Additional PowerShell Notes

Contents

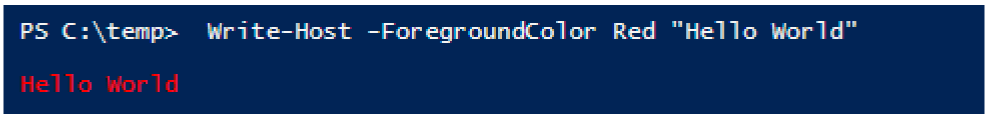
Changing Console output text colour:

To Change the colour of the "Write-host" console text you can use the "***-ForegroundColor***" option with a colour as follows:

Sample code:

```
Write-Host -ForegroundColor Red "Hello World"
```

Console output:



```
PS C:\temp>  Write-Host -ForegroundColor Red "Hello World"
Hello World
```

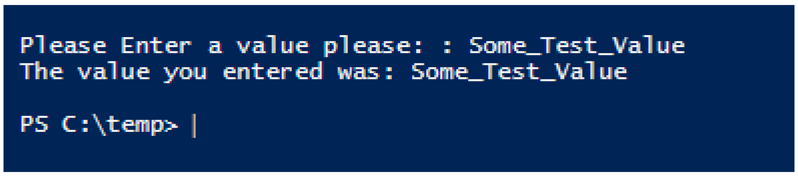
Prompting/Reading a console value:

To prompt a user to enter a value on the console you can use the "***Read-host***":

Sample code:

```
$SomeVariable = Read-Host "Please Enter a value please: "

Write-Host "The value you entered was: $SomeVariable"
```

Console output:

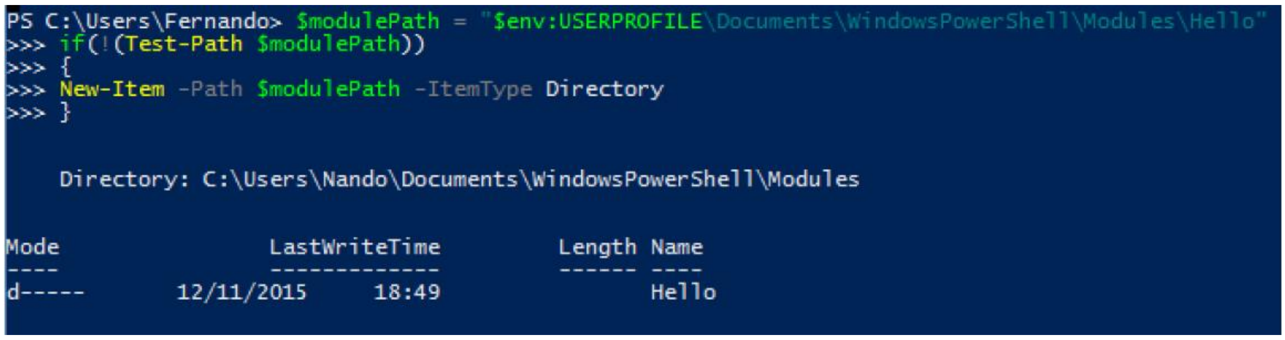


```
Please Enter a value please: : Some_Test_Value
The value you entered was: Some_Test_Value

PS C:\temp> |
```

Test if a Folder/File exits:

As already demonstrated in the notes one can use the "***Test-Path***" command to check if a file or folder exits.

The example below demonstrates checking if a path does NOT exists, and if not, then create a new Directory as stored in the variable ***$modulePath***.



```
PS C:\Users\Fernando> $modulePath = "$env:USERPROFILE\Documents\WindowsPowerShell\Modules\Hello"
>>> if(!(Test-Path $modulePath))
>>> {
>>> New-Item -Path $modulePath -ItemType Directory
>>> }


    Directory: C:\Users\Nando\Documents\WindowsPowerShell\Modules


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----       12/11/2015     18:49                Hello
```

To expand on this a little more verbosely we can use the "***-PathType***" options of "***Container***" or "***Leaf***" to specify if we are checking for folder or file, respectively.

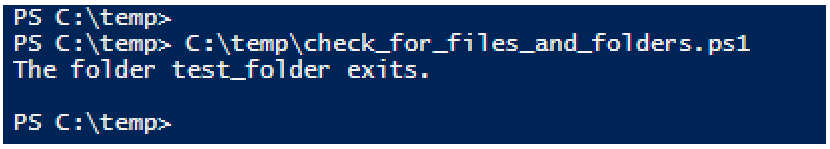
Examples as follows:

Test if a Folder exits:

Sample code:

```
$NameOfTestFolder = "test_folder"

if (Test-Path $NameOfTestFolder -PathType Container )
    {
    Write-Host "The folder $NameOfTestFolder exits."
    }
else
    {
    Write-Host "Cannot find the folder named $NameOfTestFolder."
    }
```

