Get-Process

Module: Microsoft.PowerShell.Management

Gets the processes that are running on the local computer or a remote computer.

In this article

PowerShell Copy

```
Get-Process
   [[-Name] <String[]>]
   [-ComputerName <String[]>]
   [-Module]
   [-FileVersionInfo]
   [<CommonParameters>]
```

PowerShell Copy

```
Get-Process
   [[-Name] <String[]>]
   [-IncludeUserName]
   [<CommonParameters>]
```

PowerShell Copy

```
Get-Process
   -Id <Int32[]>
```

```
    [-IncludeUserName]
    [<CommonParameters>]
```

PowerShell Copy

```
Get-Process
    -Id <Int32[]>
    [-ComputerName <String[]>]
    [-Module]
    [-FileVersionInfo]
    [<CommonParameters>]
```

PowerShell Copy

```
Get-Process
    -InputObject <Process[]>
    [-IncludeUserName]
    [<CommonParameters>]
```

PowerShell Copy

```
Get-Process
    -InputObject <Process[]>
    [-ComputerName <String[]>]
    [-Module]
    [-FileVersionInfo]
    [<CommonParameters>]
```

Description

The **Get-Process** cmdlet gets the processes on a local or remote computer.

Without parameters, this cmdlet gets all of the processes on the local computer. You can also specify a particular process by process name or process ID (PID) or pass a process object through the pipeline to this cmdlet.

By default, this cmdlet returns a process object that has detailed information about the process and supports methods that let you start and stop the process. You can also use the parameters of the **Get-Process** cmdlet to get file version information for the program that runs in the process and to get the modules that the process loaded.

Examples

Example 1: Get a list of all active processes on the local computer

PowerShell Copy

```
PS C:\> Get-Process
```

This command gets a list of all active processes running on the local computer. For a definition of each column, see the "Additional Notes" section of the Help topic for Get-Help.

Example 2: Get all available data about one or more processes

PowerShell Copy

```
PS C:\> Get-Process winword, explorer | Format-List *
```

This command gets all available data about the Winword and Explorer processes on the computer. It uses the *Name* parameter to specify the processes, but it omits the optional parameter name. The pipeline operator (|) passes the data to the Format-List cmdlet, which displays all available properties (*) of the Winword and Explorer process objects.

You can also identify the processes by their process IDs. For instance, `Get-Process -Id 664, 2060`.

Example 3: Get all processes with a working set greater than a specified size

PowerShell Copy

```
PS C:\> Get-Process | Where-Object {$_.WorkingSet -gt 20000000}
```

This command gets all processes that have a working set greater than 20 MB. It uses the **Get-Process** cmdlet to get all running processes. The pipeline operator (|) passes the process objects to the Where-Object cmdlet, which selects only the object with a value greater than 20,000,000 bytes for the **WorkingSet** property.

WorkingSet is one of many properties of process objects. To see all of the properties, type `Get-Process | Get-Member`. By default, the values of all amount properties are in bytes, even though the default display lists them in kilobytes and megabytes.

Example 4: List processes on the computer in groups based on priority

PowerShell Copy

```
PS C:\> $A = Get-Process PS C:\> Get-Process -InputObject $A | Format-Table -View priority
```

These commands list the processes on the computer in groups based on their priority class. The first command gets all the processes on the computer and then stores them in the $A variable.

The second command uses the *InputObject* parameter to pass the process objects that are stored in the $A variable to the **Get-Process** cmdlet. The pipeline operator passes the objects to the **Format-Table**

cmdlet, which formats the processes by using the Priority view. The Priority view, and other views, are defined in the PS1XML format files in the Windows PowerShell home directory ($pshome).

Example 5: Add a property to the standard Get-Process output display

PowerShell Copy

```
PS C:\> Get-Process powershell -ComputerName S1, localhost |
    ft @{Label = "NPM(K)"; Expression = {[int]($_.NPM / 1024)}},
    @{Label = "PM(K)"; Expression = {[int]($_.PM / 1024)}},
    @{Label = "WS(K)"; Expression = {[int]($_.WS / 1024)}},
    @{Label = "VM(M)"; Expression = {[int]($_.VM / 1MB)}},
    @{Label = "CPU(s)"; Expression = {if ($_.CPU) {$_.CPU.ToString("N")}}},
    Id, MachineName, ProcessName -Auto

NPM(K) PM(K) WS(K) VM(M) CPU(s)   Id MachineName ProcessName
------ ----- ----- ----- ------   -- ----------- -----------
     6 23500 31340   142 1.70   1980 S1          powershell
     6 23500 31348   142 2.75   4016 S1          powershell
    27 54572 54520   576 5.52   4428 localhost   powershell
```

This example provides a `Format-Table` (alias = ft) command that adds the MachineName property to the standard `Get-Process` output display.

