About PowerShell

Windows PowerShell is a command line utility and new scripting language provided by Microsoft. Why should I study and care about PowerShell? Because it is a new-style utility? Of course, every new utility is claimed to be 'different' from the old ones, but PowerShell has some components that really distinguish it from other utilities.

Windows PowerShell is a command line utility and new scripting language provided by Microsoft.Why should I study and care about PowerShell?Because it is a new-style utility?Of course, every new utility is claimed to be 'different' from the old ones, but PowerShell has some components that really distinguish it from other utilities.In this article we will look at some of the scripting components of PowerShell and create an example PowerShell script from seemingly messy things.

Brief history of the Windows command line utility

After Windows NT was released, CMD.EXE became the command line utility for Windows. Although CMD.EXE inherits some components of its predecessor DOS (COMMAN.COM), it still relies on a rather 'primitive' scripting language: using Windows Command files (.CMD and .BAT).). The addition of Windows Scripting Host and VBScript and JScript languages ??has significantly enhanced the scripting capabilities for the utility.

These technologies are a well-balanced combination of advanced command line utilities and scripting environments. Actually the problem can be manipulated with how many CMD.EXE, .CMD and Windows Scripting Host files are not a real concern. What makes people complain and is most concerned about completing some seemingly simple tasks.

Using the 'framework' of command-line tools and scripts, any moderately synthesized script also requires a combination of both batch commands, Windows Scripting Host and independent executables. Each script uses different conventions for execution and request, parsing, and returning data.

Weak variables supported in CMD.EXE, inconsistent interfaces and limited access to Windows settings, combined with another weakness that makes the command line script more difficult to deploy and use . You will probably wonder what 'another weakness' here? Please say that it is plain text (text). In these technologies, everything is in text format. The output of a command or script is text and must be parsed as well as reformatted to act as input for the next instruction. This is the basic starting point that PowerShell removed from all traditional utilities.

PowerShell script = Batch files on Steroids

PowerShell itself is written in the .NET language and relies heavily on the .NET Framework. So PowerShell is designed as an object-oriented utility and scripting language. All in PowerShell are viewed as an object with full functionality of the .NET Framework. A command provides a set of objects that can be used by using the properties and methods of that object type. When you want to put the output of a command into the pipeline for

another command, PowerShell will actually pass the object, not just the first line's text output. This gives the next command the ability to access all the properties and methods of the object in the pipeline.

Considering everything as an object and the ability to accept objects between commands is a major theoretical change for command line utilities. That said, PowerShell still works like a traditional shell utility. Commands, scripts, and executables can be typed and run from the command line and the results are displayed in text format. Windows .CMD and .BAT files, VBScripts, JScripts and executables work inside CMD.EXE, all still running in PowerShell. However, since they are not object-oriented, they do not have full access to the objects created and used in PowerShell. These legacy scripts and executables will still treat everything as text, but you can combine PowerShell with some other technology. This is very important if you want to start using PowerShell with a collection of existing scripts that can't be converted all at once.

A PowerShell Script

Reading and understanding the amazingness of technology is one thing, and considering it and using it is another thing! In the rest of this article, we will develop a PowerShell script to demonstrate its capabilities and uses.

DIR is one of the most popular commands in CMD.EXE. This command shows all files and subdirectories contained in a parent directory (Figure 1). Along with the name of each object, the information given also has the latest date and time updates and the size of each file. DIR also displays the aggregate size of all files in the directory, as well as the total number of files and subdirectories.

