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

PowerShell Get-Help on command 'Stop-Process'

PS C:\Users\wahid> Get-Help Stop-Process

NAME

Stop-Process

SYNOPSIS

Stops one or more running processes.

SYNTAX

Stop-Process [-Id] <System.Int32[]> [-Force] [-PassThru] [-Confirm] [-WhatIf]
[<CommonParameters>]

Stop-Process [-InputObject] <System.Diagnostics.Process[]> [-Force]
[-PassThru] [-Confirm] [-WhatIf] [<CommonParameters>]

Stop-Process [-Force] -Name <System.String[]> [-PassThru] [-Confirm] [-WhatIf]
[<CommonParameters>]

DESCRIPTION

The `Stop-Process` cmdlet stops one or more running processes. You can specify
a process by process name or process ID (PID), or pass a process object to

`Stop-Process`. `Stop-Process` works only on processes running on the local computer.

On Windows Vista and later versions of the Windows operating system, to stop a process that is not owned by the current user, you must start PowerShell by using the Run as administrator option. Also, you are not prompted for confirmation unless you specify the Confirm parameter.

PARAMETERS

-Force <System.Management.Automation.SwitchParameter>

Stops the specified processes without prompting for confirmation. By default, `Stop-Process` prompts for confirmation before stopping any process that is not owned by the current user.

To find the owner of a process, use the `Get-CimInstance` cmdlet to get a Win32_Process object that represents the process, and then use the GetOwner method of the object.

-Id <System.Int32[]>

Specifies the process IDs of the processes to stop. To specify multiple IDs, use commas to separate the IDs. To find the PID of a process, type `Get-Process`.

-InputObject <System.Diagnostics.Process[]>

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

-Name <System.String[]>

Specifies the process names of the processes to stop. You can type multiple process names, separated by commas, or use wildcard characters.

-PassThru <System.Management.Automation.SwitchParameter>

Returns an object that represents the process. By default, this cmdlet
does not generate any output.

-Confirm <System.Management.Automation.SwitchParameter>
Prompts you for confirmation before running the cmdlet.

-WhatIf <System.Management.Automation.SwitchParameter>
Shows what would happen if the cmdlet runs. The cmdlet is not run.

<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: Stop all instances of a process ----------

PS C:\> Stop-Process -Name "notepad"

This command stops all instances of the Notepad process on the computer. Each
instance of Notepad runs in its own process. It uses the Name parameter to
specify the processes, all of which have the same name. If you were to use the
Id parameter to stop the same processes, you would have to list the process
IDs of each instance of Notepad.
------- Example 2: Stop a specific instance of a process -------

PS C:\> Stop-Process -Id 3952 -Confirm -PassThru

Confirm
Are you sure you want to perform this action?
Performing operation "Stop-Process" on Target "notepad (3952)".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help
(default is "Y"):y

```
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
41       2      996       3212    31           3952 notepad
```

This command stops a particular instance of the Notepad process. It uses the process ID, 3952, to identify the process. The Confirm parameter directs PowerShell to prompt you before it stops the process. Because the prompt includes the process namein addition to its ID, this is best practice. The PassThru parameter passes the process object to the formatter for display. Without this parameter, there would be no display after a `Stop-Process` command.

--- Example 3: Stop a process and detect that it has stopped ---

```
calc
$p = Get-Process -Name "calc"
Stop-Process -InputObject $p
Get-Process | Where-Object {$_.HasExited}
```

This series of commands starts and stops the `Calc` process, and then detects processes that have stopped.

The first command starts an instance of the calculator.

The second command uses `Get-Process` gets an object that represents the `Calc` process, and then stores it in the `$p` variable.

The third command stops the `Calc` process. It uses the InputObject parameter to pass the object to `Stop-Process`.

The last command gets all of the processes on the computer that were running but that are now stopped. It uses `Get-Process` to get all of the processes on the computer. The pipeline operator (`|`) passes the results to the `Where-Object` cmdlet, which selects the ones where the value of the HasExited

property is $True. HasExited is just one property of process objects. To find all the properties, type `Get-Process | Get-Member`.

--- Example 4: Stop a process not owned by the current user ---

```
PS> Get-Process -Name "lsass" | Stop-Process

Stop-Process : Cannot stop process 'lsass (596)' because of the following
error: Access is denied
At line:1 char:34
+ Get-Process -Name "lsass" | Stop-Process <<<<

[ADMIN]: PS> Get-Process -Name "lsass" | Stop-Process

Warning!
Are you sure you want to perform this action?
Performing operation 'Stop-Process' on Target 'lsass(596)'
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default
is "Y"):

[ADMIN]: PS> Get-Process -Name "lsass" | Stop-Process -Force
[ADMIN]: PS>
```

These commands show the effect of using Force to stop a process that is not owned by the user.

The first command uses `Get-Process` to get the Lsass process. A pipeline operator sends the process to `Stop-Process` to stop it. As shown in the sample output, the first command fails with an Access denied message, because this process can be stopped only by a member of the Administrator group on the computer.

When PowerShell is opened by using the Run as administrator option, and the command is repeated, PowerShell prompts you for confirmation.

The second command specifies Force to suppress the prompt. As a result, the process is stopped without confirmation.

REMARKS

To see the examples, type: "get-help Stop-Process -examples".

For more information, type: "get-help Stop-Process -detailed".

For technical information, type: "get-help Stop-Process -full".

For online help, type: "get-help Stop-Process -online"