

Lab Sample Solutions

Chapter 3

- 1 First, run `Update-Help` and ensure it completes without errors. That will get a copy of the help on your local computer. You'll need an internet connection, and the shell needs to run under elevated privileges (which means it must say "Administrator" in the shell window's 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 only 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
```

This should show you 80 characters as the default for the PowerShell console. You would use this parameter to change it as well.

- 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?

If you run `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*
```

Chapter 4

Using 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 you can run `d` to get a directory listing.

```
New-Alias -Name list -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 the asterisk (*) 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. You can use the same cmdlet as in the previous task, but you'll need to read its help to discover the necessary parameter and its allowable values.

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

Chapter 5

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 -Path C:\
```

- 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 filesystem 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 PSPProvider's filter capability, which not all providers support. For example, you could use `DIR -filter` in the filesystem but not in the registry. But you could use `DIR -include` in the registry to achieve almost the same type of filtering result.

Chapter 6

- 1 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 filename 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`.

- 2 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 parameter values for the `-Name` parameter. For example, you could run this:

```
Stop-Service spooler
```

- 3 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 "|"
```

- 4 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 want to omit that from a particular file?

Add the `-NoTypeInfo` switch to `Export-CSV`.

- 5 `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
```

- 6 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

As always, we're assuming that you have the latest version of Windows (client or server) on a computer or virtual machine to test with.

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." Hit Enter, run a Web Connectivity check, and ask for help connecting to a specific web page. Use `http://videotraining.interfacett.com` as your test URL. We hope 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 another 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

This chapter has probably covered more, and more difficult, new concepts than any chapter to this point. We hope we were able to make sense of it all, but these exercises will help you cement what you've learned. See if you can complete all of the exercises, and remember to supplement your learning with the companion videos and sample solutions at MoreLunches.com. Some of these tasks will draw on skills you've learned in previous chapters, to refresh your memory and keep 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 the following (caution: the output will right-align, so make sure your PowerShell window doesn't have a horizontal scroll bar):

```
DayOfWeek
-----
        Monday
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. Remember that the column headers shown in a command's default output aren't necessarily the real property names—you'll need to look up the real property names to be sure.

```
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 with the oldest entries appearing first, and with entries made at the same time sorted by their index. Display the index, time, and source for each entry. Put this information into a text file (not an HTML file, but 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; a better cmdlet is available for this particular task.

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

Chapter 9

Once again, we've covered a lot of important concepts in a short amount of time. The best way to cement your new knowledge is to put it to immediate use. We recommend doing the

following tasks in order, because they build on each other to help remind you what you've learned and to help you find practical ways to use that knowledge.

To make this a bit trickier, we're going to force you to consider the `Get-ADComputer` command. Any Windows Server 2008 R2 or later domain controller has this command installed, but you don't need one. You only need to know three 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 that contains the computer's host name.
- Domain computer objects have the `TypeName` `ADComputer`, which means `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)
```

This should work because the nested `Get-ADComputer` expression will return a collection of computer names, 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. 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 we 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.

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

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

- 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)
```

- 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

See if you can complete the following tasks:

- 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
```

- 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
```

- 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

Remember that `Where-Object` isn’t the only way to filter, and it isn’t even the one you should turn to first. We’ve kept this chapter brief to allow you more time to work on the hands-on examples, so keeping in mind 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? Answer “Yes” or “No” to this question. You don’t need to write a command to accomplish this.

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 OK.

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

- 6 Display a list of all processes running with either the name “Conhost” or the name “Svchost”.

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

Chapter 12

Okay, now it’s your turn. We’re assuming that you’re working in a virtual machine or other machine that it’s okay to mess up a little in the name of learning. Please don’t do this in a production environment on a mission-critical computer!

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 make sure that the share permissions are set so that Everyone has read/write access, and local Administrators have full control. Because the share will be primarily for files, you want to set the share’s caching mode for documents. Your script should output an object showing 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

It’s time to combine some of what you’ve learned about remoting with what you’ve learned in previous chapters. See if you can accomplish the following tasks:

- 1 Make a one-to-one connection with a remote computer (or with “localhost” if you only have one 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. An alternative command to launch it would be `Start-Process Notepad`.

