Getting Started with PowerShell (PS) and Windows PowerShell Integrated Scripting Environment (ISE)

NIKOLA PETROVSKI

27 MAY 2018

1.0 Windows PowerShell

The purpose of this lab is to get familiar with the following software products: Windows PowerShell and Windows PowerShell Integrated Scripting Environment (ISE). It provides step-by-step pictorial instructions for scripting in the command-line environment that is available in Windows Server 2012 R2 operating system.

Scripts represent simple and automated approaches for performing repetitive administrative functions. Windows PowerShell contains cmdlets that run in the command-line environment. A group of related cmdlets form a script. Modules are created for applications to manage all their service aspects for example print management and network adapter configurations. One of the existing Windows PowerShell features is tab-completion that alleviates typing long lines of text. Information about each Windows PowerShell cmdlet is accessible by typing a simple "Get-Help" cmdlet. Cmdlet is a simple Windows PowerShell script that performs a single function only.

Windows PowerShell cmdlets follow the verb-noun syntax. Every noun has a collection of verbs that is related to it. Commonly used verbs in cmdlets are:

- Get
- New
- Set
- Restart
- Resume
- Stop
- Suspend
- Clear
- Limit
- Remove
- Add
- Show
- Write

It is possible to display all the verbs related to a particular Windows PowerShell noun by typing: Get-Command-Noun. The presentation of all the nouns related to a particular Windows PowerShell verb is performed by typing: Get-Command-Verb. To display all parameters related to one cmdlet, type: Get-Help cmdlet. For trying if a certain cmdlet exists type: Get-Command cmdlet.

NOTE: All Windows PowerShell parameters are typed with "-" prefix. Each cmdlet can be related to a specific set of parameters.

This example represents a start of a background job (playing audio beeps):

```
PS C:\Users\Nikola Petrovski> start-job {
>> [console]::beep(440,500)
>> [console]::beep(440,500)
>> [console]::beep(440,500)
>> [console]::beep(349,350)
>> [console]::beep(523,150)
>> [console]::beep(440,500)
>> [console]::beep(349,350)
>> [console]::beep(523,150)
>> [console]::beep(440,1000)
>> [console]::beep(659,500)
>> [console]::beep(659,500)
>> [console]::beep(659,500)
>> [console]::beep(698,350)
>> [console]::beep(523,150)
>> [console]::beep(415,500)
>> [console]::beep(349,350)
>> [console]::beep(523,150)
>> [console]::beep(440,1000)
>>  }

Id     Name            PSJobTypeName   State         HasMoreData     Location             Command
--     ----            -------------   -----         -----------     --------             -------
5      Job5            BackgroundJob   Running       True            localhost            ...


PS C:\Users\Nikola Petrovski>
```

npetrovski 0:57 27/5/18

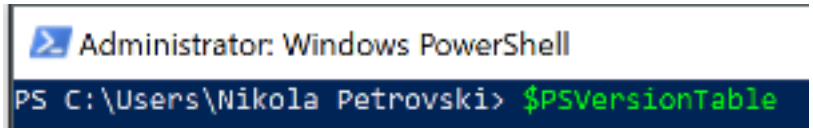
To display the current week in the year :

```
PS C:\Users\Nikola Petrovski> get-Date -uformat %W
21
PS C:\Users\Nikola Petrovski>
```

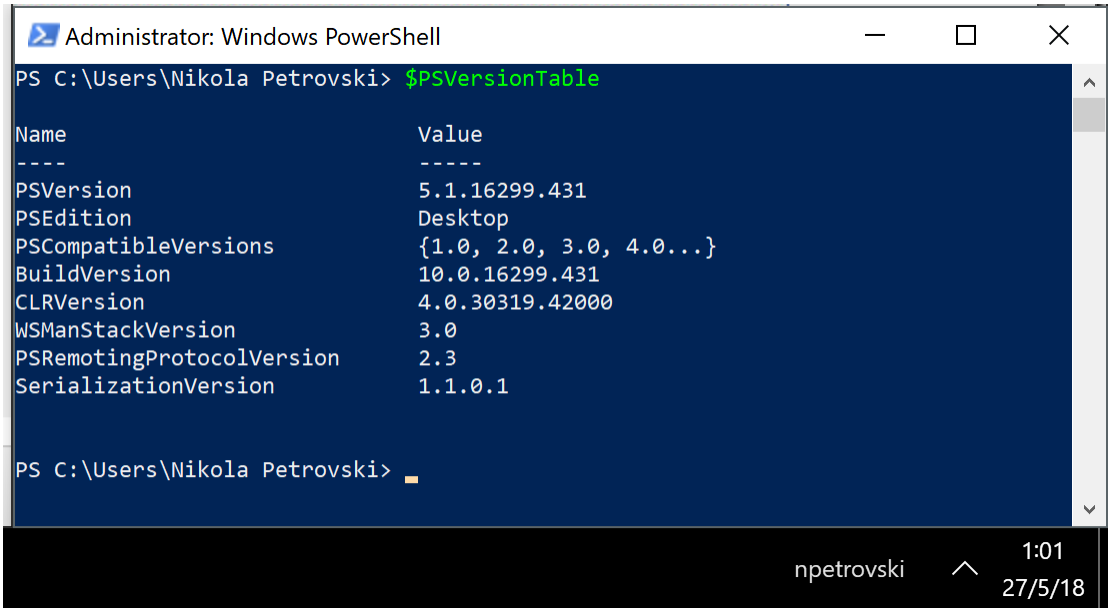
npetrovski 0:57 27/5/18

Example of tab complete:

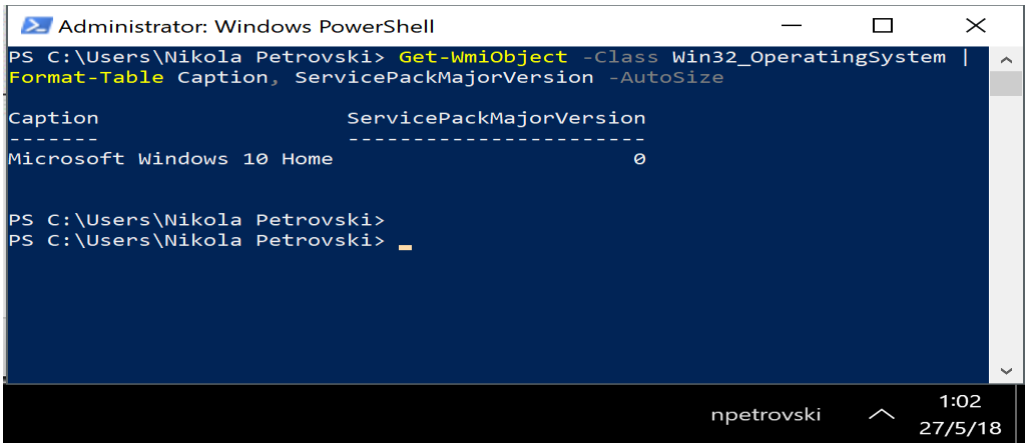
press tab



press enter



To display software version:

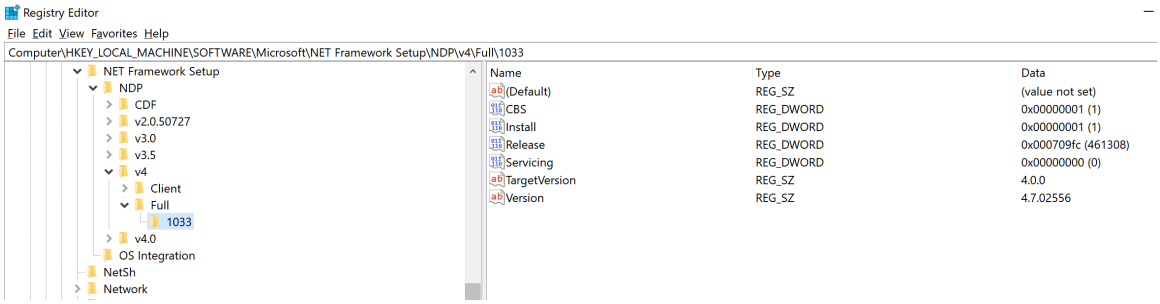


To confirm the software version number is “4.7” not “4.5”:

```
PS C:\Users\Nikola Petrovski>
PS C:\Users\Nikola Petrovski> (Get-ItemProperty -Path 'HKLM:\Software\Microsoft\NET Framework Setup\NDP\v4\Full'
-ErrorAction SilentlyContinue).Version -like '4.5*'
False
PS C:\Users\Nikola Petrovski>
PS C:\Users\Nikola Petrovski> (Get-ItemProperty -Path 'HKLM:\Software\Microsoft\NET Framework Setup\NDP\v4\Full'
-ErrorAction SilentlyContinue).Version -like '4.5*'
False
PS C:\Users\Nikola Petrovski> (Get-ItemProperty -Path 'HKLM:\Software\Microsoft\NET Framework Setup\NDP\v4\Full'
-ErrorAction SilentlyContinue).Version -like '4.7*'
True
PS C:\Users\Nikola Petrovski>
```

npetrovski 1:14 27/5/18

Using the Registry Editor is an alternate method for checking the software version :



