

How to Set an Environment Variable in PowerShell

 lazyadmin.nl/powershell/set-environment-variable

February 1, 2024

Environment variables are used by the operating system and applications to store configuration settings, system paths, and other information. All programs on your computer can access this data. We can use PowerShell to view, change, or set an environment variable.

To set environment (env) variables you commonly use the System Properties screen, but when you need to add or change a variable on multiple computers, then it's easier to use PowerShell.

In this article, I will explain the different env variables and how to set environment variables with PowerShell.

Environment Variables

Before we are going to look at how to manage the variables with PowerShell, let's first take a closer look at the environment (env) variables to get a better understanding of them. Environment variables are basically key-value pairs that are part of the environment in which a process runs.

There are three scopes on Windows where an environment variable can be defined:

- User scope
- Machine (System) scope
- Process scope

The user scope contains mostly paths to the user's profile folder, like OneDrive, Appdata, or temp folder. In the machine scope, you will find information about the device, and crucial variables for the system to find and execute commands or programs without requiring the full path to be specified each time.

The process scope contains the variables from the current process (or PowerShell session). It will also list all variables from the user and machine scopes. Custom variables added to the process scope will be lost when the session is closed.

So to set environment variables that remain available after a reboot, and for all users, you will need to store them in the machine scope. To do this you will need to have the correct permissions to do this.

Get Environment Variables with PowerShell

Before we are going to add new environment variables to our system, let's first take a look at how to get the variables in PowerShell. We can use two methods to get and set environment variables in Powershell, the .NET method `[System.Environment]` and the PowerShell Environment provider `$env:`.

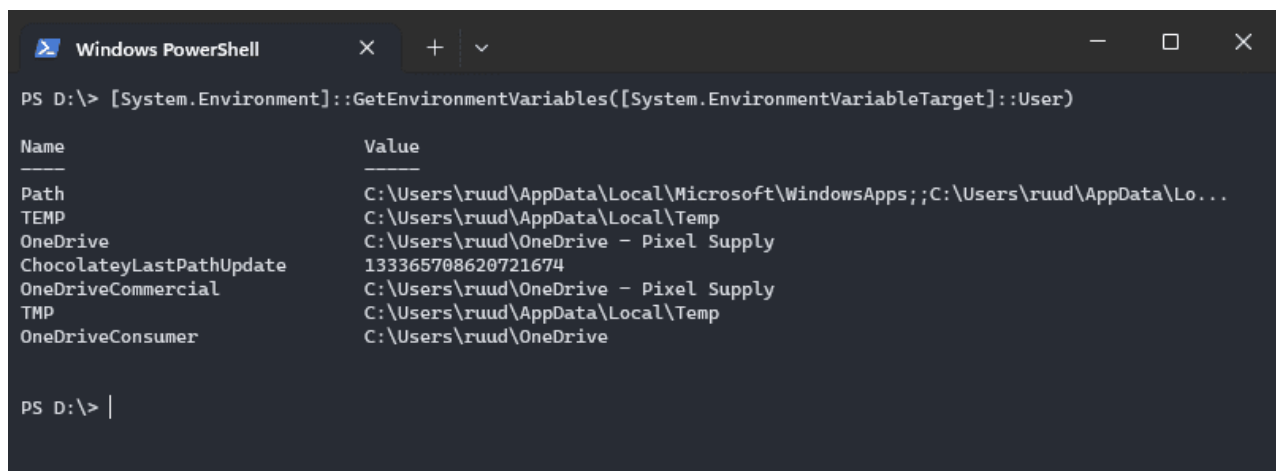
To list all environment variables you can use one of these commands:

```
# .NET method
[System.Environment]::GetEnvironmentVariables()
# Environment provider, sorting them by name
Get-Item env:* | sort-object name
# Or in short:
gi env:* | sort name
# Using the drive method
dir env:
```

The advantage of the .NET method is that you can specify the scope for the variable. This way we can only list the user environment variables for example. But when you know which variable you need, or want to view them all, then the Environment provider is a lot shorter of course.

So to view only the user environment variables, we can specify the user scope in the .NET method. Other options are `Machine` or `Process`

```
# Replace User with Machine or Process to view the other scopes
[System.Environment]::GetEnvironmentVariables([System.EnvironmentVariableTarget]::User)
```



```
Windows PowerShell
PS D:\> [System.Environment]::GetEnvironmentVariables([System.EnvironmentVariableTarget]::User)
```

Name	Value
Path	C:\Users\ruud\AppData\Local\Microsoft\WindowsApps;;C:\Users\ruud\AppData\Lo...
TEMP	C:\Users\ruud\AppData\Local\Temp
OneDrive	C:\Users\ruud\OneDrive - Pixel Supply
ChocolateyLastPathUpdate	133365708620721674
OneDriveCommercial	C:\Users\ruud\OneDrive - Pixel Supply
TMP	C:\Users\ruud\AppData\Local\Temp
OneDriveConsumer	C:\Users\ruud\OneDrive

```
PS D:\> |
```

