Research Triangle PowerShell User Group March 5, 2025

https://github.com/jdhitsolutions/PSCustomFormatting

Say More, Do More with Custom PowerShell Formatting¶

Jeff Hicks (https://jdhitsolutions.github.io)¶

When creating PowerShell tools that write custom objects to the pipeline, separate the data from how it is presented or formatted. **NEVER** write a PowerShell function like this:

In [25]:

```
Function Get-Foo {
    [cmdletbinding()]
    Param()
    $out = [PSCustomObject]@{
        Name         = $env:USERNAME
        ComputerName = $env:COMPUTERNAME
        Status       = 'Online'
        ID           = 32345
        PSVersion    = $PSVersionTable.PSVersion
    }
    $out | Format-Table
}

Get-Foo
```

```
Name ComputerName Status    ID PSVersion
---- ------------ ------    -- ---------
Jeff PROSPERO     Online 32345 7.4.6
```

All you can do is look at or save it to a text file. *Formatting the output is separate from the data.* **Let the user decide** how they want to see the data. However, you can define a default format for your custom pipeline output.

Why Does This Matter¶

- Create rich object output.
- Don't limit yourself in an attempt to make the output look pretty.
- Anticipate anything the user may want to know and include it.

You can make it pretty with custom formatting. Process objects are a good example.

In [26]:

```
$p = Get-Process -id $pid
$p
```

```
 NPM(K)    PM(M)      WS(M)     CPU(s)      Id  SI ProcessName
 ------    -----      -----     ------      --  -- -----------
    176    99.54     175.09      11.19   29852   1 dotnet
```

We get a customized and formatted output, even though the object is rich in properties.

In [27]:

```
$p | Select-Object -Property *
```

```
Name                       : dotnet
Id                         : 29852
PriorityClass              : Normal
FileVersion                : 9,0,24,52809 @Commit: 9d5a6a9aa463d6d10b0b0ba6d5982cc82f363dc3
HandleCount                : 1162
WorkingSet                 : 183599104
PagedMemorySize            : 104374272
PrivateMemorySize          : 104374272
VirtualMemorySize          : -1323032576
TotalProcessorTime         : 00:00:11.1875000
SI                         : 1
Handles                    : 1162
VM                         : 2481168064512
WS                         : 183599104
PM                         : 104374272
NPM                        : 180248
Path                       : C:\Program Files\Dotnet\dotnet.exe
CommandLine                : "C:\Program Files\Dotnet\dotnet.exe" C:\Users\Jeff\.nuget\package
                             rosoft.dotnet-interactive\1.0.611002\tools/net9.0/any/Microsoft.D
                             .Interactive.App.dll [vscode] stdio --working-dir
                             c:\presentations\RTPSUG-CustomFormatting
Parent                     : System.Diagnostics.Process (dotnet)
Company                    : Microsoft Corporation
CPU                        : 11.296875
ProductVersion             : 9.0.0 @Commit: 9d5a6a9aa463d6d10b0b0ba6d5982cc82f363dc3
Description                : .NET Host
Product                    : .NET
__NounName                 : Process
SafeHandle                 : Microsoft.Win32.SafeHandles.SafeProcessHandle
Handle                     : 4788
BasePriority               : 8
ExitCode                   :
HasExited                  : False
StartTime                  : 3/6/2025 10:12:04 AM
ExitTime                   :
MachineName                : .
MaxWorkingSet              : 1413120
```

```
MinWorkingSet              : 204800
Modules                    : {System.Diagnostics.ProcessModule (dotnet.exe),
                             System.Diagnostics.ProcessModule (ntdll.dll),
                             System.Diagnostics.ProcessModule (KERNEL32.DLL),
                             System.Diagnostics.ProcessModule (KERNELBASE.dll)...}
NonpagedSystemMemorySize64 : 180248
NonpagedSystemMemorySize   : 180248
PagedMemorySize64          : 104374272
PagedSystemMemorySize64    : 659408
PagedSystemMemorySize      : 659408
PeakPagedMemorySize64      : 104898560
PeakPagedMemorySize        : 104898560
PeakWorkingSet64           : 239218688
PeakWorkingSet             : 239218688
PeakVirtualMemorySize64    : 2481187639296
PeakVirtualMemorySize      : -1303457792
PriorityBoostEnabled       : True
PrivateMemorySize64        : 104374272
ProcessorAffinity          : 1048575
SessionId                  : 1
StartInfo                  :
Threads                    : {25756, 22844, 35268, 9780...}
VirtualMemorySize64        : 2481168064512
EnableRaisingEvents        : False
StandardInput              :
StandardOutput             :
StandardError              :
WorkingSet64               : 183599104
SynchronizingObject        :
MainModule                 : System.Diagnostics.ProcessModule (dotnet.exe)
PrivilegedProcessorTime    : 00:00:02.9375000
UserProcessorTime          : 00:00:08.4843750
ProcessName                : dotnet
MainWindowHandle           : 0
MainWindowTitle            :
Responding                 : True
Site                       :
Container                  :
```

Someone at Microsoft decided what an IT Pro would most likely want to see from processes and formatted the output accordingly.

Creating Custom Formatting¶

Custom formatting is defined in .ps1xml files. In Windows PowerShell, these are found in `$PSHome`.

In [28]:

```
powershell -noprofile -nologo -command '&{Get-ChildItem $PSHome\*.format.ps1xml}'
```

```
    Directory: C:\Windows\System32\WindowsPowerShell\v1.0


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          4/1/2024   3:22 AM          12825 Certificate.format.ps1xml
-a----          4/1/2024   3:22 AM           4994 Diagnostics.Format.ps1xml
-a----          4/1/2024   3:22 AM         138013 DotNetTypes.format.ps1xml
-a----          4/1/2024   3:22 AM          10112 Event.Format.ps1xml
-a----          4/1/2024   3:22 AM          25306 FileSystem.format.ps1xml
-a----          4/1/2024   3:22 AM          91655 Help.format.ps1xml
-a----          4/1/2024   3:22 AM         138625 HelpV3.format.ps1xml
-a----          4/1/2024   3:22 AM         206468 PowerShellCore.format.ps1xml
-a----          4/1/2024   3:22 AM           4097 PowerShellTrace.format.ps1xml
-a----          4/1/2024   3:22 AM           8458 Registry.format.ps1xml
-a----          4/1/2024   3:22 AM          16598 WSMan.Format.ps1xml
```

In PowerShell 7, these files have been moved into compiled code for performance. However, you can define custom formatting in a .ps1xml file and load it into your session.

