SVR 308

PowerShell

An Introduction

Chris Green
Technology Specialist
Chris.Green@Microsoft.COM

Make your mark
3 Main Topics...

- What is PowerShell
- How do I get started
- Examples of PowerShell in action
Agenda

- What is PowerShell
- Getting Started
- Language
- Architecture
  - Execution Engine
  - Pipeline
  - Security Model
- Customization
  - Output Format
  - Personal Settings
  - Developing cmdlets
- Scripting Examples...
- Some tips...
What Is Powershell
What is PowerShell?

- Three main **Server 2008** management components
  1. Server Core
  2. Server Manager
  3. **Powershell** - a built-in optional component

- Manage Services, Processes and Storage in all roles

- Manage Specific roles
  - IIS 7
  - Terminal Server

- Manage Specific Workloads
  - Exchange 2007
  - System Center Operations Manager 2007
V1 Released

- Nov 14, 2006 - Download Center
  - XP-SP2 (X86 and X64)
  - W2K3-SP1 (X86, X64 and IA64)
  - Versions: English, MUI, Language Packs

- Jan 30, 2007 – Download Center
  - Vista-RTM (X86 and X64)
  - Supports all the Vista languages (~100), tested for 22 languages/language packs that RTMed by 1/30/07

- Server 2008
  - Included in-band as an Optional Component (OC) starting LHS Beta3. All work complete, and we are in LHS builds
PowerShell Downloads Since RTM

Total downloads to date > 1,300,000
Where is PowerShell Used?


- **Coming:** SQL, Forefront, SharePoint, Mobilizer, AD, DNS, IIS, TS, Clustering, Server manager, SC Configuration Manager, SC Service Desk, IPTV, Windows Diagnostics, E14

- **Partner:** MQ Series, Quest, F5 networks, /n Software, ...
The Tower of Power
What is PowerShell?

• Command Line Interpreter (CLI)
  – Improves the administrative experience
• Script Execution Engine
  – Can be embedded in your applications
• Easy to learn
• Comprehensive online help
• Built in commands (cmdlets), extensible, supports existing CMD, COM, WMI, etc
  – (get-command).count == 129
  – (get-command *).count == 3091 (!?)
• Object Oriented
• Execution environment built on .Net 2.0. Exposes framework through CLI
• Software Developer Kit (SDK)
  – Improves the developer experience by making it easier to add command-line
    management capabilities using .NET
• Secure by design and by default
• Rich error handling
• For you
Why do we need another Shell !?

- Mainframe JCL, Mini Computers DCL, UNIX Bourne
- UNIX Shells set the standard for many: Bourne, Csh, Korn, Tcsh, Bash, ...
- Lots of other scripting languages: Perl, Python, Ruby ... Wikipedia lists 145

- UNIX Sys Admin ratios of 50+:1 are common
- Windows Sys Admin ratios are typically 10-20:1. 30:1 is considered excellent. Why?
- Operational Efficiency, repeatable admin tasks, centralized control, scale out admin.

Do we need another scripting language from Microsoft?
- netsh, wmic, cmd, vbscript, cscript, resource kit commands, product specific commands, MMC snapins

Is Powershell better than sh, csh, ksh, tcsh, bash, ...?
<table>
<thead>
<tr>
<th>Mailbox Statistics</th>
<th>E2K7 (PowerShell one-liner)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetObject(&quot;winmgmts:(impersonationLevel=impersonate)!\COMPUTERNAME\ROOT\MicrosoftExchangeV2).InstancesOf(&quot;Exchange_Mailbox&quot;)</td>
<td>get-mailboxstatistics -server $servername</td>
</tr>
<tr>
<td>For Each objExchange_Mailbox in listExchange_Mailboxs</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;AssocContentCount = &quot; + objExchange_Mailbox.AssocContentCount</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;DateDiscoveredAbsentInDS = &quot; + objExchange_Mailbox.DateDiscoveredAbsentInDS</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;DeletedMessageSizeExtended = &quot; + objExchange_Mailbox.DeletedMessageSizeExtended</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;LastLoggedOnUserAccount = &quot; + objExchange_Mailbox.LastLoggedOnUserAccount</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;LastLogoffTime = &quot; + objExchange_Mailbox.LastLogoffTime</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;LastLogonTime = &quot; + objExchange_Mailbox.LastLogonTime</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;LegacyDN = &quot; + objExchange_Mailbox.LegacyDN</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;MailboxDisplayName = &quot; + objExchange_Mailbox.MailboxDisplayName</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;MailboxGUID = &quot; + objExchange_Mailbox.MailboxGUID</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;ServerName = &quot; + objExchange_Mailbox.ServerName</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;Size = &quot; + objExchange_Mailbox.Size</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;StorageGroupName = &quot; + objExchange_Mailbox.StorageGroupName</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;StorageLimitInfo = &quot; + objExchange_Mailbox.StorageLimitInfo</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;StoreName = &quot; + objExchange_Mailbox.StoreName</td>
<td></td>
</tr>
<tr>
<td>WScript.echo &quot;TotalItems = &quot; + objExchange_Mailbox.TotalItems</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database Mgmt</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dim StorGroup as New CDOEXM.StorageGroup</td>
<td>move-storagegrouppath -identity &quot;First Storage Group&quot; -log &quot;C:\newlogPath&quot;</td>
</tr>
<tr>
<td>StorGroup.DataSource.Open &quot;LDAP://&quot; + DCServer + &quot;/CN=First Storage Group,CN=InformationStore,CN=&quot; + Server + &quot;,CN=Servers,CN=First Administrative Group,CN=Administrative Groups,CN=First Organization,CN=Microsoft Exchange,CN=Services,CN=Configuration,&quot; + DomainName</td>
<td></td>
</tr>
<tr>
<td>StorGroup.MoveLogFiles(&quot;C:\newlogPath&quot;, 0)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recipient Mgmt</th>
<th>enable-mailbox -identity domain\FOO -database &quot;First Storage Group\Private MDB&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set objMailbox = GetObject(&quot;LDAP://&quot; + DCServer + &quot;/CN=FOO,CN=users,&quot; + DomainName)</td>
<td></td>
</tr>
<tr>
<td>objMailbox.CreateMailbox &quot;LDAP://&quot; + DCServer + &quot;/CN=Private MDB,CN=First Storage Group,CN=InformationStore,CN=&quot; + Server + &quot;,CN=Servers,CN=First Administrative Group,CN=Administrative Groups,CN=First Organization,CN=Microsoft Exchange,CN=Services,CN=Configuration,&quot; + DomainName</td>
<td></td>
</tr>
</tbody>
</table>
Exchange 2007 – eq "Much Simpler"

