





*Full credit is given to the above companies including the OS that this PDF file was generated!*

## *PowerShell Get-Help on command 'Invoke-Command'*

***PS C:\Users\wahid> Get-Help Invoke-Command***

NAME

Invoke-Command

SYNOPSIS

Runs commands on local and remote computers.

SYNTAX

Invoke-Command [[-ConnectionUri] <System.Uri[]>] [-ScriptBlock]
<System.Management.Automation.ScriptBlock> [-AllowRedirection] [-ArgumentList
<System.Object[]>] [-AsJob] [-Authentication {Default | Basic | Negotiate |
NegotiateWithImplicitCredential | Credssp | Digest | Kerberos}]
[-CertificateThumbprint <System.String>] [-ConfigurationName <System.String>]
[-Credential <System.Management.Automation.PSCredential>]
[-EnableNetworkAccess] [-HideComputerName] [-InDisconnectedSession]
[-InputObject <System.Management.Automation.PSObject>] [-JobName
<System.String>] [-SessionOption
<System.Management.Automation.Remoting.PSSessionOption>] [-ThrottleLimit
<System.Int32>] [<CommonParameters>]

Invoke-Command [[-ConnectionUri] <System.Uri[]>] [-FilePath] <System.String>

[-AllowRedirection] [-ArgumentList <System.Object[]>] [-AsJob]

[-Authentication {Default | Basic | Negotiate |

NegotiateWithImplicitCredential | Credssp | Digest | Kerberos}]

[-ConfigurationName <System.String>] [-Credential

<System.Management.Automation.PSCredential>] [-EnableNetworkAccess]

[-HideComputerName] [-InDisconnectedSession] [-InputObject

<System.Management.Automation.PSObject>] [-JobName <System.String>]

[-SessionOption <System.Management.Automation.Remoting.PSSessionOption>]

[-ThrottleLimit <System.Int32>] [<CommonParameters>]

Invoke-Command [[-ComputerName] <System.String[]>] [-FilePath] <System.String>

[-ApplicationName <System.String>] [-ArgumentList <System.Object[]>] [-AsJob]

[-Authentication {Default | Basic | Negotiate |

NegotiateWithImplicitCredential | Credssp | Digest | Kerberos}]

[-ConfigurationName <System.String>] [-Credential

<System.Management.Automation.PSCredential>] [-EnableNetworkAccess]

[-HideComputerName] [-InDisconnectedSession] [-InputObject

<System.Management.Automation.PSObject>] [-JobName <System.String>] [-Port

<System.Int32>] [-SessionName <System.String[]>] [-SessionOption

<System.Management.Automation.Remoting.PSSessionOption>] [-ThrottleLimit

<System.Int32>] [-UseSSL] [<CommonParameters>]

Invoke-Command [[-ComputerName] <System.String[]>] [-ScriptBlock]

<System.Management.Automation.ScriptBlock> [-ApplicationName <System.String>]

[-ArgumentList <System.Object[]>] [-AsJob] [-Authentication {Default | Basic |

Negotiate | NegotiateWithImplicitCredential | Credssp | Digest | Kerberos}]

[-CertificateThumbprint <System.String>] [-ConfigurationName <System.String>]

[-Credential <System.Management.Automation.PSCredential>]

[-EnableNetworkAccess] [-HideComputerName] [-InDisconnectedSession]

[-InputObject <System.Management.Automation.PSObject>] [-JobName

<System.String>] [-Port <System.Int32>] [-SessionName <System.String[]>]

[-SessionOption <System.Management.Automation.Remoting.PSSessionOption>]

[-ThrottleLimit <System.Int32>] [-UseSSL] [<CommonParameters>]

Invoke-Command [-ScriptBlock] <System.Management.Automation.ScriptBlock>
[[-Session] <System.Management.Automation.Runspaces.PSSession[]>]
[-ArgumentList <System.Object[]>] [-AsJob] [-HideComputerName] [-InputObject
<System.Management.Automation.PSObject>] [-JobName <System.String>]
[-ThrottleLimit <System.Int32>] [<CommonParameters>]

Invoke-Command [[-Session]
<System.Management.Automation.Runspaces.PSSession[]>] [-FilePath]
<System.String> [-ArgumentList <System.Object[]>] [-AsJob] [-HideComputerName]
[-InputObject <System.Management.Automation.PSObject>] [-JobName
<System.String>] [-ThrottleLimit <System.Int32>] [<CommonParameters>]

Invoke-Command [-ScriptBlock] <System.Management.Automation.ScriptBlock>
[-VMId] <System.Guid[]> [-ArgumentList <System.Object[]>] [-AsJob]
[-ConfigurationName <System.String>] [-Credential
<System.Management.Automation.PSCredential>] [-HideComputerName] [-InputObject
<System.Management.Automation.PSObject>] [-ThrottleLimit <System.Int32>]
[<CommonParameters>]

