

# CNT 4603: System Administration Spring 2012

## Scripting – Windows PowerShell – Part 3

Instructor : Dr. Mark Llewellyn  
markl@cs.ucf.edu  
HEC 236, 4078-823-2790  
<http://www.cs.ucf.edu/courses/cnt4603/spr2012>

Department of Electrical Engineering and Computer Science  
Computer Science Division  
University of Central Florida



# The PowerShell Environment

- The version of PowerShell that we are looking at is a standard CLI (Command Line Interface) shell.
- The syntax for using PowerShell from the CLI is similar to the syntax used for other CLI shells.
- The fundamental component of a PowerShell command, is of course, the name of the command to be executed.
- In addition, the command can be made more specific by using parameters and arguments to the parameters.
- Therefore, a PowerShell command can have any of the formats shown on the next page.



# The PowerShell Environment

```
[command]
```

```
[command] -[parameter]
```

```
[command] -[parameter] -[parameter] [argument1]
```

```
[command] -[parameter] -[parameter] [argument1],[argument2]
```

- In PowerShell, a **parameter** is a variable that can be accepted by the command, script or function. An **argument** is a value assigned to a parameter.
- Although these terms are often used interchangeably, remembering the difference will be helpful when working with PowerShell.
- The following page illustrates all of these forms:



```

C:\Users\Administrator\MyScripts>dir
Volume in drive C has no label.
Volume Serial Number is 585F-AF04

Directory of C:\Users\Administrator\MyScripts

10/26/2011  11:23 AM    <DIR>          .
10/26/2011  11:23 AM    <DIR>          ..
10/26/2011  10:08 AM             138 ArrayScript.ps1
10/26/2011  11:23 AM             138 ArrayScript2.ps1
10/26/2011  11:42 AM             150 ArrayScript3.ps1
10/26/2011  11:23 AM             138 ArrayScript4.ps1
         4 File(s)                564 bytes
         2 Dir(s)  27,381,649,408 bytes free

```

```

C:\Users\Administrator\MyScripts>dir /w
Volume in drive C has no label.
Volume Serial Number is 585F-AF04

Directory of C:\Users\Administrator\MyScripts

[.]          [...]          ArrayScript.ps1      ArrayScript2.ps1
ArrayScript3.ps1  ArrayScript4.ps1
         4 File(s)                564 bytes
         2 Dir(s)  27,381,649,408 bytes free

```

```

C:\Users\Administrator\MyScripts>dir /w A*.ps1
Volume in drive C has no label.
Volume Serial Number is 585F-AF04

Directory of C:\Users\Administrator\MyScripts

ArrayScript.ps1      ArrayScript2.ps1      ArrayScript3.ps1      ArrayScript4.ps1
         4 File(s)                564 bytes
         0 Dir(s)  27,381,649,408 bytes free

```

```

C:\Users\Administrator\MyScripts>dir /w A*3.ps1, A*4.ps1
Volume in drive C has no label.
Volume Serial Number is 585F-AF04

Directory of C:\Users\Administrator\MyScripts

Directory of C:\Users\Administrator\MyScripts

ArrayScript3.ps1      ArrayScript4.ps1
         2 File(s)                288 bytes
         0 Dir(s)  27,381,649,408 bytes free

```

```

C:\Users\Administrator\MyScripts>

```



Administrator: Windows PowerShell

```
PS C:\users\Administrator\MyScripts> get-process -Name PowerShell
```

Handles	NPM(K)	PM(K)	WS(K)	UM(M)	CPU(s)	Id	ProcessName
253	7	32020	32936	168	0.95	4028	powershell

```
PS C:\users\Administrator\MyScripts> get-process
```

Handles	NPM(K)	PM(K)	WS(K)	UM(M)	CPU(s)	Id	ProcessName
46	2	1024	3804	52	0.14	1456	ApacheMonitor
71	3	2968	6020	60	0.59	1164	cmd
527	5	1724	5164	105	1.03	492	csrss
267	7	7192	7780	54	15.02	536	csrss
232	7	5812	12380	68	0.47	1476	dllhost
75	3	1292	4220	49	0.39	2500	dwm
546	14	22400	33320	171	31.08	2572	explorer
176	12	11532	13500	88	0.20	1296	httpd
99	6	6100	9912	60	0.19	3180	httpd
0	0	0	24	0	0	0	Idle
199	6	3160	7440	77	0.22	2568	jucheck
110	3	1504	5656	69	0.11	2768	jusched
596	9	2968	8300	44	1.69	636	lsass
161	3	1580	3780	29	0.17	644	lsm
167	7	2816	7120	60	0.19	772	msdtc
504	6	48424	23020	105	0.47	1660	mysqld
184	7	14968	23984	123	5.28	3824	notepad++
278	7	29388	30304	168	1.02	4028	powershell
240	6	2072	6156	36	2.58	624	services
95	3	5396	9616	39	1.50	1016	SLsvc
28	1	252	728	4	0.14	424	smss
320	10	6760	12264	99	2.16	1524	spoolsv
298	4	2628	6216	38	3.78	804	svchost
261	7	2768	6220	35	0.61	864	svchost
303	10	5612	8616	46	3.66	940	svchost
148	4	2852	5796	35	0.09	984	svchost
1028	37	38868	47492	182	5.31	1000	svchost
573	17	5932	10800	58	0.72	1072	svchost
250	8	7032	8544	66	0.55	1128	svchost
408	14	14464	15920	87	2.53	1172	svchost
266	22	5932	9868	46	0.66	1324	svchost

To direct input to this virtual machine, press Ctrl+G.



# The PowerShell Environment

- As with all CLI-based shells, you need to understand how to navigate the PowerShell CLI to use it effectively.
- The table on the following page lists the editing operations associated with various keys when using the PowerShell Console.
- Most of the features of PowerShell are native to the `cmd` prompt, which makes PowerShell adoption easier for administrators already familiar with the Windows command line.
- The only major difference is the Tab key which is enhanced in PowerShell beyond the capabilities in the `cmd` prompt. In PowerShell the Tab key can be used to auto-complete commands, variables, parameter names, and even allowable operations on variables. Try some out!



# PowerShell Console Editing Features

Keys	Editing Operation
Left and Right Arrows	Moves cursor left and right through the current command line.
Up and Down Arrows	Move up and down through the list of recently typed commands.
Insert	Switches between insert and overstrike text-entry modes.
Delete	Deletes the character at the current cursor position
Backspace	Deletes the character immediately preceding the current cursor position.
F7	Displays a list of recently typed commands in a pop-up window in the command shell. Use the up and down arrows to select a previously typed command, and then press Enter to execute the selected command. Use the ESC key to hide pop-up window.
Tab	Auto-completes command line sequences. Use the Shift+Tab sequence to move backward through a list of potential matches.







# Understanding Cmdlets In PowerShell

- Cmdlets are a fundamental part of Powershell's functionality. They are implemented as managed classes (built on the .NET Framework) that include a well-defined set of methods to process data.
- A cmdlet developer writes the code that runs when the cmdlet is classed and compiles the code into a DLL that's loaded into a PowerShell instance when the shell is started.
- You already saw in a previous set of notes that cmdlets are always named with the format **Verb-Noun** where the verb specifies the action and the noun specifies the object to operate on.



# Understanding Cmdlets In PowerShell

- Because cmdlets derive from a base class, a number of common parameters, which are available to all cmdlets, can be used to help provide a more consistent interface for PowerShell cmdlets.
- These common parameters are shown in the tables on the next two pages.



# Common Cmdlet Parameters In PowerShell

Parameter	Data Type	Description
Verbose	Boolean	Generates detailed information about the operation, much like tracing or a transaction log. This parameter is effective only in cmdlets that generate verbose data.
Debug	Boolean	Generates programmer-level detail about the operation. The cmdlet must support the generation of debug data for this parameter to be effective.
ErrorAction	Enum	Determines how the cmdlet responds when an error occurs. Values are Continue (default), Stop, SilentlyContinue, and Inquire.
ErrorVariable	String	Specifies a variable that stores errors from the cmdlet during processing. This variable is populated in addition to <code>\$error</code> .
OutVariable	String	Specifies a variable that stores output from the cmdlet during processing.



# Common Cmdlet Parameters In PowerShell

Parameter	Data Type	Description
OutBuffer	Int32	Determines the number of objects to buffer before calling the next cmdlet in the pipeline.
WhatIf	Boolean	Explains what happens if the cmdlet is executed but doesn't actually execute the command.
Confirm	Boolean	Prompts the user for permission before performing any action that modifies the system.

**NOTE:** The last two parameters in the table, WhatIf and Confirm, are special in that they require a cmdlet to support the .NET method ShouldProcess, which might not be true for all cmdlets. The ShouldProcess method confirms the operation with the user, sending the name of the resource to be changed for confirmation before performing the operation.



# Understanding Cmdlets In PowerShell

- When you're first starting to work in PowerShell, the `get-help` and `get-command` cmdlets can be quite useful.
- You already saw a few instances of the `get-help` cmdlet in the first set of PowerShell notes.
- PowerShell has two parameters for the `get-help` cmdlet:
  - `detailed`, and `-full`.
- The `-detailed` parameter displays additional information about a cmdlet, including descriptions of parameters and examples of using the cmdlet. The `-full` parameter displays the entire help file for a cmdlet, including technical information about parameters.
- The table on the next page illustrates the sections returned by `help`.



## Components of `get-help`

Help Section	Description
Name	The name of the cmdlet
Synopsis	A brief description of what the cmdlet does.
Detailed Description	A detailed description of the cmdlet's behavior, usually including usage examples.
Syntax	Specific usage details for entering commands with the cmdlet.
Parameters	Valid parameters that can be used with this cmdlet.
Input Type	The type of input this cmdlet accepts
Output Type	The type of data this cmdlet returns
Terminating Errors	If present, identifies any errors that result in the cmdlet terminating prematurely.
Non-Terminating Errors	Identifies noncritical errors that might occur while the cmdlet is running but don't cause the cmdlet to terminate.
Notes	Additional details on the cmdlet
Examples	Common usages examples for the cmdlet
Related Links	References to other cmdlets that perform similar tasks.



# Understanding Cmdlets In PowerShell

- The `get-command` cmdlet is also quite useful as it lists all the available cmdlets in a PowerShell session.
- It is more powerful than `get-help` because it lists all available commands, including cmdlets, scripts, aliases, functions, and native applications in a PowerShell session.
- The next couple of pages illustrate some variations of the `get-command` cmdlet.



Administrator: Windows PowerShell

PS C:\users\Administrator\MyScripts> **get-command**

CommandType	Name	Definition
Alias	%	ForEach-Object
Alias	?	Where-Object
Function	A:	Set-Location A:
Alias	ac	Add-Content
Cmdlet	Add-Computer	Add-Computer [-DomainName] <String>
Cmdlet	Add-Content	Add-Content [-Path] <String[]> [-Ual
Cmdlet	Add-History	Add-History [[-InputObject] <PSObjec
Cmdlet	Add-Member	Add-Member [-MemberType] <PSMemberTy
Cmdlet	Add-PSSnapin	Add-PSSnapin [-Name] <String[]> [-Pa
Cmdlet	Add-Type	Add-Type [-TypeDefinition] <String>
Alias	asnp	Add-PSSnapIn
Function	B:	Set-Location B:
Function	C:	Set-Location C:
Alias	cat	Get-Content
Alias	cd	Set-Location
Function	cd..	Set-Location ..
Function	cd\	Set-Location \
Alias	chdir	Set-Location
Cmdlet	Checkpoint-Computer	Checkpoint-Computer [-Description] <
Alias	clc	Clear-Content
Alias	clear	Clear-Host
Cmdlet	Clear-Content	Clear-Content [-Path] <String[]> [-F
Cmdlet	Clear-EventLog	Clear-EventLog [-LogName] <String[]>
Cmdlet	Clear-History	Clear-History [[-Id] <Int32[]> [[-C
Function	Clear-Host	\$space = New-Object System.Management
Cmdlet	Clear-Item	Clear-Item [-Path] <String[]> [-Forc
Cmdlet	Clear-ItemProperty	Clear-ItemProperty [-Path] <String[]
Cmdlet	Clear-Variable	Clear-Variable [-Name] <String[]> [-
Alias	clhy	Clear-History
Alias	cli	Clear-Item
Alias	clp	Clear-ItemProperty
Alias	cls	Clear-Host
Alias	clv	Clear-Variable
Alias	compare	Compare-Object
Cmdlet	Compare-Object	Compare-Object [-ReferenceObject] <P
Cmdlet	Complete-Transaction	Complete-Transaction [-Verbose] [-De
Cmdlet	Connect-WSMan	Connect-WSMan [[-ComputerName] <Stri