Move Mailbox

Completion
The wizard completed successfully. Click Finish to close this wizard.
Elapsed time 00:00:37
Summary: 1 item(s), 1 succeeded, 0 failed.

Alan Cadman

Exchange Management Shell command completed:
'milkyway.net\Parliment\Alan Cadman\move-mailbox -TargetDatabase
'ExchangeCEF011\First Storage Group\Mailbox Database'

Elapsed Time: 00:00:37
Agenda

- What is PowerShell
- **Getting Started**
- Language
- Architecture
  - Execution Engine
  - Pipeline
  - Security Model
- Customization
  - Output Format
  - Personal Settings
  - Developing cmdlets
- Scripting Examples...
- Some tips...
Getting Started
Easy to get started...

• Do I need to learn .NET before I can use PowerShell?
  – No – PowerShell provides a higher level of abstraction.

• Do I need to rewrite all my existing tools?
  – No - existing tools will run from a PowerShell prompt.

• Do I need to learn the new language?
  – No - You can easily run existing commands without modification.

• Learning the new
  – Online help is full of examples that are ready to use
  – The new language elements make it easy to interact with .NET.
  – Using .NET can help you where new tools don’t exist and opens a vast space of productivity
Getting Started

• Pre-requisites
  – Vista, Longhorn beta 3 (included), XPSP2, Server 2003
  – .Net 2.0

• Download from
  – http://www.microsoft.com/powershell

• Join the community
  – News Group
  – The Blog
    http://blogs.msdn.com/powershell

• Read the bundled documentation
Finding Information

The 4 Points of the Powershell compass

- Get-Help
- Get-Command
- Get-Member
- Get-PSDrive
Finding Information

Man-page style help on language and commands

PS> Help
PS> Help <command>
PS> <command> -?
PS> Help About_While
PS> Get-Help * |Where {$_._.Synopsis -match "process"}

Finding Commands

PS> Get-Command i*
PS> Get-Command –Noun Process
PS> Get-Command –Type {Alias | Function | Filter | Cmdlet | ExternalScript | Application | Script | All}
Agenda

- What is PowerShell
- Getting Started
- **Language**
- Architecture
  - Execution Engine
  - Pipeline
  - Security Model
- Customization
  - Output Format
  - Personal Settings
  - Developing cmdlets
- Scripting Examples...
- Some tips...
The Language
Consistency

- Consistent way to interact with the system
  - Regular naming patterns
    - Verb-Noun
      - Common syntax
        - Cmdlet –param v1 –param2 v1,v2 –flag
      - Common parameters
        - -Debug, -ErrorAction, -ErrorVariable, -OutBuffer, -OutVariable, -Verbose
        - -Whatif, -Confirm (If command has a side-effect)
  - Return objects
Common Parameters

- **-Verbose**  
  Boolean. Generates detailed information about the operation, much like tracing or a transaction log. This parameter is effective only in cmdlets that generate verbose data.

- **-Debug**  
  Boolean. Generates programmer-level detail about the operation. This parameter is effective only in cmdlets that generate debug data.

- **-ErrorAction**  
  Enum. Determines how the cmdlet responds when an error occurs. Values are: Continue [default], Stop, SilentlyContinue, Inquire.

- **-ErrorVariable**  
  String. Specifies a variable that stores errors from the command during processing. This variable is populated in addition to $error.

- **-OutVariable**  
  String. Specifies a variable that stores output from the command during processing.

- **-OutBuffer**  
  Int32. Determines the number of objects to buffer before calling the next cmdlet in the pipeline.

- **-WhatIf**  
  Boolean. Explains what will happen without actually executing the command.

- **-Confirm**  
  Boolean. Prompts the user for permission before performing any action that modifies the system.
Powershell Common Parameters
Exploring Providers

- Other forms of hierarchical storage exist in Windows
  - Registry
  - Environment
  - Certificates
- These "attach" to PowerShell as "drives" using the PSDrive provider mechanism
- *Navigate them in exactly the same way as a file system!*

PS> Get-PSDrive

PS> New-PSDrive -Name myDrive -PSProvider FileSystem -Root C:\myDir

PS> Remove-PSDrive -Name myDrive
Powershell PSDrive Providers
The Language

PowerShell Cheat Sheet

Essential Commands
To get help on any cmdlet use get-help
Get-Help Get-Service
To get all available cmdlets use get-command
Get-Command
To get all properties and methods for an object use get-member
Get-Service | Get-Member

Setting Security Policy
View and change execution policy with Get-Execution and Set-Execution policy
Get-ExecutionPolicy
Set-ExecutionPolicy remotesigned

To Execute Script
powershell.exe -noexit &"c:\myscript.psl"

Variables
Must start with $
$a = 32
Can be typed
[int]$a = 32
To initialise
$a = 1, 2, 4, 8
To query
$b = $a[3]

