Labs

Learn Windows PowerShell 3 in a Month of Lunches

Chapter 3

Lab

Using the Help System

Lab Time: 30

1. First, run Update-Help and ensure it completes without errors. That will get a copy of the help on your local computer. This requires an Internet connection, and requires that the shell be running under elevated privileges (which means it must say “Administrator” in the shell’s window title bar).

2. Can you find any cmdlets capable of converting other cmdlets’ output into HTML?

3. Are there any cmdlets that can redirect output into a file, or to a printer?

4. How many cmdlets are available for working with processes? (Hint: remember that cmdlets all use a singular noun.)

5. What cmdlet might you use to write to an event log?

6. You’ve learned that aliases are nicknames for cmdlets; what cmdlets are available to create, modify, export, or import aliases?

7. Is there a way to keep a transcript of everything you type in the shell, and save that transcript to a text file?

8. It can take a long time to retrieve all of the entries from the Security event log. How can you get just the 100 most recent entries?

9. Is there a way to retrieve a list of the services that are installed on a remote computer?

10. Is there a way to see what processes are running on a remote computer?

11. Examine the help file for the Out-File cmdlet. The files created by this cmdlet default to a width of how many characters? Is there a parameter that would enable you to change that width?

12. By default, Out-File will overwrite any existing file that has the same filename as what you specify. Is there a parameter that would prevent the cmdlet from overwriting an existing file?

13. How could you see a list of all aliases defined in PowerShell?

14. Using both an alias and abbreviated parameter names, what is the shortest command line you could type to retrieve a list of running processes from a computer named Server1?

15. How many cmdlets are available that can deal with generic objects? (Hint: remember to use a singular noun like "object" rather than a plural one like "objects").

16. This chapter briefly mentioned arrays. What help topic could tell you more about them?

17. The Help command can also search the contents of a help file. Are there any topics that might explain any breaking changes between PowerShell v1 and PowerShell v2?

Chapter 4

Lab

Running Commands

Lab Time: 20

Using just what you learned in this chapter, and in the previous chapter on using the help system, complete the following tasks in Windows PowerShell:

1. Display a list of running processes.

2. Display the 100 most recent entries from the Application event log (don't use Get-WinEvent for this – We've shown you another command that will do this task).

3. Display a list of all commands that are of the "cmdlet" type (this is tricky – we've shown you Get-Command, but you're going to have to read the help to find out how to narrow down the list as we've asked).

4. Display a list of all aliases.

5. Make a new alias, so that you can run "d" to get a directory listing.

6. Display a list of services that begin with the letter "M." Again, read the help for the necessary command – and don't forget that "*" is a near-universal wildcard in PowerShell.

7. Display a list of all Windows Firewall rules. You'll need to use Help or Get-Command to discover the necessary cmdlet!

8. Display a list only of inbound Windows Firewall rules. Same cmdlet as above, but you'll need to read its help to discover the necessary parameter and its allowable values

Chapter 5 Lab

Working with Providers

Lab Time: 15

Complete the following tasks:

1. In the registry, go to HKEY_CURRENT_USER\software\microsoft\Windows\currentversion\explorer. Locate the Advanced key, and set its DontPrettyPath property to 1.

2. Create a zero-length file named C:\Test.txt (use New-Item).

3. Is it possible to use Set-Item to change the contents of C:\Test.txt to TESTING? Or do you get an error? If you get an error, why?

4. What are the differences between the –Filter, -Include, and –Exclude parameters of Get-ChildItem?

Chapter 6 Lab

The Pipeline: Connecting Commands

Lab Time: 30

1. Create two similar, but different, text files. Try comparing them using Diff. To do so, run something like this: Diff -reference (Get-Content File1.txt) -difference (Get-Content File2.txt). If the files have only one line of text that's different, the command should work.

2. What happens if you run Get-Service | Export-CSV services.csv | Out-File from the console? Why does that happen?

3. Apart from getting one or more services and piping them to Stop-Service, what other means does Stop-Service provide for you to specify the service or services you want to stop? Is it possible to stop a service without using Get-Service at all?

4. What if you wanted to create a pipe-delimited file instead of a comma-separated file? You would still use the Export-CSV command, but what parameters would you specify?

5. Is there a way to eliminate the # comment line from the top of an exported CSV file? That line normally contains type information, but what if you wanted to omit that from a particular file?

6. Export-CliXML and Export-CSV both modify the system, because they can create and overwrite files. What parameter would prevent them from overwriting an existing file? What parameter would ask you if you were sure before proceeding to write the output file?

7. Windows maintains several regional settings, which include a default list separator. On U.S. systems, that separator is a comma. How can you tell Export-CSV to use the system's default separator, rather than a comma?

Chapter

7

Lab

Adding Commands

Lab Time: 10

For this lab, you only have one task: run the Networking troubleshooting pack. When you successfully do so, you'll be asked for an "Instance ID;" just hit Enter. Then run a Web Connectivity check and ask for help connecting to a specific Web page. Use http://videotraining.interfacett.com as your test URL. Hopefully, you'll get a "No problems were detected" report, meaning you ran the check successfully.

To accomplish this task, you'll need to discover a command capable of getting a troubleshooting pack, and one capable of executing a troubleshooting pack. You'll also need to discover where the packs are located and how they're named. Everything you need to know is in PowerShell and the help system will find it for you.

That's all the help you get!

Chapter 8

Lab

Objects: Just Data by Another Name

Lab Time: 30

This chapter has probably covered more, and more difficult, new concepts than any chapter so far. Hopefully we were able to make it all make sense, but these exercises should help you cement everything. See if you can complete them all, and remember that there are companion videos and sample solutions at MoreLunches.com. Some of these tasks will draw on skills you learned in previous chapters, as a way of refreshing your memory and keeping you sharp.

1. Identify a cmdlet that will produce a random number.

2. Identify a cmdlet that will display the current date and time.

3. What type of object does the cmdlet from task #2 produce? (What is the type name of the object produced by the cmdlet?)

4. Using the cmdlet from task #2 and Select-Object, display only the current day of the week in a table like this (caution: The output will right-align, so make sure your PowerShell window doesn't have a horizontal scroll bar):

5. Identify a cmdlet that will display information about installed hotfixes.

6. Using the cmdlet from task #5, display a list of installed hotfixes. Sort the list by the installation date, and display only the installation date, the user who installed the hotfix, and the hotfix ID.

7. Repeat task #6, but this time sort the results by the hotfix description, and include the description, the hotfix ID, and the installation date. Put the results into an HTML file.

8. Display a list of the 50 newest entries from the Security event log (you can use a different log, such as System or Application if your Security log is empty). Sort the list so that the oldest entries appear first and so that entries made at the same time are sorted by their index. Display the index, time, and source for each entry. Put this information into a text file (not an HTML file, just a plain text file). You may be tempted to use Select-Object and its –first or –last parameters to achieve this; don’t. There’s a better way. Also, avoid using Get-WinEvent for now – there’s a better cmdlet to work with for this particular task.

