How to use cURL in PowerShell

lazyadmin.nl/powershell/using-curl

November 14, 2024

The command-line tool cURL (client URL) is commonly used to send or receive data to a server. The tool's advantage is that it supports multiple protocols and offers features like authentication, proxy, transfer resume, and more.

We can also use cURL in PowerShell, but there are a few important things that you need to know. And PowerShell also has it's own methods to transfer data to and from servers.

In this article, I will explain how you can use cURL in PowerShell, and give you some examples.

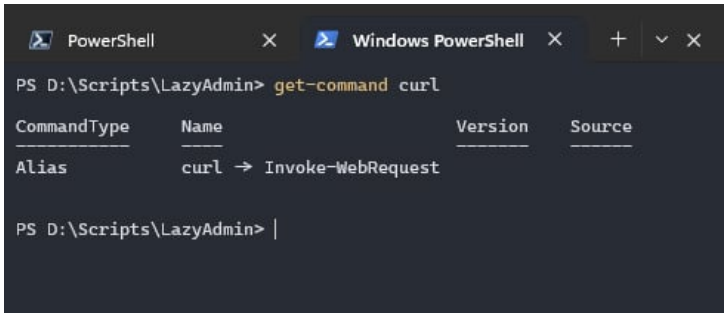
PowerShell cURL

cURL has been around since 1997 and is used to interact with web services, to send or retrieve data. The tool is commonly used in scripting or to automate tasks. Naturally, you might want to use cURL in your PowerShell scripts as well.

Now before we can use cURL in PowerShell there is one important thing that you need to know. Up to PowerShell 6, the command cURL has been an alias in PowerShell for the cmdlet `Invoke-WebRequest`. This caused many issues/confusions, so since PowerShell 6, the command cURL now actually calls the curl.exe utility.

We can see this by looking up the command with the `Get-Command` cmdlet in both PowerShell 5 and PowerShell 7:

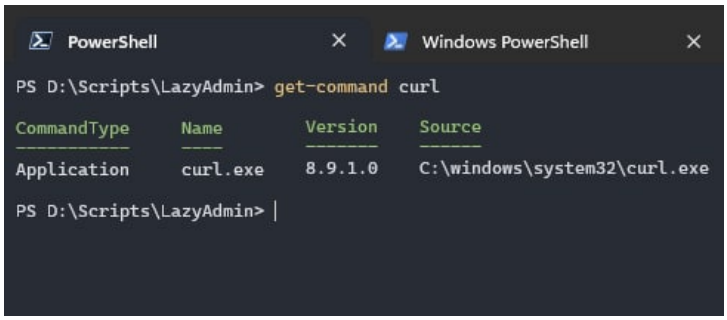
As you can see, the command curl call the curl.exe in PowerShell 7, just like we expect. But in PowerShell 5.1, the command is actually an alias.



PS 5.1

The problem with this is that the whole syntax of the `Invoke-WebRequest` cmdlet is different from cURL, so your commands won't work as you expect.

cURL is installed on Windows 10/11 by default, but if you don't have cURL installed, then you can download it here.



PS 7

Removing the Alias cURL

If you want to use the real cURL command-line utility in PowerShell 5.1, then you have two options. You can use curl.exe as a command, or remove the alias. I prefer the latter because this will make sure that your scripts also work on PowerShell 7. To remove the alias you can use the following command:

Remove-Item alias:curl

Using cURL in PowerShell

So let's take a look at some examples of how to use cURL in PowerShell. In the example below I will show the true cURL method and the PowerShell version, `Invoke-WebRequest` to give you a better understanding of the differences.

Basic GET Request

To simply get the contents of a specific URL, we can use the commands below.

curl https://httpbin.org/ip
Invoke-WebRequest -Uri "https://httpbin.org/ip"
In this case it's better to use, because the output is in JSON
Invoke-RestMethod https://httpbin.org/ip

