



python



PowerShell

FPDF Library
PDF generator

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

PowerShell Get-Help on command 'Receive-Job'

PS C:\Users\wahid> Get-Help Receive-Job

NAME

Receive-Job

SYNOPSIS

Gets the results of the PowerShell background jobs in the current session.

SYNTAX

```
Receive-Job [-Job] <System.Management.Automation.Job[]> [[-ComputerName]
<System.String[]>] [-AutoRemoveJob] [-Force] [-Keep] [-NoRecurse] [-Wait]
[-WriteEvents] [-WriteJobInResults] [<CommonParameters>]
```

```
Receive-Job [-Id] <System.Int32[]> [-AutoRemoveJob] [-Force] [-Keep]
[-NoRecurse] [-Wait] [-WriteEvents] [-WriteJobInResults] [<CommonParameters>]
```

```
Receive-Job [-InstanceId] <System.Guid[]> [-AutoRemoveJob] [-Force] [-Keep]
[-NoRecurse] [-Wait] [-WriteEvents] [-WriteJobInResults] [<CommonParameters>]
```

```
Receive-Job [-Job] <System.Management.Automation.Job[]> [[-Location]
<System.String[]>] [-AutoRemoveJob] [-Force] [-Keep] [-NoRecurse] [-Wait]
[-WriteEvents] [-WriteJobInResults] [<CommonParameters>]
```

```
Receive-Job [-Job] <System.Management.Automation.Job[]> [[-Session]
<System.Management.Automation.Runspaces.PSSession[]>] [-AutoRemoveJob]
[-Force] [-Keep] [-NoRecurse] [-Wait] [-WriteEvents] [-WriteJobInResults]
[<CommonParameters>]
```

```
Receive-Job [-Name] <System.String[]> [-AutoRemoveJob] [-Force] [-Keep]
[-NoRecurse] [-Wait] [-WriteEvents] [-WriteJobInResults] [<CommonParameters>]
```

DESCRIPTION

The `Receive-Job` cmdlet gets the results of PowerShell background jobs, such as those started by using the Start-Job` cmdlet or the AsJob parameter of any cmdlet. You can get the results of all jobs or identify jobs by their name, ID, instance ID, computer name, location, or session, or by submitting a job object.`

When you start a PowerShell background job, the job starts, but the results don't appear immediately. Instead, the command returns an object that represents the background job. The job object contains useful information about the job, but it doesn't contain the results. This method lets you continue to work in the session while the job runs. For more information about background jobs in PowerShell, see `about_Jobs (./About/about_Jobs.md)`.

The `Receive-Job` cmdlet gets the results that have been generated by the time that the Receive-Job` command is submitted. If the results aren't yet complete, you can run additional Receive-Job` commands to get the remaining results.`

By default, job results are deleted from the system when you receive them, but you can use the `Keep parameter to save the results so that you can receive them again. To delete the job results, run the Receive-Job` command again without the Keep parameter, close the session, or use the Remove-Job` cmdlet`

to delete the job from the session.

Starting in Windows PowerShell 3.0, `Receive-Job` also gets the results of custom job types, such as workflow jobs and instances of scheduled jobs. To enable `Receive-Job` to get the results a custom job type, import the module that supports the custom job type into the session before it runs a `Receive-Job` command, either by using the `Import-Module` cmdlet or by getting a cmdlet in the module. For information about a particular custom job type, see the documentation of the custom job type feature.

PARAMETERS

`-AutoRemoveJob <System.Management.Automation.SwitchParameter>`

Indicates that this cmdlet deletes the job after it returns the job results. If the job has more results, the job is still deleted, but `Receive-Job` displays a message.

This parameter works only on custom job types. It's designed for instances of job types that save the job or the type outside of the session, such as instances of scheduled jobs.

This parameter can't be used without the `Wait` parameter.

This parameter was introduced in Windows PowerShell 3.0.

`-ComputerName <System.String[]>`

Specifies an array of names of computers.

This parameter selects from among the job results that are stored on the local computer. It doesn't get data for jobs run on remote computers. To get job results that are stored on remote computers, use the `Invoke-Command` cmdlet to run a `Receive-Job` command remotely.

`-Force <System.Management.Automation.SwitchParameter>`

Indicates that this cmdlet continues waiting if jobs are in the Suspended or Disconnected state. By default, the Wait parameter of ``Receive-Job`` returns, or terminates the wait, when jobs are in one of the following states:

- Completed
- Failed
- Stopped
- Suspended
- Disconnected.

The Force parameter is valid only when the Wait parameter is also used in the command.

This parameter was introduced in Windows PowerShell 3.0.

`-Id <System.Int32[]>`

Specifies an array of IDs. This cmdlet gets the results of jobs with the specified IDs.

The ID is an integer that uniquely identifies the job in the current session. It's easier to remember and type than the instance ID, but it's unique only in the current session. You can type one or more IDs separated by commas. To find the ID of a job, use ``Get-Job``.

