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 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.

Starting with PowerShell 6.0 you can use Secure Shell (SSH) to establish a connection to and invoke commands on remote computers. SSH must be installed on the local computer and the remote computer must be configured with a PowerShell SSH endpoint. The benefit of an SSH based PowerShell remote session is that it will work across multiple platforms (Windows, Linux, macOS). For SSH based session you use the **HostName** or **SSHConnection** parameter set to specify the remote computer and relevant connection information. For more information about how to set up PowerShell SSH remoting, see PowerShell Remoting Over SSH.

Examples

Example 1: Run a script on a server

This example runs the `Test.ps1` script on the Server01 computer.

PowerShell
```
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

This example runs a `Get-Culture` command on the Server01 remote computer.

PowerShell
```
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

This example runs the same `Get-Culture` command in a session, using a persistent connection, on the remote computer named Server02.

PowerShell
```
$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. Tne **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

This example compares the effects of using **ComputerName** and **Session** parameters of `Invoke-Command`. It shows how to use a session to run a series of commands that share the same data.

PowerShell
```
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 does not 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 both 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: Enter a command stored in a local variable

This example shows how to create a command that is stored as a script block in a local variable. When the script block is saved in a local variable, you can specify the variable as the value of the **ScriptBlock** parameter.

PowerShell
```
$command = { Get-EventLog -log "Windows PowerShell" | where {$_.Message -like "*certificate*"} }
Invoke-Command -ComputerName S1, S2 -ScriptBlock $command
```

The first line saves a `Get-EventLog` command in the `$command` variable. The command is formatted as a script block. The second line uses `Invoke-Command` to run the command in `$command` on the S1 and S2 remote computers.

Example 6: Run a single command on several computers

This example demonstrates how to use `Invoke-Command` to run a single command on multiple computers.

The command uses the **ComputerName** parameter to specify the computers. The computer names are presented in a comma-separated list. The list of computers includes the localhost value, which represents the local computer. In this example, the command in the script block gets the events in the Windows PowerShell event log on each remote computer.

PowerShell
```
Invoke-Command -ComputerName Server01, Server02, TST-0143, localhost -ConfigurationName MySession.PowerShell -ScriptBlock
{Get-EventLog "Windows PowerShell"}
```

The command uses the **ConfigurationName** parameter to specify an alternate session configuration for PowerShell and the **ScriptBlock** parameter to specify the command.

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

This example gets the version of the PowerShell host program running on 200 remote computers.

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

Because only one command is run, you do not 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

This example runs a command on two remote computers. The `Invoke-Command` command uses the **AsJob** parameter so that the command runs as a background job. The commands run on the remote computers, but the job actually resides on the local computer. The results are transmitted to the local computer.

PowerShell
```
$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

This example shows how to include the values of local variables in a command run on a remote computer. The command uses the Using scope modifier to identify a local variable in a remote command. By default, all variables are assumed to be defined in the remote session. The Using scope modifier was introduced in Windows PowerShell 3.0. For more information about the Using scope modifier, see about_Remote_Variables.

PowerShell
```
$MWFO_Log = "Microsoft-Windows-Forwarding/Operational"
Invoke-Command -ComputerName Server01 -ScriptBlock {Get-EventLog -LogName $Using:MWFO_Log -Newest 10}
```

The first line stores the name of the event log in the `$MWFO_Log` variable. The `Invoke-Command` cmdlet runs `Get-EventLog` on Server01 to get the 10 newest events from the Microsoft-Windows-Forwarding/Operational event log. The value of the **LogName** parameter is the `$MWFO_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

This example shows the effect of using the **HideComputerName** parameter of `Invoke-Command`. **HideComputerName** does not change the object that this cmdlet returns. It changes only the display. You can still use the **Format** cmdlets to display the **PsComputerName** property of any of the affected objects.

PowerShell
```
`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 -ScriptBlock {Get-Process PowerShell} -HideComputerName

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**, does not include the **PsComputerName** column.

Example 11: Run a script on all the computers listed in a text file

This example uses the `Invoke-Command` cmdlet to run the `Sample.ps1` script on all the computers listed in the `Servers.txt` file. The command uses the **FilePath** parameter to specify the script file. This command lets you run the script on the remote computers, even if the script file is not accessible to the remote computers.