Chapter 9 Lab

The Pipeline, Deeper

Lab Time: 30

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 domain controllers 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)
```

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
```

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
```

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.

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).

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
```

Chapter

10 Lab

Formatting – and Why It's Done on the Right

Lab Time: 30

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.

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).

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."

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).

Chapter 11 Lab

Filtering and Comparisons

Lab Time: 30

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).

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.

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

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?

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.

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

Chapter 12

Lab

A Practical Interlude

Lab Time: 20

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.

Chapter 13 Lab

Remote Control: One to One, One to Many

Lab Time: 20

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?

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).

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.

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.

Chapter 14 Lab

Using Windows Management Instrumentation

Lab Time: 30

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.)

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).

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?

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

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

Chapter 15 Lab

Multitasking with Background Jobs

Lab Time: 20

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.

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?

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.

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

Chapter 16 Lab

Working with Bunches of Objects, One at a Time

Lab Time: 15

See if you can answer the following questions and complete the specified tasks. This is an especially important lab because it draws on skills that you've learned in many previous chapters and that you should be continuing to use and reinforce as you progress through the remainder of this book.

1. What method of a ServiceController object (produced by Get-Service) will pause the service without stopping it completely?

2. What method of a Process object (produced by Get-Process) would terminate a given process?

3. What method of a WMI Win32_Process object would terminate a given process?

4. Write four different commands that could be used to terminate all processes named "Notepad" assuming that multiple processes might be running under that same name.

Chapter 17 Lab

Security Alert!

Lab Time: 10

See if you can answer the following questions and complete the specified tasks. This is an especially Your task in this lab is simple—so simple, in fact, this is more of a break than a lab. Configure your shell to allow script execution. Use the Set-ExecutionPolicy cmdlet, and use the RemoteSigned policy setting.

Enjoy a short break and then its time to get started on Chapter 18.

Chapter 18 Lab

Variables: A Place to Store Your Stuff

Lab Time: 15

Create a background job that queries the Win32_BIOS information from two computers (use your computer's name and "localhost" if you only have one computer to experiment with). When the job finishes running, receive the results of the job into a variable. Then, display the contents of that variable. Finally, export the variable's contents to a CliXML file.

Chapter 19 Lab

Input and Output

Lab Time: 20

Write-Host and Write-Output can be a bit tricky to work with. See how many of these tasks you can complete and if you get completely stuck, it's okay to peek at the sample answers.

1. Use Write-Output to display the result of 100 multiplied by 10.

2. Use Write-Host to display the result of 100 multiplied by 10.

3. Prompt the user to enter a name, and then display that name in yellow text.

4. Prompt the user to enter a name, and then display that name only if it's longer than 5 characters. Do this all in a single line—don't use a variable.

Chapter 20 Lab

Sessions: Remote Control, with Less Work

Lab Time: 20

To complete this lab, you're going to want to have two computers: one to remote from, and another to remote to. If you only have one computer, use its computer name to remote to it. You should get the same essential experience that way.

1. Close all open sessions in your shell.

2. Establish a session to the remote computer. Save the session in a variable named $session.

3. Use the $session variable to establish a one-to-one remote shell session with the remote computer. Display a list of processes, and then exit.

4. Use the $session variable with Invoke-Command to get a list of services from the remote computer.

5. Use Get-PSSession and Invoke-Command to get a list of the 20 most recent Security event log entries from the remote computer.

6. Use Invoke-Command and your $session variable to load the ServerManager module on the remote computer.

7. Import the ServerManager module's commands from the remote computer to your computer. Add the prefix "rem" to the imported commands' nouns.

8. Run the imported Get-WindowsFeature command.

9. Close the session that’s in your $session variable.

Chapter 21 Lab

You Call This Scripting?

Lab Time: 20

The following command is for you to add to a script. You should first identify any elements that should be parameterized, such as the computer name. Your final script should define the parameter, and you should create comment-based help within the script. Run your script to test it, and use the Help command to make sure your comment-based help works properly. Don't forget to read the help files referenced within this chapter for more information.

Here's the command:

```
Get-WmiObject -class Win32_LogicalDisk -computer "localhost" -filter
   "drivetype=3" |
Where { $_.FreeSpace / $_.Size -lt .1 } |
Select -Property DeviceID,FreeSpace,Size
```

Here's a hint: There are at least two pieces of information that will need to be parameterized. This command is intended to list all drives that have less than a given amount of free disk space. Obviously, you won't always want to target localhost, and you might not want 10% (that is, .1) to be your free space threshold. You could also choose to parameterize the drive type (which is 3, here), but for this lab leave that hardcoded with the value 3.

Chapter 22 Lab

Improving Your Parameterized Script

Lab Time: 30

This lab is going to require you to recall some of what you learned in Chapter 21, because you'll be taking the below command, parameterizing it, and turning it into a script – just like you did for the lab in Chapter 21. However, this time we also want you to make the –computerName parameter mandatory and give it a "hostname" alias. Have your script display verbose output before and after it runs this command, too. Remember, you have to parameterize the computer name – but that's the only thing you have to parameterize in this case. Be sure to run the command as-is before you start modifying it, to make sure it works on your system.

```
get-wmiobject win32_networkadapter -computername localhost |
where { $_.PhysicalAdapter } |
select MACAddress,AdapterType,DeviceID,Name,Speed
```

So to reiterate, here's your complete task list:

- Make sure the command runs as-is before modifying it.
- Parameterize the computer name.
- Make the computer name parameter mandatory.
- Give the computer name parameter an alias, "hostname."
- Add comment-based help with at least one example of how to use the script.
- Add verbose output before and after the modified command.
- Save the script as Get-PhysicalAdapters.ps1.

Chapter 23 Lab

Advanced Remoting Configuration Lab

Time: 20

Create a Remoting endpoint named TestPoint on your local computer. Configure the endpoint so that the SmbShare module is loaded automatically, but so that only the Get-SmbShare cmdlet is visible from that module. Also ensure that key cmdlets like Exit-PSSession are available, but no other core PowerShell cmdlets can be used. Don't worry about specifying special endpoint permissions or designating a "Run As" credential.

Test your endpoint by connecting to it using Enter-PSSession (specify localhost as the computer name, and TestPoint as the configuration name). When connected, run Get-Command to ensure that only the designated handful of commands can be seen.

Note that this lab might only be possible on Windows 8, Windows Server 2012, and later versions of Windows – the SmbShare module didn't ship with earlier versions of Windows.

Chapter 24

Using Regular Expressions to Parse Text Files

Lab Time: 20

Make no mistakes about it, regular expressions can make your head spin so don't try to create complex regex's right off the bat. Start simple and here are a few exercises to ease you into it. Use regular expressions and operators to complete the following:

1. Get all files in your Windows directory that have a 2 digit number as part of the name.

2. Find all processes running on your computer that are from Microsoft and display the process ID, name and company name. Hint: Pipe Get-Process to Get-Member to discover property names.

3. In the Windows Update log, usually found in C:\Windows, you want to display only the lines where the agent began installing files. You may need to open the file in Notepad to figure out what string you need to select.

Chapter 26 Lab

Using Someone Else's Script

Lab Time: 30

Below is a complete script. See if you can figure out what it does, and how to use it. Can you predict any errors that this might cause? What might you need to do in order to use this in your environment? Note that this script should run as-is… but, if it doesn't on your system, do you think you can track down the cause of the problem? Keep in mind that you've seen most of these commands – and for the ones you haven't there are the PowerShell help files. Those files' examples include every technique shown in this script.

```
function get-LastOn {
<#
.DESCRIPTION
Tell me the most recent event log entries for logon or logoff.
.BUGS
Blank 'computer' column

.EXAMPLE
get-LastOn -computername server1 | Sort-Object time -Descending |
Sort-Object id -unique | format-table -AutoSize -Wrap
ID              Domain       Computer Time
--              ------       -------- ----
LOCAL SERVICE   NT AUTHORITY          4/3/2012 11:16:39 AM
NETWORK SERVICE NT AUTHORITY          4/3/2012 11:16:39 AM
SYSTEM          NT AUTHORITY          4/3/2012 11:16:02 AM

Sorting -unique will ensure only one line per user ID, the most recent.
Needs more testing

.EXAMPLE
PS C:\Users\administrator> get-LastOn -computername server1 -newest 10000
 -maxIDs 10000 | Sort-Object time -Descending |

 Sort-Object id -unique | format-table -AutoSize -Wrap
```

```
ID              Domain                 Computer Time
--              ------                 -------- ----
Administrator   USS                   4/11/2012 10:44:57 PM
ANONYMOUS LOGON NT AUTHORITY          4/3/2012 8:19:07 AM
LOCAL SERVICE   NT AUTHORITY          10/19/2011 10:17:22 AM
NETWORK SERVICE NT AUTHORITY          4/4/2012 8:24:09 AM
student         WIN7                  4/11/2012 4:16:55 PM
SYSTEM          NT AUTHORITY          10/18/2011 7:53:56 PM
USSDC$          USS                   4/11/2012 9:38:05 AM
WIN7$           USS                   10/19/2011 3:25:30 AM


PS C:\Users\administrator>

.EXAMPLE
get-LastOn -newest 1000 -maxIDs 20
Only examines the last 1000 lines of the event log

.EXAMPLE
get-LastOn -computername server1| Sort-Object time -Descending |
Sort-Object id -unique | format-table -AutoSize -Wrap
#>

param (
        [string]$ComputerName = ‘localhost’,
        [int]$Newest = 5000,
        [int]$maxIDs = 5,
        [int]$logonEventNum = 4624,
        [int]$logoffEventNum = 4647
    )

    $eventsAndIDs = Get-EventLog -LogName security -Newest $Newest |
    Where-Object {$_.instanceid -eq $logonEventNum -or
[CA]$_.instanceid -eq  $logoffEventNum} |
    Select-Object -Last $maxIDs
[CA]-Property TimeGenerated,Message,ComputerName

    foreach ($event in $eventsAndIDs) {
        $id = ($event |
        parseEventLogMessage |
        where-Object {$_.fieldName -eq “Account Name”}  |
        Select-Object -last 1).fieldValue

        $domain = ($event |
        parseEventLogMessage |
        where-Object {$_.fieldName -eq “Account Domain”}  |
        Select-Object -last 1).fieldValue

        $props = @{‘Time’=$event.TimeGenerated;
            ‘Computer’=$ComputerName;
            ‘ID’=$id
            ‘Domain’=$domain}

        $output_obj = New-Object -TypeName PSObject -Property $props
        write-output $output_obj
    }
}

function parseEventLogMessage()
{
    [CmdletBinding()]
    param (
        [parameter(ValueFromPipeline=$True,Mandatory=$True)]
        [string]$Message
    )

    $eachLineArray = $Message -split “`n”

    foreach ($oneLine in $eachLineArray) {
        write-verbose “line:_$oneLine_”
        $fieldName,$fieldValue = $oneLine -split “:”, 2
            try {
                $fieldName = $fieldName.trim()
                $fieldValue = $fieldValue.trim()
            }
            catch {
                $fieldName = “”
            }

            if ($fieldName -ne “” -and $fieldValue -ne “” )
            {
            $props = @{‘fieldName’=”$fieldName”;
                    ‘fieldValue’=$fieldValue}

            $output_obj = New-Object -TypeName PSObject -Property $props
            Write-Output $output_obj
            }
    }
}
Get-LastOn
```

Appendix A

Lab

Review Labs

Review Lab 1

Task 1

Run a command that will display the newest 100 entries from the Application event log. Note: Do not use Get-WinEvent.

Task 2

Write a command line that displays only the 5 top processes based on virtual memory (VM) usage.

Task 3

Create a CSV file that contains all services, including only the service names and status. Have Running services listed BEFORE Stopped services.

Task 4

Write a command line that changes the startup type of the BITS service to Manual.

Task 5

Display a list of all files named win*.* on your computer. Start in the C:\ folder. Note: You may need to experiment and use some new parameters of a cmdlet in order to complete this task.

Task 6

Get a directory listing for C:\Program Files. Include all subfolders, and have the directory listing go into a text file named C:\Dir.txt (remember to use the > redirector, or the Out-File cmdlet).

Task 7

Get a list of the most recent 20 entries from the Security event log, and convert the information to XML. Do not create a file on disk: Have the XML display in the console window.

Note that the XML may display as a single top-level object, rather than as raw XML data – that's fine. That's just how PowerShell displays XML. You can pipe the XML object to Format-Custom to see it expanded out into an object hierarchy, if you like.

Task 8

Get a list of services, and export the data to a CSV file named C:\services.csv.

Task 9

Get a list of services. Keep only the services' names, display names, and status, and send that information to an HTML file. Have the phrase "Installed services" displayed in the HTML file before the table of service information.

Task 10

Create a new alias named D, which runs Get-ChildItem. Export just that alias to a file. Now, close the shell and open a new console window. Import that alias into the shell. Make sure you can run D and get a directory listing.

Task 11

Display a list of event logs that are available on your system.

Task 12

Run a command that will display the current directory that the shell is in.

Task 13

Run a command that will display the most recent commands that you have run in the shell. Locate the command that you ran for Task 11. Using two commands connected by a pipeline, re-run the command from Task 11.

In other words, if "Get-Something" is the command that retrieves historical commands, if "5" is the ID number of the command from Task 11, and "Do-Something" is the command that runs historical commands, run this:

```
Get-Something –id 5 | Do-Something
```

Of course, those aren't the correct cmdlet names – you'll need to find those. Hint: Both commands that you need have the same noun.

Task 14

Run a command that modifies the Security event log to overwrite old events as needed.

Task 15

Use the New-Item cmdlet to make a new directory named C:\Review. This is not the same as running Mkdir; the New-Item cmdlet will need to know what kind of new item you want to create. Read the help for the cmdlet.

Task 16

Display the contents of this registry key:

HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders

Note: “User Shell Folders” is not exactly like a directory. If you CD into it, you won’t see anything in a directory listing. User Shell Folders is an item, and what it contains are item properties. There’s a cmdlet capable of displaying item properties (although cmdlets use singular nouns, not plural).

Task 17

Find (but please do not run) cmdlets that can…

- Restart a computer
- Shut down a computer
- Remove a computer from a workgroup or domain
- Restore a computer’s System Restore checkpoint

Task 18

What command do you think could change a registry value? Hint: It’s the same noun as the cmdlet you found for Task 16.

Review Lab 2

Task 1

Display a list of running processes in a table that includes only the process names and ID numbers. Do not let the table have a large blank area between the two columns.

Task 2

Run this:

```
Get-WmiObject -class Win32_UserAccount
```

Now run that same command again, but format the output into a table that has a Domain and UserName column. The UserName column should show the users' Name property. E.g.,

```
 Domain   UserName
 =======  ========
 COMPANY  DonJ
```

Make sure the second column header says UserName, and not Name.

Task 3

Have two computers (it's OK to use localhost twice) run this command:

```
Get-PSProvider
```

Use Remoting to do this. Ensure that the output includes the computer names.

Task 4

Use Notepad to create a file named C:\Computers.txt. In that file, put the following:

```
Localhost
localhost
```

So you should have those two names on their own line in the file – two lines total. Save the file and close Notepad. Then, write a command that will list the running services on the computer names in C:\Computers.txt.

Task 5

Query all instances of Win32_LogicalDisk. Display only those instances which have a DriveType property containing 3, and who have 50% or more free disk space. Hint: To calculate free space percentage, it's freespace/size * 100.

Note that the –Filter parameter of Get-WmiObject cannot contain mathematical expressions.

Task 6

Display a list of all WMI classes in the root\CIMv2 namespace.

Task 7

Display a list of all Win32_Service instances where the StartMode is "Auto" and the State is not "Running."

Task 8

Find a command that can send e-mail messages. What are the mandatory parameters of this command?

Task 9

Run a command that will display the folder permissions on C:\.

Task 10

Run a command that will display the permissions on every subfolder of C:\Users. Just the direct subfolders; you do not need to recurse all files and folders. You will need to pipe one command to another command to achieve this.

Task 11

Find a command that will start Notepad under a credential other than the one you've used to log into the shell.

Task 12

Run a command that makes the shell pause, or idle, for 10 seconds.

Task 13

Can you find a help file (or files) that explains the shell's various operators?

Task 14

Write an informational message to the Application event log. Use a category of 1 and raw data of 100,100.

Task 15

Run this command:

```
Get-WmiObject -Class Win32_Processor
```

Study the default output of this command. Now, modify the command so that it displays in a table. The table should include each processor's number of cores, manufacturer, and Name. Also include a column called "MaxSpeed" that contains the processor's maximum clock speed.

Task 16

Run this command:

```
Get-WmiObject -Class Win32_Process
```

Study the default output of this command, and pipe it to Get-Member if you want. Now, modify the command so that only processes with a peak working set size greater than 5,000 are displayed.

Review Lab 3

```
Start-Job
Invoke-Command
Yes
Read-Host
Write-Output
```

Task 1

Create a list of running processes. The list should include only process names, IDs, VM, and PM columns. Put the list into an HTML-formatted file named C:\Procs.html. Make sure that the HTML file has an embedded title of, “Current Processes.” Display the file in a Web browser and make sure that title appears in the browser window’s title bar.

Task 2

“`t” (backtick T inside double quotes) is PowerShell’s escape sequence for a horizontal tab. Create a tab-delimited file named C:\Services.tdf that contains all services on your computer. Include only the services’ names, display names, and status.

Task 3

Repeat Task 1, modifying your command so that the VM and PM columns of the HTML file display values in megabytes (MB), instead of bytes. The formula to calculate megabytes, displaying the value as a whole number, goes something like $_.VM / 1MB –as [int] for the VM property.

Lab Answers

Learn Windows PowerShell 3
in a Month of Lunches

Chapter 3 Lab Answers

Using the Help System

1. First, run Update-Help and ensure it completes without errors. That will get a copy of the help on your local computer. This requires an Internet connection, and requires that the shell be running under elevated privileges (which means it must say "Administrator" in the shell's window title bar).

```
Update-Help
or if you run it more than once in a single day:
Update-Help –force
```

2. Can you find any cmdlets capable of converting other cmdlets' output into HTML?

```
Help html
Or you could try with Get-Command
get-command -noun html
```

3. Are there any cmdlets that can redirect output into a file, or to a printer?

```
get-command -noun file,printer
```

4. How many cmdlets are available for working with processes? (Hint: remember that cmdlets all use a singular noun.)

```
Get-command –noun process
Or
Help *Process
```

5. What cmdlet might you use to write to an event log?

```
get-command -verb write -noun eventlog
or if you weren't sure about the noun, use a wildcard
help *log
```

6. You’ve learned that aliases are nicknames for cmdlets; what cmdlets are available to create, modify, export, or import aliases?

```
Help *alias
Or
get-command -noun alias
```

7. Is there a way to keep a transcript of everything you type in the shell, and save that transcript to a text file?

```
Help transcript
```

8. It can take a long time to retrieve all of the entries from the Security event log. How can you get just the 100 most recent entries?

```
help Get-EventLog -Parameter Newest
```

9. Is there a way to retrieve a list of the services that are installed on a remote computer?

```
help Get-Service -Parameter computername
```

10. Is there a way to see what processes are running on a remote computer?

```
Help Get-Process –Parameter computername
```

11. Examine the help file for the Out-File cmdlet. The files created by this cmdlet default to a width of how many characters? Is there a parameter that would enable you to change that width?

```
Help Out-File –full
Or
Help Out-File –Parameter Width
```

12. By default, Out-File will overwrite any existing file that has the same filename as what you specify. Is there a parameter that would prevent the cmdlet from overwriting an existing file?

```
Help Out-File –full and look at parameters you should see –NoClobber.
```

13. How could you see a list of all aliases defined in PowerShell?

```
Get-alias
```

14. Using both an alias and abbreviated parameter names, what is the shortest command line you could type to retrieve a list of running processes from a computer named Server1?

```
ps –c server1
```

15. How many cmdlets are available that can deal with generic objects? (Hint: remember to use a singular noun like "object" rather than a plural one like "objects").

```
get-command -noun object
```

16. This chapter briefly mentioned arrays. What help topic could tell you more about them?

```
help about_arrays
or if you weren't sure, use wildcards
help *array*
```

