Bart De Smet is een onafhankelijk .NET Technical Community Evangelist

Van start met Windows PowerShell

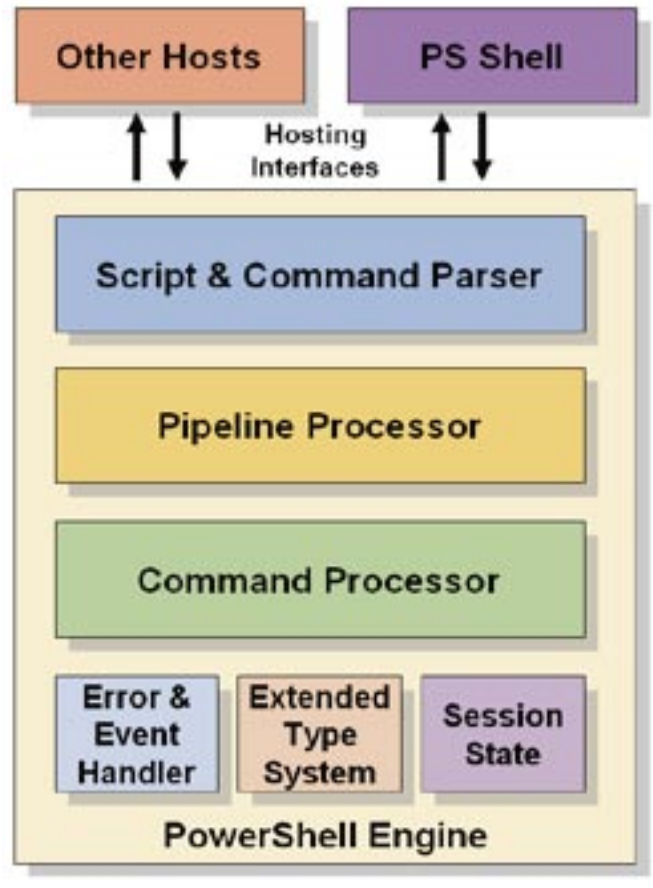
BOUW EEN WORDCOUNT CMDLET

Windows PowerShell, ook bekend onder codenamen 'Monad en MSH', is de volgende generatie commandolijnomgeving voor het Windows-platform. In dit artikel bekijken we hoe je deze omgeving kunt uitbreiden met eigen 'commandlets'.

Windows PowerShell (afgekort PS) is een nieuwe Windows-commandolijn die volledig gebaseerd is op .NET, met als doel de beheerbaarheid van applicaties via de commandolijn te verbeteren. Zo wordt het mogelijk alle UI-gedreven beheertaken van een applicatie ook via de commandolijn toegankelijk – en dus scriptable – te maken. Eén van de eerste Microsoft-producten die volledig 'PS-klaar' zal zijn, is Exchange 12. Samen met MMC 3.0 vormt PowerShell de hoeksteen van applicatie-manageability in de nabije toekomst. Om de shell te downloaden ga je naar http://www.microsoft.com/downloads en zoek je op 'Windows PowerShell'. In dit artikel gebruiken we Windows PowerShell RC1.

Hoe werkt Windows PowerShell?

Monad is gebouwd rond het concept van cmdlets, in feite lichtgewicht commando's. Cmdlets zijn geen uitvoerbare bestanden, maar instanties van .NET-klassen. Een cmdlet zal doorgaans gebruikt worden om achterliggende beheertaken toegankelijk te maken vanuit de shell, zoals het aanmaken van een mailbox. De architectuur van PS is weergegeven in afbeelding 1. Bovenaan vinden we de 'hosting interfaces' die de functionaliteit van PS beschikbaar maken aan programma's. Zo is de powershell.exe één van de programma's die de 'PowerShell Engine' host. De 'PowerShell Engine' bestaat uit een gelaagde architectuur met een parser (waardoor cmdlets niet zelf verantwoordelijk zijn voor parsing), een pipeline processor om cmdlets aan elkaar te rijgen en een command processor die de uitvoering van cmdlets regelt. Dit alles wordt verder ondersteund via een aantal services zoals foutbehandeling, sessiebeheer en een 'Extended Type System' dat toegang tot properties en methoden op objecten regelt.



