Mike Nelson – SED article

Managing the Enterprise Desktop with PowerShell

When PowerShell was first introduced by Microsoft, many in I.T. thought it was just another approach for scripting tasks and managing Windows servers. Since then, though, it has quickly become the tool of choice for managing, monitoring, and scripting all different types of hardware and software. Many vendors have embedded PowerShell into their products administration and management, and with Windows 7, Microsoft made it a core management platform for the enterprise desktop. Let's take a look at how the IT Administrator can use the robust power of PowerShell, and some free tools, to successfully manage, inventory, and monitor their desktop environment.

One of the frustrating issues with the initial version 1 release of PowerShell was its inability to run scripts and query against remote computers from a central workstation or server. With version 2, which is installed by default with Windows 7 and Server 2008R2 (it can also be downloaded for XP and Vista), PowerShell Remoting, or WinRM as it is technically called, solves those problems and allows for a centralized management solution with a the "single pane of glass" architecture. Some IT shops may already have a third-party product that does what PowerShell can do and maybe more, but most of them require the installation of sort of agent, whereas PowerShell is native to the OS, and PowerShell is a scripting language, which means it can be a very powerful asset even with the other product. So, the first steps in your quest for overall management control is to ensure that all your desktops are running version 2 with WinRM enabled.

To enable WinRM on all your desktops with version 2 installed, you must execute a command as the local Administrator from the local machines PowerShell prompt: Enable-PSRemoting –force. This enables WinRM and opens the appropriate firewall ports for communication with the central management workstation.
Once that is done, the workstation is ready for remote PowerShell commands and queries. Another cmdlet you should run as the local Administrator is to set the security of running scripts. You do this by entering Set-ExecutionPolicy Unrestricted. Of course, depending on your security requirements, you may choose a different security level, such as RemoteSigned for script execution. This ensures that any scripts that you may run in an Interactive session will be executed without error. You can find out more details on the settings and ramifications of them by typing help Set-ExecutionPolicy –detailed at the Powershell prompt.

There are really three ways to work with remote computers from a central workstation via PowerShell remoting. The first allows for one or several machines to be queried, whereas the other two are more of a 1:1 type session. The "Invoke-Command" method is the one most commonly used, as in the examples below. "Interactive" is similar to a Secure Shell or Telnet session to the remote machine.

And “Implicit” will actually import a remote PowerShell session into the central session.

PowerShell V2 itself comes with over 30 cmdlets that allow for sending remote commands to workstations. Just about any native cmdlet that accepts the –ComputerName parameter will send remote commands to any desktop that has WinRM enabled. Here is an example:
The command *Get-EventLog – Logname Application –EntryType Error –ComputerName mypc* would retrieve the Application Event log entries that contain the event type of Error from mypc. You can also extend the –ComputerName parameter to include multiple targets, such as *–Computername mypc, suzipc, tompc*. You could also utilize the scripting language power of PowerShell to turn that parameter into a dynamic variable, which allows for running remote commands against hundreds or even thousands of desktops, but that goes beyond the scope of this article.
One native cmdlet that I use quite often is the *Get-Counter* cmdlet. This allows you to see is (almost) real time the performance measurements of the remote computer. Apply this cmdlet with an extended list of remote computers, or via a dynamic variable, and you can see the counters for all the machines in your organization. Type *help get-counter –full* at a PowerShell prompt to see what syntax is applicable to your situation.

You can also perform inventory functions on the remote machine. For software inventories, you simply query a WMI Class for all .MSI installed software using the *Get-WMIObject* cmdlet. For software not installed via a .MSI installer, you could simply query the registry for all software entries. This TechNet article is an excellent resource for building your own software inventory tool from Don Jones. For hardware inventories, you would use another WMI query that specifies the hardware classes. See this PowerShellPro article for an awesome script that I have used several times.

For those of us that pretty much live in the scripting world on a daily basis, using the cmdlets from the command line is second nature. But for those that don’t have a lot of experience in the CLI, or for some of us that try to avoid extremely long “one-liners” as they are called (it’s very easy to fat-finger a command line), there is a great free tool for you to use. Quest Software makes a product called PowerGUI, which is a GUI front-end to PowerShell, and much more. One of the best features of the product is the use of PowerPacks. These pre-made scripts are created by all the scripting guru’s out there and compiled into a single file. When they are imported into PowerGUI, you get a nice tree selection menu of tasks that you can do to whatever the PowerPack was written for. Basically, PowerGUI is simply entering the PowerShell commands for you, and you can even see the actual scripts in it’s ScriptEditor, which is included, and use them to build your own. I highly recommend that, even if you are a scripting command line person, that you give it a try and be sure to download the PowerPacks for Computers, Networks, and Computer Browser.

Once you start to master the PowerShell environment for your remote desktops, and start to feel brave in your scripting, I also recommend looking at Sapien's PrimalForms for incorporating your scripts into a GUI type interface (they even have a free version). With PrimalForms, you can enter your scripts and put a nice GUI front-end on them for yourself, or better yet, for any folks that don't really feel comfortable in the command prompt. As an example, I created a GUI front-end to a bunch of scripts that I had created for a couple of Helpdesk people and it has become a staple on their desktop for their daily work with users. For an excellent step-by-step example, see Eric Sloof's site over at NTPRO.NL It is a bit older, but is still applicable.

Lastly, you should be aware of all of the literally thousands of resources for Powershell and Powershell scripts. The question I always ask folks who say that they have to create their own script to do what they want, is why reinvent the wheel? 99% of the time, someone else has written a script to do either exactly what you need it to do, or is a great start to customize your own. Use the resources and you will find a community that is open, friendly and willing to help.

Resources-
Technet Powershell ScriptCenter
PowerShell Remoting – The Administrators Guide
PowerGUI.org
PowershellPro site
Poshcode
BSonPOSH – Powershell MVP Brandon Shell's site
Powershell.com – MVP Dr. Tobias Weltner