

1

Installing and Configuring PowerShell 7

This chapter covers the following recipes:

- ▶ Installing PowerShell 7
- ▶ Using the PowerShell 7 console
- ▶ Exploring PowerShell 7 installation artifacts
- ▶ Building PowerShell 7 profile files
- ▶ Installing VS Code
- ▶ Installing the Cascadia Code font
- ▶ Exploring PSReadLine

Introduction

PowerShell 7 represents the latest step in the development of PowerShell. PowerShell, first introduced to the public in 2003, was released formally as Windows PowerShell v1 in 2006. Over the next decade, Microsoft released multiple versions, ending with PowerShell 5.1. During the development of Windows PowerShell, the product moved from being an add-in to Windows to an integrated feature. Microsoft plans to support Windows PowerShell 5.1 for a long time, but no new features are likely.

The PowerShell development team began working on an open-source version of PowerShell based on the open-source version of .NET Core. The first three versions, PowerShell Core 6.0, 6.1, and 6.2, represented a proof of concept – you could run the core functions and features of PowerShell across the Windows, Mac, and Linux platforms. But they were quite limited in terms of supporting the rich needs of the IT pro community.

With the release of PowerShell 7.0 came improved parity with Windows PowerShell. There were a few modules that did not work with PowerShell 7, and a few more that work via a compatibility mechanism described in *Chapter 3, Exploring Compatibility with Windows PowerShell*. PowerShell 7.0 shipped in 2019 and has been followed by version 7.1. This book uses the term "PowerShell 7" to include both PowerShell 7.0 and 7.1.

Once you have installed PowerShell 7, you can run it and use it just as you used the Windows PowerShell console. The command you run to start PowerShell 7 is now `pwsh.exe` (versus `powershell.exe` for Windows PowerShell). PowerShell 7 also uses different profile file locations from Windows PowerShell. You can customize your PowerShell 7 profiles to make use of the new PowerShell 7 features. You can also use different profile files for Windows PowerShell and PowerShell 7.

The Windows PowerShell **Integrated Scripting Environment (ISE)** is a tool you use with PowerShell. The ISE, however, is not supported with PowerShell 7. To replace it, use **Visual Studio Code (VS Code)**, an open-source editing project that provides all the features of the ISE and a great deal more.

Microsoft also developed a new font, Cascadia Code, to coincide with the launch of VS Code. This font is a nice improvement over Courier or other mono-width fonts. All screenshots of working code in this book use this new font.

PSReadLine is a PowerShell module designed to provide color-coding of PowerShell scripts in the PowerShell 7 console. The module, included with PowerShell 7 by default, makes editing at the command line easier and more on par with the features available in Linux shells.

Installing PowerShell 7

PowerShell 7 is not installed in Windows by default, at least not at the time of writing. The PowerShell team has made PowerShell 7.1 available from the Microsoft Store, which is useful to install PowerShell 7.1 or later on Windows 10 systems. As the Microsoft Store is not overly relevant to Windows Server installations, you can install PowerShell by using a script, `Install-PowerShell.ps1`, which you download from the internet, as shown in this recipe. You can also use this recipe on Windows 10 hosts.

Getting ready

This recipe uses SRV1, a Windows Server workgroup host. There are no features of applications loaded on this server (yet).

How to do it...

1. Setting an execution policy for Windows PowerShell (in a new Windows PowerShell console)

```
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Force
```

2. Installing the latest versions of NuGet and PowerShellGet

```
Install-PackageProvider NuGet -MinimumVersion 2.8.5.201 -Force |  
Out-Null  
Install-Module -Name PowerShellGet -Force -AllowClobber
```

3. Ensuring the C:\Foo folder exists

```
$LFHT = @{  
    ItemType      = 'Directory'  
    ErrorAction = 'SilentlyContinue' # should it already exist  
}  
New-Item -Path C:\Foo @LFHT | Out-Null
```

4. Downloading the PowerShell 7 installation script

```
Set-Location C:\Foo  
$URI = 'https://aka.ms/install-powershell.ps1'  
Invoke-RestMethod -Uri $URI |  
    Out-File -FilePath C:\Foo\Install-PowerShell.ps1
```

5. Viewing the installation script help

```
Get-Help -Name C:\Foo\Install-PowerShell.ps1
```

6. Installing PowerShell 7

```
$EXTHT = @{  
    UseMSI          = $true  
    Quiet           = $true  
    AddExplorerContextMenu = $true  
    EnablePSRemoting = $true  
}  
C:\Foo\Install-PowerShell.ps1 @EXTHT | Out-Null
```

7. For the Adventurous – installing the preview and daily builds as well

```
C:\Foo\Install-PowerShell.ps1 -Preview -Destination C:\PWSHPreview |
Out-Null
C:\Foo\Install-PowerShell.ps1 -Daily -Destination C:\PWSHDailBuild |
Out-Null
```
8. Creating Windows PowerShell default profiles

```
$URI = 'https://raw.githubusercontent.com/doctordns/Wiley20/master/' +
'Goodies/Microsoft.PowerShell_Profile.ps1'
$ProfileFile = $Profile.CurrentUserCurrentHost
New-Item $ProfileFile -Force -WarningAction SilentlyContinue |
Out-Null
(Invoke-WebRequest -Uri $uri -UseBasicParsing).Content |
Out-File -FilePath $ProfileFile
$ProfilePath = Split-Path -Path $ProfileFile
$ConsoleProfile = Join-Path -Path $ProfilePath -ChildPath 'Microsoft.
PowerShell_profile.ps1'
(Invoke-WebRequest -Uri $URI -UseBasicParsing).Content |
Out-File -FilePath $ConsoleProfile
```
9. Checking the versions of PowerShell 7 loaded

```
Get-ChildItem -Path C:\pwsh.exe -Recurse -ErrorAction SilentlyContinue
```

How it works...

In *step 1*, you open a new Windows PowerShell console and set the PowerShell execution policy to unrestricted, which simplifies using scripts to configure hosts. In production, you may wish to set PowerShell's execution policy to be more restrictive. But note that an execution policy is not truly a security mechanism – it just slows down an inexperienced administrator.

For a good explanation of PowerShell's Security Guiding Principles, see <https://devblogs.microsoft.com/powershell/powershells-security-guiding-principles/>.

The PowerShell Gallery is a repository of PowerShell modules and scripts and is an essential resource for the IT pro. This book makes use of several modules from the PowerShell Gallery. In *step 2*, you update both the NuGet package provider (to version 2.8.5.201 or later) and an updated version of the PowerShellGet module.

Throughout this book, you'll use the `C:\Foo` folder to hold various files that you use in conjunction with the recipes. In *step 3*, you ensure the folder exists.

PowerShell 7 is not installed by default, at present, in Windows (or macOS or Linux), although this could change. To enable you to install PowerShell 7 in Windows, you retrieve an installation script from GitHub and store that in the `C:\Foo` folder. In *step 4*, you use a shortcut URL that points to GitHub and then use `Invoke-RestMethod` to download the file. Note that *steps 1* through *4* produce no output.

In *step 5*, you view the help information contained in the help file, which produces the following output:

```
PS C:\Foo> # 5. Viewing the installation script help
PS C:\Foo> Get-Help -Name C:\Foo\Install-PowerShell.ps1
Install-PowerShell.ps1 [-Destination <string>] [-Daily] [-DoNotOverwrite] [-AddToPath] [-Preview] [<CommonParameters>]
Install-PowerShell.ps1 [-UseMSI] [-Quiet] [-AddExplorerContextMenu] [-EnablePSRemoting] [-Preview] [<CommonParameters>]
```

Figure 1.1: Viewing installation script help

Note that after installing PowerShell 7, the first time you run `Get-Help` you are prompted to download help text (which is not shown in this figure).

In *step 6*, you use the installation script and install PowerShell 7. The commands use an MSI, which you then install silently. The MSI updates the system execution path to add the PowerShell 7 installation folder. The code retrieves the latest supported version of PowerShell 7, and you can view the actual filename in the following output:

```
PS C:\Foo> # 6. Installing PowerShell 7
PS C:\Foo> $EXTHT = @{
    UseMSI           = $true
    Quiet            = $true
    AddExplorerContextMenu = $true
    EnablePSRemoting = $true
}
PS C:\Foo> C:\Foo\Install-PowerShell.ps1 @EXTHT | Out-Null
VERBOSE: About to download package from
'https://github.com/PowerShell/PowerShell/releases/download/v7.1.0/PowerShell-7.1.0-win-x64.msi'
```

Figure 1.2: Installing PowerShell 7

PowerShell 7 is a work in progress. Every day, the PowerShell team builds updated versions of PowerShell and releases previews of the next major release. The preview builds are mostly stable and allow you to try out new features that are coming in the next major release. The daily build allows you to view progress on a specific bug or feature. You may find it useful to install both of these (and ensure you keep them up to date as time goes by).

