

PowerShell and SUA together

Tech Note #3 in the "Interop Components in Windows" series

By Rodney Ruddock (Interop Systems)

What is PowerShell?

PowerShell is the Microsoft project originally named "Monad." It is a shell and a scripting language in the same sense that **ksh** or **tcsh** is on Unix, but for the Windows platform. To get tasks done PowerShell can be used interactively or for scripting as the tasks become more complex.

Though PowerShell is for Windows it actually has a fair amount of Unix heritage. Many of the designers and implementers of PowerShell have a Unix background. In fact one of the Program Managers and one of the Designers/Developers were at Softway Systems working on Interix before Microsoft acquired it. So the idea of a real command line shell for Windows that includes most of the features that make Unix shells powerful was very core to the design and development.

The core of PowerShell is its scripting language. Like other scripting languages such as **ksh**, Perl or PHP, real qualitative constructs can be made. Gone are the days of a Windows administrator trying to cope with CMD.EXE. The language has conditions, code blocks, functions, argument passing, regular expression handling, string manipulations, arrays, hashes, etc. -- everything that has come to be expected and needed with a powerful scripting tool.

The biggest feature extension or paradigm shift is that pipes do not stream text as they do on Unix. Instead the pipes stream objects with properties. This makes it very flexible in selecting what information to use at each command in the pipe set. So for each object coming through the pipe just a single property of each object can be selected and processed instead of having to use multiple instances of **sed**, **grep** and **awk** to filter the information to what you want. But while you may be filtering the data at each stage in a pipeline all the objects with all of the data flows through the pipeline are available right to the last stage.

PowerShell also allows the ability to access the .NET APIs, COM objects and the WMI controls as well as the Windows registry and system logs through scripting or by command line. All combined together this hands back to system administrators the ability to probe and set the Windows machines under their domain with better, repeatable consistency.

Lest you worry that this means the end of the GUI for administrators who prefer the methods of point and click – fear not. Similar to the **Tk** extensions for Tcl or Ruby there is the WinForms library that is part of .NET. This library encapsulates the ability to quickly and easily create windows with buttons, panels, scroll bars and more.

Starting with Windows Server 2008, PowerShell will be shipping as part of the base Windows OS. For earlier Windows OS versions you can download and install PowerShell from <http://www.microsoft.com/powershell>.

PowerShell is significant enough that starting with the Server 2008 release Microsoft Exchange 2007 will be using it.

Before coming to the conclusion that PowerShell is to replace everything, this is not the case. PowerShell can work with other existing methods that system administrators use now such as VBScript and the Interix technology in the Windows Subsystem for Unix-based Applications (SUA). In the next section we'll briefly explore how PowerShell and Interix can work together.

Can I run SUA/Interix commands from PowerShell?

Yes, you can run SUA/Interix commands from PowerShell. Be aware that there are a number of built-in commands with PowerShell (called “cmdlets”) that will give a similar answer to the Interix utilities. And they may be faster because a new process is not getting started. But the Unix style of output and behavior may be what you’re looking for.

There’s nothing special to do from PowerShell to work with Interix, as it is installed by default, other than naming the Interix command. For example, you can start a login **ksh** as simply as:

```
PS> ksh -l
```

In fact, at a PowerShell prompt you can pipe one Interix command to another Interix command. The pipe is a PowerShell pipe and the PowerShell pipe knows enough to stream this information to the next pipe process as text rather than as objects. But you need to be careful about which commands you think will be invoked. PowerShell has a number of built-in aliases to mimic Unix commands. So with the command:

```
PS> ls | grep Power
```

you are running the PowerShell alias **ls** and piping its output to the Interix utility **grep**. So you may want to unalias or rename some of these commands.

Can I run PowerShell from Interix?

