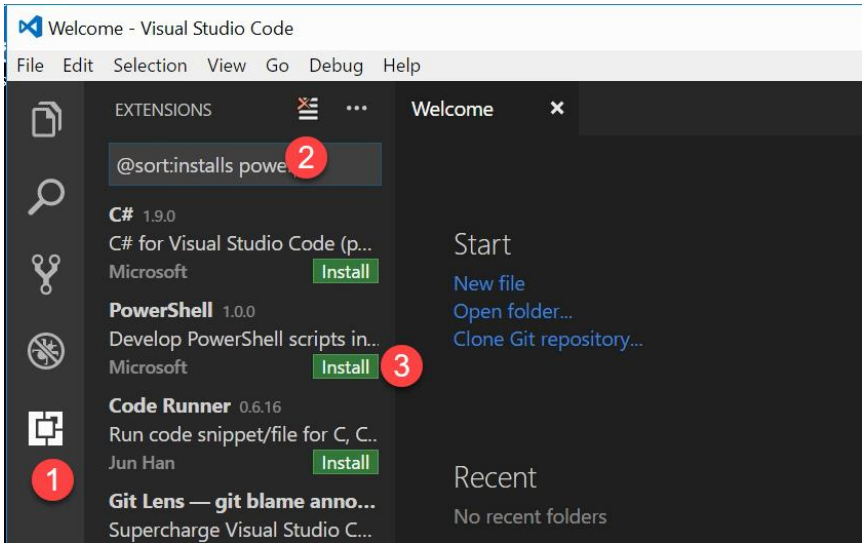
Use all the defaults (Next-Next-Next…)



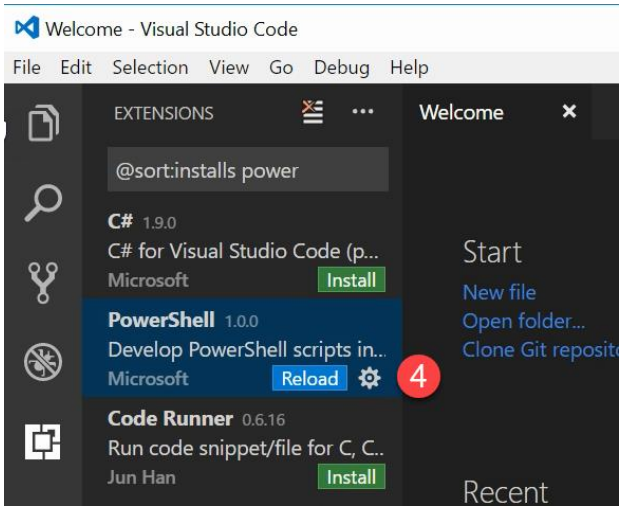
PowerShell Extension

To make the VSC usable for your PowerShell work, install the PowerShell Extension.