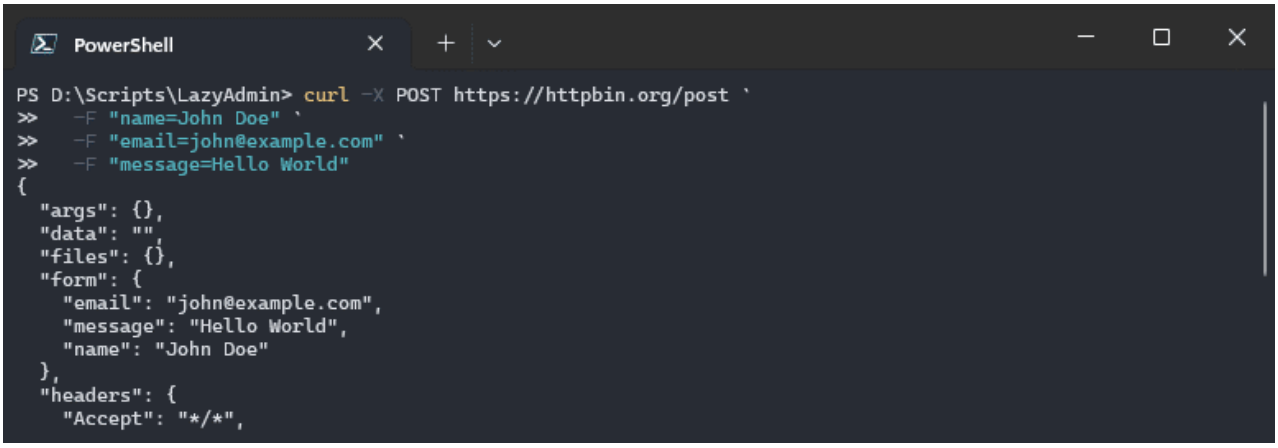
Sending Data

The httpbin endpoint that I am using in the examples below will just return the data that we send to it, but this is perfect in this example. To send data with cURL, we need to set the HTTP method to POST.

curl -X POST https://httpbin.org/anything -d "test=data"
The Invoke-WebRequest method uses the same principle, only with a different syntax:

$body = "test=data"
Invoke-WebRequest -Uri "https://httpbin.org/anything" -Method POST -Body "test=data"
In the examples above we only send a single value to the endpoint, but more common is to submit a form or send multiple fields to the endpoint. In cURL we can do this by specifying the fields using the -F parameter.

```
# POST form data with multiple fields
curl -X POST https://httpbin.org/post `
-F "name=John Doe" `
-F "email=john@example.com" `
-F "message=Hello World"
# Or for a JSON payload
curl -X POST https://httpbin.org/post -H "Content-Type: application/json" `
-d '{"field1":"value1","field2":"value2","field3":"value3"}'
```



When using the `Invoke-WebRequest` method, we can first create a hashtable with the formdata that we want to post. And to keep the code readable, we can also use a hashtable and splatting to store all the parameters

```
$formData = @{
name = "John Doe"
email = "john@example.com"
message = "Hello World"
}
$post = @{
Uri = "https://httpbin.org/post"
Method = 'Post'
Body = $formData
}
Invoke-WebRequest @post
```

Using Authentication

Most APIs require a form of authentication to be able to transfer data with it. When it comes to authentication there are different methods, but basic authentication (username & password) or bearer token authentication are the most commonly used options.

To use basic authentication with cURL, you will need to pass the username and password in the format `username:password`. In the example below we will add the username and password as well in the URL, but that is only for testing purposes:

```
curl -u "username:password" https://httpbin.org/basic-auth/username/password
```

```
# returns on succesfull authentication
{
"authenticated": true,
"user": "username"
}
```

When using the PowerShell method, `Invoke-WebRequest`, we first need to create a credential object. You can ask the user to input the credentials, with the `Get-Credential` cmdlet, but when you want to store the credentials in your script, you will need to convert them to a secure string first:

```
# Get the credentials from the user
$Credentials = Get-Credential -UserName "username" -Message "Enter your password"
# Or store the credentials in the script
$Password = ConvertTo-SecureString "password" -AsPlainText -Force
$Credentials = New-Object System.Management.Automation.PSCredential ("username", $Password)
Invoke-WebRequest -Uri "https://httpbin.org/basic-auth/username/password" -Credential $Credentials
```

Wrapping Up

It's a good thing that Microsoft removed the cURL alias in PowerShell 7, because eventhough cURL and Invoke-WebRequest may look similar, cURL has a lot more capabilites. If you are creating scripts that are going to run on other machines, then you can use the command `curl.exe` to be sure that cURL is used instead of Invoke-WebRequest.

Hope you liked this article, if you have any questions, just drop a comment below.