|  |
| --- |
| Bellevue University |
| Microsoft PowerShell |
| Research Document |
|  |

|  |
| --- |
| This document contains the research materials of Microsoft PowerShell and the related links. |

# A Beginner’s Introduction to Windows PowerShell

May 12, 2014 by [Jennifer Marsh](https://www.udemy.com/blog/author/jennifer_marsh/)

**[](http://www.udemy.com/powershellbeginner/?tc=blog.powershelltutorial&couponCode=half-off-for-blog&utm_source=blog&utm_medium=udemyads&utm_content=post99902&utm_campaign=content-marketing-blog&xref=blog)**

Windows PowerShell is an evolution of the command line – a combination of a DOS shell and scripting environment. It can help with repetitive tasks; processes acting on many files at once; automating and scheduling tasks; and configuring Windows components and services. PowerShell is not only for running DOS commands, it is far more flexible than that. It supports complex decision making, connecting to a wide variety of data sources, and even building graphical user interfaces.

[**PowerShell is now an essential skill for IT and server administrators**](http://www.udemy.com/powershellbeginner/?tc=blog.powershelltutorial&couponCode=half-off-for-blog&utm_source=blog&utm_medium=udemyads&utm_content=post99902&utm_campaign=content-marketing-blog&xref=blog), and is often used when deploying maintenance scripts across an entire organization. Getting those scripts to run safely on all machines can be a technical process, [**but tips and help are available for PowerShell users working with Enterprise Desktop machines**](http://www.udemy.com/managing-the-enterprise-windows-desktop-with-powershell/?tc=blog.powershelltutorial&utm_source=blog&utm_medium=udemyads&utm_content=post99902&utm_campaign=content-marketing-blog&xref=blog). Despite the focus on administrative tasks, home users and software developers can often save time by making use of the facilities available.

**Downloading and Running PowerShell**

Microsoft includes PowerShell with Windows 7 and 8. For users of Windows XP and Vista, you can download it from the [**downloads area**](http://technet.microsoft.com/en-us/scriptcenter/dd772288) of the Microsoft Scripting Center.

Once installed, there are various ways of running PowerShell:

* On Windows 8, from the Start Screen, begin typing powershell and either press **Enter** after typing the whole word, or click the search result named “Windows PowerShell”.
* On Windows 8, from the desktop, right click the Start menu icon, click **Run**, and then type powershell and press **Enter**.
* On all versions of Windows, press the **Windows logo key + R**, then type powershell and press **Enter**.

**Getting Help**

PowerShell introduces over a hundred new commands, and that is a lot to learn when you first start. However, it includes a comprehensive help system that can help you find commands to accomplish tasks, and it explains how to use those commands.

The command for getting help is get-help and it accepts a variety of different arguments. You can display help for a particular command by specifying the command’s name after get-help. For example, to display the help file entry for the get-help command, type:

get-help get-help

And then press Enter.

Get-help also accepts a keyword for searching the documentation, and returns a list of results containing that keyword. This can be a useful way of finding the names of commands related to specific topics. To search for commands and help files relating to Windows services, you can run:

get-help services

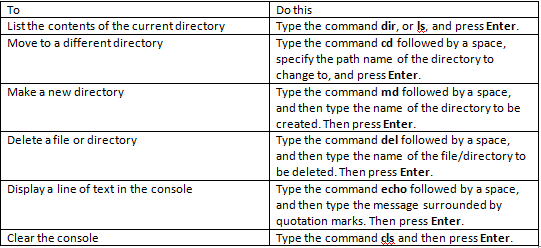
Help file entries are displayed directly in the console. However, by using the parameter “online” you can open a web browser and view the help on Microsoft’s TechNet website:

 get-help start-service -online

**Using the Console**

The console will be familiar in appearance to anyone who has used MS DOS, a DOS shell, or command line on any other operating system. In PowerShell, special commands (known as cmdlets) are typed in after the prompt, and are executed when you press the Enter key.

Navigating through the local file system is the same as when working in the earlier shells, and you can still use many of the older DOS commands. For example:

**[](https://www.udemy.com/blog/wp-content/uploads/2014/05/powershell-table.png)**

Execute the new cmdlets the same way – type their name, any additional arguments, and then press Enter. All cmdlets follow a standard naming pattern of “verb-noun”, and new PowerShell-specific cmdlets exist for all of the earlier DOS commands, although you are not required to use them.

Cmdlets appear to display their results in the console, however, most actually return an object that is processed by the console and then displayed. They do not write to the screen directly. This is a powerful concept and is where PowerShell has changed significantly from earlier shells. Instead of sending objects to the console, it is possible to send objects to other cmdlets with a technique known as piping.

For example, use the cmdlet get-itemproperty with the name of a file to display its basic attributes:

get-itemproperty c:\windows\regedit.exe

The result of this command is actually to return an object to the console, and this object contains many more pieces of information than the default view shows. By piping the object to format-list, other pieces of information are displayed:

get-itemproperty c:\windows\regedit.exe | format-list

The pipe symbol (|) is used to separate the two cmdlets, and indicates that the output of the first command is sent as the input to the second command.

When the result of a cmdlet consists of a collection of objects, piping will often process each one individually. For example:

dir | get-itemproperty | format-list

The result of dir is a collection of files and directories, which are piped one-by-one into the cmdlet get-itemproperty. The item’s properties are then piped to format-list to display the detailed file information.

The example above shows three commands connected by pipes, but more complicated tasks can use as many pipes as needed. The code below takes each piece of detailed information and pipes it to the out-file cmdlet to write the information to a file, instead of returning the text to the console for viewing on screen.

dir | get-itemproperty | format-list | out-file info.txt

**Scripting in PowerShell**

Scripts are text files that contain sequences of calls to cmdlets, and these files have the extension .ps1. However, unlike working in the console, each operation does not have to be typed in and executed immediately. This allows PowerShell to pre-process the file and extend the facilities to include those commonly associated with writing software, such as use of variables, conditional statements, and user-defined functions.

Before you can write and run scripts, you should be aware that Windows is not configured to allow the execution of unsigned scripts because they can be used to damage the system. An in-depth discussion of security models and code signing is beyond the scope of this article, but to configure your execution policy so that you can run scripts you have written, but not ones downloaded from the Internet, you can use the command set-executionpolicy:

set-executionpolicy remotesigned

As a demonstration, the following script uses the same cmdlets as the example earlier to output the detailed information for every file or directory in the current directory:

$list = dir

get-itemproperty $list | format-list | out-file out.txt

Where this script differs from the earlier console operation is that the result of dir (a collection of file and directory references) is not piped directly into get-itemproperty; it is first stored in a variable named list. Variables are declared and used in PowerShell by placing a dollar sign in front of their name.

That two-line script can be created in Notepad, then saved in the current directory with the filename “test.ps1″. You can then run the script by typing its path and filename in the console and pressing Enter.

.\test.ps1

You can rerun the script at any time, saving you from having to type all the statements again; and as the script file is portable, you can run the same script on different machines.

Scripting is highly flexible, as shown below:

$list = dir

foreach ($item in $list) {

    $fn = $item.name + "\_.txt"

    get-itemproperty $item | format-list | out-file $fn

}

As before, the directory listing is stored in the variable list. However, this time each item is processed individually by the script and not by the cmdlet.

The foreach loop runs through every item in the list and the instructions between curly braces are run for each one:

1. The current item from list is assigned to the variable item. Each item is an object describing the properties of a file or directory.
2. A new variable fn is created – this is a value calculated by taking the name (filename) property of the item, and adding an underscore and “.txt”.
3. The detailed properties for each item are then retrieved by passing the current item as an argument to get-itemproperty.
4. The returned object is piped into format-list to get the detailed view.
5. Finally, the properties of the item are written to a new text file using the filename that was calculated on the preceding line.

If you run that sample code in a directory with a lot of contents, a large number of new text files will be created. You can remove them by writing another PowerShell script:

$list = dir

foreach ($item in $list) {

    $fn = $item.name + "\_.txt"

    if (test-path $fn) {

        del $fn

    }

}

The example above processes the directory listing in a similar way to the previous script. This time, however, the conditional statement if and the cmdlet test-path are used to check if the file exists and if that is true, the script deletes the file.

**Further Reading**

As this is only a short introduction to PowerShell, it is not possible to show all of the available cmdlets or features of the scripting language. And there is not space to show how PowerShell can be used to communicate with Windows processes and services, which is an important part of its use for IT administrators. [**PowerShell for the Enterprise Desktop**](http://www.udemy.com/managing-the-enterprise-windows-desktop-with-powershell/?tc=blog.powershelltutorial&utm_source=blog&utm_medium=udemyads&utm_content=post99902&utm_campaign=content-marketing-blog&xref=blog) and [**PowerShell: A Getting Started Guide for IT Admins**](http://www.udemy.com/powershellbeginner/?tc=blog.powershelltutorial&couponCode=half-off-for-blog&utm_source=blog&utm_medium=udemyads&utm_content=post99902&utm_campaign=content-marketing-blog&xref=blog) offer a far more detailed introduction, including the concepts and commands that have not been described above.

PowerShell is based on Microsoft’s .NET framework, and so developers can write their own cmdlets using Microsoft Visual Studio and the languages supported by the common language runtime (CLR), such as C#. Cmdlets are intended to replace console applications, and [**complete beginners can learn C# programming**](http://www.udemy.com/programming-for-complete-beginners-in-csharp/?tc=blog.powershelltutorial&utm_source=blog&utm_medium=udemyads&utm_content=post99902&utm_campaign=content-marketing-blog&xref=blog) to begin writing their own, in much the same way as they would write a command line program.

**Extracted from**

[**https://www.udemy.com/blog/powershell-tutorial/**](https://www.udemy.com/blog/powershell-tutorial/)

**Links**

[**http://www.computerperformance.co.uk/powershell/index.htm**](http://www.computerperformance.co.uk/powershell/index.htm)

[**https://www.udemy.com/blog/powershell-tutorial/**](https://www.udemy.com/blog/powershell-tutorial/)

[**http://www.darkoperator.com/blog/2012/3/29/introduction-to-microsoft-powershellndash-basics-of-running.html**](http://www.darkoperator.com/blog/2012/3/29/introduction-to-microsoft-powershellndash-basics-of-running.html)

[**http://www.walkingrandomly.com/?p=5523**](http://www.walkingrandomly.com/?p=5523)