When it comes to running commands on Windows, PowerShell has become somewhat of an ace in the hole. For years enthusiasts were limited to the confines of the Windows command line but in 2006, PowerShell emerged as a powerful alternative.

Contents [hide]

What is PowerShell?

PowerShell is an interactive **Command-Line Interface** (**CLI**) and automation engine designed by Microsoft to help design system configurations and automate administrative tasks. This tool has its own command-line with a unique programming language similar to Perl. Initially, PowerShell was designed to manage objects on users' computers.

Today PowerShell offers users an extensive environment where they can execute and automate system management tasks. The user can access resources from Active Directory to Exchange Server through one program. At its core, PowerShell allows the user to access:

- **Command Prompt**
- **PowerShell Commands**
- **.NET Framework API**
- **Windows Management Instrumentation**
- **Windows Component Object Model**

As PowerShell has become an open-source application, Linux and Unix-based users can now access this versatile platform. PowerShell's is mainly used to help users automate administrative jobs. Rather than performing tedious and repetitive tasks, the user can simply create scripts and issue commands, and PowerShell will complete them automatically. The user can customize hundreds of commands, called **cmdlets**.

PowerShell Commands

Here are 25 basic PowerShell commands:

Command name	Alias	Description
Set-Location	cd, chdir, sl	Sets the current working location to a specified location.

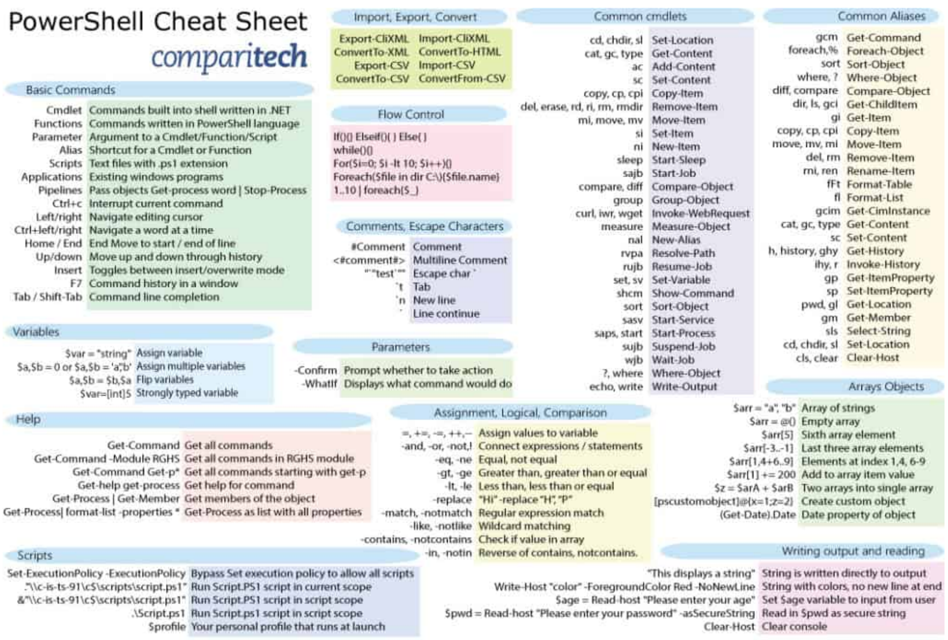
Command name	Alias	Description
Get-Content	cat, gc, type	Gets the content of the item at the specified location.
Add-Content	ac	Adds content to the specified items, such as adding words to a file.
Set-Content	sc	Writes or replaces the content in an item with new content.
Copy-Item	copy, cp, cpi	Copies an item from one location to another.
Remove-Item	del, erase, rd, ri, rm, rmdir	Deletes the specified items.
Move-Item	mi, move, mv	Moves an item from one location to another.
Set-Item	si	Changes the value of an item to the value specified in the command.
New-Item	ni	Creates a new item.
Start-Job	sajb	Starts a Windows PowerShell background job.
Compare-Object	compare, dif	Compares two sets of objects.

Command name	Alias	Description
Group-Object	group	Groups objects that contain the same value for specified properties.
Invoke-WebRequest	curl, iwr, wget	Gets content from a web page on the Internet.
Measure-Object	measure	Calculates the numeric properties of objects, and the characters, words, and lines in string objects, such as files …
Resolve-Path	rvpa	Resolves the wildcard characters in a path, and displays the path contents.
Resume-Job	rujb	Restarts a suspended job
Set-Variable	set, sv	Sets the value of a variable. Creates the variable if one with the requested name does not exist.
Show-Command	shcm	Creates Windows PowerShell commands in a graphical command window.
Sort-Object	sort	Sorts objects by property values.
Start-Service	sasv	Starts one or more stopped services.

