Microsoft®

tech·ed

North America | 2010

JUNE 7-10, 2010 | NEW ORLEANS, LA

Microsoft®

SESSION CODE: WCL313

Paradigm Shift: Microsoft Visual Basic Scripting Edition to Windows PowerShell

Jeffery Hicks
Principal Consultant
JDH Information Technology Solutions

Who Am I?

- Windows PowerShell MVP
- VBScript author
  - Advanced VBScript for Windows Administrators (with Don Jones)
  - WSH and VBScript Core: TFM
- PowerShell author
  - Windows PowerShell 2.0: TFM (with Don Jones)
  - Managing Active Directory with Windows PowerShell: TFM
- IT trainer and consultant
- http://jdhitsolutions.com/blog
- http://twitter.com/JeffHicks

Agenda

- PowerShell vs VBScript
- Paradigm Shifts
- Practical Examples
- Best Practices
- Questions
- Next Steps

What's this all about?

- Windows PowerShell is the new management technology we will all be using
- Requires a new mind-set compared to previous management and scripting techniques and tools
- "Scripters" need to transition . What do they need to know?
- My focus is on administrative automation

What's Right about VBScript?

- VBScript is still a good thing, offers value and serves a need
- It is familiar and comfortable
- A decade+ of expertise and scripts that you can draw on
- Can run just about anywhere with very little effort

What's Wrong with VBScript?

- Never designed for Windows administration
- Relatively complicated language of functions, statements and commands
- Coding then execution. A lot of rework and effort to get things right.
- Requires scripting expertise
- Some objects -- but a lot of parsing required.
- A long history of security vulnerabilities

What is Windows PowerShell?

- First and foremost an interactive management shell. Type a command and something happens.
- A batch-file like scripting language
- Part of Microsoft's Common Engineering Criteria which means new products must be managed via PowerShell
- So the question is not if you will use PowerShell, but when.

Windows PowerShell Features and Benefits

- Based on .NET Framework
  - It is a rich object-based shell
  - .NET is at the core of everything Microsoft
- Backwards compatible (with a few exceptions, mostly in the ISE)
- PowerShell is focused on discoverability of itself and your network
- PowerShell is designed to be easy to learn and use

Interactive and Flexible

- There is no “compiling”. Run a command and make it happen.
- No script engine. Run the script in a PowerShell window almost like any other command
- PowerShell expressions are flexible, using command parameters

Cmdlets and Providers

- Cmdlets are PowerShell's core "command-lets" that work with objects
- Written in a .NET language (not that it makes any difference). You don't have to write cmdlets.
- Standardized and consistent in design and implementation
- Full help documentation including plenty of examples and up-to-date online help.

Cmdlets and Providers

- Providers are great for hierarchical storage like the file system and registry
- Providers are *not* great for configuration information (in my opinion)
- Providers offer PSDrives that can be accessed via script in ways that VBScript could never do
- Application/Server specific providers: AD, IIS, WSMan

Cmdlets and Providers

DEMO

PowerShell Scripting is More Secure

- Secure by design
  - Associated with Notepad
  - Require full path to run a script
  - Supports digital signatures
- Execution Policy Restricted by default, which means you can only use the shell interactively.
- Recommended minimum policy is RemoteSigned
- Group Policy configured and enforced. Recommended.

It’s All About the Objects

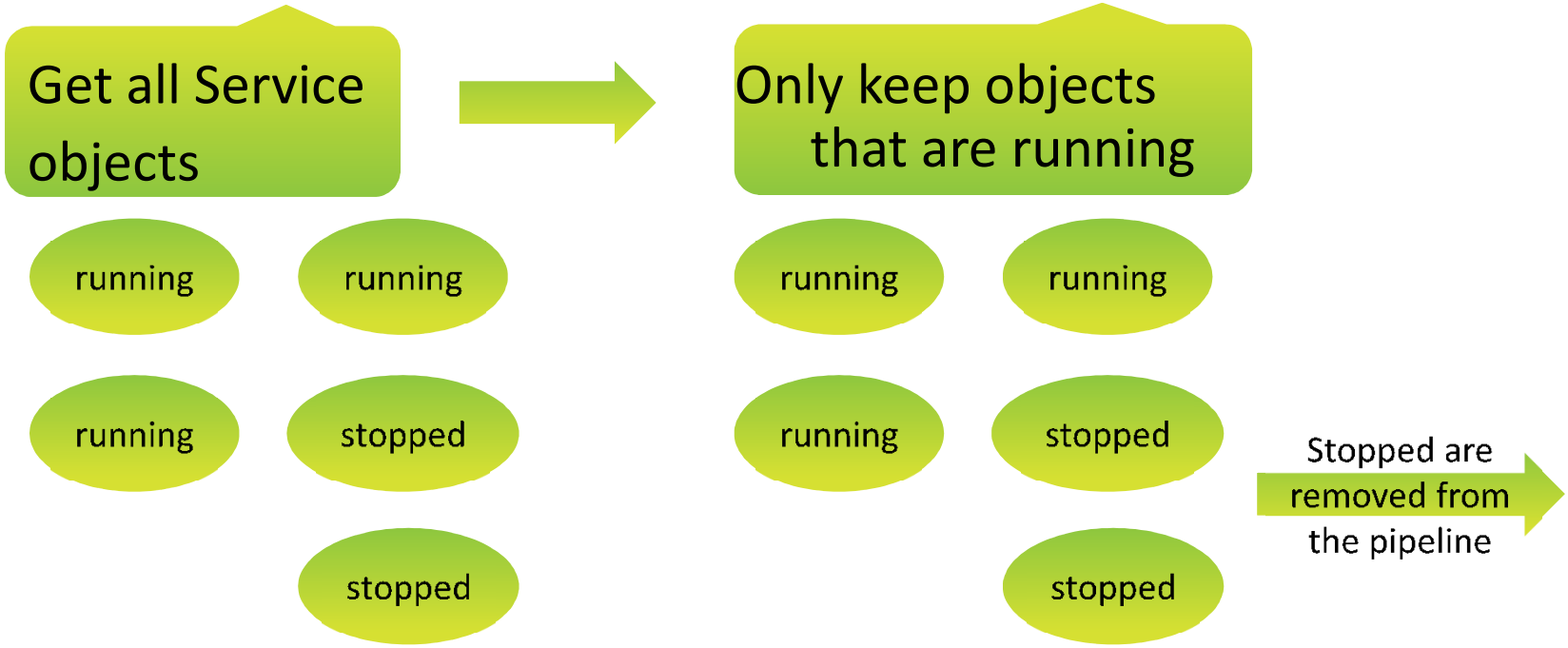
- Windows PowerShell can use .NET and COM objects
- Create your own custom object
- Everything is an object in Windows PowerShell
- Use Get-Member to discover an object’s characteristics (ie properties and methods)
- Use cmdlets or an object method to manipulate objects