This example displays all installed programs, their version, publisher and install date:

```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> Get-ItemProperty HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\* | Select-Object DisplayName, DisplayVersion, Publisher, InstallDate |
Format-Table -AutoSize

DisplayName                                                     DisplayVersion   Publisher                                          InstallDate
-----------                                                     --------------   ---------                                          -----------
Intel(R) Virtual Buttons                                        1.1.0.21         Intel Corporation
Visual Studio Enterprise 2017                                   15.5.27130.2027  Microsoft Corporation                              20180123
                                                                                 Adobe Systems Incorporated
Adobe Creative Cloud                                            4.5.0.331        Adobe Systems Incorporated
Adobe Flash Player 29 NPAPI                                     29.0.0.171       Adobe Systems Incorporated
Adobe Flash Player 29 PPAPI                                     29.0.0.171       Adobe Systems Incorporated
Adobe Shockwave Player 12.1                                     12.1.9.159       Adobe Systems, Inc.
                                                                                 Adobe Systems Incorporated                         4/30/2018
                                                                                 Adobe Systems Incorporated
Colasoft Packet Builder 1.0                                     1.0              Colasoft LLC.                                      20170725
npetrovski   1:19 27/5/18
```

To export this list in a text file the path to the text file must be followed by "." :

```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> Get-ItemProperty HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\* | Select-Object DisplayName, DisplayVersion, Publisher, InstallDate |
Format-Table -AutoSize > C:\InstalledPrograms-PS.txt.
PS C:\Users\Nikola Petrovski> NOTE: The period (dot)^ is Mandatory!
npetrovski   1:43 27/5/18
```

To compare two different lists file the previous example is used to create two separate files first and then PS allows pointing out the files differences by using (=>) or (<=):

```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> Compare-Object -ReferenceObject (Get-Content C:\InstalledPrograms-PS.txt) -DifferenceObject (Get-Content C:\InstalledPrograms-PS-Copy.txt)

InputObject
-----------
                                                                 Adobe Systems Incorporated
...
Adobe Creative Cloud                                             4.5.0.331        Adobe Systems Incorporated
                                                                                  Adobe Systems Incorporated


PS C:\Users\Nikola Petrovski>
npetrovski   1:49 27/5/18
```

In addition to Get-Command, the query can be narrowed by adding parameter(s) e.g., IP:

```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> Get-Command -Name *IP*

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           gip ->                                             1.0.0.0    NetTCPIP
Alias           ipal -> Import-Alias
Alias           ipcsv -> Import-Csv
Alias           ipmo -> Import-Module
Alias           ipsn -> Import-PSSession
Function        Add-NetIPHttpsCertBinding                          1.0.0.0    NetworkTransition
Function        Copy-NetIPsecMainModeCryptoSet                     2.0.0.0    NetSecurity
Function        Copy-NetIPsecMainModeRule                          2.0.0.0    NetSecurity
Function        Copy-NetIPsecPhase1AuthSet                         2.0.0.0    NetSecurity
Function        Copy-NetIPsecPhase2AuthSet                         2.0.0.0    NetSecurity
Function        Copy-NetIPsecQuickModeCryptoSet                    2.0.0.0    NetSecurity
npetrovski   1:54
             27/5/18
```

To query for specific module use:

```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> Get-Command -Module NetTCPIP -Name *IP*

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           gip -> Get-NetIPConfiguration                      1.0.0.0    NetTCPIP
Function        Get-NetIPAddress                                   1.0.0.0    NetTCPIP
Function        Get-NetIPConfiguration                             1.0.0.0    NetTCPIP
Function        Get-NetIPInterface                                 1.0.0.0    NetTCPIP
Function        Get-NetIPv4Protocol                                1.0.0.0    NetTCPIP
Function        Get-NetIPv6Protocol                                1.0.0.0    NetTCPIP
Function        New-NetIPAddress                                   1.0.0.0    NetTCPIP
Function        Remove-NetIPAddress                                1.0.0.0    NetTCPIP
Function        Set-NetIPAddress                                   1.0.0.0    NetTCPIP
Function        Set-NetIPInterface                                 1.0.0.0    NetTCPIP
Function        Set-NetIPv4Protocol                                1.0.0.0    NetTCPIP
npetrovski   1:55
             27/5/18
```