Command name	Alias	Description
Start-Process	saps, start	Starts one or more processes on the local computer.
Suspend-Job	sujb	Temporarily stops workflow jobs.
Wait-Job	wjb	Suppresses the command prompt until one or all of the Windows PowerShell background jobs running in the session are …
Where-Object	?, where	Selects objects from a collection based on their property values.
Write-Output	echo, write	Sends the specified objects to the next command in the pipeline. If the command is the last command in the pipeline,…

Here is our PDF version of the PowerShell Cheat Sheet. Click on the image below to open the PDF in a separate browser tab that you can save and use as a quick reference.

PowerShell cheat sheet (PDF)


PowerShell Cheat Sheet

comparitech

Basic Commands

Cmdlet	Commands built into shell written in .NET
Functions	Commands written in PowerShell language
Parameter	Argument to a Cmdlet/Function/Script
Alias	Shortcut for a Cmdlet or Function
Scripts	Text files with .ps1 extension
Applications	Existing windows programs
Pipelines	Pass objects Get-process word | Stop-Process
Ctrl+c	Interrupt current command
Left/right	Navigate editing cursor
Ctrl+left/right	Navigate a word at a time
Home / End	End Move to start / end of line
Up/down	Move up and down through history
Insert	Toggles between insert/overwrite mode
F7	Command history in a window
Tab / Shift-Tab	Command line completion

Import, Export, Convert

Export-CliXML	Import-CliXML
ConvertTo-XML	ConvertTo-HTML
Export-CSV	Import-CSV
ConvertTo-CSV	ConvertFrom-CSV

Flow Control

```
If()() Elseif()( ) Else( )
while()()
For($i=0; $i -lt 10; $i++)()
Foreach($file in dir C:\)($file.name)
1..10 | foreach($_)
```

Comments, Escape Characters

#Comment	Comment
<#comment#>	Multiline Comment
"`"test`""	Escape char `
`t	Tab
`n	New line
`	Line continue

Common cmdlets

cd, chdir, sl	Set-Location
cat, gc, type	Get-Content
ac	Add-Content
sc	Set-Content
copy, cp, cpi	Copy-Item
del, erase, rd, ri, rm, rmdir	Remove-Item
mi, move, mv	Move-Item
si	Set-Item
ni	New-Item
sleep	Start-Sleep
sajb	Start-Job
compare, diff	Compare-Object
group	Group-Object
curl, iwr, wget	Invoke-WebRequest
measure	Measure-Object
nal	New-Alias
rvpa	Resolve-Path
rujb	Resume-Job
set, sv	Set-Variable
shcm	Show-Command
sort	Sort-Object
sasv	Start-Service
saps, start	Start-Process
sujb	Suspend-Job
wjb	Wait-Job
?, where	Where-Object
echo, write	Write-Output

Common Aliases

gcm	Get-Command
foreach,%	Foreach-Object
sort	Sort-Object
where, ?	Where-Object
diff, compare	Compare-Object
dir, ls, gci	Get-ChildItem
gi	Get-Item
copy, cp, cpi	Copy-Item
move, mv, mi	Move-Item
del, rm	Remove-Item
rni, ren	Rename-Item
fFt	Format-Table
fl	Format-List
gcim	Get-CimInstance
cat, gc, type	Get-Content
sc	Set-Content
h, history, ghy	Get-History
ihy, r	Invoke-History
gp	Get-ItemProperty
sp	Set-ItemProperty
pwd, gl	Get-Location
gm	Get-Member
sls	Select-String
cd, chdir, sl	Set-Location
cls, clear	Clear-Host

Variables

$var = "string"	Assign variable
$a,$b = 0 or $a,$b = 'a','b'	Assign multiple variables
$a,$b = $b,$a	Flip variables
$var=[int]5	Strongly typed variable

Parameters

-Confirm	Prompt whether to take action
-WhatIf	Displays what command would do

Help

Get-Command	Get all commands
Get-Command -Module RGHS	Get all commands in RGHS module
Get-Command Get-p*	Get all commands starting with get-p
Get-help get-process	Get help for command
Get-Process | Get-Member	Get members of the object
Get-Process| format-list -properties *	Get-Process as list with all properties

Assignment, Logical, Comparison

=, +=, -=, ++,--	Assign values to variable
-and, -or, -not,!	Connect expressions / statements
-eq, -ne	Equal, not equal
-gt, -ge	Greater than, greater than or equal
-lt, -le	Less than, less than or equal
-replace	"Hi" -replace "H", "P"
-match, -notmatch	Regular expression match
-like, -notlike	Wildcard matching
-contains, -notcontains	Check if value in array
-in, -notin	Reverse of contains, notcontains.

