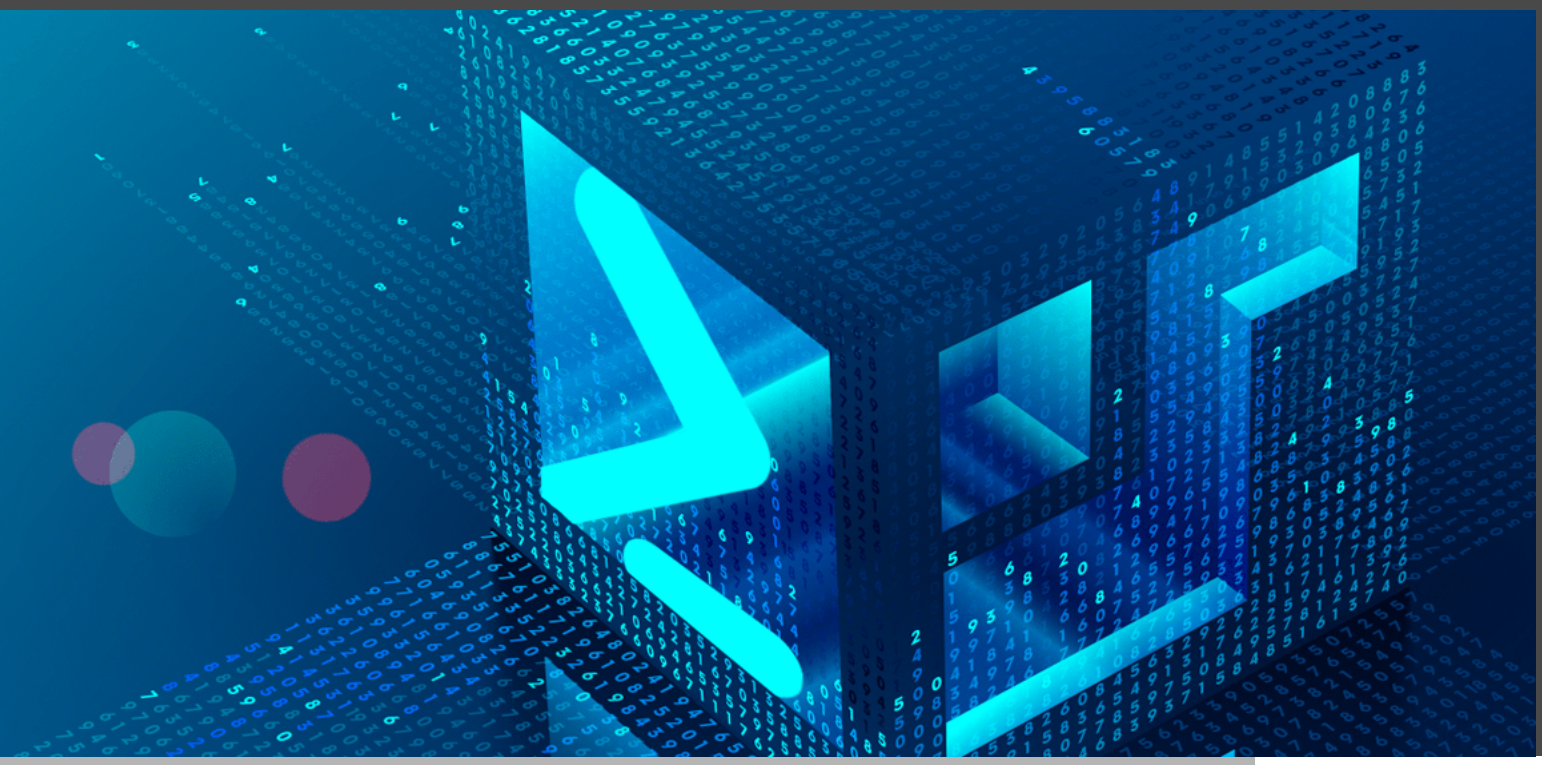
Devolutions

Local PowerShell Module Repository, No Server Required



COMMON CHALLENGE WHEN DEVELOPING POWERSHELL MODULES

Simple PowerShell scripts can go a long way on their own, but they can go even further with PowerShell modules. The largest PowerShell repository is PSGallery, where one can find modules to get the job done for just about anything. While published modules are relatively easy to install and import, that is not the case for local, unpublished modules. This is a common challenge when developing PowerShell modules, but there is hope!

