****

**Win500 – Lab 3 – Making connections**

Rev: 08/05/2018

In this lab, you will complete the following;

1. Learn to use Invoke-Command
2. Pass Variables from local to remote machines
3. Run commands in a Disconnected Session

**Invoke-Command**

***Invoke-command*** is one of the methods used to connect PowerShell to remote systems. It is probably used more often than any other method because it can connect to multiple remote systems at the same time (one to many). By default, invoke-command can access ***32 simultaneous sessions***. This can be changed from a script, but be aware that you will use more system resources on the local machine, as you increase the number.

*Invoke-command* is used when you need to send a single command or script to one or more systems. PowerShell will open the sessions(s), execute the script then close the sessions(s).

Basic Structure

**Invoke-command –computername <computer> -scriptblock{command or script}**

When entering a **-computername <computer>** , do not type in the brackets.

There are many options available for the Invoke-command. We will use some of them, but for more information, access the help system.: **Get-help Invoke-command**

**Make sure your Workstation and other VMs are running. Run all scripts from AD VM**

**Single Named System**

#output all of the services currently listed on a remote system

#Replace the –computername with one of your systems names

**Invoke-command –computername <computer> -scriptblock {get-service}**

What is the difference in the output of this command compared to running?

“Get-service” on its own?

\_\_Same output but when running with the different computer name, it will return the output of the computer which command is running on\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Multiple Named Systems**

#output all of the services currently listed in multiple systems

#replace the computernames with your own

**Invoke-command –computername <computer>, <computer> –scriptblock{get-service}**

**Note: This method is OK, when looking at a few remote systems. It is impractical when looking at**

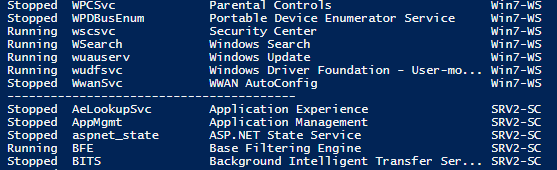
**a large number of systems**

a) **Computers contained in an Array**

# Write a script using the invoke command which will get the running services from the computers names below and separate each computer’s output with a dash line. Your output should look like the screen shot below:

**cls**

**$a = "SRV2-FS", "Win7-WS"**



b) Try this second example

#place computer names in an array

**$a = "SRV2-FS", "Win7-WS"**

#copy your command to view running services below and assign it to a variable $data

**$data = (command from a)**

#output the contents of the variable

**$data**

Look at the output of the script.

Now type in the **$data | Get-Member**

1. What type of object was returned by WinRM?

\_\_\_\_\_\_\_Array\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. On what PSDrive of the localhost is the variable $data stored? \_\_\_\_\_\_\_\_local drive\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Now repeat the exercise, save everything in the scriptblock to $data variable and run the command again?

Was anything returned to the localhost? \_\_\_\_\_\_\_\_\_no, it saved to variable data\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Go to the Win7-WS VM and type **gci variable:data . Since we saved the output to the remote system by placing $data inside the scriptblock, can you think of a reason why the data variable does not exist?**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

c) **Computer Names Read From a Text File**

# using notepad, create a text file named *System.txt*

#type in the computer names of your systems, each on its own line

#save the file to the folder/path indicated in the script below

#execute Get-service on computers listed in a text file

**Invoke-command –computername (Get-content C:\Temp\System.txt) –scriptblock {gsv}**

**Note: In this example, Invoke-command automatically reads each computer name using a**

**built-in ForEach loop**

This is a better method for handling a large number of remote systems, but it still has some serious disadvantages. If a computer is removed from the domain or its name has been changed, the list

must also be amended. When several hundred or even thousands of systems exist in a domain,

managing the list becomes a full time job.

There is an even better way to deal with a large number of remote systems. It is called

Active Directory.

**Computer Names Read From Active Directory**

First let’s see what Active Directory has for us. The comments will explain what is going on.

Run the following commands, one at a time, to see what happens;

# Ask Active Directory to send information about all of the computers it currently

# knows about. Notice we do not need Invoke-command for this. Use the filter option and the

wildcard to get all of the systems

**Get-ADcomputer –filter \***

# Too much data. We need to get the computer names and dump the rest.

**Get-adcomputer –filter \* | Select –expand name**

#Ok, now to incorporate this into Invoke-command

**Invoke-command –computername (get-adcomputer –filter \*) | select –expand name)`**

**-scriptblock {get-service}**

**Note:** While this method will allow you to run commands/scripts on multiple remote systems,

it still has some disadvantages. Errors will occur if the remote computer is not available. Using *Test-*

*Connection* to determine if a system is down will resolve these errors.

**Executing Local Script File to a Remote System**

#Open PowerShellISE and type in the following

**CLS**

**GCI c:\**

#Save the file to c:\temp\script.ps1

#On the command line execute the following, replacing the –computername with one of your virtual #systems

**Invoke-command –computername <computer> -filepath C:\temp\script.ps1**

If you received a big red error message, how would you fix the error to allow

the script to execute properly?

\_\_\_\_\_\_\_\_\_\_\_\_\_Set-executionPolicy Unrestricted\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Placing Remote data into a Variable/File**

Invoke-command will execute scripts against remote systems. If the script returns values,

they are typically displayed on the screen. We can, if necessary, place the data into a

variable or file.

1. **Save to variable**

#place returned data into a variable

**$data = Invoke-command –computername <computer> -scriptblock{get-service}**

$output the result to the screen. Too much data

**Write-host $data**

#save to file, using the variable

**$data > C:\temp\service.txt**

1. **Save to File**

There are a variety of methods to get data to a file.

