



Module 3 - System Administration PowerShell

Session 5 - Practical Uses

Presented by Tim Medin

© SANS, Cyber Aces, Red Siege. All Rights Reserved. Redistribution Prohibited.

YOUR GATEWAY TO CYBERSECURITY SKILLS AND CAREERS

Welcome to Cyber Aces, Module 3! This module provides an introduction to the latest shell for Windows, PowerShell.

SANS CYBER ACES ONLINE TUTORIALS

YOUR GATEWAY TO CYBERSECURITY SKILLS AND CAREERS

1. Introduction to Operating Systems

- 01. Linux
- 02. Windows

2. Networking

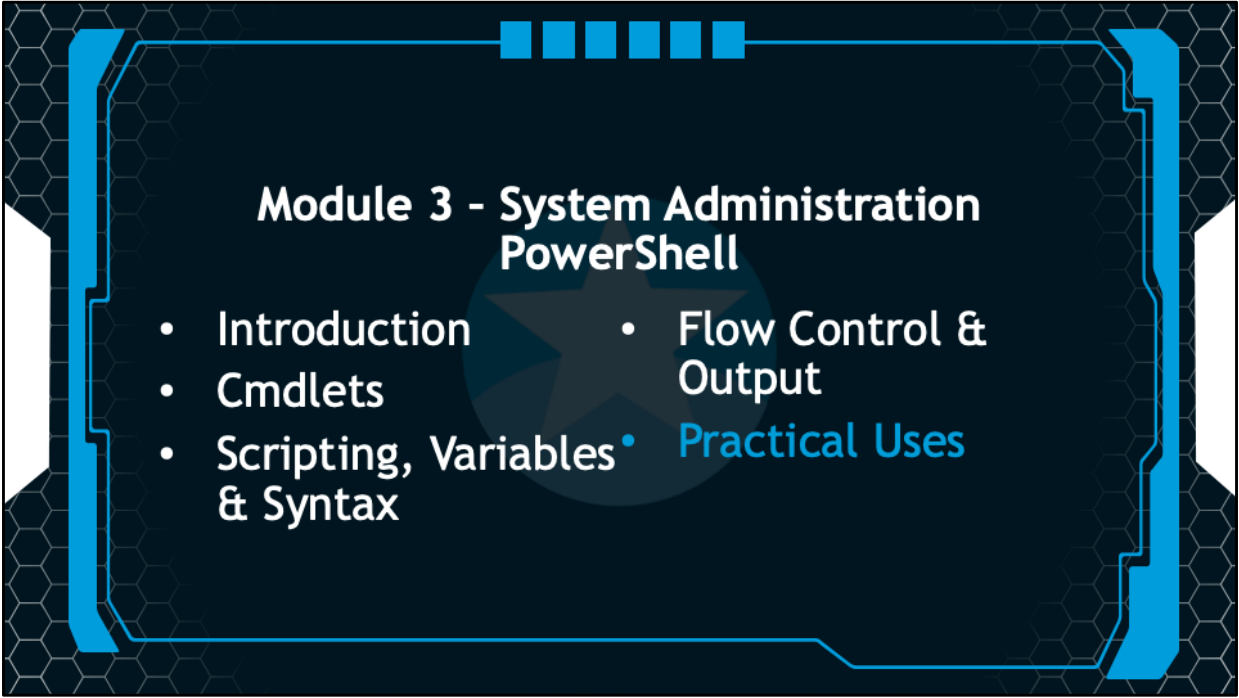
3. System Administration

- 01. Bash
- 02. PowerShell
- 03. Python

This training material was originally developed to help students, teachers, and mentors prepare for the Cyber Aces Online Competition. This module focuses on the basics of what an operating systems is as well as the two predominant OS's, Windows and Linux. In this session we will provide a walkthrough of the installation a Windows VM using VMware Fusion (MacOS) and VMware Player (Windows & Linux). These sessions include hands-on labs, but before we begin those labs we need to install the operating systems used in those labs. We will be using VMware to virtualize these operating systems. You can use other virtualization technologies if you like, but instruction for their setup and use are not included in this training.

The three modules of Cyber Aces Online are Operating Systems, Networking, and System Administration.

For more information about the Cyber Aces program, please visit the Cyber Aces website at <https://CyberAces.org/>.

The slide has a dark blue background with a hexagonal pattern. A large, stylized blue star is centered behind the text. The title 'Module 3 - System Administration PowerShell' is written in white. Below the title, there are four bullet points: 'Introduction', 'Cmdlets', 'Scripting, Variables & Syntax', and 'Flow Control & Output'. The text 'Practical Uses' is written in blue and is positioned to the right of the third bullet point. The slide is framed by a blue border with a hexagonal pattern on the outside.

Module 3 - System Administration PowerShell

- Introduction
- Cmdlets
- Scripting, Variables & Syntax
- Flow Control & Output
- Practical Uses

In this section, we use the knowledge we gained in some practical scenarios.



Ping All IP's in a Range



Task: Ping all hosts on a network with a /24 mask

```
1..254 | % { ping "192.168.0.$_" }
```

Range operator counts from 1 to 254; we skip the network address (0) and the broadcast address (255)

The numbers from 1 through 254 are piped into the ForEach-Object cmdlet

The double quotes contain a string, but will expand variables, so we get 192.168.0.1, then 192.168.0.2, etc.

The ping command is executed on each address

Say we have a network, where we would like to lookup the name of each device on the network. We can use the "Range" operator in conjunction with our "ForEach-Object" loop to pull this off.

```
PS C:\> 1..254 | % { Write-Output "192.168.0.$_" }
```

```
192.168.0.1
```

```
192.168.0.2
```

```
192.168.0.3
```

```
...
```

The "Range" operator is just a quick way of counting. The results are piped into our "ForEach-Object" loop where we display, via "Write-Output", the string. Instead of just printing the IP address, we could ping every IP address.

```
PS C:\> 1..254 | % { ping "192.168.0.$_" }
```

We could just as easily replace "ping" with "nslookup" or numerous other network commands. The possibilities are endless.

Note: This command will also work without using quotes, but will not work with single quotes.



Process Manipulation



The *-Process series of cmdlets

Start Notepad: **Start-Process notepad**

- Easier to just type notepad, but we have more power

Open a file with Notepad and maximize the window

```
Start-Process notepad -ArgumentList aa.txt -WindowStyle Maximized
```

Print

```
Start-Process notepad -ArgumentList aa.txt -Verb Print
```

Kill a process by ID

```
Stop-Process 1337
```

Kill a process by name

```
Stop-Process notepad
```

Kill all executables running from E. Phil's desktop

```
ps | ? { $_.Path -like "C:\Users\ephil\*" } | kill
```

This is the proverbial, "I brought you into this world, and I'll take you out." First, we need to bring a process into this world.

Start-Process notepad

That was easy, but we could have just typed "notepad" and accomplished the same thing. But we can do something cooler; we can use Notepad to open a file and maximize the window.

```
Start-Process notepad -ArgumentList myfile.txt  
-WindowStyle Maximized
```

Using the alias, positional parameters, shortened parameter names, and shortened options we can squish the command to this:

```
start notepad myfile.txt -win max
```

What if we wanted to print the file? We can do that too, and we can use the viewer associated with the file. It is as if we right clicked on the file and selected "Print."

```
Start-Process myfile.pdf -Verb Print
```

Ok, so starting processes isn't so neat, but killing them is. We can use "Stop-Process" (alias "kill") to stop processes. We can kill based on the Process Id...

```
Stop-Process 1337
```

...or the process name:

```
Stop-Process -Name cmd
```

What if we have a user on the system named "E. Phil", and E. Phil is evil. What if he is running executables from his desktop and we want to kill them?

```
ps | ? { $_.Path -like "C:\Users\ephil\*" } | kill
```

This command gets all the processes, filters for executables originating from E. Phil's user path, and then kills them. We have successfully defeated E Phil, and the world is

now a safer place for shells.



