CIS161 Assignment 2

HW 2.0

Objective:

1. Create and work with aliases.
2. Work with variables.
3. Build basic pipelines.
4. Learn basic algorithms.

Upload your scripts to Blackboard by clicking the corresponding link.

Script 2.1:

Write a script called set-myalias.ps1 that has the commands necessary to create two aliases:

1. An alias for notepad.exe called ed.
2. An alias for the legacy command "sc.exe" called sctl. One of the problems of running legacy commands like sc.exe in the PowerShell console is that sc is an alias for set-content. So, entering just sc in the console would execute set-content. Your alias needs to have a full-qualified file name to reference sc.exe which is found in c:\windows\system32.

This script requires the **new-alias** cmdlet.

Approach:

1. Working in the console to develop the syntax to create each alias. To test the ed alias, enter ed in console which should start Notepad with an empty file. Also test ed specifying a text file name. Notepad should execute and load the file specified.
2. Make sure each alias is working as specified before proceeding.
3. Create the script file set-myalias.ps1 and enter the commands to create the aliases.
4. Close the PowerShell console and open a new console to test your script. This is required since testing in the current instance of the console will throw an error since the aliases are already created.
5. In the new console, test your script as described below.

A PowerShell console has an associated runspace. Running a script from a console creates a new runspace in which the script executes. So, your script would create the aliases in the new runspace and not in the current runspace as shown below.

```
PS>
PS>  .\set-myalias.ps1
PS>
PS>  ed
ed : The term 'ed' is not recognized as the name of a cmdlet, function, script file, or
operable program. Check the spelling of the name, or if a path was included, verify that the
path is correct and try again.
At line:1 char:1
+ ed
+ ~~
    + CategoryInfo          : ObjectNotFound: (ed:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS>  sctl
sctl : The term 'sctl' is not recognized as the name of a cmdlet, function, script file, or
operable program. Check the spelling of the name, or if a path was included, verify that the
path is correct and try again.
At line:1 char:1
```

To remedy this, you need to specify the dot (".") operator when executing the script as shown below.

```
PS>
PS>  . .\set-myalias.ps1
PS>
PS>  sctl | out-host -Paging
DESCRIPTION:
        SC is a command line program used for communicating with the
        Service Control Manager and services.
USAGE:
        sc <server> [command] [service name] <option1> <option2>...


        The option <server> has the form "\\ServerName"
        Further help on commands can be obtained by typing: "sc [command]"
        Commands:
          query-----------Queries the status for a service, or
                          enumerates the status for types of services.
          queryex---------Queries the extended status for a service, or
                          enumerates the status for types of services.
```

Script 2.2:

The string object in PowerShell has many useful methods for working with text strings. In this problem, you will use the substring method to parse and IP address into individual octets. The substring method extracts a substring from a string and has two forms:

Substring(int startIndex)

Substring(int startIndex, int length)

In the first syntax form “Substring(int startIndex)”. Enclosed within the parentheses is the integer parameter “startindex”. This parameter identifies the beginning of the substring. Executing the method in this form returns a substring beginning at startindex and continuing until the end of the string. In the second syntax form the parameter length specifies the length of the string to extract. Remember that the index into a string is 0-based, so the first character is at index 0 and not index 1. Refer to the Variables page, String Operators section for examples on using substring.

Write a script called **parse_IPv4address** that is passed the string parameter $ipaddress. For the sake of simplicity, assume that each octet will always consists of three digits. The first line of the script is:

param ($ipaddress)

Approach:

1. Use the substring operator to extract the first octet and store it in the variable octet1.
2. Use the substring operator to extract the second octet and store it in the variable octet2.
3. Use the substring operator to extract the third octet and store it in the variable octet3.
4. Use the substring operator to extract the fourth octet and store it in the variable octet4.
5. Use the write-host cmdlet with a quoted string to display the output in the format shown below.

When executing the script always pass three-digit octets in the IP address.

```
PS>
PS>  .\2.2-parse_ipv4addr.ps1 192.168.001.152
octet1: 192  octet2: 168  octet3: 001 octet4: 152
PS>
PS>  .\2.2-parse_ipv4addr.ps1 "207.075.134.001"
octet1: 207  octet2: 075  octet3: 134 octet4: 001
PS>
```

Script 2.3:

Write a script called **show_datediff.ps1** that display the difference between the current date and the date December 7, 1941. Your script should do the following:

Approach:

1. Use the **get-date** cmdlet to get the current date and store it in a variable called $dtoday.
2. Assign the string 'December 6, 1944' to a [datetime] variable called $dpast, It's important that the variable is typed as [datetime] otherwise PowerShell will consider it a string.
3. Store the difference between the two variables in the variable $diff.
4. Use write-host to display $dtoday, $dpast, and the total day difference as shown below.