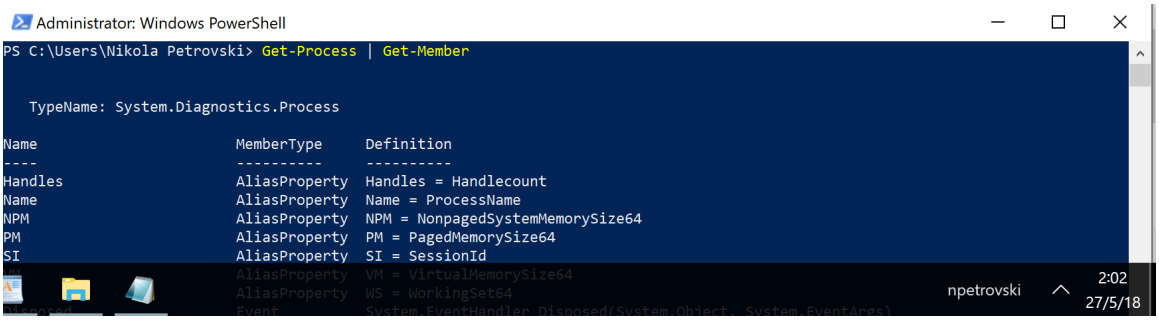
To get help:

```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> Get-Help Get-Process

NAME
    Get-Process

SYNOPSIS
    Gets the processes that are running on the local computer or a remote computer.


SYNTAX
    Get-Process [[-Name] <String[]>] [-ComputerName <String[]>] [-FileVersionInfo] [-Module] [<CommonParameters>]
    Get-Process [-ComputerName <String[]>] [-FileVersionInfo] -Id <Int32[]> [-Module] [<CommonParameters>]
    Get-Process [-ComputerName <String[]>] [-FileVersionInfo] -InputObject <Process[]> [-Module] [<CommonParameters>]
npetrovski   1:58
             27/5/18
```

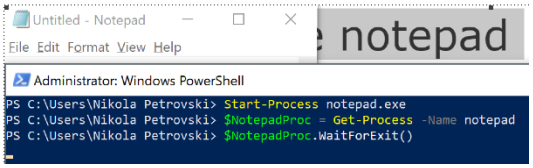
To get the properties and methods of objects:

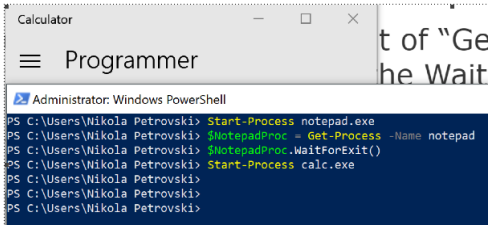
```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> Get-Help Get-Member

NAME
    Get-Member

SYNOPSIS
    Gets the properties and methods of objects.


SYNTAX
    Get-Member [[-Name] <String[]>] [-Force] [-InputObject <PSObject>] [-MemberType {AliasProperty | CodeProperty | Property |
    NoteProperty | ScriptProperty | Properties | PropertySet | Method | CodeMethod | ScriptMethod | Methods |
    ParameterizedProperty | MemberSet | Event | Dynamic | All}] [-Static] [-View {Extended | Adapted | Base | All}]
    [<CommonParameters>]
npetrovski   2:00
             27/5/18
```

To get a list of processes/objects that are running on the local/remote computer and their properties and methods:

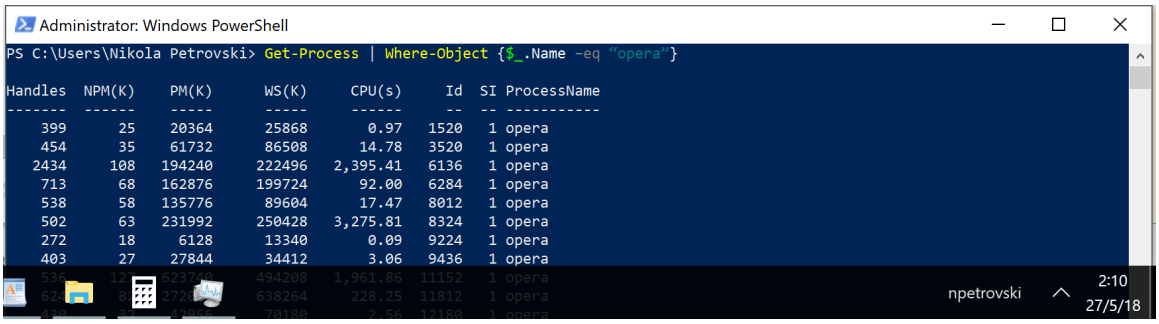


This example shows how to code a script that opens the Notepad, waits for user input and when the notepad window is closed by the user it opens the Calculator:

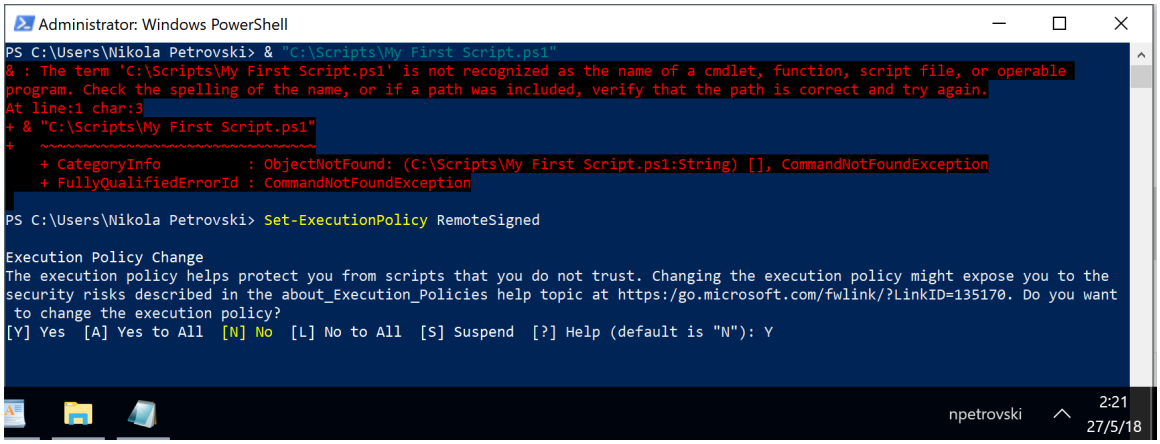




This example lists processes that match with Opera criteria:

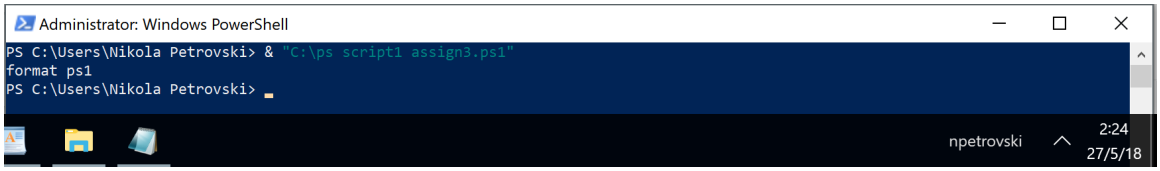


This example is a notepad file saved in ps1 format that can run in the PS:



NOTE: PS execution policy settings may prevent a script from running. Setting it to RemoteSigned makes the script runnable.

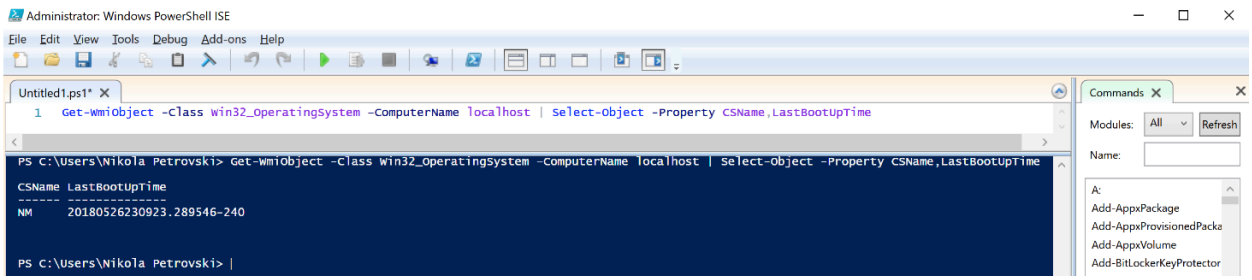
Running a script that displays “format ps1”:



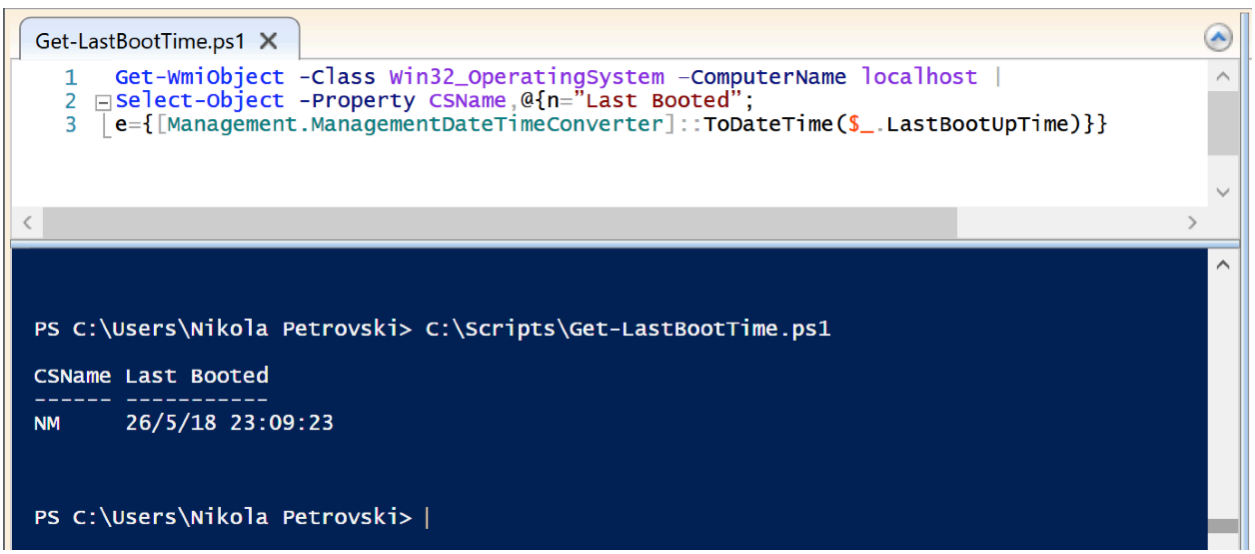
2.0 Windows PowerShell Integrated Scripting Environment (ISE)

