

# The Complete Powershell Training for Beginners

Start from Absolute Zero, and Learn to Use the Windows Powershell as it was meant to be used

By Abdelfattah Benammi

## Copyright

Copyright © 2020 by Abdelfattah Benammi All rights reserved

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review or scholarly journal.

First Printing: 2017

ISBN 978-1-67810-461-0

For permission requests, write to the publisher, at “Attention: Permissions Coordinator,” at the address below.

[Abdelfattah.ben@gmail.com](mailto:Abdelfattah.ben@gmail.com)

Dedication,

To my lovely students who enrolled in my Video training course and enjoyed it by leaving high number of positive reviews and who encouraged me to create this book version of the course

## 1. Before we Start

### 1.1. What to expect from this book?

Welcome to the course! This training is suited for you either if you work with both Windows or Linux, it will allow you to perform maintenance tasks on local and remote systems (including network devices). We will cover not only the basics of the PowerShell language (variables, strings, hash-tables, operators, providers and drivers) but also give you advices on the best way to think on solving problems with a consistent strategy for programming. When more advanced you will be able to understand and code with the basic functionality of regular expressions.

To introduce myself a little bit, I'm a system and network administrator graduated on 2008 at the Institute of Applied Technology and started using this amazing programming language back in 2009, since then I've been learning and improving on this knowledge.

### 1.2. Introduction to PowerShell

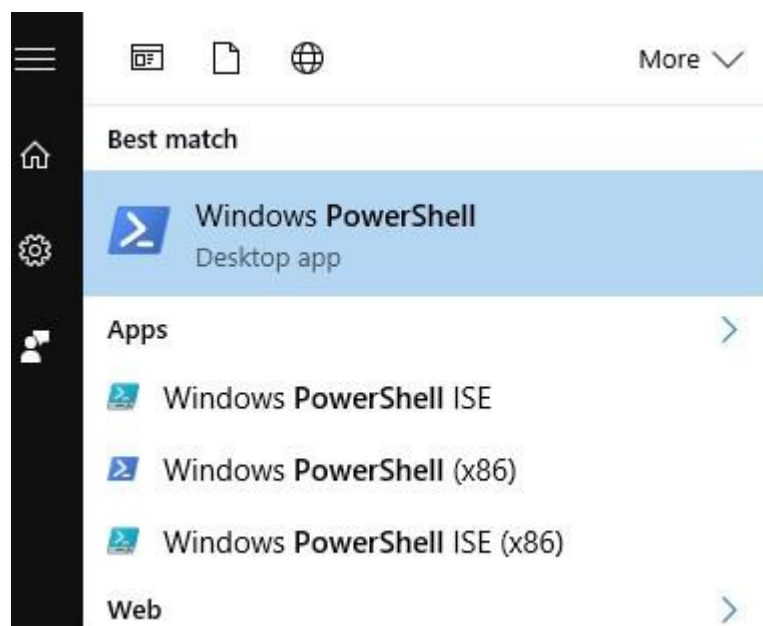
In order to access the Microsoft Windows Commandline we can go to "Start" -> "Execute" -> "CMD". We can see that it was the first implementation of a Commandline interface, since then they are improving so the commands can be better remembered. Almost all tasks that we do in the windows graphic interface can as well be done in the Commandline interface. Just to give an example we will create a shared folder on the graphical interface and show how it would be mapped as a drive letter on the Commandline interface. Let's go to "Windows Explorer" -> "C:" -> Create a New folder -> Right Click it -> On "Sharing Tab" click on "Share" button -> Choose "Everyone" -> "Add" -> "Share" Button. Now this folder is shared on the local network. It can be accessed locally on this URL: **\\127.0.0.1\New Folder**.

In order to create a folder mapping it to a drive letter on the command line we can use this command: **net use z: "\\127.0.0.1 New Folder"**. Bear in mind that the drive letter you chose must have been available. Now the shared folder is mapped as driver. We can see that we need to have that command and its syntax in mind before doing this operation, from this simple example we can start to see how the old CMD was not so intuitive (logical). The new Microsoft PowerShell try to overcome this difficulty creating commands that are both logical and easy to remember as we will see in the next sections.

## 2. Orientation and Requirements: Getting Ready for PowerShell

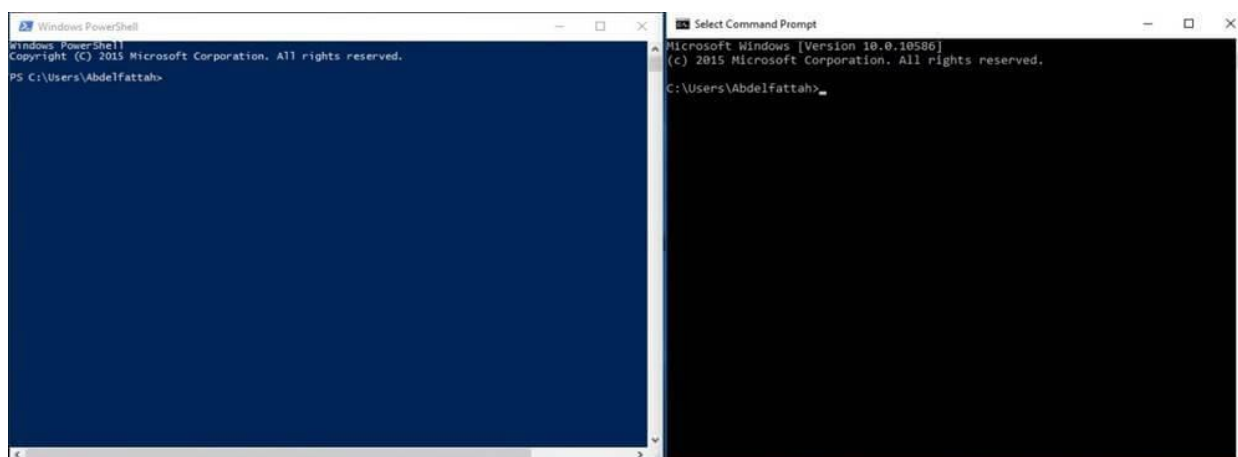
### 2.1. Discovering Windows PowerShell: Console Orientation

Even being an advanced PowerShell user, we advise you read this section because here we present some fundamental knowledge to build on. PowerShell comes on 2 different choices: Console and ISE. In our last example we showed how to access the old Windows Command Line Interface (CMD), in order to access the new PowerShell, we need just to hit “Start” -> Type: “PowerShell” and the different options will be presented:



**Figure 1: Example different PowerShell options available.**

On this example it appears the two different versions twice because the computer is for architecture x64, hence, allowing to choose from x86 versions for specific purposes. As we open the PowerShell we can see how different the native console interface is different from the CMD.



**Figure 2: Side-by-side difference from PowerShell (left) and native CMD (right).**

Note that on the PowerShell Command line the prompt has a “PS” before the name of the folder we’re at, this not happens on the CMD console. We can also start PowerShell inside the CMD Console, in order to this, simply type: PowerShell -> ENTER. We can see that the color of the console doesn’t change but it shows the “PS” before the current folder just as if you would see on opening it directly from the graphical interface. One other way of accessing the PowerShell is double-clicking on its executable file. To discover the executable file, on the PowerShell options listed on Figure 1, right click it -> “Properties” -> “Open and Find Location”. Knowing the path, you can reach it later directly navigating on the folders too. To open the PowerShell faster you can pin it to the taskbar or start menu, these are also options available when you “Right Click” on the PowerShell program.

One very important advice is **ALWAYS** open the PowerShell **as administrator**, it is necessary because most of commands (~80%) we will use on this course will only work if we are running as Administrator. When you right-click on the PowerShell this is the second option presented just below the “Open” option.

So far we have covered only the first PowerShell option, we also have available the PowerShell ISE that will be better presented latter.