Yes you can, from an Interix terminal “on the glass” (the desktop). The Interix environment doesn’t have the path to find PowerShell by default. So you can either add the path to your PATH environment variable or enter a full path to get it going:

```
% /dev/fs/C/Windows/System32/WindowsPowerShell/v1.0/powershell.exe
```

```
Windows PowerShell
```

```
Copyright 2006 (C) Microsoft Corporation. All rights reserved.
```

```
PS>
```

PowerShell starts and you can interact with it the same as if you had started a session of PowerShell from the Start Menu.

If you have specific PowerShell commands or aliases you can specify it on the same line. The example below runs the PowerShell alias **ls** and provides the same output:

```
% powershell.exe ls
```

From a telnet session PowerShell can be started (and stopped with ctrl-C). After the greeting message the PowerShell prompt will not appear. Further, any commands you enter will seem to have no effect. The special action to take is to start PowerShell with the command option:

```
% powershell.exe -commands -
```

The trailing dash above is not a mistake and is important for things to work. Commands may now be entered and output from most commands will appear. The same is true for any other connection that is pseudo-terminal based including rlogin and ssh sessions. While this may seem a bit odd, everything continues to be very functional.

You can get a list of all of the available options that can be used when starting PowerShell by starting PowerShell with the option “-?” or “-Help”.

Can I run PowerShell scripts from SUA/Interix?

This is a slightly different question than the one above. Before running any PowerShell script you have to remember to either have the script signed (as in security certificate) or adjust the PowerShell security to be more permissive about running scripts. Assuming one of these adjustments has been done, then the answer is “Yes, it can be done.” The “but” here is that in the Unix world the first line of a script is the magic that tells the system which shell is to process the script file. The Windows world tells the system by the suffix of the filename. For PowerShell the suffix is “.ps1”.

Can these be made to work together? Yes, it can be done by explicitly naming PowerShell as the utility to run with the cmdlets or script as the operand. Again this is in an Interix shell on the glass. For example:

```
% powershell.exe ls | grep Power
```

A key difference with this example compared to the similar example above is that the pipe is an Interix pipe this time. The **ls** is the PowerShell alias and the **grep** is the Interix utility. So the output from PowerShell can be manipulated in the Interix shells.

From a telnet session the same advice given previously applies. You will need to start PowerShell with the “-command” option, but this time you need to have the script as the option argument:

```
% powershell.exe -command myscript
```

How can I get PowerShell running in a SUA/Interix telnet session?

This question was addressed above but a little more detail may provide some understanding of the technical issues. This question also applies equally to an **rsh** or **ssh** session as it does to a **telnet** session.

As with many Win32 programs, PowerShell seems to want direct access to the handles of a console window. When you are on the glass a console window exists to display the output from the Interix shells. So this console window can be “overloaded” or used to provide console window handles to Win32 programs. When connected by **telnet** or **ssh** the session is run entirely on a pseudo-terminal. No console window exists.

By starting PowerShell with the “-command -” option and option argument you are explicitly telling PowerShell that all command text is to be read from standard input. This works perfectly with a pseudo-terminal because Unix-based programs always have their command inputs from standard input.

So why use PowerShell and SUA/Interix together?

From the Interix side, PowerShell can provide access to information and settings not exposed through Interix. PowerShell can provide this through its access to .NET interfaces since .NET provides complete exposure of Windows information and settings. Access to WMI can also be very useful.

From the PowerShell side, Interix can provide some tools not available on the Windows side such as **ssh**. So you can run a PowerShell script that uses **ssh** to another machine to remotely run some commands or transfer some data in a very secure manner.

Note:

SUA, as provided by Microsoft, does not include an ssh client or server. However, OpenSSH for SUA is available as a free download from the [Interop Community](#) site.

Many people have expressed the view that they “don’t need to learn yet another scripting language.” Even on Unix systems, the reality is that administrators rarely have the opportunity to stay within one scripting language. However, it does not mean they have to become experts in the several other languages that they must use occasionally. The same applies with an Interix and PowerShell mixture. Small PowerShell scripts can provide the access to, or the setting of, some raw information available through WMI or .NET while a **ksh** script drives the bigger picture in a manner more familiar or comfortable.

Feel free to post a question in the [Interop Community](#) forums.