Arrays Objects

$arr = "a", "b"	Array of strings
$arr = @()	Empty array
$arr[5]	Sixth array element
$arr[-3..-1]	Last three array elements
$arr[1,4+6..9]	Elements at index 1,4, 6-9
$arr[1] += 200	Add to array item value
$z = $arA + $arB	Two arrays into single array
[pscustomobject]@{x=1;z=2}	Create custom object
(Get-Date).Date	Date property of object

Scripts

Set-ExecutionPolicy -ExecutionPolicy	Bypass Set execution policy to allow all scripts
."\\c-is-ts-91\c$\scripts\script.ps1"	Run Script.PS1 script in current scope
&"\\c-is-ts-91\c$\scripts\script.ps1"	Run Script.PS1 script in script scope
.\Script.ps1	Run Script.ps1 script in script scope
$profile	Your personal profile that runs at launch

Writing output and reading

"This displays a string"	String is written directly to output
Write-Host "color" -ForegroundColor Red -NoNewLine	String with colors, no new line at end
$age = Read-host "Please enter your age"	Set $age variable to input from user
$pwd = Read-host "Please enter your password" -asSecureString	Read in $pwd as secure string
Clear-Host	Clear console

How to Use PowerShell

PowerShell is ideal for corporate administrators who run complex management operations over large corporate networks. Rather than collating information about hundreds of different servers and services manually (which would take a long time), you can simply run a script on PowerShell to automatically feed information back to you.

Generally speaking, PowerShell is most beneficial to users who have prior experience with command lines. To use PowerShell, you can run a variety of cmdlets, scripts, executables, and .NET classes. For the purposes of this article, we're mainly going to focus on cmdlets and scripts to help you come to grips with the fundamentals.

Udemy has a number of top-rated courses on PowerShell that you might find useful.

PowerShell vs Command Prompt

For many users, PowerShell is a better alternative to Command Prompt. The reason is that it simply has more horsepower. One of the biggest differences is that PowerShell uses cmdlets rather than commands. Cmdlets place registry management and Windows Management Instrumentation within the administrative reach of users. In contrast, Command Prompt is confined to much more simple commands.

There is some crossover in syntax between the two platforms as PowerShell will accept some command prompt commands like `i pconfigtocd`. However, these are known as aliases rather than cmdlets. Another key difference is that PowerShell is centered on objects. Every piece of data output from a cmdlet is an object rather than text. This makes it easier for the user to navigate their way around complex data. The inclusion of the .NET framework also enables PowerShell scripts to use .NET interfaces. In short, PowerShell is Command Prompt on steroids.

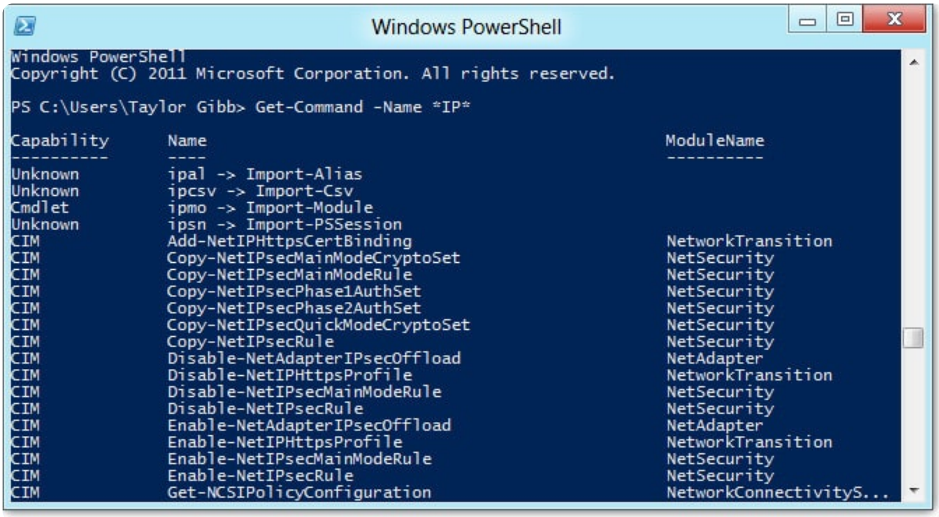
Loading Up PowerShell

Before we delve into the basics of using PowerShell, you first need to access the main interface. If you are a Windows 10 user then you will already have access to PowerShell 5. Windows 8-8.1 users have access to PowerShell 4, but if you're on Windows 7, you're going to need to install it within a .NET framework. Across all operating systems, PowerShell offers two distinct interfaces.

The more advanced is the Integrated Scripting Environment, which acts as a comprehensive GUI for experienced users. The basic alternative is the PowerShell console, which provides a command-line for the user to input their commands. Beginners are advised to stick with the latter until they learn the fundamentals of PowerShell.