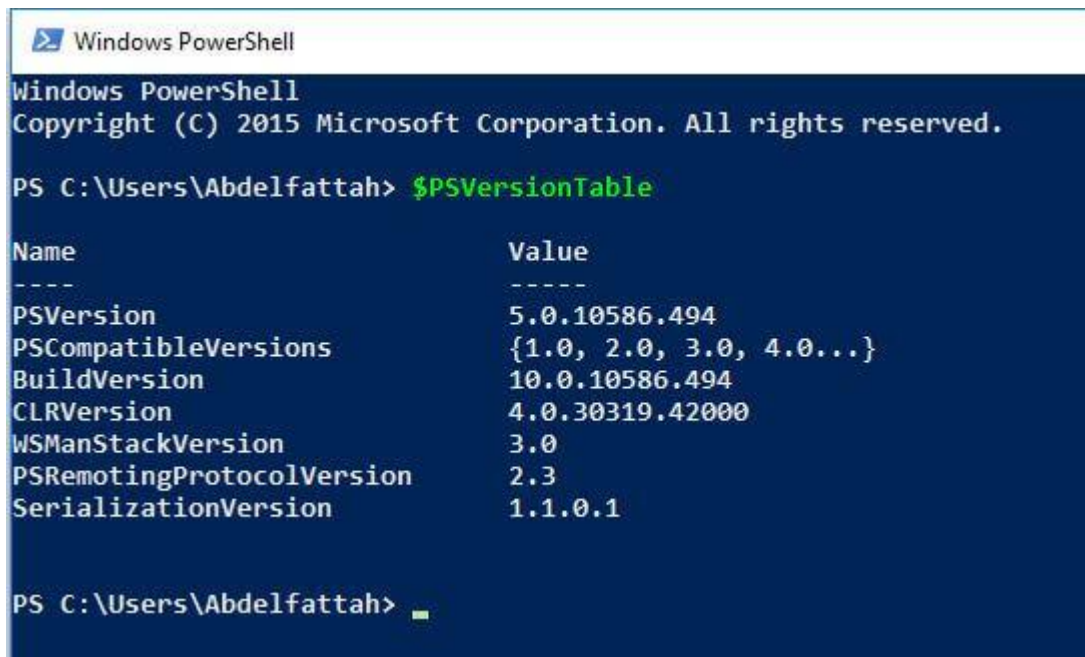
## 2.2. PowerShell version table

The Microsoft PowerShell may come in different versions with its own restrictions; these are summarized on the table below:

Version	Desktop OS	Server OS	.NET required
2	Windows XP/Later	Windows Server 2003/Later	2.0 or 3.5
3	Windows 7/Later	Windows Server 2008/Later	4.0 Full
4	Windows 7/Later	Windows Server 2008/Later	4.0 Full

**Table 1: Different versions of PowerShell with corresponding .NET framework requirements and compatibility for Desktop and Server Windows versions.**

The PowerShell version walks together with the .Net Framework; basically, when we install/upgrade the .NET Framework it is common that the same package is responsible for installing/upgrading the PowerShell as well. To find out what PowerShell version you have you can issue a very simple command (remember to open PowerShell as Administrator): **\$PSVersionTable**. On typing commands, you can always use the “TAB” button on your keyboard to let the console auto complete it for you, for instance, when issuing the aforementioned command, you could simply type: “\$PSVersion” and then “TAB” key so it automatically auto complete for you.



```
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Users\Abdelfattah> $PSVersionTable

Name                           Value
----                           -
PSVersion                      5.0.10586.494
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.10586.494
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1

PS C:\Users\Abdelfattah> 
```

**Figure 3: Example of the command showing the current PowerShell version**

On Figure 3 we can see that the installed PowerShell version is the 5 (PSVersion).

### 2.3. Customizing PowerShell for your Comfort

In the first time you open the PowerShell you will notice it comes with standard font and background. Depending on the font used it may be harder to distinguish different characters like comma (,) from punctuation mark (.). To change the font used on the console we can do the following: *Right Click on the PowerShell title bar -> Properties -> Tab “Font”*. The default font is “Raster Fonts”, a suggested font is “Consolas” either with boldface or not. You can also choose the size as well. I use the size 14.