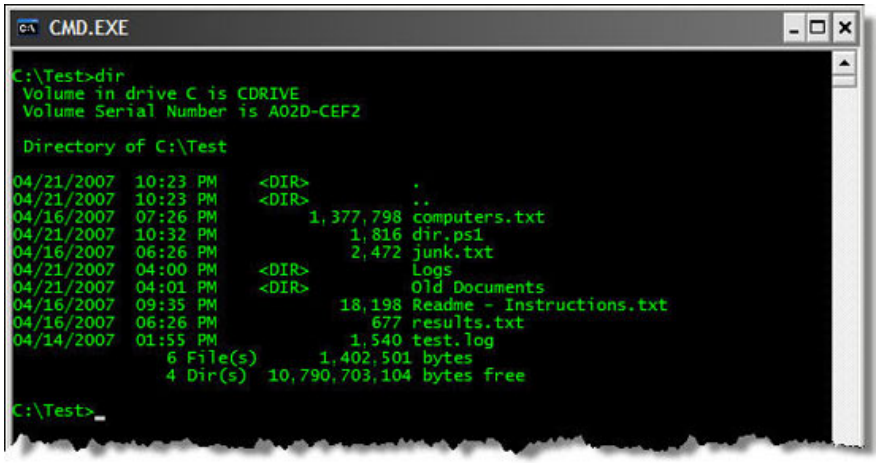

Figure 1

Running DIR in PowerShell also offers a directory listing like Figure 2, but slightly different. PowerShell does not have a DIR command, but rather a Get-ChildItem, which performs the same function. In PowerShell, DIR is an alias for Get-ChildItem. I have no intention to delve into the aliases in this article. You can imagine DIR in PowerShell as an abbreviation for Get-ChildItem.

DIR in PowerShell provides much the same information as mentioned above: a file and folder list, last updated date and time, and file size. However, it lacks the summary information that DIR in CMD.EXE provides: the total size of all files in the directory, the total number of files and the total number of subdirectories.

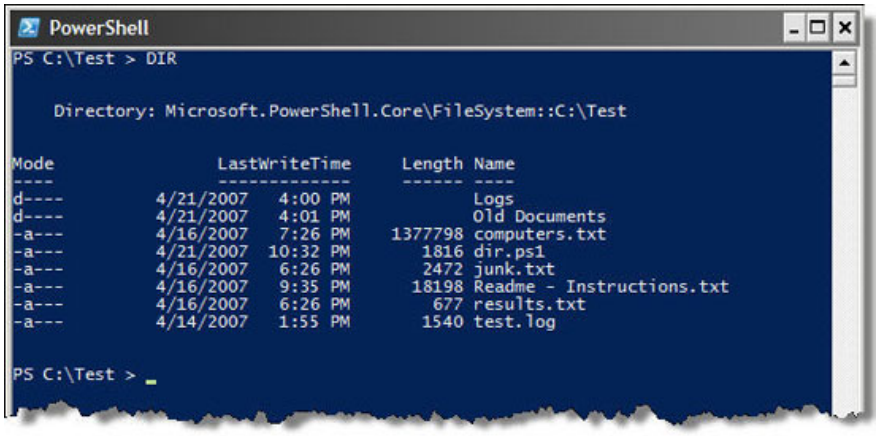

Figure 2

For the example scenario, you will need to create a PowerShell script that simulates the CMD.EXE DIR command. Below I will explain the most essential parts of a script.

DIR.PS1: Header (Header)

A PowerShell script includes PowerShell commands in a plain text file with extension .PS1. Replace DIR, you will use a text file called DIR.PS1.

To run the script, type the following command in the PowerShell screen:

.DIR.PS1 X: Folder

Where X is the drive partition character (such as C, D, E) and Folder is the folder name.

If you want to know some information about the drive partition, you will have to use Windows Management Instrumentation (WMI). Details of WMI are beyond the scope of this article so we won't mention it here. But the PowerShell code below is pretty easy to understand without using WMI help. You can create a variable '$ filter' to use with the Get-WmiObject command. This filter (ie filter) tells the Get-WmiObject command that you only want information about a specific drive. The result of the Get-WmiObject command is stored in a variable called $ volInfo. Remember, in PowerShell everything is an object; $ volInfo is now also a result object returned from Get-WmiObject.