In order to start PowerShell on Windows 10, you need to be an Administrator. Log in as an administrator, click **Start**, and scroll through your apps until you locate **Windows PowerShell**. Right-click and select **Run as Administrator**. On Windows 8.1, simply search for PowerShell in your **System** folder. Similarly, on Windows 7 the default directory for PowerShell is the **Accessories** folder after you've installed the program.

How to Run Cmdlets



```
Windows PowerShell
Copyright (C) 2011 Microsoft Corporation. All rights reserved.

PS C:\Users\Taylor Gibb> Get-Command -Name *IP*

Capability      Name                                     ModuleName
----------      ----                                     ----------
Unknown         ipal -> Import-Alias
Unknown         ipcsv -> Import-Csv
Cmdlet          ipmo -> Import-Module
Unknown         ipsn -> Import-PSSession
CIM             Add-NetIPHttpsCertBinding                NetworkTransition
CIM             Copy-NetIPsecMainModeCryptoSet           NetSecurity
CIM             Copy-NetIPsecMainModeRule                NetSecurity
CIM             Copy-NetIPsecPhase1AuthSet               NetSecurity
CIM             Copy-NetIPsecPhase2AuthSet               NetSecurity
CIM             Copy-NetIPsecQuickModeCryptoSet          NetSecurity
CIM             Copy-NetIPsecRule                        NetSecurity
CIM             Disable-NetAdapterIPsecOffload           NetAdapter
CIM             Disable-NetIPHttpsProfile                NetworkTransition
CIM             Disable-NetIPsecMainModeRule             NetSecurity
CIM             Disable-NetIPsecRule                     NetSecurity
CIM             Enable-NetAdapterIPsecOffload            NetAdapter
CIM             Enable-NetIPHttpsProfile                 NetworkTransition
CIM             Enable-NetIPsecMainModeRule              NetSecurity
CIM             Enable-NetIPsecRule                      NetSecurity
CIM             Get-NCSIPolicyConfiguration              NetworkConnectivityS...
```

In a nutshell, a cmdlet is a single-function command. You input cmdlets into the command line just as you would with a traditional command or utility. Cmdlets are the main way to interact with the CLI. In PowerShell, most cmdlets are written in C# and comprised of instructions designed to perform a function that returns a .NET object.

Over 200 cmdlets can be used in PowerShell. Windows PowerShell command prompt isn't case-sensitive, so these commands can be typed in either upper or lower case. The main cmdlets are listed below:

- **Get-Location** – Get the current directory
- **Set-Location** – Get the current directory
- **Move-item** – Move a file to a new location
- **Copy-item** – Copy a file to a new location

- **Rename** – item Rename an existing file
- **New-item** – Create a new file

For a full list of commands available to you, use the `Get-Command` cmdlet. In the command line you would enter the following:

```
PS C:\> Get-Command
```

It is important to note that Microsoft restricts users from using custom PowerShell cmdlets in its default settings. In order to use PowerShell cmdlets, you need to change the **ExecutionPolicy** from **Restricted** to **RemoteSigned**. **Remote Signed** will allow you to run your own scripts but will stop unsigned scripts from other users.

To change your Execution policy, type in the following PowerShell command:

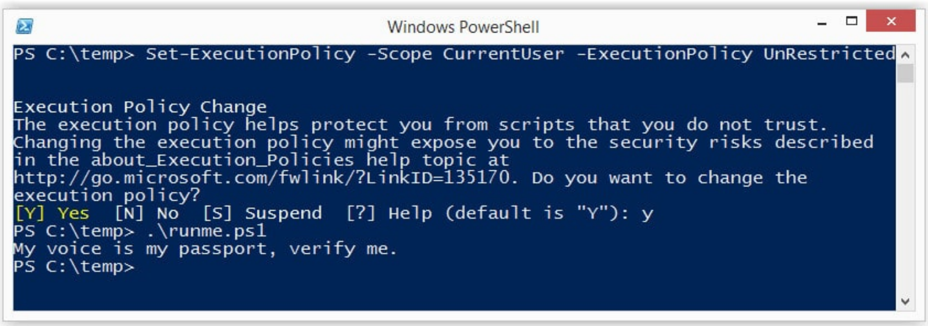
```
PS C:\>   Set-ExecutionPolicy
```

To change to **RemoteSigned**, type the following command:

```
PS C:\> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
```

Make sure you're on an Administrator account so that you have permission to set a new execution policy.

How to Run Scripts



```
PS C:\temp> Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy UnRestricted


Execution Policy Change
The execution policy helps protect you from scripts that you do not trust.
Changing the execution policy might expose you to the security risks described
in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the
execution policy?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y
PS C:\temp> .\runme.ps1
My voice is my passport, verify me.
PS C:\temp>
```

Script-based processes and commands are part of the foundation of PowerShell's versatility. In PowerShell, a script is essentially a text file with a ps1 extension in its filename. To create a new script you can simply open the Windows notepad, type your commands, and save with '.ps1' at the end of the name.

To run a script, enter its folder and filename into the PowerShell window :

```
PS c:\powershell\mynewscript.ps1
```

Once you've done this, your selected script will run.

Looking to create your own PowerShell scripts? Nearly 5k students have taken this Udemy course on Advanced Scripting with PowerShell.

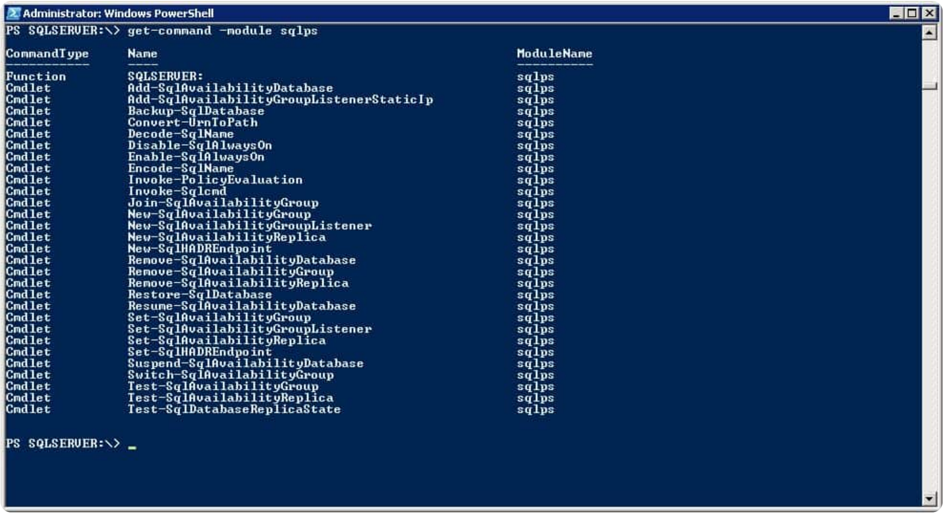
Overlap with Windows Commands

When you're new to PowerShell it can feel overwhelming to try and learn a whole new library of commands. However, what most new users don't realize is that the syntax used on Windows command-line overlaps with PowerShell. This is made easier by the fact that PowerShell isn't case sensitive.

Much like Command Prompt, on PowerShell the cd command still changes directories, and dir still provides a list of files within the selected folder. As such, it's important to remember you aren't necessarily starting from scratch. Taking this on board will help to decrease the learning curve you face when using PowerShell and decrease the number of new commands that you have to learn.

That being said, it is important to note that these aren't considered complete PowerShell commands so much as they are aliases (Powershell's name for Windows command prompt commands). So even though you can try some of Command Prompt's commands in PowerShell, you should learn as much as you can about the new ones. Nonetheless, Command Prompt experience can definitely help new users to come to grips with PowerShell and hit the ground running.

Backing Up an SQL Database



```
PS SQLSERVER:\> get-command -module sqlps

CommandType     Name                                          ModuleName
-----------     ----                                          ----------
Function        SQLSERVER:                                    sqlps
Cmdlet          Add-SqlAvailabilityDatabase                   sqlps
Cmdlet          Add-SqlAvailabilityGroupListenerStaticIp      sqlps
Cmdlet          Backup-SqlDatabase                            sqlps
Cmdlet          Convert-UrnToPath                             sqlps
Cmdlet          Decode-SqlName                                sqlps
Cmdlet          Disable-SqlAlwaysOn                           sqlps
Cmdlet          Enable-SqlAlwaysOn                            sqlps
Cmdlet          Encode-SqlName                                sqlps
Cmdlet          Invoke-PolicyEvaluation                       sqlps
Cmdlet          Invoke-Sqlcmd                                 sqlps
Cmdlet          Join-SqlAvailabilityGroup                     sqlps
Cmdlet          New-SqlAvailabilityGroup                      sqlps
Cmdlet          New-SqlAvailabilityGroupListener              sqlps
Cmdlet          New-SqlAvailabilityReplica                    sqlps
Cmdlet          New-SqlHADREndpoint                           sqlps
Cmdlet          Remove-SqlAvailabilityDatabase                sqlps
Cmdlet          Remove-SqlAvailabilityGroup                   sqlps
Cmdlet          Remove-SqlAvailabilityReplica                 sqlps
Cmdlet          Restore-SqlDatabase                           sqlps
Cmdlet          Resume-SqlAvailabilityDatabase                sqlps
Cmdlet          Set-SqlAvailabilityGroup                      sqlps
Cmdlet          Set-SqlAvailabilityGroupListener              sqlps
Cmdlet          Set-SqlAvailabilityReplica                    sqlps
Cmdlet          Set-SqlHADREndpoint                           sqlps
Cmdlet          Suspend-SqlAvailabilityDatabase               sqlps
Cmdlet          Switch-SqlAvailabilityGroup                   sqlps
Cmdlet          Test-SqlAvailabilityGroup                     sqlps
Cmdlet          Test-SqlAvailabilityReplica                   sqlps
Cmdlet          Test-SqlDatabaseReplicaState                  sqlps

PS SQLSERVER:\> _
```

