Windows PowerShell 4.0 For the IT Professional - Part 1

Module 3: Pipeline 1

Student Lab Manual

Version 2.0

Conditions and Terms of Use

Microsoft Confidential - For Internal Use Only

Contents

Lab 3: The Pipeline

Introduction

You can use pipelines to send the objects emitted by one command, as input to another command for processing. The result is a very powerful command chain or "pipeline" that is comprised of a series of simple commands.

Objectives

After completing this lab, you will be able to:

- Compose Cmdlets into sequences to solve simple administrative problems
- Understand pipeline behavior and syntax.
- Analyze data to extract frequency trends
- Sort, group, manipulate and convert data

Prerequisites

Start all VMs provided for the workshop labs.

Logon to WIN8-WS:

Username: **Contoso\Administrator**

Password: **PowerShell4**

Estimated time to complete this lab

75 minutes

NOTE: These exercises use many Windows PowerShell commands. You can type these commands into the Windows PowerShell Integrated Scripting Environment (ISE) or the Windows PowerShell console. For some exercises, you can load pre-typed lab files into the Windows PowerShell ISE, allowing you to select and execute individual commands. Each lab has its own folder under **C:\PShell\Labs** on **WIN8-WS**.

Some exercises in this workshop may require running the Windows PowerShell console or ISE as an elevated user (Run as Administrator).

We recommend that you connect to the virtual machines (VMs) for these labs through Remote Desktop rather than connecting through the Hyper-V console. This allows you to use copy and paste between VMs and the host machine. If you are using the online hosted labs, then you are already using Remote Desktop.

Exercise 3.1: Pipeline Input

Introduction

Windows PowerShell Cmdlets are easy to discover, use and obtain help. Their real power is unleashed when two or more Cmdlets are connected together using the pipeline operator (|).

Objectives

After completing this exercise, you will be able to:

- Identify the pipeline character
- Provide pipeline input
- Process and filter pipeline items to produce desired output

Task 3.1.1: Get-* Cmdlets

There are a number of ways to provide pipeline input. The Get-* Cmdlets are designed for this purpose.

1. Right-click the PowerShell icon on the taskbar and select the "Run ISE as Administrator" option. This will launch the Windows PowerShell Integrated Scripting Environment.
2. List all the Cmdlets, with the "Get" verb, using Get-Command.

```
Get-Command –verb Get
```

3. Try counting the total number of Cmdlets returned… It may take a while!
4. A faster and more accurate way to count the number of Get-* Cmdlets is to construct a pipeline.
   Using Get-Command, find a Cmdlet with the verb "Measure" and list the Cmdlet names returned below.

   1. Measure-Command
   2. Measure-Object

5. Type the following command to return the total number of "Get" Cmdlets.

```
Get-Command –verb Get | Measure-Object
```

6. Write below the number of "Get" Cmdlets found:

_____

TIP: Press the <TAB> key to autocomplete the following items:

a) **Cmdlet names**: Type a cmdlet "Verb" followed by a hyphen character (-) then press <TAB> repeatedly to automatically complete and cycle through Cmdlet names. (Use <SHIFT> + <TAB> to cycle in reverse order)

b) **Parameter names**: Type a <SPACE> and a hyphen after a completed Cmdlet name.

c) **Parameter values**: Many Cmdlet parameter values will also autocomplete.

Windows PowerShell console: <SPACE>, then <TAB>

Windows PowerShell ISE: <SPACE>

e.g. PS C:\> Get-Service –Name<SPACE><TAB>

7. Construct pipeline commands to answer the following questions using the knowledge gained from the previous steps.

   a) What are the total number of commands, of type '**Cmdlet',** in your current Windows PowerShell session?

   Get-Command -CommandType Cmdlet | Measure-Object

   Get-Command -CommandType Cmdlet | Measure-Object

   _____

   b) What are the total number of commands, of type '**Application',** in your current Windows PowerShell session?

   Get-Command -CommandType Application | Measure-Object

   _____

   c) What are the total number of commands, of type '**Function',** in your current Windows PowerShell session?

   Get-Command -CommandType Function | Measure-Object

   _____

8. List all the files in C:\Windows\System32. Note the use of the –File switch parameter to only return files.

```
Get-ChildItem –Path C:\Windows\System32 -File
```

9. Using the pipeline, count how many files and folders are in this directory and write it below.

   Get-ChildItem -Path C:\Windows -File | Measure-Object

10. Get help for the Measure-Object Cmdlet and determine which parameters are required to return the **sum** of all the files' **length** properties. Write the command to obtain the total size of all files in the C:\Windows\System32 folder.

