WolfTech
Active Directory: PowerShell

March 7th, 2012
2-4pm Daniels 201

http://activedirectory.ncsu.edu
What we are going to cover...

- Powershell Basics
- Listing Properties and Methods of Commandlets
- .Net Libraries
- Available Modules
- Securing Powershell Scripting
- Regular Expressions
- Powershell ActiveDirectory Module
- PSRemoting
- Server Administration - Roles
A PowerShell cmdlet is a pre-compiled command that can be invoked by the PowerShell runtime. There are many that come built-in to the runtime and many more can be loaded into the environment. Cmdlets have a Verb-Noun naming convention.

<table>
<thead>
<tr>
<th>Core Cmdlets</th>
<th>Core Cmdlets</th>
<th>Core Cmdlets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get-Command</td>
<td>Get-Help</td>
<td>Sort-Object</td>
</tr>
<tr>
<td>Select-Object</td>
<td>Compare-Object</td>
<td>Where-Object</td>
</tr>
<tr>
<td>Get-WmiObject</td>
<td>Get-WinEvent</td>
<td>Get-EventLog</td>
</tr>
<tr>
<td>Group-Object</td>
<td>Import-CSV</td>
<td>Export-CSV</td>
</tr>
<tr>
<td>Format-Table</td>
<td>Format-List</td>
<td></td>
</tr>
</tbody>
</table>
Getting Help

- **Get-Help <Command>**
  - `-detailed` displays more detailed help
  - `-examples` will display usage examples
  - `-full` displays everything, including detailed help, per-parameter help, and usage examples

- **About pages**
  - `Get-Help about_<topic>`
  - About pages exist at C:\Windows\System32\WindowsPowerShell\v1.0\en-US

- **To find commands** issue the `Get-Command` commandlet
  - `Get-Command get-*`  `Get-Command remove-*`
  - `Get-Command set-*`  `Get-Command *`
  - `Get-Command * | where {$_.CommandType -eq "cmdlet"}`

- **Get-Help alias is help**
Variables

● The special '$' character is used to notify the shell of a variable.
● Variables can contain text, objects, integers, or a collection of objects.
● Variables are stored in the PSDrive 'Variable'.
● Cmdlets available for managing variables:
  ○ New-Variable  ○ Get-Variable
  ○ Remove-Variable  ○ Clear-Variable
  ○ Set-Variable  ○ Dir Variable
● Piped variables pipe their content.
● Piped arrays refer to the current object using $$_$
● Powershell will attempt to convert (coerce) objects into the necessary type.
● You can inform Powershell of type using square brackets.
Variable Examples

$var = 10
$process = Get-Process

$password = Read-Host -Prompt 'Enter Password' -AsSecureString
$wmi = gwmi Win32_OperatingSystem
    $wmi.Caption
    $wmi.Reboot()

[int]$value = '7'
[string]$value = 'Hello'
Get-Member

- Shows you the members of an object that has been passed into it
- Shows the type name, the formal and unique name by which the object is known
- Can view nested objects
- Aliased to "gm"
- Can show you static methods and properties of classes

Get-Process | Get-Member
Get-Process | gm | where {$_.MemberType -eq "Property"}
Get-WmiObject Win32_OperatingSystem | gm | Out-GridView
[math] | Get-Member -Static
Comparison Operators are used to evaluate "True" or "False"
Powershell references each by $True and $False

4 < 14 is False
Boy = Girl is Fales
7 ≤ 7 is True

Execute Powershell comparisons from the command line:

4 -lt 10 returns False
"Boy" -eq "Girl" returns False
7 -le 7 returns True
More Operators

- **Common Operators**
  - `eq` : Equal to
  - `ne` : Not equal to
  - `le` : Less than or equal to
  - `ge` : Greater than or equal to
  - `gt` : Greater than
  - `lt` : Less than
  - `like` : Uses wildcards for pattern matching
  - `notlike` : Negates the `like`
  - `match` : Uses regular expressions for pattern matching
  - `notmatch` : Negates the `match`
  - `contains` : Determine elements in a group.
    - Evaluates to Boolean $True$ or $False$
  - `++`, `--` : Increment / decrement
Pipelines, and Filtering

