Lab Report - Lab 10

Name: Tyler Burleson Date: 4/1/2024

Summary

In this lab we focused on investigating Windows PowerShell and some of its programmatic capabilities. Through the course of this lab, we had several learning objectives: “use Simple Shell Commands,” “use Expressions,” explore the use of pipelines to stream data between commands, build statements to create program flow, practice formatting output, build functions, and create scripts using the previous skills. Each of the objectives built on top of each other to more advanced commands.

I went through this lab quickly since I have experience with PowerShell from my previous internships. I started with running simple commands such as “Ipconfig” and “pshome.” Additionally, I also used the command prompt to open other applications such as notepad.exe. Once I had warmed up, I moved to the next section of build expressions. During these tasks we practice using variables and concatenating strings in the command prompt. An example looked like “1> PS $arr = @("Hello There", "Goodbye") 2> PS $arr[1]”

After expressions I moved into using pipelines. Once we reached this stage, our cmdlets started to get more complex. One example from this section was “ PS get-process | where-object { $_.handlecount -gt 500 } .“ This function piped the results of get-process and filtered the results to only retrieve the objects with a handle count over 500. Now that we had seen pipelines, we were able to start creating some complex statements such as: conditionals, switch statements, and loops.

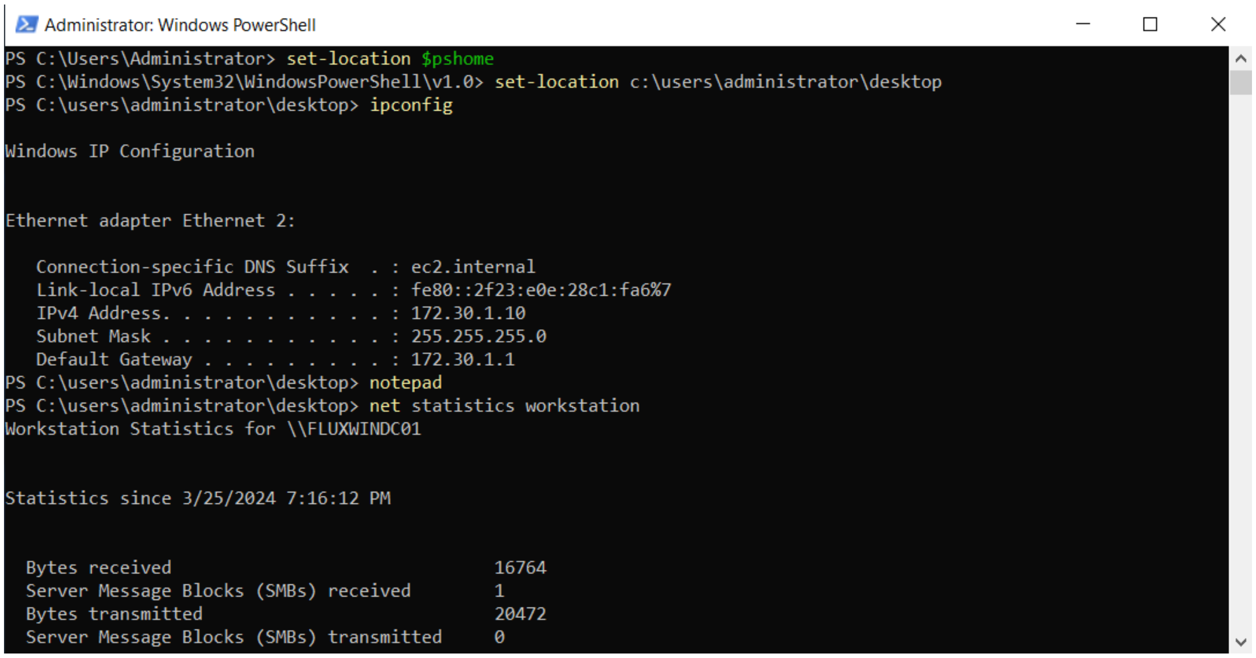
Once we reached exercise 6 we needed to review handling errors that may come from our complex statements. This process focused on three main mechanisms: retrieving the error from the system, recognizing whether the error is terminating or non-terminating, and using a trap. These three mechanisms work together for us to recognize when an error has occurred with our system, know if we need to stop the script running, and output a meaningful error message to the command line. Once we explored error handling, we then dove into formatting our output. The main options PowerShell provides us with are table view and list view. By using these commands, we can essentially create a customized view of the data we are working with.

