TryHackMe Hacking with PowerShell

Saikat Karmakar | Aug 6 : 2021

Powershell

Powershell is the Windows Scripting Language and shell environment that is built using the .NET framework.

This also allows Powershell to execute .NET functions directly from its shell. Most Powershell commands, called *cmdlets,* are written in .NET. Unlike other scripting languages and shell environments, the output of these *cmdlets* are objects - making Powershell somewhat object oriented. This also means that running cmdlets allows you to perform actions on the output object(which makes it convenient to pass output from one *cmdlet* to another). The normal format of a *cmdlet* is represented using **Verb-Noun**; for example the *cmdlet* to list commands is called `Get-Command.`

Common verbs to use include:

- Get
- Start
- Stop
- Read
- Write
- New
- Out

To get the full list of approved verbs, visit this link.

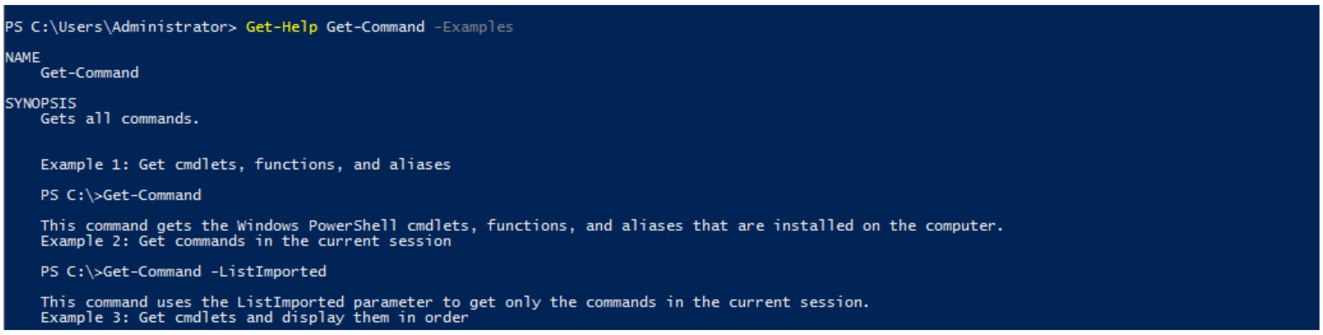
Using Get-Help

Now that we've understood how *cmdlets* works - let's explore how to use them! The main thing to remember here is that Get-Command and Get-Help are your best friends!

Get-Help displays information about a *cmdlet.* To get help about a particular command, run the following:

`Get-Help Command-Name`

You can also understand how exactly to use the command by passing in the `-examples` flag. This would return output like the following:



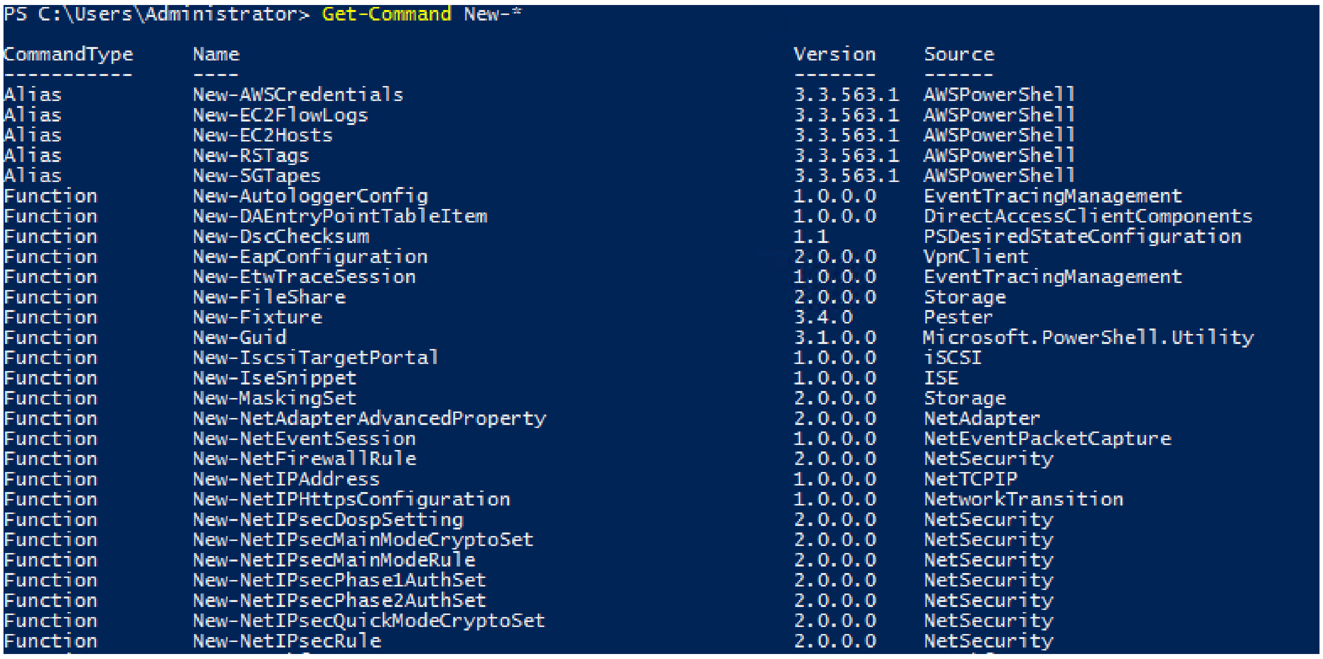
```
PS C:\Users\Administrator> Get-Help Get-Command -Examples

NAME
    Get-Command

SYNOPSIS
    Gets all commands.


    Example 1: Get cmdlets, functions, and aliases

    PS C:\>Get-Command

    This command gets the Windows PowerShell cmdlets, functions, and aliases that are installed on the computer.
    Example 2: Get commands in the current session

    PS C:\>Get-Command -ListImported

    This command uses the ListImported parameter to get only the commands in the current session.
    Example 3: Get cmdlets and display them in order
```

Using Get-Command

Get-Command gets all the *cmdlets* installed on the current Computer. The great thing about this *cmdlet* is that it allows for pattern matching like the following

`Get-Command Verb-*` or `Get-Command *-Noun`

Running `Get-Command New-*` to view all the *cmdlets* for the verb new displays the following:



```
PS C:\Users\Administrator> Get-Command New-*

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           New-AWSCredentials                                 3.3.563.1  AWSPowerShell
Alias           New-EC2FlowLogs                                    3.3.563.1  AWSPowerShell
Alias           New-EC2Hosts                                       3.3.563.1  AWSPowerShell
Alias           New-RSTags                                         3.3.563.1  AWSPowerShell
Alias           New-SGTapes                                        3.3.563.1  AWSPowerShell
Function        New-AutologgerConfig                               1.0.0.0    EventTracingManagement
Function        New-DAEntryPointTableItem                          1.0.0.0    DirectAccessClientComponents
Function        New-DscChecksum                                    1.1        PSDesiredStateConfiguration
Function        New-EapConfiguration                               2.0.0.0    VpnClient
Function        New-EtwTraceSession                                1.0.0.0    EventTracingManagement
Function        New-FileShare                                      2.0.0.0    Storage
Function        New-Fixture                                        3.4.0      Pester
Function        New-Guid                                           3.1.0.0    Microsoft.PowerShell.Utility
Function        New-IscsiTargetPortal                              1.0.0.0    iSCSI
Function        New-IseSnippet                                     1.0.0.0    ISE
Function        New-MaskingSet                                     2.0.0.0    Storage
Function        New-NetAdapterAdvancedProperty                     2.0.0.0    NetAdapter
Function        New-NetEventSession                                1.0.0.0    NetEventPacketCapture
Function        New-NetFirewallRule                                2.0.0.0    NetSecurity
Function        New-NetIPAddress                                   1.0.0.0    NetTCPIP
Function        New-NetIPHttpsConfiguration                        1.0.0.0    NetworkTransition
Function        New-NetIPsecDospSetting                            2.0.0.0    NetSecurity
Function        New-NetIPsecMainModeCryptoSet                      2.0.0.0    NetSecurity
Function        New-NetIPsecMainModeRule                           2.0.0.0    NetSecurity
Function        New-NetIPsecPhase1AuthSet                          2.0.0.0    NetSecurity
Function        New-NetIPsecPhase2AuthSet                          2.0.0.0    NetSecurity
Function        New-NetIPsecQuickModeCryptoSet                     2.0.0.0    NetSecurity
Function        New-NetIPsecRule                                   2.0.0.0    NetSecurity
```