- The Where-Object cmdlet removes objects from the pipeline based on specified criteria.
- Where-Object uses $$_ to represent the current pipelined object.
- Where-Object is aliased as 'where'.
- Using Where-Object can be computationally expensive due to the need to process each object.

```
Get-Process | Out-GridView # Short list
Get-Process | Select-Object * | Out-GridView # Long list

Get-Service | Where-Object { $_.Status -eq "Running"}
```
Quotes, Brackets, and other Punctuation

● Single quotes is a literal string
  ○ ‘This costs $100’
● Double quotes is an evaluated string
  ○ $what = ‘$100’; “This costs $what”
● Back ticks used as the escape character
  ○ “This is `”quite`” a string”; “This is on `n two lines”
● Semicolon is the command separator
● Parentheses are for required parts of control structures and inline execution
  ○ if (<test1>) {<statement list 1>}
  ○ $page = (new-object system.net.webclient).DownloadString($url)
● Braces are for the block expressions in control structures
  ○ do {<statement list>} until (<condition>)
● Brackets are used for optional elements
  ○ [string]typing variables, $indices[0], regular expressions[a-z], and function parameters
● Double colon is used for calling external methods
  ○ [System.Math]::Round($Num, 2)
● Pound is a comment, <# ... #> is for block comments
Regular Expressions

- Powershell provides multiple operators for direct text manipulation
  - -replace
  - -join
  - -split
  - -match
- Powershell supports industry-standard (Perl) regex syntax to perform pattern matching on strings
- Use Select-String cmdlet to perform both simple and regular expression matches against text.
  - Analogous to 'grep' in UNIX
Powershell Providers

● Powershell Providers are .NET programs that provide data stores as if they were mounted drives
● Simplifies access to external data outside the Powershell environment
● Provides access to data that would not otherwise be easily accessible from the command line
● View available Providers with
  ○ Get-PSProvider
● Use PSDrive to mount Powershell Providers
  ○ Get-PSDrive
Profiles

- Profiles in Powershell provide a mechanism to customize the shell environment automatically with each new session
- Preload modules, custom aliases, connect PSDrives, or anything else that is invoked via a Powershell CLI
- Powershell searches for a predetermined file in a predetermined location
  - Globally - `$PSHome\profile.ps1`
  - User - `$Profile\profile.ps1`
- Powershell profiles will not execute if connected to a remote instance of the shell
- Powershell profiles will not execute when used within other products
- Powershell profiles will not execute when execution policy is set to Restricted or if the profile is not signed while the execution policy is set to AllSigned
Powershell Modules

- Powershell modules are installed in the systemwide module folder
  - C:\Windows\System32\WindowsPowerShell\v1.0\modules
- Custom modules can be built per user. They should be installed in the user's home directory
  - C: \Users\<userName>\Documents\WindowsPowerShell\Modules\<moduleFolderName>\<customModule>.psm1
- View module search paths with
  - Get-Content env:psmodulepath
- Modules must be added into the shell to use their functionality. They must remain loaded for the entire shell session while being used.
- View installed modules with
  - Get-Module -list
Powershell Modules

- Active Directory Domain Services
- Active Directory Rights Management Services
- Applocker
- Best Practices Analyzer
- Background Intelligent Transfer Service
- Group Policy and Group Policy objects
- Network Load Balancing
- Windows Powershell Diagnostics
- Remote Desktop Services
- Server Manager
- Server Migration
- Troubleshooting Packs
- Web Administration
- Web Services for Management (WS-MAN)
- Windows Cluster Service
- Windows Server Backup
Server Manager

- Use Server Manager cmdlets to add roles, remove roles, and list roles
- Use Server Manager cmdlets, along with Compare-Object, to provide a change configuration report for role changes on a server
- Three Server Manager cmdlets exist
  - Get-WindowsFeature
  - Add-WindowsFeature
  - Remove-WindowsFeature
- Gain access to these cmdlets by importing the ServerManager module
  - Import-Module ServerManager
Powershell and ActiveDirectory