$ filter = "DeviceID = '" + $ drive + ":'"
$ volInfo = Get-WmiObject -Class Win32_LogicalDisk -Filter $ filter

You can now access all objects and methods associated with the object. The serial number of the drive partition is accessible via the VolumeSerialNumber attribute. The information returned is a sequence of 8-character strings. But often you want to format it in four separate numbers, separated by a hyphen. You can do the same as in the line below. The hyphen at the end of the first line is the line serial character in PowerShell. Basically, it just tells PowerShell that the line is not interrupted but includes the next line. When writing code without separating lines, but to reduce the width and let the code read, you should do this.

$ serial = $ volInfo.VolumeSerialNumber.SubString (0, 4) + "-" + `
$ volInfo.VolumeSerialNumber.SubString (4, 4)

Now that there is a $ volInfo object, you can write DIR header information for the screen. If the drive does not have a name, the text for the screen will be a bit different from the drive named. A simple If-Else command is used to check if the VolumeName attribute is empty. The Write-Host command is used to write each command line to the screen.

If ($ volInfo.VolumeName -eq "") {Write-Host ($ drive + "has no label")} Else {Write-Host ("Volume in drive" + $ drive + "is" + $ volInfo.VolumeName)} Write-Host ("Volume Serial Number is" + $ serial) Write-Host ("` n Directory of "+ $ args [0] +" `n")

The '`n' character at the beginning and end of the Write-Host command is used to insert new lines before and after the text. The Write-Host command adds a new line at the end of each line. Therefore the effect of '`n' is to create a blank line before and after the text line.

Did you notice the '-eq' cluster in the If command? It is a comparison operator. The table below will tell you all comparison operators:

-eq, -ieq Compare with -ne, -ine Compare not equal to -gt, -igt Compare larger -ge, -ige Compare greater than or equal to -lt, -ilt Compare smaller than -le, - ile Compare smaller or equal
The -i character before the comparison operator is to indicate that the operator is not case-sensitive.

Figure 3: Output (output) of the script you currently have

DIR.PS1: File / folder list

Now you are ready to display the content and properties of this folder. The first thing to do is call the PowerShell Get-ChildItem command to bring up the set of files and put them into the script as a parameter. The Get-ChildItem command will retrieve the set of file and folder objects, not just names, but also pipe these objects directly into the Sort-Object command to sort them. By default, the Sort-Object command will sort the object based on the Name attribute. So you don't need to describe any other parameters. The set of sorted objects will then be stored in a variable named $ items.

$ items = Get-ChildItem $ args [0] |Sort-Object

Once you have a collection of file and folder objects, you need to loop through them and display the appropriate features. The command for this is ForEach. For each file or folder, the displayed feature will be the last updated date and time, name, length or file size. The strange thing that looks like strings in round brackets is the .NET string format code. They are used to align left / right for fields and format dates, times, and numbers. Understanding these string formats is not very important, because they are not essential to reflect the nature of this script.

The If command is where you determine if there is a directory object. If the first character of the Mode attribute is 'd', the object is a directory. You need to check again because the code for the directory is often different from the code written for the file.

Notice the line $ totalDirs ++ inside the If command. This is the counter responsible for tracking the directory number. Similarly, there is a $ totalFiles variable used to track the total size of all files. These values ??are always calculated during execution. But they are only displayed when the file listing process ends.

```
ForEach ($ i In $ items)
{
$ date = "{0, -20: MM / dd / yyyy hh: mm tt}" -f $ i.LastWriteTime
$ file = $ i.Name
If ($ i.Mode.SubString (0, 1) -eq "d")
{
$ totalDirs ++
$ list = $ date + "{0, -15}" -f "
"+" "+ $ file
}
Else
{
$ totalFiles ++
$ size = "{0, 18: N0}" -f $ i.Length
$ list = $ date + $ size + "" + $ file
}
$ totalSize + = $ i.Length
Write-Host $ list
}
```

