Power Shell 101
Module 5 Hands-on Activity – PowerShell in VSCode

5/8/2019 Developed by Jin Chang, Sion Yoon
5/11/2019 Tested by Sion Yoon, Jin Chang
Center for Information Assurance (CIAE) at City University of Seattle

Learning Outcomes

- Review on using the pipeline
- Learn about different operators including the dollar sign-underscore-dot operator
- Learn how to use variables to store objects
- Learn about for and foreach

Resources

- Operators
  - https://www.varonis.com/blog/windows-powershell-tutorials/
- Using variable to store objects
  - https://docs.microsoft.com/en-us/powershell/scripting/learn/using-variables-to-store-objects?view=powershell-6
- Dollar sign-underscore-dot operator
  - http://techgenix.com/dollar-sign-underscore-dot/
- For, foreach and foreach-object
  - https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_for?view=powershell-6
  - https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_foreach?view=powershell-6
  - https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/foreach-object?view=powershell-6
  - http://techgenix.com/powershell-foreach-loop/

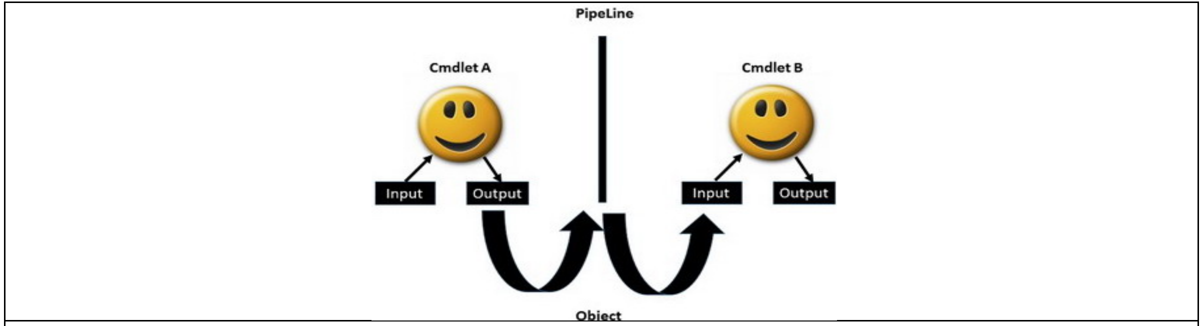
Activities

- Pipeline Review
- Operators
- Using Variable to Store Objects
- Dollar Sign-Underscore-Dot Operator
- For and Foreach
- Q&A

Pipeline Review

1. Open PowerShell in VSCode.

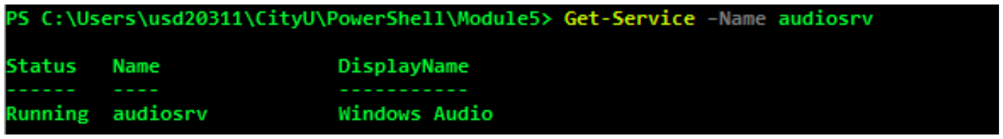
2. We all learned about the pipeline, but let's quickly review the operator. The output from one command is used as an input for the next command in the pipeline, `% A | B | C | D` and so on. Therefore, the output of A now becomes an input to B, and the output of B becomes an input to C and so on.



http://badbitboy.blogspot.com/

3. Before expanding more on this, type:

```
% Get-Service -Name audiosrv
```



```
PS C:\Users\usd20311\CityU\PowerShell\Module5> Get-Service -Name audiosrv

Status   Name               DisplayName
------   ----               -----------
Running  audiosrv           Windows Audio
```

The output from the cmdlet A is essentially an object (service object = audiosrv) that carries properties, methods, and events, etc. From the audiosrv object, we can see they have three properties listed from the command line, Status, Name, DisplayName. For instance, you can imagine a house and its property includes "Address", "Year-Built", and "Color" etc.