Invoke-Command [-ScriptBlock] <System.Management.Automation.ScriptBlock>
[-ArgumentList <System.Object[]>] [-AsJob] [-ConfigurationName
<System.String>] [-Credential <System.Management.Automation.PSCredential>]
[-HideComputerName] [-InputObject <System.Management.Automation.PSObject>]
[-ThrottleLimit <System.Int32>] -VMName <System.String[]> [<CommonParameters>]

Invoke-Command [-VMId] <System.Guid[]> [-FilePath] <System.String>
[-ArgumentList <System.Object[]>] [-AsJob] [-ConfigurationName
<System.String>] [-Credential <System.Management.Automation.PSCredential>]
[-HideComputerName] [-InputObject <System.Management.Automation.PSObject>]
[-ThrottleLimit <System.Int32>] [<CommonParameters>]

Invoke-Command [-FilePath] <System.String> [-ArgumentList <System.Object[]>]

[-AsJob] [-ConfigurationName <System.String>] [-Credential
<System.Management.Automation.PSCredential>] [-HideComputerName] [-InputObject
<System.Management.Automation.PSObject>] [-ThrottleLimit <System.Int32>]
-VMName <System.String[]> [<CommonParameters>]

Invoke-Command [-ScriptBlock] <System.Management.Automation.ScriptBlock>
[-ArgumentList <System.Object[]>] [-AsJob] [-ConfigurationName
<System.String>] -ContainerId <System.String[]> [-HideComputerName]
[-InputObject <System.Management.Automation.PSObject>] [-JobName
<System.String>] [-RunAsAdministrator] [-ThrottleLimit <System.Int32>]
[<CommonParameters>]

Invoke-Command [-FilePath] <System.String> [-ArgumentList <System.Object[]>]
[-AsJob] [-ConfigurationName <System.String>] -ContainerId <System.String[]>
[-HideComputerName] [-InputObject <System.Management.Automation.PSObject>]
[-JobName <System.String>] [-RunAsAdministrator] [-ThrottleLimit
<System.Int32>] [<CommonParameters>]

Invoke-Command [-ScriptBlock] <System.Management.Automation.ScriptBlock>
[-ArgumentList <System.Object[]>] [-InputObject
<System.Management.Automation.PSObject>] [-NoNewScope] [<CommonParameters>]

DESCRIPTION

The `Invoke-Command` cmdlet runs commands on a local or remote computer and returns all output from the commands, including errors. Using a single `Invoke-Command` command, you can run commands on multiple computers.

To run a single command on a remote computer, use the ComputerName parameter. To run a series of related commands that share data, use the `New-PSSession` cmdlet to create a PSSession (a persistent connection) on the remote computer, and then use the Session parameter of `Invoke-Command` to run the command in the PSSession . To run a command in a disconnected session, use the

InDisconnectedSession parameter. To run a command in a background job, use the AsJob parameter.

You can also use `Invoke-Command` on a local computer to a run script block as a command. PowerShell runs the script block immediately in a child scope of the current scope.

Before using `Invoke-Command` to run commands on a remote computer, read about_Remote (./About/about_Remote.md).

Some code samples use splatting to reduce the line length. For more information, see about_Splatting (./About/about_Splatting.md).

PARAMETERS

-AllowRedirection <System.Management.Automation.SwitchParameter>

Allows redirection of this connection to an alternate Uniform Resource Identifier (URI).

When you use the ConnectionURI parameter, the remote destination can return an instruction to redirect to a different URI. By default, PowerShell doesn't redirect connections, but you can use this parameter to allow it to redirect the connection.

You can also limit the number of times the connection is redirected by changing the MaximumConnectionRedirectionCount session option value. Use the MaximumRedirection parameter of the `New-PSSessionOption` cmdlet or set the MaximumConnectionRedirectionCount property of the `$PSSessionOption` preference variable. The default value is 5.

-ApplicationName <System.String>

Specifies the application name segment of the connection URI. Use this parameter to specify the application name when you aren't using the

ConnectionURI parameter in the command.

The default value is the value of the `$PSSessionApplicationName` preference variable on the local computer. If this preference variable isn't defined, the default value is WSMAN. This value is appropriate for most uses. For more information, see about_Preference_Variables (./About/about_Preference_Variables.md).

The WinRM service uses the application name to select a listener to service the connection request. The value of this parameter should match the value of the URLPrefix property of a listener on the remote computer.

-ArgumentList <System.Object[]>

Supplies the values of parameters for the scriptblock. The parameters in the script block are passed by position from the array value supplied to ArgumentList . This is known as array splatting. For more information about the behavior of ArgumentList , see about_Splatting (about/about_Splatting.md#splatting-with-arrays).

-AsJob <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet runs the command as a background job on a remote computer. Use this parameter to run commands that take an extensive time to finish.

