Power Shell 101

**Module 8 Hands-on Activity – PowerShell in VSCode**

6/2/2019 Developed by Jin Chang, Sion Yoon

6/2/2019 Tested by Sion Yoon

Center for Information Assurance (CIAE) at City University of Seattle



**Learning Outcomes**

* Continue to learn loop cmdlet
* Describe basic array and hash table concepts

**Resources**

* Loop and While
  + <https://en.wikiversity.org/wiki/PowerShell/Loops#While>
* Arrays and hash table
  + <https://www.tutorialspoint.com/powershell/powershell_array.htm>
  + <https://en.wikiversity.org/wiki/PowerShell/Arrays_and_Hash_Tables>

**Activities**

* While loop
* Array
* Hash Table

**Operators - Recap on some of the relevant while and array operations**

The following table is a list of some relevant while and array operations (some that we previously worked with).

|  |  |  |
| --- | --- | --- |
| Operator Name | Operators | Description |
| Assignment Operators | **=, +=, -=, \*=, /=, %=, ++, --** | These are used to assign, change or append values to variables.  **$a = “Hello ”**  **$a += “World”** |
| Comparison  Operators | **-eq, -ne, -gt,**  **-lt, -le, -ge** | They are used to compare values or perform a test.  **“hello” –eq “Hello”**  **1,2,3 –ne 2**  **1,2,3 –gt 1** |
| Comma Operator | **,** | The comma creates an array.  **$myArray = 1,2,3** |
| Hash Table Operator | **@{key=”value”; ; }** | A paired list of key and value. For example,  **$h = @{key="value"; name="PowerShell"; version="2.0"}**  To access the value  **$h[“name”]**  **$h[“version”]** |

**While loop**

While loop executes a very similar way as for loop that we learned in one of our previous activities. Here is the syntax for while loop. Note that while loop exits if the condition is false; otherwise, it will continue to execute the statement.

**while(<condition>) # this condition control the loop behavior**

**{**

**< statement execution >**

**}**

Here is an example. You can try this example in the command line.

**$val = 0 # By the way, this is an optional statement. When the variable is initialized, it is always set to 0;**

**while($val –ne 3){**

**Write-Host $val**

**$val++;**

**}**

The output will be:

**0**

**1**

**2**

|  |
| --- |
| **Q1**  **Using while loop, write a command that displays multiples of 10 from 0 to 100 (i.e., 0, 10 … 80, 90, 100). Insert your screen capture here.** |

**Array**

An array is a variable and a type of data structure that maintains a set of ‘same’ type of one or more ‘different’ values. For example, the following figure shows a set of ‘same’ integer type of 8 ‘different’ values. An array always has “Index” and “Elements”

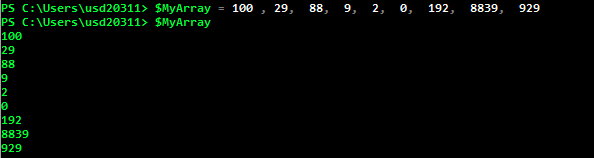
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Index** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **Elements** | **100** | **29** | **88** | **9** | **2** | **0** | **192** | **8839** | **929** |

Here is how to declare an array to hold a collection of elements using commas in PowerShell

**$myArray = 100, 29, 88, 9, 2, 0, 192, 8839, 929**

Then to see the array elements, you simply type the array variable name,

**$myArray**



We can then access an individual element by providing index value with [ *index\_value* ]. For example, $myArray[3] will print 4th value since the array type maintains a 0-th index.

|  |
| --- |
| **Q2**  **Can you print the 7th value of $MyArray? Insert your screen capture here.** |

Let’s find out the list of available methods and member variables (aka property) associated with this Array type in PowerShell. Go ahead and type

**Get-Member –InputObject $MyArray**

Can you find out which properties will show you the total number of elements of the array, **$MyArray**?

|  |
| --- |
| **Q3**  **Can you find out which properties will show you the total number of elements of the array, $myArray? Find at least two correct parameters (methods or property) to display the total size of $myArray and insert your screen capture here.** |