    Write the command used to obtain the sum and record the result below.

    Command:

    Sum:

Get-ChildItem C:\Windows -File | Measure-Object -Property length -sum

Task 3.1.2: File & external command Input

1. The Windows PowerShell pipeline can also process text files and text returned by external Windows commands. The Get-Content Cmdlet reads a text file line by line and has the ability to work with remote files.

   Type and execute the following command.

```
Get-Content '\\2012R2-DC\C$\Windows\debug\DCPROMO.LOG'
```

2. What pipeline command would you use to measure the number of lines in the file? Construct and execute the command then write it below.

```
Get-Content '\\2012R2-DC\C$\Windows\debug\DCPROMO.LOG' | Measure-Object
```

3. Get help for the Measure-Object Cmdlet (or cycle through the Cmdlet's available parameters using the <TAB> key).

   How would you count the number of Lines, Words and Characters in the file? Execute the command and write it below.

```
'\\2012R2-DC\C$\Windows\debug\DCPROMO.LOG' | Measure-Object -Line -Word -Character
```

4. Another common administrative task is searching within text files for keywords. Windows PowerShell provides the Select-String Cmdlet for this purpose.

   Search the DCPROMO.LOG file for all instances of the string "dsrole", using the command below.

```
Get-Content '\\2012R2-DC\C$\Windows\debug\DCPROMO.LOG' | Select-String –Pattern "dsrole"
```

5. How could you extend the pipeline to measure how many matches were returned. Write the command to do this below.

```
Get-Content '\\2012R2-DC\C$\Windows\debug\DCPROMO.LOG' | Select-String –Pattern "dsrole" | Measure-Object
```

6. Select-String is also able to parse external command output. Type and execute the following.

```
ipconfig /all | Select-String -Pattern "Ipv4 Address"
```

7. Get help for the Select-String Cmdlet.

   How would you search for multiple patterns, such as “Ipv4 Address” and “Subnet Mask”, using the same command? Write the command below.

```
ipconfig /all | Select-String -Pattern "Ipv4 Address","Subnet Mask"
```

Exercise 3.2: Sorting & Selecting

Introduction

Once data has been acquired using the Get-* Cmdlets, one of the many operations that can be performed is to sort the data. Windows PowerShell is aware of the type of data it is processing, and sorts it appropriately. For example, Strings sort alphabetically and numbers numerically – even dates and times sort in the correct date or time order.

Task 3.2.1: Sorting by column headers

1. Type the following command in the Windows PowerShell ISE command pane (the blue pane) and press <ENTER>.

```
Get-ChildItem –Path C:\Windows
```

2. Scroll up to the top of the list and write the column header names below.

1.
2.
3.
4.

Commented [A12]: 1.Mode
2.LastWriteTime
3.Length
4.Name

3. Sort data using the column headers returned by a Cmdlet. Complete the following pipeline for each column header listed above. (Substituting <column name> with each column header).

> **NOTE:** Files and folders sort correctly, in ascending order, according to the column's data type (number, string, date).

```
Get-ChildItem –Path C:\Windows | Sort-Object –Property <column name>
```

4. The following command uses Get help for the Sort-Object Cmdlet's -Descending parameter.

```
Get-Help Sort-Object -Parameter descending
```

5. Re-type the commands for step 3, reversing the sort order.
6. Type the following commands:

   Note the procedure of first executing the Cmdlet, without any arguments, in order to view the default column headers. The headers are used as values for the Sort-Object Cmdlet's –Property parameter.

```
Get-SmbShare
Get-SmbShare | Sort-Object –Property Description
Get-Volume
```

```
Get-Volume | Sort-Object –Property SizeRemaining
Get-Hotfix
Get-Hotfix | Sort-Object –Property installedon
```

Task 3.2.2: Selecting properties and lists

The Select-Object Cmdlet has many uses. Selecting properties, displaying the first or last items in a list or filtering out duplicate entries.

1. Find and read the help for the Select-Object Cmdlet, then type and execute the commands below.

   a) List the DisplayName and Status of all services

```
Get-Service | Select-Object –Property DisplayName, Status
```

   b) List the processes with the most open resource handles. Sort the processes in descending order and return the first five processes with the highest number of open handles.

```
Get-Process | Sort-Object handles -Descending | Select-Object -First 5
```

   c) Return the first 10 errors from the System event log. The Get-Eventlog Cmdlet returns entries in date order by default therefore sorting is not required.

```
Get-EventLog -Logname System –EntryType Error | Select-Object -First 10
```

   d) Note the duplicate process names in the output of the Get-Process Cmdlet. Remove duplicate process names with Select-Object's –Unique switch parameter.

```
Get-Process | Select-Object –Property Name
Get-Process | Select-Object –Property Name -Unique
```