Administrator: Windows PowerShell

```

PS C:\users\Administrator\MyScripts> get-command ipconfig | format-list *

HelpUri           :
FileVersionInfo  : File:           C:\Windows\system32\ipconfig.exe
                  InternalName:   ipconfig.exe
                  OriginalFilename: ipconfig.exe.mui
                  FileVersion:    6.0.6001.18000 (longhorn_rtm.080118-1840)
                  FileDescription: IP Configuration Utility
                  Product:         Microsoft Windows Operating System
                  ProductVersion:  6.0.6001.18000
                  Debug:           False
                  Patched:         False
                  PreRelease:      False
                  PrivateBuild:    False
                  SpecialBuild:    False
                  Language:        English (United States)

Path              : C:\Windows\system32\ipconfig.exe
Extension         : .exe
Definition        : C:\Windows\system32\ipconfig.exe
Visibility        : Public
OutputType        : <System.String>
Name              : ipconfig.exe
CommandType       : Application
ModuleName        :
Module            :
Parameters        :
ParameterSets     :

PS C:\users\Administrator\MyScripts>

```

Windows taskbar showing Start button, taskbar buttons for Administrator: Windo..., new 1 - Notepad++, system tray icons, and the time 3:40 PM. Below the taskbar is a VMware toolbar with icons for power, refresh, save, print, and a VMware logo.

# Variables In PowerShell

- In most shells, the only data that can be stored in a variable is text data. In advanced shells and programming languages, data stored in a variable can be almost anything, from strings, to sequences of objects.
- Similarly, PowerShell variables can hold just about anything.
- To define a PowerShell variable, you must name it with the \$ prefix, which helps delineate variables from aliases, cmdlets, filenames, and other items a shell operator might need to use.
- A variable name is case sensitive and can contain any combination of alphanumeric characters (A-Z,a-z,0-9) and the underscore (\_) character.



`$MSPProcesses = get-process | where {$_.company -match ".*Microsoft*"}`

```
PS C:\users\Administrator\MyScripts> $MSPProcesses = get-process | where {$_.company -match ".*Microsoft*"
PS C:\users\Administrator\MyScripts> $MSPProcesses
```

Handles	NPM(K)	PM(K)	WS(K)	UM(M)	CPU(s)	Id	ProcessName
519	5	1672	5076	105	0.86	496	csrss
256	8	6912	7224	111	7.14	540	csrss
233	7	5836	12404	68	0.38	2512	dllhost
75	3	1320	4212	49	0.45	1880	dwm
450	12	16936	26508	157	8.05	2116	explorer
572	9	2976	8268	44	1.39	640	lsass
159	3	1560	3764	29	0.06	648	lsm
167	7	2808	7112	60	0.20	2680	msdtc
406	8	34128	35756	179	2.53	2480	powershell
240	6	2100	6136	36	2.47	628	services
95	3	5368	9604	39	1.19	1020	SLsvc
28	1	252	728	4	0.23	428	smss
307	10	6924	12452	100	1.97	1524	spoolsv
297	4	2672	6260	38	4.11	808	svchost
253	7	2732	6160	35	0.34	868	svchost
287	9	5228	8092	46	1.53	944	svchost
124	3	1628	4664	32	0.06	988	svchost
1019	35	39880	49036	184	6.28	1004	svchost
577	16	5920	10760	59	0.73	1080	svchost
249	8	6996	8496	66	0.55	1132	svchost
406	13	14168	15396	87	1.30	1184	svchost
269	22	5964	9928	47	0.50	1328	svchost
125	5	1856	5268	36	0.11	1772	svchost
73	2	832	2880	23	0.02	1784	svchost
44	1	540	2284	15	0.00	1932	svchost
226	7	3168	4928	50	0.08	3088	svchost
137	5	1868	5904	53	0.06	1432	taskeng
247	7	2812	7584	73	0.28	2072	taskeng
98	4	1160	3932	41	0.20	548	wininit
118	3	1232	4368	32	0.41	580	winlogon
83	3	2468	5260	69	0.08	3700	wuauc lt

