<http://professorramos.com/Materiais/Documentos/PowerShell%20para%20IT%20Pro-%20Book.pdf>

O Windows PowerShell introduz o conceito de cmdlet (pronuncia-se "command-let"), uma ferramenta de linha de comando simples, de função única e compilada no shell.

Você também pode fazer o uso do ISE (Integrated Scripting Environment ) onde você pode executar comandos, gravar, testar e depurar scripts em uma interface de usuário gráfica baseada no Windows.

**PowerShell ISE**

Você pode optar por carregar o PowerShell ISE (Integrated Scripting Environment), um ambiente de programação do PowerShell que facilita o desenvolvimento de scripts, pois você pode executar comandos, gravar, testar e depurar scripts em uma interface de usuário gráfica baseada no Windows. Para isso clique em Start \ All Programs \ Accessories \ Windows PowerShell Windows PowerShell ISE.

**Painel de digitação**

O Painel de digitação nos ajuda a digitar os comandos, pois faz uso do IntelliSense uma tecnologia Microsoft que nos ajuda a completar os comandos, parâmetros e até mesmo as variáveis que você tenha criado

Existem quatro categorias de comandos do PowerShell:

* Cmdlet (Command-Lets)
* Funções PowerShell
* Scripts PowerShell
* Comandos nativos do Windows, e ainda comandos do Linux.

**Command-Lets**

Os CMDLETS (pronunciado command-lets) é um comando. Uma pequena unidade de funcionalidade relacionada a um conjunto de recursos. Você pode usar os CMDLETS da mesma maneira que usa um comando ou utilitário. Os CMDLETS não são case sensitive. Os CMDLETS usam uma convenção simples de nomes, baseada em Verbo-Substantivo. Por exemplo, o verto get que deve ser seguido de um substantivo, por exemplo, date. Na linha de comando podemos digitar então get-date e o resultado será apresentado na tela: Os verbos do PS nem sempre são verbos em inglês, mas expressam ações específicas. Os substantivos são muito parecidos aos de qualquer idioma, uma vez que descrevem tipos específicos de objetos importantes na administração do sistema.

get-command -commandtype cmdlet

**Show-Command**

Na dúvida sobre como completar os parâmetros de determinado cmdlet, você pode carregar o painel de comandos mesmo no modo de linha de comando.O painel de comandos é uma janela gráfica muito similar ao painel de comandos do PowerShell ISE e permite a visualização dos parâmetros opcionais e obrigatórios dos cmdlets. Para isso você pode usar o cmdlet Show-Command.

**Help Poderoso**

Update-Help

Depois basta fazer o uso do help que pode ser bem simplificado com os cmdlets abaixo:

1. Get-Help - Exibe o help no console
2. Get-Help -Online -Exibe o help online na biblioteca do TechNet Digamos que eu queira saber mais sobre “ForEach-Object”
3. Get-Help ForEach-Object Get-Help ForEach-Object -Online – Acessa os recursos online
4. Get-Help ForEach-Object -Examples – Exibe exemplos do comando
5. Get-Help ForEach-Object -Detailed – Exibe um help datlhado
6. Get-Help ForEach-Object -ShowWindow - Exibe uma janela

**Funções**

O comando Clear-Host não é um cmdlet, porém possui o mesmo modelo de verbosubstantivo. O comando Clear-Host é, na verdade, uma função interna. Para listar as funções utilize o comando get-command -commandtype function

**Alias**

Alias são como apelidos para os cmdlets e funções: Por exemplo, podemos usar o comando clear-host para limpar a tela, porém existe o alias chamado clear que executa o clear-host. Liste todos os Alias com o seguinte comando: get-command -commandtype alias

Um bom exemplo de Alias é para listagem de diretórios e você pode usar qualquer um dos Alias abaixo:

LS – UNIX DIR

– MS-DOS Get-ChildItem

– PowerShell

Como criar um alias?

Set-Alias Dia Get-Date

O comando acima criar um alias chamado Dia para o cmdlets Get-Date.

**Controlando a exibição (saída) de informações**

As informações que você pode coletar através do Windows Power Shell pode ser formatada de modo que facilite a visualização das informações. Um dos cmdlets que nós administradores sempre precisamos executar é o Get-Process, pois lista os processos em execução em nosso servidor ou estação: Get-Process

Usado o pipe ( | ) podemos passar a saída do comando para diversas opções. O pipe é um operador. Cada comando após o pipe recebe um objeto do comando anterior, realiza alguma operação no objeto, e depois passa adiante para o próximo comando no pipeline.

1. Get-Process | more
2. Get-Process | Format-List
3. Get-Process | Format-List | more 13
4. Get-Process | ConvertTo-HTML | Out-File "Processos.html"
5. Get-Process | Export-CSV "Processos.csv"

**Detalhes dos cmdlets out.**

Alguns cmdlets existentes criam saídas incríveis, são os casos do cmdlets out. Para listar os cmdlets “out”: Get-command out\*

1. Out-Default - Envie a saída para o formatador padrão e o cmdlet de saída padrão.
2. Out-File - Envia a saída para um arquivo
3. Out-GridView - Envia a saída para uma tabela interativa em uma janela separada
4. Out-Host - Envia a saída para a linha de comando.
5. Out-Null - Apaga saída, em vez de enviá-lo para o console
6. Out-Printer - Envia a saída para uma impressora..
7. Out-String – Envia a saida pra um seria de strings

Exemplos:

Get-Process | Out-GridView

Get-Process | out-file -filepath C:\Test1\processos.txt

Também podemos usar o bom e velho redirecionador que é o sinal de “maior que” > para criar e gravar no arquivo e usar duas vezes o comando >> para adicionar informações no fim do arquivo já existente.