In step 7, you install the daily build and the latest preview build, which looks like this:

```
PS C:\Foo> # 7. For the Adventurous -installing the preview and daily builds as well
PS C:\Foo> C:\Foo\Install-PowerShell.ps1 -Preview -Destination c:\PWSHPreview |
Out-Null
VERBOSE: Destination: C:\PWSHPreview
VERBOSE: About to download package from 'https://github.com/PowerShell/PowerShell/releases/
download/v7.1.0-preview.6/PowerShell-7.1.0-preview.6-win-x64.zip'
PowerShell has been installed at C:\PWSHPreview
PS C:\Foo> C:\Foo\Install-PowerShell.ps1 -Daily -Destination c:\PWSHDailBuild |
Out-Null
VERBOSE: Destination: C:\PWSHDailBuild
VERBOSE: About to download package from 'https://pscoretestdata.blob.core.windows.net/v7-1-
0-daily-20200909/PowerShell-7.1.0-daily.20200909-win-x64.zip'
PowerShell has been installed at C:\PWSHDailBuild
```

Figure 1.3: Installing the preview and daily builds

PowerShell uses **profile files** to enable you to configure PowerShell each time you run it (whether in the PowerShell console or as part of VS Code or the ISE). In step 8, you download a sample PowerShell profile script and save it locally. Note that the profile file you create in step 8 is for Windows PowerShell only.

The executable name for PowerShell 7 is `pwsh.exe`. In step 9, you view the versions of this file as follows:

```
PS C:\Foo> # 9. Checking versions of PowerShell 7 loaded
PS C:\Foo> Get-ChildItem -Path C:\pwsh.exe -Recurse -ErrorAction SilentlyContinue

Directory: C:\Program Files\PowerShell\7
Mode                LastWriteTime         Length Name
----                -
-a----           06/11/2020    02:33         280456 pwsh.exe

Directory: C:\PSDailyBuild
Mode                LastWriteTime         Length Name
----                -
-a----           13/11/2020    01:08         269824 pwsh.exe

Directory: C:\PSPreview
Mode                LastWriteTime         Length Name
----                -
-a----           13/11/2020    20:03         274304 pwsh.exe
```

Figure 1.4: Checking PowerShell 7 versions loaded

As you can see, there are three versions of PowerShell 7 installed on SRV1: the latest full release, the latest preview, and the build of the day.

There's more...

In *step 1*, you open a new Windows PowerShell console. Make sure you run the console as the local administrator.

In *step 4*, you use a shortened URL to download the `Install-PowerShell.ps1` script. When you use `Invoke-RestMethod`, PowerShell discovers the underlying target URL for the script. The short URL allows Microsoft and the PowerShell team to publish a well-known URL and then have the flexibility to move the target location should that be necessary. The target URL, at the time of writing, is <https://raw.githubusercontent.com/PowerShell/PowerShell/master/tools/install-powershell.ps1>.

In *step 7*, you install both the latest daily build and the latest preview versions. The specific file versions you see are going to be different from the output shown here, at least for the preview versions!

Using the PowerShell 7 console

With PowerShell 7, the name of the PowerShell executable is now `pwsh.exe`, as you saw in the previous recipe. After installing PowerShell 7 in Windows, you can start the PowerShell 7 console by clicking **Start** and typing `pwsh.exe`, then hitting *Return*. The PowerShell MSI installer does not create a Start panel or taskbar shortcut.

Getting ready

You run this recipe on SRV1 after you have installed PowerShell 7.

How to do it...

1. Running the PowerShell 7 console

From the Windows desktop in SRV1, click on the Windows key, then type `pwsh`, followed by the *Enter* key.

2. Viewing the PowerShell version

\$PSVersionTable

3. Viewing the \$Host variable
`$Host`
4. Looking at the PowerShell process
`Get-Process -Id $Pid |
Format-Custom MainModule -Depth 1`
5. Looking at resource usage statistics
`Get-Process -Id $Pid |
Format-List CPU,*Memory*`
6. Updating the PowerShell help
`$Before = Get-Help -Name about_*
Update-Help -Force | Out-Null
$After = Get-Help -Name about_*
$Delta = $After.Count - $Before.Count
"{0} Conceptual Help Files Added" -f $Delta`
7. How many commands are available?
`Get-Command |
Group-Object -Property CommandType`

How it works...

In step 1, you start the PowerShell 7 console on SRV1. The console should look like this:

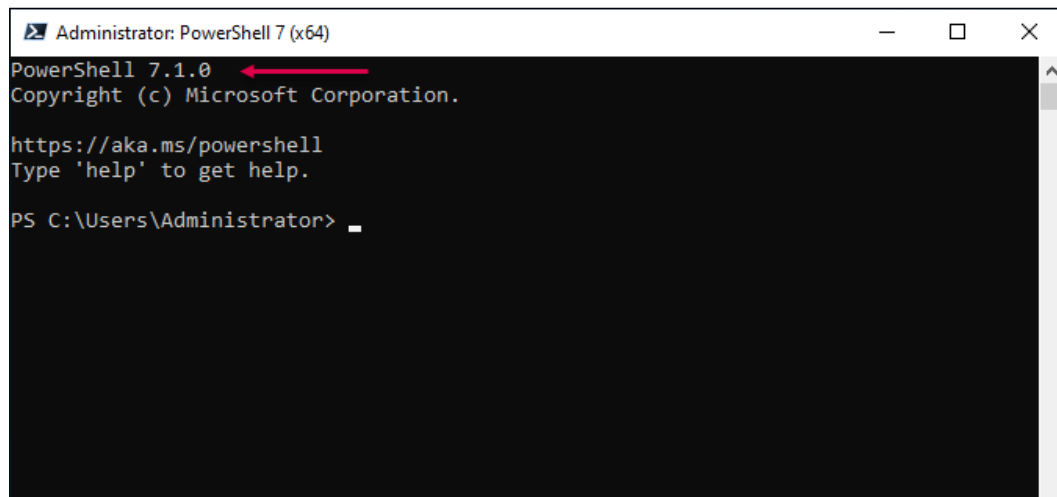


Figure 1.5: The PowerShell 7 console

In step 2, you view the specific version of PowerShell by looking at the built-in variable `$PSVersionTable`, which looks like this:

```
PS C:\Foo> # 2. Viewing the PowerShell version
PS C:\Foo> $PSVersionTable
```

Name	Value
PSVersion	7.1.0
PSEdition	Core
GitCommitId	7.1.0
OS	Microsoft Windows 10.0.20270
Platform	Win32NT
PSCompatibleVersions	{1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion	2.3
SerializationVersion	1.1.0.1
WSManStackVersion	3.0

Figure 1.6: Viewing the PowerShell version

In step 3, you examine the `$Host` variable to determine details about the PowerShell 7 host (the PowerShell console), which looks like this:

```
PS C:\Foo> # 3. Viewing the $Host variable
PS C:\Foo> $Host
```

Name	: ConsoleHost
Version	: 7.1.0
InstanceId	: ea087cdd-ddbe-43ef-b2eb-4c6cb8d8c541
UI	: System.Management.Automation.Internal.Host.InternalHostUserInterface
CurrentCulture	: en-GB
CurrentUICulture	: en-US
PrivateData	: Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
DebuggerEnabled	: True
IsRunspacePushed	: False
Runspace	: System.Management.Automation.Runspaces.LocalRunspace

Figure 1.7: Viewing the `$Host` variable

As you can see, in this case, the current culture is EN-GB. You may see a different value depending on which specific version of Windows Server you are using.

In step 4, you use `Get-Process` to look at the details of the PowerShell process, which looks like this:

```
PS C:\Foo> # 4. Looking at the PowerShell process
PS C:\Foo> Get-Process -Id $PID |
    Format-Custom -Property MainModule -Depth 1

class Process
{
    MainModule =
        class ProcessModule
        {
            ModuleName = pwsh.exe
            FileName = C:\Program Files\PowerShell\7\pwsh.exe
            BaseAddress = 140695338418176
            ModuleMemorySize = 290816
            EntryPointAddress = 140695338483552
            FileVersionInfo = File: C:\Program Files\PowerShell\7\pwsh.exe
            InternalName: pwsh.dll
            OriginalFilename: pwsh.dll
            FileVersion: 7.1.0.0
            FileDescription: pwsh
            Product: PowerShell
            ProductVersion: 7.1.0 SHA: d2953dcaf8323b95371380639ced00dac4ed209f
            Debug: False
            Patched: False
            PreRelease: False
            PrivateBuild: False
            SpecialBuild: False
            Language: Language Neutral

            Site =
            Container =
            Size = 284
            Company = Microsoft Corporation
            FileVersion = 7.1.0.0
            ProductVersion = 7.1.0 SHA: d2953dcaf8323b95371380639ced00dac4ed209f
            Description = pwsh
            Product = PowerShell
        }
}
```

Figure 1.8: Looking at the PowerShell process details

In this figure, you can see the path to the PowerShell 7 executable. This value changes depending on whether you are running the release version or the daily/preview releases.

You can see, in step 5, details of resource usage of the `pwsh.exe` process running on the `SRV1` host:

```
PS C:\Foo> # 5. Looking at resource usage statistics
PS C:\Foo> Get-Process -Id $PID |
    Format-List CPU,*Memory*

CPU                                     : 4.765625
NonpagedSystemMemorySize64           : 74552
NonpagedSystemMemorySize             : 74552
PagedMemorySize64                   : 63938560
PagedMemorySize                     : 63938560
PagedSystemMemorySize64             : 443552
PagedSystemMemorySize               : 443552
PeakPagedMemorySize64               : 72388608
PeakPagedMemorySize                 : 72388608
PeakVirtualMemorySize64             : 2204211732480
PeakVirtualMemorySize               : 893509632
PrivateMemorySize64                 : 63938560
PrivateMemorySize                   : 63938560
VirtualMemorySize64                 : 2204191739904
VirtualMemorySize                   : 873517056
```

Figure 1.9: Looking at the resource usage statistics of `pwsh.exe`

The values of each of the performance counters are likely to vary, and you may see different values.