On the “Layout” tab we can change the “Screen Buffer Size” that changes the size of the console inside the window. If you choose a buffer size bigger than the window it will appear a scrollbar. We recommend you always choose a Width buffer to be the same as the Window Width (otherwise, it would always appear a horizontal bar). Even if horizontal bars are not much for a concern for you, bear in mind that on some commands that show object lists it may present you with problems since the PowerShell always fills the screen from the right to the left (in these specific cases). It is important to note that this behavior was fixed on the version 5.

On the height, it is not a problem to have a Buffer height bigger than the screen height since we are usually used to “scroll down”, however, if you choose a big “Windows Height” it may become a problem if that height is bigger than your screen size height (it would always appear to be “cropped” on the bottom). It is best to choose a window height that is lower than your screen height.

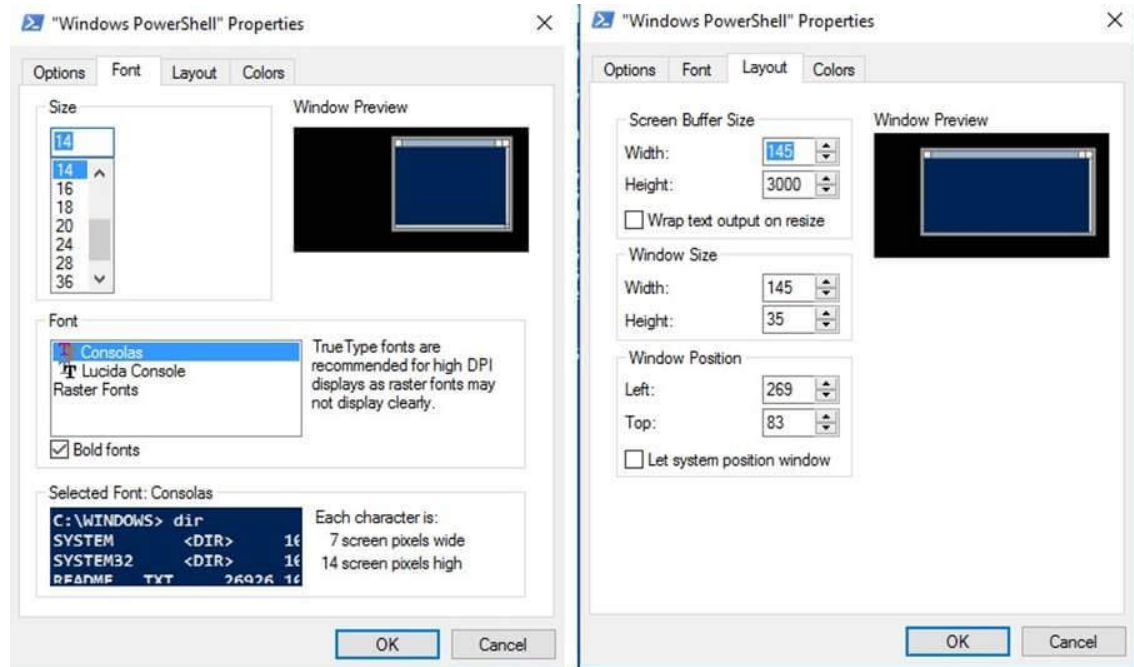


Figure 4: Configurations on Font and Layout PowerShell console

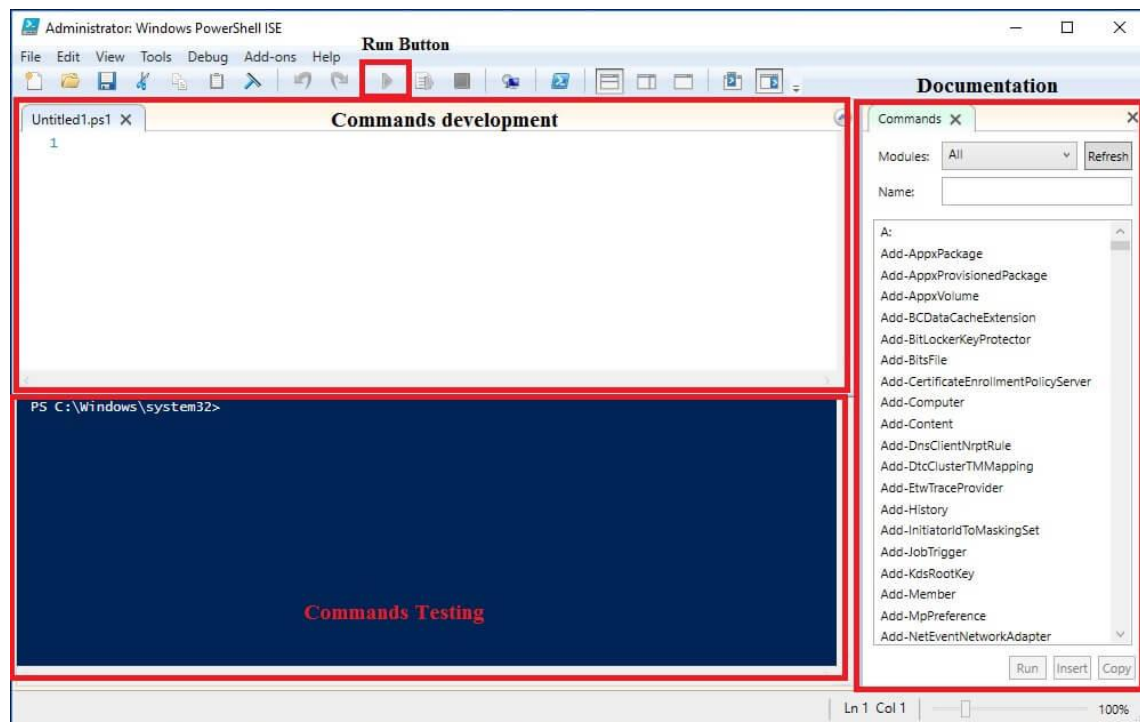
#### 2.4. Discovering PowerShell Integrated Script Environment (ISE)

Now we are going to cover the other PowerShell option called: PowerShell ISE. It is an integrated environment that allows us to quickly develop, test and search for documentation on the same window. Remember to always start your PowerShell **running as Administrator**. If you have previous experience on programming on the native batch file scripting, you know that the customary way of creating and testing a script only with default windows tools would be as follows:

- ☐ Create a file on some folder with “Right Click” -> New -> “Text Document”
- ☐ Rename that document to change its extension: “Right Click” -> Rename
- ☐ Change extension to .cmd