The script should execute. shown below.

```
PS>
PS>  .\2.3-show_datediff.ps1
The difference between 12/28/2017 09:36:01 and 12/07/1941 00:00:00 is 27780.4000185758 days
PS>
```

Don't be overly concerned about the formatting of the date and date difference output. While you are not required to do this, feel free to research ways to format the date so that the times do not show in the output and the date difference is rounded. There is also a slight flaw in the script as evidence by the fractional part of the date difference. The script is calculating the date difference from the current time (9:36 AM). The script should compute a date difference from 12:00 AM of the current day. Feel free to fix this flaw.

Script 2.4:

Alter script 2.3 **show_datediff.ps1** so that the date to compare against is not the hardcoded December 7, 1941 but is passed as a parameter.

The script would execute from the console as shown below:

2.3-show_datediff "December 7, 1941"

Either single or double quotes must be used to delimit the date parameter. Once the script is working correctly run it with a date that is not quote-delimited to see the result. Below is are sample executions of the script. Note, that the date passed in the examples below has different forms. PowerShell is able understand and convert many different forms of a date.

```
PS>
PS>  .\2.4-show_datediff.ps1 "December 7, 1941"
The difference between 12/28/2017 09:58:58 and 12/07/1941 00:00:00 is 27780.4159602243 days
PS>
PS>  .\2.4-show_datediff.ps1 "10/14/1066"
The difference between 12/28/2017 09:59:03 and 10/14/1066 00:00:00 is 347421.416012837 days
PS>
PS>  .\2.4-show_datediff.ps1 "Jan 23 1949"
The difference between 12/28/2017 09:59:57 and 01/23/1949 00:00:00 is 25176.4166354608 days
PS>
```

Script 2.5:

Write a script called **analyze_sentence.ps1** that accepts the parameter **$sentence** which contains a text string of words separated by spaces. The script produces the following output:

1. The text passed displayed in magenta.
2. The number of words in the text.
3. The first word of the text in magenta. .
4. The last work of the text in magenta.
5. The first three words of the text in magenta.
6. The last three words of the text in magenta.

The only required cmdlet is **write-host**. You will be required to display different colored text on the same line in the console (see illustration below). Doing this requires two write-host statements. Hint, autocomplete (Tab key on the keyboard) reduces the amount of typing for the parameters used in write-host.

Approach:

1. Use write-host with the "-nonewline" parameter to display the descriptive text preceding the output of the parameter.
2. Use write-host with the "-foregroundcolor magenta" to display the parameter.
3. Save the script and test it.
4. Use the split method (Refer to the Variables page, String Operators section) to split the text string contained in $sentence into words. Store the result in the variable $words.
5. $words is an array containing each individual word in the text. So, the length of the array is the number of words in the text (Refer to the Variables page, Arrays section). Use write-host to display the number of words.
6. Save the script and test it.
7. The first element in the array is the first word in the text. Use write-host as described in previous steps to display the descriptive text in the default color and the first word in magenta.
8. Save the script and test it.
9. You will need to calculate the last element in the array which contains the last word of the text. Create a variable called $ixlast assigning to it the calculation required to determine the last element in the array. This is not rocket science so don't overthink the problem. There are examples in Variables page, Arrays section of the textbook. Display the multi-colored output as described in previous steps.
10. Save the script and test it.
11. Displaying the first three words in the requires an array slice of the first three elements in the array (Do I need to write to refer to the Variables page, Arrays section of the textbook?). Display the multi-colored output as described in previous steps.
12. Save the script and test it.
13. Displaying the last three words in the array requires a calculation. Create a variable called $ixlast3 assigning it the calculation necessary to derive the index for the third element from the last element. Once again not rocket science. You know the index for the last element of

the array so derive the index for the third word preceding the last is not difficult. Once you derive $ixlast3, you can create an array slice from $xlast3 through $ixlast. Display the multi-colored output as in previous steps. There will be a bit of trial and error here but that is the essence of learning.

14. Save the script and test it.

The script should execute as below.

```
PS>
PS>  .\2.5-analyze_sentence.ps1 'The quick brown fox jumps over the lazy dog'
The sentence passed is The quick brown fox jumps over the lazy dog
There are 9 words in the sentence
The first word is The
The last word is dog
The first three words in the sentence is The quick brown
The last three words in the sentence is the lazy dog
PS>
```