5 Working with providers

One of the more potentially confusing aspects of PowerShell is its use of a *provider*. A provider gives access to specialized data stores for easier viewing and management. The data appears in a drive in PowerShell.

We warn you that some of this chapter might seem a bit remedial for you. We expect that you're familiar with the filesystem, for example, and you probably know all the commands you need to manage the filesystem from a shell. But bear with us: we're going to point things out in a specific way so that we can use your existing familiarity with the filesystem to help make the concept of providers easier to understand. Also, keep in mind that PowerShell isn't Bash. You may see some things in this chapter that look familiar, but we assure you that they're doing something quite different from what you're used to.

5.1 What are providers?

A PowerShell provider, or *PSProvider*, is an adapter. It's designed to take some kind of data storage, such as Windows Registry, Active Directory, or even the local filesystem, and make it look like a disk drive. You can see a list of installed PowerShell providers right within the shell:

```
PS C:\Scripts\ > Get-PSProvider
Name                 Capabilities                              Drives
----                 ------------                              ------
Alias                ShouldProcess                             {Alias}
Environment          ShouldProcess                             {Env}
FileSystem           Filter, ShouldProcess, Credentials        {/}
Function             ShouldProcess                             {Function}
Variable             ShouldProcess                             {Variable}
```

Providers can also be added into the shell, typically along with a module, which are the two ways that PowerShell can be extended. (We'll cover those extensions later in the book.) Sometimes, enabling certain PowerShell features may create a new PSProvider. For example, you can manipulate environment variables with the `Environment` provider, which we will cover in section 5.5 and you can see here:

```
PS C:\Scripts> Get-PSProvider
Name                 Capabilities                              Drives
----                 ------------                              ------
Alias                ShouldProcess                             {Alias}
Environment          ShouldProcess                             {Env}
FileSystem           Filter, ShouldProcess, Credentials        {/}
Function             ShouldProcess                             {Function}
Variable             ShouldProcess                             {Variable}
```

Notice that each provider has different capabilities. This is important, because it affects the ways in which you can use each provider. These are some of the common capabilities you'll see:

- `ShouldProcess`—The provider supports the use of the `-WhatIf` and `-Confirm` parameters, enabling you to "test" certain actions before committing to them.
- `Filter`—The provider supports the `-Filter` parameter on the cmdlets that manipulate providers' content.
- `Credentials`—The provider permits you to specify alternate credentials when connecting to data stores. There's a `-Credential` parameter for this.

You use a provider to create a *PSDrive*. A PSDrive uses a single provider to connect to data storage. You're creating a drive mapping, and thanks to the providers, a PSDrive is able to connect to much more than disks. Run the following command to see a list of currently connected drives:

```
PS C:\Scripts> Get-PSDrive

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
/                 159.55        306.11 FileSystem    /
Alias                                  Alias
Env                                    Environment
Function                               Function
Variable                               Variable
```

In the preceding list, you can see that we have one drive using the `FileSystem` provider, one using the `Env` provider, and so forth. The PSProvider adapts the data store, and the PSDrive makes it accessible. You use a set of cmdlets to see and manipulate the data exposed by each PSDrive. For the most part, the cmdlets you use with a PSDrive have the word `Item` somewhere in their noun:

```
PS C:\Scripts> Get-Command -Noun *item*
Capability      Name
----------      ----
Cmdlet          Clear-Item
Cmdlet          Clear-ItemProperty
Cmdlet          Copy-Item
Cmdlet          Copy-ItemProperty
Cmdlet          Get-ChildItem
Cmdlet          Get-Item
Cmdlet          Get-ItemProperty
Cmdlet          Invoke-Item
Cmdlet          Move-Item
Cmdlet          Move-ItemProperty
Cmdlet          New-Item
Cmdlet          New-ItemProperty
Cmdlet          Remove-Item
Cmdlet          Remove-ItemProperty
Cmdlet          Rename-Item
Cmdlet          Rename-ItemProperty
Cmdlet          Set-Item
Cmdlet          Set-ItemProperty
```

We'll use these cmdlets, and their aliases, to begin working with the providers on our system. Because it's probably the one you're most familiar with, we'll start with the filesystem—the `FileSystem` PSProvider.