2. Construct pipeline commands to return the following information. Ensure the commands return the correct information, and then record them below each task.

   a) List the 10 processes that have the most open Handles

```
Get-Process | Sort-Object –Property Handles -Descending | Select-Object -First 10
```

   b) List the 5 most recently modified files in C:\Windows, based upon the time they were last written.

```
Get-ChildItem -Path C:\Windows | Sort-Object -Property LastWriteTime -Descending | Select-Object -First 5
```

   c) List the 10 oldest events in the Application log, with entry type 'Warning'. Use the <Tab> key to find the "TimeGenerated" property and sort the entries in descending order.

```
Get-EventLog -Logname Application -EntryType Warning -Newest 10
```

```
Get-EventLog -LogName Application | Sort-Object -Property <TAB> | ….
```

Exercise 3.3: Formatting & Grouping

Introduction

Windows PowerShell includes a number of formatting Cmdlets to aid displaying data in a convenient way. The default display format for a Cmdlet is a table output, which can be easily modified, as this may not be optimal when viewing some types of data.

> **NOTE:** The formatting Cmdlets should be the final stage in a pipeline operation (the right-hand most command) as they completely re-structure the data.

Task 3.3.1: Displaying data in a table format

1. Format-Table uses the –Property parameter to select the displayed columns. Type the example below.

```
Get-Service | Format-Table -property Name, DependentServices
```

   Modify the command above to display the DisplayName and Status columns and write the command below.

```
Get-Service | Format-Table -property DisplayName,Status
```

2. Revise the command so that it automatically resizes the output to fit the largest piece of data per column. What change to the command will make this possible?

   -AutoSize switch parameter

3. Format-Table has a number of useful parameters. The –GroupBy parameter will group results into separate tables. Sorting the data is required prior to grouping a property. Type and run the commands below.

```
Get-Service | Sort-Object –Property DependentServices |
Format-Table -Property Name, DependentServices -GroupBy DependentServices

Get-Process | Sort-Object -Property SessionId | Format-Table -GroupBy SessionId

Get-Eventlog -LogName System -Newest 100 | Sort-Object -Property EntryType |
Format-Table -GroupBy EntryType
```

4. Using Sort-Object and the Format-Table Cmdlet's –GroupBy parameter, write commands to solve the following problems and write the solutions below.

a. Get files and folders in C:\Windows, sort and group by the Attributes property, and then display the Attributes, LastWriteTime, Length and Name columns.

```
Get-ChildItem C:\Windows | Sort-Object –Property attributes | Format-Table -GroupBy attributes -Property attributes,lastwritetime,length,name
```

Task 3.3.2: Displaying data in a list format

Sometimes a table does not provide sufficient space to present the data in which we are interested. A list format can usually solve this problem. Windows PowerShell provides the Format-List Cmdlet for this purpose.

> **NOTE:** All commands executed so far have returned output in a table format with a default set of column headers.
>
> Each Get-* Cmdlet actually returns objects containing many more "columns" than the default view presents to us (we will now refer to the "columns" as "properties"). Format-List is an efficient way to display all of this "hidden" data.

1. Type the following command.

```
Get-Process
```

How many columns can you count? Write their names below.

2. Type the following command. The wildcard character (*) forces Windows PowerShell to display all properties for each process.

```
Get-Process | Format-List –Property *
```

How many more properties can you see for each process?

3. Format-List also has a –GroupBy parameter. Use this parameter in a pipeline command that displays the process Name and Id properties and groups them by BasePriority.

```
Get-process | Sort-Object –Property basepriority | Format-List -Property Name,Id -GroupBy basepriority
```

Task 3.3.3: Calculating frequency data

Windows PowerShell also has a dedicated Cmdlet to group objects. This Cmdlet calculates frequency data for a particular property.

1. Search for this Cmdlet using Get-Command and write the Cmdlet name below.

   Group-Object

2. The following command groups the latest 1000 eventlog entries by EntryType (error, warning, information).

```
Get-Eventlog -LogName system -newest 1000 | Group-Object EntryType
```

3. In the output from the previous command, which property records how often the event type occurred?

   count

4. Using the same Group-Object Cmdlet, get all Services and group them by status. Then record the number of services that have a status of running and the number that have a status of stopped?