PowerShell
```
Invoke-Command -ComputerName (Get-Content Servers.txt) -FilePath C:\Scripts\Sample.ps1 -ArgumentList Process, Service
```

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 12: Run a command on a remote computer by using a URI

This example shows how to run a command on a remote computer that is identified by a URI. This particular example runs a `Set-Mailbox` command on a remote Exchange server.

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

Example 13: Use a session option

This example shows how to create and use a **SessionOption** parameter.

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

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 14: Manage URI redirection in a remote command

This example shows how to use the **AllowRedirection** and **SessionOption** parameters to manage URI redirection in a remote command.

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

The `New-PSSessionOption` cmdlet to 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 use the session object 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 15: Access a network share in a remote session

This example shows how to access a network share from a remote session. The command requires that CredSSP delegation be enabled in the client settings on the local computer and in the service settings on the remote computer. To run the commands in this example, you must be a member of the Administrators group on the local computer and the remote computer.

PowerShell
```
Enable-WSManCredSSP -Delegate Server02
Connect-WSMan Server02
Set-Item WSMan:\Server02*\Service\Auth\CredSSP -Value $True
$s = New-PSSession Server02
Invoke-Command -Session $s -ScriptBlock {Get-Item \\Net03\Scripts\LogFiles.ps1} -Authentication CredSSP -Credential
Domain01\Admin01
```

The `Enable-WSManCredSSP` cmdlet enables CredSSP delegation from the Server01 local computer to the Server02 remote computer. This configures the CredSSP client setting on the local computer.

The `Connect-WSMan` cmdlet connects to the Server02 computer. This action adds a node for the Server02 computer to the WSMan: drive on the local computer. You can use the WSMAN: drive to view and change the WS-Management settings on the Server02 computer.

The `Set-Item` cmdlet changes the value of the CredSSP item in the Service node of the Server02 computer to `True`. This action enables CredSSP in the service settings on the remote computer.

The `New-PSSession` cmdlet creates a PSSession on the Server02 computer that is saved in the `$s` variable.

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 **Credential** parameter with a value of **Domain01\Admin01** and the **Authentication** parameter with a value of **CredSSP**.

Example 16: Start scripts on many remote computers

This command runs a script on more than a hundred computers. To minimize the impact on the local computer, it connects to each computer, starts the script, and then disconnects from each computer. The script continues to run in the disconnected sessions.

PowerShell
```
Invoke-Command -ComputerName (Get-Content Servers.txt) -InDisconnectedSession -FilePath
\\Scripts\Public\ConfigInventory.ps1 -SessionOption @{OutputBufferingMode="Drop";IdleTimeout=43200000}
```

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 that sets the value of the **OutputBufferingMode** option to **Drop** and the value of the **IdleTimeout** option to **43200000** milliseconds (12 hours).

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

Example 17: Run a command on a remote computer using SSH

This example shows how to run a command on a remote computer using Secure Shell (SSH). If SSH is configured on the remote computer to prompt for passwords then you will get a password prompt. Otherwise you will have to use SSH key based user authentication.

PowerShell
```
Invoke-Command -HostName UserA@LinuxServer01 -ScriptBlock { Get-MailBox * }
```

Example 18: Run a command on a remote computer using SSH and specify a user authentication key

This example shows how to run a command on a remote computer using SSH and specifying a key file for user authentication. You will not get a password prompt unless the key authentication fails and the remote computer is configured to allow basic password authentication.

PowerShell
```
Invoke-Command -HostName UserA@LinuxServer01 -ScriptBlock { Get-MailBox * } -KeyFilePath c:\<path>\userAKey_rsa
```

Example 19: Run a script file on multiple remote computers using SSH as a job

This example shows how to run a script file on multiple remote computers using SSH and the **SSHConnection** parameter set. The SSHConnection parameter takes an array of hash tables that contain connection information for each computer. Note that this example requires that the target remote computers have SSH configured to support key based user authentication.

PowerShell
```
$sshConnections = @{ HostName="WinServer1"; UserName="domain\userA"; KeyFilePath="c:\users\UserA\id_rsa" }, @{
HostName="UserB@LinuxServer5"; KeyFilePath="c:\UserB\<path>\id_rsa }
$results = Invoke-Command -FilePath c:\Scripts\CollectEvents.ps1 -SSHConnection $sshConnections
```

Parameters

-AllowRedirection

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 does not 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.