Choosing a Reference PowerShell Module

We have chosen the Devolutions.Hub PowerShell module as a reference, but the same steps can be followed using a different module name and version. PowerShell 7 is required to load this cross-platform module, so Windows PowerShell 5.1 (built-in) cannot be used. Refer to the official PowerShell installation instructions, or use the following convenient one-liner on Windows:

```
iex «& { $(irm https://aka.ms/install-powershell.ps1) } -UseMSI -Quiet»
```

Once PowerShell 7 (pwsh.exe) is installed, to avoid issues ensure that it is always used instead of the older Windows PowerShell (powershell.exe).

Importing Modules by Name or Path

Published modules can be installed with the **Install-Module** command, after which they can be imported with the **Import-Module** command using the module name:

```
Install-Module -Name 'Devolutions.Hub'
Import-Module -Name 'Devolutions.Hub'
```

Modules can also be loaded using the path to the .psm1 file:

```
$ModuleBase = `
Get-Module -Name Devolutions.Hub -ListAvailable | `
Select-Object -First 1 -ExpandProperty ModuleBase
```

```
Import-Module «$ModuleBase\Devolutions.Hub.psm1»
```

Alternatively, modules can also be loaded using the path to the module directory if it has the same name as the .psm1 file it contains. Since the **Install-Module** command creates a subdirectory with the module version ("Devolutions.Hub\2021.1.0\Devolutions.Hub.psm1"), we need to make a temporary copy to obtain the desired structure ("Devolutions.Hub\Devolutions.Hub.psm1"):

```
$TempPath = [System.IO.Path]::GetTempPath()
$TempModulePath = Join-Path $TempPath «Devolutions.Hub»
Copy-Item $ModuleBase $TempModulePath -Recurse
Import-Module $TempModulePath
```

Once imported, there is no difference. Importing by name is similar to launching a program that was installed globally with an .msi installer, while importing by path is equivalent to launching an .exe using its full path. It is not that bad, but it can be annoying when scripts expect modules to be installed.

Registering a Local Repository

One solution to the module installation problem is to create a PowerShell repository on the local file system. Rather than importing modules by path, modules can be published and installed locally, without going through PSGallery, a NuGet server, or a network share. That's right: a little-known fact about the **Register-PSRepository** command is that it accepts local paths on the file system, and not just network file shares!

Create the "~/psrepo" directory, then call **Register-PSRepository** to create the 'local' repository:

```
$RepoPath = «~/psrepo»
New-Item -Path $RepoPath -ItemType 'Directory' -Force | Out-Null
```

```
Register-PSRepository -Name 'local' `
   -SourceLocation «$(Resolve-Path $RepoPath)» `
   -PublishLocation «$(Resolve-Path $RepoPath)» `
   -InstallationPolicy 'Trusted'
```

Verify that the new repository was created successfully with the **Get-PSRepository** command:

```
PS > Get-PSRepository
```

```
Name                InstallationPolicy   SourceLocation
----                ------------------   --------------
local               Trusted              C:\Users\wayk\psrepo
PSGallery           Untrusted            https://www.powershellgallery.com/api/v2
```

PSGallery is untrusted by default because its contents are not curated. We mark the local repository as trusted for the simple reason that it is local, and therefore not controlled by someone else.

