Using familiar command names

08/27/2018 • 2 minutes to read •

In this article

PowerShell supports aliases to refer to commands by alternate names. Aliasing allows users with experience in other shells to use common command names that they already know for similar operations in PowerShell.

Aliasing associates a new name with another command. For example, PowerShell has an internal function named `Clear-Host` that clears the output window. You can type either the `cls` or `clear` alias at a command prompt. PowerShell interprets these aliases and runs the `Clear-Host` function.

This feature helps users to learn PowerShell. First, most **cmd.exe** and Unix users have a large repertoire of commands that users already know by name. The PowerShell equivalents may not produce identical results. However, the results are close enough that users can do work without knowing the PowerShell command name. "Finger memory" is another major source of frustration when learning a new command shell. If you have used **cmd.exe** for years, you might reflexively type the `cls` command to clear the screen. Without the alias for `Clear-Host`, you receive an error message and won't know what to do to clear the output.

The following list shows a few of the common **cmd.exe** and Unix commands that you can use in PowerShell:

cat	dir	mount	rm
cd	echo	move	rmdir
chdir	erase	popd	sleep
clear	h	ps	sort
cls	history	pushd	tee
copy	kill	pwd	type
del	lp	r	write
diff	ls	ren	

The `Get-Alias` cmdlet shows you the real name of the native PowerShell command associated with an alias.

PowerShell

```powershell
PS> Get-Alias cls
```

Output

```
CommandType     Name
Version    Source
-----------     ----
-------    ------
Alias           cls -> Clear-Host
```

Interpreting standard aliases

The aliases we described previously were designed for name-compatibility with other command shells. Most aliases built into PowerShell are designed for brevity. Shorter names are easier to type, but are difficult to read if you don't know what they refer to.

PowerShell aliases try to compromise between clarity and brevity. PowerShell uses a standard set of aliases for common nouns and verbs.

Example abbreviations:

Noun or Verb	Abbreviation
Get	g
Set	s
Item	i
Location	l
Command	cm
Alias	al

These aliases are understandable when you know the shorthand names.

Cmdlet name	Alias
`Get-Item`	gi
`Set-Item`	si
`Get-Location`	gl
`Set-Location`	sl
`Get-Command`	gcm

Cmdlet name	Alias
`Get-Alias`	gal

Once you're familiar with PowerShell aliasing, it's easy to guess that the **sal** alias refers to `Set-Alias`.

Creating new aliases

You can create your own aliases using the `Set-Alias` cmdlet. For example, the following statements create the standard cmdlet aliases previously discussed:

PowerShell

Copy

```powershell
Set-Alias -Name gi -Value Get-Item
Set-Alias -Name si -Value Set-Item
Set-Alias -Name gl -Value Get-Loca-
tion
Set-Alias -Name sl -Value Set-Loca-
tion
Set-Alias -Name gcm -Value Get-Com-
mand
```

Internally, PowerShell uses similar commands during startup, but these aliases are not changeable. If you try to execute one of these commands, you get an error explaining that the alias can't be modified. For example:

```
PS> Set-Alias -Name gi -Value Get-Item
Set-Alias : Alias is not writeable because alias gi is read-only or constant and cannot be written to.
At line:1 char:10
+ Set-Alias  <<<< -Name gi -Value Get-Item
```