Afbeelding 1. PowerShell-architectuur

Cmdlets werken met objecten in plaats van strings in een klassieke commandolijn. Zo zal de cmdlet get-process instanties van het type System.Diagnostics.Process teruggeven. De teruggegeven objecten kunnen via een pipeline doorgegeven worden aan een andere cmdlet, bijvoorbeeld om de resultaten te filteren en te sorteren, of om administratieve acties te ondernemen. Een paar voorbeelden zijn weergegeven in afbeelding 2.

Naast cmdlets zijn er ook providers. Deze vormen de interface naar een hiërarchische gegevensbron, zoals het bestandssysteem, maar bijvoorbeeld ook het register, Active Directory, een database en noem maar op. Ooit gedacht dat je met het commando cd door om
het even welke databron zou kunnen wandelen? Neem een kijkje

```
PS C:\> get-process | sort-object -property CPU -desc | select-object -
        First 5

Handles  NPM(K)    PM(K)      WS(K) VS(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    846      24    57456      82336   307   139.27    888 devenv
    461      18    34784      62352   472    95.50   6104 WINWORD
   2255       0        0        240     2    80.06      4 System
    900      24    32792      46980   146    74.44   1972 explorer
    601      27    31076      25516   124    45.91   5824 msnmsgr

PS C:\> $p = get-process | where { $_.ProcessName -eq "notepad" }
PS C:\> $p.Kill()

PS C:\ > get-process ex* | stop-process -whatif
What if: Performing operation "stop-process" on Target "explorer (1972)".
```

Afbeelding 2. Een aantal cmdlets in actie

```
PS C:\> get-psdrive

Name       Provider      Root                    CurrentLocation
----       --------      ----                     ---------------
Alias      Alias
C          FileSystem    C:\
cert       Certificate   \
Env        Environment
Function   Function
HKCU       Registry      HKEY_CURRENT_USER
HKLM       Registry      HKEY_LOCAL_MACHINE
Variable   Variable
X          FileSystem    X:\

PS C:\> cd HKLM:\SOFTWARE\Microsoft\.NETFramework
PS HKLM:\SOFTWARE\Microsoft\.NETFramework> dir v*

   Hive: Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework

SKC  VC Name                           Property
---  -- ----                           --------
  1   0 v1.0                          {}
  1   0 v1.0.3705                     {}
  1   0 v1.1.4322                     {}
  2   0 v2.0.50727                    {}
```

Afbeelding 3. Werken met providers

```
PS C:\Program Files\Windows PowerShell\v1.0> make-shell
          -out demo –ns MSDN.Demos -ref wordcount.dll
Windows(R) PowerShell MakeKit
Copyright (C) 2005 Microsoft Corporation. All rights reserved.

Shell demo.exe is created successfully.
PS C:\Program Files\Windows PowerShell\v1.0>
```

Afbeelding 4. Een nieuwe shell bouwen met make-shell

in afbeelding 3 voor een voorbeeld.
Om een globaal overzicht te krijgen van de mogelijkheden van PS verwijzen we naar het 'Windows PowerShell RC1 Documentation Pack' op www.microsoft.com/downloads.

Een eigen cmdlet bouwen: waar te beginnen?

In dit artikel concentreren we ons op het bouwen van een eigen cmdlet die de functionaliteit van het commando wc uit UNIX naar PS brengt. Het commando wc wordt gebruikt om het aantal regels, woorden en karakters uit bestanden te tellen. We beginnen met het bouwen van een skelet voor onze cmdlet. Maak in Visual Studio 2005 een nieuw C# Class Library-project aan met de naam 'Wordcount' en voeg een referentie toe naar System.Management.Automation.dll in %programfiles%\Windows PowerShell\v1.0\. Dit

```
using System;
using System.IO; // Nodig voor het vervolg.
using System.Management.Automation;

[Cmdlet("Get", "Wordcount")]
public class WordCount : PSCmdlet
{

}
```

Codevoorbeeld 1.

```
<Target Name="AfterBuild">
  <Copy SourceFiles="$(OutDir)\WordCount.dll"
    DestinationFolder="C:\Program Files\Windows PowerShell\v1.0" />
</Target>
```