View Specific Environment Variable

If you want to check the value of a specific variable, then you will need to know the name at least. We can then use the `$env:` method to view the value of the variable:

```
# Get the OneDrive folder path
$env:OneDrive
```

If you don't know the exact name of the variable, or you want to view all OneDrive-related variables for example, then you can use the `Get-Item` cmdlet combined with a wildcard `*`:

```
# Get all variables that start with OneDrive
```

```
Get-Item env:OneDrive*
```

```
# Result
```

```
Name Value
```

```
----
```

```
OneDrive C:\Users\ruud\OneDrive - Pixel Supply
```

```
OneDriveConsumer C:\Users\ruud\OneDrive
```

```
OneDriveCommercial C:\Users\ruud\OneDrive - Pixel Supply
```

```
# Or get all variables that contain the word Path
```

```
Get-Item env:*Path*
```

We can also get the environment variable with the .NET method. For this, you will need to use the command `[System.Environment]::GetEnvironmentVariable`. Note that this command looks almost identical to the .NET method, except the `s` is missing at the end of it.

To list the OneDrive variable value, we can use the following command. Note that we can also specify the scope of the variable, but that isn't necessary.

```
[System.Environment]::GetEnvironmentVariable("OneDrive", "User")
```

FREE EMAIL SERIES!

Level Up with PowerShell

5 Emails, Endless Skills

Set ENV Variable in PowerShell

To set an environment (env) variable in PowerShell you can use two methods, the Environment provider or the `[System.Environment]::SetEnvironmentVariable` method. The latter is recommended, because it allows you to set and change variables in the machine (system) scope as well.

For example, to create a new environment variable `TenantId`, with our tenant ID as a value, we can use the following command:

```
[System.Environment]::SetEnvironmentVariable('TenantId','11e1234-28gv-4413-aaf8-a1e8b7f4c3d6')
```

Now if we look at the variable, then we can see that it's set to the desired value:



```
PS D:\> [System.Environment]::SetEnvironmentVariable('TenantId','11e1234-28gv-4413-aaf8-a1e8b7f4c3d6')
PS D:\> Get-Item env:TenantId
```

Name	Value
TenantId	11e1234-28gv-4413-aaf8-a1e8b7f4c3d6

```
PS D:\> |
```

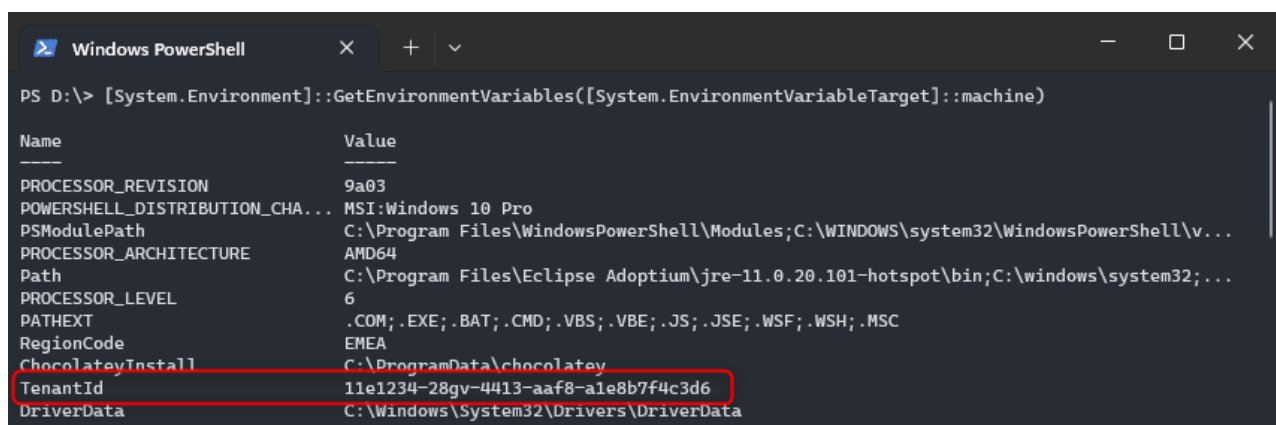
But there is one problem, if you close PowerShell, then the environment variable is gone. The reason for this is that any variable you add is stored in the process scope by default. We can solve this by adding the required scope, for example, the machine scope:

Note

To add or change variables in the machine scope, you will need to have elevated permissions. Run PowerShell in elevated (admin) mode, otherwise you get the error "Requested registry access is not allowed."

```
[System.Environment]::SetEnvironmentVariable('TenantId','11e1234-28gv-4413-aaf8-a1e8b7f4c3d6', 'Machine')
```