Many people use PowerShell to back up SQL databases. The command-line interface can conduct full database backups, file backups, and transaction log backups. There are many ways to backup a database in PowerShell, but one of the simplest is to use the `Backup-SqlDatabase` command. For example:

```
PS C:\> Backup-SqlDatabase -ServerINstance “Computer\Instance” -Datab
ase “Databasecentral”
```

This will create a database backup of a database with the name ‘Databasecentral’ (or the name of your chosen database’.

To back up a transaction log, you would input:

```
PS C:\> Backup-SqlDatabase -ServerInstance “Computer\Instance”  -Data
base “Databasecentral” -BackupAction Log
```

This will create a transaction log of the selected database.

Essential PowerShell Commands

Using aliases will only get you so far on PowerShell, so it’s important to commit to learning everything you can about PowerShell’s native commands. We touched on some of these above, but we’re going to break down the main ones in much more detail below.

Get-Help

This command should be at the very top of any new user’s list when it comes to PowerShell. The Get-Help command can be used to literally get help with any other PowerShell command. For example, if you know the name of a command, but you don’t know what it does or how to use it, the Get-Help command provides the full command syntax.

For example, if you wanted to see how Get-Process works, you would type:

```
PS C:\> Get-Help -Name Get-Process
```

```
PS C:\> Set-ExecutionPolicy
```

As touched on earlier in this guide, Microsoft has a restricted execution policy that prevents scripting on PowerShell unless you change it. When setting the execution policy, you have four options to choose from:

- **Restricted** – The default execution policy that stops scripts from running.
- **All Signed** – Will run scripts if they are signed by a trusted publisher
- **Remote Signed** – Allows scripts to run which have been created locally
- **Unrestricted** – A policy with no restrictions on running scripts

```
PS C:\> Get-ExecutionPolicy
```

If you're using PowerShell, you may not always work on a server that you're familiar with. Running the command **Get-Execution Policy** will allow you to see which policy is active on the server before running a new script. If you then see the server in question operating under a restricted policy, you can then implement the **Set-ExecutionPolicy** command to change it.

Get-Service

One of the most important commands is `Get-Service`, which provides the user with a list of all services installed on the system, both running and stopped. This cmdlet can be directed by using specific service names or objects.

For example, if you were to type `PS C:\> Get-Service`, you would be shown a list of all services on your computer, their statuses, and display names.

To use this command to retrieve specific services, type: `PS C:\ Get-Service "WMI*"` to retrieve all services that begin with WMI.

If you wanted to restrict output to active services on your computer, input the following command:

```
PS C:\ Get-Service | Where-Object {$_.Status -eq “Running”}
```

ConvertTo-HTML

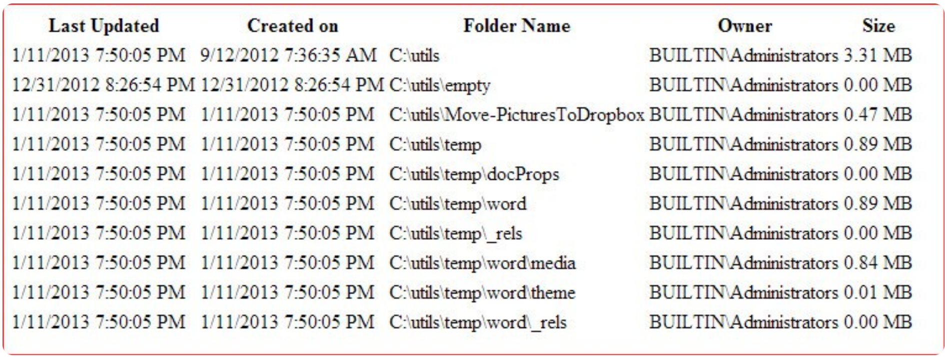
When using PowerShell, you might want to generate a report about the information you’ve seen. One of the best ways to do this is by using the **ConvertTo-HTML** command. This cmdlet allows you to build reports with tables and color, which can help to visualize complex data. Simply choose an object and add it to the command. For example, you could type:

```
Get-PSDrive | ConvertTo-Html
```