Once we completed exercise seven, we were ready to move into building functions and scripting. We started by understanding how to create a function. We do this by calling the “function” command, passed in the name of our function, and finally in “{}” what the function was going to do. By creating a function, we minimalize the amount of repetitive code we must use inside our scripts. Now that we can use functions it was time to build our first script. This was a simple script that showed us how to use “Param()” to pass in necessary parameters, and how to call them in our script using “$varName.” Ultimately, we created a script that would take two strings from the user and output what the first and second string were.

Finally, to conclude our lab we practiced creating a remote connection using PowerShell. This was the only exercise that required us to use our domain controller since we were wanting to use our naming system to connect to the member server. The first task was to attempt to enter a PS session with one statement. We did this by calling the “Enter-PSSession” command, the name of our member server, and which account we were wanting to connect to. Once this statement ran we were provided a popup to enter the password for the credentials. Once we saw how to create the connection we turned this process into a script. Our script started by asking us the name of the server we wanted to connect to. Once we provided the name of the server it then prompted for the username we wanted to use to login. Finally, the script would take the user input and add them to the statement we ran earlier, and we were able to remote in.

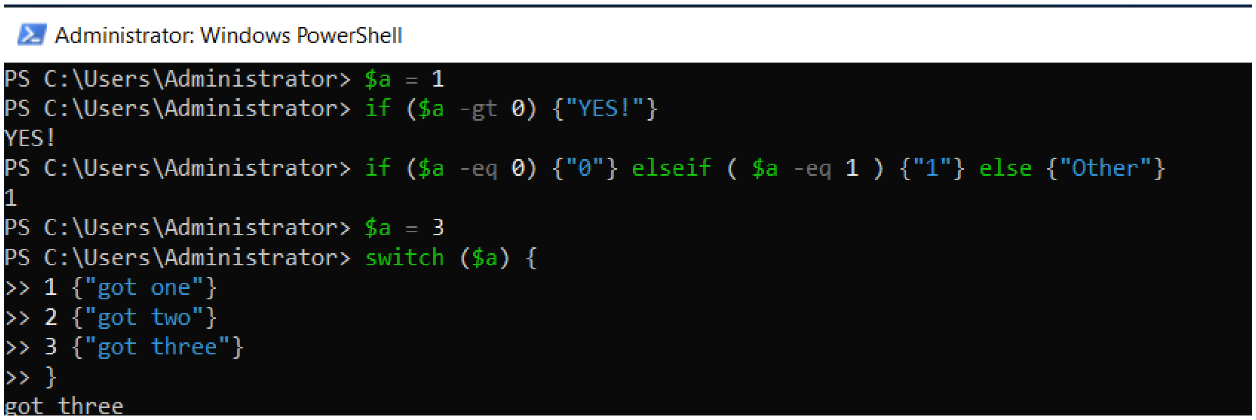
This entire lab was a review session for me, but I still feel that PowerShell isn't necessarily superior or inferior to Bash. The only way I see PowerShell being superior or different is its ability to share complex data instead of just strings like Bash. Additionally, PowerShell is slightly easier to learn starting out since its naming convention is closer to other languages.