When you use the AsJob parameter, the command returns an object that represents the job, and then displays the command prompt. You can continue to work in the session while the job finishes. To manage the job, use the `*-Job` cmdlets. To get the job results, use the `Receive-Job` cmdlet.

The AsJob parameter resembles using the `Invoke-Command` cmdlet to run a `Start-Job` cmdlet remotely. However, with AsJob , the job is created on the local computer, even though the job runs on a remote computer. The results of the remote job are automatically returned to the local computer.

For more information about PowerShell background jobs, see about_Jobs (About/about_Jobs.md) and [about_Remote_Jobs](About/about_Remote_Jobs.md).

-Authentication <System.Management.Automation.Runspaces.AuthenticationMechanism>
Specifies the mechanism that's used to authenticate the user's credentials. CredSSP authentication is available only in Windows Vista, Windows Server 2008, and later versions of the Windows operating system.

The acceptable values for this parameter are as follows:

- Default

- Basic

- Credssp

- Digest

- Kerberos

- Negotiate

- NegotiateWithImplicitCredential

The default value is Default.

For more information about the values of this parameter, see AuthenticationMechanism Enumeration (/dotnet/api/system.management.automation.runspaces.authenticationmechanism).
> [!CAUTION] > Credential Security Support Provider (CredSSP)

authentication, in which the user's credentials are > passed to a remote computer to be authenticated, is designed for commands that require > authentication on more than one resource, such as accessing a remote network share. This mechanism > increases the security risk of the remote operation. If the remote computer is compromised, the > credentials that are passed to it can be used to control the network session. For more > information, see > Credential Security Support Provider (/windows/win32/secauthn/credential-security-support-provider).

-CertificateThumbprint <System.String>

Specifies the digital public key certificate (X509) of a user account that has permission to connect to the disconnected session. Enter the certificate thumbprint of the certificate.

Certificates are used in client certificate-based authentication. They can be mapped only to local user accounts and they don't work with domain accounts.

To get a certificate thumbprint, use a `Get-Item` or `Get-ChildItem` command in the PowerShell Cert: drive.

-ComputerName <System.String[]>

Specifies the computers on which the command runs. The default is the local computer.

When you use the ComputerName parameter, PowerShell creates a temporary connection that's used only to run the specified command and is then closed. If you need a persistent connection, use the Session parameter.

Type the NETBIOS name, IP address, or fully qualified domain name of one or more computers in a comma-separated list. To specify the local computer, type the computer name, localhost, or a dot (`.`).

To use an IP address in the value of ComputerName , the command must include the Credential parameter. The computer must be configured for the HTTPS transport or the IP address of the remote computer must be included in the local computer's WinRM TrustedHosts list. For instructions to add a computer name to the TrustedHosts list, see [How to Add a Computer to the Trusted Host List](./about/about_remote_troubleshooting.md#how-to-add-a-computer-to-the-trusted-hosts-list).

On Windows Vista and later versions of the Windows operating system, to include the local computer in the value of ComputerName , you must run PowerShell using the Run as administrator option.

-ConfigurationName <System.String>

Specifies the session configuration that is used for the new PSSession .

Enter a configuration name or the fully qualified resource URI for a session configuration. If you specify only the configuration name, the following schema URI is prepended: `http://schemas.microsoft.com/PowerShell`.

The session configuration for a session is located on the remote computer. If the specified session configuration doesn't exist on the remote computer, the command fails.

The default value is the value of the `$PSSessionConfigurationName` preference variable on the local computer. If this preference variable isn't set, the default is Microsoft.PowerShell . For more information, see about_Preference_Variables (about/about_Preference_Variables.md).

-ConnectionUri <System.Uri[]>

Specifies a Uniform Resource Identifier (URI) that defines the connection endpoint of the session. The URI must be fully qualified.

The format of this string is as follows:

`<Transport>://<ComputerName>:<Port>/<ApplicationName>`

The default value is as follows:

`http://localhost:5985/WSMAN`

If you don't specify a connection URI, you can use the UseSSL and Port parameters to specify the connection URI values.

Valid values for the Transport segment of the URI are HTTP and HTTPS. If you specify a connection URI with a Transport segment, but don't specify a port, the session is created with the standards ports: 80 for HTTP and 443 for HTTPS. To use the default ports for PowerShell remoting, specify port 5985 for HTTP or 5986 for HTTPS.

If the destination computer redirects the connection to a different URI, PowerShell prevents the redirection unless you use the AllowRedirection parameter in the command.

-ContainerId <System.String[]>

Specifies an array of container IDs.

-Credential <System.Management.Automation.PSCredential>

Specifies a user account that has permission to perform this action. The default is the current user.

Type a user name, such as User01 or Domain01\User01 , or enter a PSCredential object generated by the `Get-Credential` cmdlet. If you type a user name, you're prompted to enter the password.