Saving a PowerShell Module

To avoid the trouble of building a PowerShell module from source, let's grab one from PSGallery. The **Save-Module** command can be used to fetch a module and copy it to a local path without installing it. To demonstrate how this works, any module can be used as long as it is uninstalled prior to following the instructions. The module can be reinstalled cleanly afterward, so there is no need to worry.

```
$ModuleName = 'Devolutions.Hub'
$ModuleVersion = '2021.1.0'
New-Item -Path «~/modules» -ItemType 'Directory' -Force | Out-Null
Save-Module -Name $ModuleName -RequiredVersion $ModuleVersion `
  -Repository 'PSGallery' -Path «~/modules»
$ModulePath = «~/modules/${ModuleName}/${ModuleVersion}»
```

The above saves the Devolutions.Hub module from PSGallery to "~/modules/Devolutions.Hub/2021.1.0". This directory should contain the module manifest file (.psd1).

Loading Modules from PSModulePath

How exactly does PowerShell find modules when importing them by name? It uses PSModulePath, an environment variable containing a list of directories to look into. This is very similar to the PATH environment variable that controls which directories to search for system executables. In fact, PSModulePath uses the same separating character as PATH: ';' on Windows, ':' on non-Windows.

By adding "~/modules" to PSModulePath, we can make its modules available to PowerShell. To avoid interference by a previously installed module, ensure the sample module is correctly uninstalled:

```
Uninstall-Module -Name $ModuleName -AllVersions
Get-Module -Name $ModuleName -ListAvailable -Refresh
```

Next, temporarily modify PSModulePath, check that the module becomes available, and then revert PSModulePath to its original value:

```
$PSModulePath = $Env:PSModulePath
$Env:PSModulePath += «$([IO.Path]::PathSeparator)$(Resolve-Path «~/modules»)»
Get-Module -Name $ModuleName -ListAvailable -Refresh
$Env:PSModulePath = $PSModulePath
```

The **Get-Module** command should list the sample module inside the ~/modules directory. After restoring PSModulePath to its original value, the module should no longer be found. This is important, because the next goal is to install the module such that it can be loaded without modifying PSModulePath.

Publishing Module Locally

Installing modules can only be done from a repository, so let's publish the module to the local repository:

```
$ModulePackage = «${RepoPath}/${ModuleName}.${ModuleVersion}.nupkg»
Remove-Item -Path $ModulePackage -ErrorAction 'SilentlyContinue'
Publish-Module -Path $ModulePath -Repository 'local'
```

One limitation of using **Publish-Module** with a local directory is that the -Force parameter is not sufficient to overwrite an existing published module. The workaround is to simply delete the .nupkg package file when repeatedly publishing a module of the same version. This file is a NuGet package, which is essentially a .zip file containing the PowerShell module with additional metadata. The -Verbose parameter can be added to the **Publish-Module** command to learn more about the process used to produce the .nupkg file.

Installing Module Locally

Use the **Find-Command** to list the module available for installation from the 'local' repository:

```
PS > Find-Module -Name $ModuleName -Repository 'local' | select -Property Name,
Version
Name            Version
```

```
----            -------
Devolutions.Hub 2021.1.0
```

Finally, the **Install-Module** can be called to install the module from the 'local' repository:

```
Install-Module -Name $ModuleName -Repository 'local' -Scope CurrentUser
```

To make sure the module was correctly installed, use the **Get-InstalledModule** command:

```
PS > Get-InstalledModule -Name $ModuleName | select -Property Name, Version
```

```
Name            Version
----            -------
Devolutions.Hub 2021.1.0
```

Cleaning Up

To restore the system to its original state, uninstall the test module, unregister the 'local' repository, and delete the "~/psrepo" and "~/modules" directories:

```
Uninstall-Module -Name $ModuleName
Unregister-PSRepository -Name 'local'
Remove-Item «~/psrepo» -Force -Recurse -ErrorAction 'SilentlyContinue'
Remove-Item «~/modules» -Force -Recurse -ErrorAction 'SilentlyContinue'
```

Otherwise, keep the 'local' repository in place and start using it instead of a local NuGet server, or a repository hosted on a network share.

Closing Thoughts

If you found this blog post useful, please share it! We welcome your comments and would like to learn more about your experience with PowerShell module repositories. If you had a magic wand, what would you change about PowerShell module distribution? We want to know!