Exemplo

Get-Process > teste1.txt

Get-Alias >> teste1.txt

**Filtrando resultados na tela (Where-Object)**

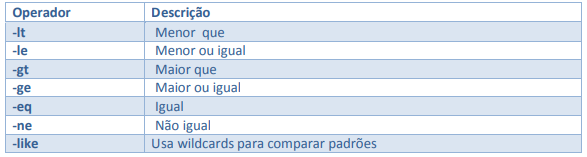
O cmdlet Where-Object fornece a capacidade de criarmos filtros específicos no retorno de outros cmdlets. Como você já deve ter percebido, alguns cmdlets exibem na tela todos os dados de determinado objeto ou recurso, como por exemplo o cmdlet Get-Service trará na tela todos os serviços estando iniciados e parados.

Com o Where-Object você pode criar um filtro e trazer apenas os serviços em execução.

get-service | Where-Object {$\_.Status -eq "Running"}

A estrutura para o cmdlet Where-Object é {\_.Campo operador valor}

Os operadores no PowerShell são:



Cada cmdlet exibe na tela diferentes resultados, portanto no momento de usar Where-Object você deve conhecer o resultado padrão e analisar quais são os nomes dos campos que deseja utilizar como campo. Os operadores lógicos serão abordados novamente mais adiante. No exemplo abaixo foi executado o cmdlet Get-ChildItem e podemos notar que existem 4 campos.

Podemos então fazer um filtro com where-object {$\_.Name -like "Windows"}

**Módulos**

Dependendo das configurações do seu servidor, você terá determinados módulos disponíveis. Se você instalar Roles e Features aumenta o seu arsenal de cmdlets relacionado aos módulos existentes. Nas versões anteriores do PowerShell, no Windows Server 2008 R2 por exemplo, você não poderia fazer a instalação de um recurso sem antes carregar o modulo ServerManager.

**Listando módulos**

Para lista os módulos do seu servidor ou estação execute o cmdlets abaixo.

Get-Module -ListAvailable

Após listar os módulos disponíveis no sistema é possível fazer uso de alguns deles, devemos usar o comando.

Get-Command -Module nomeModulo

Exemplo

Get-Command -Module NetTCPIP

E uma lista dos cmdlets específicos para esse modulo será listada. Isso é extremamente importante, pois temos muitos e muitos cmdlets para cada modulo.

**Importando módulos**

A importação de módulos no PowerShell é simples e rápida.

Import-Modulo NomeDoModulo

**Scripts no PowerShell**

Para criar um script inicial, o famoso “Hello Word” você deve utilizar um editor de texto ou o próprio PowerShell ISE que é o mais recomendado.

A extensão para execução de scripts no PowerShell é .PS1. Usando o editor de textos basta criar um arquivo e salvar como nomedesejado.ps1 A vantagem de fazer uso de scripts PowerShell é criar ferramentas poderosas de administração ou de automação de tarefas cotidianas.

Para executar um script PowerShell basta digitar o nome do script utilizando ./ na frente. Exemplo: ./script.ps1

Alguns ambientes podem não permitir a execução de scripts por motivos de segurança. Para habilitar a execução de scripts você deve definir uma política de execução com o comando: set-executionpolicy

O cmdlet Set-ExecutionPolicy permite determinar como os scripts serão permitidos para execução. Windows PowerShell tem quatro diferentes políticas de execução:

* Restricted – Nenhum script pode ser executados. Windows PowerShell pode ser usado apenas no modo interativo.
* AllSigned - Somente scripts assinados por um fornecedor confiável pode ser executado
* RemoteSigned - os scripts baixados devem ser assinados por um fornecedor confiável antes que eles possam ser executados.
* Unrestricted - Sem restrições, todos os scripts do Windows PowerShell pode ser executado.

Então para que possamos executar os scripts que criarmos devemos habilitar a política de execução irrestrita:

Set-ExecutionPolicy Unrestricted

Após definirmos um modo de execução de scripts, vamos criar o nosso primeiro script do PowerShell.

**Estrutura de um script**

A estrutura de um Script varia muito de acordo com o desenvolvedor do mesmo. Mas quase sempre seguirá um padrão como Comentários são extremamente importantes, tanto para você que criou o script como para aquele que irá executar. Os comentários são iniciando com o símbolo # e quando você usa o PowerShell ISE eles ficam em verde.

# Isso é um comentário importante sobre o scripts.

Geralmente após os comentários vem as declarações, pois devem estar no início do script.

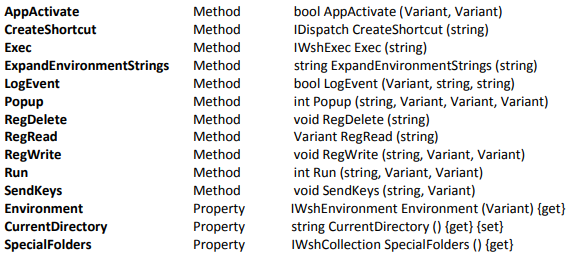
$variavel = "valor qualquer que ficará armazenada na variável"

**WScript.Shell**

A classe WScript.Shell é f para qualquer utilizador de escrita do Windows VBScript scripts de gerenciamento. Ele contém uma coleção de métodos úteis para a criação de scripts no Windows.

$wshell = New-Object -com WScript.Shell

$wshell | Get-Member



Um dos que eu mais gosta usando o WScript.Shell é o Popup

$wshell.Popup("O power shell é muito louco mesmo")

Você executar aplicativos usando o WScript.Shell. Como por exemplo a calculadora do Windows.

$wshell.Run("calc")

Outro método interessante é o SendKey.

#Alguns comandos utilizando WScript.shell

$wshell = New-Object -com WScript.shell

$wshell.Run("Notepad")

$wshell.AppActivate("Notepad")

Start-Sleep 1

$wshell.SendKeys("Interessante!!!")

**Variáveis**

Variáveis são muito importantes na criação de scripts. Pode não parecer muito útil nos exemplos básicos, mas você certamente terá a necessidade de usar variáveis pois diminui muito o trabalho e a escrita na hora de criar seus scripts. Variáveis armazenam informações que utilizamos ao longo do uso do script.