Windows PowerShell IDE alleviates the use of Windows PowerShell. It has a built-in mechanism for typing autocomplete and can provide all the information about each cmdlet and its respective parameters. It can execute PowerShell commands and save scripts for later usage. It highlights the keywords and debugs scripts.

This screenshot shows the interface for displaying and running the scripts :



To make the Last Booted date more human readable modify the script to state:



Revising the previous script to allow cross computers application of the script:

Get-LastBootTime.ps1

```
1  param(
2   [string]$ComputerName
3  )
4
5   Get-WmiObject -Class Win32_OperatingSystem -ComputerName $ComputerName |
6  Select-Object -Property CSName,@{n="Last Booted";
7   e={[Management.ManagementDateTimeConverter]::ToDateTime($_.LastBootUpTime)}}
```

```
PS C:\Users\Nikola Petrovski> help C:\Scripts\Get-LastBootTime.ps1
Get-LastBootTime.ps1 [[-ComputerName] <string>]


PS C:\Users\Nikola Petrovski>
```

To specify the name of the computer and make this input mandatory:

Get-LastBootTime.ps1

```
1  <#
2   .SYNOPSIS
3   Shows when last your PC started up.
4   .DESCRIPTION
5   This is a WMI wrapper function to get the time that your PC last started up.
6   .PARAMETER ComputerName
7   The name of the Computer you want to run the command against.
8   .EXAMPLE
9   Get-LastBootTime -ComputerName localhost
10  .LINK
11  www.howtogeek.com
12  #>
13
14 param(
15      [Parameter(Mandatory=$true)][string]$ComputerName
16 )
17
18  Get-WmiObject -Class Win32_OperatingSystem -ComputerName $ComputerName |
19 Select-Object -Property CSName,@{n="Last Booted";
20  e={[Management.ManagementDateTimeConverter]::ToDateTime($_.LastBootUpTime)}}
```

```
PS C:\Users\Nikola Petrovski> help C:\Scripts\Get-LastBootTime.ps1

NAME
    C:\Scripts\Get-LastBootTime.ps1

SYNOPSIS
    Shows when last your PC started up.


SYNTAX
    C:\Scripts\Get-LastBootTime.ps1 [-ComputerName] <String> [<CommonParameters>]


DESCRIPTION
    This is a WMI wrapper function to get the time that your PC last started up.
```

Complete script:

Get-LastBootTime.ps1

```
<#
.SYNOPSIS
Shows when last your PC started up.
.DESCRIPTION
This is a WMI wrapper function to get the time that your PC last started up.
.PARAMETER ComputerName
The name of the Computer you want to run the command against.
.EXAMPLE
Get-LastBootTime -ComputerName localhost
.LINK
www.howtogeek.com
#>

param(
      [Parameter(Mandatory=$true)][string]$ComputerName
)

 Get-WmiObject -Class Win32_OperatingSystem -ComputerName $ComputerName |
Select-Object -Property CSName,@{n="Last Booted";
e={[Management.ManagementDateTimeConverter]::ToDateTime($_.LastBootUpTime)}}
```

```
PS C:\Users\Nikola Petrovski> C:\Scripts\Get-LastBootTime.ps1
cmdlet Get-LastBootTime.ps1 at command pipeline position 1
Supply values for the following parameters:
ComputerName: |
```

Running script / selection. Press Ctrl+Break to stop. Press Ctrl+B to break into debugger.

After input:

```
PS C:\Users\Nikola Petrovski> C:\Scripts\Get-LastBootTime.ps1
cmdlet Get-LastBootTime.ps1 at command pipeline position 1
Supply values for the following parameters:
ComputerName: NM

CSName Last Booted
------ -----------
NM     26/5/18 23:09:23


PS C:\Users\Nikola Petrovski>
```

Variables, Inputs and Outputs are critical for any script. If a variable named "FirstName" is created by typing: $FirstName = "Bob" then it can be deleted as shown below:

```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> Remove-Item Variable:\FirstName
PS C:\Users\Nikola Petrovski> Variable

Name                           Value
----                           -----
$                              Variable:\FirstName
?                              True
^                              Remove-Item
args                           {}
ConfirmPreference              High
ConsoleFileName
DebugPreference                SilentlyContinue
Error                          {Cannot find path 'C:\Users\Nikola Petrovski\variable' because it does not exist., The term 'C:...
ErrorActionPreference          Continue
ErrorView                      NormalView
ExecutionContext               System.Management.Automation.EngineIntrinsics
false                          False
FormatEnumerationLimit         4
HOME                           C:\Users\Nikola Petrovski
Host                           System.Management.Automation.Internal.Host.InternalHost
InformationPreference          SilentlyContinue
input                          System.Collections.ArrayList+ArrayListEnumeratorSimple
```

Variable can store more than one thing e.g., 3 processes objects :

```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> $CPUHogs = Get-Process | Sort CPU -Descending | select -First 3
PS C:\Users\Nikola Petrovski> $CPUHogs

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
    833      12     6384      10532   3,017.92    932   0 services
   2411     111   195032     213788   2,706.47   6136   1 opera
   1195      77  1696276    1178556   1,970.91  14244   1 opera


PS C:\Users\Nikola Petrovski>
```

NOTE: In PS the variables that hold a collection of objects can be identified, specified and divided into single objects by using the array like syntax. To get the first object in the collection type: $CPUHogs[0]. Also, the order of operations determines if the result from "+" symbol will be addition or concatenation result. In PS casting is used to prevent any exceptions "[int]$Number = '5'. For example:

$2 = 10

$b = '20'

There are two combinations:

$a + $b = 30

$b + $a = 2010

Input/Output :

```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> $FirstName = Read-Host -Prompt 'Enter your first name'
Enter your first name: Nikola
PS C:\Users\Nikola Petrovski> $FirstName
Nikola
PS C:\Users\Nikola Petrovski>
```

Write-Output:

```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> Write-Output "pearls.net Rocks!"
pearls.net Rocks!
PS C:\Users\Nikola Petrovski>
```

Using quotation marks and "NoNewLine" example:

```
Administrator: Windows PowerShell
PS C:\Users\Nikola Petrovski> Write-Host -NoNewLine "Counting from 1 to 9 (in seconds): "
Counting from 1 to 9 (in seconds):
PS C:\Users\Nikola Petrovski>
PS C:\Users\Nikola Petrovski> foreach($element in 1..9){
>>
>> Write-Host -NoNewLine "${element} "
>>
>> Start-Sleep -Seconds 1
>>
>> }
1 2 3 4 5 6 7 8 9
PS C:\Users\Nikola Petrovski>
PS C:\Users\Nikola Petrovski> Write-Host ""

PS C:\Users\Nikola Petrovski>
```

Using style and color for the output example:

File Edit View Tools Debug Add-ons Help

Get-LastBootTime.ps1 | trafic lights.ps1

```
Write-Host -NoNewLine "Counting from 1 to 9 (in seconds): "

foreach($element in 1..3){

Write-Host -NoNewLine "${element} " -BackgroundColor "Green" -ForegroundColor "Black"

Start-Sleep -Seconds 1

}

foreach($element in 4..6){

Write-Host -NoNewLine "${element} " -BackgroundColor "Yellow" -ForegroundColor "Black"

Start-Sleep -Seconds 1

}

foreach($element in 7..9){

Write-Host -NoNewLine "${element} " -BackgroundColor "Red" -ForegroundColor "Black"

Start-Sleep -Seconds 1

}

Write-Host ""
```

```
PS C:\Users\Nikola Petrovski> c:\scripts\trafic lights.ps1
Counting from 1 to 9 (in seconds): 1 2 3 4 5 6 7 8 9

PS C:\Users\Nikola Petrovski>
```

Example showing the contract between two cmdlets where redirection is used:

Get-LastBootTime.ps1 | trafic lights.ps1 | write host.ps1*

```
Write-Host "Phase 1"
$start=Get-Date
Write-Output $start
Write-Host "Phase 2"
Start-Sleep -Seconds 5
Write-Host "Phase 3"
$end=Get-Date
Write-Output $end
Write-Host "Phase 4"
$diff=New-TimeSpan $start $end
Write-Output $diff
```

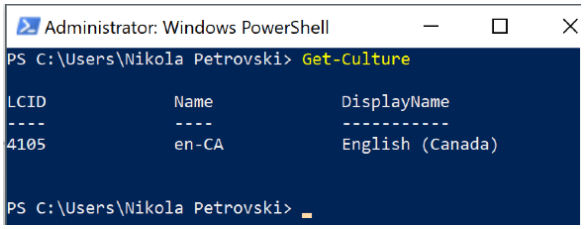
```
PS C:\Users\Nikola Petrovski> C:\Scripts\write host.ps1
Phase 1

27-May-18 3:37:06
Phase 2
Phase 3
27-May-18 3:37:11
Phase 4

Ticks             : 50364858
Days              : 0
Hours             : 0
Milliseconds      : 36
Minutes           : 0
Seconds           : 5
TotalDays         : 5.82926597222222E-05
TotalHours        : 0.00139902383333333
TotalMilliseconds : 5036.4858
TotalMinutes      : 0.08394143
TotalSeconds      : 5.0364858
```

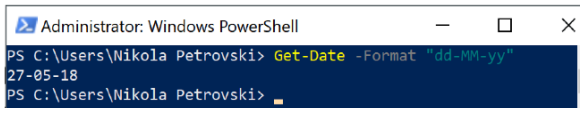
NOTE: Write-Host output is temporary. It is not captured on the screen but obsolete by the next output.