Credentials are stored in a PSCredential (/dotnet/api/system.management.automation.pscredential)object and the password is stored as a SecureString (/dotnet/api/system.security.securestring).

> [!NOTE] > For more information about SecureString data protection, see > How secure is SecureString? (/dotnet/api/system.security.securestring#how-secure-is-securestring).

-EnableNetworkAccess <System.Management.Automation.SwitchParameter>
Indicates that this cmdlet adds an interactive security token to loopback sessions. The interactive token lets you run commands in the loopback session that get data from other computers. For example, you can run a command in the session that copies XML files from a remote computer to the local computer.

A loopback session is a PSSession that originates and ends on the same computer. To create a loopback session, omit the ComputerName parameter or set its value to dot (`.`), localhost, or the name of the local computer.

By default, loopback sessions are created using a network token, which might not provide sufficient permission to authenticate to remote computers.

The EnableNetworkAccess parameter is effective only in loopback sessions. If you use EnableNetworkAccess when you create a session on a remote computer, the command succeeds, but the parameter is ignored.

You can allow remote access in a loopback session using the CredSSP value of the Authentication parameter, which delegates the session credentials to other computers.

To protect the computer from malicious access, disconnected loopback

sessions that have interactive tokens, which are those created using EnableNetworkAccess , can be reconnected only from the computer on which the session was created. Disconnected sessions that use CredSSP authentication can be reconnected from other computers. For more information, see `Disconnect-PSSession`.

This parameter was introduced in PowerShell 3.0.

-FilePath <System.String>

Specifies a local script that this cmdlet runs on one or more remote computers. Enter the path and filename of the script, or pipe a script path to `Invoke-Command`. The script must exist on the local computer or in a directory that the local computer can access. Use ArgumentList to specify the values of parameters in the script.

When you use this parameter, PowerShell converts the contents of the specified script file to a script block, transmits the script block to the remote computer, and runs it on the remote computer.

-HideComputerName <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet omits the computer name of each object from the output display. By default, the name of the computer that generated the object appears in the display.

This parameter affects only the output display. It doesn't change the object.

-InDisconnectedSession <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet runs a command or script in a disconnected session.

When you use the InDisconnectedSession parameter, `Invoke-Command` creates a persistent session on each remote computer, starts the command specified

by the ScriptBlock or FilePath parameter, and then disconnects from the session. The commands continue to run in the disconnected sessions. InDisconnectedSession enables you to run commands without maintaining a connection to the remote sessions. And, because the session is disconnected before any results are returned, InDisconnectedSession makes sure that all command results are returned to the reconnected session, instead of being split between sessions.

You can't use InDisconnectedSession with the Session parameter or the AsJob parameter.

Commands that use InDisconnectedSession return a PSSession object that represents the disconnected session. They don't return the command output. To connect to the disconnected session, use the `Connect-PSSession` or `Receive-PSSession` cmdlets. To get the results of commands that ran in the session, use the `Receive-PSSession` cmdlet. To run commands that generate output in a disconnected session, set the value of the OutputBufferingMode session option to Drop . If you intend to connect to the disconnected session, set the idle time-out in the session so that it provides sufficient time for you to connect before deleting the session.

You can set the output buffering mode and idle time-out in the SessionOption parameter or in the `$PSSessionOption` preference variable. For more information about session options, see `New-PSSessionOption` and about_Preference_Variables (./about/about_preference_variables.md).

For more information about the Disconnected Sessions feature, see about_Remote_Disconnected_Sessions (about/about_Remote_Disconnected_Sessions.md).

This parameter was introduced in PowerShell 3.0.

-InputObject <System.Management.Automation.PSObject>

Specifies input to the command. Enter a variable that contains the objects or type a command or expression that gets the objects.

When using the InputObject parameter, use the `$Input` automatic variable in the value of the ScriptBlock parameter to represent the input objects.

-JobName <System.String>

Specifies a friendly name for the background job. By default, jobs are named `Job<n>`, where `<n>` is an ordinal number.

If you use the JobName parameter in a command, the command is run as a job, and `Invoke-Command` returns a job object, even if you don't include AsJob in the command.

For more information about PowerShell background jobs, see about_Jobs (./About/about_Jobs.md).

-NoNewScope <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet runs the specified command in the current scope. By default, `Invoke-Command` runs commands in their own scope.

This parameter is valid only in commands that are run in the current session, that is, commands that omit both the ComputerName and Session parameters.

This parameter was introduced in PowerShell 3.0.

-Port <System.Int32>

Specifies the network port on the remote computer that is used for this command. To connect to a remote computer, the remote computer must be listening on the port that the connection uses. The default ports are 5985 (WinRM port for HTTP) and 5986 (WinRM port for HTTPS).