   # Running:

   # Stopped:

   Command:

```
Get-Service | Group-Object status
```

5. Use a similar pipeline command to get the files from C:\Windows\System32, group them by file extension. Sort the result by the Count property in descending order and select the first five results. Write the command below.

```
Get-ChildItem C:\Windows\System32 -file |
Group-Object -Property extension | Sort-Object
-Property Count -Descending
 | Select-Object -first 5
```

6. What was (unsurprisingly!) the most common file extension in C:\Windows\System32?

Exercise 3.4: Pipeline Output

Introduction

Windows PowerShell has a number of Cmdlets to write pipeline processed data into files and convert it to other structured file formats, such as html, csv and xml.

Task 3.4.1: Export to a text file

1. Sending pipeline output to a text file is a common operation. Type the following external command and review its output.

```
driverquery
```

2. We can extend this to filter the command's text stream for a match, using the Select-String Cmdlet. Type the command below.

```
driverquery | Select-string –Pattern "File System"
```

3. Write the resulting matches to a text file using the Out-File Cmdlet. Type the command below then open and review the file. C:\Pshell\Labs\Lab_3\drivers.txt.

```
driverquery |
Select-string –Pattern "File System" |
Out-File –FilePath C:\PShell\Labs\Lab_3\drivers.txt
```

4. Although we saved the data in a text file, we can perform further pipeline processing. Construct a single pipeline command to perform the following tasks.

   a) Get the content from the C:\PShell\Labs\Lab_3\drivers.txt file
   b) Select the string "SMB".
   c) Save the results out to the file C:\PShell\Labs\Lab_3\SMB_Drivers.txt

   Write the completed command below.

Get-Content –Path C:\PShell\Labs\Lab_3\drivers.txt | Select-String –Pattern "SMB" | Out-File –FilePath C:\PShell\Labs\Lab_3\SMB_Drivers.txt

Task 3.4.2: Exporting and Importing CSV information

The previous task performed pipeline operations on lines of text and saved the output in unstructured text files. A much better way to persist information from pipeline operations is to write a structured text file using character separated values (CSV). This provides the ability to then import the structured text file and filter the contents.

1. The external driverquery.exe command has the /FO (file option) parameter, which can accept the 'csv' parameter value. This provides the desired structured text output. Type and execute the following command:

```
driverquery /FO csv | Out-File -FilePath C:\PShell\Labs\Lab_3\drivers.csv
```

2. Next, we will import the content of the C:\PShell\Labs\Lab_3\drivers.csv file.

```
Import-CSV –Path C:\PShell\Labs\Lab_3\drivers.csv
```

3. List the columns names shown at the top of the table, below.

   1.
   2.
   3.
   4.

   Module Name, display name, driver type, link date

4. As we have seen, a structured text file can be imported using the Import-CSV Cmdlet. Manipulate the data using a pipeline of Cmdlets. Type the command below.

```
Import-CSV –Path C:\PShell\Labs\Lab_3\drivers.csv |
Sort-Object -Property "Display Name"
```

5. Modify the above command to group the results by the "Driver Type" column. Write the modified command below and record the total number of Kernel and File System drivers.

   Command:

   ```
   Import-CSV –Path C:\PShell\Labs\Lab_3\drivers.csv | Group-Object -Property "Driver Type"
   ```

   # Kernel: 223

   # File System: 27

6. Converting data emitted by the Get-* Cmdlets into structured csv text format is a convenient way to save the command's output and import it later.

   Get a list of services and export them, using the Export-CSV Cmdlet, to C:\Pshell\Labs\Lab_3\services.csv. Write the command you used to do this below.

   ```
   Get-Service | Export-Csv -Path C:\PShell\Labs\Lab_3\services.csv
   ```

7. Import the csv file produced in the previous step and create a pipeline command to group the data by ServiceType. What is the most common service type?

```
Import-Csv –Path C:\PShell\Labs\Lab_3\services.csv | group-Object –Property ServiceType
```

8. Construct a pipeline to get processes and export them to a csv file named C:\PShell\Labs\Lab_3\processes.csv

```
Get-Process | Export-csv -Path C:\PShell\Labs\Lab_3\processes.csv
```

9. Import C:\PShell\ Labs\Lab_3\processes.csv, sort by PeakWorkingSet, in descending order and display the Name, Handles, Cpu and PeakWorkingSet columns. Write the command below.

```
import-csv -Path C:\PShell\Labs\Lab_3\processes.csv | Get-Process | Sort-Object -Property peakworkingset -Descending | Format-Table name,handles,cpu,peakworkingset
```

10. During file import the Import-Csv Cmdlet is able to add a header line to a CSV file that does not have one. Get help for Import-Csv and find out how to do this.

```
Get-Help Import-Csv -Parameter Header
```

11. Import the file C:\PShell\Labs\Lab_3\UserNames.txt, adding the headers "FirstName" and "LastName" to the data during import. Write the command you used below.