17. The Help command can also search the contents of a help file. Are there any topics that might explain any breaking changes between PowerShell v1 and PowerShell v2?

```
-help *Powershell*
```

Chapter 4

Lab Answers

Running Commands

Using just what you learned in this chapter, and in the previous chapter on using the help system, complete the following tasks in Windows PowerShell:

1. Display a list of running processes.

```
get-process
```

2. Display the 100 most recent entries from the Application event log (don't use Get-WinEvent for this – We've shown you another command that will do this task).

```
get-eventlog –logname Application –newest 100
```

3. Display a list of all commands that are of the "cmdlet" type (this is tricky – we've shown you Get-Command, but you're going to have to read the help to find out how to narrow down the list as we've asked).

```
Get-Command -CommandType cmdlet
```

4. Display a list of all aliases.

```
Get-Alias
```

5. Make a new alias, so that you can run "d" to get a directory listing.

```
New-Alias -Name d -Value Get-ChildItem
```

6. Display a list of services that begin with the letter "M." Again, read the help for the necessary command – and don't forget that "*" is a near-universal wildcard in PowerShell.

```
Get-Service -Name m*
```

7. Display a list of all Windows Firewall rules. You'll need to use Help or Get-Command to discover the necessary cmdlet!

```
Get-NetFirewallRule
```

8. Display a list only of inbound Windows Firewall rules. Same cmdlet as above, but you'll need to read its help to discover the necessary parameter and its allowable values

```
Get-NetFirewallRule -Direction Inbound
```

Chapter 5 Lab Answers

Working with Providers

Complete the following tasks:

1. In the registry, go to HKEY_CURRENT_USER\software\microsoft\Windows\currentversion\explorer. Locate the Advanced key, and set its DontPrettyPath property to 1.

```
cd HKCU:\software\microsoft\Windows\currentversion\explorer
cd advanced
Set-ItemProperty -Path . -Name DontPrettyPath -Value 1
```

2. Create a zero-length file named C:\Test.txt (use New-Item).

```
New-Item -Name test.txt -ItemType file
```

3. Is it possible to use Set-Item to change the contents of C:\Test.txt to TESTING? Or do you get an error? If you get an error, why?

```
The file system provider does not support this action.
```

4. What are the differences between the –Filter, -Include, and –Exclude parameters of Get-ChildItem?

Include and exclude must be used with –Recurse or if querying a container. Filter uses the PSProviders filter capability which not all Providers support. For example, you could use DIR –filter in the file system but not in the registry. Although you could use DIR –include in the registry to achieve almost the same type of filtering result.

Chapter 6 Lab Answers

The Pipeline: Connecting Commands

1. Create two similar, but different, text files. Try comparing them using Diff. To do so, run something like this: Diff -reference (Get-Content File1.txt) -difference (Get-Content File2.txt). If the files have only one line of text that's different, the command should work.

```
PS C:\> "I am the walrus" | out-file file1.txt
PS C:\> "I'm a believer" | out-file file2.txt
PS C:\> $f1=get-content .\file1.txt
PS C:\> $f2=Get-Content .\file2.txt
PS C:\> diff $f1 $f2

InputObject                     SideIndicator
-----------                     -------------
I'm a believer           =>
I am the walrus          <=
```

2. What happens if you run Get-Service | Export-CSV services.csv | Out-File from the console? Why does that happen?

```
If you don't specify a file name with Out-File you'll get an error. But even if
you do Out-File won't really do anything because the file is actually created by
Export-CSV.
```

3. Apart from getting one or more services and piping them to Stop-Service, what other means does Stop-Service provide for you to specify the service or services you want to stop? Is it possible to stop a service without using Get-Service at all?

```
Stop-Service can accept one or more service names as a parameter values for the
   -Name parameter. For example, you could run:
Stop-Service spooler
```

4. What if you wanted to create a pipe-delimited file instead of a comma-separated file? You would still use the Export-CSV command, but what parameters would you specify?

```
get-service  | Export-Csv services.csv -Delimiter “|”
```

5. Is there a way to eliminate the # comment line from the top of an exported CSV file? That line normally contains type information, but what if you wanted to omit that from a particular file?

```
The parameter -NoTypeInformation
```

6. Export-CliXML and Export-CSV both modify the system, because they can create and overwrite files. What parameter would prevent them from overwriting an existing file? What parameter would ask you if you were sure before proceeding to write the output file?

```
get-service  | Export-Csv services.csv –noclobber
get-service  | Export-Csv services.csv -confirm
```

7. Windows maintains several regional settings, which include a default list separator. On U.S. systems, that separator is a comma. How can you tell Export-CSV to use the system’s default separator, rather than a comma?

```
get-service  | Export-Csv services.csv -UseCulture
```

Chapter 7

Lab Answers

Adding Commands

For this lab, you only have one task: run the Networking troubleshooting pack. When you successfully do so, you'll be asked for an "Instance ID;" just hit Enter. Then run a Web Connectivity check and ask for help connecting to a specific Web page. Use http://videotraining.interfacett.com as your test URL. Hopefully, you'll get a "No problems were detected" report, meaning you ran the check successfully.

To accomplish this task, you'll need to discover a command capable of getting a troubleshooting pack, and one capable of executing a troubleshooting pack. You'll also need to discover where the packs are located and how they're named. Everything you need to know is in PowerShell and the help system will find it for you.

That's all the help you get!

```
Here is one way to approach this:
get-module *trouble* -list
import-module TroubleShootingPack
get-command -Module TroubleShootingPack
help get-troubleshootingpack –full
help Invoke-TroubleshootingPack -full
dir C:\windows\diagnostics\system
$pack=get-troubleshootingpack C:\windows\diagnostics\system\Networking
Invoke-TroubleshootingPack $pack
Enter
1
2
http://videotraining.interfacett.com
```

Chapter 8

Lab Answers

Objects: Just Data by Another Name

This chapter has probably covered more, and more difficult, new concepts than any chapter so far. Hopefully we were able to make it all make sense, but these exercises should help you cement everything. See if you can complete them all, and remember that there are companion videos and sample solutions at MoreLunches.com. Some of these tasks will draw on skills you learned in previous chapters, as a way of refreshing your memory and keeping you sharp.