The variable \$MSPProcesses holds a collection of Microsoft processes that are currently running on the system.

```
PS C:\users\Administrator\MyScripts> -
```



# Variables In PowerShell

- When a PowerShell session is started, a number of built-in variables are defined automatically.
- These variables are often helpful with various system administration duties. Becoming familiar with them as well as their default values is recommended.
- The next page illustrates a partial listing of these built-in PowerShell variables.



```

Administrator: Windows PowerShell
PS C:\users\Administrator\MyScripts> set-location variable:
PS Variable:\> get-childitem

```

Name	Value
\$?	variable:
>	True
>	set-location
args	<>
ConfirmPreference	High
ConsoleFileName	
DebugPreference	SilentlyContinue
Error	<>
ErrorActionPreference	Continue
ErrorView	NormalView
ExecutionContext	System.Management.Automation.EngineIntrinsics
false	False
FormatEnumerationLimit	4
HOME	C:\Users\Administrator
Host	System.Management.Automation.Internal.Host.InternalHost
input	System.Collections.ArrayList+ArrayListEnumeratorSimple
MaximumAliasCount	4096
MaximumDriveCount	4096
MaximumErrorCount	256
MaximumFunctionCount	4096
MaximumHistoryCount	64
MaximumVariableCount	4096
MyInvocation	System.Management.Automation.InvocationInfo
NestedPromptLevel	0
null	
OutputEncoding	System.Text.ASCIIEncoding
PID	2924
PROFILE	C:\Users\Administrator\Documents\WindowsPowerShell\Microsoft.PowerShell_pr
ProgressPreference	Continue
PSBoundParameters	<>
PSCulture	en-US
PSEmailServer	
PSHOME	C:\Windows\System32\WindowsPowerShell\v1.0
PSSessionApplicationName	wsman
PSSessionConfigurationName	http://schemas.microsoft.com/powershell/Microsoft.PowerShell



# Variables In PowerShell

- These built-in PowerShell variables are divided into two types.
- The first type has a special meaning in PowerShell because they store configuration information for the current PowerShell session.
- Of these special variables, two are commonly used:
  - `$_` - contains the current pipeline object
  - `$Error` - contains error objects for the current PowerShell session
- The next page illustrates an example of both:



```
PS C:\users\Administrator\MyScripts> get-service | where-object {$_Name -match "W32Time" Name}
```

Unexpected token 'Name' in expression or statement.

At line:1 char:58

```
+ get-service | where-object {$_Name -match "W32Time" Name <<<< }  
+ CategoryInfo          : ParserError: (Name:String) [], ParentContainsErrorRecordException  
+ FullyQualifiedErrorId : UnexpectedToken
```

```
PS C:\users\Administrator\MyScripts> $Error
```

Unexpected token 'Name' in expression or statement.

```
PS C:\users\Administrator\MyScripts> get-service | where-object {$_Name -match "W32Time"}
```

Status	Name	DisplayName
Running	W32Time	Windows Time

```
PS C:\users\Administrator\MyScripts>
```



# Variables In PowerShell

- The second type of built-in variable consists of preference settings used to control the behavior of PowerShell.
- The table on the next page describes these variables.
  - NOTE: A Command Policy can be one of the following strings:
    - `SilentlyContinue`
    - `NotifyContinue`
    - `NotifyStop`
    - `Inquire`