Object Manipulation

In the previous task, we saw how the output of every *cmdlet* is an object. If we want to actually manipulate the output, we need to figure out a few things:

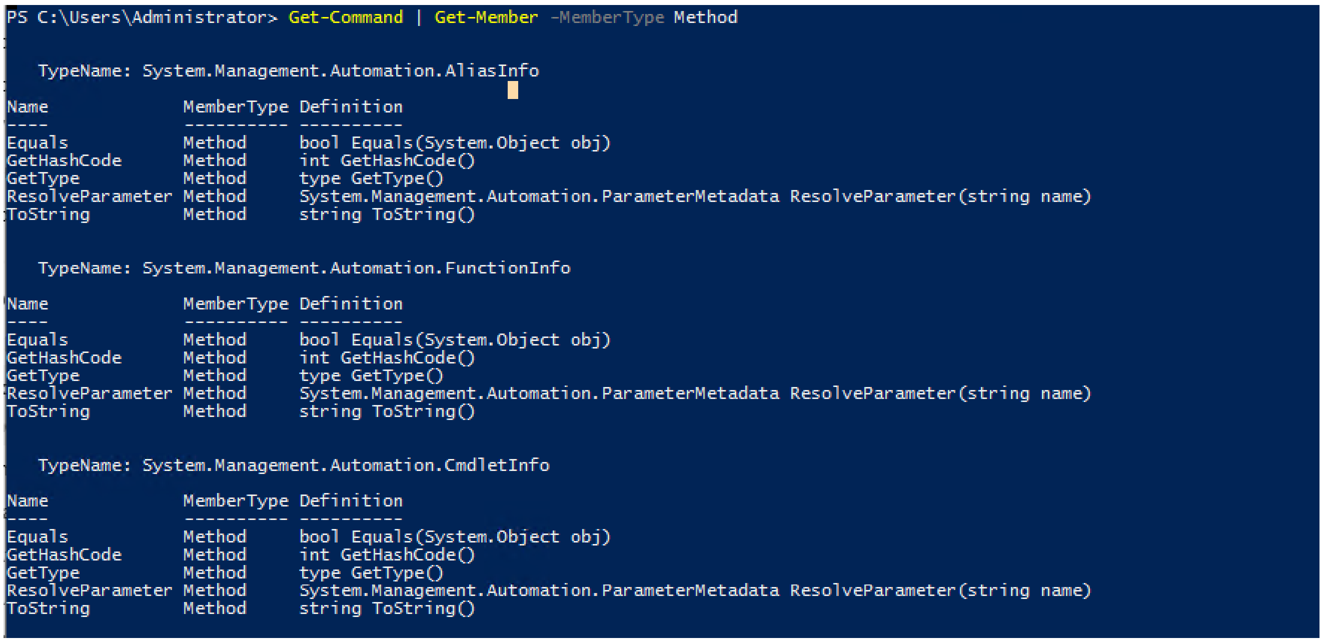
- passing output to other *cmdlets*
- using specific object *cmdlets* to extract information

The Pipeline(|) is used to pass output from one *cmdlet* to another. A major difference compared to other shells is that instead of passing text or string to the command after the pipe, powershell passes an object to the next cmdlet. Like every object in object oriented frameworks, an object will contain methods and properties. You can think of methods as functions that can be applied to output from the *cmdlet* and you can think of properties as variables in the output from a cmdlet. To view these details, pass the output of a *cmdlet* to the Get-Member *cmdlet*

`Verb-Noun | Get-Member`

An example of running this to view the members for Get-Command is:

`Get-Command | Get-Member -MemberType Method`



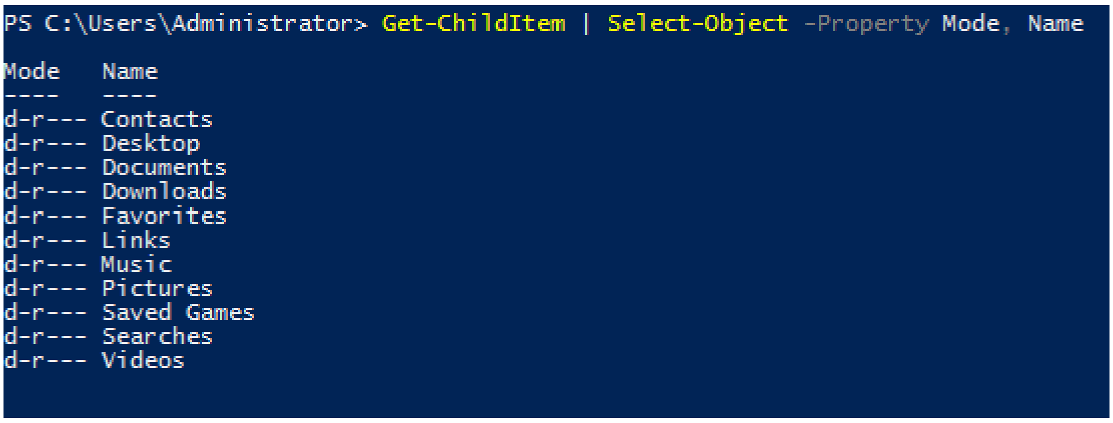
```
PS C:\Users\Administrator> Get-Command | Get-Member -MemberType Method


   TypeName: System.Management.Automation.AliasInfo

Name                MemberType Definition
----                ---------- ----------
Equals              Method     bool Equals(System.Object obj)
GetHashCode         Method     int GetHashCode()
GetType             Method     type GetType()
ResolveParameter    Method     System.Management.Automation.ParameterMetadata ResolveParameter(string name)
ToString            Method     string ToString()


   TypeName: System.Management.Automation.FunctionInfo

Name                MemberType Definition
----                ---------- ----------
Equals              Method     bool Equals(System.Object obj)
GetHashCode         Method     int GetHashCode()
GetType             Method     type GetType()
ResolveParameter    Method     System.Management.Automation.ParameterMetadata ResolveParameter(string name)
ToString            Method     string ToString()


   TypeName: System.Management.Automation.CmdletInfo

Name                MemberType Definition
----                ---------- ----------
Equals              Method     bool Equals(System.Object obj)
GetHashCode         Method     int GetHashCode()
GetType             Method     type GetType()
ResolveParameter    Method     System.Management.Automation.ParameterMetadata ResolveParameter(string name)
ToString            Method     string ToString()
```

From the above flag in the command, you can see that you can also select between methods and properties.

Creating Objects From Previous *cmdlets*

One way of manipulating objects is pulling out the properties from the output of a cmdlet and creating a new object. This is done using the `Select-Object` *cmdlet.*

Here's an example of listing the directories and just selecting the mode and the name:



```
PS C:\Users\Administrator> Get-ChildItem | Select-Object -Property Mode, Name

Mode   Name
----   ----
d-r--- Contacts
d-r--- Desktop
d-r--- Documents
d-r--- Downloads
d-r--- Favorites
d-r--- Links
d-r--- Music
d-r--- Pictures
d-r--- Saved Games
d-r--- Searches
d-r--- Videos
```

You can also use the following flags to select particular information:

- first - gets the first x object
- last - gets the last x object
- unique - shows the unique objects
- skip - skips x objects

Filtering Objects

When retrieving output objects, you may want to select objects that match a very specific value. You can do this using the `Where-Object` to filter based on the value of properties.

The general format of the using this *cmdlet* is

`Verb-Noun | Where-Object -Property PropertyName -operator Value`

`Verb-Noun | Where-Object {$_.PropertyName -operator Value}`

The second version uses the $_ operator to iterate through every object passed to the Where-Object cmdlet.

