2

Lab Answers

Windows PowerShell 3

Day 2

Chapter 9

Lab Answers

The Pipeline, Deeper

Consider the Get-ADComputer command. This command is installed on any Windows Server 2008 R2 or later domain controller – but you don't need one! You only need to know two things:

- The Get-ADComputer command has a –filter parameter; running Get-ADComputer –filter * will retrieve all computer objects in the domain.
- Domain computer objects have a Name property, which contains the computer's host name.
- Domain computer objects have the TypeName ADComputer. So, Get-ADComputer produces objects of the type ADComputer.

That's all you should need to know. With that in mind, complete these tasks:

> **NOTE:** You're not being asked to run these commands. Instead, you're being asked if these commands will function or not, and why. You've been told how Get-ADComputer works, and what it produces; you can read the help to discover what other commands expect and accept.

1. Would the following command work to retrieve a list of installed hotfixes from all servers in the specified domain? Why or why not? Write out an explanation, similar to the ones we provided earlier in this chapter.

```
Get-Hotfix -computerName (get-adcomputer -filter * |
Select-Object -expand name)
```

```
This should work because the nested Get-ADComputer expression will return a
collection of computernames and the –Computername parameter can accept an array
of values.
```

2. Would this alternative command work to retrieve the list of hotfixes from the same computers? Why or why not? Write out an explanation, similar to the ones we provided earlier in this chapter.

```
get-adcomputer -filter * |Get-HotFix
```

```
This won't work because Get-Hotfix doesn't accept any parameters by value.
Although it will accept –Computername by property name, but this command isn't
doing that.
```

3. Would this third version of the command work to retrieve the list of hotfixes from the domain controllers? Why or why not? Write out an explanation, similar to the ones I provided earlier in this chapter.

```
get-adcomputer -filter * |
Select-Object @{l='computername';e={$_.name}} |
Get-Hotfix
```

```
This should work. The first part of the expression is writing a custom object to
the pipeline that has a Computername property. This property can be bound to
the Computername parameter in Get-Hotfix because it accepts pipeline binding by
property name.
```

4. Write a command that uses pipeline parameter binding to retrieve a list of running processes from every computer in an AD domain. Don't use parentheses.

```
get-adcomputer -filter * |
Select-Object @{l='computername';e={$_.name}} | Get-Process
```

5. Write a command that retrieves a list of installed services from every computer in an AD domain. Don't use pipeline input; instead use a parenthetical command (a command in parentheses).

```
Get-Service –Computername (get-adcomputer -filter * | Select-Object –
expandproperty name)
```

6. Sometimes Microsoft forgets to add pipeline parameter binding to a cmdlet. For example, would the following command work to retrieve information from every domain controller in the domain? Write out an explanation, similar to the ones we provided earlier in this chapter.

```
get-adcomputer -filter * |
   Select-Object @{l='computername';e={$_.name}} |
Get-WmiObject -class Win32_BIOS
```

```
This will not work. The Computername parameter in Get-WMIObject doesn't take
any pipeline binding.
```

Chapter 10

Lab Answers

Formatting – and Why It's Done on the Right

See if you can complete the following tasks:

1. Display a table of processes that includes only the process names, IDs, and whether or not they're responding to Windows (the Responding property has that information). Have the table take up as little horizontal room as possible, but don't allow any information to be truncated.

```
get-process | format-table Name,ID,Responding -autosize -Wrap
```

2. Display a table of processes that includes the process names and IDs. Also include columns for virtual and physical memory usage, expressing those values in megabytes (MB).

```
get-process | format-table Name,ID,@{l='VirtualMB';e={$_.vm/1mb}},
@{l='PhysicalMB';e={$_.workingset/1MB}} -autosize
```

3. Use Get-EventLog to display a list of available event logs. (Hint: you'll need to read the help to learn the correct parameter to accomplish that.) Format the output as a table that includes, in this order, the log display name and the retention period. The column headers must be "LogName" and "RetDays."

```
Get-EventLog -List | Format-Table @{l='LogName';e={$_.LogDisplayname}},
@{l='RetDays';e={$_.MinimumRetentionDays}} -autosize
```

4. Display a list of services so that a separate table is displayed for services that are started and services that are stopped. Services that are started should be displayed first. (Hint: you'll use a -groupBy parameter).

```
Get-Service | sort Status -descending | format-table -GroupBy Status
```

Chapter 11

Lab Answers

Filtering and Comparisons

Following the principle of filter left, try to accomplish the following:

1. Import the NetAdapter module (available in the latest version of Windows, both client and server). Using the Get-NetAdapter cmdlet, display a list of non-virtual network adapters (that is, adapters whose Virtual property is False, which PowerShell represents with the special $False constant).

```
import-module NetAdapter
get-netadapter -physical
```

2. Import the DnsClient module (available in the latest version of Windows, both client and server). Using the Get-DnsClientCache cmdlet, display a list of A and AAAA records from the cache. Hint: If your cache comes up empty, try visiting a few Web pages first to force some items into the cache.

```
Import-Module DnsClient
Get-DnsClientCache -type AAAA,A
```

3. Display a list of hotfixes that are security updates.

```
Get-Hotfix -Description 'Security Update'
```

4. Using Get-Service, is it possible to display a list of services that have a start type of Automatic, but that aren't currently started?

```
No. The object that Get-Service uses doesn't have that information. We would
need to use WMI and the Win32_Service class.
```

5. Display a list of hotfixes that were installed by the Administrator, and which are updates. Note that some hotfixes won't have an "installed by" value – that's okay.

```
get-hotfix -Description Update | where {$_.InstalledBy -match "administrator"}
```

6. Display a list of all processes running as either Conhost or Svchost.

```
get-process -name svchost,conhost
```

Chapter
12

Lab Answers

A Practical Interlude

Windows 8 and Windows Server 2012 include a module for working with file shares. Your task is to create a directory called LABS on your computer and share it. For the sake of this exercise you can assume the folder and share don't already exist. Don't worry about NTFS permissions but you need to make sure share permissions are set so that everyone has Read/Write access and Administrators have full control. Since the share will be primarily for files you want to configure the share's caching mode for documents. Your script should show the new share and its permissions.

```
#hide the command output
mkdir C:\Labs | out-null

#import the module
import-module SMBShare

#this is a single line command
$share=New-smbshare -name 'LABS' -Path 'C:\Labs' -Description 'PowerShell Labs'
-ChangeAccess 'Everyone' -FullAccess 'Administrators' -CachingMode Documents

#display the share object
Write-Output $share

#pipe the share object to Get-SmbShareAccess to display permissions
$share | Get-SmbShareAccess
```

Chapter
13

Lab Answers

Remote Control: One to One, One to Many

It's time to start combining some of what you've learned about remoting with what you've learned in previous chapters. See if you can accomplish these tasks:

1. Make a one-to-one connection with a remote computer. Launch Notepad.exe. What happens?

```
Enter-PSSession Server01
[Server01] PS C:\Users\Administrator\Documents> Notepad
```

The Notepad process will launch, but there won't be any interactive process either locally or remotely. In fact, run this way, the prompt won't return until the Notepad process ends. Although an alternative command to launch it would be: Start-Process Notepad

2. Using Invoke-Command, retrieve a list of services that aren't started from one or two remote computers. Format the results as a wide list. (Hint: it's okay to retrieve results and have the formatting occur on your computer—don't include the Format- cmdlet in the commands that are invoked remotely).

```
Invoke-Command –scriptblock {get-service | where {$_.status -eq “stopped”}}
-computername Server01,Server02 | format-wide -Column 4
```

3. Use Invoke-Command to get a list of the top ten processes for virtual memory (VM) usage. Target one or two remote computers, if you can.

```
Invoke-Command -scriptblock {get-process | sort VM -Descending | Select -first
10} –computername Server01,Server02
```

4. Create a text file that contains three computer names, with one name per line. It's okay to use the same computer name three times if you only have access to one remote computer. Then use Invoke-Command to retrieve the 100 newest Application event log entries from the computer names listed in that file.

```
Invoke-Command -scriptblock {get-eventlog -LogName Application -Newest 100}
-ComputerName (Get-Content computers.txt)
```

Chapter 14

Lab Answers

Using Windows Management Instrumentation

Take some time to complete the following hands-on tasks. Much of the difficulty in using WMI is in finding the class that will give you the information you need, so much of the time you'll spend in this lab will be tracking down the right class. Try to think in keywords (I'll provide some hints), and use a WMI explorer to quickly search through classes (the WMI explorer we use lists classes alphabetically, making it easier for me to validate my guesses).

1. What class could be used to view the current IP address of a network adapter? Does the class have any methods that could be used to release a DHCP lease? (Hint: network is a good keyword here.)

You can use the Win32_NetworkAdapterConfiguration class.

