WINDOWS POWERSHELL 4.0 EXAMPLES

Created by http://powershellmagazine.com

PowerShellMagazine

How to schedule a job

Job scheduling allows you to schedule execution of a PowerShell background job for a later time. First thing you do is create a job trigger. This defines when the job will execute. Then you use the Register-ScheduledJob cmdlet to actually register the job on the system. In the following example, every day at 3am we back up our important files:

```
$trigger = New-JobTrigger -Daily -At 3am
Register-ScheduledJob -Name DailyBackup -Trigger $trigger -ScriptBlock {Copy-Item
c:\ImportantFiles d:\Backup$((Get-Date).ToFileTime()) -Recurse -Force -PassThru}
```

Once the trigger has fired and the job has run, you can work with it the same way you do with regular background jobs:

```
Get-Job -Name DailyBackup | Receive-Job
```

You can start a scheduled job manually, rather than waiting for the trigger to fire:

```
Start-Job -DefinitionName DailyBackup
```

The RunNow parameter for Register-ScheduledJob and Set-ScheduledJob eliminates the need to set an immediate start date and time for jobs by using the -Trigger parameter:

```
Register-ScheduledJob -Name Backup -ScriptBlock {Copy-Item c:\ImportantFiles d:\
Backup$((Get-Date).ToFileTime()) -Recurse -Force -PassThru} -RunNow
```

How to generate complex password

There is no built-in cmdlet to generate a password, but we can leverage a vast number of .NET Framework classes. System.Web.Security.Membership class has a static method GeneratePassword(). First, we need to load an assembly containing this class, and then we use GeneratePassword() to define password length and a number of non-alphanumeric characters.

```
$Assembly = Add-Type -AssemblyName System.Web
[System.Web.Security.Membership]::GeneratePassword(8,3)
```

How do we know all that? With a little help from the Get-Member cmdlet (and simplified where syntax):

```
[System.Web.Security.Membership]  | Get-Member -MemberType method -Static|
where name -match password
```

PipelineVariable Common Parameter

In PowerShell 4.0, the PipelineVariable lets you save the results of a piped command (or part of a piped command) as a variable that can be passed through the remainder of the pipeline. Its alias is "pv".

```
Get-ChildItem *.ps1 -pv a | Get-Acl | foreach {"{0} size is {1}" -f $a.Name, $a.length}
```

Diagnosing network connectivity

The Test-NetConnection cmdlet (available on Windows 8.1 / Server 2012 R2) displays diagnostic information for a connection.

```
#Returns detailed diagnostic information for microsoft.com
Test-NetConnection -ComputerName microsoft.com -InformationLevel Detailed

#Returns a Boolean value based on connectivity status
Test-NetConnection -ComputerName microsoft.com -Port 80 -InformationLevel Quiet
```

Working with a file hash

The new Get-FileHash cmdlet computes the hash value for a file. There are multiple hash algorithms supported with this cmdlet.

```
Get-FileHash C:\downloads\WindowsOS.iso -Algorithm SHA384
```

Dynamic method invocation

Windows PowerShell 4.0 supports method invocation using dynamic method names.

```
$fn = "ToString"
"Hello".ToString()
"Hello".$fn()
"Hello".("To" + "String")()

#Select a method from an array:
$index = 1; "Hello".(@("ToUpper", "ToLower")[$index])()
```

#Requires -RunAsAdministrator

Starting with PowerShell 4.0, we can specify that a script requires administrative privileges by including a #Requires statement with the -RunAsAdministrator switch parameter.

```
#Requires -RunAsAdministrator
```

How to parse webpage elements

The Invoke-WebRequest cmdlet parses the response, exposing collections of forms, links, images, and other significant HTML elements. The following example returns all the URLs from a webpage:

```
$iwr = Invoke-WebRequest http://blogs.msdn.com/b/powershell
$iwr.Links
```

The next example returns an RSS feed from the PowerShell team blog:

```
$irm = Invoke-RestMethod blogs.msdn.com/b/powershell/rss.aspx
$irm | Select-Object PubDate, Title
```

How to fetch data exposed by OData services

The Invoke-RestMethod cmdlet sends an HTTP(S) request to a REST-compliant web service. You can use it to consume data exposed by, for example, the OData service for TechEd North America 2014 content. Put information about TechEd NA 2014 sessions into a $sessions variable. Let's iterate through that collection of XML elements and create custom PowerShell objects using [PSCustomObject]. Pipe the output to an Out-GridView command specifying the PassThru parameter to send selected items from the interactive window down the pipeline, as input to an Export-Csv command. (While you are in a grid view window, try to filter only PowerShell Hands-on Labs, for example). If you have Excel installed the last command opens a CSV file in Excel.

```
$sessions = Invoke-RestMethod https://odata.eventpointdata.com/tena2014/
Sessions.svc/Sessions

$sessions | ForEach-Object {
   $properties = $_.content.properties
   [PSCustomObject]@{
      Code        = $properties.Code
      Day         = $properties.Day_US
      StartTime   = $properties.Start_US
      EndTime     = $properties.Finish_US
      Speaker     = $properties.PrimarySpeaker
      Title       = $properties.Title
      Abstract    = $properties.Abstract
   }
} | Out-GridView -PassThru | Export-Csv $env:TEMP\sessions.csv -NoTypeInformation

Invoke-Item $env:TEMP\sessions.csv
```

How to set custom default values for the parameters

If you, for example, need to execute one of the -PSSession cmdlets using alternate credentials you must specify the credentials for the Credential parameter every time you call the command. In PowerShell 3.0 and later, we can supply alternate credentials by using new $PSDefaultParameterValues preference variable. For more information, see about_Parameters_Default_Values.

```
$PSDefaultParameterValues = @{'*-PSSession:Credential'=Get-Credential}
```

Save-Help

In Windows PowerShell 3.0, Save-Help worked only for modules that are installed on the local computer. PowerShell 4.0 enables Save-Help to work for remote modules.

Use Invoke-Command to get the remote module and call Save-Help:

```
$m = Invoke-Command -ComputerName RemoteServer -ScriptBlock { Get Module -
Name DhcpServer -ListAvailable }
Save-Help -Module $m -DestinationPath C:\SavedHelp
```

Use a PSSession to get the remote module and call Save-Help:

```
$s = New-PSSession -ComputerName RemoteServer
$m = Get-Module -PSSession $s -Name DhcpServer -ListAvailable
Save-Help -Module $m -DestinationPath C:\SavedHelp
```

Use a CimSession to get the remote module and call Save-Help:

```
$c = New-CimSession -ComputerName RemoteServer
$m = Get-Module -CimSession $c -Name DhcpServer -ListAvailable
Save-Help -Module $m -DestinationPath C:\SavedHelp
```

How to access local variables in a remote session

In PowerShell 2.0, if you had some local variable that you wanted to use when executing a script block remotely, you had to do something like:

```
$localVar = 42
Invoke-Command -ComputerName Server 1 { param($localVar) echo $localVar } -
ArgumentList $localVar
```

In PowerShell 3.0 and later, you can use Using scope (prefix variable name with $using:):

```
Invoke-Command -ComputerName Server1 { echo $using:localVar }
```