Screen Shots



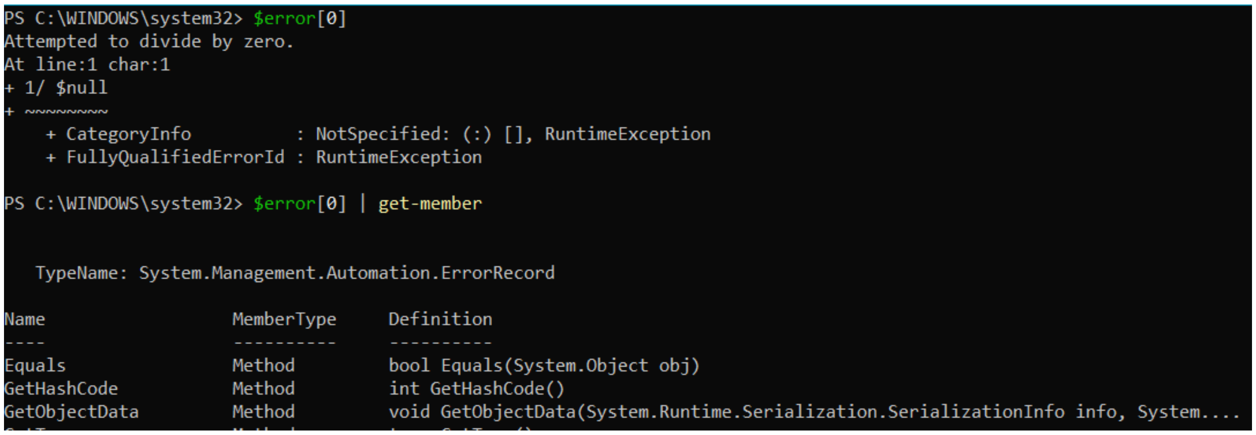
```
PS C:\Users\Administrator> set-location $pshome
PS C:\Windows\System32\WindowsPowerShell\v1.0> set-location c:\users\administrator\desktop
PS C:\users\administrator\desktop> ipconfig

Windows IP Configuration


Ethernet adapter Ethernet 2:

   Connection-specific DNS Suffix  . : ec2.internal
   Link-local IPv6 Address . . . . . : fe80::2f23:e0e:28c1:fa6%7
   IPv4 Address. . . . . . . . . . . : 172.30.1.10
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.30.1.1
PS C:\users\administrator\desktop> notepad
PS C:\users\administrator\desktop> net statistics workstation
Workstation Statistics for \\FLUXWINDC01


Statistics since 3/25/2024 7:16:12 PM


  Bytes received                               16764
  Server Message Blocks (SMBs) received        1
  Bytes transmitted                            20472
  Server Message Blocks (SMBs) transmitted     0
```

Figure 1



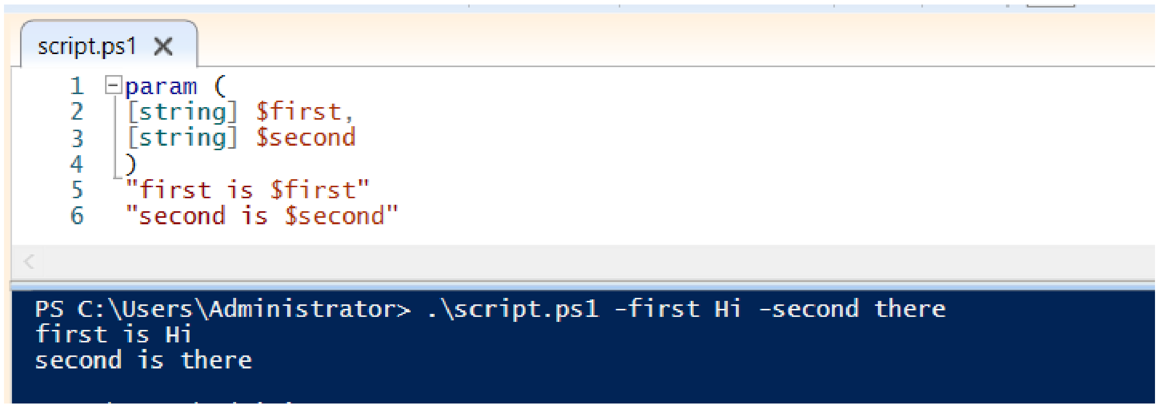
```
PS C:\Users\Administrator> $a = 1
PS C:\Users\Administrator> if ($a -gt 0) {"YES!"}
YES!
PS C:\Users\Administrator> if ($a -eq 0) {"0"} elseif ( $a -eq 1 ) {"1"} else {"Other"}
1
PS C:\Users\Administrator> $a = 3
PS C:\Users\Administrator> switch ($a) {
>> 1 {"got one"}
>> 2 {"got two"}
>> 3 {"got three"}
>> }
got three
```

Figure 2



```
PS C:\WINDOWS\system32> $error[0]
Attempted to divide by zero.
At line:1 char:1
+ 1/ $null
+ ~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [], RuntimeException
    + FullyQualifiedErrorId : RuntimeException

PS C:\WINDOWS\system32> $error[0] | get-member


   TypeName: System.Management.Automation.ErrorRecord

Name                  MemberType     Definition
----                  ----------     ----------
Equals                Method         bool Equals(System.Object obj)
GetHashCode           Method         int GetHashCode()
GetObjectData         Method         void GetObjectData(System.Runtime.Serialization.SerializationInfo info, System....
```

Figure 3



script.ps1

```
param (
[string] $first,
[string] $second
)
"first is $first"
"second is $second"
```

```
PS C:\Users\Administrator> .\script.ps1 -first Hi -second there
first is Hi
second is there
```

Figure 4

Research Questions

1. PowerShell runs what kind of commands? What does Microsoft call them?
    a. PowerShell uses commands called "cmdlets."
2. Which PowerShell control structure executes once before checking for an exit condition?
    a. PowerShell uses a "Do-While Loop" to executes once an exit condition is there.

3. In the screen shot above (Remote Connection > Task 2 > #3), what is/are the DNS servers forFluxWinMS01? If there's more than one, what can you deduce I have done with my domain?
    a. The DNS servers are 172.32.1.10 and 172.32.1.13. The administrator has configured multiple DNS servers.

4. What does the star, '*', in Get-ChildItem C:\Windows\System32 s*. * Mean? What will it do?
    a. The star is a wildcard. In this example it will provide any number and type of character after the "s" and "."

5. Can you launch a GUI-based application from PowerShell? What happens, if so?
    a. Yes, you can run a GUI application from the command line if you know the name of the executable -> notepad.exe

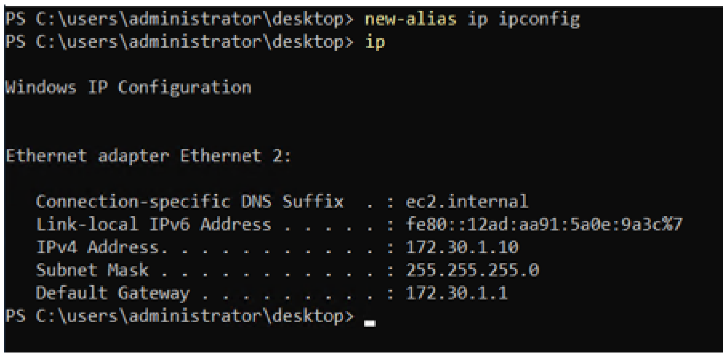
6. If you're testing against multiple potential conditions, which is the best control structure to use? Provide an example
    a. I would say it's a preference to the programmer, but several sources choose the ElseIf statement.
    b. If (cond1 = 1)
        i. { console.log(cond1)}
    c. Elseif (cond1 = 0)

7. Which of the formatting options will produce the following output for Get-ChildItem C:\Windows\System32 s*. *? In other words, entering Get-ChildItem C:\Windows\System32 s*. * What?
    a. It only returns files that begin with "s" and have any file extension.

8. Which PS cmdlet will paginate output?
    a. The "out-Host -Paging" will paginate the output.

9. Which loop construct will iterate through all of the elements in an array? Provide an example
    a. In PowerShell you can use the 'foreach' loop to iterate through all elements.
    b. $myArray = @(1, 2, 3, 4, 5)
    c. foreach ($element in $myArray) {
    d. # any additional code here, using $element
    e. Write-Host $element
    f. }

10. Create a new alias for ipconfig in PS. Capture a screenshot of its output. Will the alias persist over multiple PS sessions?



```
PS C:\users\administrator\desktop> new-alias ip ipconfig
PS C:\users\administrator\desktop> ip

Windows IP Configuration


Ethernet adapter Ethernet 2:

   Connection-specific DNS Suffix  . : ec2.internal
   Link-local IPv6 Address . . . . . : fe80::12ad:aa91:5a0e:9a3c%7
   IPv4 Address. . . . . . . . . . . : 172.30.1.10
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.30.1.1
PS C:\users\administrator\desktop>
```

   a. No it will not be saved.