5.2 Understanding how the filesystem is organized

The filesystem is organized around two main types of objects—folders and files. *Folders* are also a kind of container, capable of containing both files and other folders. *Files* aren't a type of container; they're more of an endpoint object.

You're probably most familiar with viewing the filesystem through Finder on macOS, the file browser on Linux, or Explorer on your Windows device (figure 5.1), where the hierarchy of drives, folders, and files is visually obvious.

PowerShell's terminology differs somewhat from that of the filesystem. Because a PSDrive might not point to a filesystem—for example, a PSDrive can be mapped to the Environment, Registry, or even an SCCM endpoint, which is obviously not a filesystem—PowerShell doesn't use the terms *file* and *folder*. Instead, it refers to these objects by the more generic term *item*. Both a file and a folder are considered items, although they're obviously different types of items. That's why the cmdlet names we showed you previously all use `Item` in their noun.

Items can, and often do, have properties. For example, a file item might have properties including its last write time, whether or not it's read-only, and so on. Some items, such as folders, can have *child items*, which are the items contained within that

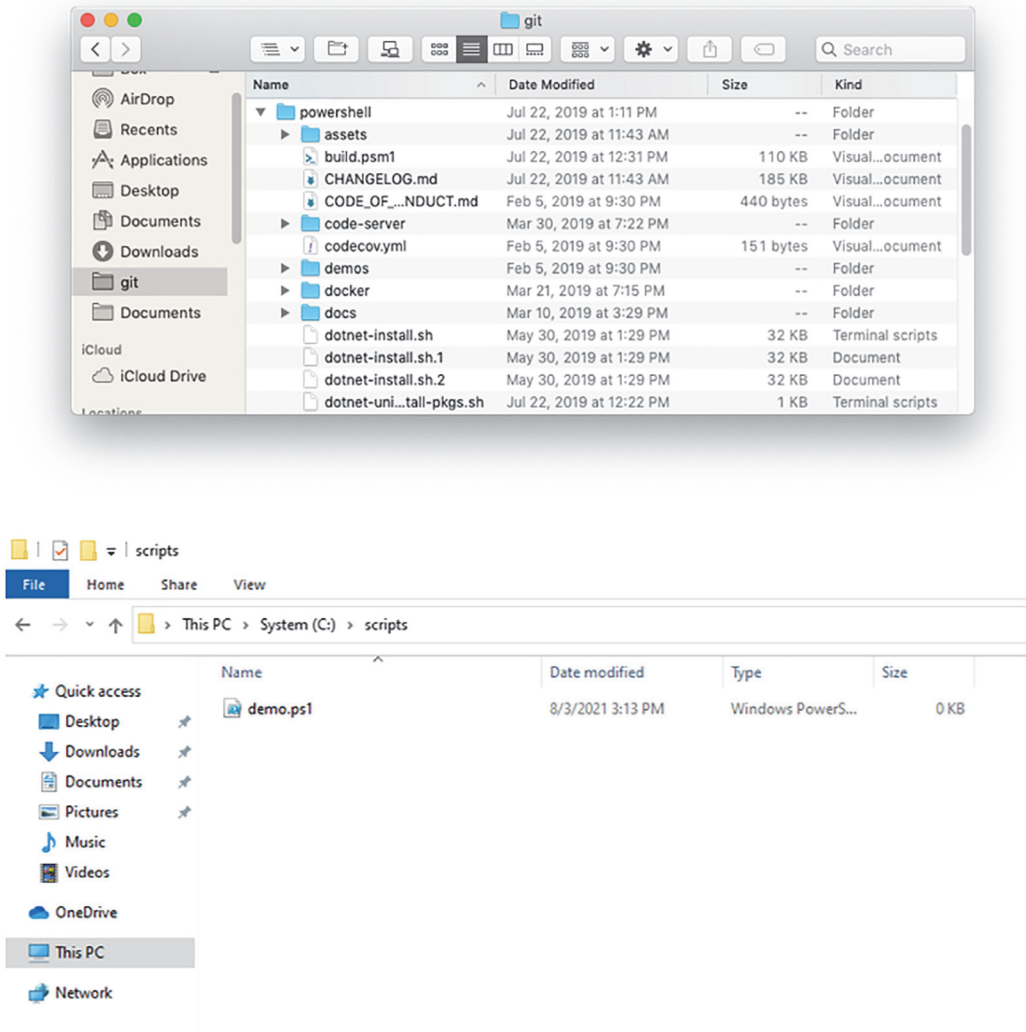

Figure 5.1 Viewing files, folders, and drives in Finder and Windows Explorer

item. Knowing those facts should help you make sense of the verbs and nouns in the command list we showed you earlier:

- Verbs such as `Clear`, `Copy`, `Get`, `Move`, `New`, `Remove`, `Rename`, and `Set` can all apply to items (e.g., files and folders) and to item properties (e.g., the date the item was last written or whether it's read-only).
- The `Item` noun refers to individual objects, such as files and folders.