**Armazenando variáveis**

Todas as variáveis iniciam com o símbolo $

#um script com uma variálve simples

$variavel = "Hello world"

Write-Output $variavel

Você pode fazer uso de diversas variáveis:

#um script com uma variável simples

$variavel1 = "Hello"

$variavel2 = "World"

Write-Output $variavel1, $variavel2

Você pode também não usar o write-output e simplesmente colocar a variável na linha que ela será exibida.

#um script com uma variálve simples

#Um script com duas variáveis e sem o cmdlet Write-Output

$variavel1 = "Hello"

$variavel2 = "World"

"$variavel1, $variavel2"

Existem variáveis especiais no PowerShell, como por exemplo o $\_ que contém um pipeline do objeto atual, é usado em blocos de script, filtros, e muito usado na declaração Where do cmdlet Where-Object como visto anteriormente.

**Recebendo dados do usuário e armazenando em variáveis**

No seu primeiro script você utilizou o cmdlet Write.Host e agora vamos utilizar o cmdlet Read.Host que irá receber a entrada de dados e armazenar em forma de variável para ser utilizada na estrutura do seu script.

# Um script que recebe dados do usuário e armazena em uma variável.

$variavel1 = "Hello"

$variavel2 = Read-Host "Qual o seu nome ? "

"$variavel1, $variavel2"

Neste script será solicitado dados para a variavel2 (na linha 3). Uma recomendação importante é usar sempre nomes de variáveis simples e intuitivos, pois quando seus programas começarem a ficar mais complexos e maiores, será muito mais fácil identificar variáveis.

**Declaração de variáveis**

Quando você cria variáveis você pode especificar o tipo e assim evitar erros que podem ocorrer em determinados casos, como por exemplo a tentativa de soma de uma variável numérica e uma variável de texto.

Você pode declarar o tipo de variável.

[int]$numero = 20

Neste exemplo foi declarado o tipo de variáveis como inteiro 32-bit

**Propriedades das variáveis**

Algumas propriedades das variáveis são interessantes e retornam dados das instâncias dos objetos (variáveis). Essas propriedades podem retornar dados importantes como:

**.length** = Retorna o tamanho da variável armazenada.

$variavel1 = “Olá Mundo!”

$variavel1.Length

**.ToLower** = Exibe o valor da variável em caracteres minúsculos

**.ToUpper** = Exibe o valor da variável em caracteres maiúsculos.

**Arrays**

Array é uma matriz e permite armazenar informações em um índice. Uma Array cria uma lista indexada de valores. Cada item é atribuído um número de índice exclusivo. Ao primeiro item em uma matriz é atribuído o valor de (0), o próximo item (1) e em (2), e assim por diante.

A criação de Array é feita com a criação de uma variável que armazena uma lista de dados separados por vírgula.

#Script que armazena uma lista de servidores em um Array e testa todos.

$computadores = @("server1","server2","server3")

$somacomputadores = $computadores.Count

Write-Host "Esse script verifica se $somacomputadores srv estão online"

$computadores

Test-Connection $computadores -Count 1

Neste exemplo cada computador na lista é armazenado como string e você pode recuperar os dados armazenados. Não esqueça que são atribuídos índices iniciando do 0.

#Script que armazena uma lista de servidores em um Array.

$computadores = @("server1","server2","server3")

$computadores[0]

$computadores[1]

$computadores[2]

**Hash Table**

Uma tabela de Hash é uma matriz que permite armazenar dados aos pares. Uma "chavevalor". A "chave" e "valor" podem ser qualquer tipo de dados e comprimento. Os elementos devem ter aspas de acaso tiverem espaços.

# Cria uma tabela de dados de servidores na minha rede.

$servidores =@{Server1="10.0.0.1" ; Server2="10.0.0.2" ; Server3 ="10.0.0.3"}

$servidores

O mais interessante em Hash table é que você pode operar os valores existentes na tabela.

Adicionar mais um valor

$servidores["Server4"] = "10.0.0.4"

Apagar um valor

$servidores.Remove(“Server4”)

Apagar todos os valores

$servidores.clear()

Tabelas HASH podem ser ordenadas de acordo com sua criação usando o parâmetro [ordered] , pois por padrão ela é exibida sem nenhuma ordenação.

# Ordenação de HashTable

@{a=1;b=2;c=3}

[ordered]@{a=1;b=2;c=3}

**PSDefaulParametersValues**

Esse novo recurso PSDefaulParametersValues do PowerShell 3.0 permite que você salve informação de uma tabela hash como parâmetros padrões.

Neste caso, podemos imaginar situações como a configuração de um script para envio de emails:

Send-MailMessage -from edilonandrade@gmail.com -SmtpServer smtp.gmail.com -UseSsl -Port 465 -Credential (Get-Credential edi.andrade@yahoo.com.br) -To edilon.andrade@inmetrics.com.br -Subject "PowerShell Rocks"

Com o recurso PSDefaultParameterValues você salva as configurações

$PSDefaultParameterValues = @{

"Send-MailMessage:from" = " seuemail@seudominio.com ";

"Send-MailMessage:SmtpServer"= " smtp.seuservidor.com ";

22

"Send-MailMessage:UseSsl"=$true;

"Send-MailMessage:Port"=587;

"Send-MailMessage:Credential" = (Get-Credential seuemail@seudominio.com )

}

# envio de emails simplicado

Send-MailMessage -to Emaiquedesejaenvia@hotmail.com -Subject "Enviado do

PowerShell" -Body "Termino do Evento"

Você pode limpar os parâmetros padrões com o seguinte comando:

$PSDefaultParameterValues["Disable"] = $true

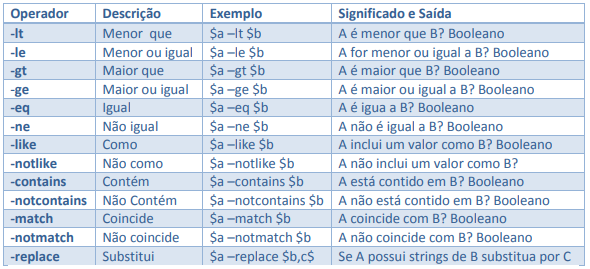
**Operadores Condicionais e Lógicos**

Assim como em outras linguagens de programação o PowerShell pode fazer o uso de operadores Condicionais e Lógicos. Os operadores são importantes tanto na criação de scripts como em operações comuns com o PowerShell e permitem fazer a comparação tanto de variáveis como de números e a ordem sempre será da esquerda para a direita.