#Send to file using redirection

**Invoke-command –computername <computer> -scriptblock{get-service}> c:\temp\service.txt**

1. **Save to Web Page**

**#send to web page**

**Invoke-command –computername <computer> -scriptblock{get-service} | convertto-html |`**

**Set-content C:\temp\service.html**

Use the preContent and PostContent parameters of ConvertTo-HTML to add a header and trailer to the table, as shown in lecture.

**Pass functions/local variables to remote systems**

You may be required to pass variables values to a remote system. Since your script is running

on the remote system, it cannot access your local functions or variable values. Therefore, you must

send them at the same time you activate Invoke-command. Try out the following.

1. Create 2 local variables one called name and the other called number. Set the value of number to 10 and the name set to “System”.
2. Create a persistent connection to the localhost and store in a variable $S1 (you may need credentials)
3. Use the invoke-command and the session variable to retrieve the 10 newest system eventlogs on the local system. What command would you type, if you are using the $using keyword?  
   1. \_\_\_\_\_\_invoke-command –computer localhost –scriptblock {get-eventlog –Logname $using:name –newest $using:number} –credential\_Administrator\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
        
      \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
4. Repeat the command except this, use the PARAM keyboard. What command would you type?
   1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Now convert the command to a function called Get-events and save the function with the parameters $name and $number inside the scriptblock. Save the file in the Administrator home directory. What command would you type?

\_\_\_\_\_\_\_\_\_\_ function Get-events ($name, $number)

{

Invoke-Command -ComputerName localhost -ScriptBlock {Get-EventLog -LogName $name -Newest $number} | Out-File $home\file.txt

}

* 1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
       
     \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Confirm that your command in 5) will work in a remote session by creating a persistent system with a VM and storing the session in a variable. Use the VM name in the computername parameter. Did the command work? \_\_\_yes\_\_\_\_\_\_\_\_\_\_ (if not make changes to make it work)
2. Use Remove-PSSession \* to remove all previous local and remote sessions.

**Questions:**

1. *Invoke-command* can be used to connect to thousands of systems, but by default it will only open a limited number of connections at the same time. This number can be increased if the system running Invoke-command has enough resources.

What is the default number of open connections PowerShell can access? \_\_\_**32**\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What is the parameter to change the number of default connections? \_\_\_\_\_-**ThrottleLimit**\_\_\_\_\_\_\_\_\_\_\_\_

1. What are the 2 protocols and port numbers PowerShell (through WinRm) uses to connect to remote systems?

Protocol: \_\_\_TCP (HTTP)\_\_\_\_\_\_\_\_\_\_\_\_ Port: \_\_\_\_5985\_\_\_\_\_\_\_\_\_\_\_\_\_

Protocol: \_\_\_TCP(HTTPS)\_\_\_\_\_\_\_\_\_\_\_\_ Port: \_\_\_\_\_5986\_\_\_\_\_\_\_\_\_\_\_\_

1. Now we will test the InDisconnecteSession parameter. This parameter is used in two situations:

* When long running scripts need to be sent to multiple computers (one to many)
* When you want the results from multiple computers to be send to one computer (many to one)

Type the following in SRV1-AD: (make sure you have computers.txt file in the current directory)

**invoke-command -computername (gc .\computers.txt) -scriptblock {$host;get-date} -InDisconnectedSession**

What are the Session names? \_\_\_\_**Session 200**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What is the Session State? \_\_\_\_\_\_\_**Disconnected**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
4) Which of the following is true when the InDiscconectedSession is enabled with the invoke-command?

1. PowerShell opens a session, runs the command and then sets the session state to disconnected
2. **PowerShell opens a session, sets the session state to disconnected and then runs the command**
3. PowerShell opens a session, partially runs the command and then sets the state to disconnected
4. PowerShell opens a session, sets the session state to disconnected and waits for future commands

(If you don’t know the answer use get-help to understand how the InDisconnectedSession parameter works)

What command is used to get the output of a disconnected session? \_\_\_\_**receive-pssession**\_\_\_\_\_\_\_\_\_\_

**Exercise**

**Part 1**

On your AD VM, open PowerShellISE and create a script that will do the following;

1. Get all the computer names from Active Directory and output only the names of the computers to a file called **ADcomputers.txt**

**Part 2**

1. Write a script called **Lab3\_SetRestrictedPolicy.ps1** to do the following:

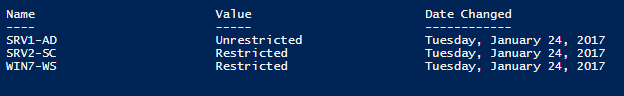
* Test to see if the computername is SRV1-AD and if true, set the execution policy to unrestricted
* For the other computers, test to see if the execution policy is restricted and if false, set the execution policy to true.
* Use a parameter to suppress the automatic prompts
* Write each line of output as a string <computername>=<policy setting>.

**Part 3**

1. Write a second script called **Lab3\_InvokeSetExecutionPolicyEx3.ps1**. This script will do the following

* Use the Invoke command to connect to the computers in the ADcomputers list
* Use the Lab3\_SetRestrictedPolicy script to check and to set execution policy for each computer
* Save the output from Part 2 to a variable on the AD server
* Convert the string variable from Part 2 to a hash table and pipe to
* The format-table cmdlet to show the properties Name, Value and a new calculated property called, “Date Changed”, showing the day’s date in long format.

Your output should look like the screen shot below. Name your file **Lab3\_ExecutionPolicy.jpeg**



**Don’t forget to change the execution policy back to RemoteSigned*.***

**Grading:**

* **LearnName\_Lab3\_SetRestrictedPolicy.ps1**
* **LearnName\_Lab3\_ExecutionPolicy.jpeg**
* **LearnName\_Lab3\_InvokeSetExecutionPolicyEx3.ps1**
* **LearnName\_Lab3\_MakingConnections.docx**