


Running Remote Commands

08/14/20183 minutes to read 

In this article

Windows PowerShell Remoting Without Configuration

Windows PowerShell Remoting

See Also

You can run commands on one or hundreds of computers with a single PowerShell command. Windows PowerShell supports remote computing by using various technologies, including WMI, RPC, and WS-Management.

PowerShell Core supports WMI, WS-Management, and SSH remoting. RPC is no longer supported.

For more information about remoting in PowerShell Core, see the following articles:

- [SSH Remoting in PowerShell Core](#)
- [WSMan Remoting in PowerShell Core](#)

Windows PowerShell Remoting Without Configuration

Many Windows PowerShell cmdlets have the `ComputerName` parameter that enables you to collect data and change settings on one or more remote computers. These cmdlets use varying communication protocols and work on all Windows operating systems without any special configuration.

These cmdlets include:

- Restart-Computer
- Test-Connection
- Clear-EventLog
- Get-EventLog
- Get-HotFix
- Get-Process
- Get-Service
- Set-Service
- Get-WinEvent
- Get-WmiObject

Typically, cmdlets that support remoting without special configuration have the ComputerName parameter and don't have the Session parameter. To find these cmdlets in your session, type:

PowerShell

Copy

```
Get-Command | where { $_.parameters.keys -  
contains "ComputerName" -and $_.parameter-  
s.keys -notcontains "Session" }
```

Windows PowerShell Remoting

Using the WS-Management protocol, Windows PowerShell remoting lets you run any Windows PowerShell command on one or more remote computers. You can establish persistent connections, start interactive sessions, and run scripts on remote computers.

To use Windows PowerShell remoting, the remote computer must be configured for remote management. For more information, including instructions, see [About Remote Requirements](#).

Once you have configured Windows PowerShell remoting, many remoting strategies are available to you. This article lists just a few of them. For more information, see [About Remote](#).

Start an Interactive Session

To start an interactive session with a single remote computer, use the [Enter-PSSession](#) cmdlet. For example, to start an interactive session with the Server01 remote computer, type:

PowerShell

Copy

[Enter-PSSession](#) Server01

The command prompt changes to display the name of the remote computer. Any commands that you type at the prompt run on the remote computer and the results are displayed on the local computer.

To end the interactive session, type:

PowerShell

Copy

[Exit-PSSession](#)

For more information about the [Enter-PSSession](#) and [Exit-PSSession](#) cmdlets, see:

- [Enter-PSSession](#)
- [Exit-PSSession](#)

Run a Remote Command

To run a command on one or more computers, use the [Invoke-Command](#) cmdlet. For example, to run a [Get-](#)

UICulture command on the Server01 and Server02 remote computers, type:

PowerShell

Copy

```
Invoke-Command -ComputerName Server01,  
Server02 -ScriptBlock {Get-UICulture}
```

The output is returned to your computer.

output

Copy

LCID	Name	DisplayName
PSComputerName		
----	----	-----

1033	en-US	English (United States)
	server01.corp.fabrikam.com	
1033	en-US	English (United States)
	server02.corp.fabrikam.com	

Run a Script

To run a script on one or many remote computers, use the `FilePath` parameter of the `Invoke-Command` cmdlet. The script must be on or accessible to your local computer. The results are returned to your local computer.

For example, the following command runs the `DiskCollect.ps1` script on the remote computers, `Server01` and `Server02`.

PowerShell

Copy

```
Invoke-Command -ComputerName Server01,  
Server02 -FilePath c:\Scripts\DiskCollect.ps1
```

Establish a Persistent Connection

Use the `New-PSSession` cmdlet to create a persistent session on a remote computer. The following example creates remote sessions on `Server01` and `Server02`. The session objects are stored in the `$s` variable.

PowerShell

Copy

```
$s = New-PSSession -ComputerName Server01,  
Server02
```

Now that the sessions are established, you can run any command in them. And because the sessions are persistent, you can collect data from one command and use it in another command.

For example, the following command runs a `Get-HotFix` command in the sessions in the `$s` variable and it saves the results in the `$h` variable. The `$h` variable is created in each of the sessions in `$s`, but it doesn't exist in the local session.

PowerShell

Copy

```
Invoke-Command -Session $s {$h = Get-HotFix}
```

Now you can use the data in the `$h` variable with other commands in the same session. The results are displayed on the local computer. For example:

PowerShell

Copy

- 17 -

```
Invoke-Command -Session $s {$h | where  
{$_ .InstalledBy -ne "NTAUTHORITY\SYSTEM"}}
```

Advanced Remoting

Windows PowerShell remote management just begins here. By using the cmdlets installed with Windows PowerShell, you can establish and configure remote sessions both from the local and remote ends, create customized and restricted sessions, allow users to import commands from a remote session that actually run implicitly on the remote session, configure the security of a remote session, and much more.

Windows PowerShell includes a WSMAN provider. The provider creates a **WSMAN:** drive that lets you navigate through a hierarchy of configuration settings on the local computer and remote computers.

For more information about the WSMAN provider, see [WSMAN Provider](#) and [About WS-Management Cmdlets](#), or in the Windows PowerShell console, type `Get-Help wsman`.

For more information, see:

- [About Remote FAQ](#)
- [Register-PSSessionConfiguration](#)
- [Import-PSSession](#)

For help with remoting errors, see [about Remote Troubleshooting](#).