1. Identify a cmdlet that will produce a random number.

```
Get-Random
```

2. Identify a cmdlet that will display the current date and time.

```
Get-Date
```

3. What type of object does the cmdlet from task #2 produce? (What is the type name of the object produced by the cmdlet?)

```
System.DateTime
```

4. Using the cmdlet from task #2 and Select-Object, display only the current day of the week in a table like this (caution: The output will right-align, so make sure your PowerShell window doesn't have a horizontal scroll bar):

```
Get-Date | select DayofWeek
```

5. Identify a cmdlet that will display information about installed hotfixes.

```
Get-Hotfix
```

6. Using the cmdlet from task #5, display a list of installed hotfixes. Sort the list by the installation date, and display only the installation date, the user who installed the hotfix, and the hotfix ID.

```
Get-HotFix  | Sort InstalledOn | Select InstalledOn,InstalledBy,HotFixID
```

7. Repeat task #6, but this time sort the results by the hotfix description, and include the description, the hotfix ID, and the installation date. Put the results into an HTML file.

```
Get-HotFix  | Sort Description | Select Description,InstalledOn,InstalledBy,
HotFixID | ConvertTo-Html -Title "HotFix Report" | Out-File HotFixReport.htm
```

8. Display a list of the 50 newest entries from the Security event log (you can use a different log, such as System or Application if your Security log is empty). Sort the list so that the oldest entries appear first and so that entries made at the same time are sorted by their index. Display the index, time, and source for each entry. Put this information into a text file (not an HTML file, just a plain text file). You may be tempted to use Select-Object and its –first or –last parameters to achieve this; don't. There's a better way. Also, avoid using Get-WinEvent for now – there's a better cmdlet to work with for this particular task.

```
Get-EventLog -LogName System -Newest 50 | Sort TimeGenerated,Index | Select
   Index,TimeGenerated,Source | Out-File elogs.txt
```

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 –scriptblock {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
```

Chapter 16

Lab Answers

Working with Bunches of Objects, One at a Time

See if you can answer the following questions and complete the specified tasks. This is an especially important lab because it draws on skills that you've learned in many previous chapters and that you should be continuing to use and reinforce as you progress through the remainder of this book.

1. What method of a ServiceController object (produced by Get-Service) will pause the service without stopping it completely?

```
get-service | Get-Member -MemberType Method
Pause() method.
```

2. What method of a Process object (produced by Get-Process) would terminate a given process?

```
get-process | Get-Member -MemberType Method
Kill() method.
```

3. What method of a WMI Win32_Process object would terminate a given process?

```
Get-CimClass win32_process | select -ExpandProperty methods

Terminate() method.
```

4. Write four different commands that could be used to terminate all processes named "Notepad" assuming that multiple processes might be running under that same name.

```
get-process Notepad | stop-process
stop-process -name Notepad
get-process notepad | foreach {$_.Kill()}
Get-WmiObject win32_process -filter {name='notepad.exe'} | Invoke-WmiMethod
-Name Terminate
```

Chapter

17 Lab Answers

Security Alert!

See if you can answer the following questions and complete the specified tasks. This is an especially Your task in this lab is simple—so simple, in fact, this is more of a break than a lab. Configure your shell to allow script execution. Use the Set-ExecutionPolicy cmdlet, and use the RemoteSigned policy setting.

Enjoy a short break and then its time to get started on Chapter 18.

Chapter 18 Lab Answers

Variables: A Place to Store Your Stuff

For this lab, create a background job that queries the Win32_BIOS information from two computers (use your computer's name and "localhost" if you only have one computer to experiment with). When the job finishes running, receive the results of the job into a variable. Then, display the contents of that variable. Finally, export the variable's contents to a CliXML file.

```
invoke-command {get-wmiobject win32_bios} -comp localhost,$env:computername
   -asjob
$results=Receive-Job 4 -keep
$results
$results | export-clixml bios.xml
```

Chapter 19

Lab Answers

Input and Output

Write-Host and Write-Output can be a bit tricky to work with. See how many of these tasks you can complete, and if you get completely stuck, it's okay to peek at the sample answers

1. Use Write-Output to display the result of 100 multiplied by 10.

```
write-output (100*10)
```

2. Use Write-Host to display the result of 100 multiplied by 10.

```
Any of these approaches would work:
$a=100*10
Write-Host $a
Write-Host “The value of 100*10 is $a”
Write-Host (100*10)
```

3. Prompt the user to enter a name, and then display that name in yellow text.

```
$name=Read-Host “Enter a name”
Write-host $name -ForegroundColor Yellow
```

4. Prompt the user to enter a name, and then display that name only if it's longer than 5 characters. Do this all in a single line—don't use a variable.

```
Read-Host “Enter a name” | where {$_.length -gt 5}
```

Chapter 20 Lab Answers

Sessions: Remote Control, with Less Work

To complete this lab, you're going to want to have two computers: one to remote from, and another to remote to. If you only have one computer, use its computer name to remote to it. You should get the same essential experience that way.

1. Close all open sessions in your shell.

```
get-pssession | Remove-PSSession
```

2. Establish a session to the remote computer. Save the session in a variable named $session.

```
$session=new-pssession –computername localhost
```

3. Use the $session variable to establish a one-to-one remote shell session with the remote computer. Display a list of processes, and then exit.

```
enter-pssession $session
Get-Process
Exit
```

4. Use the $session variable with Invoke-Command to get a list of services from the remote computer.

```
invoke-command -ScriptBlock { get-service } -Session $session
```

5. Use Get-PSSession and Invoke-Command to get a list of the 20 most recent Security event log entries from the remote computer.

```
Invoke-Command -ScriptBlock {get-eventlog -LogName System -Newest 20}
-Session (Get-PSSession)
```

6. Use Invoke-Command and your $session variable to load the ServerManager module on the remote computer.

```
Invoke-Command -ScriptBlock {Import-Module ServerManager} -Session $session
```

7. Import the ServerManager module's commands from the remote computer to your computer. Add the prefix "rem" to the imported commands' nouns.

```
Import-PSSession -Session $session -Prefix rem -Module ServerManager
```

8. Run the imported Get-WindowsFeature command.

```
Get-WindowsFeature
```

9. Close the session that's in your $session variable.

```
Remove-PSSession -Session $session
```

Chapter 21

Lab Answers

You Call This Scripting?

The following command is for you to add to a script. You should first identify any elements that should be parameterized, such as the computer name. Your final script should define the parameter, and you should create comment-based help within the script. Run your script to test it, and use the Help command to make sure your comment-based help works properly. Don't forget to read the help files referenced within this chapter for more information.

Here's the command:

```
Get-WmiObject –class Win32_LogicalDisk –computer "localhost" –filter
   "drivetype=3" |
Where { $_.FreeSpace / $_.Size –lt .1 } |
Select –Property DeviceID,FreeSpace,Size
```

Here's a hint: There are at least two pieces of information that will need to be parameterized. This command is intended to list all drives that have less than a given amount of free disk space. Obviously, you won't always want to target localhost, and you might not want 10% (that is, .1) to be your free space threshold. You could also choose to parameterize the drive type (which is 3, here), but for this lab leave that hardcoded with the value 3.