Arrays

Constants
Created without $
Set-Variable -name $b -value 3.142 -option constant
Referenced with $
$b

Functions
Parameters separate by space. Return is optional.
function sum ([int]$a, [int]$b)
{
    return $a + $b
}
sum 4 5
The Language

Creating Objects
To create an instance of a COM object
New-Object -comobject <ProgID>
    $a = New-Object -comobject "wscript.network"
    $a.username

To create an instance of a .Net Framework object. Parameters can be passed if required
New-Object -type <.Net Object>
    $d = New-Object -Type System.DateTime 2006,12,25
    $d.get_DayOfWeek()

Writing to Console
Variable Name
    $a
or
    Write-Host $a -foregroundcolor "green"

Capture User Input
Use Read-Host to get user input
    $a = Read-Host "Enter your name"
    Write-Host "Hello" $a

Passing Command Line Arguments
Passed to script with spaces
    myscript.ps1 server1 bnp
Accessed in script by $args array
    $servername = $args[0]
    $username = $args[1]

Miscellaneous
Line Break `
    Get-Process | Select-Object `name, ID
Comments#
    # code here not executed
Merging lines;
    $a=1;$b=3;$c=9
Pipe the output to another command |
    Get-Service | Get-Member
The Language

Do While Loop
Can repeat a set of commands while a condition is met
$a=1
Do {$a; $a++}
While ($a -lt 10)

Do Until Loop
Can repeat a set of commands until a condition is met
$a=1
Do {$a; $a++}
Until ($a -gt 10)

For Loop
Repeat the same steps a specific number of times
For ($a=1; $a -le 10; $a++)
{$a}

For Each - Loop Through Collection of Objects
Loop through a collection of objects
Foreach ($i in Get-ChildItem c:\windows)
{$i.name; $i.creationtime}

If Statement
Run a specific set of code given specific conditions
$a = "white"
if ($a -eq "red")
    {"The colour is red"}
elseif ($a -eq "white")
    {"The colour is white"}
else
    {"Another colour"}

Switch Statement
Another method to run a specific set of code given specific conditions
$a = "red"
switch ($a)
{
    "red" {"The colour is red"}
    "white"{"The colour is white"}
    default{"Another colour"}
}
The Language

Reading From a File
Use Get-Content to create an array of lines. Then loop through array
$a = Get-Content "c:\servers.txt"
foreach ($i in $a)
{$i}

Writing to a Simple File
Use Out-File or > for a simple text file
$a = "Hello world"
$a | out-file test.txt
Or use > to output script results to file
.	est.psl > test.txt

Writing to an Html File
Use ConvertTo-Html and >
$a = Get-Process
$a | ConvertTo-Html -property Name,Path,Company > test.htm

Writing to a CSV File
Use Export-Csv and Select-Object to filter output
$a = Get-Process
$a | Select-Object Name,Path,Company | Export-Csv -path test.csv
The Language

Windows PowerShell Quick Reference

How to Access Arguments
To access command line arguments used when starting a script use the automatic variable $args. You can cycle through the individual arguments in the Sargs collection by using code similar to this.
```powershell
foreach ($i in $args) {($i)}
```
To access a particular argument use the collection index number, with 0 representing the first item in the collection, 1 representing the second item, etc.
```powershell
#$args[0]
```
You can reference the item in a collection by using the index number-1.
```powershell
#$args[-1]
```

How to Solicit Input
To solicit input from a user, use the Read-Host cmdlet, followed by the prompt to be displayed.
```powershell
$Name = Read-Host "Please enter your name"
```

How to Insert Line Breaks
To insert a line break into a Windows PowerShell script use the backslash (\).
```powershell
Write-Host "This is a continuation of the line."
```
You can also break lines at the pipe separator (|) character (ensuring your line uses the pipeline).
```powershell
Get-ChildItem C:\scripts | Sort-Object Length -descending
```

How to Use Colored Text
To display text in a different color use the Write-Host cmdlet and specify a foreground color:
```powershell
Write-Host "test" -foregroundcolor "green"
```
You can also specify a different background color.
```powershell
Write-Host "test" -backgroundcolor "red"
```

How to Insert a Paragraph Return
To insert a paragraph return in your output use the newline character in:
```powershell
Write-Host "Line 1
Line 2"
```

How to Write in Reverse Video
To echo a string in reverse video use the Write-Warning cmdlet.
```powershell
Write-Warning "An error has occurred."
```

How to Insert Comments
To insert a comment use the pound sign (#):
```powershell
# This is a comment, not a line to be run.
```

How to Create Multi-Command Lines
To put multiple commands on a single line, separate those commands using a semicolon:
```powershell
$Name = 1,2,3,4,5; $Age = 21; Write-Host $Name $Age
```

How to Make Comparisons
Windows PowerShell provides (like Where-Object) a special set of comparison operators, including those shown in the following table.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lt</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>-gt</td>
<td>Greater than</td>
</tr>
<tr>
<td>-le</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>-eq</td>
<td>Equal to</td>
</tr>
<tr>
<td>-ne</td>
<td>Not equal to</td>
</tr>
<tr>
<td>-like</td>
<td>Like (cases w/ or w/o matching)</td>
</tr>
<tr>
<td>-notlike</td>
<td>Not like (cases w/ or w/o matching)</td>
</tr>
</tbody>
</table>

How to Read a Text File
To read the contents of a text file into a variable, call the Get-Content cmdlet followed by the path to the text file.
```powershell
$Lines = Get-Content C:\scripts\test.txt
```
Each line in the file ends up as an item in the array. If you want to access a single line in the file you can simply specify the index number corresponding to that line.
```powershell
$Lines[1]
```
This command echoes back the last line in $Lines.
```powershell
$Lines[-1]
```

Bonus: To determine the number of lines, words, and characters in a text file use this command:
```powershell
(Get-Content C:\scripts\test.txt | Measure).count
```

How to Write to a Text File
To save data to a text file use the Out-File cmdlet.
```powershell
Get-Process | Out-File C:\scripts\test.txt
```
To append data to an existing file, add the -append parameter:
```powershell
Get-Process | Out-File C:\scripts\test.txt -append
```
You can also use the MS DOS redirection characters (> for write, >> for append) when using Windows PowerShell. This command writes data to the file C:\Scripts\Test.txt.
```powershell
Get-Process > C:\Scripts\Test.txt
```

Another option is to use the Export-CSV cmdlet to save data as a comma-separated values file.
```powershell
Get-Process | Export-CSV C:\Scripts\test.csv
```

How to Write Conditional Statements
To write an if statement use code similar to this:
```powershell
if ($Name = "white") {
    if ($Age -le 10) {
        # "The color is red."
    } else ($Name = "white") {
        # "The color is blue."
    }
}
```
Instead of writing a series of if statements you can use a Switch statement, which is equivalent to VBScript's Select Case statement.
```powershell
$Name = "blue"
switch ($Name) {
    1 {"The color is red.
        2 {"The color is blue."
        3 {"The color is white."
    4 {"The color is yellow."
    default {"Other."
```}

How to Write For and For Each Loops
To write a For statement use code similar to this:
```powershell
for ($i = 1; $i -le 10; $i++) {
    #
}
```
By comparison a ForEach statement might look like this:
```powershell
foreach ($i in Get-ChildItem C:\scripts) {
    #
```}

How to Write Do Loops
To write a Do loop use code like this following, replacing the code between the curly braces with the code to be executed (if ($i=0)|Default of the loop C0). Or replacing this code inside the parenthesis with the loop condition.
```powershell
$i = 1              #
do {($i++)} while ($i -le 10)
```

How to Print Data
To print data to the default printer use the Out-Printer cmdlet.
```powershell
Get-Process | Out-Printer
```

How to Print Data
To print data to the default printer use the Out-Printer cmdlet.
```powershell
Get-Process | Out-Printer
```
### Windows PowerShell Quick Reference

#### How to Create a COM Object
To work with a COM object use the New-Object cmdlet followed by the -comobject parameter and the appropriate ProgID:

```
$a = New-Object -comobject 'Excel.Application'
$a.Visible = $true
```

#### How to Create a .NET Object
To instantiate and use a .NET Framework object choose the class name in square brackets, then separate the class name and the method using a pair of colons:

```
$System.Net.CMS = get-type('System.Net.CMS')
```

To create an object reference to a .NET Framework object using the New-Object cmdlet:

```
$new = New-Object -type system.diagnostics.eventlog -argumentlist system
```

**Note:** This is a cursory overview of working with .NET. The tools techniques shown here will not necessarily work with all .NET classes.

#### How to Work with WMI
To get computer information using WMI call the Get-WMIObject cmdlet followed by the class name:

```
Get-WMIObject Win32_BIOS
```

If the class you are interested in does not reside in the win32 namespace, simply include the -namespace parameter:

```
Get-WMIObject SystemRestore -namespace root\default
```

To access data on another computer use the -computername parameter:

```
Get-WMIObject Win32_BIOS -computername stl-vs-01
```

To limit returned data, use a WQL query and the -query parameter:

```
Get-WMIObject query, 'Select * From Win32_Service Where Status = "Stopped"
```

#### How to Get Help
To get complete help information for a Windows PowerShell cmdlet, use the Get-Help cmdlet along with the -full parameter. For example, to view the help information for the Get-Process cmdlet, type the following:

```
Get-Help Get-Process -full
```

To view the example commands for a cmdlet use the -examples parameter:

```
Get-Help Get-Process -examples
```

If you can't remember the exact name for a cmdlet use Get-Command to retrieve a list of all the cmdlets available to you:

```
Get-Command
```

For a list of available aliases, use the Get-Alias cmdlet:

```
Get-Alias
```

#### How to Copy and Paste
To enable simple copying and pasting in the Windows PowerShell console do the following:

Start Windows PowerShell, then click on the icon in the upper left-hand corner and choose Properties in the Windows PowerShell Properties dialog box on the Options tab, select QuickEdit Mode and then click OK.

To copy text in the console window, select the text and then press ENTER. To paste text into the window click the right mouse button.

#### How to Run a Script
To run a script from within Windows PowerShell, type the full path to the script or (on the Windows PowerShell properties form of the script file) which is stored in a folder that is part of your Windows path:

```
C:\Scripts\Test.ps1
```

If the path name includes blank spaces you must prefix the path with an ampersand and enclose the path in double quotes. For example:

```
C:\Scripts\My Script.ps1
```

From outside Windows PowerShell (e.g., from the Run dialog box or from a command window) you must call Windows PowerShell and then pass the script path as an argument to that call:

```
powershell -ExecutionPolicy RemoteSigned -File C:\Scripts\Test.ps1
```

The -noexit parameter ensures that the PowerShell window remains open after the script finishes running.

### How to Change Security Settings
To run scripts from within Windows PowerShell you will need to change your security settings, by default, PowerShell only runs scripts signed by a trusted authority. To enable PowerShell to run all locally-created scripts (regardless of whether or not they have been signed) use the following command:

```
Set-ExecutionPolicy RemoteSigned
```

#### How to “Interrogate” an Object
To get information about the properties and methods of an object, remove an instance of that object and then “pipe” the object to the Get-Member cmdlet. For example, this command returns the properties and methods available when working with processes:

```
Get-Process | Get-Member
```

#### How to Clear the Console Window
To clear the PowerShell window, use the Clear-Host function (or its alias, cls).
Agenda

- What is PowerShell
- Getting Started
- Language
- Architecture
  - Execution Engine
  - Pipeline
  - Security Model
- Customization
  - Output Format
  - Personal Settings
  - Developing cmdlets
- Scripting Examples...
- Some tips...
Architecture
Hosting Applications

(PowerShell.exe) PS CLI > _

Microsoft.PowerShell.Host, Security, Utility, etc

Exchange 2007 ...

PSSnapins

cmdletProcessor Pipeline Processor Script Parser Providers


.Net 2.0

.netsh wmic CMD.exe cscript WSHost VBscript PHP, Perl, etc...

Windows

- **Script / Parser** - processes language constructs such as scripts, predicates, conditionals, etc.
- **Pipeline Processor** - manages inter-command communication via pipes
- **Command Processor** - manages command execution, registration and associated metadata
- **Session State** - variables and state accessible to script and cmdlets
- **Extended Type System** - a common interface to accessing properties, methods independent of the underlying object type
- **Error Handler** - manages exception to error mapping and reporting

Objects in the Pipeline

- Eg. To find a process by name pattern

  - c:\> tasklist /fi “imagename eq csrss.exe” (how do I match patterns ?)

  - UNIX> ps –efww|grep ss.exe|grep –v grep|awk ‘{print $8}’|sed ‘s/^.* //’
  (The result of this is an ASCII string)

  - PS> get-process|where-object { $_.processname –like “*ss.exe”}
  - or PS> gps|?$($_.processname –like “*ss.exe”}
  (The result is an array of System.Diagnostic.Process objects 😊)
The power of results as Objects
Security
The Security Model - Design

• Secure by design and by default
  • *.ps1 files are associated with notepad.exe
  • Scripts will not run
    • default execution policy is restricted
  • Must use qualified pathname to scripts
  • Can’t run by association because qualified path required
  • Can use shortcuts to run scripts from UI
  • CLI will prompt for action when attempting to run unsigned scripts
**The Security Model - Default**

- **Set-ExecutionPolicy**
  - **Restricted** *(default)*
    - Permits individual commands, but scripts cannot run.
  - **AllSigned**
    - Scripts can run but requires a digital signature from a trusted publisher on all scripts.
    - Prompts you before running scripts from trusted publishers.
    - Risks running signed, but malicious, scripts.
  - **RemoteSigned**
    - Scripts can run but requires a digital signature from a trusted publisher on scripts downloaded from the Internet.
    - Does not require digital signatures on scripts run from the local computer.
    - Does not prompt you before running scripts from trusted publishers.
    - Risks running signed, but malicious, scripts.
  - **Unrestricted**
    - Unsigned scripts can run but warns you that the file originated from the Internet.
    - Risks running malicious scripts.
When Windows (XP SP2, W2k3 SP1, Vista, W2k8) downloads a file from an external “zone” it also creates an alternate stream named `{FILENAME}:Zone.Identifier` which will contain something like

```
[Zonetransfer]
ZoneId=3
```

Based on the zones enumerated in `System.Security.SecurityZone`

```java
public enum SecurityZone {
    NoZone = -1,
    MyComputer = 0,
    Intranet = 1,
    Trusted = 2,
    Internet = 3,
    Untrusted = 4,
}
```
The Security Model - Default

- Use `dir /r` to identify files with the alternate stream and use `notepad {filename}` to see the contents.

- PowerShell doesn't support alternate streams.

- Unsigned, untrusted external code will be marked as "blocked" by the OS.

- `Set-executionpolicy` will partially solve the problem because you will still be prompted to permit the execution of the normally blocked code.

- You have to either delete the alternate stream filename or use the properties dialog to unblock specific files.
Agenda

- What is PowerShell
- Getting Started
- Language
- Architecture
  - Execution Engine
  - Pipeline
  - Security Model
- Customization
  - Output Format
  - Personal Settings
  - Developing cmdlets
- Scripting Examples...
- Some tips...
Customization
Customizing the User profile

- All aliases, functions and variables are only added to the current session. In order to keep changes you need to add them to your profile.
- Each of the profiles listed below are applied in the order shown with subsequent profiles overriding previous settings

%windir%\system32\WindowsPowerShell\v1.0\profile.ps1
This profile applies to all users and all shells.

%windir%\system32\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1
This profile applies to all users, but only to the Microsoft.PowerShell shell.

%UserProfile%\My Documents\WindowsPowerShell\profile.ps1
This profile applies only to the to the current user, but affects all shells.

%UserProfile%\My Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1
This profile ($profile) applies only to the current user and the Microsoft.PowerShell shell.
Customizing the User profile

- What should I store in my $profile?
  - Alias definitions
    - Remap command names
  - Prompt
    ```
    Function Prompt {
      #Anything that returns a string
      "$(pwd) >"
    }
    ```
  - Many preferences controllable via variables
    - PS> dir Variable:*preference*
    - PS> dir Variable:Maximum*
    - PS> dir Variable:Report*
Customizing Output Formats

PS> get-process |where { $_.mainwindowtitle -ne "" } 

<table>
<thead>
<tr>
<th>Handles</th>
<th>NPM(K)</th>
<th>PM(K)</th>
<th>WS(K)</th>
<th>VM(M)</th>
<th>CPU(s)</th>
<th>Id</th>
<th>ProcessName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1063</td>
<td>43</td>
<td>96144</td>
<td>101468</td>
<td>334</td>
<td>72.63</td>
<td>6748</td>
<td>iexplore</td>
</tr>
<tr>
<td>312</td>
<td>6</td>
<td>28028</td>
<td>26256</td>
<td>161</td>
<td>0.86</td>
<td>7284</td>
<td>powershell</td>
</tr>
<tr>
<td>874</td>
<td>31</td>
<td>65660</td>
<td>49980</td>
<td>225</td>
<td>8.03</td>
<td>7044</td>
<td>wmplayer</td>
</tr>
</tbody>
</table>