## PowerShell Preference Setting Built-in Variables

Name	Allowed Values	Description
\$DebugPreference	Command Policy	Action to take when data is written via Write-Debug in a script or WriteDebug() in a cmdlet.
\$ErrorActionPreference	Command Policy	Action to take when data is written via Write-Error in a script or WriteError() in a cmdlet.
\$MaximumAliasCount	Integer	Maximum number of allowed aliases
\$MaximumDriveCount	Integer	Maximum number of allowed drives
\$MaximumErrorCount	Integer	Maximum number of errors held by \$Error
\$MaximumFunctionCount	Integer	Maximum number of functions that can be created
\$MaximumVariableCount	Integer	Maximum number of variables that can be created
\$MaximumHistoryCount	Integer	Maximum number of entries saved in the command history
\$ShouldProcessPreference	Command Policy	Action to take when ShouldProcess is used in a cmdlet
\$ProcessReturnPreference	Boolean	ShouldProcess returns this setting
\$ProgressPreference	Command Policy	Action to take when data is written via Write-Progress in a script or WriteProgress() in a cmdlet.
\$VerbosePreference	Command Policy	Action to take when data is written via Write-Verbose in a script or Write-Verbose() in a cmdlet.



# Understanding Aliases In PowerShell

- Unless you are using a script, PowerShell can require a fair amount of typing to run various command sequences.
- As with many scripting languages, PowerShell has an aliasing mechanism for cmdlets and executables, which can cut down on the amount of typing needed.
- Consider the two versions of the command shown on the next pages.
- NOTE: this example doesn't provide a major reduction in typing per se, but aliases can save you some time and prevent typos. To see the list of PowerShell aliases supported in the current session use the `get-alias` cmdlet as shown on page 29.



```
Administrator: Windows PowerShell
PS C:\users\Administrator\MyScripts> get-process | where-object {$_.Company -match ".*Microsoft*"} | form
Id, Path -AutoSize

Name          Id Path
----          -
csrss         496 C:\Windows\system32\csrss.exe
csrss         540 C:\Windows\system32\csrss.exe
dllhost       2136 C:\Windows\system32\dllhost.exe
dwm           3432 C:\Windows\system32\Dwm.exe
explorer      3504 C:\Windows\Explorer.EXE
lsass         640 C:\Windows\system32\lsass.exe
lsm           648 C:\Windows\system32\lsm.exe
msdtc         2248 C:\Windows\System32\msdtc.exe
powershell   3124 C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe
services     628 C:\Windows\system32\services.exe
SLsvc        1024 C:\Windows\system32\SLsvc.exe
smss          428 C:\Windows\system32\smss.exe
spoolsv      1524 C:\Windows\System32\spoolsv.exe
svchost       812 C:\Windows\system32\svchost.exe
svchost       872 C:\Windows\system32\svchost.exe
svchost       948 C:\Windows\System32\svchost.exe
svchost       992 C:\Windows\system32\svchost.exe
svchost      1008 C:\Windows\system32\svchost.exe
svchost      1080 C:\Windows\system32\svchost.exe
svchost      1136 C:\Windows\System32\svchost.exe
svchost      1160 C:\Windows\system32\svchost.exe
svchost      1320 C:\Windows\system32\svchost.exe
svchost      1732 C:\Windows\system32\svchost.exe
svchost      1744 C:\Windows\system32\svchost.exe
svchost      1892 C:\Windows\System32\svchost.exe
svchost      2940 C:\Windows\System32\svchost.exe
taskeng       1432 C:\Windows\system32\taskeng.exe
taskeng       3396 C:\Windows\system32\taskeng.exe
wininit       548 C:\Windows\system32\wininit.exe
winlogon      580 C:\Windows\system32\winlogon.exe
wuauc.lt     3840 C:\Windows\system32\wuauc.lt.exe

PS C:\users\Administrator\MyScripts>
```



```
Administrator: Windows PowerShell

PS C:\users\Administrator\MyScripts> gps ! ? <{$_Company -match ".*Microsoft*"} ! ft Name, Id, Path -Aut

Name          Id Path
----          -
csrss         496 C:\Windows\system32\csrss.exe
csrss         540 C:\Windows\system32\csrss.exe
dllhost       2136 C:\Windows\system32\dllhost.exe
dwm           3432 C:\Windows\system32\Dwm.exe
explorer      3504 C:\Windows\Explorer.EXE
lsass         640 C:\Windows\system32\lsass.exe
lsm           648 C:\Windows\system32\lsm.exe
msdtc         2248 C:\Windows\System32\msdtc.exe
powershell   3124 C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe
services      628 C:\Windows\system32\services.exe
SLsvc        1024 C:\Windows\system32\SLsvc.exe
smss          428 C:\Windows\system32\smss.exe
spoolsv      1524 C:\Windows\System32\spoolsv.exe
svchost       812 C:\Windows\system32\svchost.exe
svchost       872 C:\Windows\system32\svchost.exe
svchost       948 C:\Windows\System32\svchost.exe
svchost       992 C:\Windows\system32\svchost.exe
svchost      1008 C:\Windows\system32\svchost.exe
svchost      1080 C:\Windows\system32\svchost.exe
svchost      1136 C:\Windows\System32\svchost.exe
svchost      1160 C:\Windows\system32\svchost.exe
svchost      1320 C:\Windows\system32\svchost.exe
svchost      1732 C:\Windows\system32\svchost.exe
svchost      1744 C:\Windows\system32\svchost.exe
svchost      1892 C:\Windows\System32\svchost.exe
svchost      2940 C:\Windows\System32\svchost.exe
taskeng      1432 C:\Windows\system32\taskeng.exe
taskeng      3396 C:\Windows\system32\taskeng.exe
wininit       548 C:\Windows\system32\wininit.exe
winlogon      580 C:\Windows\system32\winlogon.exe
wuauc lt     3840 C:\Windows\system32\wuauc lt.exe

PS C:\users\Administrator\MyScripts> _
```