If you run Get-Wmiobject for this class and pipe to Get-Member you should see a number of DHCP related methods. You can also find this using a CIM cmdlet:

```
Get-CimClass win32_networkadapterconfiguration | select -expand methods |
where Name -match "dhcp"
```

2. Create a table that shows a computer name, operating system build number, operating system description (caption), and BIOS serial number. (Hint: you've seen this technique, but you'll need to reverse it a bit and query the OS class first, then query the BIOS second).

```
get-wmiobject win32_operatingsystem | Select BuildNumber,Caption,
@{l='Computername';e={$_.__SERVER}},
@{l='BIOSSerialNumber';e={(gwmi win32_bios).serialnumber  }} | ft –auto
```

or using the CIM cmdlets:

```
get-ciminstance win32_operatingsystem | Select BuildNumber,Caption,
@{l='Computername';e={$_.CSName}},
@{l='BIOSSerialNumber';e={(get-ciminstance win32_bios).serialnumber  }} |
ft -auto
```

3. Query a list of hotfixes using WMI. (Hint: Microsoft formally refers to these as quick fix engineering). Is the list different from that returned by the Get-Hotfix cmdlet?

```
Get-wmiobject -class win32-QuickfixEngineering
```

4. Display a list of services, including their current status, their start mode, and the account they use to log on.

```
get-wmiobject win32_service | Select Name,State,StartMode,StartName
OR
get-ciminstance win32_service | Select Name,State,StartMode,StartName
```

5. Can you find a class that will display a list of installed software products? Do you consider the resulting list to be complete?

```
get-wmiobject -list *product
Get-wmiobject -class win32_Product
```

Chapter 15

Lab Answers

Multitasking with Background Jobs

The following exercises should help you understand how to work with the different types of jobs and tasks in PowerShell. As you work through these exercises, don't feel you have to write a one-line solution. Sometimes it is easier to break things down into separate steps.

1. Create a one-time background job to find all PowerShell scripts on the C: drive. Any task that might take a long time to complete is a great candidate for a job.

```
Start-Job {dir c:\ -recurse –filter '*.ps1'}
```

2. You realize it would be helpful to identify all PowerShell scripts on some of your servers. How would you run the same command from the previous exercise on a group of remote computers?

```
Invoke-Command –scritpblock {dir c:\ -recurse –filter *.ps1} –computername (get-
content computers.txt) -asjob
```

3. Create a background job that will get the latest 25 errors from the system event log on your computer and export them to a CLIxml file. You want this job to run every day, Monday through Friday at 6:00AM so that it is ready for you to look at when you come in to work.

```
$Trigger=New-JobTrigger -At "6:00AM" -DaysOfWeek "Monday","Tuesday","Wednesday",
"Thursday","Friday" –Weekly
$command={ Get-EventLog -LogName System -Newest 25 -EntryType Error |
Export-Clixml c:\work\25SysErr.xml}
Register-ScheduledJob -Name "Get 25 System Errors" -ScriptBlock $Command
-Trigger $Trigger
#check on what was created
Get-ScheduledJob | Select *
```

4. What cmdlet do you use to get the results of a job and how would you save the results in the job queue?

```
Receive-Job –id 1 -keep
```

Review Lab 2

Task 1

```
Get-Process | Format-Table –Property Name,ID –AutoSize
```

Task 2

```
Get-WmiObject –class Win32_UserAccount |
Format-Table –Property Domain,@{n='UserName';e={$_.Name}}
```

Task 3

```
Invoke-Command –ScriptBlock { Get-PSProvider } –computerName Computer1,Computer2
```

Task 4

```
Get-Service –computerName (Get-Content C:\Computers.txt)
```

Task 5

```
Get-WmiObject –class Win32_LogicalDisk –Filter "drivetype=3" |
Where-Object { $_.FreeSpace / $_.Size * 100 –gt 50}
```

Task 6

```
Get-WmiObject –namespace root\CIMv2 –list
```

Task 7

```
Get-WmiObject –class Win32_Service –filter "StartMode='Auto' AND State<>'Running'
```

…or…

```
Get-WmiObject –class Win32_Service |
Where-Object { $_.StartMode –eq 'Auto' –and $_.State –ne 'Running' }
```

Task 8

```
Send-MailMessage (read the full help to determine mandatory parameters)
```

Task 9

```
Get-ACL –Path C:\
```

Task 10

```
Get-ChildItem C:\Users | Get-ACL
```

Task 11

```
Start-Process
```

Task 12

```
Start-Sleep –seconds 10
```

Task 13

```
Help *operators*
```

Task 14

```
Use the Write-EventLog command
```

Task 15

```
Get-WmiObject –class Win32_Processor |
Select-Object –property Manufacturer,NumberOfCores,Name,@{
  n='MaxSpeed';e={$_.MaxClockSpeed}}
```

Task 16

```
Get-WmiObject –class Win32_Process | Where { $_.PeakWorkingSet –gt 5000 }
```