Você pode testar digitando no PS:

O resultado será no formato Booleano (Falso ou verdadeiro), neste caso o resultado deve ser Falso. Já caso você inverta o valor 8 com o 4 certamente o resultado será verdadeiro.

**Operações de comparação**



**Operadores Aritméticos**

O Windows PowerShell também suporte operadores matemáticos.

Os operadores de Adição e Multiplicação aceitam variáveis e arrays.



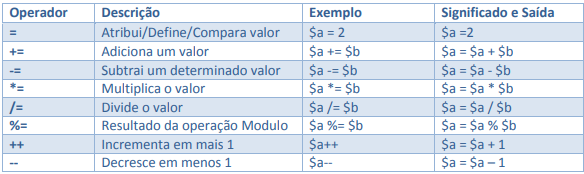
**Operadores Lógicos**

Os operados lógicos são amplamente utilizados em scripts.



**Operadores de atribuição**

Os operadores de atribuição fornecem a capacidade de fazer operações numéricas atribuindo um ou mais valores as variáveis existentes.



O PowerShell suporta valores de armazenamento computacional como:

* Kilobytes (KB)
* Megabytes (MB)
* Gigabytes (GB)
* Terabytes (TB)
* Petabytes (PB)

O que pode ser muito útil na hora de calcular unidades de disco, cálculo de setores ou de fitas de armazenamento de backup. Por exemplo, você tem um disco com 2TB de dados e deve fazer o backup em fitas de 250GB. Quantas fitas você precisa?

**IF...ELSE**

Quando estamos criando scripts precisamos quase sempre controlar os fluxos das operações baseados em condições. A forma mais simples de fazer isso é utilizando simplesmente o IF.

IF (condição) {comando}

if (Get-ChildItem -File = C:\Users\QA\Documents\security.log) {echo OK}

Nesse caso se a condição for verdadeira o comando será executado, do contrário uma mensagem de erro será exibida. Você pode criar dois caminhos para a condição utilizando o IF...ELSE

IF (condição) {bloco de comando} [ELSE {bloco de comandos2}]

if (Get-ChildItem -File = C:\Users\QA\Documents\security.log) {echo OK} Else {Get-ChildItem -

File}

**FOR…FOREACHE…WHILE**

Além do controle de condicional, você pode querer fazer o uso de looping para executar repetidamente um determinado comando e para cada tipo de valor fazer uma determinada execução. Resumindo – Você pode querer fazer um PING para cada endereço IP terminando com 1 até o 245.

Os controles de looping no PS podem ser executados de muitas maneiras, com os comandos:

* For
* ForEach
* While
* Do While
* Do Until

**FOR**

O mais simples é o cmdlet FOR.

for (inicio; condição; proximovalor) {Código de repetição}

for ($a=1; $a -le 10; $a++) {echo 192.168.1.$a}

**FOREACH**

O ForEach é mais utilizado para executar cada item em uma coleção de informação, seja uma informação coletada em um comando ou filtro ou dados em um array.

A sintaxe do Foreach é bem simples:

ForEach ($variavel e items da coleção) {código de execução}

foreach ($numeros in 1,2,3,4,5,6,7,8,9,10) { $numeros }

Agora que você conhece os recursos IF, ELSE, FOR e ForEach, você pode combinar e começar a ter resultados bem interessantes e poderosos.

foreach ($file in Get-ChildItem) {

if ($file.IsReadOnly) {

Write-Host $file.FullName }

}

O Windows PowerShell entende que a estrutura criada deve exibir cada valor. Ou seja ele usa automaticamente o Foreach. Veja, você pode usar o script:

$process = Get-Process

$processos = $process | foreach { $\_.name }

$processos

Ou

$processo = Get-Process

$processo.name

Você pode usar o ForEach de diversas maneiras. Vamos supor que você deseja “matar” vários processos do Notepad. Você usa o comando Get-Process e “para cada” processo do Notepad você mata.

Get-process Notepad | foreach kill

**WHILE**

$i = 1

while ($i -le 5)

{

$i++ ;$i

}

**Funções**

Funções são comandos em um script, que dura apenas durante a sessão em que estiver sendo executado. Quando terminar de usar os comandos em uma função ela não existirá mais na memória e isso é bem simples assim mesmo e fácil de implementar.

Function Dia-Hora {

write-host "Essa função fica na memoria durante a sessão ";

Get-Date

}

Dia-Hora

Durante a sessão você pode chamar suas funções quantas vezes forem necessária, pois fica disponível como se fosse um comando. Nesse exemplo o comando é o Dia-Hora Abaixo na imagem você pode notar que a função foi retraída e podemos clicar no símbolo + ao lado da função para expandir e visualizar o código.

O PowerShell ISE auto completa as suas funções.

**Workflow**

Voce pode escrever workflows no PowerShell ou usar o XAML (Extensible Application Markup Language).

Um workflow funciona como se fosse um cmdlet do PowerShell, com a diferença de fazer o uso do Windows Workflow Foundation que habilita a criação de scripts de longa execução gerenciáveis. Usando um Workflow você pode Interromper, suspender, reiniciar, repedir e fazer execução paralela, o que o torna uma excelente ferramenta em ambientes de computação em nuvem, já que permite aos administradores automatizarem tarefas de execução diária e repetidas. Workflow suporta interrupções de maquinas e rede , como por exemplo a reinicialização de um servidor.