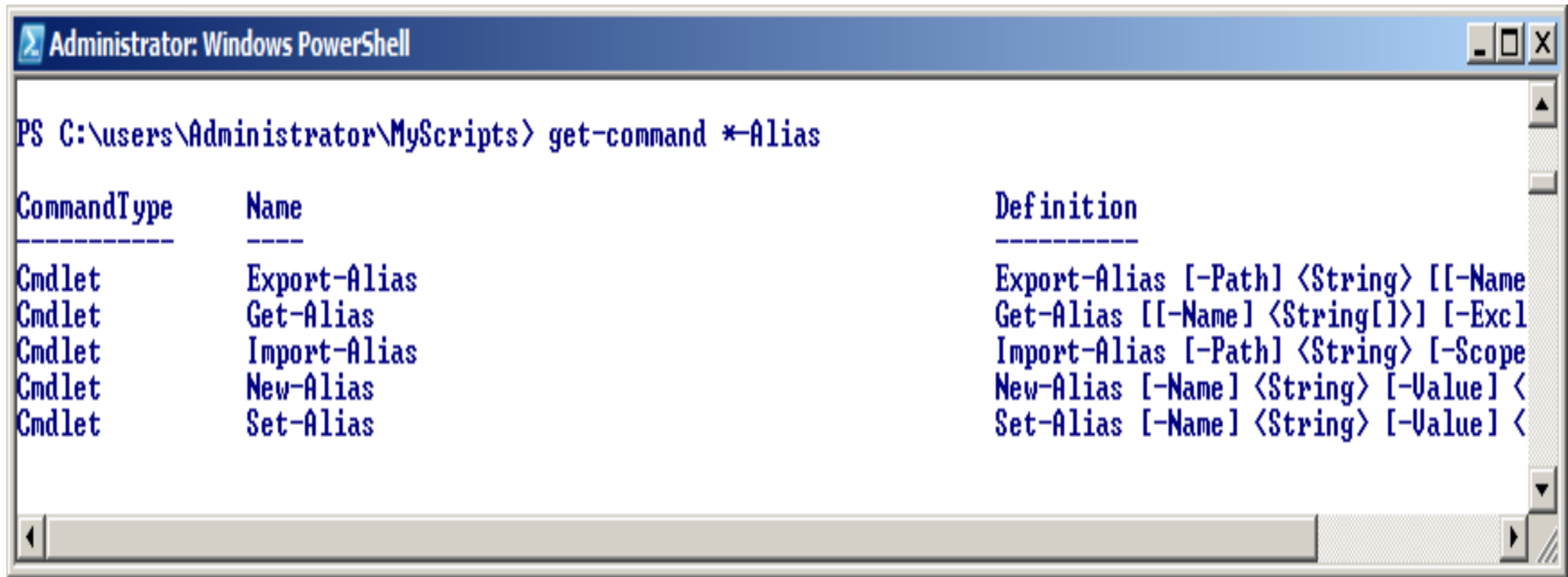


PS C:\users\Administrator\MyScripts > **get-alias**

CommandType	Name	Definition
Alias	%	ForEach-Object
Alias	?	Where-Object
Alias	ac	Add-Content
Alias	asnp	Add-PSSnapIn
Alias	cat	Get-Content
Alias	cd	Set-Location
Alias	chdir	Set-Location
Alias	clc	Clear-Content
Alias	clear	Clear-Host
Alias	clhy	Clear-History
Alias	cli	Clear-Item
Alias	clp	Clear-ItemProperty
Alias	cls	Clear-Host
Alias	clv	Clear-Variable
Alias	compare	Compare-Object
Alias	copy	Copy-Item
Alias	cp	Copy-Item
Alias	cpfi	Copy-Item
Alias	cpp	Copy-ItemProperty
Alias	cvpa	Convert-Path
Alias	dbp	Disable-PSBreakpoint
Alias	del	Remove-Item
Alias	diff	Compare-Object
Alias	dir	Get-ChildItem
Alias	ebp	Enable-PSBreakpoint
Alias	echo	Write-Output
Alias	epal	Export-Alias
Alias	epcsv	Export-Csv
Alias	epsn	Export-PSSession
Alias	erase	Remove-Item
Alias	etsn	Enter-PSSession
Alias	exsn	Exit-PSSession
Alias	fc	Format-Custom
Alias	fl	Format-List
Alias	foreach	ForEach-Object
Alias	ft	Format-Table
Alias	fw	Format-Wide

# Understanding Aliases In PowerShell

- There are several cmdlets that deal with aliases in PowerShell. The previous page illustrates the `get-alias` cmdlet.
- There are a few more that allow you to define your own aliases and to import and export aliases to other PowerShell sessions.



```
Administrator: Windows PowerShell
PS C:\users\Administrator\MyScripts> get-command *Alias
```

CommandType	Name	Definition
Cmdlet	Export-Alias	Export-Alias [-Path] <String> [[-Name
Cmdlet	Get-Alias	Get-Alias [[-Name] <String[]>] [-Excl