If we now look at the env variables from the machine scope, then you will see that it's listed:



```
PS D:\> [System.Environment]::GetEnvironmentVariables([System.EnvironmentVariableTarget]::machine)
```

Name	Value
PROCESSOR_REVISION	9a03
POWERSHELL_DISTRIBUTION_CHA...	MSI:Windows 10 Pro
PSModulePath	C:\Program Files\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerShell\v...
PROCESSOR_ARCHITECTURE	AMD64
Path	C:\Program Files\Eclipse Adoptium\jre-11.0.20.101-hotspot\bin;C:\windows\system32;...
PROCESSOR_LEVEL	6
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
RegionCode	EMEA
ChocolateyInstall	C:\ProgramData\chocolatey
TenantId	11e1234-28gv-4413-aaf8-a1e8b7f4c3d6
DriverData	C:\Windows\System32\Drivers\DriverData

Another option to add a persistent environment variable is to add the variable in your PowerShell Profile. This way the environment variable gets added/updated every time you open PowerShell.

For example, to add a new variable and to update the default Path variable you can add the following two lines in your Powershell profile:

```
$Env:UserPrincipalName = 'lab@lazyadmin.nl'
```

```
$Env:Path += ';C:\Tools'
```

Adding Temporary ENV Variables

When you don't specify a scope when adding a variable, then it's automatically stored in the process scope. This means that the variable is lost when you close your PowerShell session. So this method is great when you need to add a temporary environment variable.

To do this we can use the .NET method described earlier, or we can use the variable syntax or Environment provider. The variable syntax `$Env:` is the easiest method to quickly set or update a variable:

```
$env:UserPrincipalName = 'lab@lazyadmin.nl'
```

If you need to update the value, you can simply change it by setting a new value. The values are always a string, which means we can append values to the variable with the `+=` operator. This allows you to add a path to the existing PATH variable, for example.

```
# Change the value of the variable
```

```
$env:UserPrincipalName = 'test@lazyadmin.nl'
```

```
# Adding a value to the existing one:
```

```
$env:Path += ';d:\scripts'
```

Another method to create or update the variables is to use the Environment provider. For example, to quickly add an environment variable you can use the `New-Item` cmdlet:

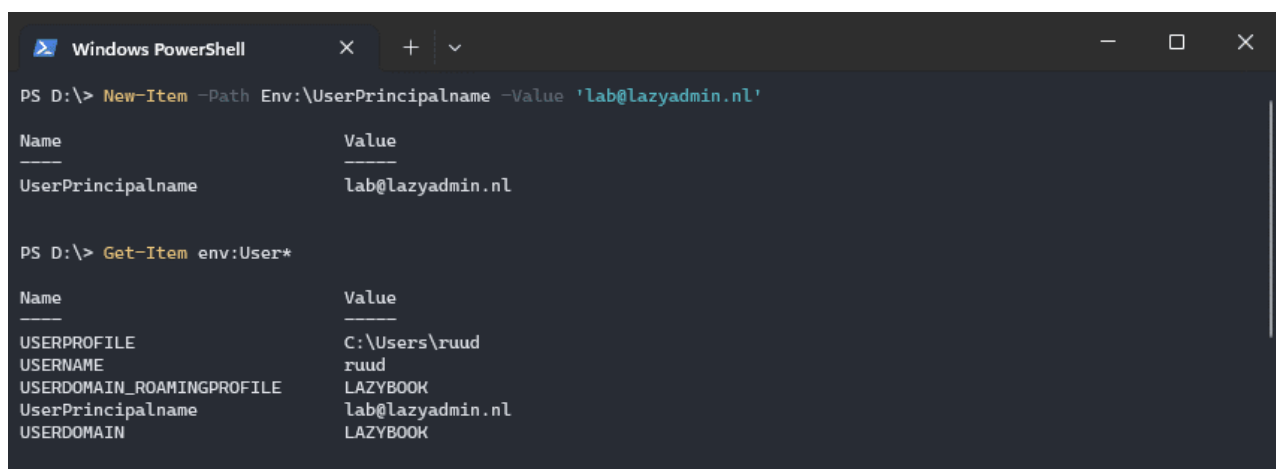
```
New-Item -Path Env:\UserPrincipalname -Value 'lab@lazyadmin.nl'
```

```
# Result
```

```
Name Value
```

```
----
```

```
UserPrincipalname lab@lazyadmin.nl
```



```
Windows PowerShell
PS D:\> New-Item -Path Env:\UserPrincipalname -Value 'lab@lazyadmin.nl'

Name                Value
----                -
UserPrincipalname    lab@lazyadmin.nl

PS D:\> Get-Item env:User*

Name                Value
----                -
USERPROFILE          C:\Users\ruud
USERNAME              ruud
USERDOMAIN_ROAMINGPROFILE LAZYBOOK
UserPrincipalname    lab@lazyadmin.nl
USERDOMAIN            LAZYBOOK
```

