Advanced Scripting   
Remoting Sessions

Last Updated: 4/18/2024 9:36 AM Version 1  
Document Prepared for: CYBER360 Student

# Name Click here to enter name ID Click here to enter id

# Instructions

Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

When you use **Invoke-Command** with a **ComputerName**, the command first creates a PowerShell instance on the remote machine, then passes the script block to the remote machine for processing. The results are passed back to the calling machine. When the script is done the PowerShell instance on the remote machine is terminated. If you use **Invoke-Command** frequently, such as in a loop, many instances of PowerShell are created and destroyed. Effectively, the remote target does a lot of extra work to satisfy these invocations. You can avoid this overhead by creating a PowerShell *session*. When the session is first started, an instance of PowerShell is started on the remote machine. That instance is reused as long as the session exists. Sessions exist until the user closes them, or the calling shell is terminated.

# Requirements

* SSH client (There is already a command-line OpenSSH client available in current versions of Windows, macOS, and Linux)
* Internet access

# Setup

Use SSH as a remote access tool to connect to the lab:  
Computer: **cit361-lab.cit.byui.edu**Port: **443** *(Important: for SSH, this is a* nonstandard *transport layer port number.)*  
Username: The mailbox portion of your BYU-I email address, usually three letters and five numbers. (Example: If your BYU-I email is **mer23079@byui.edu** you would use **mer23079** for your username.)  
Password: Your I-Number.

If you are using OpenSSH from a CLI, the parameters above can be specified like this:  
**ssh -p 443 mer23079@cit361-lab.cit.byui.edu**   
If you successfully type the correct password when prompted, you will see a prompt like this:  
PS C:\Users\mer23079>

# Task 1—Interactive Remote PowerShell: **Enter-Session**

An interactive session is useful for directly operating a remote machine.

## Steps

1. After using SSH to log in, enter the **hostname** command. When you connected remotely from the Internet, this host was identified by the name **cit361-lab.cit.byui.edu**. Now that you are logged in, by what name does this machine identify itself on its *local* subnet? Click or tap here to enter text. *This should be the same as the environment variable* **$Env:COMPUTERNAME**. It is common for a *bastion host* to have different identifiers for different networks.
2. For the rest of this exercise, we will refer to this bastion host as the *local* machine. Enter:   
   **powershell -NoLogo**   
   to launch a new Windows PowerShell *child process* on the local machine. *(The* **-NoLogo** *switch parameter is optional. It merely turns off the message that otherwise gets displayed when a PowerShell interactive session starts up.)*
3. Connect to the virtual machine named DC:   
   Enter-PSSession DC
4. You should now see a prompt that starts with **[DC]**. This indicates you are using an *interactive* remote session. Verify you really are running on DC by entering one of these two commands:  
   $Env:COMPUTERNAME   
   or  
   hostname   
   (Your output should show **DC**.)
5. So anything you do in this session will be executed on DC. Get a directory of its filesystem root:   
   dir \
   1. What directories are listed? Use a **-join** operator to connect them all on one line separated by commas, then paste your result here: Click or tap here to enter text.
6. To terminate the interactive remote session just type exit:   
   exit   
   Then exit again to terminate your Windows PowerShell child process.

# Task 2—Persistent (kind of) Session objects

You can create a PowerShell session *object* that can be reused.

## Steps

1. Create a *new* Windows PowerShell child process:   
   **powershell**   
   Then create a remote session to DC:   
   $dc=New-PSSession -ComputerName dc
   1. Send the **GetType()** message to the object stored in the variable:   
      **$dc.GetType()**   
      What is its abstract class **Name**? Click or tap here to enter text.
   2. Enter   
      **$dc.GetType().FullName**   
      What is the **FullName** of its class? Click or tap here to enter text.
2. View your session:   
   Get-PSSession
   1. How many sessions are shown? Click or tap here to enter text.
   2. What is the **Id** number of the session you just created? Click or tap here to enter text.
   3. What is the **Name** of the session you just created? Click or tap here to enter text.
3. Start another session, this time to a different lab VM:   
   $sg=New-PSSession -ComputerName slaygore
4. View all of your sessions:   
   **Get-PSSession**
   1. How many sessions are shown? Click or tap here to enter text.
   2. What is the **Name** of the session you just created? Click or tap here to enter text.
5. Send a command to one of the sessions:   
   Invoke-Command -Session $dc -ScriptBlock {Get-Date}
6. Send a command to multiple sessions, all in one command line:   
   invoke-command -session $dc,$sg -scriptblock {"The time on $($env:computername) is $(get-date)"}
   1. Is the time the same on both machines? If not, what’s different? Click or tap here to enter text.
7. Close your Windows PowerShell child process.
8. Open *another* Windows PowerShell child process. View the sessions:   
   Get-PSSession   
   Are the sessions still there? Click or tap here to enter text.
9. Type **exit** to close your Windows PowerShell child process.

