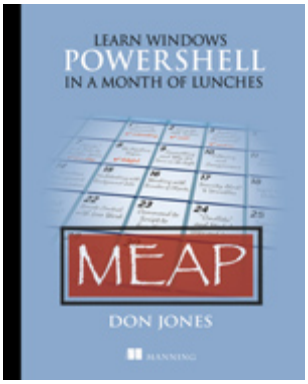
MANNING PUBLICATIONS



Learn Windows PowerShell in a Month of Lunches
By Don Jones

There are two kinds of extension for PowerShell v2: modules and snapins. In this article, based on chapter 5 of Learn Windows PowerShell in a Month of Lunches, author Don Jones explains how to find and add modules.

Extensions: Finding and Adding Modules

There are two kinds of extension for PowerShell v2: modules and snapins. Actually, the proper name is a *PSSnapin*, which distinguishes these from snapins for the graphical MMS. PSSnapins were first created for PowerShell v1. A PSSnapin generally consists of one or more DLL files, accompanied by additional XML files that contain configuration settings and help text. PSSnapins have to be installed and registered in order for PowerShell to know they exist.

The second type of extension supported by PowerShell v2 (and not available in v1) is a *module*. Modules are designed to be a little more self-contained and somewhat easier to distribute, but they work similarly to PSSnapins. You do need to know a bit more about them in order to find and use them.

Modules don't require advanced registration. Instead, PowerShell will automatically look in a certain set of paths to find modules. The environment variable PSModulePath defines the paths where modules are expected to live:

```
PS C:\> get-content env:psmodulepath
C:\Users\Administrator\Documents\WindowsPowerShell\Modules;C:\Windows
\system32\WindowsPowerShell\v1.0\Modules\
```

As you can see, there are two possible locations: one in the operating system folder, where system modules live, and one in My Documents folder, where any personal modules can be added. You can also add a module from any other location, provided you know its full path.

There are a couple of ways to see what modules are available. One is to simply get a directory listing of those two paths. I'll just do the system path:

```
PS C:\> dir C:\windows\System32\WindowsPowerShell\v1.0\Modules


    Directory: C:\windows\System32\WindowsPowerShell\v1.0\Modules


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d---s        11/21/2009   9:58 AM            ActiveDirectory
d----         7/13/2009  10:41 PM            ADRMS
d---s         7/13/2009  10:41 PM            AppLocker
d----         7/13/2009  10:41 PM            BestPractices
d---s         7/13/2009  10:41 PM            BitsTransfer
d----        11/21/2009  10:08 AM            GroupPolicy
d----         7/13/2009  10:37 PM            PSDiagnostics
d----         7/13/2009  10:41 PM            ServerManager
d----         7/13/2009  10:41 PM            TroubleshootingPack
d----        11/21/2009  10:02 AM            WebAdministration
```

Of course, this doesn't help you locate any modules that might be installed in other locations but, hopefully, if you install such a module, its documentation will help you figure out where it is. The list above shows the modules that come with Windows Server 2008 R2 or, at least, the modules installed on my server. Adding additional server roles or features may also add modules to support those roles and features, so it's worth checking this location anytime you've installed something new.

Another way to get a list of available modules is to use `Get-Module`:

```
PS C:\> get-module -listavailable

ModuleType Name                      ExportedCommands
---------- ----                      ----------------
Manifest   ActiveDirectory           {}
Manifest   ADRMS                     {}
Manifest   AppLocker                 {}
Manifest   BestPractices             {}
Manifest   BitsTransfer              {}
Manifest   GroupPolicy               {}
Manifest   PSDiagnostics             {}
Manifest   ServerManager             {}
Manifest   TroubleshootingPack       {}
Manifest   WebAdministration         {}
```

This list includes all modules installed in any path listed in the PSModulePath environment variable. In other words, these are the modules that the shell knows how to find. Any modules installed elsewhere won't be included on this list.

There are two ways to add a module, depending on whether or not the module is installed in one of the predefined paths. If the module is installed in one of those predefined paths, you use Import-Module and the module's name. You can then run `Get-Module`, with no parameters, to verify that the module loaded:

```
PS C:\> import-module activedirectory
PS C:\> get-module

ModuleType Name                      ExportedCommands
---------- ----                      ----------------
Manifest   activedirectory           {Set-ADOrganizationalUnit, Ge...
```

If the module is located elsewhere, then you would need to specify the complete path to the module, such as C:\MyPrograms\Something\MyModule, rather than just the module name. If you have a Start menu shortcut for a product-specific shell—say, SharePoint Server—and you don't know where that product installed its PowerShell module, just open the properties for the Start menu shortcut. The Target property of the shortcut will contain the `Import-Module` command used to load the module, and that will show you the module name and path.

Once a module is loaded, you can find out what commands it added by using `Get-Command` again:

```
PS C:\> gcm -module activedirectory

CommandType     Name                      Definition
-----------     ----                      ----------
Cmdlet          Add-ADComputerServiceAc... Add-ADComputerServiceAc...
Cmdlet          Add-ADDomainControllerP... Add-ADDomainControllerP...
Cmdlet          Add-ADFineGrainedPasswo... Add-ADFineGrainedPasswo...
Cmdlet          Add-ADGroupMember          Add-ADGroupMember [-Ide...
Cmdlet          Add-ADPrincipalGroupMem... Add-ADPrincipalGroupMem...
Cmdlet          Clear-ADAccountExpiration  Clear-ADAccountExpirati...
```

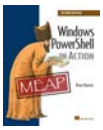
This time, I used the `-module` parameter to specify the module name, limiting the list of commands to just those that are included with the specified module.

Modules can also add PSDrive providers, and you would use the same technique as you did for PSSnapins to identify any new providers: Run `Get-PSProvider`.

Summary

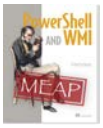
One of the primary strengths of PowerShell is its extensibility. As Microsoft continues to invest in PowerShell, they develop more and more commands for products like Exchange Server, SharePoint Server, the System Center family, SQL Server, and so on. Typically, installing the management tools for these products gives you both a graphical management console of some kind and one or more extensions for Windows PowerShell.

Here are some other Manning titles you might be interested in:



Windows PowerShell in Action, Second Edition
Bruce Payette

PowerShell in Practice
Richard Siddaway



PowerShell and WMI
Richard Siddaway

Last updated: February 10, 2011