`-InstanceId <System.Guid[]>`

Specifies an array of instance IDs. This cmdlet gets the results of jobs

with the specified instance IDs.

An instance ID is a GUID that uniquely identifies the job on the computer.

To find the instance ID of a job, use the ``Get-Job`` cmdlet.

`-Job <System.Management.Automation.Job[]>`

Specifies the job for which results are being retrieved.

Enter a variable that contains the job or a command that gets the job. You can also pipe a job object to ``Receive-Job``.

`-Keep <System.Management.Automation.SwitchParameter>`

Indicates that this cmdlet saves the aggregated stream data in the system, even after you have received them. By default, aggregated stream data is erased after viewed with ``Receive-Job``.

Closing the session, or removing the job with the ``Remove-Job`` cmdlet also deletes aggregated stream data.

`-Location <System.String[]>`

Specifies an array of locations. This cmdlet gets only the results of jobs in the specified locations.

`-Name <System.String[]>`

Specifies an array of friendly names. This cmdlet gets the results of jobs that have the specified names. Wildcard characters are supported.

`-NoRecurse <System.Management.Automation.SwitchParameter>`

Indicates that this cmdlet gets results only from the specified job. By default, ``Receive-Job`` also gets the results of all child jobs of the specified job.

`-Session <System.Management.Automation.Runspaces.PSSession[]>`

Specifies an array of sessions. This cmdlet gets the results of jobs that were run in the specified PowerShell session (`PSSession`). Enter a variable that contains the `PSSession` or a command that gets the `PSSession` , such as a `Get-PSSession` command.

`-Wait <System.Management.Automation.SwitchParameter>`

Indicates that this cmdlet suppresses the command prompt until all job results are received. By default, `Receive-Job` immediately returns the available results.

By default, the `Wait` parameter waits until the job is in one of the following states:

- Completed

- Failed

- Stopped

- Suspended

- Disconnected

To direct the `Wait` parameter to continue waiting if the job state is `Suspended` or `Disconnected`, use the `Force` parameter together with the `Wait` parameter.

This parameter was introduced in Windows PowerShell 3.0.

`-WriteEvents <System.Management.Automation.SwitchParameter>`

Indicates that this cmdlet reports changes in the job state while it waits for the job to finish.

This parameter is valid only when the Wait parameter is used in the command and the Keep parameter is omitted.

This parameter was introduced in Windows PowerShell 3.0.

`-WriteJobInResults <System.Management.Automation.SwitchParameter>`

Indicates that this cmdlet returns the job object followed by the results.

This parameter is valid only when the Wait parameter is used in the command and the Keep parameter is omitted.

This parameter was introduced in Windows PowerShell 3.0.

`<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: Get results for a particular job -----

```
$job = Start-Job -ScriptBlock {Get-Process}
Start-Sleep -Seconds 1
Receive-Job -Job $job
```

These commands use the Job parameter of `Receive-Job` to get the results of a particular job.

The first command starts a job with `Start-Job` and stores the job object in the `$job` variable.

The second command uses the `Receive-Job` cmdlet to get the results of the

job. It uses the Job parameter to specify the job.

----- Example 2: Use the Keep parameter -----

```
$job = Start-Job -ScriptBlock {Get-Service dhcp, fakeservice}
```

```
Start-Sleep -Seconds 1
```

```
$job | Receive-Job -Keep
```

Cannot find any service with service name 'fakeservice'.

```
+ CategoryInfo      : ObjectNotFound: (fakeservice:String)
```

```
[Get-Service], ServiceCommandException
```

```
+ FullyQualifiedErrorId :
```

```
NoServiceFoundForGivenName,Microsoft.PowerShell.Commands.GetServiceCommand
```

```
+ PSComputerName     : localhost
```

```
Status Name          DisplayName
```

```
----- ----          -
```

```
Running dhcp         DHCP Client
```

```
$job | Receive-Job -Keep
```

Cannot find any service with service name 'fakeservice'.

```
+ CategoryInfo      : ObjectNotFound: (fakeservice:String)
```

```
[Get-Service], ServiceCommandException
```

```
+ FullyQualifiedErrorId :
```

```
NoServiceFoundForGivenName,Microsoft.PowerShell.Commands.GetServiceCommand
```

```
+ PSComputerName     : localhost
```

```
Status Name          DisplayName
```

```
----- ----          -
```

```
Running dhcp         DHCP Client
```

This example stores a job in the `\$job` variable, and pipes the job to the `Receive-Job` cmdlet. The `-Keep` parameter is also used to allow all

aggregated stream data to be retrieved again after first view.