Workflow MeuWorkFLow {Write-Output -InputObject "Olá esse é o meu Workflow!"}

Você pode então fazer a execução do seu WorkFlow simplesmente executando nome do

MeuWorkFLow

Um bom exemplo de como podemos fazer o uso de Workflow é criando um script de longa duração e invocando os mesmo como um job. Para entender melhor execute a sequência de scripts criados especificamente para entendimento desse recurso.

# Crie e execute o workflow

workflow MeuWorkFlow

{

while(1)

{

(get-date).ToString() + " Script Demorado"

Start-Sleep -seconds 3

}

}

#invokar o comando como um trabalho (JOB)

$wfjob = MeuWorkFlow -AsJob

# Consulte e use o WFjob E não se esqueça de consular os cmdlets \*-job.

$wfjob

Receive-Job $wfjob

# Você então pode suspender o trabalho e notar que ele não exibe informações, pois está suspenso.

Suspend-Job $wfjob -Force -Wait

$wfjob

Receive-Job $wfjob

# Por fim você pode resumir o trabalho.

Resume-Job $wfjob -Wait

# Finalizar o trabalho !!

Get-Job | Remove-Job -Force

Você pode visitar o Blog do MSDN e conhecer o mais completo modulo para trabalhar com workflow. http://bit.ly/X4KAeA Este modulo, trabalha com criação de discos em paralelo para diversas maquinas em ambiente.

**Execução Paralela**

Quando usamos or recurso de WorkFlow, podemos tomar vantagem do uso da execução paralela. Em scripts sendo executados em vários servidores e que precisam executar varioas tarefas, pode ser necessário usar o Parallel.

Neste exemplo, vamos criar um workflow que irá iniciar os dois editores de texto nativos de Windows para demonstrar o recurso de execução em paralela.

Workflow Start-Editores {

Parallel {

Start-Process –FilePath Notepad

Start-Process –FilePath Wordpad

}

}

**Jobs e Scheduled Jobs**

**Jobs** – Permitem a execução de comandos em Background no computador local ou remoto. Os trabalhos são executados em background assincronamente É a melhor solução para comandos de longa duração.

São vários os cmdlets que podem ser utilizados para trabalhar com Job

Start-Job - Inicia um trabalho.

Get-Job - Exibe os trabalhos associados a atual sessão.

Wait-Job - Aguarda pelo trabalho até que esteja pronto.

Receive-Job – Exibe o resultado de um trabalho em background.

Stop-Job - Para um trabalho.

Remove-Job - Remove um trabalho

Exemplo de um trabalho que exibe valores randomicos e entra em pausa por 5 minutos:

Start-Job { while($true) { Get-Random; Start-Sleep 5 } } -Name Dorminhoco

**Scheduled Jobs** - Trabalhos agendados são extremamente uteis quando você tem tarefas que são executadas com maior frequência ou com recorrência e principalmente quando são atividades de longa duração.

Os trabalhos agendados do Windows PowerShell podem ser gerenciados através do “Agendador de Tarefas do Windows”.

Executando o console ou o ISE como administrador você pode criar um trabalho agendado usando os seguintes cmdlets:

$agendamento = New-JobTrigger –Daily –At 4pm

Register-ScheduledJob –Name TrabalhoAgendado –ScriptBlock {Get-Process} –

Trigger $agendamento

**Write-Progress**

Alguns scripts podem fazer trabalhos demorados, principalmente scripts do tipo que fazem busca e eliminação de arquivos no disco ou cópias de base de dados ou qualquer atividade demorada. Você pode então fazer uso do cmdlet Write-Progress que irá apresentar o tempo decorrente da atividade relacionada . Você pode usar um script básico para entender melhor como funciona esse cmdlet

for($i = 1 ; $i -le 10 ; $i++)

{

Write-Progress -Activity "Contando até 10" -status "`$i equals $i"

sleep 1

}

Neste script a atividade é a contagem de 1 até 10 controlados pelo “sleep 1”, ou seja uma pausa de 1 minuto entre cada valor incremental. A atividade será exibida na barra de progressão do Write-Progress.

Ou bom exemplo onde será mostrado o valor em porcentagem:

$lista = Get-ChildItem

$contagem = 0

foreach ($arquivo in $lista) {

$contagem++

$contagem

$lista.count

Write-Host $arquivo

Write-Progress -Activity "Listando Diretorios" -status "Andamento" -PercentComplete (($contagem / $lista.count)\*100)

sleep 0.6

}]

**Executando o PowerShell remotamente**

Em muitas situações você não estará interativamente logado no computador que você deseja administrar e nesse caso você pode fazer o uso de sessões remotas do PowerShell. Você pode iniciar sessões remotas de diversas maneiras como será descrito adiante.

Remoting PowerShell é construído sobre WinRM para fornecer acesso remoto a máquinas usando os Enter-PSSession e o Invoke-Command. Sempre que um usuário se conecta a um computador remoto usando uma dessas cmdlets, o serviço cria WinRM uma nova instância de um processo para sediar a sessão

**Sessões Persistentes**

Ao fazer execuções de comandos do PS remotamente você está sujeito a interrupções causadas for falhas na rede. O PowerShell suporta sessões persistentes ou resilientes (PSSessions). Uma sessão persiste permite criar e salvar uma sessão em um computador remoto e você pode desconectar a sessão e então reconectar quando necessário. O diferencial desse recurso é os comandos do PowerShell continuarão em execução no computador remoto, mesmo quando você não estiver conectado na sessão o que antes do PS 3.0 não podia ser feito.

As sessões persistentes são criadas com o cmdlet New-PSSession

**Solicitando credenciais**

Quando estiver trabalhando com scripts e ou executando comandos em ambiente de domínio, você pode solicitar credenciais especificas para execução de determinadas tarefas, principalmente aquelas que exigem privilégios administrativos.