Review Exercises



- 1) The Blah Company is using 256 networks, 10.0.0.X/24 through 10.0.255.X/24. On each network, they have a network gateway and its IP address ends in 254 (i.e. 10.0.0.254, 10.0.1.254...). Write a command to ping each gateway.
- 2) Which command will NOT kill all processes with "bad" in the process name?
 - a. `ps -name *bad* | kill`
 - b. `ps | ? { $_.Name -like "*bad*" } | kill`
 - c. `Get-Process | Where-Object { $_.Name -contains "*bad*" } | Stop-Process`
 - d. `kill -name *bad*`
 - e. `Get-Process -Name "*bad*" | Stop-Process`

- 1) The Blah Company is using 256 networks with a /24 CIDR mask, 10.0.0.X/24 through 10.0.255.X/24. On each network, they have a network gateway and its IP address ends in .254 (i.e. 10.0.0.254, 10.0.1.254...). Write a command to ping each gateway.
- 2) Which command will NOT kill all processes with "bad" in the process name?
 - a. `ps -name *bad* | kill`
 - b. `ps | ? { $_.Name -like "*bad*" } | kill`
 - c. `Get-Process | Where-Object { $_.Name -contains "*bad*" } | Stop-Process`
 - d. `kill -name *bad*`
 - e. `Get-Process -Name "*bad*" | Stop-Process`



Answers



- 1) The Blah Company is using 256 networks, 10.0.0.X/24 through 10.0.255.X/24. On each network, they have a network gateway and its IP address ends in .254 (i.e. 10.0.0.254, 10.0.1.254...). Write a command to ping each gateway.

One possible answer:

```
0..255 | % { ping 10.0.$_.254 }
```

- This is very similar to the earlier example, the only difference is the octet

- 2) Which command will NOT kill all processes with "bad" in the process name?

- The answer is C

```
Get-Process | Where-Object { $_.Name -contains "*bad*" } | Stop-Process
```

- The `-contains` operator won't work here as it is used to search an array/collection for a matching item

- 1) The Blah Company is using 256 networks, 10.0.0.X/24 through 10.0.255.X/24. On each network, they have a network gateway and its IP address ends in .254 (i.e. 10.0.0.254, 10.0.1.254...). Write a command to ping each gateway.

One possible answer:

```
0..255 | % { ping 10.0.$_.254 }
```

This is very similar to the earlier example, the only difference is the octet

- 2) Which command will NOT kill all processes with "bad" in the process name?

c. `Get-Process | Where-Object { $_.Name -contains "*bad*" } | Stop-Process`

The `-contains` operator won't work here as it is used to search an array/collection for a matching item, not for finding a string in another string. The `-match` and `-like` operators are used to search strings using regular expressions and wildcards respectively.



Iterate through Files in a Folder



Get-ChildItem works on any hierarchy and the file system is one such hierarchy

The difference between a directory and a file is that a directory is a container (PSIsContainer is True for a folder)

To find all files we would filter out any containers (directories)

```
Get-ChildItem | Where-Object {-not  
$_.PSIsContainer}
```

Shortened (note: ! is short for -not)

```
ls | ? { !$_.PSIsContainer }
```

View hidden/system files with the -Force option (shorten to -fo)

```
ls -fo | ? { !$_.PSIsContainer }
```

Before we iterate through all the files in a directory, we need to figure out how to filter out directories. Let's start by looking at a regular directory listing:

```
PS C:\> ls  
Directory: C:\  
Mode                LastWriteTime         Length      Name  
d-r--             1/3/2011 7:14 AM  
d-r--             12/8/2010 8:56 AM      Users  
d----             1/3/2011 9:58 AM      Windows  
-a---             6/10/2009 4:42 PM        24      autoexec.bat  
-a---             6/10/2009 4:42 PM        10      config.sys
```

It looks like all the files have the "a" bit set, but that isn't a guarantee to find only files. The "a" stands for "Archive", meaning the file has been modified since the last backup. We need another option. We could filter out anything with the "d" bit set. That works, but it is still a little cheesy. The best option is to look at the properties of the objects to see if there is a better option.

```
PS C:\> ls | gm
```

If you run the command (output not shown here as it is too long), the best option is "PSIsContainer", since it is Boolean and doesn't require string comparison of the "mode" property, so it is faster. Directories are containers and files are not. We can use this property with "Where-Object" (alias "?") to quickly get all the file objects. Also, Get-ChildItem doesn't return hidden items, but we can use the "-Force" option to find those files as well.

```
PS C:\> ls -fo | ? { !$_.PSIsContainer }  
Directory: C:\  
Mode                LastWriteTime         Length      Name  
-a---             6/10/2009 4:42 PM        24      autoexec.bat  
-a---             6/10/2009 4:42 PM        10      config.sys  
-a-hs             1/3/2011 7:36 AM    1073741824  pagefile.sys
```

Ok, so now we have only files. Let's do something with them.