PS> foreach ( $p in (get-process |where { $_.mainwindowtitle -ne "" })) { $p.mainwindowtitle }

Microsoft Office Live Meeting - viewReg - Windows Internet Explorer
My Customized PowerShell Session
Windows Media Player

PS> gps |?{$_.mainwindowtitle -ne ""}|select processname,cpu,mainwindowtitle

<table>
<thead>
<tr>
<th>ProcessName</th>
<th>CPU</th>
<th>MainWindowTitle</th>
</tr>
</thead>
<tbody>
<tr>
<td>POWERPNT</td>
<td>223.3310316</td>
<td>Microsoft PowerPoint</td>
</tr>
<tr>
<td>powershell</td>
<td>5.928038</td>
<td>Windows PowerShell</td>
</tr>
</tbody>
</table>
Customizing Output Formats
Customizing Output Formats

DotNetTypes.format.ps1.xml

<View>
  <Name>process</Name>
  <ViewSelectedBy>
    <TypeName>System.Diagnostics.Process</TypeName>
    <TypeName>Deserialized.System.Diagnostics.Process</TypeName>
  </ViewSelectedBy>
  <TableControl>
    <TableHeaders>
      <TableColumnHeader>
        <Label>Handles</Label>
        <Width>7</Width>
        <Alignment>right</Alignment>
      </TableColumnHeader>
    </TableHeaders>
    <TableRowEntries>
      <TableRowEntry>
        <TableColumnItems>
          <TableColumnItem>
            <PropertyName>HandleCount</PropertyName>
          </TableColumnItem>
        </TableColumnItems>
      </TableRowEntry>
    </TableRowEntries>
  </TableControl>
</View>
### Customizing Output Formats

DotNetTypes.format.ps1.xml

```xml
<TableColumnHeader>
  <Label>WS(K)</Label>
  <Width>10</Width>
  <Alignment>right</Alignment>
</TableColumnHeader>

<TableColumnItem>
  <ScriptBlock>[int]($_.WS / 1024)</ScriptBlock>
</TableColumnItem>
```
## Customizing Output Formats

```xml
c:\demo\My-DotNetTypes.format.ps1xml
...
<TableHeaders>
  <TableColumnHeader>
    <Label>Window Title</Label>
    <Width>40</Width>
    <Alignment>left</Alignment>
  </TableColumnHeader>
</TableHeaders>
...

<TableRowEntry>
  <TableColumnItems>
    <TableColumnItem>
      <PropertyName>MainWindowTitle</PropertyName>
    </TableColumnItem>
  </TableColumnItems>
</TableRowEntry>
```
Customizing Output Formats

PS> update-formatdata –prepend c:\demo\My-DotNetTypes.format.ps1xml

PS> get-process

<table>
<thead>
<tr>
<th>Window Title</th>
<th>NPM(K)</th>
<th>PM(K)</th>
<th>WS(K)</th>
<th>VM(M)</th>
<th>CPU(s)</th>
<th>Id</th>
<th>ProcessName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Download details:...</td>
<td>11</td>
<td>11272</td>
<td>18112</td>
<td>115</td>
<td>15.83</td>
<td>3836</td>
<td>ieuser</td>
</tr>
<tr>
<td></td>
<td>48</td>
<td>120656</td>
<td>75356</td>
<td>382</td>
<td>91.47</td>
<td>6748</td>
<td>iexplore</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>5272</td>
<td>536</td>
<td>79</td>
<td></td>
<td>344</td>
<td>InoRpc</td>
</tr>
<tr>
<td>Microsoft PowerPo...</td>
<td>35</td>
<td>78336</td>
<td>6204</td>
<td>325</td>
<td>205.89</td>
<td>4864</td>
<td>POWERPNT</td>
</tr>
<tr>
<td>My Customized Pow...</td>
<td>7</td>
<td>37856</td>
<td>37184</td>
<td>161</td>
<td>3.48</td>
<td>7164</td>
<td>powershell</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>16012</td>
<td>540</td>
<td>126</td>
<td></td>
<td>7304</td>
<td>FontCache</td>
</tr>
</tbody>
</table>

PS>
A cmdlet of Your Own

• Powershell is extensible

• Automate your own tasks
  – E.g. In house developed application maintenance

• Download Windows SDK 6.0 from [here](#)

• Use the samples to get started

• The basics are simple...
A cmdlet of Your Own

• The Runspace will call these methods for each cmdlet in the pipeline.
• Override these methods to implement your desired behaviour.
• ProcessRecord() is the most common/important method to override.
using System;
using System.Diagnostics;
using System.ComponentModel;

// This sample introduces the creation of a simple Cmdlet namespace 
namespace Microsoft.Samples.PowerShell.Commands
{

#region GetProcCommand

/// <summary>
/// Class that implements the GetProcPSSnapIn01 cmdlet
/// </summary>

[Cmdlet(VerbsCommon.Get, "Proc")]
public class GetProcCommand : Cmdlet
{
    #region Cmdlet Overrides


/// <summary>
/// Class that implements the GetProcPSSnapIn01 cmdlet
/// </summary>
[Cmdlet(VerbsCommon.Get, "Proc")]
public class GetProcCommand : Cmdlet
{
    protected override void ProcessRecord()
    {
        // Get the current processes

        // Write the processes to the pipeline making them available
        // to the next Cmdlet. The "true" tells the system to
        // enumerate the array, and send one process at a time to
        // the pipeline.
        WriteObject(processes, true);
    }
}

/// GetProcCommand
A cmdlet of Your Own

1. Navigate to the folder containing the compiled DLL.
   DOS C:\> cd C:\Program Files\Microsoft SDKs\Windows\v6.0\Samples\SysMgmt\WindowsPowerShell\...