To update or change the value of the variable you can use the `Set-Item` cmdlet:

```
# Change the variable
```

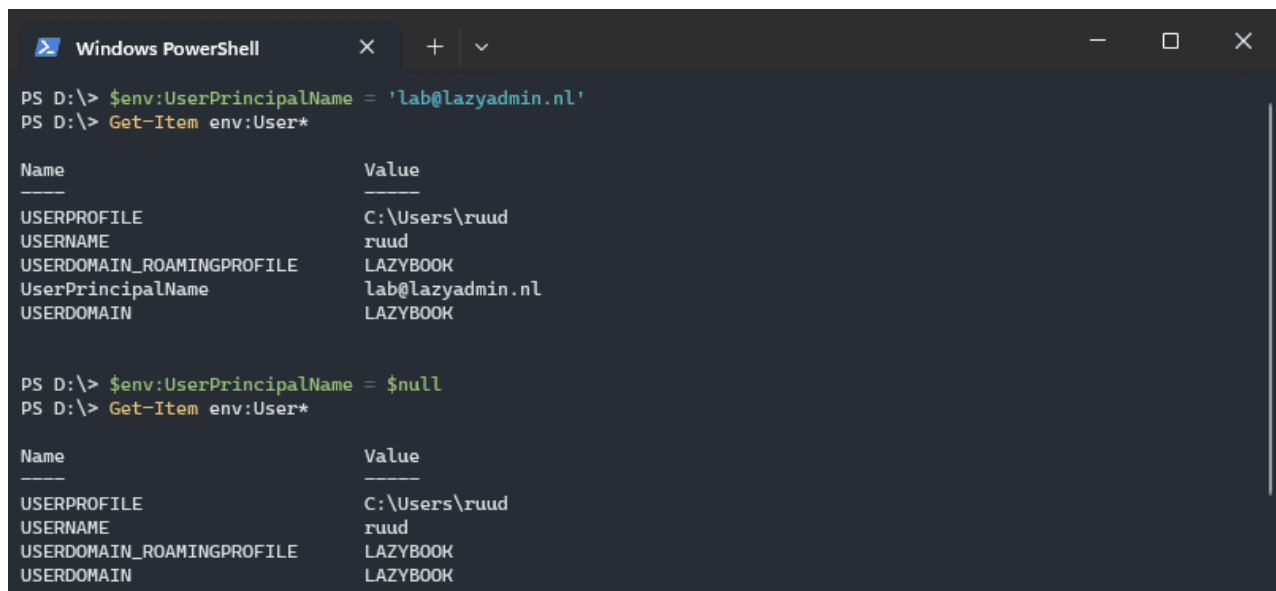
```
Set-Item -Path Env:\UserPrincipalname -Value 'test@lazyadmin.nl'
```

Removing Env Variables

An environment variable can't be empty. So to remove it, we can either set the value to an empty string (or `$null`), or we can use the `Remove-Item` cmdlet to remove the variable.

To set an env variable in PowerShell to an empty string you can use one of the methods described earlier:

```
# Set the variable to $null
$env:UserPrincipalName = $null
# Change it to an empty string
$env:UserPrincipalName = ""
# Set it to an empty string with the .NET method
[Environment]::SetEnvironmentVariable('UserPrincipalName', '', 'Machine')
# Or use the Environment provider
Set-Item -Path Env:\UserPrincipalname -Value ""
```



The screenshot shows a Windows PowerShell window with the following commands and output:

```
PS D:\> $env:UserPrincipalName = 'lab@lazyadmin.nl'
PS D:\> Get-Item env:User*
```

Name	Value
USERPROFILE	C:\Users\ruud
USERNAME	ruud
USERDOMAIN_ROAMINGPROFILE	LAZYBOOK
UserPrincipalName	lab@lazyadmin.nl
USERDOMAIN	LAZYBOOK

```
PS D:\> $env:UserPrincipalName = $null
PS D:\> Get-Item env:User*
```

Name	Value
USERPROFILE	C:\Users\ruud
USERNAME	ruud
USERDOMAIN_ROAMINGPROFILE	LAZYBOOK
USERDOMAIN	LAZYBOOK

Another option is to use the `Remove-Item` cmdlet. It will have the same effect as the methods above, but it may maybe a bit more clear that you are actually removing the variable.

```
Remove-Item -Path Env:\UserPrincipalname
```

Wrapping Up

There are different methods to set an env variable PowerShell. The easiest method is of course to use the environment variable, `$env:.` But if you want to create a persistent variable, then you will need to use the `.Net` method and set the scope to machine.

Keep in mind when creating a variable that double check if the variable not already exists. Overwriting system variables can have negative effects on other programs.

I hope this article helped you, 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.