12. Using the Cmdlets Import-CSV, Sort-Object and Group-Object, find the most common LastName in the file. Write the command below

```
Import-CSV -Path C:\Pshell\Labs\Lab_3\UserNames.txt -Header "Firstname","Lastname" | Group-Object -Property Lastname | Sort-Object -Property Count -Descending | Select-Object -First 1
```

Task 3.4.3: Importing and Exporting as XML

Extensible Markup Language (XML) is a common structured data storage format that lends itself well to describing the type of data Windows PowerShell Cmdlets

exchange. XML allows a Cmdlet's output to be stored in a text file for later analysis.

1. Get all running processes and export them to a file, in XML format.

```
Get-Process | Export-Clixml –Path C:\PShell\Labs\Lab_3\processes.xml
```

2. Use Get-Command to find the Cmdlet needed to Import the XML file. Write the command name below.

Import-Clixml

3. Import the XML file and use the pipeline to group the processes by BasePriority. Write the command to achieve this below.

```
Import-Clixml –Path
C:\PShell\Labs\Lab_3\processes.xml |
Group-Object –Property basepriority
```

4. How many processes have a BasePriority of 8 (see the Name column)?

Task 3.4.4: Creating an HTML file

A common requirement for IT professionals is to produce easily accessible reports. Windows PowerShell makes this almost trivial with the ConvertTo-Html Cmdlet, which creates an html table from pipeline data.

1. Get a list of Hotfixes, sort them by InstalledOn date property, in descending order, and convert the data to an html table. Write the command below.

```
Get-HotFix |
Sort-Object -Property InstalledOn -
Descending |
ConvertTo-Html
```

2. What output do you receive?
   How would you output to a file named C:\PShell\Labs\Lab_3\hotfixes.htm?
   Write the completed pipeline command below and execute it.

```
Get-HotFix | Select-Object -Property Description, hotfixid, installedon | Sort-Object -Property InstalledOn -Descending | ConvertTo-Html | Out-File -FilePath C:\PShell\hotfix.html
```

3. The html can easily be styled with a .css (cascading style sheet) file.

   Get help for the ConvertTo-Html Cmdlet to discover how to do this. Use the file provided in C:\PShell\Labs\Lab_3\Hotfix_Style.css to apply styling to a new .htm file that contains the following information.

   a. Using Get-EventLog, select the newest 20 entries of type "Error" and "Warning" from the System event log.
   b. Select the TimeWritten, EntryType and Source properties.
   c. Convert the data to html format, specifying the C:\PShell\Labs\Lab_3\Hotfix_style.css file as the CSSUri parameter value, during the conversion.
   d. Save the output to the file C:\PShell\Labs\Lab_3\Events.htm.

   Write the working pipeline command below.

Considering adding an invoke or something to pop IE to let students actually see the file they just made and styled.

```
Get-EventLog -LogName System -Newest 20 -EntryType error,warning | Select-Object -Property Time,EntryType,Source | ConvertTo-Html -CssUri C:\PShell\Labs\Lab_3\Hotfix_style.css | Out-File -FilePath C:\PShell\Labs\Lab_3\events.htm
```

TIP: To display the HTML file within the default browser (e.g. Internet Explorer) use the Invoke-Item Cmdlet.

Invoke-Item "C:\PShell\Labs\Lab_3\Events.htm"

Task 3.4.5: View output in an interactive table

Windows PowerShell is an interactive, command-driven environment. The Out-GridView Cmdlet allows the user to sort, select and query pipeline data within a graphical user interface (GUI).

1. Get help for the Out-GridView Cmdlet.
2. Type the following command.

```
Get-Hotfix |
Out-GridView –Passthru |
Export-Csv –Path C:\PShell\Labs\Lab_3\hotfixes.csv
```

3. Interact with the gridview.
    a. Sort by clicking on the Description column header.
    b. Add a filter criteria to display Descriptions that contain the string "Security".
    c. Select any three rows using <Ctrl > + left mouse-click.
    d. Press the "OK" button at the bottom-right of the Grid view window.
4. Import the csv file C:\PShell\Labs\Lab_3\hotfixes.csv and display the contents in table format. Are the rows you selected in the grid-view in step 3c visible? Are these rows the only rows within the output? Why?
5. Execute the command in step 2 again, but this time remove the -Passthru switch parameter. What difference do you observe in the pipeline command's behavior? Is the OK button still visible?

Just trying to get the student to think about the fact that the output from the Out-Gridview including the filtering was passed along the pipeline to the Out-Csv cmdlet