Find a String in a File



To search a file for a specific string, we can use "Select-String". This cmdlet is similar to Linux's "grep".

```
PS C:\> Select-String -path *.txt -Pattern pass  
user1.txt:1:my password is P@ssw0rd1  
user3.txt:1:my password is blank
```

Select-String doesn't recursively search the filesystem, but we can use input from Get-ChildItem (alias ls) and even use Get-ChildItem's filter and force options

```
PS C:\> ls -fo -fi *.txt -r | select-string pass  
user1.txt:1:my password is P@ssw0rd1  
user3.txt:1:my password is blank  
\Users\john\Desktop\p.txt:1:password is 4dm1nD00d
```

To search a file for a specific string, we can use "Select-String". This cmdlet is similar to Linux's "grep".

```
PS C:\> Select-String -path *.txt -Pattern password  
user1.txt:1:my password is P@ssw0rd1  
user3.txt:1:my password is blank
```

...or shorter:

```
PS C:\> Select-String password *.txt  
user1.txt:1:my password is P@ssw0rd1  
user3.txt:1:my password is blank
```

The "Select-String" cmdlet searches the file for the search pattern. The pattern can even be a Regular Expression. One notable limitation of "Select-String" is its inability to recursively search the filesystem. To do the recursive search, we have to use "Get-ChildItem" in conjunction with "Select-String". We can use the `-fi[ter]` option to only look in .txt files, `-r[ecursive]` to walk the file system, and `-fo[rce]` to look in hidden files and directories.

```
PS C:\> ls -fi *.txt -r -fo | select-string password  
user1.txt:1:my password is P@ssw0rd1  
user3.txt:1:my password is blank  
\Users\john\Desktop\p.txt:1:my password is 4dm1n1st4t0r
```



Total, Sort, and Count Numbers in a File



We start with a text file containing three lines:

- John Doe 90
- Jane Doe 89
- Freak Bean 97

Before we do any math, we need to extract the numbers from the file. There are many ways to do it, but we will use Import-Csv

Normally, a csv has header information and each field is separated by a comma or a tab, but we have spaces

We can use the Import-CSV command and specify a header and the delimiter (a space)

```
PS C:\> Import-Csv -Delimiter " " -Path scores.txt -Header "First",  
"Last", "Score"
```

First	Last	Score
----	----	-----
John	Doe	89
Jane	Doe	90
Frank	Bean	97

Import-CSV is an extremely powerful cmdlet. It is used to read all sorts of input. Many times output from other programs is saved in a CSV format where each field is separated with spaces, commas, or tabs. This command will quickly import the data and allow you to use cmdlets to filter, format, and process the data.

Let's say we have a text file containing these three lines of text that we want to manipulate with PowerShell:

```
John Doe 90  
Jane Doe 89  
Freak Bean 97
```

Before we can use the data, we need to parse it. We could do it manually, but Import-CSV is much easier. Commonly, a .csv file contains a header and the fields are comma (or tab) delimited. We can tell the cmdlet to use a different delimiter character and we can provide header information.

```
PS C:\> Import-Csv -Delimiter " " -Path scores.txt -  
Header "First", "Last", "Score"
```

First	Last	Score
----	----	-----
John	Doe	89
Jane	Doe	90
Frank	Bean	97

Now that the data has been objectified, we can use other cmdlets to sort, parse, manipulate, or measure the data.



Total, Sort, and Count Numbers in a File (2)



We've just taken a text file and converted it into objects!

We can sort

```
... | sort -Property Score -Descending
```

We can do statistics and use shortened parameter names

```
... | Measure-Object -Property Score -Ave -Min -Max
```

```
Count      : 3
Average    : 92
Sum        :
Maximum    : 97
Minimum    : 89
Property   : Score
```

The Measure-Object cmdlet is very handy for counting output, as well as other basic statistics

Let's take the data and sort it by the score:

```
PS C:\> Import-Csv -Delimiter " " -Path scores.txt -
Header "First", "Last", "Score" | sort -
Property Score -Descending
```

First	Last	Score
Frank	Bean	97
Jane	Doe	90
John	Doe	89

Once the data is converted to objects, PowerShell can be leveraged to perform statistics or other operations on the data. The Measure-Object cmdlet is just one of many options.

```
PS C:\> $scores = Import-Csv -Delimiter " " -Path
scores.txt -Header "First", "Last", "Score"
PS C:\> $scores | Measure-Object -Property Score -Ave
-Min -Max
```

```
Count      : 3
Average    : 92
Sum        :
Maximum    : 97
Minimum    : 89
Property   : Score
```