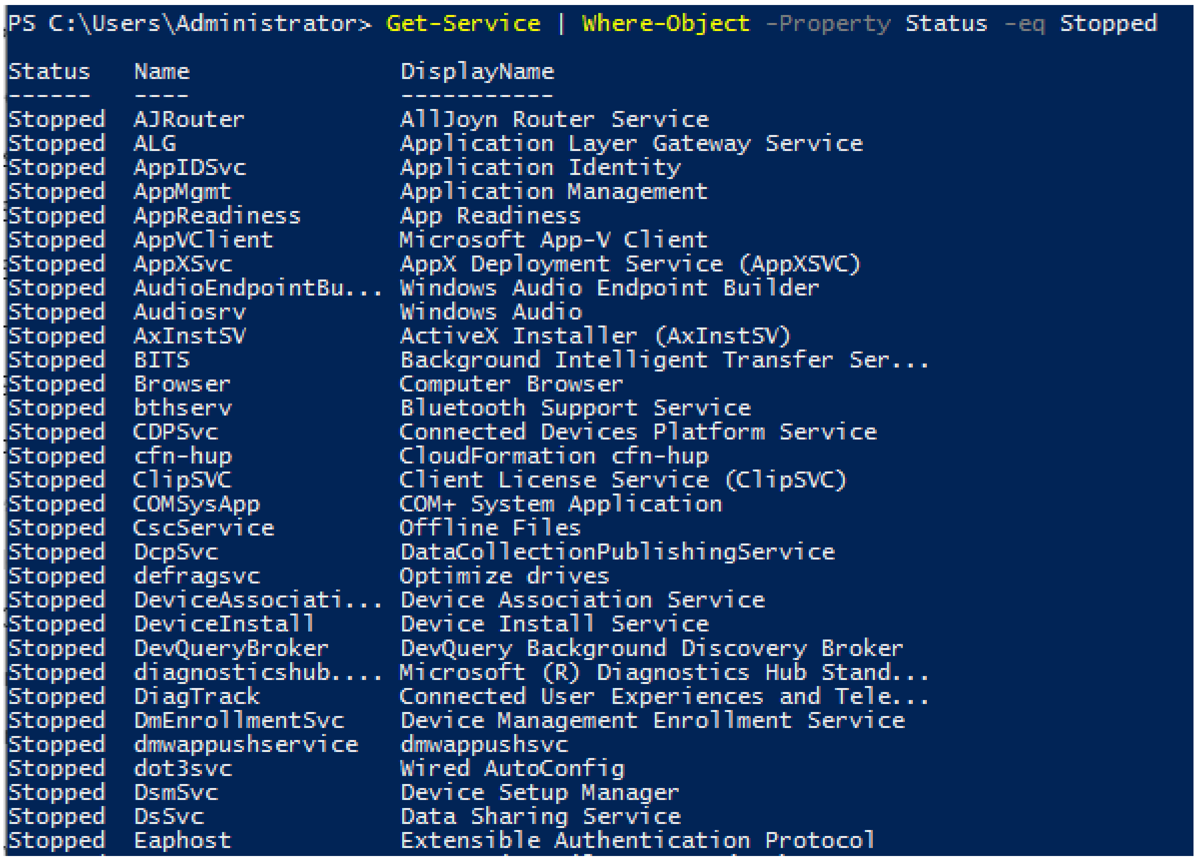
Powershell is quite sensitive so make sure you don't put quotes around the command!

Where `-operator` is a list of the following operators:

- -Contains: if any item in the property value is an exact match for the specified value
- -EQ: if the property value is the same as the specified value
- -GT: if the property value is greater than the specified value

For a full list of operators, use this link.