- ActiveDirectory module is installed when RSAT is installed
- Includes cmdlets that facilitate ActiveDirectory administration
- Its cmdlets provide the functionality that powers the graphical Active Directory Administrative Center Console
- Adding the ActiveDirectory module also adds a PSDrive
- It is bad practice to query every object in Active Directory at once
  - Is computationally expensive
  - Can impact Domain Controller's performance
- Most Active Directory cmdlets have a defined mandatory parameter called '-filter'
- Find Active Directory cmdlets with 'Get-Command *-AD*'
Powershell and ActiveDirectory Cont.

```
Import-Module ActiveDirectory
Get-Command *-AD*

Get-ADUser -Filter {Name -eq "ncwolf"}
```
Powershell Code Signing

Get-ExecutionPolicy
Get-Help about_signing
Execution Policies: Restricted / AllSigned / RemoteSigned / Unrestricted
Example Error Message:
The file C:\my_script.ps1 cannot be loaded. The execution of scripts is disabled on this system. Please see "Get-Help about_signing" for more details.

WolfTech Certificate Services
WolfTech Default Execution Policy is "AllSigned"

NCSU-Departmental OU Admins can request a code signing cert:
  You are responsible for the code if you sign it
  Use-Time cert password required on any script used outside your OU
  Certs must be published to “Trusted Publishers” store via GPO
  Scripts need to be saved as ANSI to be signed

PS C:\> Set-AuthenticodeSignature c:\foo.ps1 @(Get-ChildItem cert:\CurrentUser\My-codesigning)[0]
Powershell ISE

- PowerShell ISE can run commands, write, test, and debug scripts in a single GUI
- Provides multiline editing, tab completion, syntax coloring, selective execution, context-sensitive help
MSSQL/Failover Clustering Example

Using the SQL Server cmlets
Using the SQL Server PowerShell Provider

Note: Be Careful of Version (10.0) and Bit level (x86)
SQLPS: C:\Program Files\Microsoft SQL Server\100\Tools\Binn\sqlps.exe
Add-PSSnapin SqlServerProviderSnapin100
Add-PSSnapin SqlServerCmdletSnapin100
Get-PSSnapin

Failover Cluster Cmdlets in Windows PowerShell
Move all cluster resources off of "classdemosrv2":
Import-Module FailoverClusters
Get-ClusterGroup
Get-ClusterNode classdemosrv2 | Get-ClusterGroup | Move-ClusterGroup

Get-ClusterResource | Sort-Object -Property OwnerGroup
Powershell and VMware Example

**PowerCLI Powershell API for VMware VCenter**
Uses a Powershell SnapIn
Pipelining and Filtering

cd "C:\Program Files (x86)\VMware\Infrastructure\vSphere PowerCLI"
Add-PSSnapin -Name VMware.VimAutomation.Core
Scripts\Initialize-PowerCLIEenvironment.ps1
MySQL .Net Example

Connector/Net: fully managed ADO.NET driver written in 100% pure C#

MySQL Powershell Module
MySql.Data.MySqlClient API Reference

[void][system.reflection.Assembly]::LoadFrom("$pwd\MySQL.Data.dll")
Import-Module $pwd\MySQL.psm1
$insert = Prepare-MySQL -server 'server' -database 'db' -user 'user' -password 'password'
$insert | Get-Member

If you have a .Net library and the API documentation, you can use it in Powershell

How do I know if something is using .Net?
Process Explorer
C:\Windows\assembly
The Powershell Script for mapping J and K drives has examples of:

- Looping
- Conditional Statements
- Functions
- Error Checking
- Accessing AD
- Balloon Tips
Additional Examples

Web Services - **Jigsaw**:

```
.\DellWebsiteFunctions.ps1
Get-DellWarranty IDKFA
Get-WMIObject -Class "Win32_BIOS" -Computer . | select SerialNumber
```

DFS/AD Groups - **Celerra**

UI:
Forms
WMI Explorer

**SCCM Powershell Module**
Where Can I Go for Help?

AD Site
- http://activedirectory.ncsu.edu

Mailing Lists
- activedirectory@lists.ncsu.edu

Jabber
- "activedirectory" on conference.jabber.eos.ncsu.edu

Remedy
- wolftech_ad_technical@remedy.ncsu.edu

Governance Committees
- http://activedirectory.ncsu.edu/governance/
Q & A