The Power of the Pipeline

- Objects move through the pipeline via cmdlets, your scripts or functions
- Objects can change through the pipeline
- At the end of the pipeline Windows PowerShell displays remaining objects

Pipeline Example

```
PS C:\> Get-Service | Where {$_.Status –eq 'running'}
```



Pipeline Example Results

```
PS C:\> Get-Service | Where {$_.Status –eq 'running'}

Status   Name               DisplayName
------   ----               -----------
Running  Appinfo            Application Information
Running  AudioEndpointBu... Windows Audio Endpoint Builder
Running  AudioSrv           Windows Audio
Running  BFE                Base Filtering Engine
Running  BITS               Background Intelligent Transfer Ser...
Running  bthserv            Bluetooth Support Service
Running  cmdAgent           COMODO Internet Security Helper Ser...
Running  ConfigFree Gadg... ConfigFree Gadget Service
Running  ConfigFree Service ConfigFree Service
Running  CryptSvc           Cryptographic Services
```

Objects and Properties

DEMO

File System

DEMO

Windows Management Instrumentation

DEMO

Return vs Write-Output

- VBScript functions return single “value”
- PowerShell functions multiple “values” (think objects)
- Return keyword means 1 value is “returned”
- Write-Output sends to the pipeline
- Think “Writing objects to the pipeline” instead of returning values

Return vs Write-Object

DEMO

Selecting vs Formatting

- Select objects and properties anywhere in the pipeline
- Formatting is a separate process
  - Format-Table
  - Format-List
  - Format-Wide
- Formatting directives should be at end of your expression except when outputting to a file or printer.

Selecting vs Formatting

DEMO

Best Practices

- Always think “object-ively” Don’t spin your wheels parsing text.
- Leverage the pipeline but don’t feel you have to use complex one-liners
- Focus on re-use and modularization in scripts and functions.

Best Practices

- Use cmdlets whenever possible. Use “raw” .NET objects where there are no alternatives.
- Use Write-Host for messages Write-Output for pipelined data
- Build-in trace messages from the beginning
- Keep formatting separate from your script or function

Best Practices

- No aliases in scripts. Use full cmdlet names
- Use full parameter names in scripts
- Use standard naming conventions where possible for scripts and functions
- Documentation (do I really have to say it?). Build it in from the very beginning

Your Action Plan

- Do Nothing (don't re-invent the wheel). If you have VBScripts that work, continue to use them.
- Deploy, configure and secure Windows PowerShell 2.0
- Invest in training and tools
- Look for "Quick Wins" that demonstrate Windows PowerShell's strengths
- Begin using Windows PowerShell daily

Resources

- PoshCode.org
- The Lonely Administrator (http://jdhitsolutions.com/blog)
- Windows PowerShell 2.0: TFM by Don Jones and Jeffery Hicks
- Windows PowerShell in Action 2nd Ed. By Bruce Payette
- Windows PowerShell Team blog (http://blogs.msdn.com/powershell)
- Prof. PowerShell (http://mcpmag.com/articles/list/prof-powershell.aspx)

Resources

Microsoft tech·ed Online

Sessions On-Demand & Community

www.microsoft.com/teched

Microsoft | Learning

Microsoft Certification & Training Resources

www.microsoft.com/learning

Microsoft TechNet

Resources for IT Professionals

http://microsoft.com/technet

msdn

Resources for Developers

http://microsoft.com/msdn