----- Example 3: Get results of several background jobs -----

```
# Use the Invoke-Command cmdlet with the -AsJob parameter to start a
background job that
# runs a Get-Service command on three remote computers. Store the resulting
job object in
# the $j variable
$j = Invoke-Command -ComputerName Server01, Server02, Server03 -ScriptBlock
{Get-Service} -AsJob
# Display the value of the **ChildJobs** property of the job object in $j. The
display
# shows that the command created three child jobs, one for the job on each
remote
# computer. You could also use the -IncludeChildJobs parameter of the Get-Job
cmdlet.
$j.ChildJobs
```

Id	Name	State	HasMoreData	Location	Command
2	Job2	Completed	True	Server01	Get-Service
3	Job3	Completed	True	Server02	Get-Service
4	Job4	Completed	True	Server03	Get-Service

```
# Use the Receive-Job cmdlet to get the results of just the Job3 child job
that ran on the
# Server02 computer. Use the *Keep* parameter to allow you to view the
aggregated stream
# data more than once.
Receive-Job -Name Job3 -Keep
```

Status	Name	DisplayName	PSComputerName
--------	------	-------------	----------------

Running AeLookupSvc	Application Experience	Server02
Stopped ALG	Application Layer Gateway Service	Server02
Running Appinfo	Application Information	Server02
Running AppMgmt	Application Management	Server02

Example 4: Get results of background jobs on multiple remote computers

```
# Use the New-PSSession cmdlet to create three user-managed PSSessions on
three servers,
# and save the sessions in the $s variable.
$s = New-PSSession -ComputerName Server01, Server02, Server03
# Use Invoke-Command run a Start-Job command in each of the PSSessions in the
$s variable.
# The code creates a new job with a custom name to each server. The job
outputs the
# datetime from each server. Save the job objects in the $j variable.
$invokeCommandSplat = @{
    Session = $s
    ScriptBlock = {
        Start-Job -Name $('MyJob-' + $env:COMPUTERNAME) -ScriptBlock {
            (Get-Date).ToString()
        }
    }
}
$j = Invoke-Command @invokeCommandSplat
# To confirm that these job objects are from the remote machines, run Get-Job
to show no
# local jobs running.
Get-Job`
# Display the three job objects in $j. Note that the Localhost location is not
the local
# computer, but instead localhost as it relates to the job on each Server.
```

\$j

Id	Name	State	HasMoreData	Location	Command
1	MyJob-Server01	Completed	True	Localhost	(Get-Date).ToString()
2	MyJob-Server02	Completed	True	Localhost	(Get-Date).ToString()
3	MyJob-Server03	Completed	True	Localhost	(Get-Date).ToString()

Use Invoke-Command to run a Receive-Job command in each of the sessions in the \$s

variable and save the results in the \$results variable. The Receive-Job command must be

run in each session because the jobs were run locally on each server.

```
$results = Invoke-Command -Session $s -ScriptBlock {  
    Receive-Job -Name $('MyJob-' +$env:COMPUTERNAME)  
}
```

3/22/2021 7:41:47 PM

3/22/2021 7:41:47 PM

3/22/2021 9:41:47 PM

This example shows how to get the results of background jobs run on three remote computers. Unlike the previous example, using `Invoke-Command` to run the `Start-Job` command actually started three independent jobs on each of the three computers. As a result, the command returned three job objects representing three jobs run locally on three different computers.

----- Example 5: Access child jobs -----

```
Start-Job -Name TestJob -ScriptBlock {dir C:\, Z:\}
```

Without the Keep parameter, aggregated child job data is displayed once.

Then destroyed.

Receive-Job -Name TestJob

Directory: C:\

Mode	LastWriteTime	Length	Name
d-r---	1/24/2019 7:11 AM		Program Files
d-r---	2/13/2019 8:32 AM		Program Files (x86)
d-r---	10/3/2018 11:47 AM		Users
d-----	2/7/2019 1:52 AM		Windows

Cannot find drive. A drive with the name 'Z' does not exist.

+ CategoryInfo : ObjectNotFound: (Z:String) [Get-ChildItem],

DriveNotFoundException

+ FullyQualifiedErrorId :

DriveNotFound,Microsoft.PowerShell.Commands.GetChildItemCommand

+ PSComputerName : localhost

It would seem that the child job data is gone.

Receive-Job -Name TestJob

Using the object model, you can still retrieve child job data and streams.

\$job = Get-Job -Name TestJob

\$job.ChildJobs[0].Error

Cannot find drive. A drive with the name 'Z' does not exist.

+ CategoryInfo : ObjectNotFound: (Z:String) [Get-ChildItem],

DriveNotFoundException

+ FullyQualifiedErrorId :

DriveNotFound,Microsoft.PowerShell.Commands.GetChildItemCommand

+ PSComputerName : localhost

REMARKS

To see the examples, type: "get-help Receive-Job -examples".

For more information, type: "get-help Receive-Job -detailed".

For technical information, type: "get-help Receive-Job -full".

For online help, type: "get-help Receive-Job -online"