```
<#
.Synopsis
Get drives based on percentage free space
.Description
This command will get all local drives that have less than the specified
percentage of free space available.
.Parameter Computername
The name of the computer to check. The default is localhost.
.Parameter MinimumPercentFree
The minimum percent free diskspace. This is the threshhold. The default
value is 10. Enter a number between 1 and 100.
.Example
PS C:\> Get-Disk -minimum 20
```

```
Find all disks on the local computer with less than 20% free space.
.Example
PS C:\> Get-Disk -comp SERVER02 -minimum 25

Find all local disks on SERVER02 with less than 25% free space.
#>

Param (
$Computername='localhost',
$MinimumPercentFree=10
)

#Convert minimum percent free
$minpercent = $MinimumPercentFree/100

Get-WmiObject -class Win32_LogicalDisk -computername $computername -filter
   "drivetype=3" |
Where { $_.FreeSpace / $_.Size -lt $minpercent } |
Select -Property DeviceID,FreeSpace,Size
```

Chapter 22

Lab Answers

Improving Your Parameterized Script

This lab is going to require you to recall some of what you learned in Chapter 21, because you'll be taking the below command, parameterizing it, and turning it into a script – just like you did for the lab in Chapter 21. However, this time we also want you to make the –computerName parameter mandatory and give it a "hostname" alias. Have your script display verbose output before and after it runs this command, too. Remember, you have to parameterize the computer name – but that's the only thing you have to parameterize in this case. Be sure to run the command as-is before you start modifying it, to make sure it works on your system.

```
get-wmiobject win32_networkadapter -computername localhost |
where { $_.PhysicalAdapter } |
select MACAddress,AdapterType,DeviceID,Name,Speed
```

So to reiterate, here's your complete task list:

- Make sure the command runs as-is before modifying it.
- Parameterize the computer name.
- Make the computer name parameter mandatory.
- Give the computer name parameter an alias, "hostname."
- Add comment-based help with at least one example of how to use the script.
- Add verbose output before and after the modified command.
- Save the script as Get-PhysicalAdapters.ps1.

```
#Get-PhysicalAdapters.ps1

<#
.Synopsis
Get physical network adapters
.Description
Display all physical adapters from the Win32_NetworkAdapter class.
.Parameter Computername
```

```
The name of the computer to check. The default is localhost.
.Example
PS C:\> c:\scripts\Get-PhysicalAdapters -computer SERVER01
#>
[cmdletbinding()]
Param (
[Parameter(Mandatory=$True,HelpMessage=”Enter a computername to query”)]
[alias(‘hostname’)]
[string]$Computername
)

Write-Verbose “Getting physical network adapters from $computername”

Get-Wmiobject -class win32_networkadapter –computername $computername |
 where { $_.PhysicalAdapter } |
 select MACAddress,AdapterType,DeviceID,Name,Speed

Write-Verbose “Script finished.”
```

Chapter 23 Lab Answers

Advanced Remoting Configuration

Create a Remoting endpoint named TestPoint on your local computer. Configure the endpoint so that the SmbShare module is loaded automatically, but so that only the Get-SmbShare cmdlet is visible from that module. Also ensure that key cmdlets like Exit-PSSession are available, but no other core PowerShell cmdlets can be used. Don't worry about specifying special endpoint permissions or designating a "Run As" credential.

Test your endpoint by connecting to it using Enter-PSSession (specify localhost as the computer name, and TestPoint as the configuration name). When connected, run Get-Command to ensure that only the designated handful of commands can be seen.

Note that this lab might only be possible on Windows 8, Windows Server 2012, and later versions of Windows – the SmbShare module didn't ship with earlier versions of Windows.

```
#create the session configuration file in the current location
#this is one long line
New-PSSessionConfigurationFile -Path .\SMBShareEndpoint.pssc -ModulesToImport
   SMBShare -SessionType RestrictedRemoteServer -CompanyName "My Company"
   -Author "Jane Admin" -Description "restricted SMBShare endpoint"
   -PowerShellVersion '3.0'

#register the configuration
Register-PSSessionConfiguration -Path .\SMBShareEndpoint.pssc -Name TestPoint

#enter the restricted endpoint

Enter-PSSession -ComputerName localhost -ConfigurationName TestPoint
get-command
exit-pssession
```

Chapter 24 Lab Answers

Using Regular Expressions to Parse Text Files

Make no mistakes about it, regular expressions can make your head spin so don't try to create complex regex's right off the bat. Start simple and here are a few exercises to ease you into it. Use regular expressions and operators to complete the following:

1. Get all files in your Windows directory that have a 2 digit number as part of the name.

```
dir c:\windows | where {$_.name -match "\d{2}"}
```

2. Find all processes running on your computer that are from Microsoft and display the process ID, name and company name. Hint: Pipe Get-Process to Get-Member to discover property names.

```
get-process | where {$_.company -match "^Microsoft"} | Select Name,ID,Company
```

3. In the Windows Update log, usually found in C:\Windows, you want to display only the lines where the agent began installing files. You may need to open the file in Notepad to figure out what string you need to select.

```
get-content .\WindowsUpdate.log | Select-string "Start[\w+\W+]+Agent: Installing
   Updates"
```

Chapter 26 Lab Answers

Using Someone Else's Script

Below is a complete script. See if you can figure out what it does, and how to use it. Can you predict any errors that this might cause? What might you need to do in order to use this in your environment? Note that this script should run as-is… but, if it doesn't on your system, do you think you can track down the cause of the problem? Keep in mind that you've seen most of these commands – and for the ones you haven't there are the PowerShell help files. Those files' examples include every technique shown in this script.

```
function get-LastOn {
<#
.DESCRIPTION
Tell me the most recent event log entries for logon or logoff.
.BUGS
Blank ‘computer’ column

.EXAMPLE
get-LastOn -computername server1 | Sort-Object time -Descending |
Sort-Object id -unique | format-table -AutoSize -Wrap
ID              Domain       Computer Time
--              ------       -------- ----
LOCAL SERVICE   NT AUTHORITY          4/3/2012 11:16:39 AM
NETWORK SERVICE NT AUTHORITY          4/3/2012 11:16:39 AM
SYSTEM          NT AUTHORITY          4/3/2012 11:16:02 AM

Sorting -unique will ensure only one line per user ID, the most recent.
Needs more testing

.EXAMPLE
PS C:\Users\administrator> get-LastOn -computername server1 -newest 10000
 -maxIDs 10000 | Sort-Object time -Descending |

 Sort-Object id -unique | format-table -AutoSize -Wrap
```

```
ID              Domain       Computer Time
--              ------       -------- ----
Administrator   USS                   4/11/2012 10:44:57 PM
ANONYMOUS LOGON NT AUTHORITY          4/3/2012 8:19:07 AM
LOCAL SERVICE   NT AUTHORITY          10/19/2011 10:17:22 AM
NETWORK SERVICE NT AUTHORITY          4/4/2012 8:24:09 AM
student         WIN7                  4/11/2012 4:16:55 PM
SYSTEM          NT AUTHORITY          10/18/2011 7:53:56 PM
USSDC$          USS                   4/11/2012 9:38:05 AM
WIN7$           USS                   10/19/2011 3:25:30 AM


