3

Labs

Windows PowerShell 3

Day 3

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
```

Review Lab 3 (Based on Chapters 1-19

```
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.