Here's an example of checking the stopped processes:



```
PS C:\Users\Administrator> Get-Service | Where-Object -Property Status -eq Stopped

Status   Name               DisplayName
------   ----               -----------
Stopped  AJRouter           AllJoyn Router Service
Stopped  ALG                Application Layer Gateway Service
Stopped  AppIDSvc           Application Identity
Stopped  AppMgmt            Application Management
Stopped  AppReadiness       App Readiness
Stopped  AppVClient         Microsoft App-V Client
Stopped  AppXSvc            AppX Deployment Service (AppXSVC)
Stopped  AudioEndpointBu... Windows Audio Endpoint Builder
Stopped  Audiosrv           Windows Audio
Stopped  AxInstSV           ActiveX Installer (AxInstSV)
Stopped  BITS               Background Intelligent Transfer Ser...
Stopped  Browser            Computer Browser
Stopped  bthserv            Bluetooth Support Service
Stopped  CDPSvc             Connected Devices Platform Service
Stopped  cfn-hup            CloudFormation cfn-hup
Stopped  ClipSVC            Client License Service (ClipSVC)
Stopped  COMSysApp          COM+ System Application
Stopped  CscService         Offline Files
Stopped  DcpSvc             DataCollectionPublishingService
Stopped  defragsvc          Optimize drives
Stopped  DeviceAssociati... Device Association Service
Stopped  DeviceInstall      Device Install Service
Stopped  DevQueryBroker     DevQuery Background Discovery Broker
Stopped  diagnosticshub.... Microsoft (R) Diagnostics Hub Stand...
Stopped  DiagTrack          Connected User Experiences and Tele...
Stopped  DmEnrollmentSvc    Device Management Enrollment Service
Stopped  dmwappushservice   dmwappushsvc
Stopped  dot3svc            Wired AutoConfig
Stopped  DsmSvc             Device Setup Manager
Stopped  DsSvc              Data Sharing Service
Stopped  Eaphost            Extensible Authentication Protocol
```

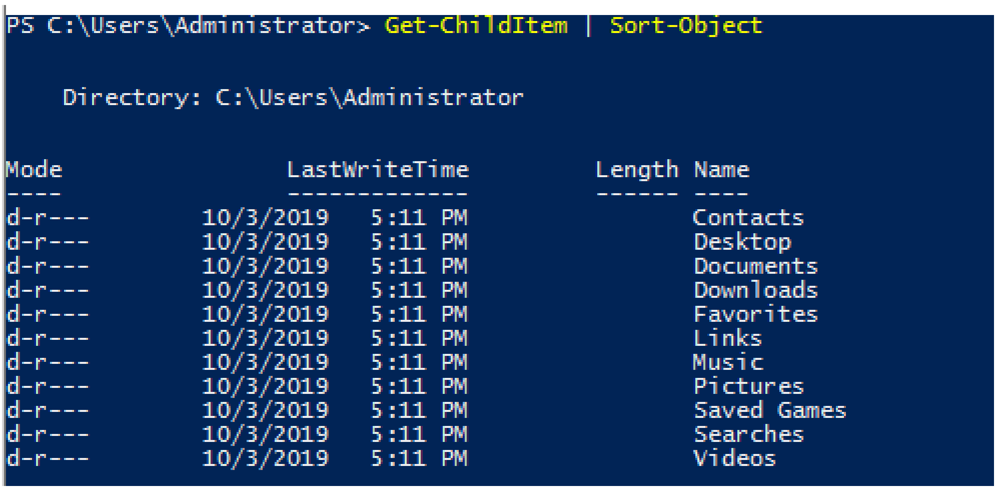
Sort Object

When a *cmdlet* outputs a lot of information, you may need to sort it to extract the information more efficiently. You do this by pipe lining the output of a *cmdlet* to the `Sort-Object` *cmdlet*.

The format of the command would be

`Verb-Noun | Sort-Object`

Here's an example of sort the list of directories:



```
PS C:\Users\Administrator> Get-ChildItem | Sort-Object


    Directory: C:\Users\Administrator


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-r---        10/3/2019   5:11 PM                Contacts
d-r---        10/3/2019   5:11 PM                Desktop
d-r---        10/3/2019   5:11 PM                Documents
d-r---        10/3/2019   5:11 PM                Downloads
d-r---        10/3/2019   5:11 PM                Favorites
d-r---        10/3/2019   5:11 PM                Links
d-r---        10/3/2019   5:11 PM                Music
d-r---        10/3/2019   5:11 PM                Pictures
d-r---        10/3/2019   5:11 PM                Saved Games
d-r---        10/3/2019   5:11 PM                Searches
d-r---        10/3/2019   5:11 PM                Videos
```

Enumeration

The first step when you have gained initial access to any machine would be to enumerate. We'll be enumerating the following:

- users
- basic networking information
- file permissions
- registry permissions

- scheduled and running tasks
- insecure files

Scripting

Now that we have run powershell commands, let's actually try write and run a script to do more complex and powerful actions.