2. DOS C:\> installutil "GetProcessSample01.dll"

3. Start PowerShell

4. PS> Add-PSSnapin GetProcPSSnapIn01
   (this adds the PowerShell snap-in to the shell)

5. Run your new Cmdlet
   PS> get-proc

6. Put Add-PSSnapin command in your $profile
Agenda

- What is PowerShell
- Getting Started
- Language
- Architecture
  - Execution Engine
  - Pipeline
  - Security Model
- Customization
  - Output Format
  - Personal Settings
  - Developing cmdlets
- Scripting Examples...
- Some tips...
Scripting
Typical Admin Script Cycle...

- **Explore** at the command line
- **Formalize** in production a script
- **Parameterize** in an informal script
- Find the correct **command line(s)** for the job
Scripting with PowerShell

- Script against various sources:
  - Instrumentation: Text, COM, .NET, Cmdlets
  - Object types: WMI, XML, ADO, ADSI
  - Datastores: Filesystem, Registry, Certs, etc

- Ad-hoc scripts need to be quick and easy
  - Arguments don’t need to be named or typed
  - Scripts don’t need to be signed

- Production scripts need to be production quality
  - Arguments should be typed, named and have initializers
  - Rich error handling
  - Protection from uninitialized variables
  - Data can be written to multiple streams
    - Output, Error, Debug, Verbose, Warning, Progress
  - Scripts should be digitally signed
Scripting

- Supports existing scripting models
  - *NIX model of text processing including regular expressions
  - Visual Basic Script model of COM automation
- Supports scripting .NET objects
- Support loose, strong, and extensible typing models
- Uniform syntax to access to a wide range of types
  - WMI, XML, COM, ADSI, ADO
- Rich variable semantics (typed, read-only, constraints, descriptions)
- Rich operators and control structures (Visual C#-like with access to cmds and utilities)
- Functions (positional, named, typed, constrained params)
function MyEcho
{
    for ($i = 0; $i < $args.count; $i++)
    {
        "arg{0} = {1}" -f $i, $args[$i]
    }
}
function Get-Total (  
[String]$property = $(throw "Property Required"),  
[string]$formatString = "Total {1} = {0}"  
)
{
    \{  
        $total = 0  
    \}

    Process  
    {  
        $total += $_.$property  
    }

    \{  
        $formatString -f $total,$property  
    }

}

Arguments **are** declared, typed & initialized
function Get-Types ( [ScriptBlock]$AssemblyFilter = {1}, [ScriptBlock]$TypeFilter = {$_._IsPublic -eq "TRUE"} )
{
    TRAP [System.Exception] {Write-Error "ERROR: $_.Exception"}
    foreach ($a in [AppDomain]::CurrentDomain.GetAssemblies() | where $AssemblyFilter)
    {
        $a.GetTypes() | where $TypeFilter
    }
}
Explicit Object Formatting

- Built-in formatters for lists, tables, wide, and custom-views
  
  PS> Get-Command Format-*

- Formatters allow you to specify properties, propertysets and property expressions

  PS> gps |format-table name,id,handlecount
  PS> gps |format-table Configuration -auto
  PS> gps |format-table name,@{ Expression={$_.mainmodule.filename}; Label="File" }
  PS> gps |format-list name,*size64
Object Utilities

- **Iteration**
  ```powershell
  PS> gps |foreach {"$_.Name): $(($_.Handles)}
  PS> gps |foreach { $l = 0 } { $l += $_.length } { $l }
  ```

- **Grouping**
  ```powershell
  PS> gps |group Company
  ```

- **Measuring**
  ```powershell
  PS> gps |Measure-Object Handles –Sum –Ave –Min –Max
  ```

- **Filtering**
  ```powershell
  PS> gps |Select name,id –expand modules
  PS> gps |Sort handles |Select –first 10
  ```

- **Conditional selection**
  ```powershell
  PS> gps |where {$_.handles –ge 500}
  ```
.NET String class is the foundation
- Clone, CompareTo, Contains, CopyTo, EndsWith, Equals, IndexOf, IndexOfAny, Insert, LastIndexOf, LastIndexOfAny, Length, PadLeft, PadRight, Remove, Replace, Split, StartsWith, Substring, ToCharArray, ToLower, ToLowerInvariant, ToString, ToUpper, ToUpperInvariant, Trim, TrimEnd, TrimStart

Native support for useful datatypes
- REGEX, XML, Arrays, Associative arrays

Rich string operators
- +, *, -, replace, -match, -like, -eq, -ne, gt, -ge, -lt, -le
- Implicit/explicit casting and coercion

Here-strings
```
PS> @”
A
Here-string
“@```
COM Automation

- Bind to COM objects via `New-Object`:
  ```powershell
  PS> $ie = New-Object -Com InternetExplorer.Application
  ```

- Reflect against properties/methods:
  ```powershell
  PS> $ie |get-member
  ```

- Access properties/methods:
  ```powershell
  PS> $ie.Navigate2("www.microsoft.com")
  PS> $ie.Visible=$true
  ```

- Use in pipelines:
  ```powershell
  PS> $word = new-object -Com Word.Application
  PS> $word.RecentFiles | sort name | format-table name,index,path -auto
  ```
Loose

```powershell
PS> $d="4/20/2005"
PS> Function foo() {$args[0]}
```

Strong

```powershell
PS> [datetime]$d="4/20/2005"
PS> function foo([datetime]$date) {$date}
PS> [int] [char] “a”
```

Extensible