# Task 3—Disconnecting/Reconnecting to a Persistent (sort of) Session

You can disconnect from a PowerShell session, and it will remain persistent.

## Steps

1. Start *another* Windows PowerShell child process, and in it enter   
   Invoke-Command -ComputerName dc -ScriptBlock {$Answer=42}   
   to create a variable named **Answer** *remotely* on the **DC** VM and initialize it to the value **42**.
2. Try to get the value from **Answer**:   
   Invoke-Command -ComputerName dc -ScriptBlock {"Answer is $Answer"}
   1. Your output: Click or tap here to enter text.
   2. So what is the value contained in Answer on DC? Enter:  
      Invoke-Command -ComputerName dc -ScriptBlock {gci Variable:Answer}   
      Your output: Click or tap here to enter text.
   3. In your own words, what happened to the Answer you created in step 1? Click or tap here to enter text.
3. Get a list of sessions. Are there any sessions? Click or tap here to enter text.
4. Create a new session to DC:   
   $dc=New-PSSession -ComputerName DC
5. Set the $Answer variable again, but this time use the session:   
   Invoke-Command -Session $dc -ScriptBlock {$Answer=42}
6. Read the value using the session:   
   Invoke-Command -session $dc -ScriptBlock {"Answer is $Answer"}
   1. What value is in Answer?.Click or tap here to enter text.
   2. Explain: Click or tap here to enter text.
7. Disconnect the session  
   Disconnect-PSSession $dc   
   Then get a list of sessions. Are there any sessions? Click or tap here to enter text.
8. Reconnect to the session:   
   $dc=Connect-PSSession -ComputerName DC   
   Then get the value of Answer:   
   Invoke-Command -Session $dc -ScriptBlock {"Answer is $Answer"}
   1. What value is in Answer? Click or tap here to enter text.
   2. Explain: Click or tap here to enter text.
9. Terminate the session.  
   Remove-PSSession $dc   
   Then again **exit** your Windows PowerShell child process.

# Task 4—Importing Commands from a Remote Session

Now you will make a command on a remote system. Then you will *import* it, which will make it appear as if it is on the local system, but it will still execute on the remote system.

## Steps

1. Launch yet another Windows PowerShell child process.
2. Again, create a session on DC:   
   $dc=new-pssession -computername dc
3. Use the session to define a command (a PowerShell function) on the remote machine. Type all of this on one command line:   
   invoke-command $dc -scriptblock {function who{"Hi, I'm $($env:computername)"}}
4. Prove the function does not exist on the local machine:   
   who
5. Prove the function exists on the remote machine:   
   invoke-command $dc {who}
6. Import that function to the local machine:   
   Import-PSSession $dc -CommandName who
7. Try it on the local machine:   
   who   
   Was it executed on the local machine or the remote machine? Click or tap here to enter text.
8. Clean up:   
   Remove-PSSession $dc
9. Prove the imported command is gone:   
   who
10. Close your Windows PowerShell child process.

# Task 5—PowerShell Core Sessions

PowerShell Core supports multiple transport mechanisms for remoting.

## Steps

1. Enter:   
   **pwsh**   
   to launch a new PowerShell Core for Windows *child process* on the local machine.
2. Use one command line to create remote sessions to two of the lab VMs:   
   $dc,$sg=New-PSSession -ComputerName dc,slaygore
   1. Look at the object in $dc:   
      **$dc**   
      Look at the object in $sg:   
      **$sg**   
      Were they assigned in the expected order? If not, swap them:   
      **$dc,$sg = $sg,$dc**
   2. Does the type of the object in **$dc** have the same **FullName** that your PSSesson object had in Task 2 step 1.2? Click or tap here to enter text.
3. View all your sessions:   
   Get-PSSession
   1. How many sessions are shown? Click or tap here to enter text.   
        