Before using an alternate port, configure the WinRM listener on the remote computer to listen at that port. To configure the listener, type the following two commands at the PowerShell prompt:

`Remove-Item -Path WSMan:\Localhost\listener\listener* -Recurse`

`New-Item -Path WSMan:\Localhost\listener -Transport http -Address * -Port <port-number>`

Don't use the Port parameter unless you must. The port that is set in the command applies to all computers or sessions on which the command runs. An alternate port setting might prevent the command from running on all computers.

-RunAsAdministrator <System.Management.Automation.SwitchParameter>
Indicates that this cmdlet invokes a command as an Administrator.

-ScriptBlock <System.Management.Automation.ScriptBlock>
Specifies the commands to run. Enclose the commands in braces (`{ }`) to create a script block. When using `Invoke-Command` to run a command remotely, any variables in the command are evaluated on the remote computer.

> [!NOTE] > Parameters for the scriptblock can only be passed in from ArgumentList by position. Switch > parameters cannot be passed by position. If you need a parameter that behaves like a > SwitchParameter type, use a Boolean type instead.

-Session <System.Management.Automation.Runspaces.PSSession[]>
Specifies an array of sessions in which this cmdlet runs the command. Enter a variable that contains PSSession objects or a command that creates or gets the PSSession objects, such as a `New-PSSession` or `Get-PSSession` command.

When you create a PSSession , PowerShell establishes a persistent connection to the remote computer. Use a PSSession to run a series of related commands that share data. To run a single command or a series of unrelated commands, use the ComputerName parameter. For more information, see about_PSSessions (./About/about_PSSessions.md).

-SessionName <System.String[]>

Specifies a friendly name for a disconnected session. You can use the name to refer to the session in subsequent commands, such as a `Get-PSSession` command. This parameter is valid only with the InDisconnectedSession parameter.

This parameter was introduced in PowerShell 3.0.

-SessionOption <System.Management.Automation.Remoting.PSSessionOption>

Specifies advanced options for the session. Enter a SessionOption object, such as one that you create using the `New-PSSessionOption` cmdlet, or a hash table in which the keys are session option names and the values are session option values.

> [!NOTE] > If you specify a hashtable for SessionOption , PowerShell converts the hashtable into a > System.Management.Autiomation.Remoting.PSSessionOption object. The values for keys specified > in the hashtable are cast to the matching property of the object. This behaves differently from > calling `New-PSSessionOption`. For example, the System.TimeSpan values for the timeout > properties, like IdleTimeout , convert an integer value into ticks instead of milliseconds. > For more information on the PSSessionOption object and its properties, see > PSSessionOption (/dotnet/api/system.management.automation.remoting.pssessionoption)The default values for the options are determined by the value of the `$PSSessionOption` preference variable, if it's set. Otherwise, the

default values are established by options set in the session configuration.

The session option values take precedence over default values for sessions set in the `$PSSessionOption` preference variable and in the session configuration. However, they don't take precedence over maximum values, quotas, or limits set in the session configuration.

For a description of the session options that includes the default values, see `New-PSSessionOption`. For information about the `$PSSessionOption` preference variable, see about_Preference_Variables (About/about_Preference_Variables.md). For more information about session configurations, see about_Session_Configurations (About/about_Session_Configurations.md).

-ThrottleLimit <System.Int32>

Specifies the maximum number of concurrent connections that can be established to run this command. If you omit this parameter or enter a value of 0, the default value, 32, is used.

The throttle limit applies only to the current command, not to the session or to the computer.

-UseSSL <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet uses the Secure Sockets Layer (SSL) protocol to establish a connection to the remote computer. By default, SSL isn't used.

WS-Management encrypts all PowerShell content transmitted over the network. The UseSSL parameter is an additional protection that sends the data across an HTTPS, instead of HTTP.

If you use this parameter, but SSL isn't available on the port that's used for the command, the command fails.

-VMId <System.Guid[]>

Specifies an array of IDs of virtual machines.

-VMName <System.String[]>

Specifies an array of names of virtual machines.

<CommonParameters>

This cmdlet supports the common parameters: Verbose, Debug,
ErrorAction, ErrorVariable, WarningAction, WarningVariable,
OutBuffer, PipelineVariable, and OutVariable. For more information, see
about_CommonParameters (https:/go.microsoft.com/fwlink/?LinkID=113216).

---------- Example 1: Run a script on a server ----------

Invoke-Command -FilePath c:\scripts\test.ps1 -ComputerName Server01

The FilePath parameter specifies a script that is located on the local
computer. The script runs on the remote computer and the results are returned
to the local computer.

--------- Example 2: Run a command on a remote server ---------

```
Invoke-Command -ComputerName Server01 -Credential Domain01\User01 -ScriptBlock
{
    Get-Culture
}
```

The ComputerName parameter specifies the name of the remote computer. The
Credential parameter is used to run the command in the security context of
Domain01\User01, a user who has permission to run commands. The ScriptBlock
parameter specifies the command to be run on the remote computer.

In response, PowerShell requests the password and an authentication method for
the User01 account. It then runs the command on the Server01 computer and

returns the result.

----- Example 3: Run a command in a persistent connection -----

```
$s = New-PSSession -ComputerName Server02 -Credential Domain01\User01
Invoke-Command -Session $s -ScriptBlock { Get-Culture }
```

The `New-PSSession` cmdlet creates a session on the Server02 remote computer and saves it in the `$s` variable. Typically, you create a session only when you run a series of commands on the remote computer.

The `Invoke-Command` cmdlet runs the `Get-Culture` command on Server02. The Session parameter specifies the session saved in the `$s` variable.

In response, PowerShell runs the command in the session on the Server02 computer.

Example 4: Use a session to run a series of commands that share data

```
Invoke-Command -ComputerName Server02 -ScriptBlock { $p = Get-Process
PowerShell }
Invoke-Command -ComputerName Server02 -ScriptBlock { $p.VirtualMemorySize }
$s = New-PSSession -ComputerName Server02
Invoke-Command -Session $s -ScriptBlock { $p = Get-Process PowerShell }
Invoke-Command -Session $s -ScriptBlock { $p.VirtualMemorySize }

17930240
```

The first two commands use the ComputerName parameter of `Invoke-Command` to run commands on the Server02 remote computer. The first command uses the `Get-Process` cmdlet to get the PowerShell process on the remote computer and to save it in the `$p` variable. The second command gets the value of the VirtualMemorySize property of the PowerShell process.

When you use the ComputerName parameter, PowerShell creates a new session to

run the command. The session is closed when the command completes. The `$p` variable was created in one connection, but it doesn't exist in the connection created for the second command.

The problem is solved by creating a persistent session on the remote computer, then running both of the commands in the same session.

The `New-PSSession` cmdlet creates a persistent session on the computer Server02 and saves the session in the `$s` variable. The `Invoke-Command` lines that follow use the Session parameter to run both of the commands in the same session. Since both commands run in the same session, the `$p` value remains active.

Example 5: Invoke a command with a script block stored in a variable

```
$command = {
    Get-EventLog -LogName 'Windows PowerShell' |
        Where-Object { $_.Message -like '*certificate*' }
}
Invoke-Command -ComputerName S1, S2 -ScriptBlock $command
```

The `$command` variable stores the `Get-EventLog` command that's formatted as a script block. The `Invoke-Command` runs the command stored in `$command` on the S1 and S2 remote computers.

----- Example 6: Run a single command on several computers -----

```
$parameters = @{
  ComputerName      = 'Server01', 'Server02', 'TST-0143', 'localhost'
  ConfigurationName = 'MySession.PowerShell'
  ScriptBlock = { Get-EventLog 'Windows PowerShell' }
}
Invoke-Command @parameters
```

The ComputerName parameter specifies a comma-separated list of computer names.

The list of computers includes the localhost value, which represents the local computer. The ConfigurationName parameter specifies an alternate session configuration. The ScriptBlock parameter runs `Get-EventLog` to get the Windows PowerShell event logs from each computer.

Example 7: Get the version of the host program on multiple computers

```
$version = Invoke-Command -ComputerName (Get-Content Machines.txt)
-ScriptBlock {
    (Get-Host).Version
}
```

Because only one command is run, you don't have to create persistent connections to each of the computers. Instead, the command uses the ComputerName parameter to indicate the computers. To specify the computers, it uses the `Get-Content` cmdlet to get the contents of the Machine.txt file, a file of computer names.

The `Invoke-Command` cmdlet runs a `Get-Host` command on the remote computers. It uses dot notation to get the Version property of the PowerShell host.

These commands run one at a time. When the commands complete, the output of the commands from all of the computers is saved in the `$version` variable. The output includes the name of the computer from which the data originated.

- Example 8: Run a background job on several remote computers -

```
$s = New-PSSession -ComputerName Server01, Server02
Invoke-Command -Session $s -ScriptBlock { Get-EventLog system } -AsJob

Id   Name    State      HasMoreData   Location          Command
---  ----    -----      -----         -----------       ---------------
1    Job1    Running    True          Server01,Server02  Get-EventLog system

$j = Get-Job
```

```
$j | Format-List -Property *

HasMoreData   : True
StatusMessage :
Location      : Server01,Server02
Command       : Get-EventLog system
JobStateInfo  : Running
Finished      : System.Threading.ManualResetEvent
InstanceId    : e124bb59-8cb2-498b-a0d2-2e07d4e030ca
Id            : 1
Name          : Job1
ChildJobs     : {Job2, Job3}
Output        : {}
Error         : {}
Progress      : {}
Verbose       : {}
Debug         : {}
Warning       : {}
StateChanged  :

$results = $j | Receive-Job
```

The `New-PSSession` cmdlet creates sessions on the Server01 and Server02 remote computers. The `Invoke-Command` cmdlet runs a background job in each of the sessions. The command uses the AsJob parameter to run the command as a background job. This command returns a job object that contains two child job objects, one for each of the jobs run on the two remote computers.