Console output:



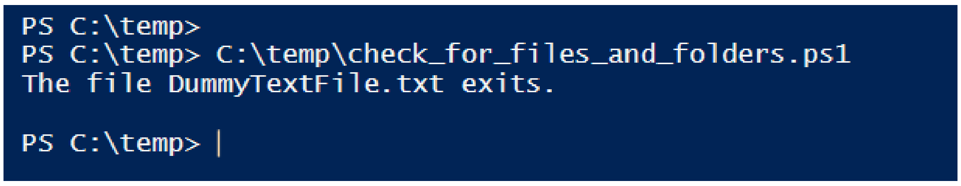
```
PS C:\temp>
PS C:\temp> C:\temp\check_for_files_and_folders.ps1
The folder test_folder exits.

PS C:\temp>
```

Test if a file exists:

Sample code:

```
$NameOfTestFile = "DummyTextFile.txt"

if (Test-Path $NameOfTestFile -PathType Leaf )
    {
    Write-Host "The file $NameOfTestFile exits."
    }
else
    {
    Write-Host "Cannot find the file named $NameOfTestFile."
    }
```

Console output:



```
PS C:\temp>
PS C:\temp> C:\temp\check_for_files_and_folders.ps1
The file DummyTextFile.txt exits.

PS C:\temp> |
```

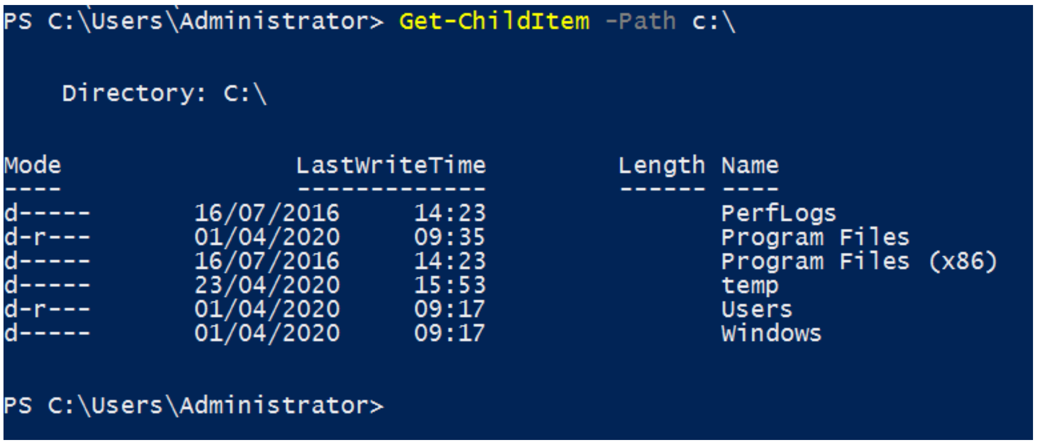
Basic Counting in PowerShell:

Sometimes counting lines/elements is a task we are required to execute as a system administrator. In PowerShell we have a number of methods available to us. The best depends on the scenario, or even personal choice.

Here I will present a few counting methods, using the directory listing of the root of C-drive as the subject matter.

Directory listing of C:\

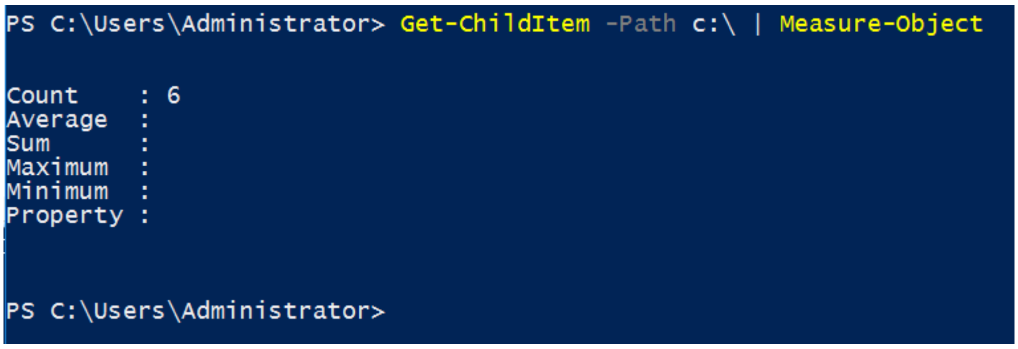
To obtain a directory listing of the c-drive we can use our usual "***dir***" command or use the "***Get-ChildItem***" cmdlet, as can be seen below there are 6 file/folders:



```
PS C:\Users\Administrator> Get-ChildItem -Path c:\

    Directory: C:\

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----       16/07/2016     14:23                PerfLogs
d-r---       01/04/2020     09:35                Program Files
d-----       16/07/2016     14:23                Program Files (x86)
d-----       23/04/2020     15:53                temp
d-r---       01/04/2020     09:17                Users
d-----       01/04/2020     09:17                Windows

PS C:\Users\Administrator>
```