$cred = Get-Credential

Ou você pode especificar ao conta para facilitar ainda mais...

$admin = Get-Credential contoso\administrator

Exemplo 1:

$c = Get-Credential

Get-WmiObject Win32\_DiskDrive -ComputerName Server01 -Credential $c

Exemplo 2:

Get-WmiObject Win32\_BIOS -ComputerName Server01 -Credential (get-credential Domain01\User01)

**Invocando comandos**

O cmdlet Invoke-Command executa os comandos em computadores remotos e exibe o resultado no seu console. Isso é extremamente útil em ambientes com vários servidores ou computadores e você deseja executar comandos remotamente.

Invoke-Command -ComputerName osiris -ScriptBlock {Get-WmiObject Win32\_Bios}

Você pode criar uma lista de servidores em um arquivo de TXT

Invoke-Command -ComputerName (Get-Content Servers.txt) -ScriptBlock {GetProcess Notepad}

**Dica** – Use o cmdlet Get-ADComputer para criar uma lista de servidores.

Get-ADComputer -Identity "User01-SRV1" -Properties \*

**Habilitando gerenciamento Remoto**

Para trabalhar com execuções remotas você deve antes habilitar o gerenciamento remoto. Execute o PowerShell como administrador e execute o cmdlet.

Enable-PSRemoting

Você será prontificado a responder uma série de opções sobre configurações do Winrm e do cmdlet “Set-PSSessionConfiguration"

Você analisar cada uma e escolher entre as opções Y para (Sim), A para (Sim para Todos), N para (Não) e L para (Não para todos).

Muitos fatores podem bloquear o seu acesso ao servidor remoto, a rede, o Firewall do Windows e principalmente se seu computador não fizer parte da rede do domínio. Nesse caso você provavelmente terá que adicionar seu computador a lista de computadores confiáveis no gerenciamento do Winrm. Execute o seguinte comando no prompt de comando (Não no PowerShell)

winrm set winrm/config/client @{TrustedHosts="ComputadorRemoto"}

**Sessão Remota**

Logo após habilitar você pode entrar nas sessões remotas usando o cmdlet Enter-PSSession usando o console do PowerShell.

São vários os cmdlets relacionados a sessões remotas e você pode conferir usando o comando:

get-command \*pssession

Dica – Não se esqueça de habilitar o gerenciamento remoto Enable-PSRemoting.

Para iniciar uma nova sessão remota com um computador chamado server1:

New-PSSession -ComputerName server1

E assim que estiver conectado você pode fazer suas configurações remotas.

Enter-PSSession –Computername nomedoservidor -Credential:Credenciais

Exemplo

Enter-PSSession -computername WS2012-HPV01 -Credential:Administrator

**Sessões persistentes**

Sessões persistentes são importantes na execução de scripts de longa duração é possível manter as sessões em execução mesmo com falhas na rede.

Quando as conexões de rede são perdidas o PowerShell tenta fazer novas conexões a cada 4 minutos.

Para iniciar uma sessão persistente você deve usar o cmdlet New-PSSession

$remoto = New-PSSession localhost

Invoke-Command $remoto { “Tem alguém ai” }

As consultas podem ser feitas com o cmdlet:

Get-PSSession

Caso você esteja utilizando scripts ou funções que sejam de longas durações você pode desconectar da sessão que o trabalho continua em execução remota:

Disconnect-PSSession $remoto

Ou

Disconnect-PSSession -id 4

Use o seguinte script para entender e testar sessões persistentes. Esse script cria uma sessão remota com um servidor chamado Servidor1 e fica contando números sem parar.

$s = New-PSSession -name Contar -ComputerName servidor1

Invoke-Command -Session $s -ScriptBlock {

1..1000000 | % {"Contando $\_";sleep 1 }

}

Você pode então reconectar a sessão criada anteriormente com o cmdlet Connect-PSSession

Connect-PSSession $s

Depois é só terminar a sessão remota

Remove-PSSession -ComputerName localhost

**Sessão Remota Server Manager do Server 2012**

Se você é um administrador do Windows Server 2012, você pode iniciar uma sessão através do próprio Server Manager enquanto estiver gerenciando servidores no seu Pool de servidores. Basta clicar com o lado direito do mouse sobre o servidor que deseja gerenciar e selecionar Windows PowerShell.

**Sessão Remota no PowerShell ISE**

Outra maneira simples de iniciar um sessão remota é usando o PowerShell ISE, clicando em File > New Remote PowerShell Tab ou sobre o ícone.

**PowerShell WEB Access (PSWA)**

No Windows Server 2012 é possível habilitar o acesso ao Windows PowerShell via WEB com o recurso PowerShell Web Access. Com essa facilidade você pode administrar os servidores a partir de qualquer dispositivo com acesso à Internet e que possua um navegador compatível

* Internet Explorer nas versões 8.0 ou superior
* Mozilla Firefox 10.0.2 ou superior
* Google Chrome 17.0.963.56m for Windows
* Apple Safari 5.1.2 for Windows e para Mac OS

Você pode usar dispositivos móveis desde que possuam sistemas e browsers compatíveis.

* Windows Phone 7 ou superior
* Android 2.2.1 ou superior
* Apple Safari for iPhone ou IPad 2

**Instalação e configuração do PSWA**

Faça a instalação da feature **Windows PowerShell Web Access.**

Install-WindowsFeature WindowsPowerShellWebAccess –IncludeManagementTools

É necessário um certificado digital para acessar o site que executará o PSWA, ou se vocêe estiver usando um ambiente de teste, você pode usar o comando que irá usar um certificado de teste também.

Se acaso você já instalou a aplicação, desinstale usando o comando **unistal-lPswaWebApplication** e em seguida instale novamente.