- The `ItemProperty` noun refers to attributes of an item, such as read-only, creation time, length, and so on.
- The `ChildItem` noun refers to the items (e.g., files and subfolders) contained within an item (e.g., a folder).

Keep in mind that these cmdlets are intentionally generic, because they're meant to work with a variety of data stores. Some of the cmdlets' capabilities don't make sense in certain situations. As an example, because the `FileSystem` provider doesn't support the `Transactions` capability, none of the cmdlets' `-UseTransaction` parameters will work with items in the filesystem drives.

Some PSProviders don't support item properties. For example, the `Environment` PSProvider is what's used to make the `ENV:` drive available in PowerShell. This drive provides access to the environment variables, but as the following example shows, they don't have item properties:

```
PS C:\Scripts> Get-ItemProperty -Path Env:\PSModulePath
Get-ItemProperty : Cannot use interface. The IPropertyCmdletProvider
interface is not supported by this provider.
```

The fact that not every PSProvider is the same is perhaps what makes providers so confusing for PowerShell newcomers. You have to think about what each provider is giving you access to, and understand that even when the cmdlet knows how to do something, that doesn't mean the particular provider you're working with will support that operation.

5.3 Navigating the filesystem

Another cmdlet you need to know when working with providers is `Set-Location`. This is what you use to change the shell's current location to a different container-type item, such as a folder:

```
Linux / macOS
PS /Users/tplunk> Set-Location -Path /
PS />

Windows
PS C:\Scripts > Set-Location -Path /
PS />
```

You're probably more familiar with this command's alias, `cd`, which corresponds to the `Change Directory` command from Bash. Here we use the alias and pass the desired path as a positional parameter:

```
Linux / macOS
PS /Users/tplunk> cd /usr/bin
PS /usr/bin>

Windows
PS C:\Scripts\> cd C:\Users\tplunk
PS C:\Users\tplunk>
```

> **Drives on non-Windows operating systems**
> macOS and Linux don't use *drives* to refer to discrete attached storage devices. Instead, the entire operating system has a single root, represented by a slash (in PowerShell, a backslash is also accepted). But PowerShell still provides PSDrives in non-Windows operating systems for other providers. Try running `Get-PSDrive` to see what's available.

One of the trickier tasks in PowerShell is creating new items. For example, how do you create a new directory? Try running `New-Item` and you'll get an unexpected prompt:

```
PS C:\Users\tplunk\Documents> New-Item testFolder
Type:
```

Remember, the `New-Item` cmdlet is generic—it doesn't know you want to create a folder. It can create folders, files, and much more, but you have to tell it the type of item you want to create:

```
PS C:\Users\tplunk\Documents> New-Item testFolder -ItemType Directory


    Directory: C:\Users\tplunk\Documents

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          5/26/19 11:56 AM                testFolder
```

Windows PowerShell did include a `mkdir` command, which most people think is an alias for `New-Item`. But using `mkdir` doesn't require you to specify the directory `-ItemType`. Because of the conflict with the built-in `mkdir` command, the `mkdir` function was removed in PowerShell Core for non-Windows platforms.

5.4 Using wildcards and literal paths

Most providers allow you to specify paths in two ways using the `Item` cmdlets. This section will discuss these two ways of specifying paths. The `Item` cmdlets include the `-Path` parameter, and by default that parameter accepts wildcards. Looking at the full help for `Get-ChildItem`, for example, reveals the following:

```
-Path <String[]>
    Specifies a path to one or more locations. Wildcards are
      permitted. The default location is the current directory (.).
    Required?                    false
    Position?                    1
    Default value                Current directory
    Accept pipeline input?       true (ByValue, ByPropertyName)
    Accept wildcard characters?  True
```

The * wildcard stands in for zero or more characters, whereas the ? wildcard stands in for any single character. You've doubtless used this time and time again, probably with the Dir alias for Get-ChildItem:

```
PS C:\Scripts > dir y*


    Directory: C:\Scripts

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
--r---            5/4/19 12:03 AM          70192 yaa
--r---            5/4/19 12:02 AM          18288 yacc
--r---            5/4/19 12:03 AM          17808 yes
```

In Linux and macOS, most of these wildcards are allowed as part of the names of items in the filesystem as well as in most other stores. In the Environment, for example, you'll find a few values with names that include ?. This presents a problem: When you use * or ? in a path, is PowerShell supposed to treat it as a wildcard character or as a literal character? If you look for items named variable?, do you want the item with variable? as its name, or do you want ? treated as a wildcard, giving you items such as variable7 and variable8 instead?

PowerShell's solution is to provide an alternate -LiteralPath parameter. This parameter doesn't accept wildcards:

```
-LiteralPath <String[]>
    Specifies a path to one or more locations. Unlike the Path
    parameter, the value of the LiteralPath parameter is used exactly
    as it is typed. No characters are interpreted as wildcards. If
    the path includes escape characters, enclose it in single
    quotation marks. Single quotation marks tell PowerShell
    not to interpret any characters as escape sequences.
    Required?                    true
    Position?                    named
    Default value
    Accept pipeline input?       true (ByValue, ByPropertyName)
    Accept wildcard characters?  False
```

When you want * and ? taken literally, you use -LiteralPath instead of the -Path parameter. Note that -LiteralPath isn't positional; if you plan to use it, you have to type -LiteralPath. If you provide a path in the first position (such as y* in our first example), it'll be interpreted as being for the -Path parameter. Wildcards are also treated as such.

5.5 Working with other providers

One of the best ways to get a feel for these other providers, and how the various item cmdlets work, is to play with a PSDrive that isn't the filesystem. Of the providers built into PowerShell, the Environment is probably the best example to work with (in part because it's available on every system).

We will create an environment variable. Note that we are using an Ubuntu terminal for this exercise, but you can follow along just the same regardlessly if you are on a Windows or macOS machine (the wonders of cross-platform). Start by listing all environment variables:

```
PS /Users/tplunk> Get-ChildItem env:*

Name                           Value
----                           -----
XPC_FLAGS                      0x0
LANG                           en_US.UTF-8
TERM                           xterm-256color
HOME                           /Users/tplunk
USER                           tplunk
PSModulePath                   /Users/tplunk/.local/share/powershell/Modu…
HOMEBREW_EDITOR                code
PWD                            /Users/tplunk
COLORTERM                      truecolor
XPC_SERVICE_NAME               0
```

Next, set the environment variable `A` to the value `1`:

```
PS /Users/tplunk> Set-Item -Path Env:/A -Value 1

PS /Users/tplunk> Get-ChildItem Env:/A*

Name                           Value
----                           -----
A                              1
```

5.5.1 Windows Registry

Another provider we can look at on a Windows machine is the Registry. Let's start by changing to the `HKEY_CURRENT_USER` portion of the Registry, exposed by the `HKCU:` drive:

```
PS C:\> set-location -Path hkcu:
```

NOTE You may have to launch PowerShell as administrator.

Next, navigate to the right portion of the Registry:

```
PS HKCU:\> set-location -Path software
PS HKCU:\software> get-childitem
    Hive: HKEY_CURRENT_USER\software
Name                           Property
----                           --------
7-Zip                          Path64 : C:\Program Files\7-Zip\
                               Path   : C:\Program Files\7-Zip\
Adobe
Amazon
AppDataLow
```

```
AutomatedLab
BranchIO
ChangeTracker
Chromium
Clients

PS HKCU:\software> set-location microsoft
PS HKCU:\software\microsoft> Get-ChildItem
    Hive: HKEY_CURRENT_USER\software\microsoft
Name                           Property
----                           --------
Accessibility
Active Setup
ActiveMovie
ActiveSync
AppV
Assistance
AuthCookies
Avalon.Graphics
Clipboard                      ShellHotKeyUsed : 1
Common
CommsAPHost
ComPstUI
Connection Manager
CTF
Device Association Framework
DeviceDirectory                LastUserRegistrationTimestamp : {230, 198,
     218, 150…}
Edge                           UsageStatsInSample                 : 1
                               EdgeUwpDataRemoverResult           : 2
                               EdgeUwpDataRemoverResultDbh        : 1
                               EdgeUwpDataRemoverResultRoaming    : 0
                               EdgeUwpDataRemoverResultData       : 1
                               EdgeUwpDataRemoverResultBackupData : 1
EdgeUpdate                     LastLogonTime-Machine : 132798161806442449
EdgeWebView                    UsageStatsInSample : 1
EventSystem
Exchange
F12
Fax
```

You're almost finished. You'll notice that we're sticking with full cmdlet names rather than using aliases to emphasize the cmdlets themselves:

```
PS HKCU:\software\microsoft> Set-Location .\Windows
PS HKCU:\software\microsoft\Windows> Get-ChildItem
    Hive: HKEY_CURRENT_USER\software\microsoft\Windows
Name                           Property
----                           --------
AssignedAccessConfiguration
CurrentVersion
```

```
DWM                             Composition                  : 1
                                ColorPrevalence              : 0
                                ColorizationColor            : 3288334336
                                ColorizationColorBalance     : 89
                                ColorizationAfterglow        : 3288334336
                                ColorizationAfterglowBalance : 10
                                ColorizationBlurBalance      : 1
                                EnableWindowColorization     : 0
                                ColorizationGlassAttribute   : 1
                                AccentColor                  : 4278190080
                                EnableAeroPeek               : 1
Shell
TabletPC
Windows Error Reporting         LastRateLimitedDumpGenerationTime :
➥ 132809598562003780
Winlogon
```

Note the `EnableAeroPeek` Registry value. Let's change it to `0`:

```
PS HKCU:\software\microsoft\Windows> Set-ItemProperty -Path dwm -PSProperty
EnableAeroPeek -Value 0
```

You also could have used the `-Name` parameter instead of `-PSProperty`. Let's check it again to make sure the change "took":

```
PS HKCU:\software\microsoft\Windows> Get-ChildItem
    Hive: HKEY_CURRENT_USER\software\microsoft\Windows
Name                            Property
----                            --------
AssignedAccessConfiguration
CurrentVersion
DWM                             Composition                  : 1
                                ColorPrevalence              : 0
                                ColorizationColor            : 3288334336
                                ColorizationColorBalance     : 89
                                ColorizationAfterglow        : 3288334336
                                ColorizationAfterglowBalance : 10
                                ColorizationBlurBalance      : 1
                                EnableWindowColorization     : 0
                                ColorizationGlassAttribute   : 1
                                AccentColor                  : 4278190080
                                EnableAeroPeek               : 0
Shell
TabletPC
Windows Error Reporting         LastRateLimitedDumpGenerationTime :
➥ 132809598562003780
Winlogon
```

Mission accomplished! Using these same techniques, you should be able to work with any provider that comes your way.

5.6 Lab

NOTE For this lab, you need any computer running PowerShell v7.1 or later.

Complete the following tasks from a PowerShell prompt:

1. Create a new directory called Labs.
2. Create a zero-length file named /Labs/Test.txt (use `New-Item`).
3. Is it possible to use `Set-Item` to change the contents of /Labs/Test.txt to `-TESTING`? Or do you get an error? If you get an error, why?
4. Using the Environment provider, display the value of the system environment variable `PATH`.
5. Use help to determine what the differences are between the `-Filter`, `-Include`, and `-Exclude` parameters of `Get-ChildItem`.

5.7 Lab answers

1. `New-Item -Path ~/Labs -ItemType Directory`
2. `New-Item -Path ~/labs -Name test.txt -ItemType file`
3. The `FileSystem` provider doesn't support this action.
4. Either of these commands works:

```
Get-Item env:PATH
Dir env:PATH
```

5. `-Include` and `-Exclude` must be used with `-Recurse` or if you're querying a container. `Filter` uses the PSProvider's filter capability, which not all providers support. For example, you could use `DIR -filter` in the filesystem.

Above and beyond

Did you run into any issues with task 4? PowerShell on a Windows machine is case insensitive, meaning uppercase and lowercase letters don't matter. `PATH` is the same as `path`. However, in a Linux or macOS machine, capitalization matters: `PATH` is not the same as `path`.