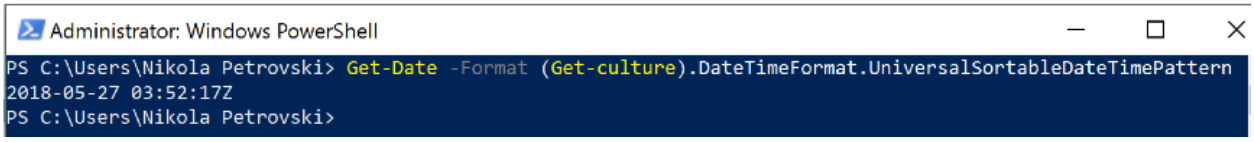
3.0 Summary

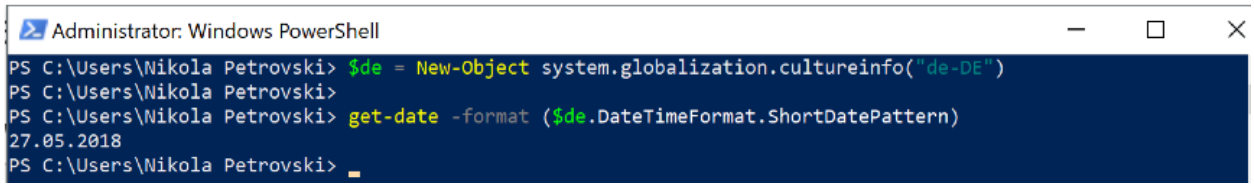
There are various scripts for folder permissions or NTFS permissions (PowerShell), for exporting Active Directory users to CSV as well as setting date formats.

Examples:









Get-LastBootTime.ps1 | trafic lights.ps1 | write host.ps1* | date formats1.ps1

```
$cultures = "en-US","en-GB","fr-CA","fr-FR","ms-MY","zh-HK"

foreach ($c in $cultures)

{

 $culture = New-Object system.globalization.cultureinfo($c)

 $date = get-date -format ($culture.DateTimeFormat.ShortDatePattern)

 New-Object psobject -Property @{"name"=$culture.displayname; "date"=$date}

}
```

```
PS C:\Users\Nikola Petrovski> C:\Scripts\date formats1.ps1

name                                    date
----                                    ----
English (United States)                 5/27/2018
English (United Kingdom)                27/05/2018
French (Canada)                         2018-05-27
French (France)                         27/05/2018
Malay (Malaysia)                        27/05/2018
Chinese (Traditional, Hong Kong S.A.R.) 27/5/2018


PS C:\Users\Nikola Petrovski>
```

Completed | Ln 6 Col 1

Get-LastBootTime.ps1 | trafic lights.ps1 | write host.ps1* | date formats1.ps1 | date formats2.ps1

```
$cultures = [globalization.cultureinfo]::GetCultures("allCultures") |

  where-object {$_.name -match '^fr'}

foreach ($c in $cultures)

{

 $culture = New-Object system.globalization.cultureinfo($c)

 $date = get-date -format ($culture.DateTimeFormat.ShortDatePattern)

 New-Object psobject -Property @{"name"=$culture.displayname; "date"=$date}

}
```

```
PS C:\Users\Nikola Petrovski> C:\Scripts\date formats2.ps1

name                              date
----                              ----
French                            27/05/2018
French (Caribbean)                27/05/2018
French (Belgium)                  27-05-18
French (Burkina Faso)             27/05/2018
French (Burundi)                  27/05/2018
French (Benin)                    27/05/2018
French (Saint Barthélemy)         27/05/2018
French (Canada)                   2018-05-27
French (Congo DRC)                27/05/2018
French (Central African Republic) 27/05/2018
French (Congo)                    27/05/2018
French (Switzerland)              27.05.2018
French (Côte d'Ivoire)            27/05/2018
French (Cameroon)                 27/05/2018
French (Djibouti)                 27/05/2018
French (Algeria)                  27/05/2018
French (France)                   27/05/2018
French (Gabon)                    27/05/2018
```

Completed | Ln 15 Col 2

4.0 References

https://docs.microsoft.com/en-us/powershell/

https://docs.microsoft.com/en-us/powershell/scripting/core-powershell/ise/introducing-the-windows-powershell-ise?view=powershell-6