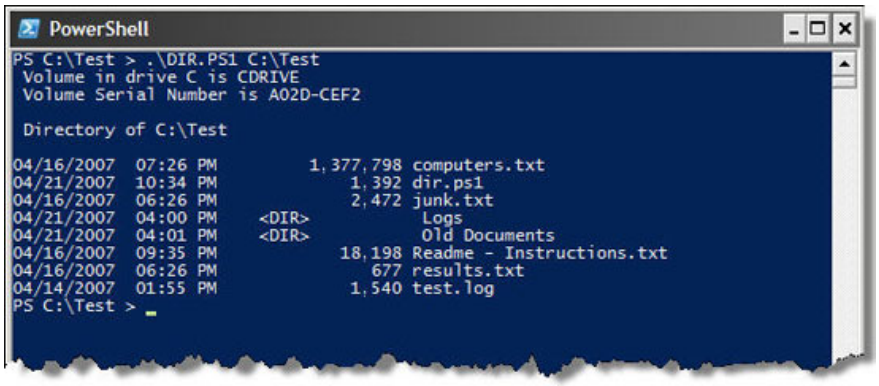

Figure 4: Displaying the output of the updated script.

DIR.PS1: Postfix (Footer)

The only thing left is to write on the screen the total number of files, folders, total size of all files and free space on this drive partition. To do this you will need to use the counter variables ($ totalFiles, $ totalDirs, $ totalSize) created in the previous section. You can know the amount of free space from the $ volInfo variable created from the start of writing the script.

```
Write-Host ("{0, 16: N0}" -f $ totalFiles + "File (s)" + `
"{0, 15: N0}" -f $ totalSize + "bytes")
```

```
Write-Host ("{0, 16: N0}" -f $ totalDirs + "Dir (s)" + `
"{0, 16: N0}" -f $ volInfo.FreeSpace + "bytes free`n")
```

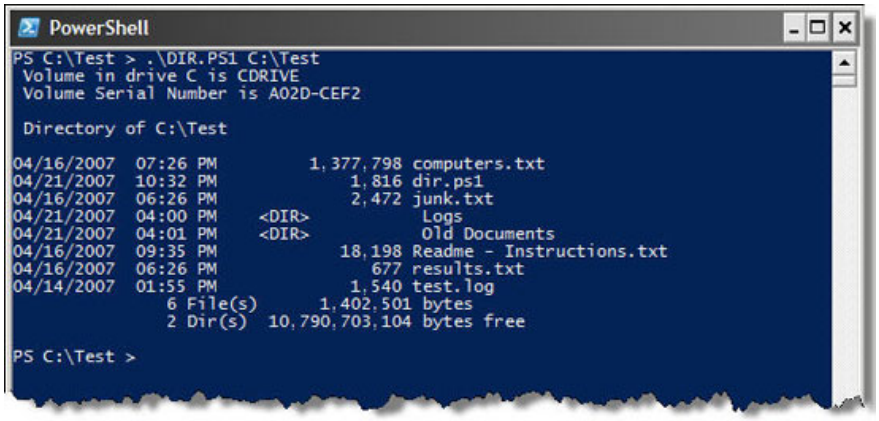

Figure 5: Completely displaying the output of the script.

The forecasts and the ability to improve can

Although the script you created gives the same output data as the CMD.EXE DIR command, there are some forecasts you need to know and some enhancements are possible.

1. This script does not perform any error checking.
2. If a valid path is not entered into the script, the script will fail with a PowerShell error message.
3. The total number of directories given in the script is less than 2 from the CMD.EXE DIR command because the Get-ChildItem command does not count two directories '.' and '.' as in CMD.EXE.
4. Your script only sorts order by file name, directory name and does not provide any other sorting properties.
5. Your script does not have the ability to display directory content and all subdirectories.

Conclude

Although PowerShell is a powerful scripting and scripting utility, you only need to spend some time being able to capture and use it, especially when not familiar with the .NET Framework environment. I hope this article and the example script will be useful for anyone who wants to understand PowerShell. But the script created in the example is quite simple. Believe that it can be built and developed more fully to better serve more complex applications.