By default, PowerShell 7, like Windows PowerShell, ships with minimum help files. You can, as you can see in step 6, use the `Update-Help` command to download updated PowerShell 7 help content, like this:

```
PS C:\Users\Administrator> # 6. Updating the PowerShell help
PS C:\Users\Administrator> $Before = Get-Help -Name about_*
PS C:\Users\Administrator> Update-Help -Force | Out-Null
Update-Help: Failed to update Help for the module(s) 'PSReadline' with UI culture(s) {en-US} :
One or more errors occurred. (Response status code does not indicate success: 404
(The specified blob does not exist.)).
English-US help content is available and can be installed using: Update-Help -UICulture en-US.
PS C:\Users\Administrator> $After = Get-Help -Name about_*
PS C:\Users\Administrator> $Delta = $After.Count - $Before.Count
PS C:\Users\Administrator> $Delta = $After.Count - $Before.Count
PS C:\Users\Administrator> "{0} Conceptual Help Files Added" -f $Delta
128 Conceptual Help Files Added
```

Figure 1.10: Updating PowerShell help

As you can see from the output, not all help files were updated. In this case, the ConfigDefender module does not, at present, have updated help information. Also note that although the UK English versions of help details may be missing, there are US English versions that you can install that may be useful. There is no material difference between the UK and US texts.

Commands in PowerShell include functions, cmdlets, and aliases. In *step 7*, you examine how many of each type of command is available by default, like this:

```
PS C:\Foo> # 7. How many commands are available?
PS C:\Foo> Get-Command |
             Group-Object -Property CommandType
```

Count	Name	Group
58	Alias	{Add-AppPackage, Add-AppPackageVolume, Add-AppProvisionedPackage, Add-ProvisionedAppPa...
1144	Function	{A:, Add-BCDataCacheExtension, Add-DnsClientDohServerAddress, Add-DnsClientNrptRule, A...
587	Cmdlet	{Add-AppxPackage, Add-AppxProvisionedPackage, Add-AppxVolume, Add-BitsFile, Add-Certif...

Figure 1.11: Examining the number of each type of command available

There's more...

In *step 1*, you open the PowerShell console for the version of PowerShell you installed in the *Installing PowerShell 7* recipe. With the release of PowerShell 7.1, the version number you would see is 7.1.0. By the time you read this book, that version number may have advanced. To ensure you have the latest released version of PowerShell 7, re-run the `Install-PowerShell.ps1` script you downloaded in the *Installing PowerShell 7* recipe.

Also, in *step 1*, you can see the output generated by the `Write-Host` statements in the profile file you set up in *Installing PowerShell 7*. You can remove these statements to reduce the amount of output you see each time you start up PowerShell.

In *step 4*, you use the variable `$PID`, which contains the Windows process identifier of the PowerShell 7 console process. The actual value of `$PID` changes each time you run PowerShell, but the value always contains the process ID of the current console process.

In *step 6*, you can see an error in the `PSReadLine` module's help information (*Figure 1.10*). The developers of this module have, in later versions, renamed this module to `PSReadLine` (capitalizing the L in Line). However, help URLs are, for some reason, case-sensitive. You can fix this by renaming the module on disk and capitalizing the folder name.

In *step 7*, you saw that you had 1786 commands available. This number changes as you add more features (and their accompanying modules) or download and install modules from repositories such as the PowerShell Gallery.

Exploring PowerShell 7 installation artifacts

In PowerShell 7, certain objects added by the PowerShell 7 installer (and PowerShell 7) differ from those used by Windows PowerShell.

Getting ready

This recipe uses SRV1 after you have installed PowerShell 7. In this recipe, you use the PowerShell 7 console to run the steps.

How to do it...

1. Checking the version table for the PowerShell 7 console

```
$PSVersionTable
```

2. Examining the PowerShell 7 installation folder

```
Get-Childitem -Path $env:ProgramFiles\PowerShell\7 -Recurse |  
Measure-Object -Property Length -Sum
```

3. Viewing the PowerShell 7 configuration JSON file

```
Get-ChildItem -Path $env:ProgramFiles\PowerShell\7\powershell*.json |  
Get-Content
```

4. Checking the initial Execution Policy for PowerShell 7

```
Get-ExecutionPolicy
```

5. Viewing the module folders

```
$I = 0  
$ModPath = $env:PSModulePath -split ';'   
$ModPath |  
Foreach-Object {  
    "[{0:N0}]    {1}" -f $I++, $_  
}
```

6. Checking the modules

```
$TotalCommands = 0
Foreach ($Path in $ModPath){
    Try { $Modules = Get-ChildItem -Path $Path -Directory -ErrorAction Stop
        "Checking Module Path: [$Path]"
    }
    Catch [System.Management.Automation.ItemNotFoundException] {
        "Module path [$path] DOES NOT EXIST ON $(hostname)"
    }
    $CmdsInPath = 0
    Foreach ($Module in $Modules) {
        $Cmds = Get-Command -Module ($Module.name)
        $TotalCommands += $Cmds.Count
    }
}
```

7. Viewing the total number of commands and modules

```
$Mods = (Get-Module * -ListAvailable | Measure-Object).count
"{0} modules providing {1} commands" -f $Mods,$TotalCommands
```

How it works...

In step 1, you examine the `$PSVersionTable` variable to view the version information for PowerShell 7, which looks like this:

```
PS C:\Users\Administrator> # 1. Checking the version table for PowerShell 7 console
PS C:\Users\Administrator> $PSVersionTable
```

Name	Value
PSVersion	7.1.0
PSEdition	Core
GitCommitId	7.1.0
OS	Microsoft Windows 10.0.20270
Platform	Win32NT
PSCompatibleVersions	{1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion	2.3
SerializationVersion	1.1.0.1
WSManStackVersion	3.0

Figure 1.12: Checking the version information for PowerShell 7

The PowerShell 7 installation program installs PowerShell 7 into a different folder (by default) from that used by Windows PowerShell. In step 2, you see a summary of the files installed into the PowerShell 7 installation folder as follows:

```
PS C:\Users\Administrator> # 2. Examining the PowerShell 7 installation folder
PS C:\Users\Administrator> Get-Childitem -Path $env:ProgramFiles\PowerShell\7 -Recurse |
    Measure-Object -Property Length -Sum

Count          : 982
Average        :
Sum            : 252056323
Maximum        :
Minimum        :
StandardDeviation :
Property       : Length
```

Figure 1.13: Examining the installation folder

PowerShell 7 stores configuration values in a JSON file in the PowerShell 7 installation folder. In step 3, you view the contents of this file:

```
PS C:\Users\Administrator> # 3. Viewing PowerShell configuration JSON file
PS C:\Users\Administrator> Get-ChildItem -Path $env:ProgramFiles\PowerShell\7\powershell*.json |
    Get-Content

{
  "WindowsPowerShellCompatibilityModuleDenyList": [
    "PSScheduledJob",
    "BestPractices",
    "UpdateServices"
  ],
  "Microsoft.PowerShell:ExecutionPolicy": "RemoteSigned"
}
```

Figure 1.14: Viewing the JSON configuration file

In step 4, you view the execution policy for PowerShell 7, as follows:

```
PS C:\Users\Administrator> # 4. Checking initial Execution Policy for PowerShell 7
PS C:\Users\Administrator> Get-ExecutionPolicy
RemoteSigned
```

Figure 1.15: Checking the execution policy for PowerShell 7

As with Windows PowerShell, PowerShell 7 loads commands from modules. PowerShell uses the `$PSModulePath` variable to determine which file store folders PowerShell 7 should use to find these modules. Viewing the contents of this variable, and discovering the folders, in step 5, looks like this:

```
PS C:\Users\Administrator> # 5. Viewing module folders
PS C:\Users\Administrator> $I = 0
PS C:\Users\Administrator> $ModPath = $env:PSModulePath -split ';'
PS C:\Users\Administrator> $ModPath |
    Foreach-Object {
        "[{0:N0}]  {1}" -f $I++, $_
    }

[0] C:\Users\Administrator\Documents\PowerShell\Modules
[1] C:\Program Files\PowerShell\Modules
[2] c:\program files\powershell\7\Modules
[3] C:\Program Files\WindowsPowerShell\Modules
[4] C:\Windows\system32\WindowsPowerShell\v1.0\Modules
```

Figure 1.16: Viewing the module folders

With those module folders defined (by default), you can check how many commands exist in each folder, in step 6, the output of which looks like this:

```
PS C:\Users\Administrator> # 6. Checking the modules
PS C:\Users\Administrator> $TotalCommands = 0
PS C:\Users\Administrator> Foreach ($Path in $ModPath){
    Try { $Modules = Get-ChildItem -Path $Path -Directory -ErrorAction Stop
        "Checking Module Path:  [$Path]"
    }
    Catch [System.Management.Automation.ItemNotFoundException] {
        "Module path [$path] DOES NOT EXIST ON $(hostname)"
    }
    $CmdsInPath = 0
    Foreach ($Module in $Modules) {
        $Cmds = Get-Command -Module ($Module.name)
        $TotalCommands += $Cmds.Count
    }
}

Module path [C:\Users\Administrator\Documents\PowerShell\Modules] DOES NOT EXIST ON SRV1
Module path [C:\Program Files\PowerShell\Modules] DOES NOT EXIST ON SRV1
Checking Module Path:  [c:\program files\powershell\7\Modules]
Checking Module Path:  [C:\Program Files\WindowsPowerShell\Modules]
Checking Module Path:  [C:\Windows\system32\WindowsPowerShell\v1.0\Modules]
```

Figure 1.17: Checking how many commands exist in each module folder

In *step 7*, you can view the results to see how many commands exist in each of the modules in each module path. The output looks like this:

```
PS C:\Users\Administrator> # 7. Viewing totals of commands and modules
PS C:\Users\Administrator> $Mods = (Get-Module * -ListAvailable | Measure-Object).count
PS C:\Users\Administrator> "{0} modules providing {1} commands" -f $Mods,$TotalCommands
72 modules providing 4930 commands
```

Figure 1.18: Viewing the total number of modules and commands

There's more...

