# Carpentry PowerShell Lesson

## Introduction

### What is PowerShell and why would I want to use it

PowerShell is the Microsoft equivalent of the Bash shell in Linux. Except that it has been designed from the outset to control the Windows environment and the applications and sub-systems running within it.

It is entirely object-oriented and can be used to mange all aspects of both local and remote windows environments.

Today we are only going to look at a very small part of it, namely the part which allows us to manipulate the file system and run some programs

Because it is designed to work with all parts of Windows, the names of some of the commands (or cmdlets as PowerShell refers to them) may seem a bit vague.

For example it is not obvious that you would want to use;

Get-Childitem

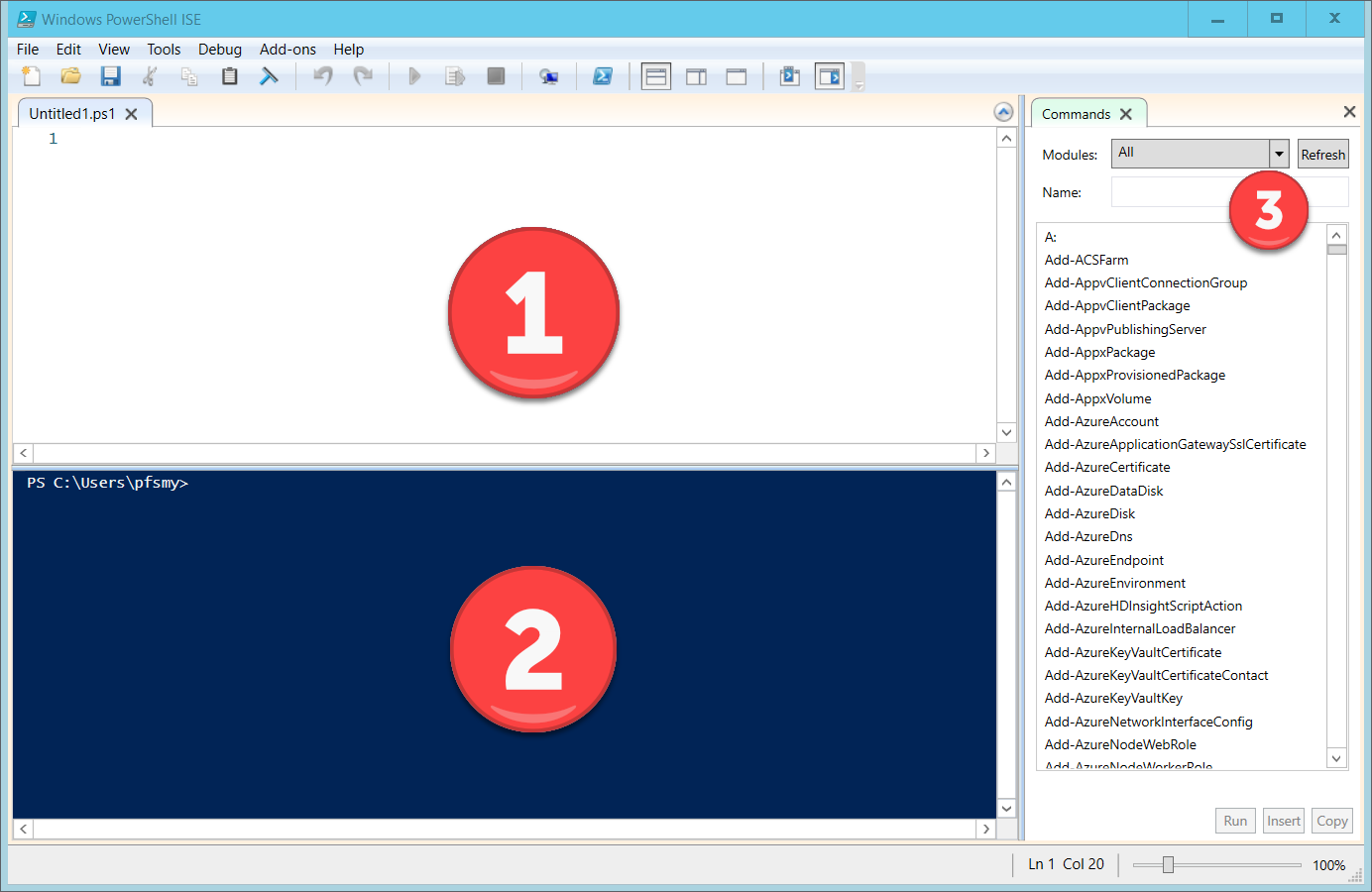
to list the contents of a folder or directory, but bear in mind that the same cmdlet is also used to list portions of the Windows registry.

### The PowerShell ISE

The PowerShell ISE (Interactive Scripting Environment) is install as part of the Windows 10 installation. You can find it using search or under Windows PowerShell in the alphabetical list of applications.

By default there will be 4 options. We will use the ‘PowerShell ISE’.

The initial display is a window with three panes. The panes can be placed or removed as you wish, but the default configuration will look like this.



Pane 1 is tabbed and is used to hold the script files that you create or open. By default it opens with a ‘Untitled1.ps1’ script file. This can be saved and renamed as you like. All PowerShell scripts have the suffix of ‘.ps1’.

Pane2 is a PowerShell console. You can execute PowerShell commands directly from here or more typically code that you select and run from the script pane is automatically copied to the console pane from where it is actually run. Any output or error messages from the code appear in the console pane.

Pane 3 provides a list of all of the available commands (PowerShell commands are usually referred to as cmdlets). You can use the name textbox to filter the commands. When you select one, you will have the ability to provide parameter values and either run the command in the console , insert the command into the console pane without running it or of copying the command, including the parameters you have set allowing you to manually (ctrl+v) pasting it into your script in the script pane.

