Running Remote Commands - PowerShell

By sdwheeler

Archived: 2026-04-05 19:40:20 UTC

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

PowerShell supports WMI, WS-Management, and SSH remoting. In PowerShell 7 and higher, RPC is supported only on Windows.

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

- SSH Remoting in PowerShell
- WSMan Remoting in PowerShell

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:

```
 Get-Command | Where-Object {
    $_.Parameters.Keys -contains "ComputerName" -and
    $_.Parameters.Keys -notcontains "Session"
}
```

Windows PowerShell remoting

By 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 configure 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:

```
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:

```
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:

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

The output is returned to your computer.

```
 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.

```
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.

```
$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.

```
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:

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

Advanced Remoting

PowerShell includes cmdlets that allow you to:

- Configure and create remote sessions both from the local and remote ends
- Create customized and restricted sessions
- Import commands from a remote session that actually run implicitly on the remote session
- Configure the security of a remote session

PowerShell on Windows 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:

- PowerShell Remoting FAQ
- Register-PSSessionConfiguration
- Import-PSSession

For help with remoting errors, see about_Remote_Troubleshooting.

See Also

- about_Remote
- about_Remote_Requirements
- about_Remote_Troubleshooting
- about_PSSessions
- about_WS-Management_Cmdlets
- Invoke-Command
- Import-PSSession
- New-PSSession
- Register-PSSessionConfiguration
- WSMan Provider