Cmdlet	Import-Alias	Import-Alias [-Path] <String> [-Scope
Cmdlet	New-Alias	New-Alias [-Name] <String> [-Value] <
Cmdlet	Set-Alias	Set-Alias [-Name] <String> [-Value] <



# Understanding Aliases In PowerShell

- The `Export-Alias` and `Import-Alias` cmdlets are used to export and import aliases from one PowerShell session to another.
- The `New-Alias` and `Set-Alias` cmdlets allow you to define new aliases for the current PowerShell session.
- Note that by default, all aliasing pertains only to a PowerShell session. Exiting PowerShell discards any existing aliases.
- For an alias to be persistent, it must be defined using the `set-alias` cmdlet and defined in the `profile.ps1` file. You can find the location of this file on your machine by typing `$profile` at the PowerShell prompt.



## CAUTION: Using Aliases In PowerShell

- Although command shortening may seem appealing, extensive use of aliasing is not recommended.
- One reason is that aliases are not very portable to scripts. For example, if you are using a lot of aliases in a script, you must include a `set-alias` sequence at the start of the script to ensure that those aliases are present, regardless of the machine, or session profile, when the script runs.
- However, a bigger concern is the probability that an alias can obscure or confuse the true meaning of commands or scripts. The aliases you define might make sense to you, but not everyone may share your logic in defining aliases. In general, functions are a better way to go than extensive aliasing.

