Learning PowerShell command names

08/24/20185 minutes to read

In this article

Learning names of commands and parameters requires a significant time investment with most command-line interfaces. The issue is that there are few patterns. Memorization is the only way to learn the commands and parameters that you need to use on a regular basis.

When you work with a new command or parameter, you can't always use what you already know. You have to find and learn a new name. Traditionally, command-line interfaces start with a small set of tools and grow with incremental additions. It's easy to see why there's no standard structure. This seems logical for command names since each command is a separate tool. PowerShell has a better way to handle command names.

Learning command names in traditional shells

Most commands are built to manage elements of the operating system or applications, such as services or processes. The commands have names that may or may not fit into a family. For example, on Windows systems, you can use the `net start` and `net stop` commands to start and stop a service. **Sc.exe** is another service control tool for Windows. That name does not fit into the naming pattern for the **net.exe**service commands. For process management, Windows has the **tasklist.exe** command to list processes and the **taskkill.exe** command to kill processes.

Also, these commands have irregular parameter specifications. You can't use the `net start` command to start a service on a remote computer. The **sc.exe** command can start a service on a remote computer. But to specify the remote computer, you must prefix its name with a double backslash. To start the spooler service on a remote computer named DC01, you type `sc.exe \\DC01 start spooler`. To list tasks running on DC01, you use the **/S**parameter and the computer name without backslashes. For example, `tasklist /S DC01`.

Note

Prior to PowerShell v6, `sc` was an alias for the `Set-Content` cmdlet. Therefore, to run the **sc.exe** command in a version of PowerShell prior to v6, you must include the full filename **sc.exe**including the file extension **exe**.

Services and processes are examples of manageable elements on a computer that have well-defined life cycles. You may start or stop services and processes, or get a list of all currently running services or processes. Although there are important technical distinctions between them, the actions you perform on services and processes are conceptually the same. Furthermore, the choices we make to customize an action by specifying parameters may be conceptually similar as well.

PowerShell exploits these similarities to reduce the number of distinct names you need to know to understand and use cmdlets.

Cmdlets use verb-noun names to reduce command memorization

PowerShell uses a "verb-noun" naming system. Each cmdlet name consists of a standard verb hyphenated with a specific noun. PowerShell verbs are not always English verbs, but they express specific actions in PowerShell. Nouns are very much like nouns in any language. They describe specific types of objects that are important in system administration. It's easy to demonstrate how these two-part names reduce learning effort by looking at a few examples.

PowerShell has a recommended set of standard verbs. Nouns are less restricted, but always describe what the verb acts upon. PowerShell has commands such as `Get-Process`, `Stop-Process`, `Get-Service`, and `Stop-Service`.

For this example of two nouns and verbs, consistency does not simplify learning that much. Extend that list to a standardized set of 10 verbs and 10 nouns. Now you only have 20 words to understand. But those words can be combined to form 100 distinct command names.

It's easy to understand what a PowerShell command does by reading its name. The command to shut down a computer is `Stop-Computer`. The command to list all computers on a network is `Get-Computer`. The command to get the system date is `Get-Date`.

You can list all commands that include a particular verb with the **Verb** parameter for `Get-Command`. For example, to see all cmdlets that use the verb `Get`, type:

Copy

```
PS> Get-Command -Verb Get
```

```
CommandType     Name
Definition
-----------     ----
----------
Cmdlet          Get-Acl
Get-Acl [[-Path] <String[]>]...
Cmdlet          Get-Alias
Get-Alias [[-Name] <String[]...
Cmdlet          Get-AuthenticodeSignature
Get-AuthenticodeSignature [-...
Cmdlet          Get-ChildItem
Get-ChildItem [[-Path] <Stri...
...
```

Use the **Noun** parameter to see a family of commands that affect the same type of object. For example, run following command to see the commands available for managing services:

Copy

```
PS> Get-Command -Noun Service

CommandType     Name
Definition
-----------     ----
----------
Cmdlet          Get-Service
Get-Service [[-Name] <String...
Cmdlet          New-Service
New-Service [-Name] <String>...
Cmdlet          Restart-Service
Restart-Service [-Name] <Str...
Cmdlet          Resume-Service
Resume-Service [-Name] <Stri...
Cmdlet          Set-Service
Set-Service [-Name] <String>...
```

```
Cmdlet          Start-Service
Start-Service [-Name] <Strin...
Cmdlet          Stop-Service
Stop-Service [-Name] <String...
Cmdlet          Suspend-Service
Suspend-Service [-Name] <Str...
...
```

Cmdlets use standard parameters

As noted earlier, commands used in traditional command-line interfaces don't always have consistent parameter names. Parameters are often single-character or abbreviated words that are easy to type but aren't easily understood by new users.

Unlike most other traditional command-line interfaces, PowerShell processes parameters directly, and it uses this direct access to the parameters along with developer guidance to standardize parameter names. This guidance encourages but does not guarantee that every cmdlet conforms to the standard.

PowerShell also standardizes the parameter separator. Parameter names always have a '-' prepended to them with a PowerShell command. Consider the following example:

PowerShell

Copy

```powershell
Get-Command -Name Clear-Host
```

The parameter's name is **Name**, but it is typed as `-Name` when used on the command line as a parameter.

Here are some of the general characteristics of the standard parameter names and usages.

The Help parameter (?)

When you specify the `-?` parameter on any cmdlet, PowerShell displays help for the cmdlet. The cmdlet is not executed.

Common parameters

PowerShell has several *common parameters*. These parameters are controlled by the PowerShell engine. Common parameters always behave the same way. The common parameters are **WhatIf**, **Confirm**, **Verbose**, **Debug**, **Warn**, **ErrorAction**, **ErrorVariable**, **OutVariable**, and **OutBuffer**.

Recommended parameter names

The PowerShell core cmdlets use standard names for similar parameters. The use of these standard names is not enforced, but there is explicit guidance to encourage standardization.

For example, the recommended name for a parameter that refers to a computer is **ComputerName**, rather than Server, Host, System, Node, or some other common alternative. Other important recommended parameter names are **Force**, **Exclude**, **Include**, **PassThru**, **Path**, and **CaseSensitive**.