PS C:\Users\administrator>

.EXAMPLE
get-LastOn -newest 1000 -maxIDs 20
Only examines the last 1000 lines of the event log

.EXAMPLE
get-LastOn -computername server1| Sort-Object time -Descending |
Sort-Object id -unique | format-table -AutoSize -Wrap
#>

param (
        [string]$ComputerName = 'localhost',
        [int]$Newest = 5000,
        [int]$maxIDs = 5,
        [int]$logonEventNum = 4624,
        [int]$logoffEventNum = 4647
    )

    $eventsAndIDs = Get-EventLog -LogName security -Newest $Newest |
    Where-Object {$_.instanceid -eq $logonEventNum -or
[CA]$_.instanceid -eq  $logoffEventNum} |
    Select-Object -Last $maxIDs
[CA]-Property TimeGenerated,Message,ComputerName

    foreach ($event in $eventsAndIDs) {
        $id = ($event |
        parseEventLogMessage |
        where-Object {$_.fieldName -eq "Account Name"}  |
        Select-Object -last 1).fieldValue

        $domain = ($event |
        parseEventLogMessage |
        where-Object {$_.fieldName -eq “Account Domain”}  |
        Select-Object -last 1).fieldValue

        $props = @{‘Time’=$event.TimeGenerated;
            ‘Computer’=$ComputerName;
            ‘ID’=$id
            ‘Domain’=$domain}

        $output_obj = New-Object -TypeName PSObject -Property $props
        write-output $output_obj
    }
}

function parseEventLogMessage()
{
    [CmdletBinding()]
    param (
        [parameter(ValueFromPipeline=$True,Mandatory=$True)]
        [string]$Message
    )

    $eachLineArray = $Message -split “`n”

    foreach ($oneLine in $eachLineArray) {
        write-verbose “line:_$oneLine_”
        $fieldName,$fieldValue = $oneLine -split “:”, 2
            try {
                $fieldName = $fieldName.trim()
                $fieldValue = $fieldValue.trim()
            }
            catch {
                $fieldName = “”
            }

            if ($fieldName -ne “” -and $fieldValue -ne “” )
            {
            $props = @{‘fieldName’=”$fieldName”;
                    ‘fieldValue’=$fieldValue}

            $output_obj = New-Object -TypeName PSObject -Property $props
            Write-Output $output_obj
            }
    }
}
Get-LastOn
```

The script file seems to define 2 functions which won't do anything until called. At the end of the script is a command, Get-LastOn, which is the same name as one of the functions so we can assume that is what is executed. Looking at that function it has a number of parameter defaults which explains why nothing else needs to be called. The comment based help also explains what the function does. The first part of this function is using Get-Eventlog.

```
$eventsAndIDs = Get-EventLog -LogName security -Newest $Newest
   |     Where-Object {$_.instanceid -eq $logonEventNum -or $_.instanceid -eq
   $logoffEventNum} |
    Select-Object -Last $maxIDs -Property TimeGenerated,Message,ComputerName
```

If this was a new cmdlet, we would look at help and examples. The expression seems to be getting the newest security eventlogs. $Newest comes from a parameter and has a default value of 5000. These eventlogs are then filtered by Where-Object looking for two different event log values, also from the parameter.

Next it looks like something is done with each event log in the foreach loop. Here's a potential pitfall: if the eventlog doesn't have any matching errors the code in this loop will likely fail unless it has some good error handling.

In the foreach loop it looks like some other variables are getting set. The first one is taking the event object and piping it to something called parseEventmessage. This doesn't look like a cmdlet name but we did see it as one of the functions. Jumping to it, we can see that it takes a message as a parameter and splits each one into an array. We might need to research the –Split operator.

Each line in the array is processed by another ForEach loop. It looks like lines are split again and there is a Try/Catch block to handle errors. Again, we might need to read-up on that to see how it works. Finally there is an IF statement where it appears that if the split up strings are not empty, then a variable called $props is created as a hash table or associative array. This function would be much easier to decipher if the author had included some comments. Anyway, the parsing function ends by calling New-Object, another cmdlet to read up on.

This function's output is then passed to the calling function. It looks like the same process is repeated to get $domain.

Oh, look another hash table and New-Object but by now we should understand what the function is doing. This is the final output from the function and hence the script.

Appendix A Lab Answers

Review Labs

Review Lab 1

Task 1
```
Get-EventLog –LogName Security –Newest 100
```

Task 2
```
Get-Process | Sort –Property VM -Descending | Select –First 5
```

Task 3
```
Get-Service | Select –Property Name,Status | Sort –Property Status –Descending
   | Export-CSV services.csv
```

Task 4
```
Set-Service –Name BITS –StartupType Automatic
```

Task 5
```
Get-ChildItem –Path C:\ -Recurse –Filter ‘Win*.*’
```

Task 6
```
Get-ChildItem –Path ‘c:\program files’ –recurse | Out-File c:\dir.txt
```

Task 7
```
Get-EventLog –LogName Security –Newest 20 | ConvertTo-XML
```

Task 8
```
Get-Service | Export-CSV C:\services.csv
```

Task 9
```
Get-Service | Select –Property Name,DisplayName,Status | ConvertTo-HTML –Pre-
   Content “Installed Services” | Out-File c:\services.html
```

Task 10
```
New-Alias –Name D –Value Get-ChildItem –PassThru | Export-Alias c:\alias.xml
After opening a new PowerShell window…
Import-Alias c:\alias.xml
PS> D
```

Task 11

```
Get-EventLog –List
```

Task 12

```
Get-Location
```

Task 13

```
Get-History
```

After running this, locate the command that you ran for Task 11. You will need its ID number, which you will put in place of x in the next command:

```
Get-History –id x | Invoke-History
```

Task 14

```
Limit-EventLog –LogName Security –OverwriteAction OverwriteAsNeeded
```

Task 15

```
New-Item –Name C:\Review –Type Directory
```

Task 16

```
Get-ItemProperty -Path 'HKCU:\Software\Microsoft\Windows\CurrentVersion\Explor-
   er\User Shell Folders'
```

Task 17

```
Restart-Computer
Stop-Computer
Remove-Computer
Restore-Computer
```

Task 18

```
Set-ItemProperty
```

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 }
```

Review Lab 3

```
Start-Job
Invoke-Command
Yes
Read-Host
Write-Output
```

Task 1

```
Get-Process | Select-Object -property Name,ID,VM,PM |
ConvertTo-HTML -Title “Current Processes” | Out-File C:\Procs.html
```

Task 2

```
Get-Service | Export-CSV c:\services.tdf -Delimiter “`t”
```

Task 3

```
Get-Process | Select-Object -property Name,ID,VM,PM |
Select-Object -Property Name,ID,@{n=’VM’;e={$_.VM / 1GB -as [int]}},
  @{n=’PM’;e={$_.PM / 1GB -as [int]}} |
ConvertTo-HTML -Title “Current Processes” | Out-File C:\Procs.html
```