A.K.A. – Using PowerShell Aliases

An interesting capability baked into the PowerShell environment is alias commands. Much like we can find in IOS (The Cisco one that is) and with other CLI environments, there are short commands available to save you some typing during your day-to-day scripting.

Aliases can be found very easily with the Get-Alias CmdLet. The resulting output of the **Get-Alias** CmdLet is a table of aliases and their associated full CmdLet name.

There are a few different opinions about using aliases versus using full command syntax. My preference for writing scripts is to use full CmdLet structure. This is for readability, and because quite often I'm sharing my code with my peers who may have less familiarity with the commands. With that said, I do use aliases in the interactive shell to save time.

Aliases that conflict with commands

This is something that you may discover if you were used to using the command line for administration with other tools like the Windows Resource Kit, or even with native command line tools that came with Microsoft Windows.

An example of this is the **SC.EXE** tool. This is the Service Control utility which was a Resource Kit tool with Windows 2000, but was packaged into the OS in recent releases. You can use this to start, stop, install, uninstall and perform other management tasks with Windows services.

The challenge with this comes when you are using the PowerShell command shell there is a conflict. In PowerShell SC is an alias for the **Set-Content** CmdLet.

Traditional DOS command shell

In a regular DOS shell (cmd.exe) we run the SC command and get the expected results because it locates the SC.EXE in the system path as we see here:

PowerShell command shell

In the PowerShell session we get a different result because the priority to PowerShell CmdLets and functions trumps the Windows path and this is the result when we try to use the **SC** command:

This is one example, but there are others that you may encounter depending on what tools you use. The workaround is to test your scripts thoroughly, and where conflicts occur, you should use the **Invoke-Expression** CmdLet referring to the literal path to the executable file.

Setting your own Aliases

One of the creative ways to leverage PowerShell is to create your own aliases for commands and functions. For those who are new to PowerShell, this may not be the first thing that you run to, but once you begin to expand your PowerShell toolkit to include your own functions you will quickly find that aliases can add that extra layer of awesome to your shell.

As an example, I want to create a function named DiscoPosse which will search the current directory for a file which contains the phrase DiscoPosse in the file name.

```
function DiscoPosse { dir . | where { $_.name -match “DiscoPosse” } }
```

And once we define our function, we can call it by typing the function name in the shell as seen below:

Another cool feature is that once we define a new function you can use the auto-complete by typing the first part of the command (e.g. discop) and then hitting the Tab key to complete the command.

Let’s say that I want to assign the alias diggity to my new DiscoPosse function. The way to do this is to use the **Set-Alias** CmdLet with the syntax of ***Set-Alias <alias name> <command name>*** as follows:

```
Set-Alias diggity DiscoPosse
```

Now we test out our alias by typing diggity in the PowerShell command line to see the results:

Pretty cool isn’t it? Once again we can also use the Tab auto-complete capability because it also extends into aliases.

I hope that this is a useful tip, and in a future post I’ll delve further into how you can back up and restore your custom aliases to allow you to transfer them from machine to machine. In the spirit of “less is more” you will learn to love aliases as a part of the great PowerShell toolkit.

DiscoPosse Review: Windows PowerShell for Developers by Doug Finke – O’Reilly Media

For those fans of PowerShell, and for those just getting on-board, PowerShell MVP Doug Finke has built the bridge for your journey with his latest book Windows PowerShell for Developers from O'Reilly Media.

As a long time user of PowerShell for everything from quick scripts to more challenging workflows and processes, this book has really upped my game. Doug's conversational and well-worded writing style help to bring you through the process of firming up, and advancing your skills with the PowerShell environment.

The book covers the exciting new features of PowerShell 3.0, while also adding to the foundation of PowerShell scripting that is available today so don't be afraid to jump right in even if you haven't had any exposure to 3.0 yet.

While having development experience will help, I found that it was not necessary. The well crafted instructional style guides you through how to extend the already great capabilities of PowerShell with .NET tools and a number of other widely available standards and tools such as XML, JSON, web services and Microsoft Excel. Even as more of a scripter than a developer it was well within my level of understanding.

Along with the chapters come great example code which you can download, test, tweak and even reuse which was very helpful with the more advanced concepts that are touched on in the book. I found that I was able to read the book from cover to cover, and then revisit to dig into the specific topics and dive deeper into the great examples and map them against some real use cases that I wanted to try.

There are also lots of great references within the chapters to external resources. The explanation of the history of PowerShell and other web concepts within help to either add to, or reinforce your knowledge. Whether you are new to PowerShell, are transitioning from other scripting languages, or have been with PowerShell since the beginning, you will get value from this book.

With the release of Windows Server 2012 and Windows 8 around the corner, the game is changing and this will get you on the road to great automation, workflow, data management and building exciting scripts and tools to leverage the over 2300 CmdLets that will become available in those environments.

I highly recommend that you get on over to O'Reilly, so click here to get the book and I hope that you will enjoy it and gain as much value from it as I have.

Thanks to Doug and to the great folks at O'Reilly Media! And to quote Doug: "If you repeat it, PowerShell it."

Get-Started: PowerShell 3.0 – Mastering parameters with the Show-Command CmdLet

Welcome to the first PowerShell 3.0 post in the Get-Started learning series. Today we are going to

work with PowerShell 3.0 and the **Show-Command** CmdLet. Because this is a PowerShell 3.0 CmdLet I am assuming that you are working with Windows 8, Windows Server 2012 or that you have installed the PowerShell CTP for your existing environment.

The **Show-Command** CmdLet is new with PowerShell 3.0 and what it brings us is a really great way to explore and extend our understanding of parameters.

For our example we will use the Out-FileCmdLet as our target. In your PowerShell console you simply type ***Show-Command <CmdLet name here>***

```
Show-Command Out-File
```

This will bring up a GUI window that presents all of the parameters you can edit for this particular CmdLet. It is context sensitive so the parameters will change depending on which CmdLet you launch.

This is what we see with our **Out-File** CmdLet:

We can see that for **Out-File** we can use the following parameters:

- Append
- Confirm
- Encoding
- Force
- InputObject
- NoClobber
- WhatIf
- Width

You would find these same parameters listed if you were to use the ***Get-Help Out-File*** command also, but for many people it can be confusing to understand the help format and it is especially helpful for those who are used to working with the GUI.

One extra feature we have is that not only are the CmdLet specific parameters shown, but there are also Common Parameters available. These include:

- Debug
- ErrorAction
- ErrorVariable
- OutBuffer
- OutVariable
- Verbose
- WarningAction
- WarningVariable

Common Parameters are those which apply to all CmdLets so these will be more familiar once you

get used to using them. You may not need to use them, but it is good to explore them as your skill level increases. They will add great features to your scripts when you begin to work with error handling.

In this case I will only add a couple of parameters which are ***-Append, -NoClobber*** and my file path which is ***c:tempMyOutputFile.txt***.

Once we populate our parameters we can click on the **Run** button to launch the full command. But before we do that, you may also notice that there is a **Copy** button. It is exactly what you think it is, which is a way to copy the full command to the clipboard for pasting elsewhere.

Once we click the **Copy** button we can now paste the output into your editor of choice to use for a script file. I'm a big fan of Notepad myself so this is where I will paste my result to see what the full command looks like.

What I recommend is that you use the **Show-Command** CmdLet to explore what parameters can be used for your various CmdLets and then eventually you will build your comfort and understanding of the capabilities and options of each one for more advanced scripting.

One note about the **Show-Command** is that because it is a GUI, you can only use it on the console and not in a remote PowerShell session (PSSession).

I recommend that you go through as many CmdLets with the **Show-Command** CmdLet now and get more and more experience with how to get the most of each and every CmdLet you use.

Happy scripting!

Get-Started – The DiscoPosse PowerShell 101 series

With so much focus on PowerShell for systems administrators today, one thing I wanted to do was to bring scripting to the non-scripters. What I have found quite often is that many of today's new administrators have not had any exposure to shell commands and scripting because they have not had a need to during early training.

This is why I am introducing the DiscoPosse Get-Started series. These posts will be geared towards those who are new to using scripting and the PowerShell language.

Since I have come up from a command line world with DOS and Linux and moved into Microsoft Windows afterwards, the jump back to dabble in the command shell was an easy transition. My goal

with this series is to bring that comfortability with the shell to those who are new to it.

 Nearly every task that can be performed in the GUI can be done with a PowerShell CmdLet or script. Even more importantly, some things can be done in the PowerShell console that cannot be done in the GUI.

We are going to start with simple walk throughs with the CmdLets and understanding aspects of the PowerShell language which will ramp up your ability and comfort with the language and its capabilities.

Thank you for coming here to learn, and I hope that you enjoy the Get-Started series!

Get-Started Series Article List (links will be added as new articles are online)

PowerShell 3.0 Series

- **Get-Started: PowerShell 3.0 – Mastering parameters with the Show-Command**

PowerShell 2.0 Series

- **Getting Help – Using Get-Command and Get-Help (COMING SOON!)**