- Using `Invoke-Command`, retrieve a list of services that aren't started from one or two remote computers (it's OK to use "localhost" twice if you only have one computer). Format the results as a wide list. (Hint: it's OK to retrieve results and have the formatting occur on your computer—don't include the `Format-Command` cmdlets in the commands that are invoked remotely).

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

- 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; if you only have one computer, target "localhost" twice.

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

- Create a text file that contains three computer names, with one name per line. It's OK to use the same computer name three times, or "localhost," 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

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). Keep in mind that PowerShell's help system can't help you find WMI classes.

- What class can you use to view the current IP address of a network adapter? Does the class have any methods that you could use 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"
```

- 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 use 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
```

The output is the same as that of the Get-Hotfix cmdlet; that cmdlet merely provides a wrapper around this WMI class.

- 4 Display a list of services, including their current statuses, their start modes, and the accounts 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
```

In fact this list is probably not complete for all computers. It only lists software installed through the Windows Installer mechanism in Windows and requires that Windows first revalidate all installed packages, which can be time-consuming.

Chapter 15

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's 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 task 1 on a group of remote computers?

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

- 3 Create 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 6:00 a.m. 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 would 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

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

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

Find the methods like this,

```
get-service | Get-Member -MemberType Method
```

and you should see a Pause() method.

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

Find the methods like this,

```
get-process | Get-Member -MemberType Method
```

and you should see a Kill() method. You could verify by checking the MSDN documentation for this process object type. You shouldn't need to invoke the method, because there is a cmdlet, Stop-Process, that will do the work for you.

- 7 What method of a WMI Win32_Process object would terminate a given process?

You could search the MSDN documentation for the Win32_Process class. Or you might use the CIM cmdlets, because they also work with WMI to list all of the possible methods.

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

In either event, you should see the Terminate() method.

- 8 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 18

Flip back to chapter 15 and refresh your memory on working with background jobs. Then, at the command line, do the following:

- 1 Create a background job that queries the Win32_BIOS information from two computers (use “localhost” twice if you only have one computer to experiment with).

```

invoke-command {get-wmiobject win32_bios} -comp localhost,$env:computername
-asjob

```

- 2 When the job finishes running, receive the results of the job into a variable.

```

$results=Receive-Job 4 -keep

```

- 3 Display the contents of that variable.

```

$results

```

- 4 Export the variable’s contents to a CliXML file.

```

$results | export-clixml bios.xml

```

Chapter 19

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 stuck, it’s OK to peek at the sample answers available on MoreLunches.com.

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

```

write-output (100*10)

```

Or simply type the formula: 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

```

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

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

Chapter 20

To complete this lab, you'll 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 a similar experience that way.

- Close all open sessions in your shell.

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

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

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

- 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
```

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

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

- 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)
```

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

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

- 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
```

- Run the imported `Get-WindowsFeature` command.

```
Get-RemWindowsFeature
```

- Close the session that's in your `$session` variable.

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

Chapter 21

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 Win32_LogicalDisk -comp "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 threshold. 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

This lab is going to require you to recall some of what you learned in chapter 21, because you'll be taking the following command, parameterizing it, and turning it into a script—just like you did for the lab in chapter 21. But 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
```

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 24

Make no mistake about it, regular expressions can make your head spin so don't try to create complex regexes right off the bat—start simple. 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 two-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

Listing 26.2 shows 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.

Listing 26.2 Get-LastOn.ps1

```
function get-LastOn {
<#
.DESCRIPTION
Tell me the most recent event log entries for logon or logoff.
.BUGS
Blank 'computer' column
.EXAMPLE
get-LastOn -computersname 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
➤ $_.instanceid -eq $logoffEventNum} |
Select-Object -Last $maxIDs
➤ -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 two functions that 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 you can assume that's what is executed.

The `Get-LastOn` function 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, you could look at help and examples. The expression seems to be getting the newest security event logs. `$Newest` comes from a parameter and has a default value of 5000. These event logs are then filtered by `Where-Object`, which is 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 event log 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 you should have seen it as one of the functions. If you look at it, you can see that it takes a message as a parameter and splits each one into an array. You 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, you 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, a variable called `$props` is created as a hashtable 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 hashtable and `New-Object`, but by now you should understand what the function is doing. This is the final output from the function and hence the script.