Codevoorbeeld 2.

```
PS C:\Program Files\Windows PowerShell\v1.0> cd \ #korter pad voor bladschikking
PS C:\> $NEWSHELLID = "MSDN.Demos.demo"
PS C:\> $REGPATH = "HKLM:/software/microsoft/powershell/1/shellids/
$NEWSHELLID"
PS C:\> mkdir $REGPATH

   Hive: Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\
software\microsoft\powershell\1\shellids

SKC  VC Name                           Property
---  -- ----                           --------
  0   0 MSDN.Demos.demo                {}

PS C:\> new-itemproperty -path $REGPATH -name Path -value "C:\Program
Files\Wind
ows PowerShell\v1.0\demo.exe"

PSPath       : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\
               software\microsoft\powershell\1\shellids\MSDN.Demos.demo
PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\
               software\microsoft\powershell\1\shellids
PSChildName  : MSDN.Demos.demo
PSDrive      : HKLM
PSProvider   : Microsoft.PowerShell.Core\Registry
Path         : C:\Program Files\Windows PowerShell\v1.0\demo.exe
```

Afbeelding 5. De shell registreren in het register

doe je via het menu Project, Add Reference, tabblad Browse.
Tijd om code te schrijven. Hernoem Class1.cs naar WordCount.cs en vervang de code door codevoorbeeld 1.
Het definiëren van een cmdlet bestaat erin van de basisklasse PSCmdlet en het attribuut Cmdlet toe te passen. Dit attribuut bevat twee parameters, een verb en een noun. In PowerShell worden deze door een koppelteken van elkaar gescheiden bij het oproepen van de cmdlet. In ons geval wordt dit dus get-wordcount.
Voordat we het project kunnen compileren, dienen we ervoor te zorgen dat de gecompileerde assembly naar de PS-installatiemap wordt gekopieerd zodat we gemakkelijk kunnen testen. Je zou dit steeds handmatig kunnen doen, maar door aanpassing van het MSBuild- projectbestand is dit gemakkelijk te automatiseren. Bewaar alle bestanden in Visual Studio, open het WordCount.csproj-bestand in Notepad en wijzig de target 'AfterBuild' onderaan het bestand, als volgt:

Controleer of de target zich niet binnen XML-commentaar-tekens bevindt (<!-- ... -->) en dat het DestinationFolder-pad correct is ingesteld. Bouw vervolgens het project en controleer of de WordCount.dll-assembly naar de PS-installatiemap is gekopieerd. Hiermee heb je een eerste (niet-functionele) cmdlet gebouwd. De code vullen we later in, maar eerst bouwen we een shell die onze cmdlet zal hosten. Start Windows PowerShell op via het startmenu of via powershell.exe en ga naar de installatiemap via cd. Bouw nu een nieuwe shell via make-shell zoals

```
private string[] files;

/// <summary>
/// Lijst met bestanden waarop "wordcount" uitgevoerd zal worden.
/// </summary>
[Parameter(Mandatory = true,
           Position = 0,
           ValueFromPipelineByPropertyName = true)]
public string[] FullName
{
  get { return files;  }
  set { files = value; }
}
```

Codevoorbeeld 3.

```
/// <summary>
/// Doe lijn-, woord- en karakter-telling.
/// </summary>
protected override void ProcessRecord()
{
  //
  // Resultaat-object dat zal worden teruggegeven.
  //
  WordCountResult res = new WordCountResult();
  //
  // Bestanden waarvoor telling wordt uitgevoerd.
  //
  res.Files = files;
  //
  // Behandel elk van de opgegeven 'bestanden'.
  //
  foreach (string file in files)
  {
    ProviderInfo provider;
    //
    // Ondersteuning voor wildcard-syntax.
    //
    foreach (string path in
          GetResolvedProviderPathFromPSPath(file, out provider))
    {
      //
      // Ondersteuning voor -whatif en -confirm opties.
      //
      if (ShouldProcess("Perform count on " + path))
      {
        //
        // Doe de eigenlijke verwerking (hulpmethode).
        //
        Process(path, ref res);
        //
        // Ondersteuning voor -verbose optie.
        //
        WriteVerbose(path + " " + res);
      }
    }
  }
  //
  // Rapporteer het resultaat.
  //
  WriteObject(res);
}
```

Codevoorbeeld 4.

weergegeven in afbeelding 4.