Type:	SwitchParameter
Position:	Named
Default value:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-ApplicationName

Specifies the application name segment of the connection URI. Use this parameter to specify the application name when you are not 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 is not defined, the default value is WSMAN. This value is appropriate for most uses. For more information, see about_Preference_Variables.

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.

Type:	String
Position:	Named
Default value:	None
Accept pipeline input:	True (ByPropertyName)
Accept wildcard characters:	False

-ArgumentList

Supplies the values of local variables in the command. The variables in the command are replaced by these values before the command is run on the remote computer. Enter the values in a comma-separated list. Values are associated with variables in the order that they are listed. The alias for **ArgumentList** is Args.

The values in the **ArgumentList** parameter can be actual values, such as 1024, or they can be references to local variables, such as `$max`.

To use local variables in a command, use the following command format:

```
{param($<name1>[, $<name2>]...) <command-with-local-variables>} -ArgumentList <value> -or- <local-variable>
```

The **param** keyword lists the local variables that are used in the command. **ArgumentList** supplies the values of the variables, in the order that they are listed.

Type:	Object[]

Aliases:	Args
Position:	Named
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-AsJob

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, and the results of the remote job are automatically returned to the local computer.

For more information about PowerShell background jobs, see about_Jobs and about_Remote_Jobs.

Type:	SwitchParameter
Position:	Named
Default value:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-Authentication

Specifies the mechanism that is used to authenticate the user's credentials. The acceptable values for this parameter are:

- Default
- Basic
- Credssp
- Digest

- Kerberos
- Negotiate
- NegotiateWithImplicitCredential

The default value is Default.

CredSSP authentication is available only in Windows Vista, Windows Server 2008, and later versions of the Windows operating system.

For more information about the values of this parameter, see AuthenticationMechanism Enumeration.

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.

Type:	AuthenticationMechanism
Accepted values:	Default, Basic, Negotiate, NegotiateWithImplicitCredential, Credssp, Digest, Kerberos
Position:	Named
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-CertificateThumbprint

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; they do not work with domain accounts.

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

Type:	String
Position:	Named
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-ComputerName

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 is 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. Also, the computer must be configured for HTTPS transport or the IP address of the remote computer must be included in the WinRM TrustedHosts list on the local computer. For instructions for adding a computer name to the TrustedHosts list, see "How to Add a Computer to the Trusted Host List" in about_Remote_Troubleshooting.

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

Type:	String[]
Aliases:	Cn
Position:	0
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-ConfigurationName

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`.

When used with SSH, this specifies the subsystem to use on the target as defined in sshd_config. The default value for SSH is the `powershell` subsystem.

The session configuration for a session is located on the remote computer. If the specified session configuration does not 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 is not set, the default is **Microsoft.PowerShell**. For more information, see about_Preference_Variables.

Type:	String
Position:	Named
Default value:	None
Accept pipeline input:	True (ByPropertyName)
Accept wildcard characters:	False

-ConnectionUri

Specifies a 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 do not 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 do not specify a port, the session is created with 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.

Type:	Uri[]
Aliases:	URI, CU
Position:	0
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-ContainerId

Specifies an array of container IDs.

Type:	String[]
Position:	Named
Default value:	None
Accept pipeline input:	True (ByPropertyName)
Accept wildcard characters:	False

-Credential

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, such as one generated by the `Get-Credential` cmdlet. If you type a user name, this cmdlet prompts you for a password.

Type:	PSCredential

Position:	Named
Default value:	None
Accept pipeline input:	True (ByPropertyName)
Accept wildcard characters:	False

-EnableNetworkAccess

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 by 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 also allow remote access in a loopback session by 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 by 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 Windows PowerShell 3.0.

Type:	SwitchParameter
Position:	Named
Default value:	False
Accept pipeline input:	False

Accept wildcard characters: False
-FilePath

Specifies a local script that this cmdlet runs on one or more remote computers. Enter the path and file name of the script, or pipe a script path to `Invoke-Command`. The script must reside 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.

Type:	String
Aliases:	PSPath
Position:	1
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-HideComputerName

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 does not change the object.

Type:	SwitchParameter
Aliases:	HCN
Position:	Named
Default value:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-HostName

Specifies an array of computer names for a Secure Shell (SSH) based connection. This is similar to the ComputerName parameter except that the connection to the remote computer is made using SSH rather than Windows WinRM.