```powershell
PS> add-member $d Note Description "MMS TALK"
PS> $d.Description
```

Windows PowerShell works on any .NET type - not a fixed set of “scripting types”
Debugging

Common parameters
PS> get-process -ErrorAction {Continue*, Stop, SilentlyContinue, Inquire}
PS> get-process|stop-process -WhatIf or -Confirm
PS> get-something -Verbose or -Debug

Tracing
PS> start-transcript, stop-transcript
PS> set-PSDebug -Trace <0,1,2>
PS> set-PSDebug -Strict

Break points
PS> $host.EnterNestedPrompt()

$error
PS> $error
PS> get-process -ErrorVariable “myErrors”
Powershell Debugging
Powershell WMI and Event Logs
Powershell Users
A Big User: MySpace*

MySpace is using PowerShell in their production environment to manage thousands of IIS servers

- Using Powershell in production for over a year now

- Increased server count three fold to meet demand in that period
  - No additional admins
  - Attributed to Powershell automation

Partners
Partner Offerings

The following partners have developed products that leverage Windows PowerShell to help with management of Windows or to improve applications running on Windows. Most of these products have a free component that can be used by all Windows administrators.

• **F5 Networks**: Support use of Windows PowerShell to manage F5's BIG-IP network hardware.

• **Full Armor**: Group-Policy Management via Windows PowerShell.

• **/n Software**: Network management, messaging and ability to remotely manage Windows via Windows PowerShell.

• **PowerGadgets**: Visualization product that allows users to run real-time Gadgets such as charts, gauges and maps. Uses Windows PowerShell to allow users or administrators to easily create Windows Vista Sidebar gadgets.

• **PowerGUI by Quest Software**: Graphical user interface for executing Windows PowerShell commands and writing scripts.

• **PowerShell Analyzer by Shelltools**: Graphical User Interface for developers as well as administrators.
Agenda

- What is PowerShell
- Getting Started
- Language
- Architecture
  - Execution Engine
  - Pipeline
  - Security Model
- Customization
  - Output Format
  - Personal Settings
  - Developing cmdlets
- Scripting Examples...
- Some tips...
Version 2 ???

http://microsoft.com/download
Search for "powershell 2.0 ctp"
V2 CTP Features

Any or all of this could change before the V2 RTM. Participate and send your feedback through http://blogs.msdn.com/powershell

• Remoting.
  – 1:n (Fan-out):
    • Admin connects to 1000 server and performs a set of management tasks capturing the results on the client machine.
  – n:1 (Fan-in):
    • Hosting model: Many users connects to one server to perform their management tasks remotely.
  – 1:1 Implicit invocation and Push/Pop-Runspace
    • Admins should be able to interact with the remote machine seamlessly.
  – Async Job management.
    • Admins should be able to manage and schedule background jobs locally and on the remote servers
V2 CTP Features (Continued)

Any or all of this could change before the V2 RTM. Participate and send your feedback through http://blogs.msdn.com/powershell

• Engine.
  – Debugging:
    • New tools to improve debugging such as setting breakpoints, stepping, tracing and examining the session state
  – Modules and Packages
    • Enabling deployment of scripts and cmdlets without the need for registration.
  – SQM
    • Already in LHS.
  – Eventing
V2 CTP Features (Continued)

Any or all of this could change before the V2 RTM. Participate and send your feedback through http://blogs.msdn.com/powershell

• Improved scripting experience
  – Graphical PowerShell.
    • Enhanced interactive shell, a script editing environment, or a combination of both.
  – ScriptCmdlets
    • Enable the creation of cmdlets through scripts
  – Script Internationalization
    • Enable localization support for scripts
  – DatamLanguage mode
    • a safe subset of the PowerShell language for the purpose of script and data sharing
  – Misc items
    • Hash table adapter, Active Directory adapter, Splatting, Closures, Type adapter API, Split & Join ...
  – Performance
  – Cmdlet Cleanup
V2 CTP Features (Continued)

Any or all of this could change before the V2 RTM. Participate and send your feedback through http://blogs.msdn.com/powershell

• Hosting
  – Constrained Runspace:
    • Provide a mechanism for restricting the runspace-configuration entities that are available for invocation to the user based on some rules/configuration.
  – Delegated Admin support

  – Hosting APIs and Runspace pool
    • New API for enabling PowerShell Hosting scenarios and runspace pooling

• Cmdlet Designer tool
Reference Material
PowerShell Community Resources

- Channel 9 tag: http://channel9.msdn.com/tags/Monad
- Team blog: http://blogs.msdn.com/PowerShell/
- OMark van Orsouw’s blog: http://mow001.blogspot.com
- Script Center:
- O’Reilly book – Monad: Introducing the MSH Command Shell and Language
- Manning Press book by Bruce Payette: PowerShell in Action
  - http://manning.com/powershell/
- Sapien Press book by Don Jones: PowerShell
  - http://www.sapienpress.com/
- MshAnalyzer tool
  - http://www.karlprosser.com/coder/?cat=8
More PowerShell Resources

• Technical Chats and Webcasts

• Microsoft Learning and Certification
  – http://www.microsoft.com/learning/default.mspx

• MSDN & TechNet
  – http://microsoft.com/msdn
  – http://microsoft.com/technet

• Virtual Labs

• Newsgroups
  – http://communities2.microsoft.com/
  – communities/newsgroups/en-us/default.aspx

• Technical Community Sites
  – http://www.microsoft.com/communities/default.mspx

• User Groups
Summary

• Download from
  – http://www.microsoft.com/powershell

• Read the bundled documentation

• Join the community
  – News Group
  – The Blog
    http://blogs.msdn.com/powershell
Your potential. Our passion.