The `Get-Job` command saves the job object in the `$j` variable. The `$j` variable is then piped to the `Format-List` cmdlet to display all properties of the job object in a list. The last command gets the results of the jobs. It pipes the job object in `$j` to the `Receive-Job` cmdlet and stores the results in the `$results` variable.

Example 9: Include local variables in a command run on a remote computer

```
$Log = 'Windows PowerShell'
Invoke-Command -ComputerName Server01 -ScriptBlock {
   Get-EventLog -LogName $Using:Log -Newest 10
}
```

The `$Log` variable stores the name of the event log, Windows PowerShell. The `Invoke-Command` cmdlet runs `Get-EventLog` on Server01 to get the ten newest events from the event log. The value of the LogName parameter is the `$Log` variable, which is prefixed by the `Using` scope modifier to indicate that it was created in the local session, not in the remote session.

---------- Example 10: Hide the computer name ----------

```
Invoke-Command -ComputerName S1, S2 -ScriptBlock { Get-Process PowerShell }

PSComputerName    Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id
ProcessName
--------------    -------  ------    -----      ----- -----   ------     --
-----------
S1                575      15        45100      40988   200     4.68     1392
PowerShell
S2                777      14        35100      30988   150     3.68     67
PowerShell

Invoke-Command -ComputerName S1, S2 -HideComputerName -ScriptBlock {
   Get-Process PowerShell
}

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id   ProcessName
-------  ------    -----      ----- -----   ------     --   -----------
575      15        45100      40988   200     4.68     1392 PowerShell
777      14        35100      30988   150     3.68     67   PowerShell
```

The first two commands use `Invoke-Command` to run a `Get-Process` command for the PowerShell process. The output of the first command includes the PsComputerName property, which contains the name of the computer on which the command ran. The output of the second command, which uses HideComputerName , doesn't include the PsComputerName column.

----- Example 11: Use the Param keyword in a script block -----

```
$parameters = @{
    ComputerName = 'Server01'
    ScriptBlock  = {
        Param ($param1, $param2)
        Get-ChildItem -Name $param1 -Include $param2
    }
    ArgumentList = 'a*', '*.pdf'
}
Invoke-Command @parameters
```

```
aa.pdf
ab.pdf
ac.pdf
az.pdf
```

`Invoke-Command` uses the ScriptBlock parameter that defines two variables, `$param1` and `$param2`. `Get-ChildItem` uses the named parameters, Name and Include with the variable names. The ArgumentList passes the values to the variables.

Example 12: Use the $args automatic variable in a script block

```
$parameters = @{
    ComputerName = 'Server01'
    ScriptBlock  = { Get-ChildItem $args[0] $args[1] }
    ArgumentList = 'C:\Test', '*.txt*'
```

```
}
Invoke-Command @parameters

Directory: C:\Test

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          6/12/2019    15:15            128 alog.txt
-a---          7/27/2019    15:16            256 blog.txt
-a---          9/28/2019    17:10             64 zlog.txt
```

`Invoke-Command` uses a ScriptBlock parameter and `Get-ChildItem` specifies the `$args[0]` and `$args[1]` array values. The ArgumentList passes the `$args` array values to the `Get-ChildItem` parameter positions for Path and Filter .

Example 13: Run a script on all the computers listed in a text file

```
$parameters = @{
    ComputerName = (Get-Content Servers.txt)
    FilePath     = 'C:\Scripts\Sample.ps1'
    ArgumentList = 'Process', 'Service'
}
Invoke-Command @parameters
```

When you submit the command, the content of the `Sample.ps1` file is copied into a script block and the script block is run on each of the remote computers. This procedure is equivalent to using the ScriptBlock parameter to submit the contents of the script.

-- Example 14: Run a command on a remote computer using a URI --

```
$LiveCred = Get-Credential
$parameters = @{
  ConfigurationName = 'Microsoft.Exchange'
```

```
  ConnectionUri     = 'https://ps.exchangelabs.com/PowerShell'
  Credential        = $LiveCred
  Authentication    = 'Basic'
  ScriptBlock       = { Set-Mailbox Dan -DisplayName 'Dan Park' }
}
Invoke-Command @parameters
```

The first line uses the `Get-Credential` cmdlet to store Windows Live ID credentials in the `$LiveCred` variable. PowerShell prompts the user to enter Windows Live ID credentials.