In *step 1*, you viewed the PowerShell version table. Depending on when you read this book, the version numbers you see may be later than shown here. You would see this when the PowerShell team releases an updated version of PowerShell.

In *step 4*, you viewed PowerShell 7's execution policy. Each time PowerShell 7 starts up, it reads the JSON file to obtain the value of the execution policy. You can use the `Set-ExecutionPolicy` to reset the policy immediately, or change the value in the JSON file and restart the PowerShell 7 console.

In *step 5*, you viewed the default folders that PowerShell 7 uses to search for a module (by default). The first folder is your personal modules, followed by PowerShell 7, and then the Windows PowerShell modules. We cover the Windows PowerShell modules and Windows PowerShell compatibility in more detail in *Chapter 2, Introducing PowerShell 7*.

Building PowerShell 7 profile files

In Windows PowerShell and PowerShell 7, profile files are PowerShell scripts that PowerShell runs each time you start a new instance of PowerShell (whether the console, ISE, or VS Code). These files enable you to pre-configure PowerShell 7. You can add variables, PowerShell `PSDrives`, functions, and more using profiles. As part of this book, you download and install initial profile files based on samples that you can download from GitHub.

This recipe downloads and installs the profile files for the PowerShell 7 console.

Getting ready

You run this recipe on SRV1 after you have installed PowerShell 7.

How to do it...

1. Discovering the profile filenames

```
$ProfileFiles = $PROFILE | Get-Member -MemberType NoteProperty  
$ProfileFiles | Format-Table -Property Name, Definition
```
2. Checking for the existence of each PowerShell profile file

```
ForEach ($ProfileFile in $ProfileFiles){  
    "Testing $($ProfileFile.Name)"  
    $ProfilePath = $ProfileFile.Definition.split('=')[1]  
    If (Test-Path $ProfilePath){  
        "$($ProfileFile.Name) DOES EXIST"  
        "At $ProfilePath"  
    }  
    Else {  
        "$($ProfileFile.Name) DOES NOT EXIST"  
    }  
    ""  
}
```
3. Displaying the Current User/Current Host profile

```
$CUCHProfile = $PROFILE.CurrentUserCurrentHost  
"Current User/Current Host profile path: [$CUCHPROFILE]"
```
4. Creating a Current User/Current Host profile for the PowerShell 7 console

```
$URI = 'https://raw.githubusercontent.com/doctordns/PACKT-PS7/master/' +  
    'scripts/goodies/Microsoft.PowerShell_Profile.ps1'  
New-Item $CUCHProfile -Force -WarningAction SilentlyContinue |  
    Out-Null  
(Invoke-WebRequest -Uri $URI).Content |  
    Out-File -FilePath $CUCHProfile
```
5. Exiting the PowerShell 7 console

```
Exit
```
6. Restarting the PowerShell 7 console and viewing the profile output at startup

```
Get-ChildItem -Path $Profile
```

How it works...

In *step 1*, you discover the names of each of the four profile files (for the PowerShell 7 console), then view their name and location (that is, the definition), which looks like this:

```
PS C:\Users\Administrator> # 1. Discovering the profile file names
PS C:\Users\Administrator> $ProfileFiles = $profile | Get-Member -MemberType NoteProperty
PS C:\Users\Administrator> $ProfileFiles | Format-Table -Property Name, Definition
```

Name	Definition
AllUsersAllHosts	string AllUsersAllHosts=C:\Program Files\PowerShell\7\profile.ps1
AllUsersCurrentHost	string AllUsersCurrentHost=C:\Program Files\PowerShell\7\Microsoft.PowerShell_profile.ps1
CurrentUserAllHosts	string CurrentUserAllHosts=C:\Users\Administrator\Documents\PowerShell\profile.ps1
CurrentUserCurrentHost	string CurrentUserCurrentHost=C:\Users\Administrator\Documents\PowerShell\Microsoft.PowerShell_...

Figure 1.19: Discovering the profile filenames

In *step 2*, you check to see which, if any, of the profile files exist, which looks like this:

```
PS C:\Users\Administrator> # 2. Checking for existence of each PowerShell profile file
PS C:\Users\Administrator> Foreach ($ProfileFile in $ProfileFiles){
    "Testing $($ProfileFile.Name)"
    $ProfilePath = $ProfileFile.Definition.split('=')[1]
    If (Test-Path $ProfilePath){
        "$($ProfileFile.Name) DOES EXIST"
        "At $ProfilePath"
    }
    Else {
        "$($ProfileFile.Name) DOES NOT EXIST"
    }
    ""
}
```

```
Testing AllUsersAllHosts
AllUsersAllHosts DOES NOT EXIST

Testing AllUsersCurrentHost
AllUsersCurrentHost DOES NOT EXIST

Testing CurrentUserAllHosts
CurrentUserAllHosts DOES NOT EXIST

Testing CurrentUserCurrentHost
CurrentUserCurrentHost DOES NOT EXIST
```

Figure 1.20: Checking for the existence of PowerShell profile files

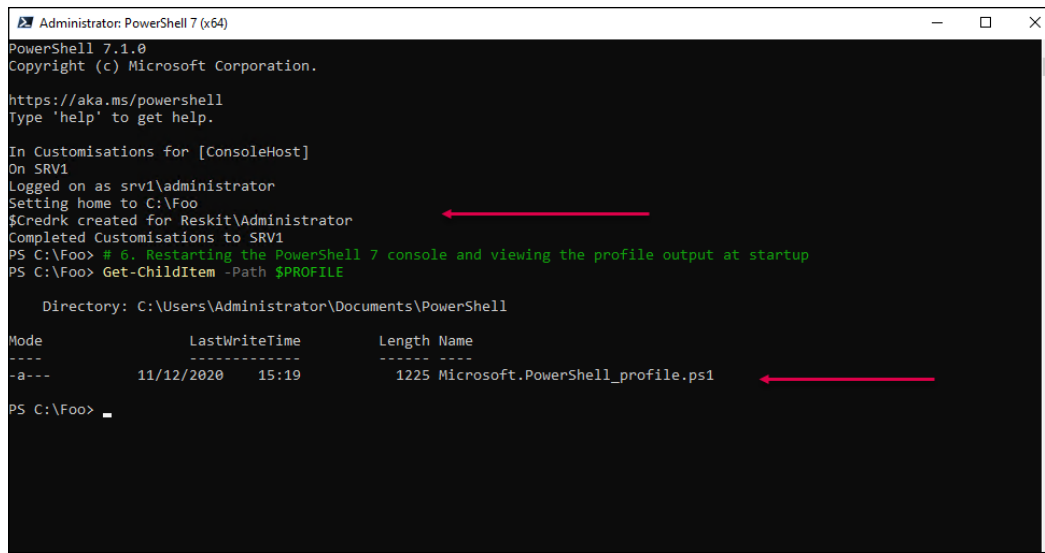
In *step 3*, you obtain and display the filename of the Current User/Current Host profile file, which looks like this:

```
PS C:\Users\Administrator> # 3. Discovering Current User/Current Host profile
PS C:\Users\Administrator> $CUCHProfile = $PROFILE.CurrentUserCurrentHost
PS C:\Users\Administrator> "Current User/Current Host profile path: [$CUCHPROFILE]"
Current User/Current Host profile path: [C:\Users\Administrator\Documents\PowerShell\Microsoft.PowerShell_profile.ps1]
```

Figure 1.21: Discovering the Current User/Current Host profile file

In *step 4*, you create an initial Current User/Current Host profile file. This file is part of the GitHub repository that supports this book. In *step 5*, you exit the current PowerShell 7 console host. These two steps create no output.

In *step 6*, you start a new PowerShell profile. This time, as you can see here, the profile file exists, runs, and customizes the console:



```
Administrator: PowerShell 7 (x64)
PowerShell 7.1.0
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

In Customisations for [ConsoleHost]
On SRV1
Logged on as srv1\administrator
Setting home to C:\Foo
$Credrk created for ResKit\Administrator
Completed Customisations to SRV1
PS C:\Foo> # 6. Restarting the PowerShell 7 console and viewing the profile output at startup
PS C:\Foo> Get-ChildItem -Path $PROFILE

Directory: C:\Users\Administrator\Documents\PowerShell

Mode                LastWriteTime         Length Name
----                -
-a---          11/12/2020   15:19           1225 Microsoft.PowerShell_profile.ps1
```

Figure 1.22: The profile file in action

There's more...