- ☐ Edit it with notepad: “Right Click” -> Edit
- ☐ Insert some code, for instance: calc.exe
- ☐ Save the document
- ☐ Close it
- ☐ Double-click the .cmd file



On this example we just created a simple batch script that when executed opens the Calculator. To avoid this actions and provide the user with more productivity, Microsoft created the ISE environment so all these steps can be avoided. Let's take a look at the environment itself.



**Figure 5: Overview of PowerShell ISE and 4 main parts.**

At the PowerShell ISE we have 4 main parts, the first one is the “Commands Development” in which we can enter our code freely as if we were creating a script in Notepad on the previous example. For each line of code, you enter here, it will be automatically entered on the console below (“Commands Testing” part) with its corresponding output.

The “Documentation” part is very useful because commands are grouped in modules; therefore, you can also select a module and scroll down to discover all available commands. If you have additional modules installed they will be automatically listed here. If you know some part of the command, you can search it by name. When using a command out of this section, it generates graphical inputs for all specific command options for you, when you fill all the options accordingly, just click “Insert” and it is inserted on the “Commands Testing” part. We will cover more details on modules later on this course.

After you write some code you can promptly test it just hitting the “Run Button” in which it runs everything you just typed. In our previous example, we could replace all these steps to just these ones using the ISE:

- ☐ Type: “calc.exe” on the “Commands Development” part
- ☐ Click the “Run Button”

So we can see how the PowerShell ISE offers us with productivity gain combining very important functions in just one screen.