Make-shell is de cmdlet voor de MakeKit-tool waarmee nieuwe shell-executables kunnen worden gebouwd. Op deze manier kun je bijvoorbeeld een speciale shell bouwen voor het beheer van een serverapplicatie. Onze oproep naar make-shell bouwt een nieuwe shell demo.exe in de (shell-)namespace MSDN.Demos en refereert naar onze wordcount.dll-assembly. Om de nieuwe shell correct te laten werken, dienen we deze eerst te registeren in het register. We kunnen dit rechtstreeks in PS uitvoeren dankzij de registry-provider; zie afbeelding 5.

Ga nu via Verkenner naar de PS-installatiemap en start demo.exe op. In deze shell wordt get-wordcount reeds herkend, hoewel het resultaat nog niet bijster interessant is:

```
PS C:\Program Files\Windows PowerShell\v1.0> get-wordcount
PS C:\Program Files\Windows PowerShell\v1.0>
```

Sluit deze shell terug af. Tijd voor het echte werk: de cmdlet parametriseren. We beginnen met het definiëren van de parameters voor onze cmdlet; zie codevoorbeeld 3.

Voor de werking van onze cmdlet hebben we slechts één parameter nodig waarbij alle te behandelen bestanden in een array zijn opgenomen. Parameters worden als properties gedeclareerd en gemarkeerd via een Parameter-attribuut. Via de properties van het Parameter-attribuut stellen we in dat de FullName-parameter verplicht is (Mandatory = true) en dat het de eerste parameter (Position = 0) is. Via ValueFromPipelineByPropertyName wordt pipelining mogelijk gemaakt. Hierdoor kunnen we de uitvoer van get-childitem (of kortweg dir) als invoer gebruiken voor onze cmdlet. De get-childitem cmdlet geeft als uitvoer objecten van het type System.IO.FileInfo terug. Dit type bevat een property FullName met de volledige bestandsnaam, inclusief het pad. Door onze parameter ook FullName te noemen, kan PS de nodige pipelining verzorgen. Tip: voer eens onderstaand commando uit.

```
PS C:\> get-childitem *.* | get-member –MemberType Properties
```

Nu is het tijd voor de eigenlijke implementatie. Deze bevindt zich in de methode ProcessRecord die opgeroepen wordt voor elk 'record' dat behandeld moet worden. Bij een alleenstaande aanroep naar get-wordcount wordt deze methode slechts éénmaal opgeroepen. Is de cmdlet echter geschakeld in een pipelinestructuur, dan wordt deze voor elk doorgegeven object opgeroepen.

Deze code voert de volgende taken uit. Eerst wordt een object geconstrueerd dat teruggegeven zal worden. De definitie van het

```
/// <summary>
/// Voorstelling van een resultaat geproduceerd door get-wordcount.
/// </summary>
public class WordCountResult
{
  /// <summary>
  /// Het aantal gevonden lijnen.
  /// </summary>
  public int Lines;

  /// <summary>
  /// Het aantal gevonden woorden.
  /// </summary>
  public int Words;

  /// <summary>
  /// Het aantal gevonden karakters.
  /// </summary>
  public int Characters;

  /// <summary>
  /// Bestanden waarvoor telling werd uitgevoerd.
  /// </summary>
  public string[] Files;

  /// <summary>
  /// Gebruiksvriendelijke tekstuele representatie van
  /// telresultaat.
  /// </summary>
  /// <returns>
  /// String met het aantal lijnen, woorden en karakters.
  /// </returns>
  public override string ToString()
  {
    return Lines + " " + Words + " " + Characters;
  }
}
```

Codevoorbeeld 5. Het resultaatobject gedefinieerd

```
private void Process(string file, ref WordCountResult res)
{
  //
  // Gebruik IDisposable-patroon voor veilig gebruik van resources.
  //
  using (FileStream fs = new FileStream(file, FileMode.Open,
                                        FileAccess.Read))
  using (StreamReader reader = new StreamReader(fs))
  {
    for (string line; (line = reader.ReadLine()) != null;
        res.Lines++)
    {
      bool firstWordFound = false;
      bool inWord = false;

      foreach (char c in line)
      {
        res.Characters++;

        if (!firstWordFound && !char.IsWhiteSpace(c))
        {
          firstWordFound = true;
          inWord = true;
          res.Words++;
        }

        if (firstWordFound)
        {
          if (inWord && char.IsWhiteSpace(c))
            inWord = false;

          if (!inWord && !char.IsWhiteSpace(c))
          {
            inWord = true;
            res.Words++;
          }
        }
      }
    }
  }
}
```

Codevoorbeeld 6. Een helper-methode om de telling uit te voeren

WordCountResult-type is terug te vinden in codevoorbeeld 5.