4. Now using the questions in our Module 4 hands-on-activity as an example, type:

```
% Start-Process PowerShell -Verb RunAs

% Get-Service -Name audiosrv | Stop-Service

% Get-Service -Name audiosrv | Start-Service
```

We have the audiosrv object being passed over to the next cmdlet (Stop-Service or Start-Service), and it'll use the service object as an input to finish its operation. Note that you can simply use the following two cmdlets as it is easier with less typing. This is to give you practice using the pipeline.

```
% stop-service audiosrv

% start-service audiosrv
```

Operators

5. Below is a list of operators that are commonly used:

Operator Name	Operators	Description
Comment block Operator	`#`	Words after # will be ignored. `Get-Process # getting all the Windows processes`

Arithmetic Operators	`+, -, *, /, %`	These can be used with the following data types, Int, String, DateTime, HashTable.
		`"Hello " + "World"`
		`(Get-Date).AddDays(-1)`
		`7%2`

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`

Split/Merge Operators	`-split`
	`-join`
		These are useful to manipulate strings, dividing, combining substrings.
		`-split "red yellow blue green"`
		`"Lastname:Firstname:SSN" -split ":"`
		`-join "a", "b", "c"`
		`-join("a", "b", "c")`
		`"CityU", "Seattle" –join "@"`

Call Operator	`&`	This executes commands that are stored in variables and represented by strings or script. Note that this doesn't parse additional parameters.
		`$c = "Get-ExecutionPolicy"`
		`@ $c`

Comma Operator	`,`	The comma creates an array. `$myArray = 1,2,3`
Cast Operators	`[ ]`	This converts to objects to the specified type.

		`[datetime]$birthday = "1/28/1993"`
		`[int64]$a = 34`
		Also, provides access to an array.
		See below.

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"]`

XML Operators	**`[xml]`**	XML parsing operator. For example,
		`$x = [xml]"<doc><intro>Once upon a time …</intro></doc>"`
		`$x["doc"]`

Range Operator	`..`	Generates the sequential integers or characters defined in the range.
		`$list=1..10`
		`$max=10`
		`Foreach ($list in 1..$max) { Write-Host $list}`

Array Subexpression Operator	**`@( )`**	It returns the result of one or more statements as an array. **`$ray = @(1,2,3,4)`**

Subexpression Operator	**`$( )`**	Command from $(*cmd*) is executed when it is combined with " "
		`$ray = @(1,2,3,4)`
		`Write-Host "$ray.length"`
		`Write-Host "$($ray.length)"`

6. Let's practice using these operators. The first cmdlet below will simply get you the date but what if you want the day of the year, the day of the week, the hour, or today's day? You can use the parenthesis and dot operator to narrow your search. Type,

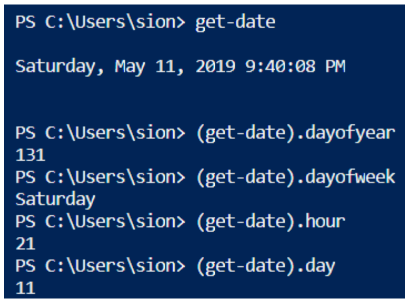
```
% get-date

% (get-date).dayofyear

% (get-date).dayofweek

% (get-date).hour

% (get-date).day
```



```
PS C:\Users\sion> get-date

Saturday, May 11, 2019 9:40:08 PM


PS C:\Users\sion> (get-date).dayofyear
131
PS C:\Users\sion> (get-date).dayofweek
Saturday
PS C:\Users\sion> (get-date).hour
21
PS C:\Users\sion> (get-date).day
11
```

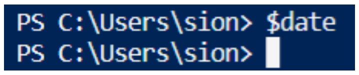
7. If your supervisor tells you that you have 10 days to complete a project and you want to know the date after 10 days, type:

```
% (get-date).adddays(10)
```

Using Variable to Store Objects

8. You can get the same results by creating a variable. Choose what you want to name the variable. I'm going name my variable "date". PowerShell only creates the variable if it does not exist. Type your variable along with a dollar sign in the front of the variable to see if it returns a result.

```
% $date
```

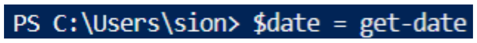


```
PS C:\Users\sion> $date
PS C:\Users\sion>
```

It returns no result because it does not have a value. It means that you can create a variable and assign it a value.

9. The following command will send the output of get-date to $date. Do the same for the name of the variable that you have created.

```
% $date = get-date
```

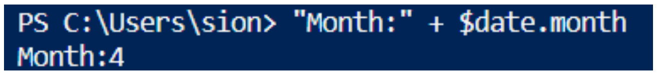


```
PS C:\Users\sion> $date = get-date
```

10. Now type your variable along with a dollar sign in the front of the variable to see if it returns a result. You should now get the date information.

11. Type "Month:" in quotes to display the text and use ".month" after the variable that you have created to get the month.

```
% "Month: " + $date.month
```

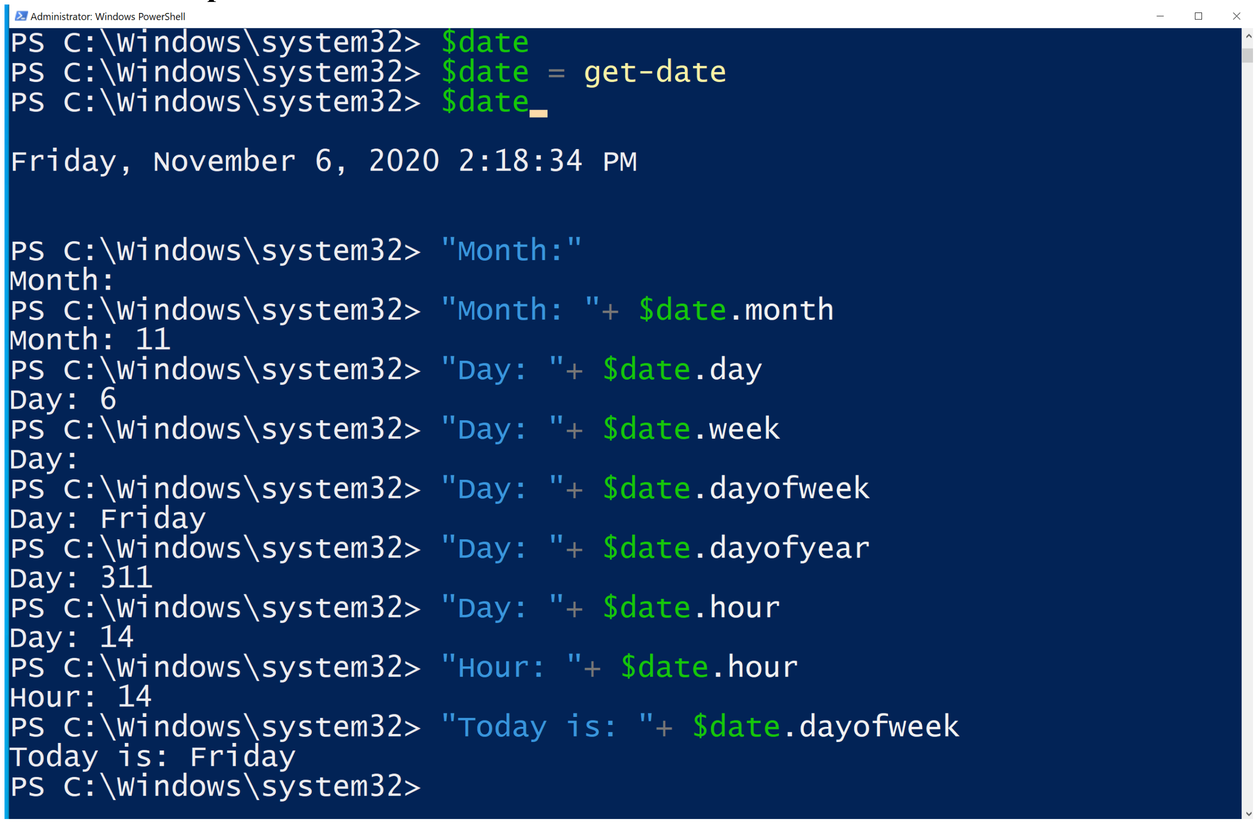


```
PS C:\Users\sion> "Month:" + $date.month
Month:4
```

Do the same for the day of the year, the day of the week, the hour, or today's day.
Screenshot and paste your results here:

Q&A #1

Your screen capture →



```
PS C:\Windows\system32> $date
PS C:\Windows\system32> $date = get-date
PS C:\Windows\system32> $date

Friday, November 6, 2020 2:18:34 PM


PS C:\Windows\system32> "Month:"
Month:
PS C:\Windows\system32> "Month: "+ $date.month
Month: 11
PS C:\Windows\system32> "Day: "+ $date.day
Day: 6
PS C:\Windows\system32> "Day: "+ $date.week
Day:
PS C:\Windows\system32> "Day: "+ $date.dayofweek
Day: Friday
PS C:\Windows\system32> "Day: "+ $date.dayofyear
Day: 311
PS C:\Windows\system32> "Day: "+ $date.hour
Day: 14
PS C:\Windows\system32> "Hour: "+ $date.hour
Hour: 14
PS C:\Windows\system32> "Today is: "+ $date.dayofweek
Today is: Friday
PS C:\Windows\system32>
```

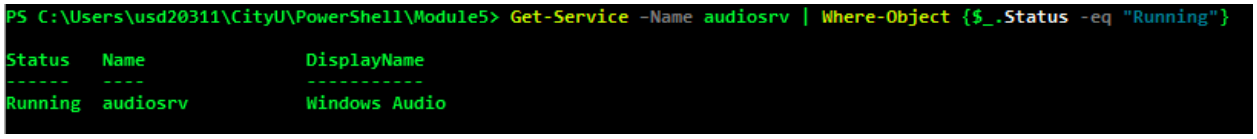
Dollar Sign-Underscore-Dot Operator

12. Now, let's take a special operator, `$_.` This is used in conjunction with pipe. Often times, you will come across the sign "dollar-underscore-dot" from the various pipe related examples in the books and online resources. What is it? When can I use it? First, it is a type of variable. From the part above, we know that $ followed by a name makes a variable. That same "$" can be concatenated by "`_.`" And used by a variable in the pipeline. When the output of the previous command is stored into "$_." and the next command will access the objects via "$_." for the current operation.

    `Where-Object {$_.}` is the way the object from the previous cmdlet carries over to the next cmdlet. Type:

```
% Get-Service -Name audiosrv | Where-Object {$_.Status -eq
"Running"}
```

This will display an audiosrv service object if its status is "equal" to "Running"

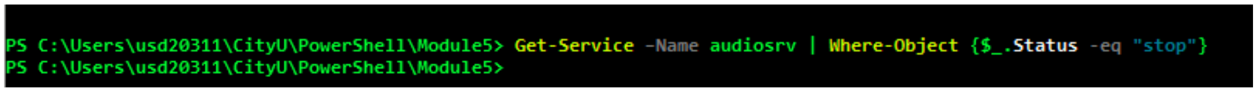


```
PS C:\Users\usd20311\CityU\PowerShell\Module5> Get-Service -Name audiosrv | Where-Object {$_.Status -eq "Running"}

Status   Name               DisplayName
------   ----               -----------
Running  audiosrv           Windows Audio
```

But if you try the following command, nothing will return.

```
% Get-Service -Name audiosrv | Where-Object {$_.Status -eq
"Stop"}
```



```
PS C:\Users\usd20311\CityU\PowerShell\Module5> Get-Service -Name audiosrv | Where-Object {$_.Status -eq "stop"}
PS C:\Users\usd20311\CityU\PowerShell\Module5>
```

13. Finally, let's then open ALL the services that have their status in "Running"

```
% Get-Service | Where-Object {$_.Status -eq "Running"}
```

For and Foreach

14. `for` loop is used for continuous running of statements based on a conditional test. Here is the syntax.

```
for (<initialization>; <condition>; <repeat>)
{
   < statement execution >
}
```

15. Here is an example. You can try this example in the command line. Make sure the drop down box on the top right of the terminal is set to "powershell", not the PowerShell Integrated Console".

```
for($i = 1; $i -le 10; $i++)
{
  Write-Host $i
}
```

16. `foreach` traverse all the items in a collection of items. The syntax is following.

```
foreach ($<item> in $<collection>)
{
    < statement execution >
}
```

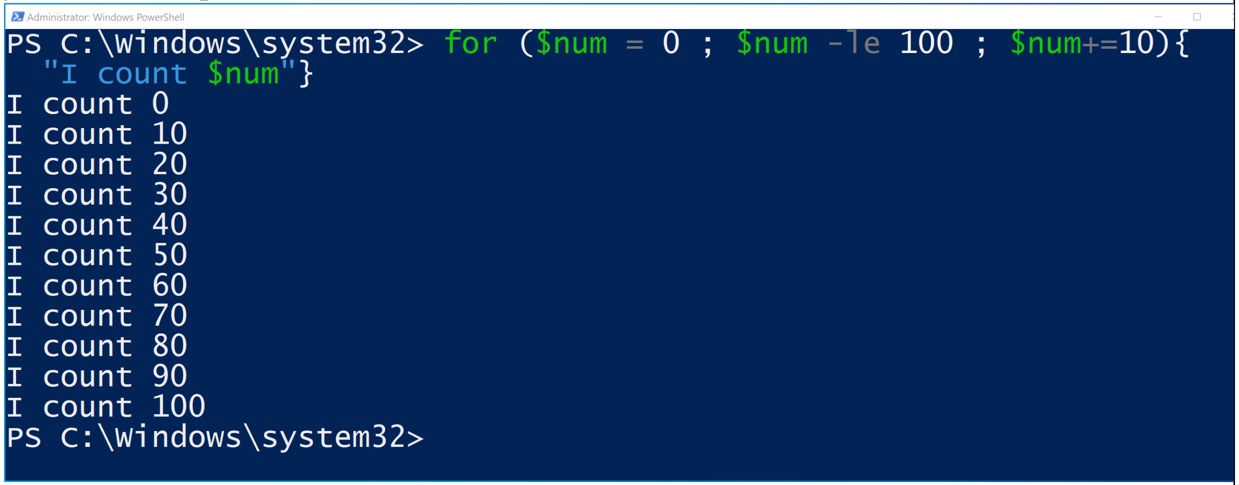
17. Here is an example. You can try this example in the command line.

```
$Array = "a","b","c","d"

foreach($letter in $Array)
{
  Write-Host $letter
}
```

Q&A #2

Using `for` loop, write a cmdlet that displays the multiples of 10 from 0 to 100. Insert your screen capture here.



```
PS C:\Windows\system32> for ($num = 0 ; $num -le 100 ; $num+=10){
  "I count $num"}
I count 0
I count 10
I count 20
I count 30
I count 40
I count 50
I count 60
I count 70
I count 80
I count 90
I count 100
PS C:\Windows\system32>
```

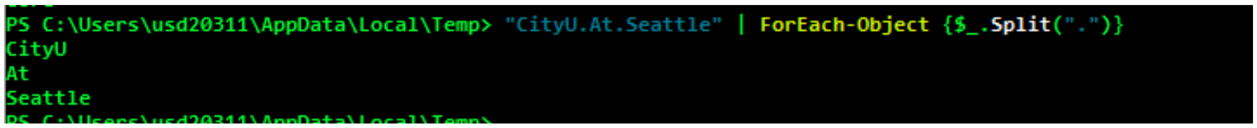
18. Let's check out another loop cmdlet. This time, it is called **`foreach-object`**. This is a pipeline variation of the **`foreach`** loop. **`foreach-object`** operates on each item in a *collection of input objects* rather than a single object (cf. **`where-object`**)

<cmdlet> or <object> | `foreach-object`

To use this cmdlet, you would pass a collection of objects via cmdlet or array of strings or number or objects, and then pipe to **`foreach-object`**.

Here is an example.

```
"CityU.At.Seattle" | foreach-object {$_.Split(".")}
```



```
PS C:\Users\usd20311\AppData\Local\Temp> "CityU.At.Seattle" | ForEach-Object {$_.Split(".")}
CityU
At
Seattle
```

PUSH YOUR WORK TO GITHUB

Once you completed the Hands-on practice, **export this document to PDF**, save in the same Module 5 folder you cloned, then do the following to push your work to GitHub

Open the terminal from the VSCode by hitting the control + ~ key, make sure you are in the right path, for example: KimNguyen/Desktop/ISEC505/HOP05-KimNguyenMai/Module 5

Type the following command:

>>> **git add .** (to copy all changes you have made)
>>> **git commit -m "Submission for Module 5 – YOUR GITHUB USERNAME"** (To add a message to your submission)
>>> **git push origin master** (to upload your work to Github)