This parameter was introduced in PowerShell 6.0.

Type:	String[]
Position:	Named
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-InDisconnectedSession

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. Also, 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 cannot use **InDisconnectedSession** with the **Session** parameter or the **AsJob** parameter.

Commands that use **InDisconnectedSession** return a **PSSession** object that represents the disconnected session. They do not 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.

For more information about the Disconnected Sessions feature, see about_Remote_Disconnected_Sessions.

This parameter was introduced in Windows PowerShell 3.0.

Type:	SwitchParameter
Aliases:	Disconnected
Position:	Named
Default value:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-InputObject

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.

Type:	PSObject
Position:	Named
Default value:	None
Accept pipeline input:	True (ByValue)
Accept wildcard characters:	False

-JobName

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 do not include **AsJob** in the command.

For more information about PowerShell background jobs, see about_Jobs.

Type:	String
Position:	Named

Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-KeyFilePath

Specifies a key file path used by Secure Shell (SSH) to authenticate a user on a remote computer.

SSH allows user authentication to be performed via private/public keys as an alternative to basic password authentication. If the remote computer is configured for key authentication then this parameter can be used to provide the key that identifies the user.

This parameter was introduced in PowerShell 6.0.

Type:	String
Aliases:	IdentityFilePath
Position:	Named
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-NoNewScope

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 Windows PowerShell 3.0.

Type:	SwitchParameter
Position:	Named
Default value:	False

Accept pipeline input:	False
Accept wildcard characters:	False

-Port

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, which is the WinRM port for HTTP, and 5986, which is the 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\>
```

Do not 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.

Type:	Int32
Position:	Named
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-RemoteDebug

Used to run the invoked command in debug mode in the remote PowerShell session.

Type:	SwitchParameter
Position:	Named
Default value:	False
Accept pipeline input:	False

Accept wildcard characters:	False

-RunAsAdministrator

Indicates that this cmdlet invokes a command as an Administrator.

Type:	SwitchParameter
Position:	Named
Default value:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-SSHConnection

This parameter takes an array of hashtables where each hashtable contains one or more connection parameters needed to establish a Secure Shell (SSH) connection (HostName, Port, UserName, KeyFilePath).

The hashtable connection parameters are the same as defined for the HostName parameter set.

The SSHConnection parameter is useful for creating multiple sessions where each session requires different connection information.

This parameter was introduced in PowerShell 6.0.

Type:	Hashtable[]
Position:	Named
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-SSHTransport

Indicates that the remote connection is established using Secure Shell (SSH).

By default PowerShell uses Windows WinRM to connect to a remote computer. This switch forces PowerShell to use the HostName parameter set for establishing an SSH based remote connection.

This parameter was introduced in PowerShell 6.0.

Type:	SwitchParameter
Accepted values:	true
Position:	Named
Default value:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-ScriptBlock

Specifies the commands to run. Enclose the commands in braces ({ }) to create a script block. This parameter is required.

By default, any variables in the command are evaluated on the remote computer. To include local variables in the command, use **ArgumentList**.

Type:	ScriptBlock
Aliases:	Command
Position:	0
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-Session

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.

Type:	PSSession[]
Position:	0
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-SessionName

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 Windows PowerShell 3.0.

Type:	String[]
Position:	Named
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-SessionOption

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

The default values for the options are determined by the value of the `$PSSessionOption` preference variable, if it is 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 do not 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. For more information about session configurations, see about_Session_Configurations.

Type:	PSSessionOption
Position:	Named
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-ThrottleLimit

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.

Type:	Int32
Position:	Named
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-UseSSL

Indicates that this cmdlet uses the Secure Sockets Layer (SSL) protocol to establish a connection to the remote computer. By default, SSL is not 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 is not available on the port that is used for the command, the command fails.

Type:	SwitchParameter

Position:	Named
Default value:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-UserName

Specifies the user name for the account used to run a command on the remote computer. User authentication method will depend on how Secure Shell (SSH) is configured on the remote computer.

If SSH is configured for basic password authentication then you will be prompted for the user password.

If SSH is configured for key based user authentication then a key file path can be provided via the KeyFilePath parameter and no password prompt will occur. Note that if the client user key file is located in an SSH known location then the KeyFilePath parameter is not needed for key based authentication, and user authentication will occur automatically based on the user name. See SSH documentation about key based user authentication for more information.

This is not a required parameter. If no UserName parameter is specified then the current log on user name is used for the connection.

This parameter was introduced in PowerShell 6.0.

Type:	String
Position:	Named
Default value:	None
Accept pipeline input:	False
Accept wildcard characters:	False

-VMId

Specifies an array of IDs of virtual machines.

Type:	Guid[]
Aliases:	VMGuid

Position:	0
Default value:	None
Accept pipeline input:	True (ByPropertyName)
Accept wildcard characters:	False

-VMName

Specifies an array of names of virtual machines.

Type:	String[]
Position:	Named
Default value:	None
Accept pipeline input:	True (ByPropertyName)
Accept wildcard characters:	False

Inputs

System.Management.Automation.ScriptBlock

You can pipe a command in a script block to `Invoke-Command`. Use the `$Input` automatic variable to represent the input objects in the command.

Outputs

System.Management.Automation.PSRemotingJob, System.Management.Automation.Runspaces.PSSession, or the output of the invoked command

This cmdlet returns a job object, if you use the **AsJob** parameter. If you specify the **InDisconnectedSession** parameter, `Invoke-Command` returns a **PSSession** object. Otherwise, it returns the output of the invoked command, which is the value of the **ScriptBlock** parameter.

Notes

- On Windows Vista, and later versions of the Windows operating system, to use the **ComputerName** parameter of `Invoke-Command` to run a command on the local computer, you must open PowerShell by using the Run as administrator option.
- When you run commands on multiple computers, PowerShell connects to the computers in the order in which they appear in the list. However, the command output is displayed in the order that it is received from the remote computers, which might be different.
- Errors that result from the command that `Invoke-Command` runs are included in the command results. Errors that would be terminating errors in a local command are treated as non-terminating errors in a remote command. This strategy makes sure that terminating errors on one computer do not close the command on all computers on which it is run. This practice is used even when a remote command is run on a single computer.
- If the remote computer is not in a domain that the local computer trusts, the computer might not be able to authenticate the credentials of the user. To add the remote computer to the list of trusted hosts in WS-Management, use the following command in the WSMAN provider, where `<Remote-Computer-Name>` is the name of the remote computer:

  ```
  Set-Item -Path WSMan:\Localhost\Client\TrustedHosts -Value \<Remote-Computer-Name\>
  ```

- In Windows PowerShell 2.0, you cannot use the `Select-Object` cmdlet to select the **PSComputerName** property of the object that `Invoke-Command` returns. Instead, to display the value of the **PSComputerName** property, use the dot method to get the **PSComputerName** property value ($result.PSComputerName), use a **Format** cmdlet, such as the `Format-Table` cmdlet, to display the value of the **PSComputerName** property, or use a `Select-Object` command where the value of the property parameter is a calculated property that has a label other than PSComputerName. This limitation does not apply to Windows PowerShell 3.0 or later versions of Windows PowerShell or PowerShell Core.
- When you disconnect a **PSSession**, such as by using **InDisconnectedSession**, the session state is Disconnected and the availability is None. The value of the **State** property is relative to the current session. Therefore, a value of Disconnected means that the **PSSession** is not connected to the current session. However, it does not mean that the **PSSession** is disconnected from all sessions. It might be connected to a different session. To determine whether you can connect or reconnect to the session, use the **Availability** property.

  An **Availability** value of None indicates that you can connect to the session. A value of Busy indicates that you cannot connect to the PSSession because it is connected to another session.

  For more information about the values of the **State** property of sessions, see RunspaceState Enumeration.

  For more information about the values of the **Availability** property of sessions, see RunspaceAvailability Enumeration.

- The HostName and SSHConnection parameter sets were included starting with PowerShell 6.0. They were added to provide PowerShell remoting based on Secure Shell (SSH). Both SSH and PowerShell are supported on multiple platforms (Windows, Linux, macOS) and PowerShell remoting will work over these platforms where PowerShell and SSH are installed and configured. This is separate from the previous Windows only remoting that is based on WinRM and much of the WinRM specific features and limitations do not apply. For example WinRM based quotas, session options, custom endpoint configuration, and disconnect/reconnect features are currently not supported. For more information about how to set up PowerShell SSH remoting, see PowerShell Remoting Over SSH.