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
    <ViewDefinitions>
        <View>
            <Name>OBJECT.TYPE or name of the view</Name>
            <ViewSelectedBy>
                <TypeName>OBJECT.TYPE</TypeName>
            </ViewSelectedBy>
            <TableControl>
                <!-- ################ TABLE DEFINITIONS ################ -->
                <TableHeaders>
                    <TableColumnHeader>
                        <Label>Name</Label>
                        <Width>7</Width>
                        <Alignment>right</Alignment>
                    </TableColumnHeader>
                </TableHeaders>
                <TableRowEntries>
                    <TableRowEntry>
                        <TableColumnItems>
                            <TableColumnItem>
                                <PropertyName>Name</PropertyName>
                            </TableColumnItem>
                        </TableColumnItems>
                    </TableRowEntry>
                </TableRowEntries>
            </TableControl>
        </View>
        <View>
            <Name>OBJECT.TYPE or name of the view</Name>
            <ViewSelectedBy>
                <TypeName>OBJECT.TYPE</TypeName>
            </ViewSelectedBy>
            <ListControl>
                <!-- ################ LIST DEFINITIONS ################ -->
                <ListEntries>
                    <ListEntry>
                        <EntrySelectedBy>
                            <TypeName>OBJECT.TYPE</TypeName>
                        </EntrySelectedBy>
                        <ListItems>
                            <ListItem>
                                <PropertyName>Name</PropertyName>
                            </ListItem>
                        </ListItems>
                    </ListEntry>
                </ListEntries>
            </ListControl>
        </View>
    </ViewDefinitions>
</Configuration>
```

You can mix and match format types in the same file.

Requirements¶

Your custom object must have a defined and **unique type** name. It cannot be a generic PSCustomObject.

```
<ViewSelectedBy>
    <TypeName>OBJECT.TYPE</TypeName>
</ViewSelectedBy>
```

I typically do this when creating custom objects:

```
[PSCustomObject]@{
    PSTypeName   = "PSFoo"  #<--- unique type name
    Name         = $env:USERNAME
    ComputerName = $env:COMPUTERNAME
    Status       = 'Online'
    ID           = 32345
    PSVersion    = $PSVersionTable.PSVersion
}
```

PowerShell classes are defined with a type name.

```
class PSFoo {
    [string]$Name
    [string]$ComputerName
    [string]$Status
    [int]$ID
    [version]$PSVersion
}
```

Or, you can insert a type name into an existing object.

```
$out.PSObject.TypeNames.Insert(0,"PSFoo")
```

Let's look at a more practical example.

In [29]:

```
Function Get-ServerStatus {
    [cmdletbinding(DefaultParameterSetName = 'name')]
    [OutputType('ServerStatus')]
    [alias("gst")]
    Param(
        [Parameter(
            Position = 0,
            ValueFromPipeline,
            ValueFromPipelineByPropertyName,
            HelpMessage = 'Enter the name of a computer',
            ParameterSetName = 'name')
        ]
        [ValidateNotNullOrEmpty()]
        [string]$Computername = $env:computername,
        [Parameter(ParameterSetName = 'name')]
        [PSCredential]$Credential,
        [Parameter(ParameterSetName = 'Session', ValueFromPipeline)]
        [CimSession]$CimSession,
        [Parameter(HelpMessage = "Format values as [INT]")]
        [switch]$AsInt
    )

    Begin {
        Write-Verbose "[$((Get-Date).TimeOfDay)] Starting $($MyInvocation.MyCommand)"
    } #begin

    Process {
        Write-Verbose "[$((Get-Date).TimeOfDay)] Using parameter set $($PSCmdlet.ParameterSetNa

        $sessParams = @{
            ErrorAction  = 'stop'
            computername = $null
        }
        $cimParams = @{
            ErrorAction = 'stop'
            classname   = $null
```

```
}

if ($PSCmdlet.ParameterSetName -eq 'name') {
    #create a temporary CimSession
    $sessParams.Computername = $Computername
    if ($Credential) {
        $sessParams.Credential = $credential
    }
    #if localhost use DCOM - it will be faster to create the session
    if ($Computername -eq $env:computername) {
        Write-Verbose "[$((Get-Date).TimeOfDay)] Creating a local session using DCOM"
        $sessParams.Add("SessionOption", (New-CimSessionOption -Protocol DCOM))
    }
    Try {
        Write-Verbose "[$((Get-Date).TimeOfDay)] $computername"
        $CimSession = New-CimSession @sessParams
        $tempSession = $True
    }
    catch {
        Write-Error $_
        #bail out
        return
    }
}

if ($CimSession) {
    $hash = [ordered]@{
        PSTypename   = "ServerStatus"
        Computername = $CimSession.computername.toUpper()
    }
    Try {
        $cimParams.classname = 'Win32_OperatingSystem'
        $cimParams.CimSession = $CimSession
        Write-Verbose "[$((Get-Date).TimeOfDay)] Using class $($cimParams.classname)"
        $OS = Get-CimInstance @cimParams
        $uptime = (Get-Date) - $OS.lastBootUpTime
        $hash.Add("Uptime", $uptime)

        $pctFreeMem = [math]::Round(($os.FreePhysicalMemory / $os.TotalVisibleMemorySi
        if ($AsInt) {
            $pctFreeMem = $pctFreeMem -as [int]
        }
        $hash.Add("PctFreeMem", $pctFreeMem)

        $cimParams.classname = 'Win32_Logicaldisk'
        $cimParams.filter = "deviceid='C:'"

        Write-Verbose "[$((Get-Date).TimeOfDay)] Using class $($cimParams.classname)"
```

```
                Get-CimInstance @cimParams | ForEach-Object {
                    $name = "PctFree{0}" -f $_.deviceid.substring(0, 1)
                    $pctFree = [math]::Round(($_.FreeSpace / $_.size) * 100, 2)
                    if ($AsInt) {
                        $pctFree = $pctFree -as [int]
                    }
                    $hash.add($name, $pctFree)
                }

                New-Object PSObject -Property $hash
            }
            catch {
                Write-Error $_
            }

            #only remove the CimSession if it was created in this function
            if ($tempSession) {
                Write-Verbose "[$((Get-Date).TimeOfDay)] Removing temporary CimSession"
                Remove-CimSession -CimSession $CimSession
            }
        } #if CimSession
    } #process

    End {
        Write-Verbose "[$((Get-Date).TimeOfDay)] Ending $($MyInvocation.MyCommand)"
    } #end
} #close function
```

```
Get-Help Get-ServerStatus
```

```
NAME
    Get-ServerStatus

SYNTAX
    Get-ServerStatus [[-Computername] <string>] [-Credential <pscredential>] [-AsInt]
    [<CommonParameters>]

    Get-ServerStatus [-CimSession <CimSession>] [-AsInt] [<CommonParameters>]


ALIASES
    gst


REMARKS
    None
```

The hashtable defines the type name.

```
$hash = [ordered]@{
    PSTypename   = "ServerStatus"
    Computername = $CimSession.computername.toUpper()
}
```

Let's try this without formatting.

In [30]:

```
Get-ServerStatus -OutVariable r
```

```
Computername Uptime      %FreeMem PctFreeC
------------ ------      -------- --------
PROSPERO     00.17:33:28    49.32    10.21
```

PowerShell defaulted to a table because there were five or fewer properties.

In [31]:

```
$r | Get-Member
```

```
   TypeName: ServerStatus

Name         MemberType   Definition
----         ----------   ----------
Equals       Method       bool Equals(System.Object obj)
GetHashCode  Method       int GetHashCode()
GetType      Method       type GetType()
ToString     Method       string ToString()
Computername NoteProperty string Computername=PROSPERO
PctFreeC     NoteProperty double PctFreeC=10.21
PctFreeMem   NoteProperty double PctFreeMem=49.32
Uptime       NoteProperty timespan Uptime=17:33:28.2568477
```

Creating Custom Formatting the Easy Way¶

Manually creating the formatting XML is a pain. I use New-PSFormatXML from the PSScriptTools module. You can create list or table views. All you need is a sample object with values for all the properties you want to include. I'll include all properties.

```
$r | New-PSFormatXML -path c:\temp\ServerStatus.format.ps1xml -FormatType Table
code c:\temp\ServerStatus.format.ps1xml
```

Format files typically follow the naming convention `typename.format.ps1xml`.

Use the custom format file to *add value* to your output. Here is my modified file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This file was created using the New-PSFormatXML command that is part
of the PSScriptTools module.
https://github.com/jdhitsolutions/PSScriptTools
-->
<Configuration>
```

```
<ViewDefinitions>
  <View>
    <Name>default</Name>
    <ViewSelectedBy>
      <TypeName>ServerStatus</TypeName>
    </ViewSelectedBy>
    <TableControl>
      <!--Delete the AutoSize node if you want to use the defined widths.-->
      <AutoSize />
      <TableHeaders>
        <TableColumnHeader>
          <Label>Computername</Label>
          <Width>15</Width>
          <Alignment>left</Alignment>
        </TableColumnHeader>
        <TableColumnHeader>
          <Label>Uptime</Label>
          <Width>21</Width>
          <Alignment>left</Alignment>
        </TableColumnHeader>
        <TableColumnHeader>
          <!--Customized column header-->
          <Label>%FreeMem</Label>
          <Width>13</Width>
          <!-- Customized alignment -->
          <Alignment>right</Alignment>
        </TableColumnHeader>
        <TableColumnHeader>
          <Label>PctFreeC</Label>
          <Width>11</Width>
          <Alignment>right</Alignment>
        </TableColumnHeader>
      </TableHeaders>
      <TableRowEntries>
        <TableRowEntry>
          <TableColumnItems>
            <TableColumnItem>
              <PropertyName>Computername</PropertyName>
            </TableColumnItem>
            <TableColumnItem>
              <!--Customized output-->
              <ScriptBlock>
                "{0:dd\.hh\:mm\:ss}" -f $_.Uptime
              </ScriptBlock>
            </TableColumnItem>
            <TableColumnItem>
              <PropertyName>PctFreeMem</PropertyName>
            </TableColumnItem>
```

```
              <TableColumnItem>
                <PropertyName>PctFreeC</PropertyName>
              </TableColumnItem>
            </TableColumnItems>
          </TableRowEntry>
        </TableRowEntries>
      </TableControl>
    </View>
  </ViewDefinitions>
</Configuration>
```

Watch your casing on `<ScriptBlock>` nodes.

Use `Update-FormatData` to load the custom format file.

In [32]:

```
Update-FormatData .\serverstatus.format.ps1xml
```

Validate formatting with `Get-FormatData`.

In [33]:

```
Get-FormatData -TypeName ServerStatus
```

```
TypeNames      FormatViewDefinition
---------      --------------------
{ServerStatus} {default, default}
```

In [34]:

```
(Get-FormatData -TypeName ServerStatus).FormatViewDefinition
```

```
Name    Control
----    -------
default System.Management.Automation.TableControl
default System.Management.Automation.TableControl
```

In [35]:

```
(Get-FormatData serverstatus).FormatViewDefinition.Control
```

```
Headers          : {System.Management.Automation.TableControlColumnHeader,
                   System.Management.Automation.TableControlColumnHeader,
                   System.Management.Automation.TableControlColumnHeader,
                   System.Management.Automation.TableControlColumnHeader}
Rows             : {System.Management.Automation.TableControlRow}
AutoSize         : True
HideTableHeaders : False
GroupBy          :
OutOfBand        : False

Headers          : {System.Management.Automation.TableControlColumnHeader,
                   System.Management.Automation.TableControlColumnHeader,
                   System.Management.Automation.TableControlColumnHeader,
                   System.Management.Automation.TableControlColumnHeader}
Rows             : {System.Management.Automation.TableControlRow}
AutoSize         : True
HideTableHeaders : False
GroupBy          :
OutOfBand        : False
```

In [36]:

```
(Get-FormatData serverstatus).FormatViewDefinition.Control.Headers
# Width is ignored because I am using AutoSize - see above
```

```
Label        Alignment Width
-----        --------- -----
Computername      Left    15
Uptime            Left    21
%FreeMem         Right    13
PctFreeC         Right    11
Computername      Left    15
Uptime            Left    21
%FreeMem         Right    13
%FreeC           Right    11
```

In [37]:

```
(Get-FormatData serverstatus).FormatViewDefinition.Control.Rows.Columns
```

```
Alignment DisplayEntry
--------- ------------
Undefined property: Computername
Undefined script: ...
Undefined property: PctFreeMem
Undefined property: PctFreeC
Undefined property: Computername
Undefined script: ...
Undefined script: ...
Undefined script: ...
```

The formatting is immediate and persistent for the duration of my session.

In [38]:

```
Get-ServerStatus
```

```
Computername Uptime      %FreeMem PctFreeC
------------ ------      -------- --------
PROSPERO     00.17:33:29    49.27    10.21
```

The output is easier to read.

- Formatted Uptime time span to strip off milliseconds
- Custom header `%FreeMem`
- Aligned values

But why stop there? I can use custom formatting to add value and information.

```
<TableColumnItem>
  <ScriptBlock>
  <!--Switch statements don't appear to work properly in script blocks-->
  if ($_.PctFreeMem -le 30) {
      <!--alert-->
      <!--Or use $PSStyle-->
      $Style = "`e[5;38;5;197m"
    }
    elseif ($_.PctFree -le 60) {
      <!--Warning-->
      $Style = "`e[38;5;216m"
    }
    else {
      <!--OK-->
      $Style = "`e[38;5;155m"
    }
    <!--script block output-->
    "$Style$($_.PctFreeMem)$($PSStyle.Reset)"
  </ScriptBlock>
</TableColumnItem>
```

I have defined script blocks to display `%FreeMem` in different colors based on the value.

