



# 3 ways to download files with PowerShell

3 Apr 2015 | Jourdan Templeton | 4 minute read

Perhaps the greatest strength of PowerShell is it's foundation on the .NET framework. The .NET framework enables almost unlimited possibilites inside the scripting realm. This blessing can equally be a curse as things can get complicated. Fast.

This post will describe three methods for downloading files using PowerShell - weighed up with their pros and cons.

## Test setup

Today's testing is not highly scientific. The point is to show the difference in execution time and performance.

The test setup consists of PowerShell 4 running on Windows 8.1 x64 with my VDSL connection curently synced at 35.9/10.4 mbps.

I will be downloading a test file from Internode at the following URL: <http://mirror.internode.on.net/pub/test/-10meg.test>. The scripts will be executed 10 times each with the average displayed as the result.

Let's get started!

## 1. Invoke-WebRequest

The first and most obvious option is the Invoke-WebRequest cmdlet. It is built into PowerShell and can be used in the following method:

powershell

```
$url = "http://mirror.internode.on.net/pub/test/10meg.test"
$output = "$PSScriptRoot\10meg.test"
$start_time = Get-Date

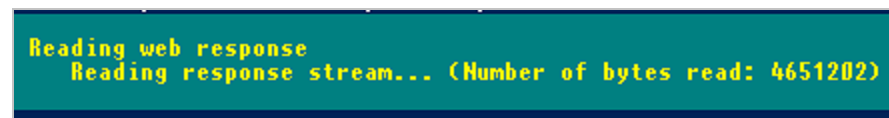
Invoke-WebRequest -Uri $url -OutFile $output

Write-Output "Time taken: $((Get-Date).Subtract($start_time).Seconds) second(s)"
```

Time taken: 27 seconds

## Pros

With the cmdlet already available it is super easy to get started and use. Integration with Write-Progress is handy while watching paint-dry scripts run (assuming you know the total file size). Cookies can also be persisted between mutiple requests through the use of the **-Session** and **-WebSession** parameters.



## Cons

memory issues for large files. If anyone knows specifics on how this cmdlet operates, let me know!.

```
Writing web request
Writing request stream... (Number of bytes remaining: 5790000)
```

Another potentially serious con for this method is the reliance on Internet Explorer. For example, this cmdlet cannot be used on Windows Server Core edition servers as the Internet Explorer binaries are not included by default. In some cases you can use the **-UseBasicParsing** parameter, but it does not work in all cases.

### Verdict

This cmdlet shines when you need to persist cookies across multiple requests (for instance HTTP Forms Auth before downloading the file).

Performance is *good enough* for small downloads, but there are definitely better options for situations where speed is required. If the script is to be run on a server running Windows Server Core, choose a more universal method.

## 2. System.Net.WebClient

A common .NET class used for downloading files is the System.Net.WebClient class.

powershell

```
$url = "http://mirror.internode.on.net/pub/test/10meg.test"
$output = "$PSScriptRoot\10meg.test"
$start_time = Get-Date

$wc = New-Object System.Net.WebClient
$wc.DownloadFile($url, $output)
#OR
(New-Object System.Net.WebClient).DownloadFile($url, $output)

Write-Output "Time taken: $((Get-Date).Subtract($start_time).Seconds) second(s)"
```

Time taken: 7 seconds

### Pros

This method is also easy to use. Not as syntactically nice as Invoke-RestMethod - yet can still be executed on a single line. Speed is great as the HTTP response stream is buffered to disk throughout the download process.

There is also the option of **System.Net.WebClient.DownloadFileAsync()**. This can be very handy if you'd like your script to continue while the file downloads in parallel.

### Cons

There is no visible progress indicator (or any way to query the progress mid transfer). It essentially blocks the thread until the download completes or fails. This isn't a major con, however sometimes it is handy to know how far through the transfer you are.

### Verdict

machines and 100% compatible with your Azure Automation runbooks.

### 3. Start-BitsTransfer

If you haven't heard of BITS before, [check this out](#). BITS is primarily designed for asynchronous file downloads, but works perfectly fine synchronously too (assuming you have BITS enabled).

```
powershell

$url = "http://mirror.internode.on.net/pub/test/10meg.test"
$output = "$PSScriptRoot\10meg.test"
$start_time = Get-Date

Import-Module BitsTransfer

Start-BitsTransfer -Source $url -Destination $output

#OR

Start-BitsTransfer -Source $url -Destination $output -Asynchronous

Write-Output "Time taken: $((Get-Date).Subtract($start_time).Seconds) second(s)"
```

Time taken: 6 seconds

### Pros

This method proved to be the fastest in my test cases! Extensive integration with Write-Progress gives you a clear indicator of the file size and progress. The **-Asynchronous** flag can be used to queue transfers asynchronously. This method is also incredibly flexible supporting separate credentials for the destination server AND web proxy, if required.

Personally, the biggest benefit to using the Start-BitsTransfer method is the ability to set retry actions on failure and limiting the amount of bandwidth available to a transfer.

```
BITS Transfer
This is a file transfer that uses the Background Intelligent Transfer Service (BITS).
[ooooooooooooooooooooooooooooooooooooooooooooooooooooo]
Transferring
```

### Cons

While BITS is enabled by default on many machines, you can't guarantee it is enabled on all (unless you are actively managing this). Also with the way BITS is designed, if other BITS jobs are running in the background, your job could be queued or run at a later time hindering the execution of your script.

### Verdict

This method is perfect for scenarios where you want to limit the bandwidth used in a file download or where time isn't a major issue. I have used this to sync files nightly at full speed and during the day at half speed using Transfer Policies. BITS is also easy to monitor and audit.


### Conclusion

We can see the obvious/easiest choice isn't always the best. I would recommend System.Net.WebClient due to it's universal nature and performance. BITS my second choice due to it's flexibility and managability.







Do you know other methods? Let me know!


36 Comments

 Login ▾



LOG IN WITH





OR SIGN UP WITH DISQUS 

 19

• Share

[Best](#) [Newest](#) [Oldest](#)



**Honza Kužel** 

9 years ago

Nice article :). In method 2. System.Net.WebClient you can use DownloadProgressChanged and/or DownloadFileCompleted event to get progress.

Little and example (with waiting until download has been completed) based on your article: