How to use Start Process in PowerShell

lazyadmin.nl/powershell/start-process

March 10, 2022

To run an application, process, or script from within PowerShell you could simply enter the path to file. But this will start the process in the same environment and within the same context as your PowerShell session.

When you want to run the process as a different user, start it in a new window, or even start multiple processes simultaneously then you will need to use the Start-Process cmdlet.

In this article, we are going to take a look at the start-process cmdlet. How we can run a process with elevated permissions, run it in a new window, or even completely hidden. I will also give you a couple of useful examples to get started.

Using Start-Process in PowerShell

The Start-Process cmdlet allows you to run one or multiple processes on your computer from within PowerShell. It's designed to run a process asynchronously or to run an application/script elevated (with administrative privileges).

You don't need to use the Start-Process cmdlet if you need to run a script or other console program synchronously in PowerShell. The reason for this is that you can redirect the output of it to PowerShell.

This is one of the downsides of the cmdlet, you can't redirect the output or error streams to PowerShell. The only option that you have is to redirect the output to text files.

So let's take a look at how to use the cmdlet. We can use the following parameters to start a process:

Parameter	Description
-FilePath	Specify the file, application, or process to run
-ArgumentList	Specifies parameters to use with the process to start
-Credential	User account to run the process with
-WorkingDirectory	The location where the process should start in
-NoNewWindow	Don’t open a new window for the process
-RedirectStandardError	Specify text file to redirect error output to
-RedirectStandardInput	Text file with input for the process
-RedirectStandardOutput	Specify text file to redirect output to
-WindowStyle	Normal, Hidden, Minimized, or Maximized
-Wait	Wait for the process to finish before continuing with the script
-UseNewEnvironment	The process will use its own environment variables instead of those of the PowerShell session

Start-Process parameters

So to simply open an application with PowerShell we could use the following command:

```
Start-Process Notepad.exe
# Simply typing notepad.exe in PowerShell will have the same result:
Notepad.exe
```

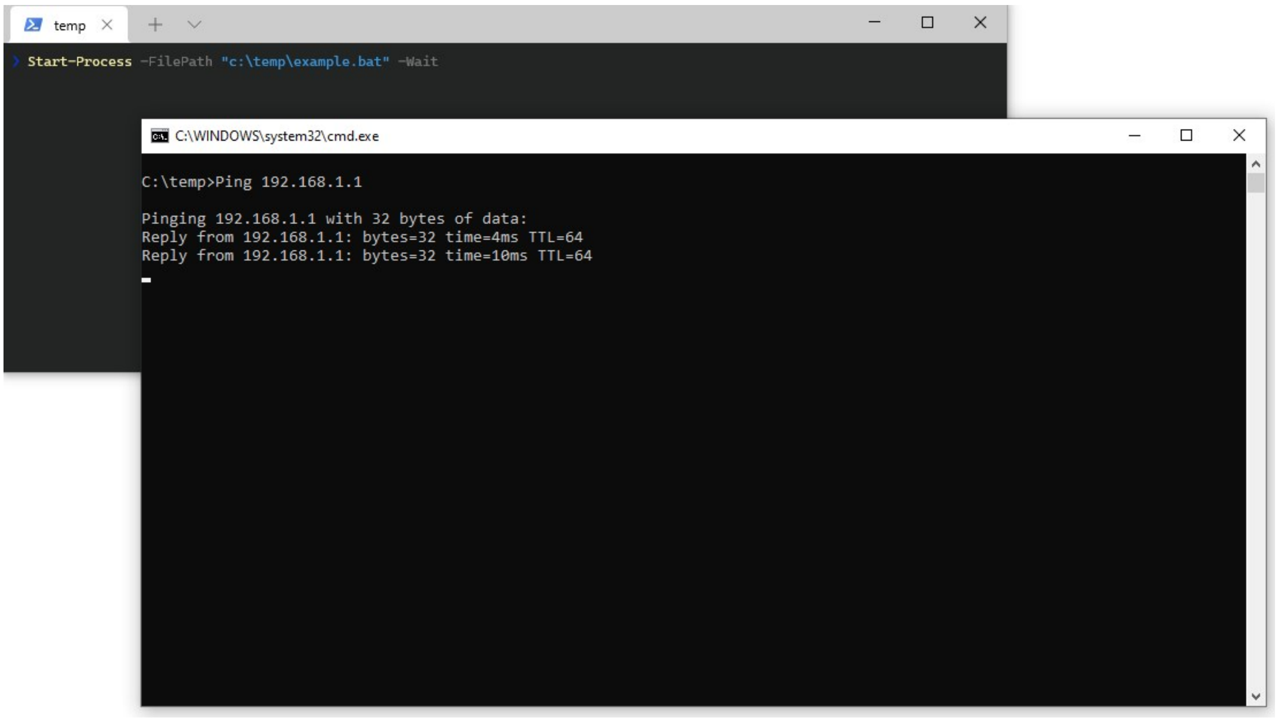
This will open Notepad in a new window with the same privileges as the PowerShell session. The process is run asynchronously, which means that PowerShell will continue the script, even if the process isn’t finished yet.

Waiting for a process to finish

The most common wait to start a process in PowerShell is to wait for it to finish. We can use the `-wait` parameter for this, which will make sure that PowerShell will wait until the process and all child-process are finished.

Let’s say we have a bat file that we want to start from PowerShell and wait for the process to finish:

```
# Start the process example.bat and wait for it to finish
Start-Process -FilePath "c:\temp\example.bat" -Wait
```



```
> Start-Process -FilePath "c:\temp\example.bat" -Wait
```

```
C:\temp>Ping 192.168.1.1

Pinging 192.168.1.1 with 32 bytes of data:
Reply from 192.168.1.1: bytes=32 time=4ms TTL=64
Reply from 192.168.1.1: bytes=32 time=10ms TTL=64
```

Start-Process running Example.bat in a new window

This will run the bat file and wait for it to finish before continuing the script or resuming input. Keep in mind that any output of the process isn't captured by default. So you won't know if the bat file failed or successfully completed.

Window size and Hidden processes

If we run the bat file with the example above, it will run the bat file in a new window. The window will have a normal size and close when the process completes. In the start-process cmdlet, we can specify the window size or even hide it completely.

We can choose between using the parameters `-WindowStyle` and `-NoNewWindow`. Obviously, we can't use both parameters together 😉

To run the bat file without any windows we can use the following command in PowerShell:

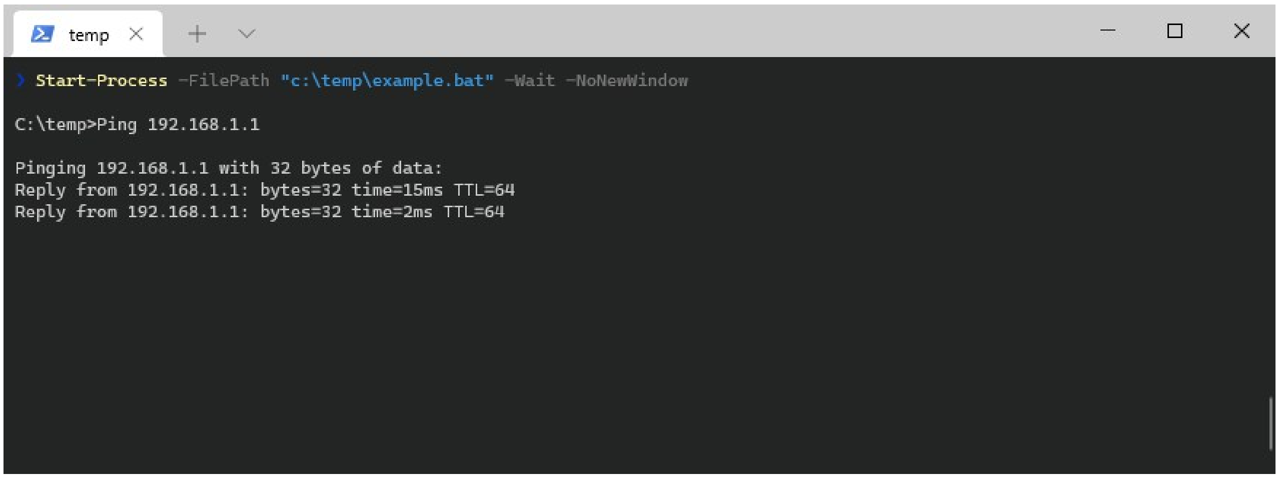
# Start the process example.bat, without any window and wait for it to finish
Start-Process -FilePath "c:\temp\example.bat" -Wait -WindowStyle Hidden
You won't get any feedback, except that your script will continue when the process is finished. However we can redirect the result of the process to a text file, more about that later.

To run a process in a maximized, normal, or minimized window you can use the following options:

# Default behavior:
Start-Process -FilePath "c:\temp\example.bat" -Wait -WindowStyle Normal
# Maximized
Start-Process -FilePath "c:\temp\example.bat" -Wait -WindowStyle Maximized
# Minimized

Start-Process -FilePath "c:\temp\example.bat" -Wait -WindowStyle Minimized
Another parameter that we can use is -NoNewWindow. This will run the process in the same window as the PowerShell script. This option only works with command-line-based processes. You can't for example, open Notepad in the same window as PowerShell.

Start-Process -FilePath "c:\temp\example.bat" -Wait -NoNewWindow



```
❯ Start-Process -FilePath "c:\temp\example.bat" -Wait -NoNewWindow

C:\temp>Ping 192.168.1.1

Pinging 192.168.1.1 with 32 bytes of data:
Reply from 192.168.1.1: bytes=32 time=15ms TTL=64
Reply from 192.168.1.1: bytes=32 time=2ms TTL=64
```

Run example.bat in the same PowerShell Window

Using Arguments with Start-Process

Some processes or scripts that you want to start require parameters (arguments). Probably your first thought is to add the arguments between the quotes in the filepath, but as you might have noticed, that won't work.

To pass arguments to the process that you want to start, you will need to use the `-arguments` parameter.

Let's run an MSI from PowerShell as an example. To run the MSI silently we will need to supply the arguments `/quiet` or `/qn` and we probably don't want to restart as well, so we add `/norestart` to it.

Start-Process -FilePath "C:\temp\example.msi" -Wait -ArgumentList "/quiet /norestart"
# Or arguments as string array:
Start-Process -FilePath "C:\temp\example.msi" -Wait -ArgumentList "/quiet","/norestart"

PowerShell Start-Process Elevated

When you start a process with Start-Process it will run in the same user context as the PowerShell session. But some processes may need elevated permissions to run. To do this we can use the -Verb parameter.

Keep in mind that you can't combine -Verb and -NoNewWindow because the process that you want to start must be opened in a new window.

To run the example.bat with elevated permissions we can use the following command:

Start-Process -FilePath "c:\temp\example.bat" -Wait -Verb RunAs
Depending on the file extension other options are also possible. We could for example print a text file with `-Verb Print`.

Start Process as a different user

It’s also possible to run a process as a different user. By default, the process will be executed with the credentials of the currently logged-on user.

First, you will need t to create a PSCredential object and store it as a secure string. Then you can pass the credentials to the cmdlet with the parameter -Credential.

Keep in mind that secure strings are not super secure to use, so make sure that you keep the secure string as safe as possible.

```
# Create credential object
# You can store these also in a text file
$username = Read-Host "Enter your username"
$secureStringPwd = Read-Host -assecurestring "Please enter your password"
# Create credential object
$credObject = New-Object System.Management.Automation.PSCredential -ArgumentList
$username, $secureStringPwd
Start-Process -FilePath "c:\temp\example.bat" -Wait -Credentials $credObject
```

Redirecting the Output

The output of the Start-Process cmdlet can’t be passed through to PowerShell. The only option that we have is to redirect the output to a text file. Variables won’t work.

So what you can do to capture the output is:

```
# Redirect the output to example-output.txt
Start-Process -FilePath "c:\temp\example.bat" -Wait -RedirectStandardOutput
c:\temp\example-output.txt
# Read the contents of example-output.txt
$output = Get-Content c:\temp\example-output.txt
```

What won’t work is :

```
# Storing the output into a variable will throw an error
Start-Process -FilePath "c:\temp\example.bat" -Wait -RedirectStandardOutput $output
# $result will be empty
$result = Start-Process -FilePath "c:\temp\example.bat" -Wait
```

If you want to capture only the error of the process then you can use:

```
Start-Process -FilePath "c:\temp\example.bat" -Wait -RedirectStandardError
c:\temp\example-output.txt
```

Getting the Process ID

The last option that I want to explain is the -Passtru parameter. It will return the process object of the process that we have started. This can be useful when you want to automatically stop a process when it's running for too long.

```
$process = Start-Process "C:\temp\example.bat" -PassThru
# Get the process ud
$process.ID
# Wait 1 second
Start-Sleep 1
# Kill the process
Stop-Process -id $process.Id
```

Wrapping Up

The Start-Process cmdlet is great to run one or multiple applications or scripts from within PowerShell. Unfortunately, we can't easily capture the output of the process, but with a small workaround, we are still able to inspect the results in PowerShell.

I hope you found this article useful, 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

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