Vervolgens wordt elk 'bestand' uit de files-array behandeld. Vermits de gebruiker met wildcards kan werken, is een omzetting naar concrete paden nodig om de achterliggende bestanden in te lezen. Dit gebeurt in een geneste foreach-lus die een lijst van te behandelen bestanden verkrijgt via een oproep naar GetResolvedProviderPathFromPSPath uit de basisklasse PSCmdlet. Hierna wordt gecontroleerd of het gevonden item effectief behandeld dient te worden via een oproep naar ShouldProcess. Indien de cmdlet opgeroepen wordt met de parameter –confirm, zal PS voor elk item de gebruiker om bevestiging vragen. Bij het gebruik van –whatif zal PS zélf telkens "Nee" antwoorden op deze bevestiging. Om dit alles mogelijk te maken dient het Cmdlet-attribuut als volgt aangepast te worden:

```
[Cmdlet("Get", "Wordcount", SupportsShouldProcess = true)]
```

Uiteindelijk wordt een helpermethode (zie codevoorbeeld 6) opgeroepen om het eigenlijke werk te verrichten en het resultaatobject in te vullen.

De oproep naar WriteVerbose wordt gebruikt om de verbose-optie te ondersteunen, wat handig is om meer informatie te krijgen over wat het commando aan het doen is. Vervolgens wordt het Word-

```
PS C:\Program Files\Windows PowerShell\v1.0> get-wordcount about_Alias.help.txt
  Lines          Words        Characters Files
  -----          -----        ---------- -----
  155        1038          6657 {about_Alias.hel...
```

Afbeelding 5. Onze wordcount-cmdlet in actie

```
PS C:\Program Files\Windows PowerShell\v1.0> $count = wc about_Alias.help.txt
PS C:\Program Files\Windows PowerShell\v1.0> $count

  Lines               Words          Characters Files
  -----               -----          ---------- -----
    155               1038                6657 {about_Alias.hel...

PS C:\Program Files\Windows PowerShell\v1.0> "Aantal woorden: " +
                                              $count.Words
Aantal woorden: 1038
```

Afbeelding 6. Alias aanmaken en aanroepen

```
PS C:\Program Files\Windows PowerShell\v1.0> wc *.txt

  Lines               Words          Characters Files
  -----               -----          ---------- -----
   6665               37629              264009 {*.txt}
```

Afbeelding 7. Wordcount cmdlet met wildcards

```
PS C:\Program Files\Windows PowerShell\v1.0> get-childitem *.txt | wc

  Lines               Words          Characters Files
  -----               -----          ---------- -----
    155                1038                6657 {C:\Program File...
     76                 489                3349 {C:\Program File...
    249                1610                9913 {C:\Program File...
```

Afbeelding 8. Wordcount cmdlet met wildcards

```
PS C:\Program Files\Windows PowerShell\v1.0> get-item *.txt | wc |
foreach { $x = 0 } { $x += $_.Words } { "Total words: " + $x }
Total words: 37629
```

Afbeelding 9. ForEach-lusconstructie

```
PS C:\Program Files\Windows PowerShell\v1.0> get-item *.txt | wc |
foreach { $x = new-object WordCountResult } { $x.Words += $_.Words;
$x.Characters += $_.Characters; $x.Lines += $_.Lines } { "Total: " + $x }
Total: 6665 37629 264009
```

Afbeelding 10. WordCountResult-object

```
PS C:\Program Files\Windows PowerShell\v1.0> get-childitem *.txt | wc |
foreach { $x = new-object WordCountResult } { $x = $x + $_ } { $x }

  Lines               Words          Characters Files
  -----               -----          ---------- -----
   6665               37629              264009 {C:\Program File...
```

