Why PowerShell is an essential tool

Microsoft's PowerShell is not just for scripting, but is now a core component for managing and automating Windows servers and Microsoft online services. Tim Anderson reports

MICKEING/ADOBE

HOME

The history of PowerShell, Microsoft's scripting and automation platform, goes back to 2002, when the company started talking about a project codenamed "Monad", described in an early whitepaper as "the next-generation platform for administrative automation". The paper, called the *Monad manifesto*, was written by Jeffrey Snover, who is now chief architect for the enterprise cloud group at Microsoft, including Windows Server, Azure Stack and System Center.

Although there are differences between what was described in the whitepaper and what was eventually released, many of the themes are there. Monad was to be based on the .Net platform, extensible through components called "cmdlets", and would support "pipelining of .Net objects" so that the output from one could form the input to the next. Monad would also be based on the Bourne Shell syntax, which remains a strong influence on command shells on Unix-like systems – Bash stands for Bourne Again Shell.

The *Monad manifesto* was also a recognition that Microsoft's existing options for scripting on Windows – either batch files run by CMD.exe, or the scripting host CScript.exe which runs VBScript of Jscript – were not good enough to support future versions of Windows Server.

Core component of Windows

PowerShell has become a core component of Windows and an essential tool for system administrators. If there is one word that sums up the reason why, it is automation, for which graphical user interface (GUI) tools are unsuitable. Anything that can be

scripted can also be automated, and automation is required for DevOps and for administration of cloud platforms. Another factor is that Windows Server does not always have a GUI. Server Core and Nano Server are stripped down, making PowerShell essential. If you use Office 365, services such as Exchange Online can only be managed via a web browser or through PowerShell, and it is PowerShell that provides the most complete and powerful access. Even if you run Exchange on-premise, PowerShell is required for some tasks, and it pays to know the PowerShell way of doing things.

> *A day rarely goes by without using some form of shell script or another. Learn how those scripts work and start writing them.*

PowerShell is particularly valuable for tasks that involve iterating through lists. Active Directory is an obvious example. The Active Directory module is installed automatically on domain controllers, or when you install the Remote Server Administration Tools. The Get-ADUser cmdlet retrieves all users into a collection. You can filter the list or use a foreach loop to inspect each user object, and use Set-ADUser to modify user attributes or group memberships.

Learning the language

PowerShell is a dynamic, case-insensitive language based on the .Net Framework. It is designed for both immediate interactive use and for scripting. Although developers familiar with C# should find PowerShell a relatively easy transition, it is not an English-like language, and the way code is sprinkled with dollar signs, vertical bar and hyphen characters can be confusing for newcomers.

Variables in PowerShell are prefixed with a ***$*** symbol, and there is a clear separation between assignment operators such as **=** (assign), ***+=*** (add and assign), ***-=*** (subtract and assign), ***/=*** (divide and assign); and comparison operators such as ***-eq*** (equality), ***-gt*** (greater than) and ***-lt*** (less than). The ***-match*** operator uses regular expressions, and there are also operators for containment, replace and bitwise operations.

PowerShell cmdlets support short aliases that enable more concise code or quicker typing, but also make code less readable, so use them with caution.

POWERSHELL IS NOT AN ENGLISH-LIKE LANGUAGE, AND THE WAY CODE IS SPRINKLED WITH DOLLAR SIGNS, VERTICAL BAR AND HYPHEN CHARACTERS CAN BE CONFUSING FOR NEWCOMERS

Objects and pipeline

PowerShell is an object-oriented language, which makes it different to other command shells, even though it may look similar. For example, if you type ***dir*** you get a list of files in the current directory, similar but not identical to that in a standard Windows command shell. However in PowerShell, ***dir*** (as well as ***ls***, for those familiar

with Linux or Unix) is an alias for ***get-childitem***, which returns a collection of objects. By default, the output from ***dir*** only shows four properties of each file object: Mode, LastWriteTime, Length and Name. What if you wanted to know the CreationTime? How do you even know that there is a CreationTime property?

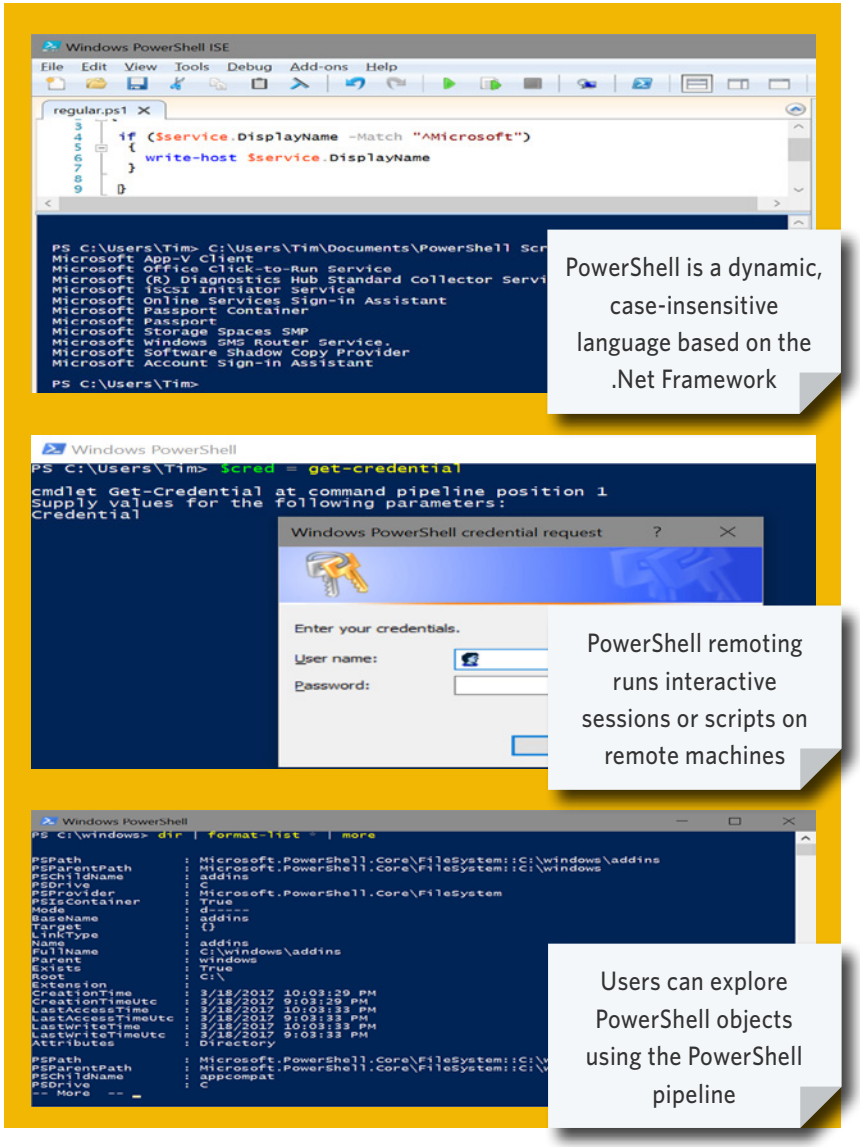
You can explore PowerShell objects using a couple of key features. One is the PowerShell pipeline. The pipeline character is the vertical bar **|**, and using this passes the output to the following cmdlet. The format-list cmdlet will show all the properties, when used with the * argument.