### 3. Finding and Discovering Commands

#### 3.1. Windows PowerShell Commands Formulation

All commands that we use on PowerShell are called “Cmdlet”, it was created in a way that commands are consisted on two words involving a **verb** and a **noun**. Because of this design choice we can see how the CmdLets were built for being both easy to use and to remember. For instance, the command “**Get-Date**” obtains the complete date at the time it was issued, notice that it starts with “Get” which is the verb and ends with “Date” which is the noun.

#### 3.2. Pssnapin and Modules Commands

In PowerShell there is 2 different ways for defining Cmdlets and functions. These are provided by: “**PsSnapin**” and “**Module**”. The PsSnapins are a collection of commands that comes from a provider built on .NET, compiled and linked to DLL files. The PsSnapins form the base of PowerShell and although they are not used so extensively as the modules, they provide the base in which modules build on. The next kind of commands is the “**Module**”, they combine a collection of commands that are created to be used directly to the user. When we saw the PowerShell ISE, the menu at the right shows exactly all the Modules available with their respective Cmdlets provided.

#### 3.3. Discovering the different Pssnapin Commands

It is important paying attention to the PowerShell profile; it allows us to save information about your session so you can use it latter. When we open a new console window, not that it starts blank that happens because (by default) the PowerShell profile stays on RAM memory and is lost as soon as the console window closes. In order to take a look at what is saved on the profile we can use the following command: “**Get-PsSnapin –Registered**”. Profiles in PowerShell work a lot like the registry in Windows, for example, when we install a game, play it and reach a certain level (let’s say 5), then we decide to uninstall the game but keep the profile, when we decide to install it again, it will import the profile and remember we stopped at the level 5 so you can keep playing from there.

When we use the command “*Get-PsSnapin*” it will show only the “Microsoft.PowerShell.Core” PsSnapin because this is the one loaded by default regardless you ask to load it explicitly or not. It may seem overwhelming, but you don’t need to focus on this point in memorizing this command buy try to understand how the software was built to work. We can use another command called: “*Get-PsProvider / -format list -name PSSnapIn*” which will show all available providers with their respective snapins.

```
PS C:\Users\Abdelfattah> Get-PSProvider | Format-List Name,PSSnapIn

Name      : Registry
PSSnapIn  : Microsoft.PowerShell.Core

Name      : Alias
PSSnapIn  : Microsoft.PowerShell.Core

Name      : Environment
PSSnapIn  : Microsoft.PowerShell.Core

Name      : FileSystem
PSSnapIn  : Microsoft.PowerShell.Core

Name      : Function
PSSnapIn  : Microsoft.PowerShell.Core

Name      : Variable
PSSnapIn  : Microsoft.PowerShell.Core

Name      : Certificate
PSSnapIn  :

Name      : WSMAN
PSSnapIn  :
```

**Figure 6: Example of listing available providers and snapins.**

The providers group the functions provided by a PSSnapin, in this case, it shows only the providers of the Core PsSnapin.

### 3.4. Discovering the different Modules Commands

To list the modules that are loaded on the session we can use the command “*Get-Module*”.

```
PS C:\Windows\system32> Get-Module

ModuleType Version      Name                               ExportedCommands
-----
Manifest 3.1.0.0 Microsoft.PowerShell.Management {Add-Computer, Add-Content, Checkpoint-
Manifest 3.1.0.0 Microsoft.PowerShell.Utility   {Add-Member, Add-Type, Clear-Variable,
Script    1.1         PSReadline                     {Get-PSReadlineKeyHandler, Get-PSReadli
```

**Figure 7: Using the command “Get-Module” to list the session loaded modules.**

Another way to see the modules is at the ISE interface at the right side when we showed that it allows us to search even for modules that are not loaded. Another option is to navigate inside the PowerShell folder structure and see the different folders that provide different modules, for instance, we can go to: “C:\Windows\system32\WindowsPowerShell\v1.0\Modules” and each folder inside this will represent a different module.

Finally, there is a specific option for the “Get-Module” command explained above that instructs the PowerShell to list all the available modules of the system: “**Get-Module -ListAvailable**”.