For this ask, we'll be using PowerShell ISE(which is the Powershell Text Editor). To show an example of this script, let's use a particular scenario. Given a list of port numbers, we want to use this list to see if the local port is listening. Open the listening-ports.ps1 script on the Desktop using Powershell ISE. Powershell scripts usually have the *.ps1* file extension.

```powershell
$system_ports = Get-NetTCPConnection -State Listen
$text_port = Get-Content -Path C:\Users\Administrator\Desktop\ports.txt
foreach($port in $text_port){
    if($port -in $system_ports.LocalPort){
        echo $port
     }
}
```
language-powershell

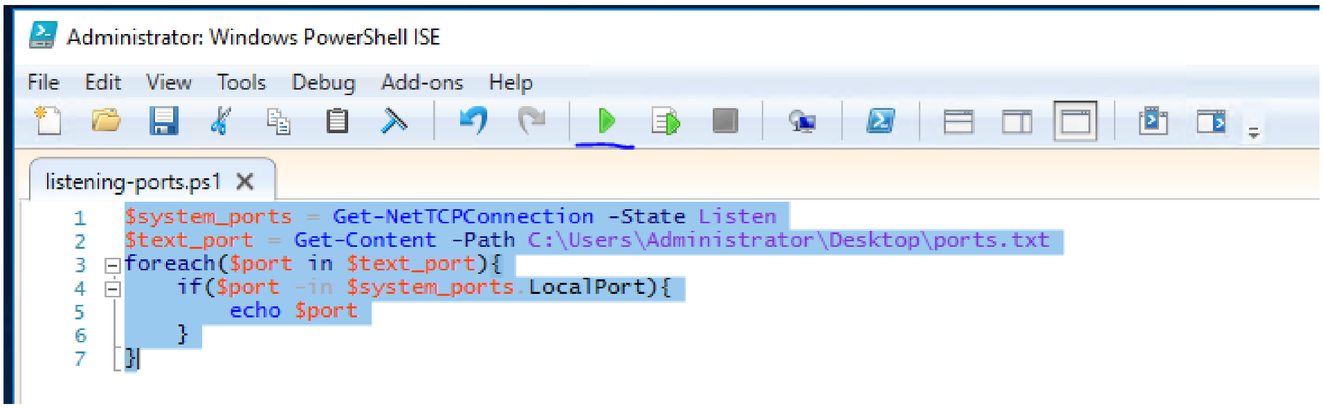
On the first line, we want to get a list of all the ports on the system that are listening. We do this using the Get-NetTCPConnection *cmdlet*. We are then saving the output of this *cmdlet* into a variable. The convention to create variables is used as:

```powershell
$variable_name = value
```
language-powershell

On the next line, we want to read a list of ports from the file. We do this using the Get-Content *cmdlet.* Again, we store this output in the variables. The simplest next step is iterate through all the ports in the file to see if the ports are listening. To iterate through the ports in the file, we use the following

```powershell
foreach($new_var in $existing_var){}
```
language-powershell

This particular code block is used to loop through a set of object. Once we have each individual port, we want to check if this port occurs in the listening local ports. Instead of doing another for loop, we just use an if statement with the `-in` operator to check if the port exists the LocalPort property of any object. A full list of if statement comparison operators can be found here. To run script, just call the script path using Powershell or click the green button on Powershell ISE:



```powershell
$system_ports = Get-NetTCPConnection -State Listen
$text_port = Get-Content -Path C:\Users\Administrator\Desktop\ports.txt
foreach($port in $text_port){
    if($port -in $system_ports.LocalPort){
        echo $port
    }
}
```