# PowerShell Modules: A Beginner’s Guide to Extending Functionality

adamtheautomator.com/powershell-modules-guide

January 10, 2025

## Discovering Available PowerShell Modules

PowerShell modules are essential building blocks that extend its functionality. But, not all modules automatically load into your session—they often remain on your system, waiting to be discovered.

To explore the modules installed on your system, use the `Get-Module` command:

```
Get-Module
```

You might notice a short list of modules. The reason is because `Get-Module` only shows modules already imported into your session.

To see all available modules on your system, including those not yet loaded, append the `-ListAvailable` parameter:

```
Get-Module -ListAvailable
```

This command lists all modules available on your system, whether loaded into memory or not. PowerShell automatically imports modules as you use commands from them, so you typically don’t need to load them manually.

Modules can be of various types—script, binary, or manifest.

To group modules by type for easier inspection, try this:

```
gmo -ListAvailable | group ModuleType
```

Here, you use aliases for convenience.

For example, `gmo` is shorthand for `Get-Module`, and `group` is an alias for `Group-Object`. Aliases save time when running commands in the console.

## Uncovering Module Versions and Command Details

Each module has a version representing its feature set or changes over time. Versioning allows you to manage updates or revert to a previous version if necessary.

Modules also contain commands that are accessible via the `ExportedCommands` property.

To see the commands in the `Microsoft.PowerShell.Management` module, run:

```
gmo Microsoft.PowerShell.Management | Select-ExpandProperty ExportedCommands
```

This command displays a list of commands available in the module.

## Locating Module Directories

Modules don't magically work in PowerShell—they live in specific directories on your system, and PowerShell needs to know where to find them.

To show the locations of available PowerShell modules:

```
gmo -list
```

Common locations include:

- *C:\Program Files\PowerShell\7\Modules*
- *C:\Program Files\WindowsPowerShell\Modules*
- *C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules*

Even when running PowerShell Core, modules from Windows PowerShell directories may still appear because PowerShell Core can use them.

Modules have dedicated categories by their `PSEdition` property, which indicates compatibility:

- `Core` – Built for PowerShell Core.
- `Desk` – Designed for Windows PowerShell.
- `Both` – Compatible with both editions.

## Managing the `PSModulePath` Variable

PowerShell doesn't automatically search your entire system for modules—it relies on specific paths defined by the `PSModulePath` environment variable. If a module isn't in one of these paths, PowerShell won't find it, even if the system/you installed it elsewhere.

To view the `PSModulePath` environment variable:

```
$env:PSModulePath
```

The variable contains a semicolon-separated list of directories.

For easier readability, split it into an array:

```
$env:PSModulePath -split ';'
```

PowerShell searches for modules in:

- User-level directories (e.g., your user profile's Documents folder).
- Shared directories in Program Files.

- System-level directories, such as *C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules*.

You can also add custom directories to the search path:

```
$env:PSModulePath + ';C:\MyNewModulePath'
$env:PSModulePath
```

This change applies to the current session, directing PowerShell to look for modules in the specified path.

## Conclusion

In this article, you explored how to discover available modules, view their versions, inspect their commands, and locate their storage directories. You also learned how to leverage the `PSModulePath` environment variable to customize where PowerShell looks for modules.

These foundational skills enable you to unlock the full potential of PowerShell for your automation tasks. Continue putting this knowledge into practice. Start by exploring the modules already available on your system and identifying those that suit your current needs.

With a deeper understanding of modules, you'll be ready to tackle complex automation challenges confidently and efficiently!