      (*For the following two steps, enter your answers all on one line separated by commas*.)
   2. What are the **Id** numbers of the sessions you just created? Click or tap here to enter text.
   3. What are the **Name**s of these sessions? Click or tap here to enter text.
   4. Compare/contrast these from the **Name** in Task 2 step 2: Click or tap here to enter text.
   5. What is the **Transport** of the sessions you just created? Click or tap here to enter text.
4. Send a command pipeline to *each* of these two new sessions:   
   invoke-command -session $dc,$sg -scriptblock {get-service|measure}
   1. For each VM, list its name and the number of services configured on it: Click or tap here to enter text.
5. Now create another remote session, this time to the VM named eric:   
   $e=New-PSSession -HostName eric -UserName ps   
   (When prompted, enter the password *PowerShellRocks!*) Using the **-HostName** parameter instead of **-ComputerName** signals the cmdlet to use *Secure Shell (SSH)* as its transport mechanism.
6. Get a list of sessions.
   1. What is the **Transport** type of the session to eric? Click or tap here to enter text.
   2. What is this session’s **Name**? Click or tap here to enter text.
   3. What is this session’s **Id**? Click or tap here to enter text.

# Task 6—PowerShell Core Linux Remoting

PowerShell Core for Linux also executes Linux commands.

1. View eric’s PowerShell info:   
   **invoke-command -session $e -scriptblock {$psversiontable}**
   1. What version of PowerShell is running on eric? Click or tap here to enter text.
   2. What is eric’s OS? Click or tap here to enter text.
   3. Identify the edition and version of PowerShell on eric: Click or tap here to enter text.
2. Execute a PowerShell command alias on eric:   
   invoke-command -Session $e -ScriptBlock {gci}   
   Execute a bash command on eric:   
   invoke-command -Session $e -ScriptBlock {ls -l}   
   In your own words, explain your outputs: Click or tap here to enter text.
3. Use the session in **$e** to connect interactively:   
   **Enter-PSSession $e**   
   Notice the prompt now includes both username and machine name. Explore the machine using PowerShell commands such as **Get-Location**, **Set-Location**, **Get-ChildItem**, **Get-Process**, etc. (Or, if you prefer, explore using equivalent bash commands: **pwd**, **cd**, **ls**, **ps**, etc.)
   1. What is the process ID of your PowerShell Core process on eric? Click or tap here to enter text.
   2. Use the bash commands **who** and **w** to see who is currently logged into eric. Did the Linux operating system identify your PowerShell session as a logged-in interactive user? Click or tap here to enter text.
   3. In your own words, explain the security implications of this observation: Click or tap here to enter text.
4. When you are finished exploring, use **exit** to close the interactive session. Now that you are back in your PowerShell Core child process, get the list of sessions again. Was the session to eric closed? Click or tap here to enter text.
5. As of PowerShell Core version 7.0.2 and later, you can also start SSH interactive sessions directly:   
   **Enter-PSSession -HostName eric -UserName ps**
   1. In your new interactive session, enter the Linux bash command   
      **ps fax**   
      This produces a list of the running process in a *tree* format, which makes it a little easier to see parent-child relationships between processes, and determine which processes were responsible for launching other processes. Look carefully at your output and find the **ps fax** process that you just executed. What was the Process ID (PID) of that process? Click or tap here to enter text.
   2. What was the PID of *your* interactive **pwsh** (the “parent” of **ps fax**, or in other words, the PID of the process with which you spawned **ps fax**)? Click or tap here to enter text.
   3. What is the command line of the “grandparent” process, or in other words, the process that spawned your interactive pwsh? Click or tap here to enter text.
   4. Look at the PIDs of the other running **pwsh** processes. Do any of them match the ID of the persistent PowerShell session you identified in step 6.1? Click or tap here to enter text.
   5. Type **exit** to leave your interactive session to eric.
6. Use the session in **$sg** to connect interactively to slaygore:   
   Enter-PSSession $sg
   1. What path is your initial working directory ? Click or tap here to enter text.
   2. Type **exit** to leave your interactive session to slaygore.
7. Close all of your sessions:   
   Remove-PSSession $dc, $sg, $e
8. Exit your PowerShell Core child process, then exit your connection to **cit361-lab.cit.byui.edu**:  
   exit   
   exit

# Deliverable

Upload this document with completed answers to I-Learn Canvas.