This returns a mass of information, so it’s a good idea to limit it to a file with the Out-File command. A better alternative command is:

```
Get-PSD Drive | ConvertTo-Html | Out-File -FilePath PSDrives.html
```

This will then generate an HTML file in table form. For example:



Last Updated	Created on	Folder Name	Owner	Size
1/11/2013 7:50:05 PM	9/12/2012 7:36:35 AM	C:\utils	BUILTIN\Administrators	3.31 MB
12/31/2012 8:26:54 PM	12/31/2012 8:26:54 PM	C:\utils\empty	BUILTIN\Administrators	0.00 MB
1/11/2013 7:50:05 PM	1/11/2013 7:50:05 PM	C:\utils\Move-PicturesToDropbox	BUILTIN\Administrators	0.47 MB
1/11/2013 7:50:05 PM	1/11/2013 7:50:05 PM	C:\utils\temp	BUILTIN\Administrators	0.89 MB
1/11/2013 7:50:05 PM	1/11/2013 7:50:05 PM	C:\utils\temp\docProps	BUILTIN\Administrators	0.00 MB
1/11/2013 7:50:05 PM	1/11/2013 7:50:05 PM	C:\utils\temp\word	BUILTIN\Administrators	0.89 MB
1/11/2013 7:50:05 PM	1/11/2013 7:50:05 PM	C:\utils\temp_rels	BUILTIN\Administrators	0.00 MB
1/11/2013 7:50:05 PM	1/11/2013 7:50:05 PM	C:\utils\temp\word\media	BUILTIN\Administrators	0.84 MB
1/11/2013 7:50:05 PM	1/11/2013 7:50:05 PM	C:\utils\temp\word\theme	BUILTIN\Administrators	0.01 MB
1/11/2013 7:50:05 PM	1/11/2013 7:50:05 PM	C:\utils\temp\word_rels	BUILTIN\Administrators	0.00 MB

You can then add your own colors and borders to refine its presentation.

Export-CSV (and Get-Service)

No less important for increasing visibility is the Export-CSV command. It allows you to export PowerShell data into a CSV file. Essentially, this command creates a CSV file compiling all of the objects you've selected in PowerShell. Every object has its own line or row within the CSV file. This command is primarily used to create spreadsheets and share data with external programs.

To use this command, you would type:

```
PS C:\> Get-Service | Export-CSV c:\service.csv
```

It's important to remember not to format objects before running the Export-CSV command. This is because formatting objects results in only the formatted properties being placed into the CSV file rather than the original objects themselves. In the event that you want to send specific properties of an object to a CSV file, you would use the **Select-Object** cmdlet.

To use the **Select-Object** cmdlet, type:

```
PS C:\> Get-Service | Select-Object Name, Status | Export-CSV c:\Serv
ice.csv
```

Get-Process

If you want to view all processes currently running on your system, the **Get-Process** command is very important. To get a list of all active processes on your computer, type:

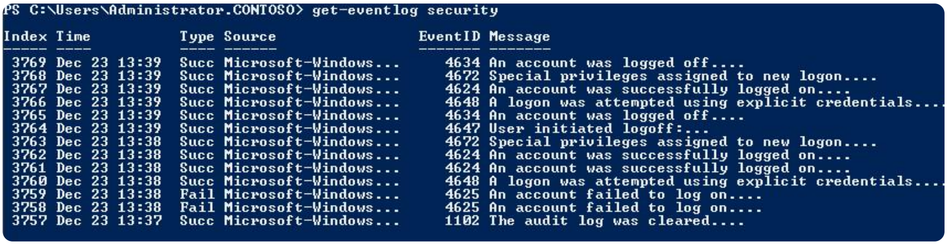
```
PS C:\ Get-Process
```

Notice that if you don't specify any parameters, you'll get a breakdown of every active process on your computer. To pick a specific process, narrow the results down by process name or process ID and combine that with the **Format-List** cmdlet, which displays all available properties. For example:

```
PS C:\ Get-Process windowrd, explorer | Format-List *
```

This provides you with comprehensive oversight of all active processes.