Install-PswaWebApplication –webApplicationName PSWA –useTestCertificate

Após a instalação do recurso é necessário conceder acesso e você pode fazê-lo usando o comando:

Add-PswaAuthorizationRule –UserName dominio\usuário –ComputerName \* -ConfigurationName microsoft.powershell

Exemplo

Add-PswaAuthorizationRule -UserName Contoso\Administrator -ComputerName \* -ConfigurationName microsoft.powershell

Para maiores detalhes sobre os parâmetros do cmdlet Add-PswaAuthorizationRule use o comando:

Show-Command Add-PswaAuthorizationRule

Após as configurações você pode acessar o PSWA usando a seguinte URL: <https://nome_do_servidor/pswa>

**Gerenciando Servidores**

**Instalação de Features**

O PowerShell possui nativamente muitos módulos com centenas de cmdlets que podem nos ajudar nas tarefas diárias de administração. Para maximizar a utilização dos recursos e simplificar a maneira de enumerar os cmdlets, enumero aqui alguns dos Módulos relacionados à administração de servidores. Lembre-se que você pode listar os cmdlets de cada modulo usando o comando: Get-command –module nomedomodulo

Existem alguns módulos que não fazem parte do core do PowerShell, mas que você pode fazer o download e aumentar a lista de ferramentas de manutenção e gerenciamento de servidores. Você pode encontrar uma seleção de módulos interessantes para administração nesse link do TechNet wiki: <http://bit.ly/10jWIu0>

Gerenciamento de Features do Windows Server 2012.

Não existe melhor método de instalação de roles e Features do que a feita através do PowerShell.

A forma mais simples de instalar qualquer recurso e iniciando pela busca baseada no nome da Feature e você não precisa saber o nome completo.

Neste exemplo eu procuro pela Feature tem parte do nome do nome DNS e como resultado tenho o DNS e as Ferramentas de administração do DNS. Usando o comando mais o Pipe de redirecionamento podemos fazer a instalação da feature em questão e suas dependências. Para instalar o DNS você pode usar o cmdlet

Install-WindowsFeature DNS –IncludeManagementTools

**Instalação do Active Directory Domain Services**

O Modulo ADDSDeployment possui dezenas de cmdlets que ajudam os administradores a instalar desde um domínio único a uma complexa floresta do Active Directory Domain Services.

Para ter este modulo disponível é necessário que os arquivos binários do ADDS estejam instalados.

Add-WindowsFeature AD-Domain-Services

Para instalar o primeiro controlador de dominio da Floresta,

#

# Windows PowerShell script for AD DS Deployment

#

Import-Module ADDSDeployment

Install-ADDSForest `

-CreateDnsDelegation:$false `

-DatabasePath "C:\Windows\NTDS" `

-DomainMode "Win2012" `

-DomainName "contoso.com" `

-DomainNetbiosName "CONTOSO" `

-ForestMode "Win2012" `

-InstallDns:$true `

-LogPath "C:\Windows\NTDS" `

-NoRebootOnCompletion:$false `

-SysvolPath "C:\Windows\SYSVOL" `

-Force:$true

**Gerenciando TCP/IP**

Antes do PowerShell o mais rápido comando para verificar o IP da máquina era o IPconfig e o IPConfig /All. Esse comando ainda é útil e rápido e pode ser completado com o TAB, mas não os seus parâmetros.

E como adicionar um endereço IP ? Usando o comando Netsh que ainda permite tal tarefa ou é claro com o PowerShell.

Podemos gerenciar vários recursos de rede usando os cmdlets relacionados que você pode listar com o comando:

Get-Command \*NetIP\* e Get-Command \*NetAdap\*

E claro como o verbo teremos:

Get

Remove

Set

Enable, Disable

E outros….

Aqui vamos listar os cmdlets mais simples e comuns que podem fazer parte da sua rotina de gerenciamento de servidores.

Get-NetIPAddress - Exibe informações a respeito do endereçamento IP, importante para verificar qual o index na placa do IP que você está buscando.

Get-NetAdapter - Exibe informações a respeito dos seus adaptadores de rede.

**Listando as placas de redes**

Get-NetAdapter

Você pode filtrar o resultado com parâmetros.

Get-NetAdapter -Name Ethernet

**Alterando o endereço IP da placa de rede**

Get-NetIPAddress

New-NetIPAddress –InterfaceIndex 12 –IPAddress 192.168.0.1

Neste outro exemplo está sendo definido o endereço IP a máscara em forma de prefixo e o gateway:

New-NetIPAddress –InterfaceIndex 12 –IPAddress 192.168.0.1 -PrefixLength 24 -

DefaultGateway 192.168.0.5

**Para remover:**

Remove-NetIPAddress –IPAddress 192.168.0.1 -DefaultGateway 192.168.0.5

**Habilitando o DHCP na Interface**

Set-NetIPInterface –InterfaceIndex 12 –Dhcp Enabled

Set-NetIPInterface –InterfaceIndex 12 –Dhcp Disabled

**Adicionando um DNS**

Set-DnsClientServerAddress -InterfaceAlias Ethernet -ServerAddresses 192.168.0.1

**Renomeando a placa de rede.**

Rename-NetAdapter -Name Ethernet -NewName "Rede Ethernet"

**Habilitando e Desabilitando uma Placa de rede**

Disable-NetAdapter -name Ethernet

Enable-NetAdapter -name Ethernet

**NIC Teaming**

NIC Teaming é um novo recurso do Windows Server 2012 que pode ser usado para separar o tráfego que utiliza VLANs e com 2 até 32 placas de rede você pode fazer um balanceamento de rede e failover . Você pode listar os cmdlets para gerenciar NIC Teaming com o comando:

Get-Command \*NetLBFO\*

**Criando um NIC Team**

New-NetLbfoTeam -Name NICTeam1 -TeamMembers Ethernet1, Ethernet2

**Gerenciando o Hyper-V**

O gerenciamento do Hyper-V através do PowerShell é bem rico e não é possível enumerar todos os cmdlets e funções que podemos faz uso aqui nesse e-book. Mas você encontrar uma lista completa dos cmdlets em <http://technet.microsoft.com/enus/library/hh848559(v=wps.620).aspx>

Mesmo assim, vamos enumerar alguns cmdlets que podem ser uteis e também nos orientar no gerenciamento do Hyper-V.