So by typing ***dir | format-list*** * you can see all the properties of each file object, including CreationTime. You could also use the Select-Object (or select) cmdlet to display the exact properties you want, or the Where-Object (or where) cmdlet to filter the output. This would show files larger than 100MB in or below the current folder and save the results in a text file: ***dir -r | where length -gt 100mb | select fullname,length | out-file bigfiles.txt***.

Remoting capability

PowerShell remoting is the ability to run interactive sessions or scripts on remote machines without needing a full remote desktop. It is similar to using Secure Shell (SSH) on Linux to manage a remote server.

PowerShell remoting uses the concept of sessions. The Enter-PSSession cmdlet starts a new session, which may be local or remote. You can use Enter-PSSession with the ***-computer*** argument to start a new interactive session on the specified computer.



Another approach is to create a new session variable using the New-PSSession cmdlet, and then use Import-PSSession to let you use cmdlets that execute remotely. Import-PSSession does not import cmdlets that already exist in the local session, unless you use the AllowClobber parameter. Connections to Exchange Online normally use Import-PSSession.

> *With a PowerShell cmdlet or two, you can create Hyper-V checkpoints, get a list of checkpoints for a VM or revert a VM to a specific checkpoint.*

You can also connect to Azure AD, the Office 365 directory, using the special cmdlet Connect-MsolService; or for the most recent Azure AD module, Connect-AzureAD.

Until recently, PowerShell remoting by default only worked with administrator credentials on the remote computer, as well as requiring Windows Remote Management (WinRM). With Server 2016, Microsoft introduced Just Enough Administration (JEA) which lets you specify users and roles so that full administration rights are not required.

Windows Management Framework bundle

The Windows Management Framework (WMF) is a bundle of management tools, including PowerShell, which you can install together. The latest version of WMF can typically be installed on releases of Windows and Windows Server up to two versions older than is current, so that you can perform remote management. Interdependencies between the WMF components mean you should deploy WMF as a whole to deploy PowerShell. In other words, it is the delivery mechanism for PowerShell.

WMF has several components:

- PowerShell and the PowerShell Integrated Script Environment (ISE) lets you write and execute PowerShell scripts.
- Desired State Configuration (DSC) lets you specify the configuration of a machine, such as which Windows features are installed, and apply it.
- Windows Remote Management (WinRM) is a remote management service and XML protocol for managing Windows. PowerShell Remoting uses the WinRM service.
- Windows Management Instrumentation (WMI) is an implementation of web-based enterprise management (WBEM) which uses Common Information Model (CIM) to represent system components. WMI providers are implemented by Windows components to enable remote management. You can use WMI providers from PowerShell.

Windows PowerShell is built on the full .Net Framework and runs only on Windows. PowerShell Core runs on .Net Core, and therefore on a variety of operating systems including Linux and Mac

Other WMF components include PowerShell Web Services, Software Inventory Logging and a Common Information Model provider for Server Manager.

Windows PowerShell is built on the full .Net Framework and runs only on Windows. PowerShell Core, currently in pre-release, runs on .Net Core, and therefore on a variety of operating systems including Linux and Mac.

Certain editions of Windows, such as Nano Server and Windows 10 IoT Core – which runs on Raspberry Pi – only support .Net Core, and in these cases PowerShell Core is also used.

The purpose of PowerShell on Linux is not to attempt to replace Bash or to improve on existing Linux management utilities. Rather, it is to enable easier management of Windows from Linux and vice versa, including automation with PowerShell DSC.

It also means you can write PowerShell scripts that call native Linux utilities, extending the value of PowerShell skills. It will also be possible to write PowerShell cmdlets in Python. PowerShell on Linux is at an early stage though, and is not yet production-ready.

Fostering a strong ecosystem is critical to the future of PowerShell. Microsoft provides PowerShell to offer a way for third-party developers to extend it by creating their own cmdlets. Currently there are more than 1,500 cmdlets in the gallery.

A glance at the statistics though shows that users are particularly focused on Microsoft Azure, with most of the top 50 downloads being Azure-related. ■