Advanced Scripting   
Pipelines

Last Updated: 9/25/2021 2:03 PM Version 1  
Document Prepared for: CIT361 Student

# Name Zach Lucas ID 895649438

# Instructions

Save a copy of this document. Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

Pipelines are a powerful way to manipulate objects in PowerShell. They allow the output from one command to be the input to another command. You can string as many commands together as needed to complete the task. This exercise will introduce you to many cmdlets that are fundamental for processing data. You will use the get-process cmdlet to give us stuff to work with.

# Requirements:

* PowerShell
* Psfiles (You should have these but if not you can get them from <http://cf.esage.com/psfiles.zp>)

# Task 1—Pipes

As we learned in the last exercise cmdlets can return an array of items as well as a single item. In either case PowerShell can use that output as the input to another cmdlet. This is known as piping. Many “sets” cmdlets are designed with piping in mind. Often the get- version of the cmdlet is used to retrieve the data then another cmdlet say stop- will act upon the piped object.

## Steps

1. First make sure there are no instances of notepad running.  
   Get-Process notepad
   1. You should get an error, if you get something other than an error find and close those instances of notepad.
2. Start notepad  
   notepad
3. Get a list of the notepad processes then stop those processes by piping the output of Get-Process to Stop-Process  
   Get-Process notepad|Stop-Process
   1. Your notepad should be closed.
4. Pipes work on multiple items as well. This time you will start several notepad instances.  
   notepad  
   notepad  
   notepad
5. Now close them all  
   Get-Process notepad|Stop-Process
6. Thought problem. What do you think would happen if you entered get-process|stop-process  
   It wouldn’t work because you didn’t pass any rules/parameters into the Get-Process

# Task 2—Get-Member

A super useful cmdlet that needs to be used with a pipeline is Get-member. Get-Member retrieves a list of Methods and Properties from the object(s) piped to it.

## Steps

1. Getting the members of literals. Enter:  
   'somestring'|get-member
   1. How many methods are there? 48
   2. How many properties? 2
2. Now try a number literal  
   1830|Get-Member
   1. How many methods are there? 21
   2. How many properties? 0
3. Now try the results of a cmdlet  
   Get-Date|Get-Member
4. Using your previous knowledge figure out what type Get-Date returns.
   1. What expression did you use? (Get-Date).GetType()
   2. What is the type name? DateTime
   3. What is the type fullname? System.DateTime
5. Static members. Types are defined by classes, classes can have static members, static members are members that you can access without having an instance of the class. All you need to know is the class name. The -static switch parameter of Get-Member can be used to get static members as well. Find the static members of the type returned by get-date:  
   Get-Date|Get-Member -Static
6. To use a static method you use the type name then a double colon :: then the static member name. Use the now static method to find the current date and time.  
   [datetime]::Now
7. Try the isleapyear() method, it requires a year represented by an integer argument. Is 2030 a leap year?  
   [datetime]::IsLeapYear(2030)
   1. Is 2030 a leap year? No (False)
8. Challenge. Is this year a leap year? Do not use literal numbers, discover the year with code.
   1. Is this year a leap year? No
   2. What expression did you use to answer the question? [datetime]::IsLeapYear((Get-Date).Year)

# Task 3—Organizing data with Select-Object, Sort-Object, and Group-Object

Organizing data with PowerShell is pretty simple. Note: the -object cmdlets have aliases defined for them by default that omit the -Object.

## Steps

1. Sorting
   1. Change to the psfiles\data folder.
   2. Get a directory listing sorted by filesize  
      dir|Sort-Object Length
      1. What is the third file in the list? Metals.csv
   3. Now sort from largest to smallest  
      dir|Sort-Object Length -Descending
      1. What is the fourth item in the list? RushSongs.txt
2. Selecting:
   1. You can select just the properties you want from an object and make a new object with just those properties with select-object. Specify the properties you want selected as a comma separated list. Enter:  
      dir|Select-Object Name,Length
3. Combining cmdlets.
   1. You can combine multiple pipes  
      dir|Select name,Length|Sort Length -Descending
   2. You can sort on multiple properties as well  
      gps|select company, name, ws|sort company, name
4. Grouping
   1. You can group objects by a property as well  
      gps|Group-Object company
   2. Group processes by company  
      gps|Group-Object company
      1. How many companies have software running on your system? 14
      2. How many processes are from Microsoft? 1
      3. How many processes don’t have a company name associated with them? 1

# Task 4—Measuring data with Measure-Object

The Measure-Object cmdlet does simple stats. It provides count, average, maximum, and minimum. To calculate any value other than count you must include the switch parameter by the same name as calculation.

## Steps

1. What is the average size of the file in the data directory  
   dir|Measure-Object lenth -Average
   1. What is the average size of the files? 188224.333333333
2. If you want stats for multiple properties you can send an array (comma separated list) of property names.  
   gps|measure WS,handles -Average -Minimum -Maximum
   1. What is the average handle count? 458.01872659176
3. In task 2 you counted the number of Properties and method of a string object. PowerShell can do this work.   
   'somestring'|Get-Member -MemberType Property|measure   
   or better yet  
   ('somestring'|Get-Member -MemberType method|measure).count

# Task 5—Filtering data with Where-Object

You don’t always want to keep all the data. Filtering allows you to keep a subset of the data that is going through the pipeline. The key to remember is that items that cause the filter statement to return true are passed through the pipeline. The $\_ variable in used by PowerShell to keep track of the item passing through the pipe. You can reference the current item in the pipe with $\_.