In *step 4*, you download a sample profile file from GitHub. This sample profile file contains customizations for the PowerShell console configuration, including changing the default starting folder (to C:\Foo) and creating some aliases, PowerShell drives, and a credential object. These represent sample content you might consider including in your console profile file. Note that VS Code, which you install in the next recipe, uses a separate Current User/Current Host profile file, which enables you to customize PowerShell at the console and in VS Code differently.

Installing VS Code

The Windows PowerShell ISE was a great tool that Microsoft first introduced with Windows PowerShell v2 (and vastly improved with v3). This tool has reached feature completeness, and Microsoft has no plans for further development.

In its place, however, is Visual Studio Code, or VS Code. This open-source tool provides an extensive range of features for IT pros and others. For IT professionals, this should be your editor of choice. While there is a learning curve (as for any new product), VS Code contains all the features you find in the ISE and more.

VS Code, and the available extensions, are works in progress. Each new release brings additional features which can be highly valuable. A recent addition from Draw.io, for example, is the ability to create diagrams directly in VS Code. Take a look at this post for more details on this diagram tool: <https://tf109.blogspot.com/2020/05/over-weekend-i-saw-tweet-announcing-new.html>.

There are a large number of other extensions you might be able to use, depending on your workload. For more details on VS Code, see <https://code.visualstudio.com/>.

For details of the many VS Code extensions, you can visit <https://code.visualstudio.com/docs/editor/extension-marketplace>.

Getting ready

You run this recipe on SRV1 after you have installed PowerShell 7 and have created a console profile file.

How to do it...

1. Downloading the VS Code installation script from the PowerShell Gallery

```
$VSCPATH = 'C:\Foo'
Save-Script -Name Install-VSCode -Path $VSCPATH
Set-Location -Path $VSCPATH
```

2. Running the installation script and adding in some popular extensions

```
$Extensions = 'Streetsidesoftware.code-spell-checker',
              'yzhang.markdown-all-in-one',
              'hediet.vscode-drawio'

$InstallHT = @{
    BuildEdition      = 'Stable-System'
    AdditionalExtensions = $Extensions
    LaunchWhenDone    = $true
}
.\Install-VSCode.ps1 @InstallHT
```

3. Exiting VS Code

Close the VS Code window

4. Restarting VS Code as an administrator

Click on the Windows key and type **code**, which brings up the VS Code tile in the Windows Start panel. Then, right click the VS Code tile and select **Run as Administrator** to start VS Code as an administrator.

5. Opening a new VS Code terminal window

Inside VS Code, type *Ctrl + Shift + `*.

6. Creating a Current User/Current Host profile for VS Code

```
$SAMPLE =  
  'https://raw.githubusercontent.com/doctordns/PACKT-PS7/master/' +  
  'scripts/goodies/Microsoft.VSCode_profile.ps1'  
(Invoke-WebRequest -Uri $Sample).Content |  
  Out-File $Profile
```

7. Updating the user settings for VS Code

```
$JSON = @'  
{  
  "workbench.colorTheme": "PowerShell ISE",  
  "powershell.codeFormatting.useCorrectCasing": true,  
  "files.autoSave": "onWindowChange",  
  "files.defaultLanguage": "powershell",  
  "editor.fontFamily": "'Cascadia Code', Consolas, 'Courier New'",  
  "workbench.editor.highlightModifiedTabs": true,  
  "window.zoomLevel": 1  
}  
'@  
$JHT = ConvertFrom-Json -InputObject $JSON -AsHashtable  
$PWSH = "C:\\Program Files\\PowerShell\\7\\pwsh.exe"  
$JHT += @{  
  "terminal.integrated.shell.windows" = "$PWSH"  
}  
$Path = $Env:APPDATA  
$CP = '\\Code\\User\\Settings.json'  
$Settings = Join-Path $Path -ChildPath $CP  
$JHT |  
  ConvertTo-Json |  
  Out-File -FilePath $Settings
```

8. Creating a shortcut to VS Code

```

$SourceFileLocation = "$env:ProgramFiles\Microsoft VS Code\Code.exe"
$ShortcutLocation   = "C:\foo\vscode.lnk"
# Create a new wscript.shell object
$WScriptShell       = New-Object -ComObject WScript.Shell
$Shortcut            = $WScriptShell.CreateShortcut($ShortcutLocation)
$Shortcut.TargetPath = $SourceFileLocation
#Save the Shortcut to the TargetPath
$Shortcut.Save()

```

9. Creating a shortcut to PowerShell 7

```

$SourceFileLocation = "$env:ProgramFiles\PowerShell\7\pwsh.exe"
$ShortcutLocation   = 'C:\Foo\pwsh.lnk'
# Create a new wscript.shell object
$WScriptShell       = New-Object -ComObject WScript.Shell
$Shortcut            = $WScriptShell.CreateShortcut($ShortcutLocation)
$Shortcut.TargetPath = $SourceFileLocation
#Save the Shortcut to the TargetPath
$Shortcut.Save()

```

10. Building an updated layout XML

```

$XML = @'
<?xml version="1.0" encoding="utf-8"?>
<LayoutModificationTemplate
  xmlns="http://schemas.microsoft.com/Start/2014/LayoutModification"
  xmlns:defaultlayout=
    "http://schemas.microsoft.com/Start/2014/FullDefaultLayout"
  xmlns:start="http://schemas.microsoft.com/Start/2014/StartLayout"
  xmlns:taskbar="http://schemas.microsoft.com/Start/2014/TaskbarLayout"
  Version="1">
<CustomTaskbarLayoutCollection>
<defaultlayout:TaskbarLayout>
<taskbar:TaskbarPinList>
  <taskbar:DesktopApp DesktopApplicationLinkPath="C:\Foo\vscode.lnk" />
  <taskbar:DesktopApp DesktopApplicationLinkPath="C:\Foo\pwsh.lnk" />
</taskbar:TaskbarPinList>
</defaultlayout:TaskbarLayout>

```

```

</CustomTaskbarLayoutCollection>
</LayoutModificationTemplate>
'@
$XML | Out-File -FilePath C:\Foo\Layout.Xml

```

11. Importing the start layout XML file

```
Import-StartLayout -LayoutPath C:\Foo\Layout.Xml -MountPath C:\
```

12. Logging of

```
logoff.exe
```

13. Log back in to Windows and observe the taskbar

14. Run the PowerShell console from the shortcut

15. Run VS Code from the new taskbar shortcut and observe the profile file running

How it works...

In *step 1*, you download the VS Code installation script from the PowerShell Gallery. This step produces no output.

Then, in *step 2*, you run the installation script and add in three specific extensions. Running this step in the PowerShell 7 console looks like this:

```

PS C:\Foo> # 2. Running the installation script and adding in some popular extensions
PS C:\Foo> $Extensions = 'Streetsidesoftware.code-spell-checker',
                        'yzhang.markdown-all-in-one',
                        'hediet.vscode-drawio'

PS C:\Foo> $InstallHT = @{
    BuildEdition      = 'Stable-System'
    AdditionalExtensions = $Extensions
    LaunchWhenDone    = $true
}

PS C:\Foo> .\Install-VSCode.ps1 @InstallHT | Out-Null

Installing extension ms-vscode.PowerShell...
Installing extensions...
Installing extension 'ms-vscode.powershell' v2020.6.0...

Installing extension Streetsidesoftware.code-spell-checker...

Installing extension yzhang.markdown-all-in-one...

Installing extension hediet.vscode-drawio...

Installation complete, starting Visual Studio Code (64-bit)...

```

Figure 1.23: Running the installation script and adding some extensions

Once VS Studio has started, you will see the initial opening window, which looks like this:

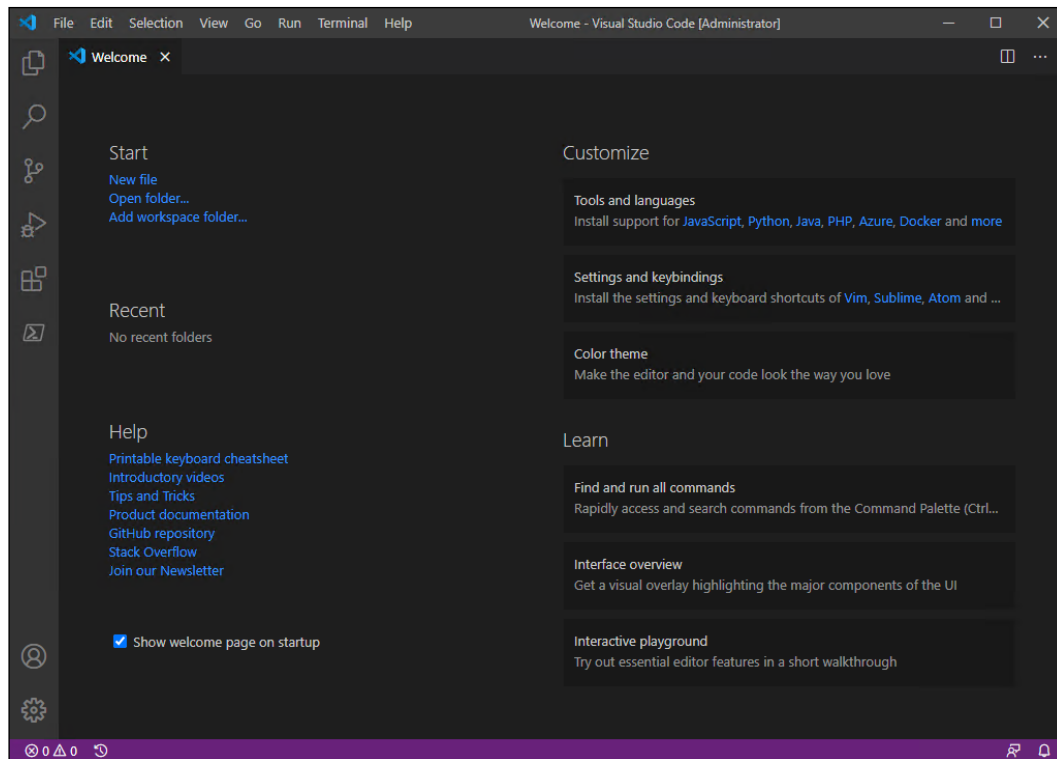


Figure 1.24: The VS Code Welcome window

Installing and Configuring PowerShell 7

In step 3, you close this window. Then, in step 4, you run VS Code as an administrator. In step 5, you open a new VS Code Terminal and run PowerShell 7, which now looks like this:

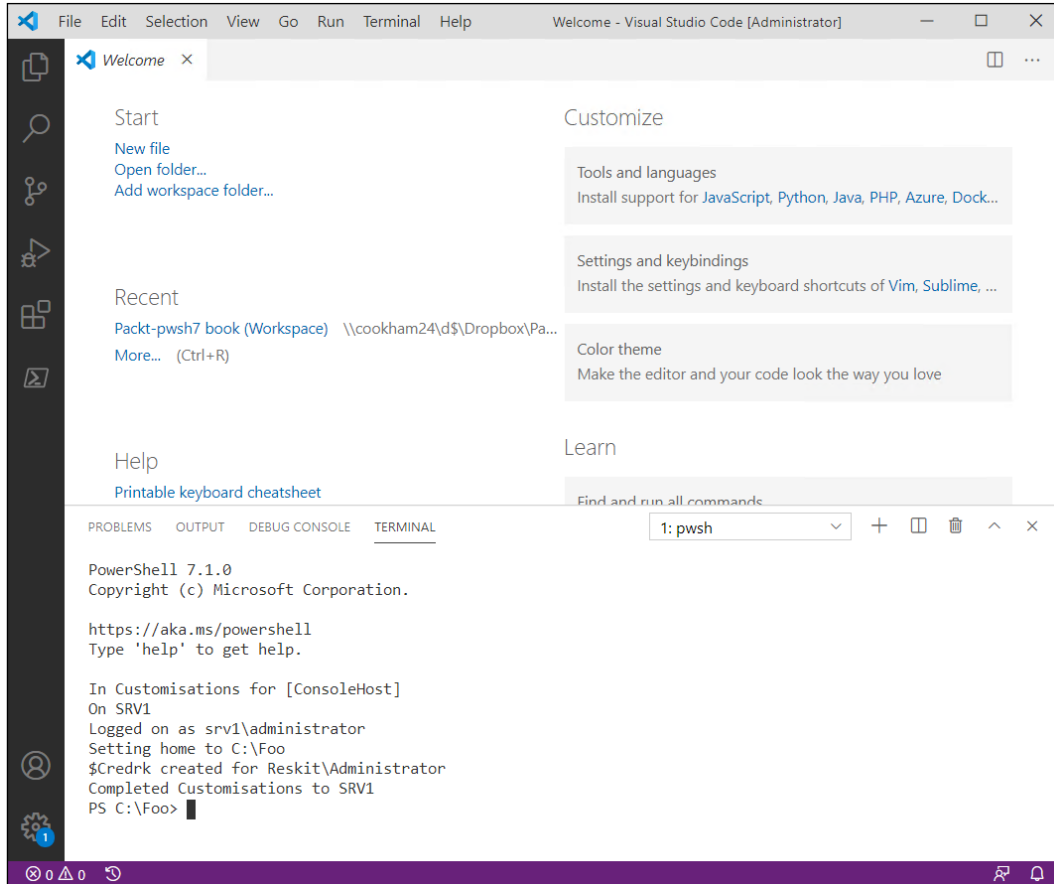


Figure 1.25: Running PowerShell 7 in VS Code

In *step 6*, you create a VS Code sample profile file. The VS Code PowerShell extension uses this profile file when you open a .PS1 file. This step generates no output.

In *step 7*, you update several VS Code runtime options. In *step 8*, you create a shortcut to VS Code, and in *step 9*, you create a shortcut to the PowerShell 7 console. You use these later in this recipe to create a shortcut on your Windows taskbar.

In *step 10*, you update the XML that describes the Windows taskbar to add the shortcuts to VS Code and the PowerShell console you created previously. In *step 11*, you import the updated task pane description back into Windows. These steps produce no output as such.

Next, in *step 12*, you log off from Windows. In *step 13*, you re-login and note the updated taskbar, as you can see here:

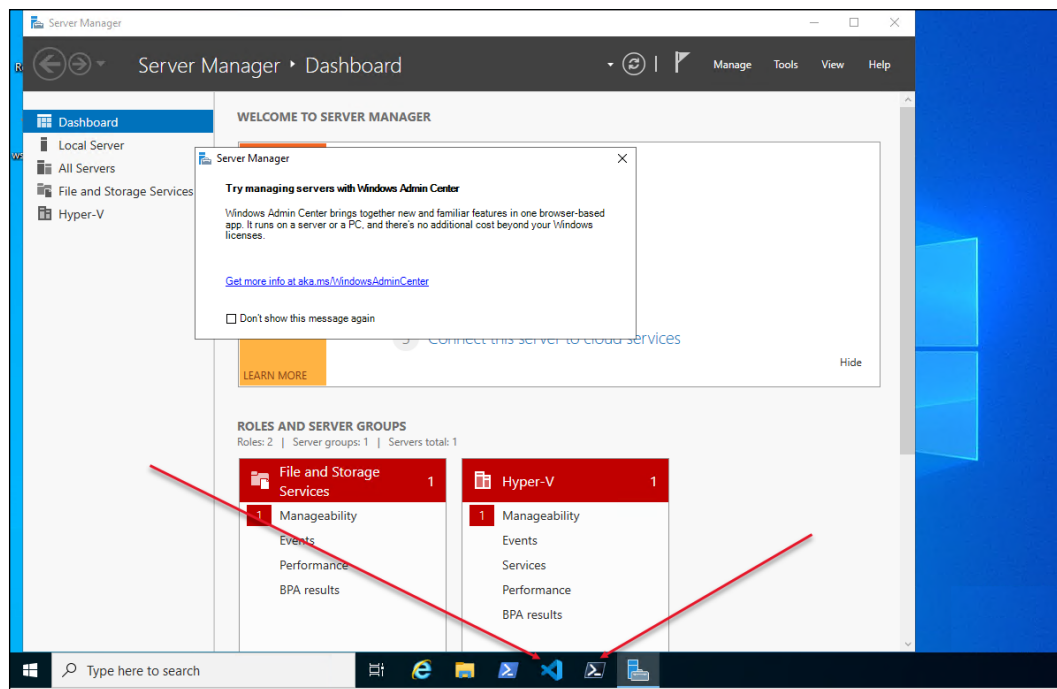
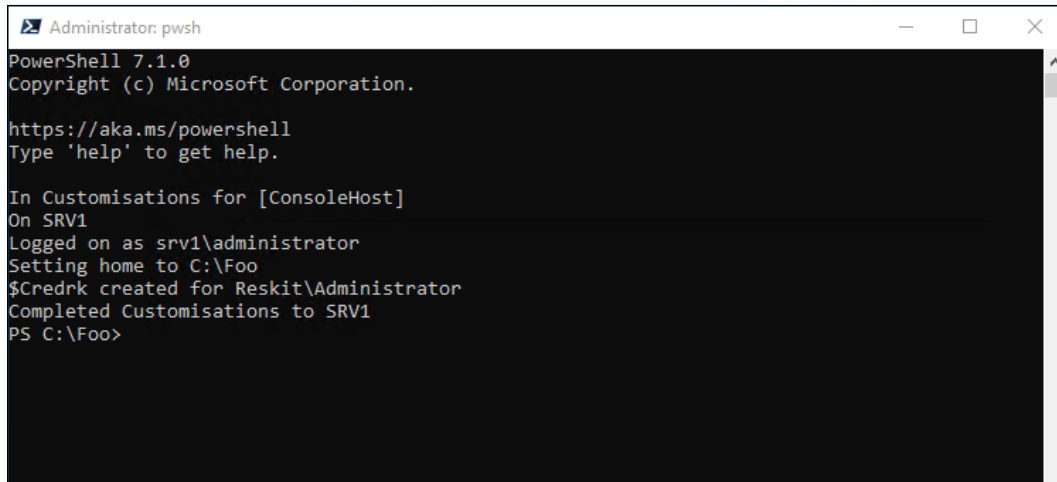


Figure 1.26: Updated taskbar with shortcuts

In step 14, you open a PowerShell 7 console, which looks like this:



```
Administrator: pwsh
PowerShell 7.1.0
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

In Customisations for [ConsoleHost]
On SRV1
Logged on as srv1\administrator
Setting home to C:\Foo
$Credrk created for Reskit\Administrator
Completed Customisations to SRV1
PS C:\Foo>
```