Get-EventLog



```
PS C:\Users\Administrator.CONTOSO> get-eventlog security

Index Time          Type Source                EventID Message
----- ----          ---- ------                ------- -------
 3769 Dec 23 13:39  Succ Microsoft-Windows...     4634 An account was logged off....
 3768 Dec 23 13:39  Succ Microsoft-Windows...     4672 Special privileges assigned to new logon....
 3767 Dec 23 13:39  Succ Microsoft-Windows...     4624 An account was successfully logged on....
 3766 Dec 23 13:39  Succ Microsoft-Windows...     4648 A logon was attempted using explicit credentials....
 3765 Dec 23 13:39  Succ Microsoft-Windows...     4634 An account was logged off....
 3764 Dec 23 13:39  Succ Microsoft-Windows...     4647 User initiated logoff:...
 3763 Dec 23 13:38  Succ Microsoft-Windows...     4672 Special privileges assigned to new logon....
 3762 Dec 23 13:38  Succ Microsoft-Windows...     4624 An account was successfully logged on....
 3761 Dec 23 13:38  Succ Microsoft-Windows...     4624 An account was successfully logged on....
 3760 Dec 23 13:38  Succ Microsoft-Windows...     4648 A logon was attempted using explicit credentials....
 3759 Dec 23 13:38  Fail Microsoft-Windows...     4625 An account failed to log on....
 3758 Dec 23 13:38  Fail Microsoft-Windows...     4625 An account failed to log on....
 3757 Dec 23 13:37  Succ Microsoft-Windows...     1102 The audit log was cleared....
```

If you ever want to access your computer’s event logs (or logs on remote computers) while using PowerShell, then you’re going to need the **Get-EventLog** command. This cmdlet only works on classic event logs, so you’ll need the **Get-WinEvent** command for logs later than Windows Vista.

To run the event log command, type:

```
PS C:\> Get-EventLog -List
```

This will show all event logs on your computer.

One of the most common reasons users look at event logs is to see errors. If you want to see error events in your log, simply type:

```
PS C:\> Get-EventLog -LogName System -EntryType Error
```

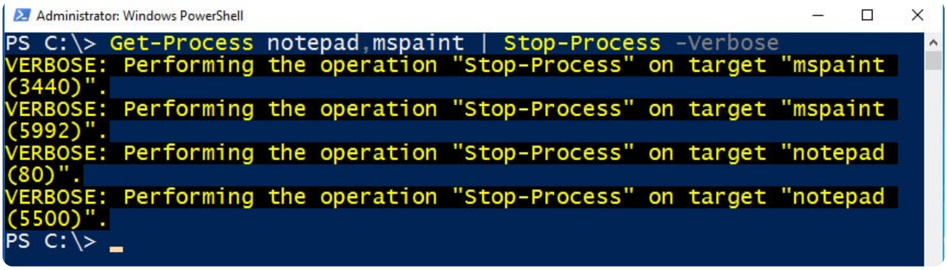
If you want to get event logs from multiple computers, specify which devices you want to view (listed below as “Server1” and “Server2”). For example:

```
PS C:\> Get-EventLog - LogName “Windows PowerShell” -ComputerName “lo
cal computer”, “Server1”, “Server2”.
```

Parameters you can use to search event logs include:

- **After** – User specifies a date and time and the cmdlet will locate events that occurred after
- **AsBaseObject** – Provides a System.Diagnostics.EventLogEntry for each event
- **AsString** – Returns the output as strings
- **Before** – User specifies a date and time and the cmdlet will locate events that occurred before
- **ComputerName** – Used to refer to a remote computer
- **EntryType** – Specifies the entry type of events (Error, Failure Audit, Success Audit, Information, Warning)
- **Index** – Specifies index values the cmdlet finds events from
- **List** – Provides a list of event logs
- **UserName** – Specifies usernames associated with a given event

Stop-Process



```
PS C:\> Get-Process notepad,mspaint | Stop-Process -Verbose
VERBOSE: Performing the operation "Stop-Process" on target "mspaint
(3440)".
VERBOSE: Performing the operation "Stop-Process" on target "mspaint
(5992)".
VERBOSE: Performing the operation "Stop-Process" on target "notepad
(80)".
VERBOSE: Performing the operation "Stop-Process" on target "notepad
(5500)".
PS C:\> _
```

When using PowerShell, it’s not uncommon to experience a process freezing up. Whenever this happens, you can use **Get-Process** to retrieve the name of the process experiencing difficulties and then stop it with the **Stop-Process** command. Generally, you terminate a process by its name. For example:

```
PS C:\> Stop-Process -Name “notepad”
```

In this example, the user has terminated Notepad by using the `Stop-Process` command.

PowerShell: A Powerful Command Line Interface

Although making the transition to PowerShell can seem quite complex, it’s command-line interface operates much the same as any other. It may have its own unique cmdlets, but a wealth of online resources can help you with any administrative task you can think of. To get the most out of PowerShell, you simply need to get used to the multitude of commands available to you.

As a new user, it is easy to become daunted by PowerShell’s 200-plus cmdlets. Make sure you start out with the command line interface before graduating to the full-blown GUI. Regardless of whether you’re new to PowerShell or command-line interfaces, more than enough information is available online to help you make the most of this powerful tool.