The `$parameters` variable is a hash table containing the parameters to be passed to the `Invoke-Command` cmdlet. The `Invoke-Command` cmdlet runs a `Set-Mailbox` command using the Microsoft.Exchange session configuration. The ConnectionURI parameter specifies the URL of the Exchange server endpoint. The Credential parameter specifies the credentials stored in the `$LiveCred` variable. The AuthenticationMechanism parameter specifies the use of basic authentication. The ScriptBlock parameter specifies a script block that contains the command.

---------- Example 15: Use a session option ----------

```
$so = New-PSSessionOption -SkipCACheck -SkipCNCheck -SkipRevocationCheck
$parameters = @{
   ComputerName  = 'server01'
   UseSSL        = $true
   ScriptBlock   = { Get-HotFix }
   SessionOption = $so
   Credential    = 'server01\user01'
}
Invoke-Command @parameters
```

The `New-PSSessionOption` cmdlet creates a session option object that causes the remote end not to verify the Certificate Authority, Canonical Name, and

Revocation Lists while evaluating the incoming HTTPS connection. The SessionOption object is saved in the `$so` variable.

> [!NOTE] > Disabling these checks is convenient for troubleshooting, but obviously not secure.

The `Invoke-Command` cmdlet runs a `Get-HotFix` command remotely. The SessionOption parameter is given the `$so` variable.
---- Example 16: Manage URI redirection in a remote command ----

```
$max = New-PSSessionOption -MaximumRedirection 1
$parameters = @{
  ConnectionUri    = 'https://ps.exchangelabs.com/PowerShell'
  ScriptBlock      = { Get-Mailbox dan }
  AllowRedirection = $true
  SessionOption    = $max
}
Invoke-Command @parameters
```

The `New-PSSessionOption` cmdlet creates a PSSessionOption object that is saved in the `$max` variable. The command uses the MaximumRedirection parameter to set the MaximumConnectionRedirectionCount property of the PSSessionOption object to 1.

The `Invoke-Command` cmdlet runs a `Get-Mailbox` command on a remote Microsoft Exchange Server. The AllowRedirection parameter provides explicit permission to redirect the connection to an alternate endpoint. The SessionOption parameter uses the session object stored in the `$max` variable.

As a result, if the remote computer specified by ConnectionURI returns a redirection message, PowerShell redirects the connection, but if the new destination returns another redirection message, the redirection count value of 1 is exceeded, and `Invoke-Command` returns a non-terminating error.

---- Example 17: Access a network share in a remote session ----

```
Enable-WSManCredSSP -Role Client -DelegateComputer Server02
$s = New-PSSession Server02
Invoke-Command -Session $s -ScriptBlock { Enable-WSManCredSSP -Role Server
-Force }
$parameters = @{
  Session        = $s
  ScriptBlock    = { Get-Item \\Net03\Scripts\LogFiles.ps1 }
  Authentication = 'CredSSP'
  Credential     = 'Domain01\Admin01'
}
Invoke-Command @parameters
```

The `Enable-WSManCredSSP` cmdlet enables CredSSP delegation from the Server01 local computer to the Server02 remote computer. The Role parameter specifies Client to configure the CredSSP client setting on the local computer.

`New-PSSession` creates a PSSession object for Server02 and stores the object in the `$s` variable.

The `Invoke-Command` cmdlet uses the `$s` variable to connect to the remote computer, Server02. The ScriptBlock parameter runs `Enable-WSManCredSSP` on the remote computer. The Role parameter specifies Server to configure the CredSSP server setting on the remote computer.

The `$parameters` variable contains the parameter values to connect to the network share. The `Invoke-Command` cmdlet runs a `Get-Item` command in the session in `$s`. This command gets a script from the `\Net03\Scripts` network share. The command uses the Authentication parameter with a value of CredSSP and the Credential parameter with a value of Domain01\Admin01 .

------ Example 18: Start scripts on many remote computers ------

```
$parameters = @{
  ComputerName          = (Get-Content -Path C:\Test\Servers.txt)
  InDisconnectedSession = $true
  FilePath              = '\\Scripts\Public\ConfigInventory.ps1'
  SessionOption         = @{
     OutputBufferingMode = 'Drop'
     IdleTimeout         = [timespan]::FromHours(12)
  }
}
Invoke-Command @parameters
```

The command uses `Invoke-Command` to run the script. The value of the ComputerName parameter is a `Get-Content` command that gets the names of the remote computers from a text file. The InDisconnectedSession parameter disconnects the sessions as soon as it starts the command. The value of the FilePath parameter is the script that `Invoke-Command` runs on each computer.

The value of SessionOption is a hash table. The OutputBufferingMode value is set to `Drop` and the IdleTimeout value is set to 12 hours.

To get the results of commands and scripts that run in disconnected sessions, use the `Receive-PSSession` cmdlet.

REMARKS

To see the examples, type: "get-help Invoke-Command -examples".
For more information, type: "get-help Invoke-Command -detailed".
For technical information, type: "get-help Invoke-Command -full".
For online help, type: "get-help Invoke-Command -online"