1. Select Extensions
2. Look for powershell
3. Install the PowerShell Extension

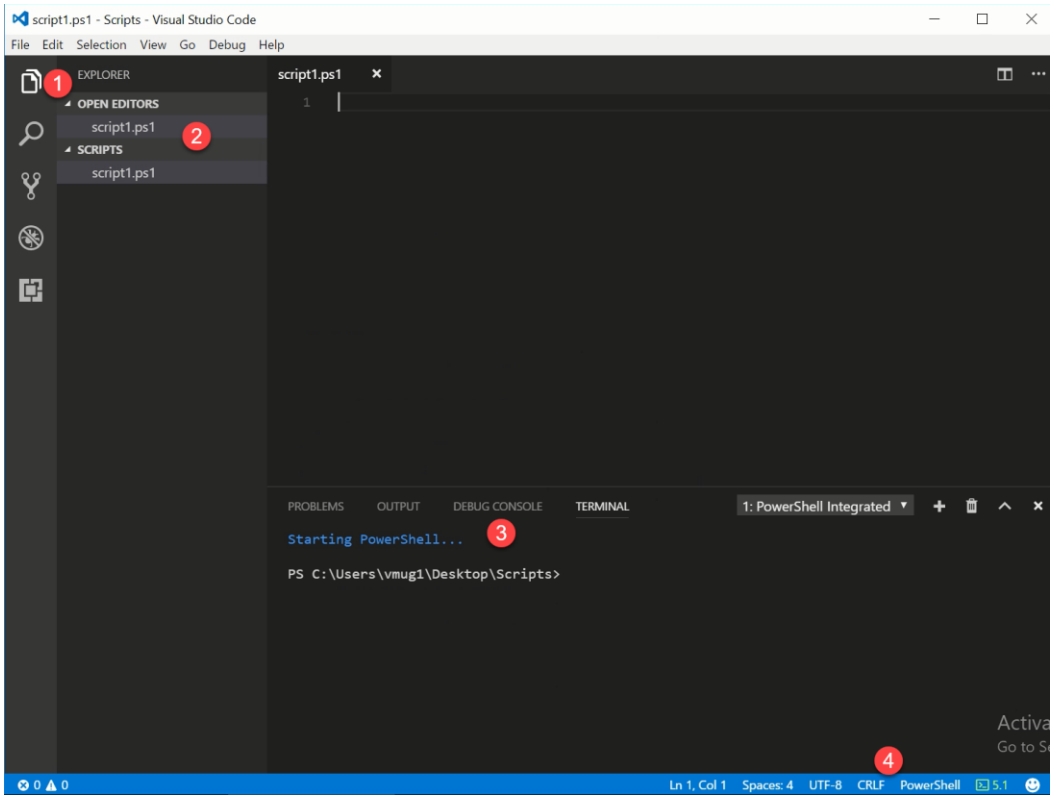


4. Reload the Extension



We can now start writing, debugging and running PowerShell scripts from the VSC.

1. From the Explorer
2. we can select a Folder and create or open files
3. The VSC offers a PowerShell prompt
4. And it recognizes automatically that we are working with a .ps1 file and switches to the PowerShell context.



We now have a basic working environment, and we will start exploring PowerShell/PowerCLI.

PowerShell

The Four

There are four cmdlets that you should remember for always. They will help you in not learning all the others by hearth.

Get-Help

Shows the help for a cmdlet or an about_ article.

```
C:\Users\vmug1\Desktop\Scripts> Get-Help -Name New-VM


NAME
    New-VM

SYNOPSIS
    This cmdlet creates a new virtual machine.


SYNTAX
    New-VM [-AdvancedOption <AdvancedOption[]>] [[-VMHost] <VMHost>] [-Version <VMVersion>] -Name <String>
    [-ResourcePool <VIContainer>] [-VApp <VApp>] [-Location <Folder>] [-Datastore <StorageResource>] [-DiskMB
    <Int64[]>] [-DiskGB <Decimal[]>] [-DiskPath <String[]>] [-DiskStorageFormat <VirtualDiskStorageFormat>]
    [-MemoryMB <Int64>] [-MemoryGB <Decimal>] [-NumCpu <Int32>] [-CoresPerSocket <Int32>] [-Floppy] [-CD]
    [-GuestId <String>] [-AlternateGuestName <String>] [-NetworkName <String[]>] [-Portgroup
    <VirtualPortGroupBase[]>] [-HARestartPriority <HARestartPriority>] [-HAIsolationResponse
```

The **ShowWIndow** switch comes in handy.

```
PS C:\Users\vmug1> Get-Help -Name Get-PowerCLIVersion -ShowWindow
PS C:\Users\vmug1>
```

Get-PowerCLIVersion Help

Find: Previous Next Settings

```
Synopsis
    This cmdlet retrieves the versions of the installed PowerCLI snapins.

Description
    This cmdlet retrieves the versions of the installed PowerCLI snapins.


Syntax

Inputs


Outputs
    One or more PowerCLIVersion objects

Notes


Examples
    --------------  Example 1 --------------
    C:\PS>Get-PowerCLIVersion
```

Get-Command

Retrieve detailed information about a command

```
PS C:\Users\vmug1> Get-Command -Name Connect-VIServer -Syntax

Connect-VIServer [-Server] <string[]> [-Port <int>] [-Protocol <string>] [-Credential <pscredential>] [-User <string>] [
-Password <string>] [-Session <string>] [-NotDefault] [-SaveCredentials] [-AllLinked] [-Force] [<CommonParameters>]

Connect-VIServer -Menu [<CommonParameters>]
```

Get-Member

Displays the properties and methods of an object.

```
C:\Users\vmug1\Desktop\Scripts> Get-VMHost | Get-Member



   TypeName: VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl

Name                 MemberType Definition
----                 ---------- ----------
ConvertToVersion     Method     T VersionedObjectInterop.ConvertToVersion[T]()
Equals               Method     bool Equals(System.Object obj)
GetHashCode          Method     int GetHashCode()
GetType              Method     type GetType()
IsConvertableTo      Method     bool VersionedObjectInterop.IsConvertableTo(type type)
LockUpdates          Method     void ExtensionData.LockUpdates()
ToString             Method     string ToString()
UnlockUpdates        Method     void ExtensionData.UnlockUpdates()
ApiVersion           Property   string ApiVersion {get;}
```

Get-Process

Returns information about the processes running on your system. Can come in handy when debugging or analyzing problems.

```
PS C:\Users\vmug1> Get-Process

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
    236      15     3728      21008       0.23   5912   1 ApplicationFrameHost
    140       9     1336       6752              5044   0 armsvc
  12519      51    77564     826732     367.56   5336   1 chrome
   1180      52    45592     102816      43.91   6004   1 chrome
    165      11     2008       8512       0.02   6704   1 chrome
    147      12     2156      10192       0.03   6900   1 chrome
    440      26    40580      61920      41.39   7660   1 chrome
    432      34   135804     182380     321.31   7720   1 chrome
    257      19    11240      25500       1.20    784   1 Code
    266      25    27408      41528       5.58   5280   1 Code
    462      56   114224     141116     362.50   5720   1 Code
    378      41    55628     102904     481.70   6176   1 Code
    366      31    39340      56664       6.98   6368   1 Code
   1046      53    31028      79680      71.09   6440   1 Code
    197      13     4208      12720       0.03   8036   1 Code
    490      15    10264      14612       0.08   1040   1 CodeHelper
    227      13     3852      20208       0.20   3336   1 conhost
```

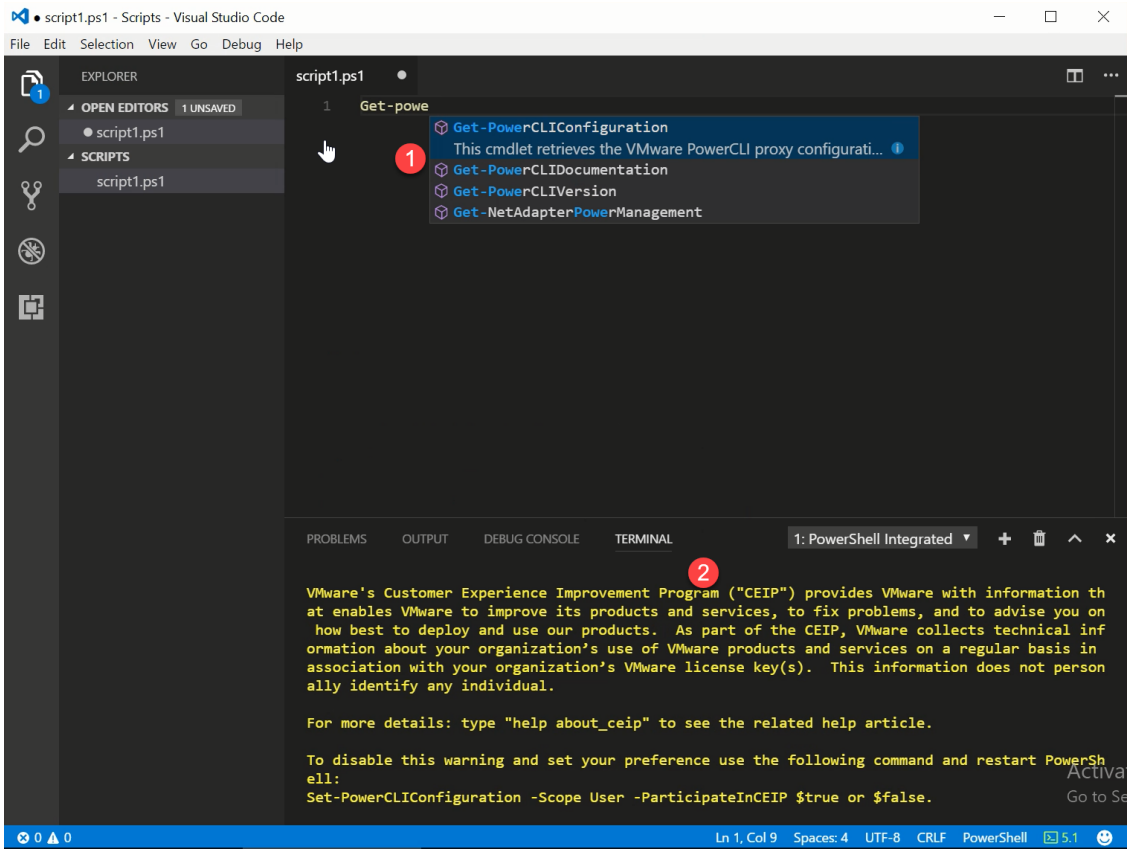
There are always multiple ways to do things in PowerShell.