Counting using Measure-object:

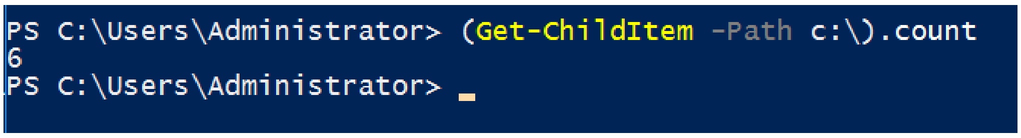
One method of counting the number of items is using the Measure-object cmdlet. The PowerShell measure-object cmdlet calculates the numeric properties of objects, and the characters, words, and lines in string objects, such as files of text. In the example below we use it to count the number of files/folders in the root of the C-drive. More information is available in the PowerShell documentation (link on Moodle).



```
PS C:\Users\Administrator> Get-ChildItem -Path c:\ | Measure-Object

Count    : 6
Average  :
Sum      :
Maximum  :
Minimum  :
Property :


PS C:\Users\Administrator>
```

Counting using the ().count method

One very quick way of counting is to use the ().count method, as can be seen below. In this example I have placed the "***Get-ChildItem***" command inside brackets and used the .count method.

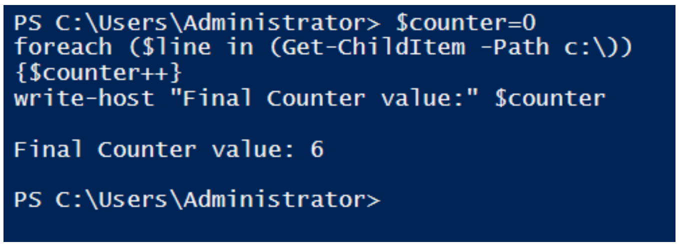


```
PS C:\Users\Administrator> (Get-ChildItem -Path c:\).count
6
PS C:\Users\Administrator> _
```

Counting using a loop:

Sometimes we may need to conduct the counting a little more manually; therefore using something like a "***foreach***" loop might be more useful.

In this example a loop is used to iterate through lines returned by the "***Get-ChildItem***" command, and to increment a counter each iteration, before finally displaying the last counter value.



```
PS C:\Users\Administrator> $counter=0
foreach ($line in (Get-ChildItem -Path c:\))
{$counter++}
write-host "Final Counter value:" $counter

Final Counter value: 6

PS C:\Users\Administrator>
```

Automatic Variables

[*Microsoft*] These are variables that store state information for PowerShell. These variables are created and maintained by PowerShell.

For more information visit:

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_automatic_variables?view=powershell-7.1

E.g.:

$?

The variable contains the execution status of the last command. It contains **True** if the last command succeeded and **False** if it failed.

$args

Contains an array of values for ***undeclared parameters*** that are passed to a **function, script, or script block**.

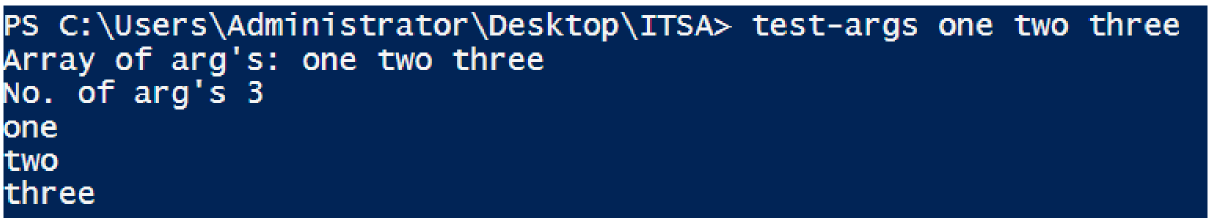
When you create a function, you can declare the parameters by using the ***param*** keyword or by adding a comma-separated list of parameters in parentheses after the function name.

In an event action, the ***$Args*** variable contains objects that represent the event arguments of the event that is being processed. This variable is populated only within the Action block of an event registration command. The value of this variable can also be found in the ***SourceArgs*** property of the ***PSEventArgs*** object that ***Get-Event*** returns.

Sample code of accessing command line (function) parameters using *$args*

Sample code:

```
function test-args
{
	write-host "Array of arg's:" $args
	Write-Host "No. of arg's" $args.count
	foreach ($item in $args) { write-host $item}
}
```



```
PS C:\Users\Administrator\Desktop\ITSA> test-args one two three
Array of arg's: one two three
No. of arg's 3
one
two
three
```