

Read a File Line by Line in PowerShell

 lazyadmin.nl/powershell/read-file-line-by-line

October 18, 2024

We can use PowerShell to read a file line by line, whether it's a text file or a CSV file. This method allows us to process information from each line in the file individually.

To read a file, we use the `Get-Content` cmdlet in PowerShell. This cmdlet will output objects (the content) one by one into the pipeline. But if you assign it to a variable, then the content will be stored in an array.

In this article, I will show you how to use the `Get-Content` cmdlet to read a file line by line with some examples and give tips for working with larger files.

Read a File line by Line

To read the contents of a text file in PowerShell, you can use the `Get-Content` cmdlet. By default, this cmdlet will return each line of the text file as a separate string to the pipeline. This means that you can process each line immediately after it's read by PowerShell.

For example, to read the contents of the text file "file1.txt" and show each line in the console, we can do:

```
Get-Content -path "C:\temp\File1.txt" | ForEach-Object {  
    Write-Host "The content of this line is - $_"  
}
```

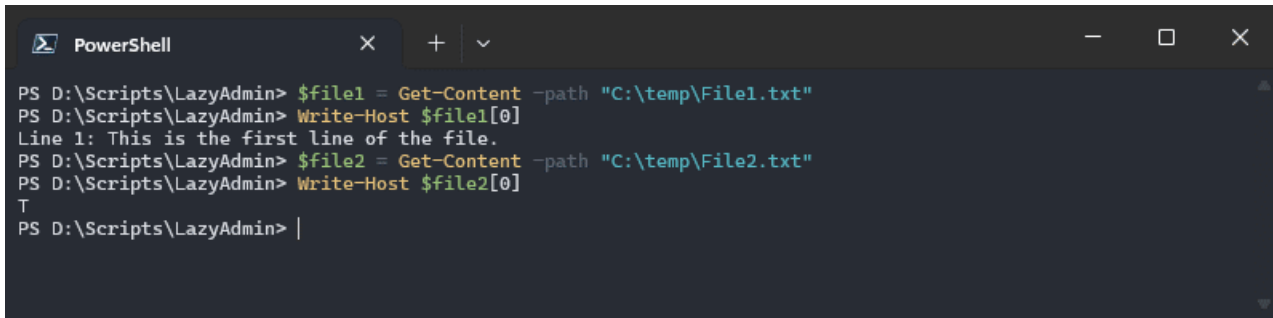
Another option is to assign the results to a variable. In this case, each line will be stored in an array as a string, allowing you to loop through it later on:

```
$file1 = Get-Content -path "C:\temp\File1.txt"  
ForEach($line in $file1) {  
    Write-Host $line  
}
```

There is however an important note to make here. If the file only contains one line of text, then the content is stored as a string, and not as an array. In most situations, this won't be an issue, but if you, for example, want to get the first line from the array, then you can get some unexpected results:

```
$file1 = Get-Content -path "C:\temp\File1.txt"  
# Return the first line:  
Write-Host $file1[0]  
# Result : Line 1: This is the first line of the file.  
$file2 = Get-Content -path "C:\temp\File2.txt"  
# Return the first line:  
Write-Host $file2[0]
```

Result : T

A screenshot of a PowerShell terminal window. The window title is 'PowerShell'. The prompt is 'PS D:\Scripts\LazyAdmin>'. The user enters '\$file1 = Get-Content -path "C:\temp\File1.txt"'. The prompt changes to 'PS D:\Scripts\LazyAdmin>'. The user enters 'Write-Host \$file1[0]'. The output is 'Line 1: This is the first line of the file.'. The prompt changes to 'PS D:\Scripts\LazyAdmin>'. The user enters '\$file2 = Get-Content -path "C:\temp\File2.txt"'. The prompt changes to 'PS D:\Scripts\LazyAdmin>'. The user enters 'Write-Host \$file2[0]'. The output is 'T'. The prompt changes to 'PS D:\Scripts\LazyAdmin> |'.

To prevent this, we can use the array-subexpression operator, which ensures that the result is always an array:

```
$arrayFromFile = @(Get-Content -path "C:\temp\File2.txt")
```

or cast an array:

```
[array]$arrayFromFile = @Get-Content -path "C:\temp\File2.txt"
```

Reading Large Files

When you need to read large files, or when performance in general is a concern, then it's better to use the direct .NET method to read the files.

Important to note is that when you use the .NET method, you should always use the absolute path to file, instead of the relative path. The reason for this is that the .NET method sometimes has a different working directory than PowerShell.

```
$arrayFromFileNET = [IO.File]::ReadAllLines("C:\temp\File2.txt")
```

I have done some testing to measure the difference between the two methods, and the .NET method is almost 10x faster than the Get-Content method. For the test, I used a 13 MB log file and ran each test 5 times to get a good average.

```
9  # Run the test 5 times
10 for ($i = 1; $i -le 5; $i++) {
11     # Measure the time for Get-Content
12     $timeGetContent = Measure-Command {
13         $ArrayFromFile = Get-Content -Path 'D:\php_errorlog'
14     }
15
16     # Measure the time for [IO.File]::ReadAllLines
17     $timeReadAllLines = Measure-Command {
18         $arrayFromFileNET = [IO.File]::ReadAllLines('D:\php_errorlog')
19     }
20 }
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\Scripts\LazyAdmin>
PS D:\Scripts\LazyAdmin> . 'D:\Scripts\LazyAdmin\dev2.ps1'
Iteration 1 : Get-Content took 552.2796 ms
Iteration 1 : [IO.File]::ReadAllLines took 46.4149 ms
Iteration 2 : Get-Content took 782.2115 ms
Iteration 2 : [IO.File]::ReadAllLines took 42.402 ms
Iteration 3 : Get-Content took 602.6876 ms
Iteration 3 : [IO.File]::ReadAllLines took 66.3308 ms
Iteration 4 : Get-Content took 861.114 ms
Iteration 4 : [IO.File]::ReadAllLines took 43.5873 ms
Iteration 5 : Get-Content took 531.8015 ms
Iteration 5 : [IO.File]::ReadAllLines took 88.9022 ms
Average time taken by Get-Content: 666.01884 ms
Average time taken by [IO.File]::ReadAllLines: 57.52744 ms
```

Reading the First or Last line only

Sometimes you don't need to read all the lines of a text file, but you might only want to read the first or last line of the file. To do this, we can use the `-First` and the `-Tail` parameter of the `Get-Content` cmdlet.

To read the last line of a text file:

```
Get-Content -Path "C:\temp\File1.txt" -Tail 1
```

And to read only the first line you can do:

```
Get-Content -Path "C:\temp\File1.txt" -First 1
```

You can increase the number to reach the first `n <number>` or last `n <number>` of lines of a file.

Wrapping Up

For most cases, the `Get-Content` cmdlet is the best option to read a file line by line in PowerShell. But when you need to process large files, then it's definitely worth using the `.NET` method instead. It's a lot faster then compared to the PowerShell cmdlets.

Hope you like this article, make sure to download the [free PowerShell Cheat Sheet](#) and if you have any questions, just drop a comment below.

Did you **Liked** this **Article**?

Get the latest articles like this **in your mailbox**
or share this article

Не удается связаться с сервисом reCAPTCHA. Проверьте подключение к Интернету и перезагрузите страницу.

I hate spam to, so you can unsubscribe at any time.