In [39]:

```
# load the updated format file
Update-FormatData -append .\ServerStatus.Format2.ps1xml
```

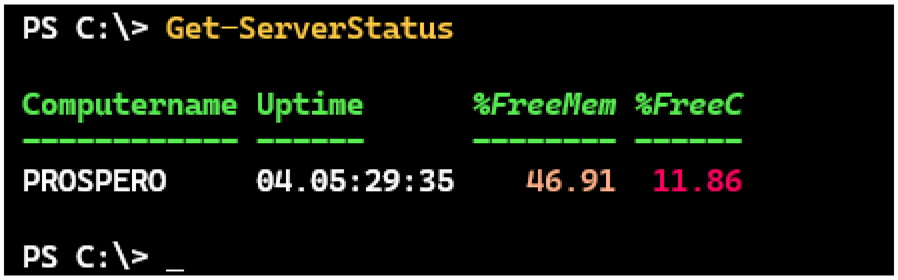
This should overwrite the previous formatting directives.

In [40]:

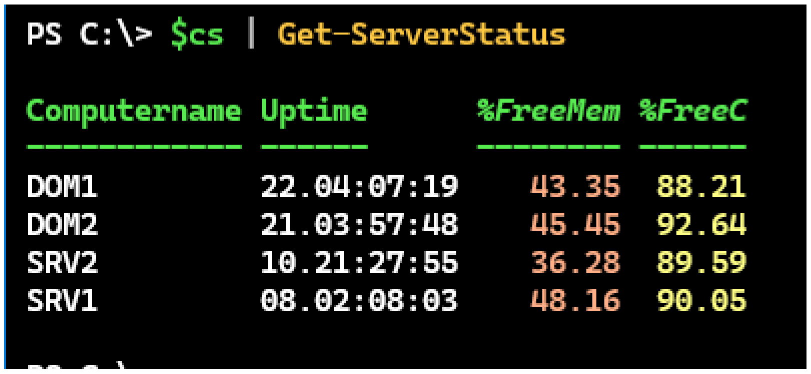
```
Get-ServerStatus
#this may not format properly in VSCode
```

```
Computername Uptime      %FreeMem PctFreeC
------------ ------      -------- --------
PROSPERO     00.17:33:30     49.3    10.21
```

Percent Free values are now color-coded based on the value. Low values will blink.



```
PS C:\> Get-ServerStatus

Computername Uptime      %FreeMem %FreeC
------------ ------      -------- ------
PROSPERO     04.05:29:35    46.91  11.86

PS C:\> _
```



```
PS C:\> $cs | Get-ServerStatus

Computername Uptime       %FreeMem %FreeC
------------ ------       -------- ------
DOM1         22.04:07:19     43.35  88.21
DOM2         21.03:57:48     45.45  92.64
SRV2         10.21:27:55     36.28  89.59
SRV1         08.02:08:03     48.16  90.05
```

Another Example¶

Here is a function that writes a larger rich object to the pipeline.

```
[PSCustomObject]@{
    PSTypeName        = 'PSServerDetail'
    Computername      = $os.CSName
    OperatingSystem   = $os.Caption
    InstallDate       = $os.InstallDate
    Memory            = $os.TotalVisibleMemorySize
    FreeMemory        = $os.FreePhysicalMemory
    RunningProcesses = $os.NumberOfProcesses - 2  #subtract System and Idle processes
    RunningServices   = $svc.Count
    LastBoot          = $os.LastBootUpTime
    Shares            = $shares
}
```

Let's see it. The script file also defines a few type extensions for the object such as alias properties.

In [41]:

```
. .\Get-ServerDetail.ps1
$n = Get-ServerDetail
$n
```

```
   Server: PROSPERO [Microsoft Windows 11 Pro]

LastBoot             Uptime      MemGB Processes Services
--------             ------      ----- --------- --------
3/5/2025 7:42:37 PM 00.17:33:31    64        390      161
```

I want a default table view. This is a great way to prototype.

In [42]:

```
$n | Format-Table -GroupBy ComputerName -Property LastBoot,Uptime,
@{Name="MemGB";Expression={$_.Memory/1mb -as [int]}},
@{Name="Processes";Expression={$_.RunningProcesses}},
@{Name="Services";Expression = {$_.RunningServices}}
```

```
   Computername: PROSPERO
```

```
LastBoot             Uptime           MemGB Processes Services
--------             ------           ----- --------- --------
3/5/2025 7:42:37 PM 17:33:31.1702468    64       390      161
```

I will create my format file. `New-PSFormatXML` will use the expression script blocks in the XML file.

```
$n | New-PSFormatXML -GroupBy ComputerName -Properties LastBoot,Uptime,
@{Name="MemGB";Expression={$_.Memory/1mb -as [int]}},
@{Name="Processes";Expression={$_.RunningProcesses}},
@{Name="Services";Expression = {$_.RunningServices}} -Path .\PSServerDetail.format.ps1xml
```

I customized the grouping in the XML file.

```
<GroupBy>
  <ScriptBlock>
  <!--Display domain controllers with a different color-->
    if ($_.Computername -Match "dom") {
      $fg = "`e[1;38;5;48m"
    }
    else {
      $fg = "`e[1;38;5;147m"
    }
    <!--output-->
    "$fg{0}`e[0m [`e[3m{1}`e[0m]" -f $_.ComputerName,$_.OS.replace("Evaluation","")
    </ScriptBlock>
  <Label>Server</Label>
</GroupBy>
```

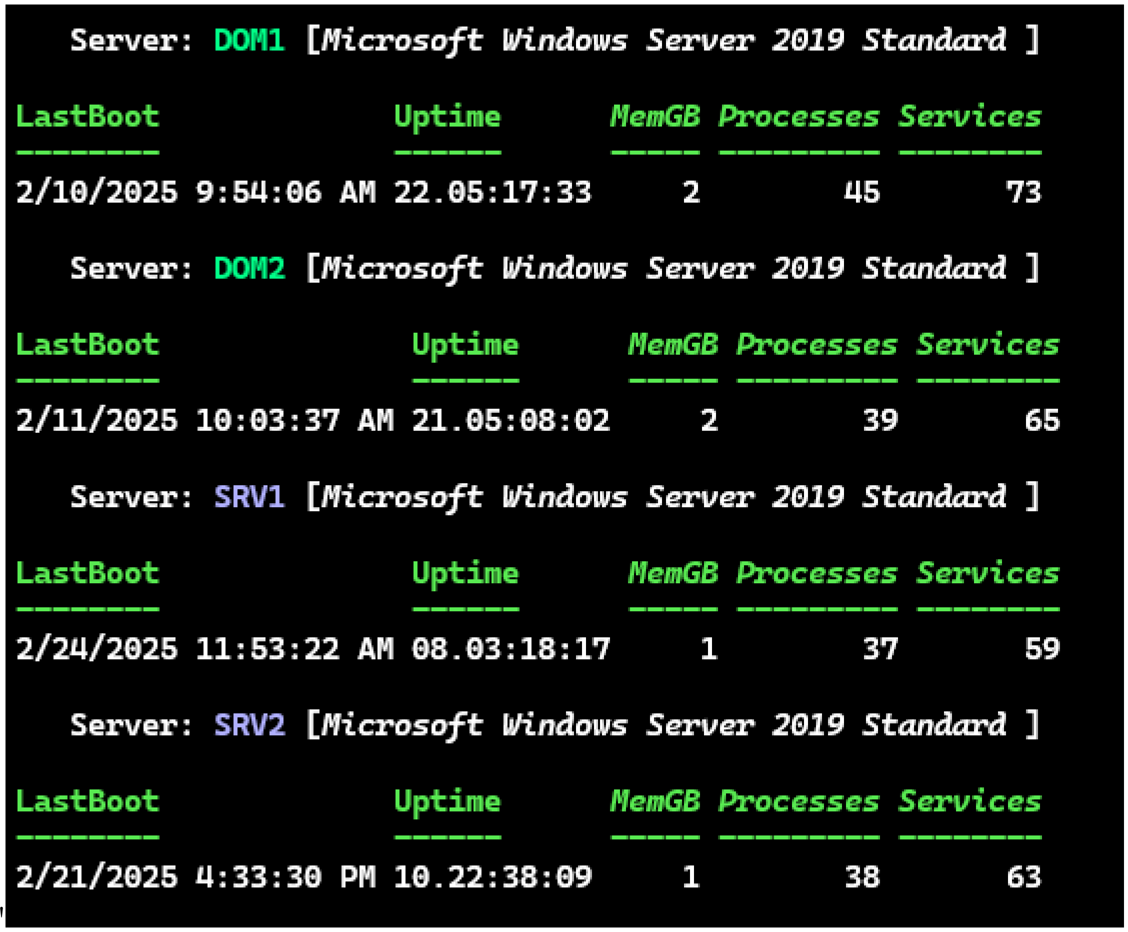
And made other minor adjustments.