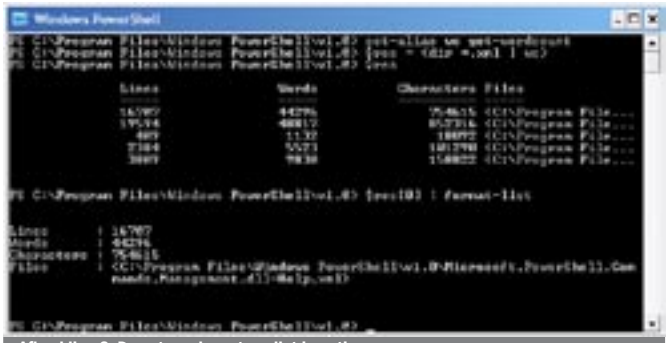
Afbeelding 11. WordCountResult-object met operator overload

CountResult-object aan de gebruiker teruggegeven via WriteObject waarna het afgedrukt kan worden op het scherm of doorgegeven kan worden in de pipeline.

Onze cmdlet in actie

Als alles goed loopt, is onze cmdlet nu na compilatie volledig functioneel. Tijd om te compileren en demo.exe op te starten. In de installatiemap van PowerShell staan een heleboel txt-bestanden met informatie over diverse cmdlets. Laten we even onze get-wordcount cmdlet hierop loslaten; zie afbeelding 5. Om het typewerk te verlichten zullen we eerst een alias creëren via set-alias:

```
PS C:\Program Files\Windows PowerShell\v1.0> set-alias wc get-wordcount
```



Afbeelding 8. De get-wordcount cmdlet in actie

Nu kunnen we ook gewoon wc gebruiken om de cmdlet op te roepen. Om je ervan te vergewissen dat get-wordcount effectief een object teruggeeft, probeer je het volgende zoals dat in afbeelding 6 is te zien. En wat met wildcards? Een voorbeeld hiervan in afbeelding 7. Met deze oproep wordt get-wordcount slechts éénmaal opgeroepen met parameter *.txt. Uiteindelijk wordt dus slechts één WordCountResult-object teruggegeven in dit geval. Bij het gebruik van pipelining ligt de situatie anders, zie afbeelding 8. Nu geeft get-childitem telkens een object terug dat netjes wordt doorgegeven aan de get-wordcount cmdlet. Hierdoor worden bestanden één voor één behandeld. Veronderstel dat we nu ook het totale aantal woorden willen tellen voor alle bestanden tezamen, dan kunnen we de foreach-lusconstructie uit PS inschakelen; zie afbeelding 9. Of waarom geen gebruik maken van een WordCountResult-object? Zie hiervoor afbeelding 10. Je zou zelfs nog een stap verder kunnen gaan en een operator overload voor + kunnen definiëren in WordCountResult. Het gebruik zou er dan als in afbeelding 11 kunnen gaan uitzien. De code die je kunt downloaden voor dit artikel bevat deze uitbreiding. Tot slot kun je ook nog eens de volgende commando's uitproberen om het effect van de aanroepen naar ShouldProcess en WriteVerbose in actie te zien:

```
wc *.txt -whatif
wc *.txt –confirm
wc *.txt -verbose
```

Beter beheer

Naast het bouwen van generieke cmdlets zoals get-wordcount zijn applicatiespecifieke cmdlets niet te versmaden om de beheerbaarheid van een applicatie te verhogen. Vanaf vandaag kun je je al voorbereiden op deze evolutie door goede interfaces te definiëren voor je managementklassen. Later kun je deze dan ook via PowerShell toegankelijk maken in de vorm van cmdlets en voorgoed vriendschap sluiten met de collega's van de IT-afdeling die de applicaties dan naar hartenlust via scripts kunnen beheren en automatiseren in de productieomgeving.

Bart De Smet is een onafhankelijk .NET Technical Community Evangelist uit België en sinds 2003 MVP voor Visual C#. Bart's blog is te vinden op http://blogs.bartdesmet.net/bart. Voor vragen of opmerkingen kan je mailen naar info@bartdesmet.net.

Referenties

blogs.msdn.com/monad
www.reskit.net/monad
www.microsoft.com/technet/scriptcenter/hubs/msh.mspx
channel9.msdn.com/wiki/default.aspx/Channel9.MSHQuickStart
www.microsoft.com/technet/scriptcenter/topics/msh/download.mspx