Figure 1.27: PowerShell 7 console (from shortcut)

In step 15, you open VS Code, which looks like this:

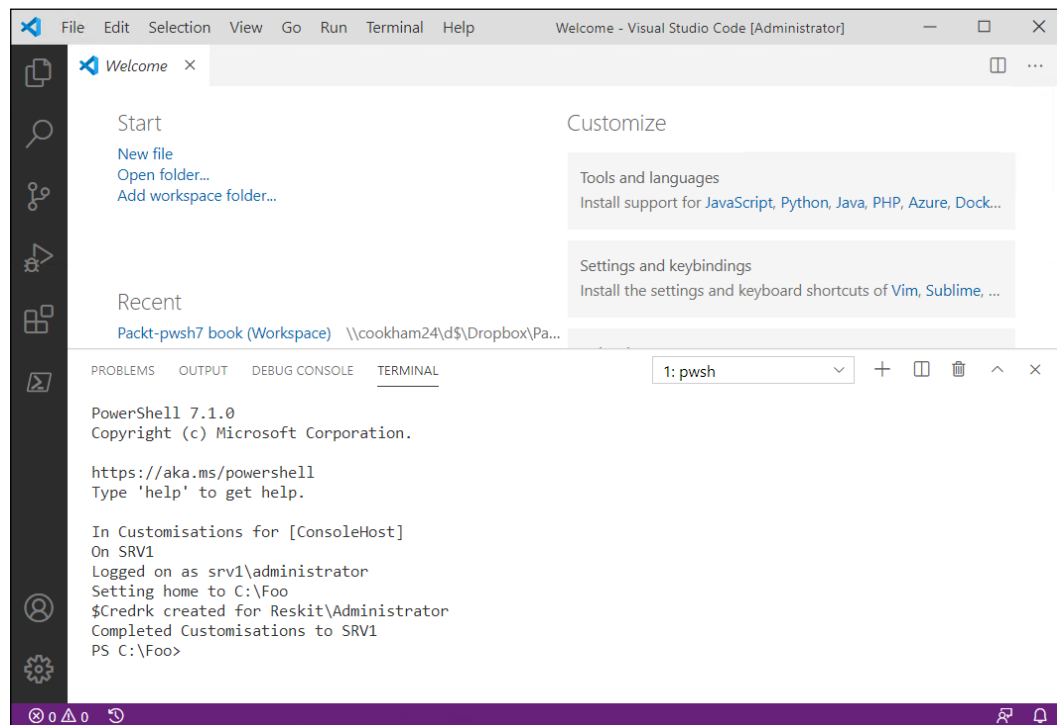


Figure 1.28: VS Code (from shortcut)

There's more...

In *step 2*, you install VS Code and three additional VS Code extensions. The `Streetsidesoftware.code-spell-checker` extension provides spell-checking for your scripts and other files. The `yzhang.markdown-all-in-one` extension supports the use of Markdown. This extension is useful if, for example, you are writing documentation in GitHub or updating the existing public PowerShell 7 help information. The `hediet.vscod-drawio` extension enables you to create rich diagrams directly in VS Code. Visit the VS Code Marketplace for details on these and other extensions.

In *step 4*, you ensure you are running VS Code as an administrator. Some of the code requires this and fails if you are not running PowerShell (inside VS Code) as an admin.

In *step 5*, you open a terminal inside VS Code. In VS Code, the "terminal" is initially a Windows PowerShell console. You can see in the output the results of running the profile file for Windows PowerShell. This terminal is the one you see inside VS Code by default.

In *step 7*, you update and save some updates to the VS Code settings. Note that in this step, you tell VS Code where to find the version of PowerShell you wish to run. You can, should you choose, change this to run a preview version of PowerShell or even the daily build.

Note that VS Code has, in effect, two PowerShell profile files at play. When you open VS Code on its own, you get the terminal you see in *step 5* – this is the default terminal. The PowerShell VS Code runs a separate terminal whenever you open a `.PS1` file. In that case, VS Code runs the VS Code-specific profile you set in *step 6*.

In *step 8* and *step 9*, you create shortcuts to VS Code and the PowerShell 7 console. In *step 10*, you update the layout of the Windows taskbar to include the two shortcuts. Unfortunately, you have to log off (as you do in *step 12*) before logging back in to Windows where you can observe and use the two shortcuts.

Installing the Cascadia Code font

As part of the launch of VS Code, Microsoft also created a new and free font that you can download and use both at the PowerShell 7 console and inside VS Code. This recipe shows how you can download the font, install it, and set this font to be the default in VS Code.

Getting ready

You run this recipe on SRV1 after you have installed both PowerShell 7 and VS Code.

How to do it...

1. Getting the download locations for the Cascadia Code font

```
$CascadiaFont = 'Cascadia.ttf' # font file name
$CascadiaRelURL = 'https://github.com/microsoft/cascadia-code/releases'
$CascadiaRelease = Invoke-WebRequest -Uri $CascadiaRelURL # Get all of them
$CascadiaPath = "https://github.com" + ($CascadiaRelease.Links.href |
    Where-Object { $_ -match "($CascadiaFont)" } |
    Select-Object -First 1)
$CascadiaFile = "C:\Foo\$CascadiaFont"
```
2. Downloading the Cascadia Code font file

```
Invoke-WebRequest -Uri $CascadiaPath -OutFile $CascadiaFile
```
3. Installing the Cascadia Code font

```
$FontShellApp = New-Object -Com Shell.Application
$FontShellNamespace = $FontShellApp.Namespace(0x14)
$FontShellNamespace.CopyHere($CascadiaFile, 0x10)
```
4. Restarting VS Code

Click on the shortcut in the taskbar

How it works...

In *step 1*, you discover the latest version of the Cascadia Code font, and in *step 2*, you download this font.

In *step 3*, you install the font into Windows. Since Windows does not provide any cmdlets to perform the font installation, you rely on the older Windows Shell.Application COM object.

In step 4, you restart VS Code and note that the new font now looks like this:

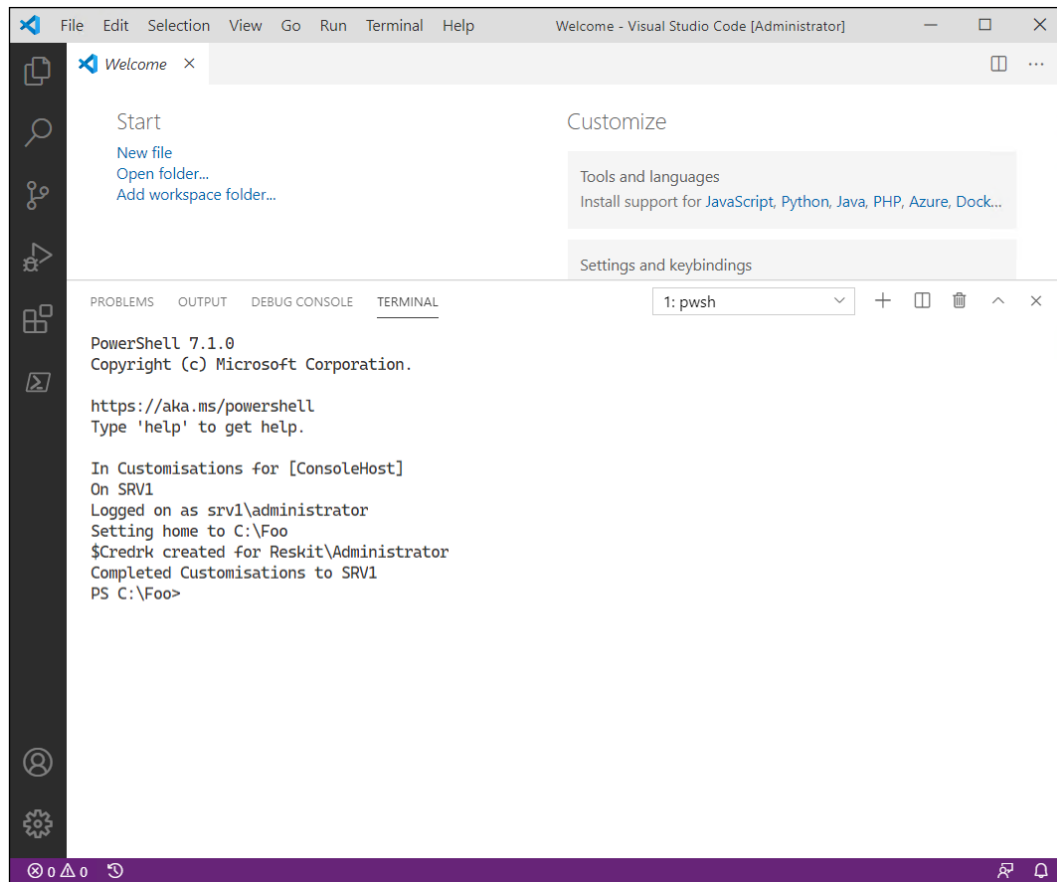


Figure 1.29: Looking at the new font

There's more...

Like so many things in customizing PowerShell 7, you have a wide choice of fonts to use with VS Code. The Cascadia Code font is a good, clear font and many like it. You can always change it if you do not like it.

Exploring PSReadLine

PSReadLine is a PowerShell module that provides additional console editing features within both PowerShell 7 and Windows PowerShell. The module provides a command-line editing experience that is on par with the best of the Linux command shells (such as Bash).

When you type into a PowerShell console, PSReadLine intercepts your keystrokes to provide syntax coloring, simple syntax error notification, and a great deal more. PSReadLine enables you to customize your environment to suit your personal preferences. Some key features of the module include:

- ▶ Syntax coloring of the command-line entries
- ▶ Multi-line editing
- ▶ History management
- ▶ Customizable key bindings
- ▶ Highly customizable

For an overview of PSReadLine, see https://docs.microsoft.com/powershell/module/psreadline/about/about_psreadline. And for more details, you can view the PSReadLine GitHub README file: <https://github.com/PowerShell/PSReadLine/blob/master/README.md>.

An important issue surrounds the naming of this module. The original name of the module was PSReadline. At some point, the module's developers changed the name of the module to PSReadLine (capitalizing the L character in the module name).

The PSReadLine module ships natively with PowerShell 7. At startup, in both the console and VS Code, PowerShell imports PSReadLine so it is ready for you to use. You can also use PSReadLine in Windows PowerShell. To simplify the updating of this module's help, consider renaming the module to capitalize the L in PSReadLine.

The PSReadLine module is a work in progress. Microsoft incorporated an early version of this module within Windows PowerShell v3. Windows PowerShell 5.1 (and the ISE) inside Windows Server ships with PSReadLine v2. If you are still using earlier versions of Windows PowerShell, you can download and utilize the latest version of PSReadLine.

The module has improved and, with the module's v2 release, some changes were made that are not backward-compatible. Many blog articles, for example, use the older syntax for `Set-PSReadLineOption`, which fails with version 2 (and later) of the module. You may still see the old syntax if you use your search engine to discover examples. Likewise, some of the examples in this recipe fail should you run them utilizing PSReadline v1 (complete with the old module name's spelling).

Getting ready

This recipe uses SRV1 after you have installed PowerShell 7, VS Code, and the Cascadia Code font (and customized your environment).

How to do it...

1. Getting the commands in the PSReadLine module

```
Get-Command -Module PSReadLine
```

2. Getting the first 10 PSReadLine key handlers

```
Get-PSReadLineKeyHandler |
  Select-Object -First 10 |
  Sort-Object -Property Key |
  Format-Table -Property Key, Function, Description
```

3. Counting the unbound key handlers

```
$Unbound = (Get-PSReadLineKeyHandler -Unbound).count
"$Unbound unbound key handlers"
```

4. Getting the PSReadLine options

```
Get-PSReadLineOption
```

5. Determining the VS Code theme name

```
$Path      = $Env:APPDATA
$CP        = '\Code\User\Settings.json'
$JsonConfig = Join-Path $Path -ChildPath $CP
$ConfigJSON = Get-Content $JsonConfig
$Theme = $ConfigJson |
  ConvertFrom-Json |
  Select-Object -ExpandProperty 'workbench.colorTheme'
```

6. Changing the color scheme if the theme name is Visual Studio Light

```
If ($Theme -eq 'Visual Studio Light') {
  Set-PSReadLineOption -Colors @{
    Member      = "`e[33m"
    Number      = "`e[34m"
    Parameter    = "`e[35m"
    Command     = "`e[34m"
  }
}
```

How it works...

In step 1, you view the commands in the PSReadLine module, which looks like this:

```
PS C:\Foo> # 1. Getting commands in the PSReadLine module
PS C:\Foo> Get-Command -Module PSReadLine
```

CommandType	Name	Version	Source
Function	PSConsoleHostReadLine	2.1.0	PSReadLine
Cmdlet	Get-PSReadLineKeyHandler	2.1.0	PSReadLine
Cmdlet	Get-PSReadLineOption	2.1.0	PSReadLine
Cmdlet	Remove-PSReadLineKeyHandler	2.1.0	PSReadLine
Cmdlet	Set-PSReadLineKeyHandler	2.1.0	PSReadLine
Cmdlet	Set-PSReadLineOption	2.1.0	PSReadLine

Figure 1.30: Viewing commands in the PSReadLine module

In step 2, you display the first 10 PSReadLine key handlers currently in use, which looks like this:

```
PS C:\Foo> # 2. Getting the first 10 PSReadLine key handlers
PS C:\Foo> Get-PSReadLineKeyHandler |
    Select-Object -First 10
    Sort-Object -Property Key |
    Format-Table -Property Key, Function, Description
```

```
Basic editing functions
=====
```

Key	Function	Description
Enter	AcceptLine	Accept the input or move to the next line if input is missing a closing token.
Shift+Enter	AddLine	Move the cursor to the next line without attempting to execute the input
Backspace	BackwardDeleteChar	Delete the character before the cursor
Ctrl+h	BackwardDeleteChar	Delete the character before the cursor
Ctrl+Home	BackwardDeleteLine	Delete text from the cursor to the start of the line
Ctrl+Backspace	BackwardKillWord	Move the text from the start of the current or previous word to the cursor to the kill ring
Ctrl+w	BackwardKillWord	Move the text from the start of the current or previous word to the cursor to the kill ring
Ctrl+C	Copy	Copy selected region to the system clipboard. If no region is selected, copy the whole line
Ctrl+c	CopyOrCancelLine	Either copy selected text to the clipboard, or if no text is selected, cancel editing the line with CancelLine.
Ctrl+x	Cut	Delete selected region placing deleted text in the system clipboard

Figure 1.31: Displaying the first 10 PSReadLine key handlers

PSReadLine provides over 160 internal functions to which you can bind a key combination. In step 3, you discover how many functions are currently unbound, which looks like this:

```
PS C:\Foo> # 3. Discovering a count of unbound key handlers
PS C:\Foo> $Unbound = (Get-PSReadLineKeyHandler -Unbound).count
PS C:\Foo> "$Unbound unbound key handlers"
116 unbound key handlers
```

Figure 1.32: Discovering the count of unbound key handlers

In step 4, you view the PSReadLine options as you can see here:

```
PS C:\Foo> # 4. Getting the PSReadLine options
PS C:\Foo> Get-PSReadLineOption

EditMode                               : Windows
AddToHistoryHandler                    : System.Func`2[System.String,System.Object]
HistoryNoDuplications                  : True
HistorySavePath                        : C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\
                                       PowerShell\PSReadLine\Visual Studio Code Host_history.txt
HistorySaveStyle                       : SaveIncrementally
HistorySearchCaseSensitive             : False
HistorySearchCursorMovesToEnd         : False
MaximumHistoryCount                   : 4096
ContinuationPrompt                    : >>
ExtraPromptLineCount                  : 0
PromptText                            : {> }
BellStyle                             : Audible
DingDuration                          : 50
DingTone                              : 1221
CommandsToValidateScriptBlockArguments : {ForEach-Object, %, Invoke-Command, icm, Measure-Command, New-Module,
                                       nmo, Register-EngineEvent, Register-ObjectEvent, Register-WMIEvent,
                                       Set-PSBreakpoint, sbp, Start-Job, sajb, Trace-Command, trcm,
                                       Use-Transaction, Where-Object, ?, where}
CommandValidationHandler               :
CompletionQueryItems                   : 100
MaximumKillRingCount                  : 10
ShowToolTips                          : True
ViModeIndicator                       : None
WordDelimiters                        : ;,:.[]{}()^&*~+=+ ' " —
AnsiEscapeTimeout                     : 100
CommandColor                          : "e[34m"
CommentColor                          : "e[32m"
ContinuationPromptColor                : "e[37m"
DefaultTokenColor                     : "e[37m"
EmphasisColor                         : "e[96m"
ErrorColor                            : "e[91m"
KeywordColor                          : "e[92m"
MemberColor                           : "e[33m"
NumberColor                           : "e[34m"
OperatorColor                         : "e[90m"
ParameterColor                        : "e[35m"
SelectionColor                        : "e[30;47m"
StringColor                           : "e[36m"
TypeColor                             : "e[37m"
VariableColor                         : "e[92m"
```

Figure 1.33: Getting the PSReadLine options

In *step 5* and *step 6*, you determine the current VS Code theme name and update the colors used by PSReadLine for better clarity. These two steps produce no output, although you should see the colors of specific PowerShell syntax tokens change (run `Get-PSReadLineOption` to see the changes).

There's more...

In *step 1*, you view the commands inside the PSReadLine module. The `PSConsoleHostReadLine` function is the entry point to this module's functionality. When the PowerShell console starts, it loads this module and invokes the `PSConsoleHostReadLine` function.

One small downside to PSReadLine is that it does not work well with all Windows screen reader programs. For accessibility reasons, if the PowerShell console detects you are using any screen reader program, the console startup process does not load PSReadLine. You can always load PSReadLine manually either in the console or by updating your startup profile(s).

In *step 2*, you view the key handlers currently used by PSReadLine. Each key handler maps a keystroke combination (for example, `Ctrl + L`) to one of over 160 internal PSReadLine functions. By default, `Ctrl + L` maps to the PSReadLine `ClearScreen` function – when you type that key sequence, PSReadLine clears the screen in the console.

In *step 5* and *step 6*, you detect the VS Code theme name and adjust the color scheme to match your preferences. To some degree, the PSReadLine and VS Code color themes can result in hard-to-read color combinations. If that is the case, you can persist the specific settings into your VS Code profile file and change the color scheme to meet your tastes.