In [43]:

```
Update-FormatData .\PSServerDetail.format.ps1xml
$n
```

```
   Server: PROSPERO [Microsoft Windows 11 Pro]

LastBoot             Uptime      MemGB Processes Services
--------             ------      ----- --------- --------
3/5/2025 7:42:37 PM 00.17:33:31    64       390      161
```

Here's an example from my test domain that better displays the custom formatting.



```
   Server: DOM1 [Microsoft Windows Server 2019 Standard ]

LastBoot             Uptime      MemGB Processes Services
--------             ------      ----- --------- --------
2/10/2025 9:54:06 AM 22.05:17:33     2        45       73

   Server: DOM2 [Microsoft Windows Server 2019 Standard ]

LastBoot              Uptime      MemGB Processes Services
--------              ------      ----- --------- --------
2/11/2025 10:03:37 AM 21.05:08:02     2        39       65

   Server: SRV1 [Microsoft Windows Server 2019 Standard ]

LastBoot              Uptime      MemGB Processes Services
--------              ------      ----- --------- --------
2/24/2025 11:53:22 AM 08.03:18:17     1        37       59

   Server: SRV2 [Microsoft Windows Server 2019 Standard ]

LastBoot             Uptime      MemGB Processes Services
--------             ------      ----- --------- --------
2/21/2025 4:33:30 PM 10.22:38:09     1        38       63
```

I think this is easier to read and more informative than the default table view.

Custom Views¶

I have a default table view. I can also create a default list view and add it to the same formatting file.

```
$n  | New-PSFormatXML -Append -Path .\PSServerDetail.format.ps1xml -FormatType List `
 -properties Computername,OS,Memory,RunningProcesses,RunningServices,LastBoot,Uptime
```

The currently loaded format file already has this view.

In [44]:

```
$n | Format-List
```

```
Computername : PROSPERO
OS           : Microsoft Windows 11 Pro
MemoryGB     : 64
Processes    : 390
Services     : 161
LastBoot     : 3/5/2025 7:42:37 PM
Uptime       : 00.17:33:31
```

I can create additional views so that I don't have to run commands like this:

```
$n | Select Computername,
@{Name="MemGB";Expression = {$_.Memory/1mb -as [int]}},
@{Name="FreeMemGB";Expression= {$_.FreeMemory/1mb -as [int]}},
@{Name="PctFreeMem";Expression={($_.FreeMemory/$_.Memory)*100}}
```

Instead, I'll define a custom view.

```
$n | New-PSFormatXML -Append -Path .\PSServerDetail.format.ps1xml -ViewName memory `
 -FormatType Table -properties Computername,
@{Name="MemGB";Expression = {$_.Memory/1mb -as [int]}},
@{Name="FreeMemGB";Expression= {$_.FreeMemory/1mb -as [int]}},
@{Name="PctFreeMem";Expression={($_.FreeMemory/$_.Memory)*100}}
```

In [ ]:

```
code .\PSServerDetail.format.ps1xml
```

In [ ]:

```
$n | Format-Table -view memory
```

```
Computername MemGB FreeMemGB PctFreeMem
------------ ----- --------- ----------
PROSPERO        64        31      47.94
```

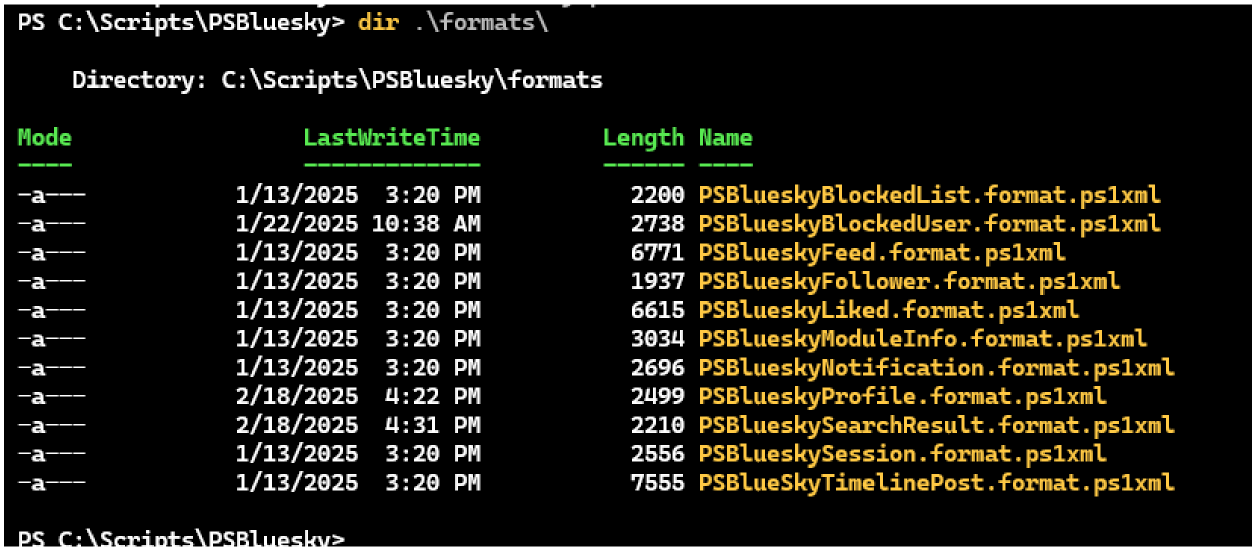
You could also create custom type extensions or property sets.