There is NO single, correct script!

PowerCLI

The latest PowerCLI version supports auto-loading, no need to explicitly load the modules.

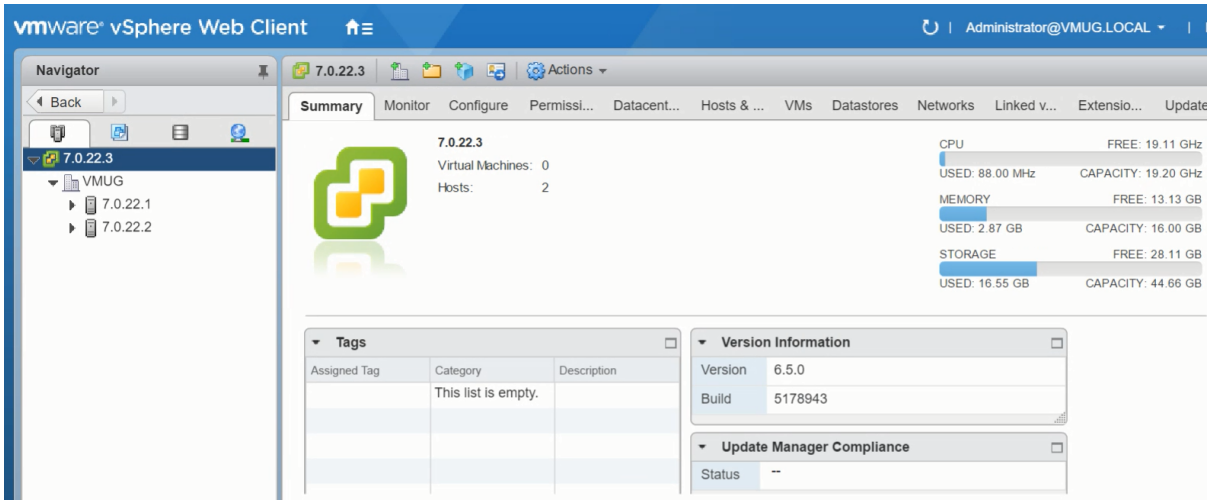
1. Intellisense shows possibilities when we start typing. There are no Import-Module commands executed.
2. The auto-load is actually loading the module, that is why we see the messages in the Terminal



vSphere Environment

We have access to a lab for the exercises we will run during this workshop.

Connection details and credentials will be communicated.



Connecting

The Connect-ViServer cmdlet creates a connection to a vSphere Server. This can be a vCenter but also an ESXi node.

You can use the Session ID to make a connection without having to provide the credentials again.

```
Connect-VIServer -Server 7.0.22.3

$global:defaultviserver | select *

Connect-VIServer -Server 7.0.22.3 -Session $global:defaultviserver.SessionId

$global:defaultviserver | select *
```

You can check how many open sessions there are in your PowerShell session.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    1: PowerShell Integrated

SessionId     : "b414f990c32f3c3228af12d05ec314ebbd2400f2"
User          : VMUG.LOCAL\Administrator
Uid           : /VIServer=vmug.local\administrator@7.0.22.3:443/
Version       : 6.5.0
Build         : 5178943
ProductLine   : vpx
InstanceUuid  : 8959deb7-cea0-4d09-9934-567fcd7f2cdc
RefCount      : 2
ExtensionData : VMware.Vim.ServiceInstance
Client        : VMware.VimAutomation.ViCore.Impl.V1.VimClient
```

EXERCISE

Use the Start-Job cmdlet to run 3 tasks in parallel. The task shall use at least one PowerCLI cmdlet, for example Get-VMHost.

Hint: You can pass arguments into a ScriptBlock

Looking around (Get)

The cmdlets that start with the Get verb can do no lasting harm to your environment. These cmdlets are ideal for retrieving information from the environment, think reporting.

EXERCISE

How many Get cmdlets are there in all the PowerCLI modules, and in each individual PowerCLI module ?

Hint: Get-Command accepts masking

OBN

PowerCLI has a handy feature which is called Object By Name.

It allows to pass a string, the Name, instead of the actual object.

```
C:\Users\vmug1\Desktop\Scripts> Get-VMHostNetworkAdapter -VMHost 7.0.22.1

Name       Mac               DhcpEnabled IP              SubnetMask      DeviceName
----       ---               ----------- --              ----------      ----------
vmnic0     00:50:56:8d:d8:b9 False                                           vmnic0
vmk0       00:50:56:8d:d8:b9 False       7.0.22.1        255.255.0.0          vmk0
```

EXERCISE

Find the description of OBN.

Searching/Filtering

There are several methods available to search for one or more specific objects. The obvious cmdlet to filter results, is the Where-Object cmdlet, which can be inserted in a pipeline construct.

```
C:\Users\vmug1\Desktop\Scripts> Get-Datastore | Where-Object {$_.Type -eq 'NFS41'}

Name                                FreeSpaceGB      CapacityGB
----                                -----------      ----------
NFS                                      24,273          39,655

C:\Users\vmug1\Desktop\Scripts> Get-Datastore | Where-Object {$_.Type -like 'NFS*'}

Name                                FreeSpaceGB      CapacityGB
----                                -----------      ----------
NFS                                      24,273          39,655

C:\Users\vmug1\Desktop\Scripts> Get-Datastore | Where-Object {$_.Type -match '^NFS'}

Name                                FreeSpaceGB      CapacityGB
----                                -----------      ----------
NFS                                      24,273          39,655
```

Changing stuff (Set)

Another important class of cmdlet are the cmdlets that start with the Set verb. They allow you to change existing object/settings.

EXERCISE

Change the default size of the syslog files on each ESXi node whose name ends in an odd number, to 2048.

Hint: try the newer Set-AdvancedSetting cmdlet

Creating stuff (New)

The third important group of PowerCLI cmdlets are the cmdlets that start with the New verb. These are used to create new objects in your vSphere environment.

EXERCISE

Create a series of VMs with a display name that has this layout “<username>-VM-n” (with N:1-3). The VM shall be for an Ubuntu 64-bit Windows OS. The hard disk of the VM shall be Thin provisioned and the VM shall be stored on the NFS datastore.

Hint: check the VirtualMachineGuestOsIdentifier enumerator

Diving Deep (use the API)

The PowerCLI cmdlets use the (in)famous 80%-20% rule. For the 20% that the cmdlets do not cover, you have to revert to the SDK methods and properties.

EXERCISE

Find for each ESXi node the partitions that can be used as diagnostic partitions.

Hint: use the API Reference