When you have selected a command there is also a help button you can use to get the complete standard help for the command. This is displayed in a new pop-up window.

### PowerShell security

Although it doesn’t affect us when we are using the ISE, by default the PowerShell security system will not allow us to run .ps1 files directly from the commandline or PowerShell console. PowerShell is designed to provide a system of complete automated control over the Windows environment. Consequently, there are many very powerful PowerShell commands that can cause damage to the system. So, to prevent scripts containing such commands from being accidently run the default setting for the PowerShell security environment is ‘Restricted’ i.e. cannot be run.

You can check the current setting by using the

Get-ExecutionPolicy

cmdlet. The possible response are:

* **Restricted** — Restricted is the default execution policy and locks PowerShell down so that commands can be entered only interactively. PowerShell scripts are not allowed to run.
* **All Signed** — If the execution policy is set to All Signed then scripts will be allowed to run, but only if they are signed by a trusted publisher.
* **Remote Signed** — If the execution policy is set to Remote Signed, any PowerShell scripts that have been locally created will be allowed to run. Scripts created remotely are allowed to run only if they are signed by a trusted publisher.
* **Unrestricted** — As the name implies, Unrestricted removes all restrictions from the execution policy.

If you need to and have appropriate Admin rights on your system, you can change the setting using the

Set-ExecutionPolicy

cmdlet.

### Cmdlet naming conventions

All of the Powershell cmdlets are named using the Verb-Noun convention. I.e. What do you want to do and What do you want to do it to.

A complete list of the commands available can be seen in the Commands window. If you install Powershell add-ons, for example to manage Azure or MS SQL Server, then these too will appear in the commands window.

### Alias’

Unfortunately several of the commonly used cmdlets have rather un-obvious names. For example

Get-Childitem

could mean just about anything. In some ways it does mean many different things. In our case we are going to use it to list files in a folder.

### Objects

Everything in PowerShell is an object. cmdlets always return an object. All objects have variables within them, which typically you ‘get’ and ‘set’ and methods which are the functions which can be called on the object.

## The basics

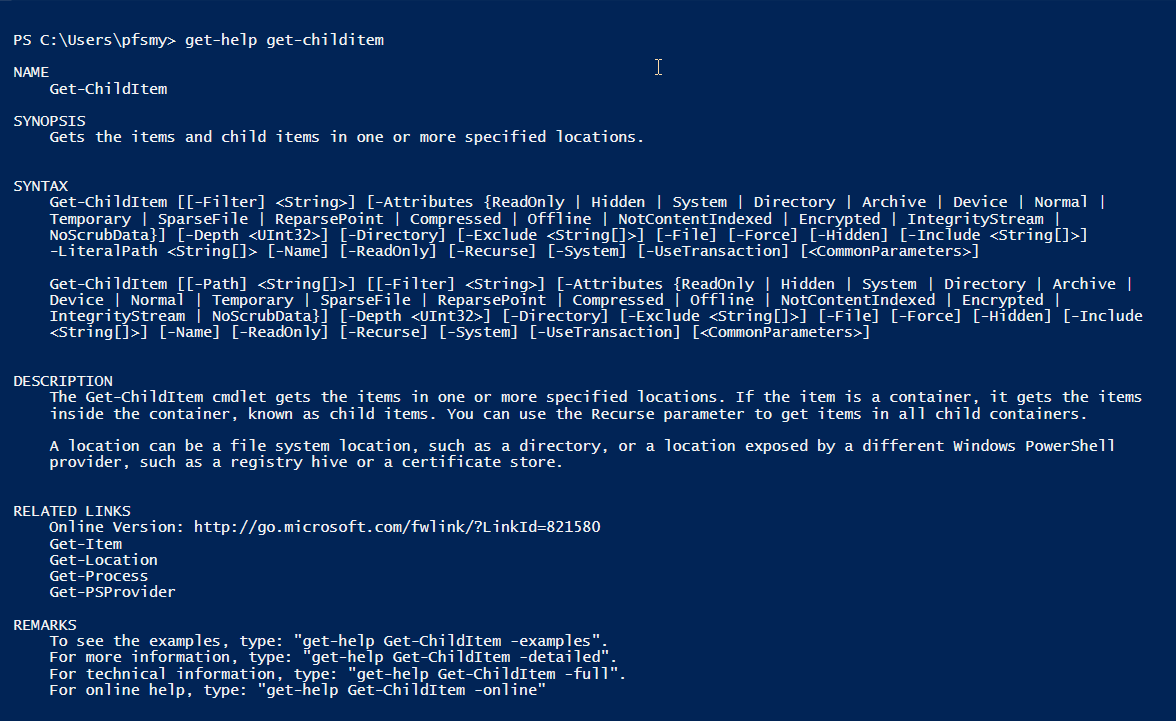
### Getting Help

see comments.ps1

To get help on any of the cmdlets, you use the ‘Get-Help’ cmdlet and specify the name of the command you want help on.

Get-Help Get-Childitem

will produce the following output:



The contents of the output will vary from cmdlet to cmdlet, but the format will always be the same.

### Creating variables

see variables.ps1

### Control structures

see control\_structures.ps1

### File & Folder commands

see basic\_file\_commands.ps1

### Reading and Writing files

see Reading\_and\_Writing.ps1

### Pipes in PowerShell

We will use pipes throughout the code in our PowerShell examples. We indicate a pipe with the ‘|’ character. Pipes are used to pass the output from one cmdlet to the input of the next cmdlet. You can chain as many pipes as you need.

### The Task

Details of the Task we are looking to complete in PowerShell is contained in the Scenario.docx file.