## Steps

1. Where-Object
   1. List the files that are larger than 50kb  
      dir |Where-Object {$\_.length -gt 50kb}
   2. Aliases galore. Where-Object has two aliases, where and the question mark. These expressions are equivalent to the preceding expression.  
      dir |Where {$\_.length -gt 50kb}  
      dir |? {$\_.length -gt 50kb}
   3. The expression in the {} can be as simple or as complex as you need. There is a shorthand version of the cmdlet, if you are only checking one property against one value you can write the condition like this:  
      dir | ? length -gt 50kb
2. Another useful filtering cmdlet is Select-String, select string is used to pattern match strings for passing through the pipe.
   1. First let’s show you how to get some strings. The Get-Content cmdlet will read a text file and create an array of strings where each line in the file is an element in the array. Try this:  
      get-content colors.txt
   2. Store the colors in a variable so we don’t need to read it in every time. This is by no means necessary, but it is useful to know how.  
      $colors=Get-Content colors.txt
   3. Select-String works pretty much like grep on Unix, it uses regular expression pattern matching. If there is a match the string is passed through the pipe. My son’s favorite color is green so let’s see what color names have **green** in them.  
      $colors|select-string green
   4. Using your knowledge write an expression that returns the number of colors that have green in them.
      1. Record your expression here: $colors|Select-String green|Measure-Object
      2. How many colors have green in them? 91
   5. How many colors don’t have green in them? 938
   6. What expression(s) did you use to find this out? $colors|Select-String -NotMatch green|measure

# Task 6—Manipulating objects with Foreach-Object

Often you will want to manipulate the data as it flows through the pipe. The Foreach-Object cmdlet does exactly that. Foreach-Object allows you to run a script block for each item as it passes through the pipe. You can access the current object with the $\_ variable just like with the Where-Object cmdlet.

## Steps

1. The range operator .. can be used to make a quick array of integer. If you type 1..10 it will create an array with the integers from 1 to 10. We can use that as a source for a pipe. Enter  
   1..10  
   just for fun. You should see the number 1 though 10.
2. Now let’s use that to show how you can execute script fore each element in the array. Enter  
   1..10|ForEach-Object {$\_ \* 10}
3. You should see the numbers 10,20,30 and so forth. We simply multiplied the item in the pipe by 10.
4. Just like Where-Object, Foreach-Object has two aliases, foreach and the percent sign if the percent sign is immediately after a pipe. These expressions are equivalent to the preceding expression.  
   1..10|ForEach {$\_ \* 10}  
   1..10|% {$\_ \* 10}
5. You don’t need to do anything with the item coming though the pipe.  
   1..10|%{'hi mom'}
6. The script block in the {} can be as simple or as complex as you want. If you want to put multiple statements in the script block you can use the “;” or just press return, PowerShell will figure it out.  
   1..10|%{'hi mom';$\_ \* 10}
7. Want to launch a bunch of notepads? Enter:  
   1..20|%{notepad}
   1. Now kill them  
      gps notepad|%{$\_.kill()}
   2. But that was harder than just using stop-process, there are many ways of doing the same thing in PowerShell.
8. There is also a shorthand version of foreach, if you just want to access a single property or method you can do this:  
   get-history|% commandline  
   To get just the command property values from the results of Get-History.

# Task 7—Formatting data with Format-Table(ft), Format-List(fl), Out-Host, Out-Gridview.

Sometimes you just want to make your data easier to understand. PowerShell has great formatting cmdlets to help out. Sometimes it seems like a mystery how PowerShell is going to display data from a cmdlet. In truth it is up to the cmdlet developer to decide. However, if the cmdlet outputs 5 or fewer properties, it is displayed as a table, more than 5 then it is displayed as a list. Also cmdlets often return objects with many more properties than what are shown by default. You must use get-member to find them.

## Steps

1. As you have seen many times Get-Process returns data in a table format. If you would rather see it in a list format you can just use the Format-List cmdlet:  
   Get-Process|Format-List
2. Notice Get-Process shows only a few properties, but the process object that is returned from Get-Process has many more.  
   gps|gm
   1. Maybe you want to see the process name, it’s memory usage and amount of time in the CPU. You can use the Format-Table cmdlet to see just what you want.  
      gps|ft Name,WorkingSet,TotalProcessorTime
      1. Discover How much cpu time and memory powershell is using.
      2. What expression did you use? Get-Process powershell |ft Name,WorkingSet,TotalProcessorTime
      3. How much memory is being used? 143499264
      4. How much CPU time has been used? 00:00:15.5468750
3. Lets look at your history. Enter the command:  
   get-history
   1. What columns are displayed? ID, CommandLine
4. There are two kind of interesting columns names: **StartExecutionTime** and **StopExecutionTime**. Add them to your output:  
   **Get-History|ft id,commandline,StartExecutionTime,EndExecutionTime**
5. That is interesting, but it would be more interesting if we could see how long each command took to run. Custom columns can solve the problem. The format commands allow you to create columns from expressions. Next you will use a custom expression to see how long each command in your command history took to run. Enter the command(all on one line):  
   Get-History|ft id,commandline,@{label="Ms";expression={($\_.EndExecutionTime-$\_.StartExecutionTime).TotalMilliseconds}}
   1. Note: The calculated column is defined by a hash table. We will cover them in detail later
6. **Out-GridView** is a handy tool if you want to see your output more graphically. Try this:  
   **Get-Process|Out-GridView**
   1. The tabular window has sorting and filtering capabilities, play with them for a while.
7. Finally, **Out-Host** sends output to the PowerShell host, since this is the default if you don’t specify where the output should go it is rarely used. The one case you will want to use it is to page the output. To page the output use the -Paging. Try this  
   gps|Out-Host -Paging
   1. You can shorten it by using the alias and jus a partial parameter.  
      gps|oh -p
   2. What key is used to advance one page? Space

# Deliverable

Upload this document with completed answers to i-learn.