Example 6: Get version information for a process

PowerShell Copy

```
PS C:\> Get-Process powershell -FileVersionInfo



ProductVersion   FileVersion      FileName
--------------   -----------      --------
6.1.6713.1       6.1.6713.1 (f... C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe
```

This command uses the *FileVersionInfo* parameter to get the version information for the PowerShell.exe file that is the main module for the PowerShell process.

To run this command with processes that you do not own on Windows Vista and later versions of Windows, you must open Windows PowerShell with the Run as administrator option.

Example 7: Get modules loaded with the specified process

PowerShell Copy

```
PS C:\> Get-Process SQL* -Module
```

This command uses the *Module* parameter to get the modules that have been loaded by the process. This command gets the modules for the processes that have names that begin with SQL.

To run this command on Windows Vista and later versions of Windows with processes that you do not own, you must start Windows PowerShell with the Run as administrator option.

Example 8: Find the owner of a process

PowerShell Copy

```
PS C:\> Get-Process powershell -IncludeUserName

Handles      WS(K)   CPU(s)     Id UserName            ProcessName
-------      -----   ------     -- --------            -----------
    782     132080     2.08   2188 DOMAIN01\user01     powershell

PS C:\> $p = Get-WmiObject Win32_Process -Filter "name='powershell.exe'"
PS C:\> $p.GetOwner()


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 3
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
Domain           : DOMAIN01
ReturnValue      : 0
User             : user01
```

The first command shows how to find the owner of a process. The **IncludeUserName** parameter requires elevated user rights (Run as Administrator). The output reveals that the owner is Domain01\user01.

The second and third command are another way to find the owner of a process.

The second command uses `Get-WmiObject` to get the PowerShell process. It saves it in the $p variable.

The third command uses the GetOwner method to get the owner of the process in $p. The output reveals that the owner is Domain01\user01.

Example 9: Use an automatic variable to identify the process hosting the current session

PowerShell Copy

```
PS C:\> Get-Process powershell

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
```

```
308      26        52308      61780   567     3.18   5632 powershell
377      26        62676      63384   575     3.88   5888 powershell


PS C:\> Get-Process -Id $PID

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
396      26        56488      57236   575     3.90   5888 powershell
```

These commands show how to use the $PID automatic variable to identify the process that is hosting the current PowerShell session. You can use this method to distinguish the host process from other PowerShell processes that you might want to stop or close.

The first command gets all of the PowerShell processes in the current session.

The second command gets the PowerShell process that is hosting the current session.

Example 10: Get all processes that have a main window title and display them in a table

PowerShell Copy

```
PS C:\> Get-Process | where {$_.mainWindowTitle} | Format-Table id, name, mainwindowtitle -
autosize
```

This command gets all the processes that have a main window title, and it displays them in a table with the process ID and the process name.

The **mainWindowTitle** property is just one of many useful properties of the **Process** object that **Get-Process** returns. To view all of the properties, pipe the results of a **Get-Process** command to the Get-Member cmdlet (get-process | get-member).

Required Parameters

`-Id`

Specifies one or more processes by process ID (PID). To specify multiple IDs, use commas to separate the IDs. To find the PID of a process, type `Get-Process`.

Type:
Int32[]

Aliases:
PID

Position:

Named

Default value:
None

Accept pipeline input:
True (ByPropertyName)

Accept wildcard characters:
False

`-IncludeUserName`

Indicates that the UserName value of the **Process** object is returned with results of the command.

Type:
SwitchParameter

Position:
Named

Default value:
None

Accept pipeline input:
False

Accept wildcard characters:
False

`-InputObject`

Specifies one or more process objects. Enter a variable that contains the objects, or type a command or expression that gets the objects.

Type:
Process[]

Position:
Named

Default value:
None

Accept pipeline input:
True (ByValue)

Accept wildcard characters:
False

Optional Parameters

`-ComputerName`

Specifies the computers for which this cmdlet gets active processes. The default is the local computer.

Type the NetBIOS name, an IP address, or a fully qualified domain name (FQDN) of one or more computers. To specify the local computer, type the computer name, a dot (.), or localhost.

This parameter does not rely on Windows PowerShell remoting. You can use the *ComputerName* parameter of this cmdlet even if your computer is not configured to run remote commands.

Type:
String[]

Aliases:
Cn

Position:
Named

Default value:
None

Accept pipeline input:
True (ByPropertyName)

Accept wildcard characters:
False

`-FileVersionInfo`

Indicates that this cmdlet gets the file version information for the program that runs in the process.

On Windows Vista and later versions of Windows, you must open Windows PowerShell with the Run as administrator option to use this parameter on processes that you do not own.

You cannot use the *FileVersionInfo* and *ComputerName* parameters of the **Get-Process** cmdlet in the same command. To get file version information for a process on a remote computer, use the Invoke-Command cmdlet.

Using this parameter is equivalent to getting the **MainModule.FileVersionInfo** property of each process object. When you use this parameter, **Get-Process** returns a **FileVersionInfo** object (System.Diagnostics.FileVersionInfo), not a process object. So, you cannot pipe the output of the command to a cmdlet that expects a process object, such as Stop-Process.

Type:
SwitchParameter

Aliases:
FV, FVI

Position:
Named

Default value:
None

Accept pipeline input:
False

Accept wildcard characters:
False

`-Module`

Indicates that this cmdlet gets the modules that have been loaded by the processes.

On Windows Vista and later versions of Windows, you must open Windows PowerShell with the Run as administrator option to use this parameter on processes that you do not own.

You cannot use the *Module* and *ComputerName* parameters of the **Get-Process** cmdlet in the same command. To get the modules that have been loaded by a process on a remote computer, use the **Invoke-Command** cmdlet.

This parameter is equivalent to getting the Modules property of each process object. When you use this parameter, this cmdlet returns a **ProcessModule** object (System.Diagnostics.ProcessModule), not a process object. So, you cannot pipe the output of the command to a cmdlet that expects a process object, such as Stop-Process.

When you use both the *Module* and *FileVersionInfo* parameters in the same command, this cmdlet returns a **FileVersionInfo** object with information about the file version of all modules.

Type:
SwitchParameter

Position:
Named

Default value:
None

Accept pipeline input:
False

Accept wildcard characters:
False

-Name

Specifies one or more processes by process name. You can type multiple process names (separated by commas) and use wildcard characters. The parameter name ("Name") is optional.

Type:
String[]

Aliases:
ProcessName

Position:
0

Default value:
None

Accept pipeline input:
True (ByPropertyName)

Accept wildcard characters:
True

Inputs

System.Diagnostics.Process

You can pipe a process object to this cmdlet.

Outputs

System.Diagnostics.Process, System.Diagnotics.FileVersionInfo, System.Diagnostics.ProcessModule

By default, this cmdlet returns a **System.Diagnostics.Process** object. If you use the *FileVersionInfo* parameter, it returns a **System.Diagnotics.FileVersionInfo** object. If you use the *Module* parameter, without the *FileVersionInfo* parameter, it returns a **System.Diagnostics.ProcessModule** object.

Notes

- You can also refer to this cmdlet by its built-in aliases, ps and gps. For more information, see about_Aliases.
- On computers that are running a 64-bit version of Windows, the 64-bit version of Windows PowerShell gets only 64-bit process modules and the 32-bit version of Windows PowerShell gets only 32-bit process modules.
- You can use the properties and methods of the Windows Management Instrumentation (WMI) Win32_Process object in Windows PowerShell. For information, see Get-WmiObject and the WMI SDK.
- The default display of a process is a table that includes the following columns. For a description of all of the properties of process objects, see Process Properties in the MSDN library.
    - Handles: The number of handles that the process has opened.
    - NPM(K): The amount of non-paged memory that the process is using, in kilobytes.
    - PM(K): The amount of pageable memory that the process is using, in kilobytes.
    - WS(K): The size of the working set of the process, in kilobytes. The working set consists of the pages of memory that were recently referenced by the process.
    - VM(M): The amount of virtual memory that the process is using, in megabytes. Virtual memory includes storage in the paging files on disk.
    - CPU(s): The amount of processor time that the process has used on all processors, in seconds.
    - ID: The process ID (PID) of the process.
    - ProcessName: The name of the process.

    For explanations of the concepts related to processes, see the Glossary in Help and Support Center and the Help for Task Manager.
- You can also use the built-in alternate views of the processes available with Format-Table, such as StartTime and Priority, and you can design your own views.

Related Links

- Debug-Process
- Get-Process
- Start-Process
- Stop-Process
- Wait-Process