Modules and Custom Formats¶

For stand-alone functions I typically insert this code at the end of the script.

```
Update-FormatData -AppendPath $PSScriptRoot\PSServerDetail.format.ps1xml
```

For modules, I typically store format files in a subfolder. For example, these are the files for the PSBluesky module



```
PS C:\Scripts\PSBluesky> dir .\formats\

    Directory: C:\Scripts\PSBluesky\formats

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          1/13/2025  3:20 PM           2200 PSBlueskyBlockedList.format.ps1xml
-a---          1/22/2025 10:38 AM           2738 PSBlueskyBlockedUser.format.ps1xml
-a---          1/13/2025  3:20 PM           6771 PSBlueskyFeed.format.ps1xml
-a---          1/13/2025  3:20 PM           1937 PSBlueskyFollower.format.ps1xml
-a---          1/13/2025  3:20 PM           6615 PSBlueskyLiked.format.ps1xml
-a---          1/13/2025  3:20 PM           3034 PSBlueskyModuleInfo.format.ps1xml
-a---          1/13/2025  3:20 PM           2696 PSBlueskyNotification.format.ps1xml
-a---          2/18/2025  4:22 PM           2499 PSBlueskyProfile.format.ps1xml
-a---          2/18/2025  4:31 PM           2210 PSBlueskySearchResult.format.ps1xml
-a---          1/13/2025  3:20 PM           2556 PSBlueskySession.format.ps1xml
-a---          1/13/2025  3:20 PM           7555 PSBlueSkyTimelinePost.format.ps1xml

PS C:\Scripts\PSBluesky>
```

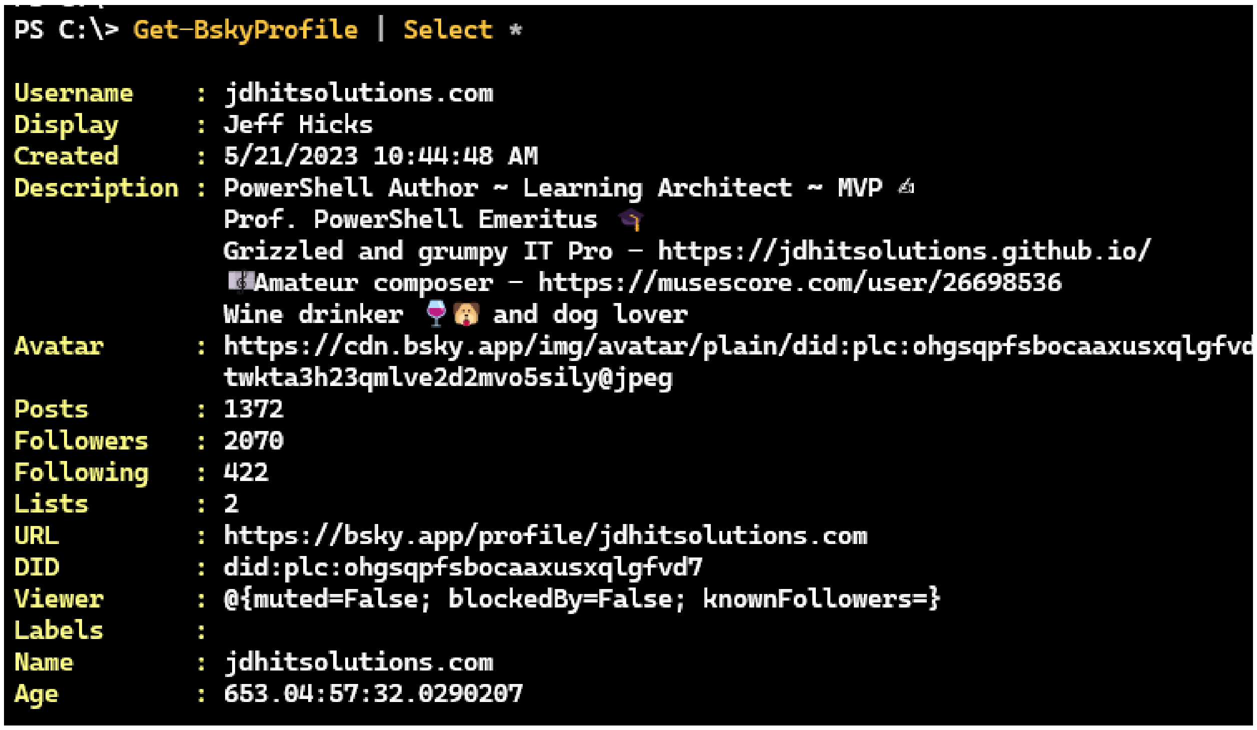
The files are loaded in the module manifest.

```
FormatsToProcess     = @(
    'formats\PSBlueSkyTimelinePost.format.ps1xml',
    'formats\PSBlueskyBlockedUser.format.ps1xml',
    'formats\PSBlueskyBlockedList.format.ps1xml',
    'formats\PSBlueskyProfile.format.ps1xml',
    'formats\PSBlueskyFollower.format.ps1xml',
    'formats\PSBlueskyFeed.format.ps1xml',
    'formats\PSBlueskyLiked.format.ps1xml',
    'formats\PSBlueskySession.format.ps1xml',
    'formats\PSBlueskyNotification.format.ps1xml',
    'formats\PSBlueskySearchResult.format.ps1xml',
    'formats\PSBlueskyModuleInfo.format.ps1xml'
)
```

I've done **a lot** of formatting customization, including true custom formatting.

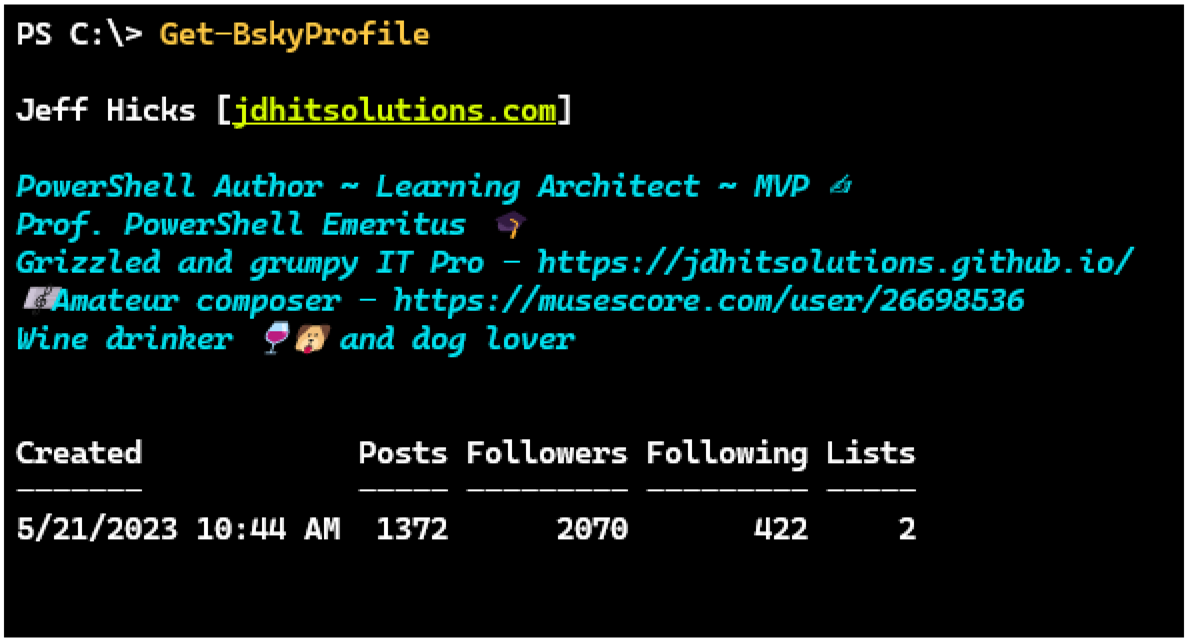
```
<CustomControl>
    <CustomEntries>
        <CustomEntry>
            <CustomItem>
                <ExpressionBinding>
                    <ScriptBlock>
                    <!--
                        18 Feb 2025 Added optional code to use pwshSpectreConsole module
                        to display the user's avatar. This will only work if the module
                        is installed and the console properly configured.
                    -->
                    Try {
                        $avt = (Get-SpectreImage $_.avatar -MaxWidth 5 -errorAction Stop |
                        Out-SpectreHost).Trim()
                    }
                    Catch {
                        $avt = $Null
                    }
                    "{2} {0} [$($bskyPreferences['UserName']){1}$($PSStyle.Reset)]" -f `
                    $_.Display.trim(),$($PSStyle.FormatHyperLink($_.UserName,$_.Url)),
                    $avt
                    </ScriptBlock>
                </ExpressionBinding>
...
```

The Bluesky profile object is rich in properties.



```
PS C:\> Get-BskyProfile | Select *

Username    : jdhitsolutions.com
Display     : Jeff Hicks
Created     : 5/21/2023 10:44:48 AM
Description : PowerShell Author ~ Learning Architect ~ MVP ✍
              Prof. PowerShell Emeritus 🎓
              Grizzled and grumpy IT Pro - https://jdhitsolutions.github.io/
              🎼Amateur composer - https://musescore.com/user/26698536
              Wine drinker 🍷🐶 and dog lover
Avatar      : https://cdn.bsky.app/img/avatar/plain/did:plc:ohgsqpfsbocaaxusxqlgfvd
              twkta3h23qmlve2d2mvo5sily@jpeg
Posts       : 1372
Followers   : 2070
Following   : 422
Lists       : 2
URL         : https://bsky.app/profile/jdhitsolutions.com
DID         : did:plc:ohgsqpfsbocaaxusxqlgfvd7
Viewer      : @{muted=False; blockedBy=False; knownFollowers=}
Labels      :
Name        : jdhitsolutions.com
Age         : 653.04:57:32.0290207
```

But easier to consume with custom formatting.



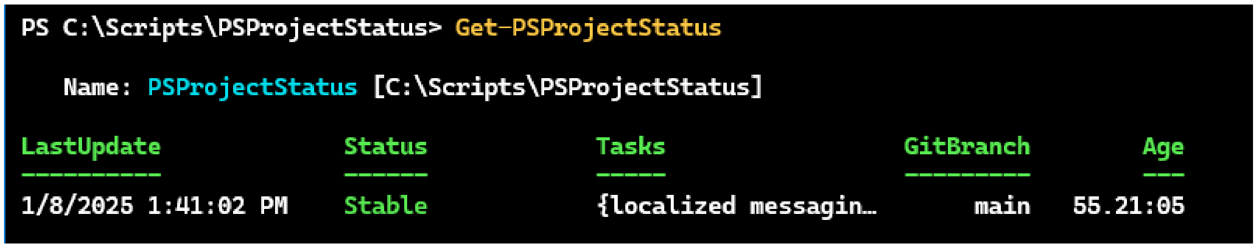
```
PS C:\> Get-BskyProfile

Jeff Hicks [jdhitsolutions.com]

PowerShell Author ~ Learning Architect ~ MVP ✍
Prof. PowerShell Emeritus 🎓
Grizzled and grumpy IT Pro - https://jdhitsolutions.github.io/
🎼Amateur composer - https://musescore.com/user/26698536
Wine drinker 🍷🐶 and dog lover


Created              Posts Followers Following Lists
-------              ----- --------- --------- -----
5/21/2023 10:44 AM    1372      2070       422     2
```

Formatting includes hyperlinks created with `$PSStyle`.

```
"{2} {0} [$($bskyPreferences['UserName']){1}$($PSStyle.Reset)]" -f $_.Display.trim(),
$($PSStyle.FormatHyperLink($_.UserName,$_.Url)),$avt
111
```

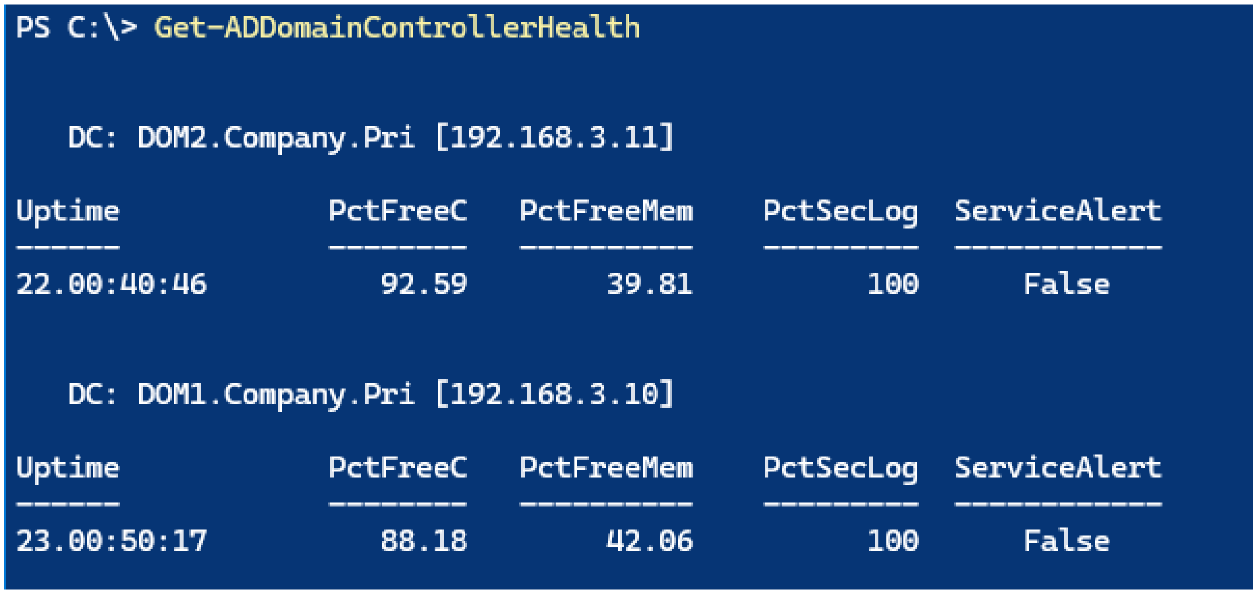
Other Module Examples¶

- PSProjectStatus (https://github.com/jdhitsolutions/PSProjectStatus/tree/main/formats)



