CSC 360 Lab Assignment #6
Spring 2015 Due: March 13, 2015

The following are the end-of-chapter Labs from Chapter 7 through Chapter 12 of our textbook. Answer each question or describe the PowerShell command(s) necessary to accomplish the task described.

Chapter 7: *Adding commands*

NOTE: For this lab, you'll need a Windows 7, Windows Server 2008 R2, or later computer running PowerShell v3.

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!

Chapter 8: *Objects: data by another name*

NOTE: For this lab, you'll need any computer running PowerShell v3.

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.

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 the following (caution: the output will right-align, so make sure your PowerShell window doesn't have a horizontal scroll bar):

```
DayOfWeek
---------
   Monday
```

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. 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.

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; a better cmdlet is available for this particular task.

Chapter 9: *The pipeline, deeper*

NOTE: For this lab, you'll need any computer running PowerShell v3.

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 I 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 I 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 Active Directory (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 computer in the domain? Write out an explanation, similar to the ones I provided earlier in this chapter.

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

Chapter 10: *Formatting—and why it's done on the right*

NOTE: For this lab, you'll need any computer running PowerShell v3.

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).

Chapter11: *Filtering and comparisons*

NOTE: For this lab, you'll need a Windows 8 or Windows Server 2012 computer running PowerShell v3.

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 versions of Windows, both client and server). Using the `Get-NetAdapter` cmdlet, display a list of non-virtual adapters (that is, adapters whose `Virtual` property is `False`, which PowerShell represents with the special value `$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? Answer "Yes" or "No" to this question. You don't need to write a command to accomplish this.

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.

6. Display a list of all processes running with the name "Conhost" or the name "Svchost".

Chapter 12: *A practical interlude*

NOTE: For this lab, you'll need a Windows 8 or Windows Server 2012 computer running PowerShell v3.

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.