Exercise



- 1) Which command could produce the following output?

```
The file aaa.txt has a length of 12 bytes.
dir | % { "The file $_ has a length of $Length bytes." }
dir | % { "The file $_ has a length of `$_`.Length bytes." }
dir | % { "The file $_ has a length of $_.Length bytes." }
dir | % { "The file $_ has a length of $($_.Length) bytes." }
dir | % { "The file $_ has a length of ($_.Length) bytes." }
```

- 2) Which filter can be used with the "Get-ChildItem" cmdlet (alias "ls", "dir" and "gci") to find all files modified in the past day?

```
? { $_.LastAccessTime -ge (Get-Date).AddDays(-1) }
? { $_.LastAccessTime > (Get-Date).AddDays(-1) }
? { -not $_.PSIsContainer && $_.LastAccessTime > (Get-Date).AddDays(-1) }
? { -not $_.PSIsContainer && $_.LastAccessTime -ge (Get-Date).AddDays(-1) }
? { -not $_.PSIsContainer -and $_.LastAccessTime -ge (Get-Date).AddDays(-1) }
```

Exercise

- 1) Which command could produce the following output?

```
The file aaa.txt has a length of 12 bytes.
dir | % { "The file $_ has a length of $Length bytes." }
dir | % { "The file $_ has a length of `$_`.Length bytes." }
dir | % { "The file $_ has a length of $_.Length bytes." }
dir | % { "The file $_ has a length of $($_.Length) bytes." }
dir | % { "The file $_ has a length of ($_.Length) bytes." }
```

- 2) Which filter can be used with the "Get-ChildItem" cmdlet (alias "ls", "dir" and "gci") to find all files modified in the past day?

```
? { $_.LastAccessTime -ge (Get-Date).AddDays(-1) }
? { $_.LastAccessTime > (Get-Date).AddDays(-1) }
? { -not $_.PSIsContainer && $_.LastAccessTime > (Get-Date).AddDays(-1) }
? { -not $_.PSIsContainer && $_.LastAccessTime -ge (Get-Date).AddDays(-1) }
? { -not $_.PSIsContainer -and $_.LastAccessTime -ge (Get-Date).AddDays(-1) }
```



Answers



1) Which command could produce the following output?

The file aaa.txt has a length of 12 bytes.

```
dir | % { "The file $_ has a length of $($_.Length) bytes." }
```

- In a string you have to use the sub-expression operator to expand the object and its property

2) Which filter can be used with the "Get-ChildItem" cmdlet (alias "ls", "dir" and "gci") to find all files modified in the past day?

```
? { -not $_.PSIsContainer -and $_.LastAccessTime -ge (Get-Date).AddDays(-1) }
```

- This is the only option using the correct Logical AND (-and) and the correct comparison operator (-ge)

Answers

1) Which command could produce the following output?

The file aaa.txt has a length of 12 bytes.

```
dir | % { "The file $_ has a length of $($_.Length) bytes." }
```

In a string you have to use the sub-expression operator to expand the object and its property

2) Which filter can be used with the "Get-ChildItem" cmdlet (alias "ls", "dir" and "gci") to find all files modified in the past day?

```
? { -not $_.PSIsContainer -and $_.LastAccessTime -ge (Get-Date).AddDays(-1) }
```

This is the only option using the correct Logical AND (-and) and the correct comparison operator (-ge)



Conclusion for Module 3 - PowerShell



This concludes Module 3

- We've learned about the newest shell in Windows, how to interact with it, and basic scripting
- This shell is the most advanced method of interacting with Windows and Windows Server Software and being skilled in its use will provide a distinct advantage in the real world vs. those "stuck" with the GUI

We've learned about the newest shell in Windows, how to interact with it, and basic scripting. This shell is the most advanced method of interacting with Windows and Windows Server Software and being skilled in its use will provide a distinct advantage in the real world vs. those "stuck" with the GUI.



Module 3 - System Administration PowerShell

- Introduction
- Cmdlets
- Scripting, Variables & Syntax
- Flow Control & Output
- Practical Uses