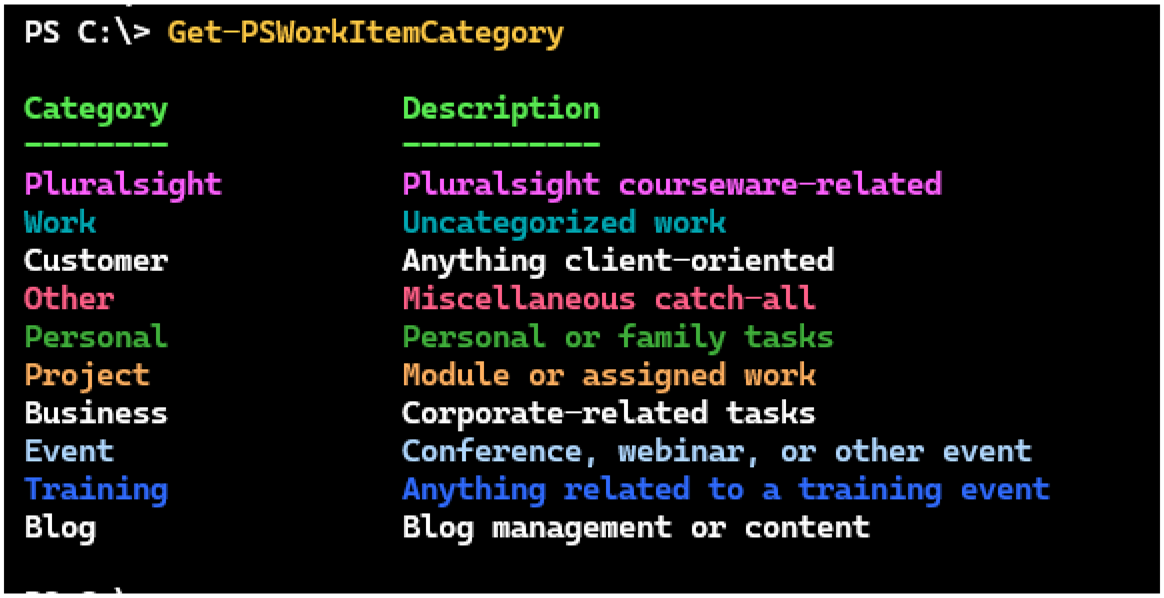
```
PS C:\Scripts\PSProjectStatus> Get-PSProjectStatus

   Name: PSProjectStatus [C:\Scripts\PSProjectStatus]

LastUpdate             Status           Tasks                  GitBranch        Age
----------             ------           -----                  ---------        ---
1/8/2025 1:41:02 PM    Stable           {localized messagin…        main   55.21:05
```

- AD Reporting Tools (https://github.com/jdhitsolutions/ADReportingTools/tree/main/formats)



```
PS C:\> Get-ADDomainControllerHealth


   DC: DOM2.Company.Pri [192.168.3.11]

Uptime             PctFreeC   PctFreeMem    PctSecLog  ServiceAlert
------             --------   ----------    ---------  ------------
22.00:40:46           92.59        39.81          100       False


   DC: DOM1.Company.Pri [192.168.3.10]

Uptime             PctFreeC   PctFreeMem    PctSecLog  ServiceAlert
------             --------   ----------    ---------  ------------
23.00:50:17           88.18        42.06          100       False
```

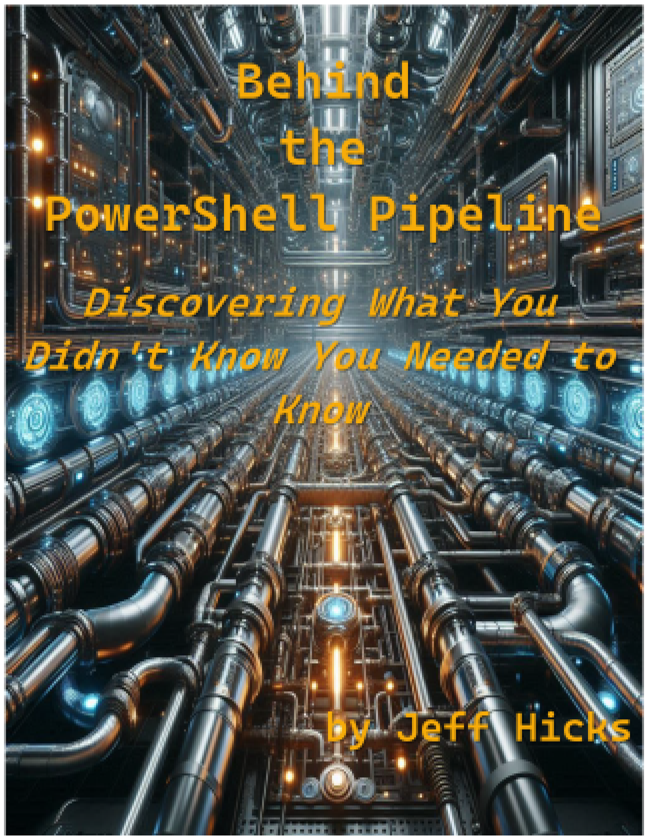
- PSWorkItem (https://github.com/jdhitsolutions/PSWorkItem/tree/main/formats)



```
PS C:\> Get-PSWorkItemCategory

Category            Description
--------            -----------
Pluralsight         Pluralsight courseware-related
Work                Uncategorized work
Customer            Anything client-oriented
Other               Miscellaneous catch-all
Personal            Personal or family tasks
Project             Module or assigned work
Business            Corporate-related tasks
Event               Conference, webinar, or other event
Training            Anything related to a training event
Blog                Blog management or content
```

Behind the PowerShell Pipeline¶

Knowing how and when to use a feature like custom formatting isn't always obvious or clearly documented. That's why I wrote this.


Behind the PowerShell Pipeline
Discovering What You Didn't Know You Needed to Know
by Jeff Hicks

Available on Leanpub at https://leanpub.com/behind-the-pspipeline. The book is drawn from my premium PowerShell newsletter [https://buttondown.com/behind-the-powershell-pipeline] I started three years ago.

Questions and Answers¶

Session materials can be found at https://github.com/jdhitsolutions/PSCustomFormatting. I have enabled Discussions for follow-up questions.

THANK
YOU