Sometimes, we need to read or write some of the values of an array. From the available methods, there is a method called “Get()” or “Getvalue()” for reading and “Set()” or “SetValue()” for writing a value of the array by passing a target index. Note that index is an integer type and element is a System.Object (i.e. int, float, double, string, etc.). From the Get-Member cmdlet, we now know that the syntaxes for those methods are following:





|  |
| --- |
| **Q4-A**  **Find cmdlet that prints the 4th value of $myArray using Get() method and insert your screen capture here.**    **Q4-B**  **Find the cmdlet that replaces the 4th value with 999 of $myArray using Set() method and print the whole array. Insert your screen capture here.** |

**Hash Table (aka Hash Map)**

As we discussed, an array is a collection of the ‘same’ types of elements stored in an object or variable. And you can access the value via index. HashTable is a special type of array, but it is a little different in how you access the array to retrieve the value, “associative manner” using keys. It is a data structure that maps “Key” to “Value.” In fact, many of the cmdlets utilize hash tables to arrange their inputs for further processing. As you can see in operator table at the beginning of this activity, it starts with “@” followed by “{“.

Again, here is the definition of a hash table to a variable called “$h”

**$h = @{**

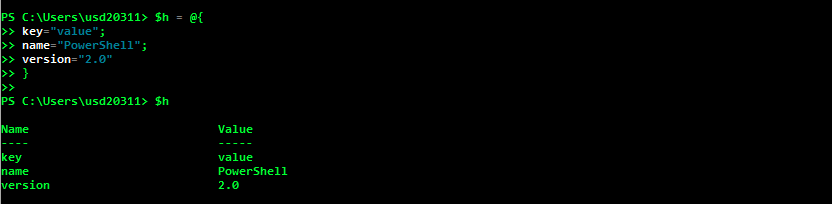
**key="value";**

**name="PowerShell";**

**version="2.0"**

**}**

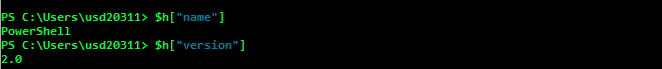
If you want to print out the values in the hash table, simply type $h. As shown below, the values are assigned to its associated keys such as “key”, “name”, and “version”.



To print the values defined in a hash table, you simply access via “key” rather than a direct “index.” Well, to be exact, there is a hash function that generates an index when operating with hash table internally but we will discuss that later time.

**$h[“name”] # not $h[0]**

**$h[“version”] # not $h[2]**



A hash table can be created in several steps. Here is the example and type each step in the command line as you follow.

1. Create an empty hash table and assign the table to a variable called **$hash**

**$hash = $null #good practice to initialize any variable prior to use**

**$hash = @{} #empty hash table**

1. Obtain the data to store. For this example, we’ll use get-process to store only unique processes. Let’s just check to see if this cmdlet works as expected.

**Get-Process | Sort-Object -Property Name –Unique # unique is**

**used to avoid any collision while accessing the value via the key.**

1. Store the collected data in a variable.

**$proc = Get-Process | Sort-Object -Property Name -Unique**

1. Now, you can traverse the table by using a loop such as **foreach**. While looping the table, “**add**” method can be used to add the ***key-value*** pairs to the **$hash**.

**foreach ($p in $proc){**

**$hash.add($p.name, $p.id)**

**}**

|  |
| --- |
| **Q5.**  **Print the hash table ($hash), and insert the screen capture.**    **Q6**  **Find the WINWORD process id by accessing the hash table you created, insert the screen capture including the cmdlet you used to print the process id.** |

Finally, we discussed the difference between Get-EventLog and Get-WinEvent. One of the greatest thing about Get-WinEvent is that the cmdlet uses the hash table and it is really fast to add, read, write, or replace by using the key.

For instance, if I need to display the events from the Application log, I would use one of the followings. Try both, and see how long it takes to display the Microsoft-Windows-Defrag logs.

**Get-EventLog -LogName Application | Where-Object Source -Match defrag**

**Get-WinEvent -LogName Application | Where-Object {$\_.ProviderName -Match 'defrag'} # you can kill the process by pushing ctrl-c if it takes too long**

But, if you utilize the hash table parameter by “-FilterHashtable”, it is much faster. It is O(n) vs. O(1) difference where n is the number of entries in the Application log.

**Get-WinEvent -FilterHashtable @{LogName='Application'; ProviderName='\*defrag'} # one line cmdlet**

|  |
| --- |
| **Q7**  **Insert the screen capture of the command output that you got using the hash table.** |