**Criar uma máquina virtual chamada “Máquina Virtual 1” com 512MB**

New-VM -Name "Virtual Machine 1" -MemoryStartupBytes 512MB

**Criar um novo disco virtual dinâmico VHDx com 10 GB na pasta c:\VHD**

New-VHD –Path c:\vhd\Base.vhdx –SizeBytes 10GB

**Adicionar VHDx para uma máquina virtual**

Add-VMHardDiskDrive -VMName "Virtual Machine 1" -Path c:\vhd\Base.vhdx

**Criar um SNAPSHOT**

Checkpoint-VM -Name "Virtual Machine 1" -SnapshotName AntesdoServicePack

Get-VMSnapshot –VMName "Virtual Machine 1"

**Ligar e Desligar uma VM**

Start-VM –Name Virtual\*

Stop-VM –Name Virtual\* -Force

**Exportar e Importar maquinas virtuais**

Export-VM –Name "Virtual Machine 1" –Path C:\VM

Import-VM –Name "VM1" –Path 'C:\VM\413BADFD-E712-4373-8C06-68285BEEE794.XML'

**Fazendo o PowerShell Falar**

Alguns recursos podem não ser tão importantes, mas certamente são interessantes e podem ser explorados para trazer maior interação nos seus scripts.

$a = New-Object –ComObject SAPI.SPVoice

$a.speak(“Você não tem privilégios para executar essa operação”)

Usando o recurso do objeto SAPI.SPvoice você criar aplicações como essa do site http://bit.ly/Zn7ivH que lê os feeds do seu twitter ou sobre uma palavra chave, que nesse caso é o próprio PS.

$webClient = New-Object -TypeName "System.Net.WebClient"

$s = New-Object -ComObject "SAPI.SPVoice"

$s.Rate = -1;

$url = "http://search.twitter.com/search.atom?q=PowerShell"

$x = [XML]$webClient.DownloadString($url)

$x.feed.entry | foreach {

$s.Speak("Tweet from: " + $\_.author.name)

$s.Speak($\_.title)

}

**Gerando Senhas Complexas**

Não existe um cmdlet especifico para isso, mas o pessoal do “PowerShell Magazine” explorando uma classe do .Net Framework chamada System.Web.Security.Membership notaram que existe uma classe chamada GeneratePassword().

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

Foi daí que surgiu esse script que no meu exemplo gera uma senha de 15 caracteres e 3 não alfabéticos

$Assembly = Add-Type -AssemblyName System.Web [System.Web.Security.Membership]::GeneratePassword(15,3)

**PowerShell como Shell padrão no Windows Server Core**

Essa configuração depende de modificações no registro do servidor que podem ser feitas através do próprio PS. Porém antes de alterar qualquer configuração no registro do Windows é recomendado fazer o backup do mesmo. E aqui também é interessante salvar a chave do Shell Padrão, para caso você deseje voltar a usá-lo no futuro.

get-itemproperty “hklm:\software\Microsoft\windows NT\currentversion\winlogon”

Shell | select -ExpandProperty Shell | out-file shellpadrao.txt

Agora sim, você pode definir o PowerShell como o Shell Padrão:

set-itemproperty ”hklm:\software\microsoft\windows nt\currentversion\winlogon” shell powershell.exe

Para voltar o Shell que você salvou:

$defaultshell=get-content .\shellpadrao.txt

set-itemproperty “hklm:\software\microsoft\windows nt\currentversion\winlogon”

shell $defaultshell

**Alterando o Prompt do PowerShell**

Essa dica é apenas para personalizar o seu Prompt do PowerShell. Nada tão complexo e importante, mas interessante.

function prompt { "Power Shell 3.0 > " }

Fazendo um sorteio de números ou nomes

Isso foi muito útil durante um treinamento, onde eu precisei fazer um sorteio e já não estava afim de usar fazer perguntas, ou escolher aqueles que fazem aniversario no dia ou próximo. Eis que o PowerShell surgiu para nos salvar nesse momento. Você pode fazer um sorteio de números usando o Objeto System.Random.

Get-Random -minimum 1 -maximum 101

Em palestras e salas de aula nós temos nomes e esse é o método mais simples de se fazer um sorteio:

($a = "Daniel","Suzana","Felipe","Henrique","Karol") | Get-Random

Você pode sortear 3 de uma vez:

($a = "Daniel","Suzana","Felipe","Henrique","Karol") | Get-Random -Count 3

Outro recurso interessante é que você pode usar um arquivo de texto e sortear o conteúdo.

Get-Content C:\Scripts\Teste.txt | Get-Random

**Exibindo RSS Feed de um site**

Este script exibe os últimos Feeds de noticias de um determinado site com RSS habilitado:

$irm = Invoke-RestMethod http://www.mcsesolution.com/Noticias/feed/rss.html

$irm | Select-Object PubDate, Title

**Enviando E-mail através do PS.**

Esse script é muito útil em tarefas do dia a dia, pois você pode criar tarefas agendadas e receber notificações assim que as mesmas forem completadas.

Send-MailMessage -from seuemail@seudominio.com `

-SmtpServer smtp.seuservidor.com `

-UseSsl `

-Port 587 `

-Credential (Get-Credential seuemail@seudominio.com) `

-To seuemail@seudominio.com `

-Subject "PowerShell Rocks"

**Alterando a cor do PowerShell**

Você pode dar mais vida e ênfase aos seus scripts usando o cores nos parâmetros - foregroundcolor e –backgroundcolor

Exemplo:

Write-Host "Hello World" -foregroundcolor Green -backgroundcolor Black

Você pode usar várias cores:

