

Windows PowerShell 3.0



**Um guia do Windows PowerShell
desenvolvido especificamente para
profissionais de Infraestrutura.**

1ª Edição - Fevereiro de 2013

Daniel Donda

www.mcseSolution.com



Introdução

Este e-book foi escrito especificamente para os profissionais de infraestrutura de redes (IT Professional).

A intenção de criar esse material começou quando eu comprei alguns livros de PowerShell e como IT Pro senti muita dificuldade em entender alguns conceitos, pois em sua maioria os livros eram inclinados para a área de DEV, Ou tratava de assuntos sem maiores introduções que certamente era de conhecimento daqueles que são desenvolvedores.

Eu também precisa de um ponto de referência para quando eu fosse começar a criar os meus scripts ou usar os comandos. Foi então que eu comecei a compilar diversas informações e coloca-las em uma ordem de fácil assimilação e que possa ser utilizada tanto como fonte de aprendizado como de consulta para aqueles que já iniciaram no PowerShell.

O PowerShell é hoje parte integrante dos sistemas operacionais da Microsoft e um grande aliado aos administradores de sistemas. Não deixe para depois, comece agora mesmo a usar o PowerShell no seu dia a dia.

O e-book do PowerShell em seus primeiros dias já é um enorme sucesso. Recebi muitos e-mails de agradecimento tanto do Brasil como de fora. Alias, fiquei muito feliz com o contato de **Shay Levy**, Windows PowerShell MVP e um dos fundadores do site PowerShellMagazine.com de onde usamos diversos exemplos e inclusive a imagem da capa, que é de um dos muitos papeis de parede que você encontra em

<http://www.powershellmagazine.com/2011/09/23/powershell-wallpapers/>

Importante é saber que esse material não está finalizado e depende de você para que novas versões sejam editadas e atualizadas. Muito Obrigado.



Daniel Donda

www.mcseolution.com

*Todo o conteúdo está sob licença da **Creative Commons** Attribution 3.0 Unported License*

<http://bit.ly/ZnVDOD>

Sumário

Introdução	0
Windows PowerShell	5
Versões do PowerShell	5
PowerShell 3.0	6
PowerShell ISE	7
Command-Lets	9
Show-Command	10
Help Poderoso	10
Funções	12
Alias	12
Como criar um alias?	12
Controlando a exibição (saída) de informações	12
Detalhes dos cmdlets out	13
Filtrando resultados na tela (Where-Object)	14
Módulos	15
Listando módulos	16
Importando módulos	16
Scripts no PowerShell	16
Script (Hello World)	17
Estrutura de um script	17
WScript.Shell	18
Variáveis	18
Armazenando variáveis	19
Recebendo dados do usuário e armazenando em variáveis	19
Declaração de variáveis	19
Propriedades das variáveis	20
Arrays	20
Hash Table	20
PSDefaultParametersValues	21
Operadores Condicionais e Lógicos	22
Operadores de comparação	22
Operadores Aritméticos	22
Operadores Lógicos	23
Operadores de atribuição	23
IF...ELSE	24

FOR...FOREACHE...WHILE	24
FOR	25
FOREACH	25
WHILE	26
Funções	26
Workflow.....	27
Execução Paralela.....	28
Jobs e Scheduled Jobs	28
Write-Progress	29
Executando o PowerShell remotamente	30
Sessões Persistentes	30
Solicitando credenciais.....	30
Invocando comandos	31
Habilitando gerenciamento Remoto.....	31
Sessão Remota	32
Sessões persistentes	33
Sessão Remota Server Manager do Server 2012	34
Sessão Remota no PowerShell ISE	34
PowerShell WEB Access (PSWA)	35
Instalação e configuração do PSWA.....	35
Gerenciando Servidores.....	37
Instalação de Features	37
Instalação do Active Directory Domain Services.....	39
Gerenciando TCP/IP	39
Listando as placas de redes.....	40
Alterando o endereço IP da placa de rede.....	40
Habilitando o DHCP na Interface.....	40
Adicionando um DNS	40
Renomeando a placa de rede.	40
Habilitando e Desabilitando uma Placa de rede	41
NIC Teaming	41
Criando um NIC Team	41
Gerenciando o Hyper-V.....	41
Criar uma máquina virtual chamada “Máquina Virtual 1” com 512MB	41
Criar um novo disco virtual dinâmico VHDx com 10 GB na pasta c:\VHD	41
Adicionar VHDx para uma máquina virtual.....	41

Criar um SNAPSHOT	41
Ligar e Desligar uma VM.....	41
Exportar e Importar maquinas virtuais	41
Dicas e Truques	42
Adicionar “Tile” Shutdown/Restart/Logoff no Windows 8	42
Fazendo o PowerShell Falar	42
Gerando Senhas Complexas.....	43
PowerShell como Shell padrão no Windows Server Core.....	43
Alterando o Prompt do PowerShell	44
Exibindo RSS Feed de um site	44
Enviando E-mail através do PS.	45
Alterando a cor do PowerShell.....	45
Referências Bibliográficas	46
Sobre o autor.....	46

Windows PowerShell.

O Windows PowerShell é o novo shell de linha de comando do Windows. Shell é uma interface que permite aos usuários interagir com o sistema operacional e pode ser tanto no modo gráfico *Graphical User Interface (GUI)* quanto em modo texto *Command-Line Interface (CLI)*.

O Windows PowerShell que agora irei chamar apenas de **PS**, inclui um prompt interativo e um ambiente para criação de scripts para administração do sistema e automação. Compilado sobre o **CLR (Common Language Runtime)** do .NET Framework permite que profissionais de TI e desenvolvedores controlem e automatizem a administração do Windows e aplicativos.

O PS usa linguagem de script expressiva, com expressões regulares e permite o uso do .NET Framework, Windows Management Instrumentation (WMI), COM, Registro do Windows e muito mais.

Porém este livro é voltado para IT Pros. Administradores que estão migrando seus scripts do bom e velho “batizinho” para essa nova e poderosa linguagem.

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.

O propósito desse livro é introduzir noções de scripting para automação e gerenciamento de servidores e estações usando o PowerShell do Windows 8 e Windows Server 2012. Claro que os exemplos serão em sua maioria simplificados e utilizados em uma forma mais ampla para uma melhor assimilação e aproveitamento.

Em resumo, este e-book deve ser usado como material de consulta para criação de scripts relacionados a administração em infraestrutura de TI.

Versões do PowerShell

A **versão 1.0** do Power Shell foi lançada em 2006 para Windows XP SP2/SP3 e o Windows Vista. No Windows Server 2008 o PS é um “Features”.

A **Versão 2.0** está integrada com o Windows 7 e o Windows Server 2008 R2. Também é possível a instalação para Windows XP Service Pack 3, Windows Server 2003 com SP3 e Windows Vista SP

PowerShell 3.0

A **Versão 3.0** pode ser instalado nos sistemas Windows 7 Service Pack 1 e Windows Server 2008 R2 SP1 usando o Link <http://bit.ly/10ldOHT>

Já as versões Windows Server 2012 e o Windows 8 executam nativamente o Windows PowerShell 3.0

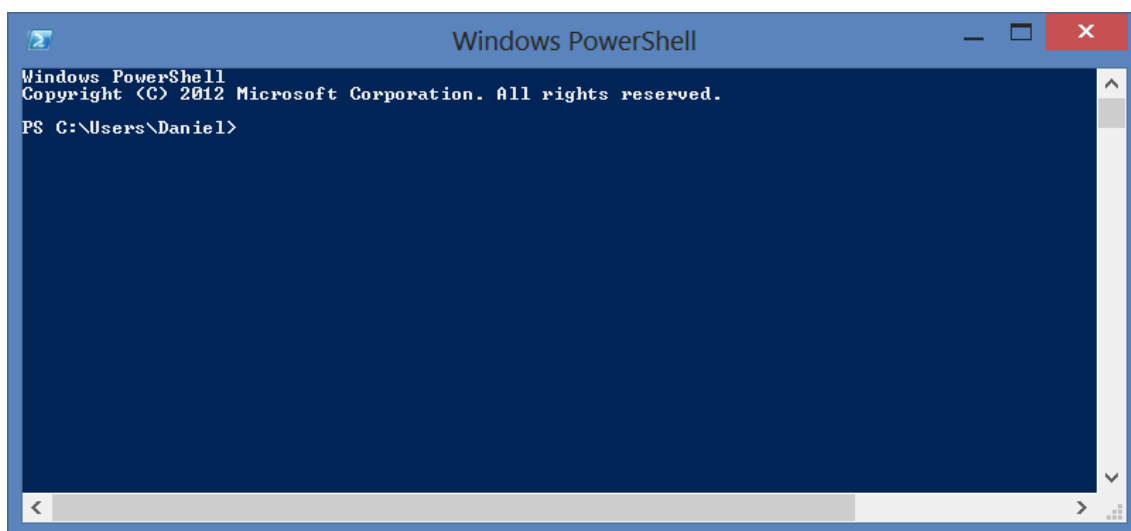
O PowerShell 3.0 inclui novos recursos que adiciona maior flexibilidade e mais poder de gerenciamento para ambientes nas nuvens e gerenciamento Multi-servidor.

Como comparativo entre as versões O Windows Server 2008 R2 possuía aproximadamente 230 cmdlets.

O Windows Server 2012 conta com aproximadamente 2.430 cmdlets, permitindo que você possa automatizar quase tudo no seu servidor.

Confira em <http://bit.ly/XV9hc5>

Se você já executa um sistema com o PowerShell instalado, você pode carregar o Windows Power Shell 3.0 você pode clicar no ícone do PowerShell no menu iniciar ou digitar **PowerShell** em uma janela do **prompt** de comando.



Você só irá aprender a usar o PS se você usá-lo e só irá usá-lo se precisar não é mesmo? Siga aqui as dicas e entenda mais o poder do PS e certamente você terá boas razões para começar a usar o PS.

O primeiro passo para começar a usar o PS é trocar o bom e velho `cmd.exe` pelo PS, pois os comandos que usamos com frequência também são aceitos pelo PS.

`DIR`
`CLS`
`IPCONFIG`
`PING`

E muitos outros...

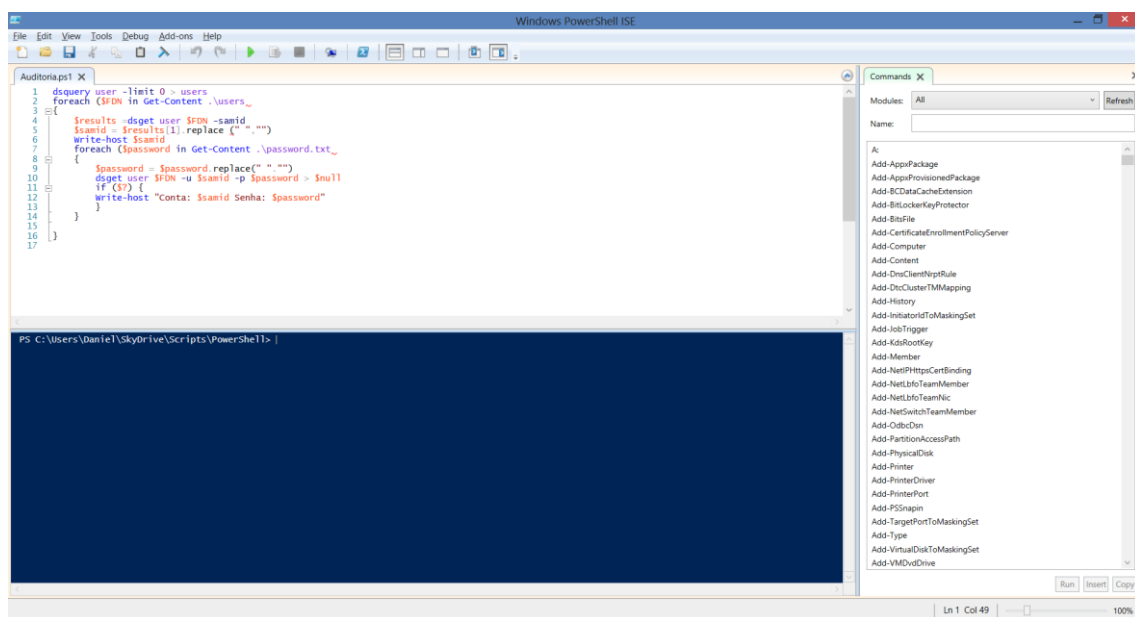
Assim você começará a usar o PowerShell para os comandos que você já usa.

E a ideia desse material é realmente dar o entendimento para que você tenha o acesso a todos os recursos dessa maravilhosa ferramenta.

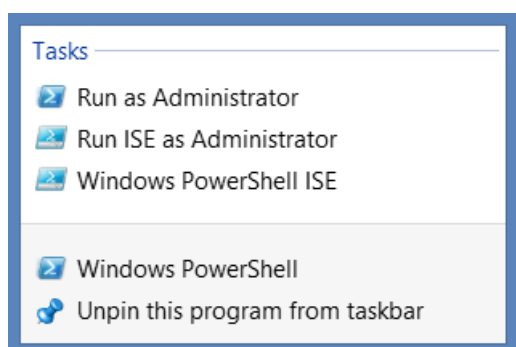
PowerShell ISE

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**



Dica – Clique com o lado direito sobre o ícone do PowerShell que está Fixado na barra de tarefas e você terá a opção “PowerShell ISE” na lista.



O PowerShell ISE é um ambiente bem simples de trabalhar. Existe um painel de digitação, um de execução e um contendo um lista de cmdlets.

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.

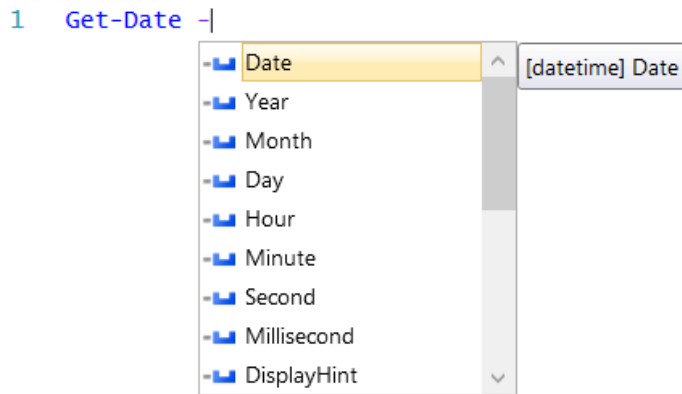







Figura 1 - IntelliSense em ação

A barra do PowerShell ISE além dos comandos padrões como Salvar, Recortar e Colar, também possui alguns comandos que são muito importantes para trabalhar com Scripts.



	Executar o Script (F5)
	Executar a seleção (F8)
	Para a operação (Ctrl+Break)
	Novo PowerShell Remoto (Digite o nome do computador remoto)
	Iniciar uma janela do PowerShell

Existem quatro categorias de comandos do PowerShell:

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

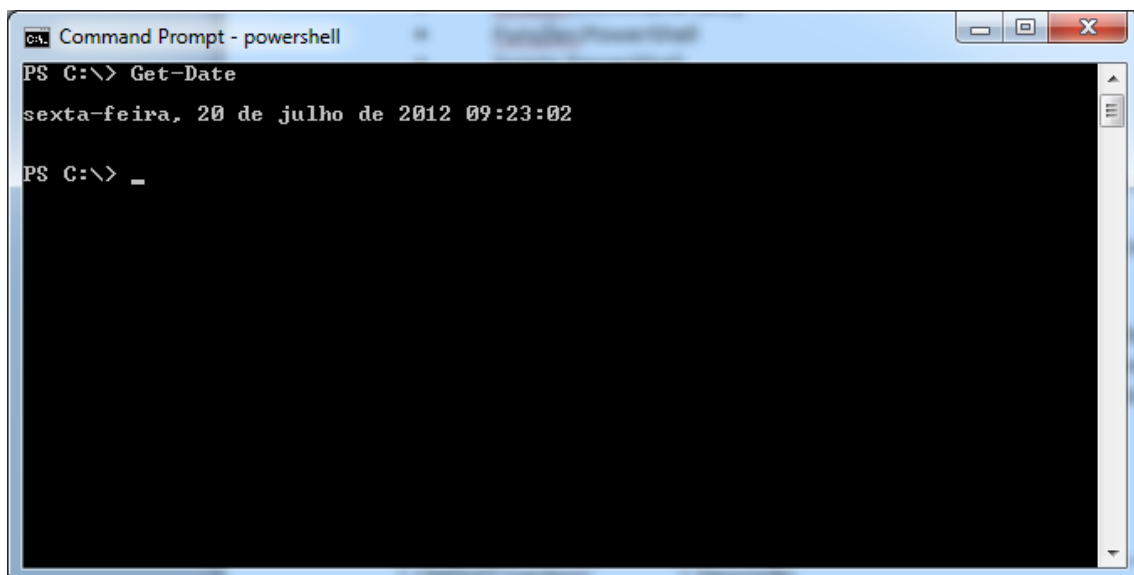
Command-Lets

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 verbo **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.



A seguir uma pequena lista de dos Verbos mais comuns usados no PowerShell:

CMDLET (verbos)	Descrição
Add	Adiciona um recurso ou anexa um item em outro item. Exemplo: Add-Computer Assim como tem o Add existe o Remove .
Clear	Remove um recurso. Exemplo: Clear-Content
Close	Altera o estado de um recurso. Assim como existe Close existe o Open
Format	Formata (arruma) objetos ou saídas em determinados layouts.
Get	Ação que recupera informações, por exemplo, uma lista de objetos. Exemplo: Get-Command

Move	Move recursos de uma localização para outra
New	Cria um novo recurso de um item, como uma variável ou um evento.
Show	Exibe informações relacionadas ao “substantivo”
Start	Inicia uma instancia de um item como um serviço ou processo.
Stop	Para uma instancia de um item como um serviço ou processo.

Esses são apenas alguns dos mais usados cmdlets e que certamente você usará com mais frequência que os demais.

Para ter uma lista completa dos cmdlets existentes use o comando:

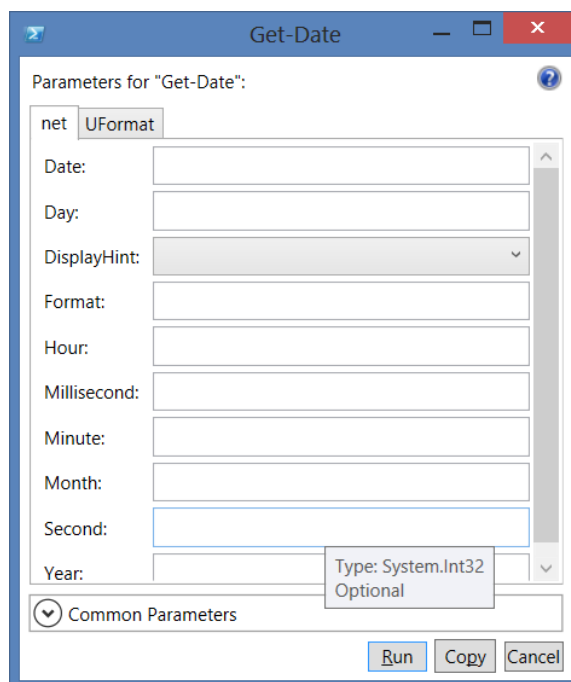
```
get-command -commandtype cmdlet
```

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.

```
Show-Command Get-Date
```



Help Poderoso

Um fator muito importante no uso de um programa ou linguagem de programação é ter uma base de conhecimento completa e atualizada.

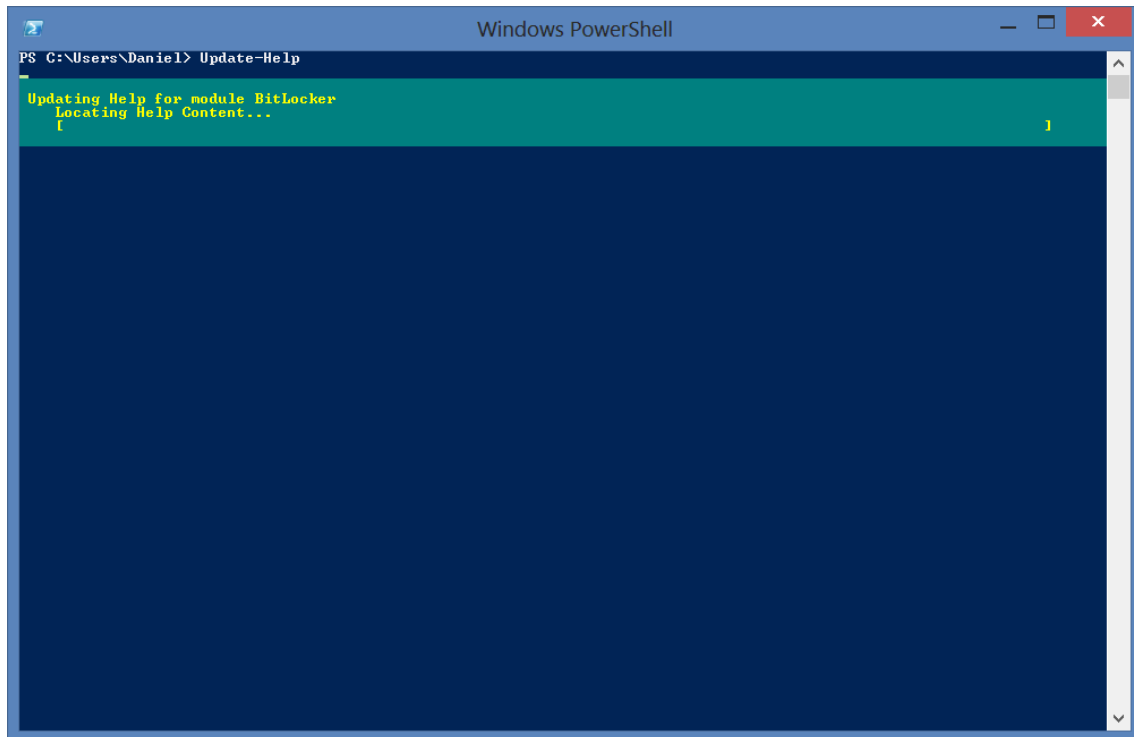
Com o PowerShell 3.0 temos um help atualizável e fácil de se usar.

Marque essa página e quando estiver estudando o resto desse material e tiver dúvida sobre determinado comando ou sintaxe lembre-se que você pode pedir uma ajudinha ao Help do PowerShell.

Com acesso à Internet você pode atualizar a sua base de dados de ajuda.

Lembre-se de executar o PowerShell como administrador.

Update-Help



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

`Get-Help <cmdlet>` - Exibe o help no console

`Get-Help <cmdlet> -Online` - Exibe o help online na biblioteca do TechNet

Digamos que eu queira saber mais sobre **"ForEach-Object"**

`Get-Help ForEach-Object`

`Get-Help ForEach-Object -Online` - Acessa os recursos online

`Get-Help ForEach-Object -Examples` - Exibe exemplos do comando

`Get-Help ForEach-Object -Detailed` - Exibe um help detalhado

`Get-Help ForEach-Object -ShowWindow` - Exibe uma janela

Funções

O comando **Clear-Host** não é um cmdlet, porém possui o mesmo modelo de verbo-substantivo. 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:

<code>LS</code>	– UNIX
<code>DIR</code>	– MS-DOS
<code>Get-ChildItem</code>	– 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.

```
Get-Process | more
Get-Process | Format-List
Get-Process | Format-List | more
```

```
Get-Process | ConvertTo-HTML | Out-File "Processos.html"
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*
```

Out-Default - Envie a saída para o formatador padrão e o cmdlet de saída padrão.

Out-File - Envia a saída para um arquivo.

Out-GridView - Envia a saída para uma tabela interativa em uma janela separada

Out-Host - Envia a saída para a linha de comando.

Out-Null - Apaga saída, em vez de enviá-lo para o console.

Out-Printer - Envia a saída para uma impressora.

Out-String – Envia a saída para um serial de strings.

Exemplos:

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

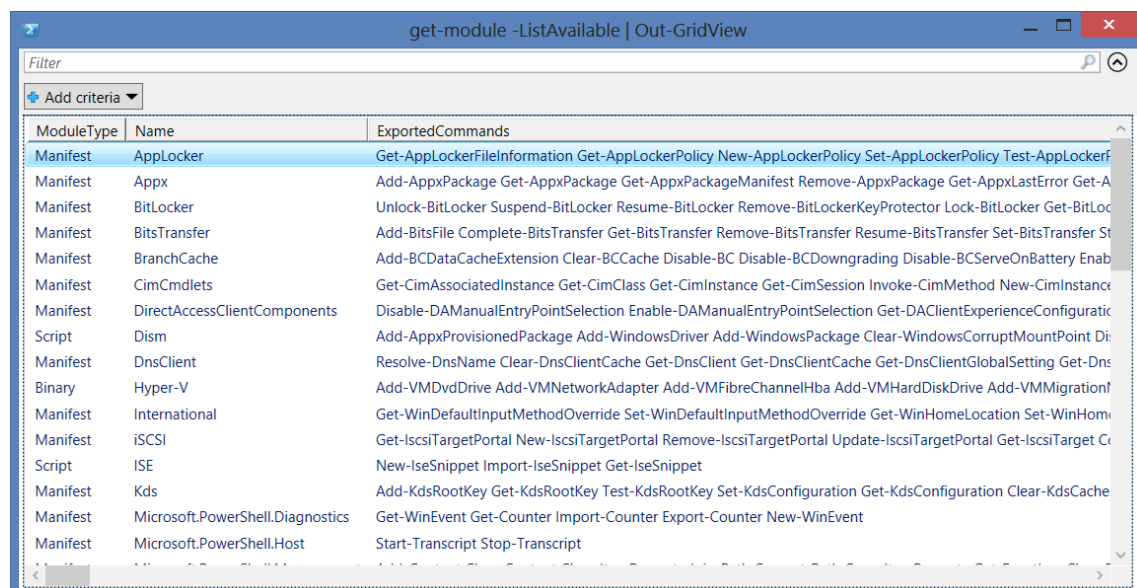


Figura 2 - Comando Get-Module

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:

Operador	Descrição
-lt	Menor que
-le	Menor ou igual
-gt	Maior que
-ge	Maior ou igual
-eq	Igual
-ne	Não igual
-like	Usa wildcards para comparar padrões

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.

```
Windows PowerShell

PS C:\> Get-ChildItem

Directory: C:\

Mode                LastWriteTime         Length Name
----                -
d-----         17/01/2013         22:39         Intel
d-----         26/07/2012         04:33         PerfLogs
d-r-----        22/01/2013         11:25         Program Files
d-r-----        23/01/2013         15:19         Program Files (x86)
d-r-----        17/01/2013         20:26         Users
d-----         21/01/2013         11:22         Windows
-a-----        19/01/2013         20:28         24576 AddWin7inBoot

PS C:\>
```

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

```
Windows PowerShell

PS C:\> Get-ChildItem | where-object {$_.Name -like "Windows"}

Directory: C:\

Mode                LastWriteTime         Length Name
----                -
d-----        21/01/2013         11:22         Windows

PS C:\>
```

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 módulo ServerManager.

Listando módulos

Para listar os módulos do seu servidor ou estação execute o cmdlets:

```
Get-Module -ListAvailable
```

E você verá que existem diversos módulos interessantes e alguns deles vamos utilizar ao longo desse material.

Se você listou os módulos disponíveis em seu sistema e que agora começar a fazer uso de algum deles, você deve usar o comando

```
Get-Command -Module NomeDoModulo
```

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-Module NomeDoModulo
```

Scripts no PowerShell

É possível criar scripts PowerShell e sempre que necessário executa-lo como se fosse o bom e velho **batizinho** (Batch Files).

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.

Script (Hello World)

Eu sempre uso o PowerShell ISE para .

```
1 # Este é o meu primeiro Script do PowerShell
2 Write-Output "Hello world"
```

Salve o arquivo com "Scrip01.ps1" por exemplo e execute com o comando ./script01.ps1 não se esqueça de definir o modo de execução com o cmdlets Set-ExecutionPolicy.

DICA - Ou invés do Write-OutPut, você pode usar o echo. Ex. Echo "Hello word"

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 armazenado 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
```

AppActivate	Method	bool AppActivate (Variant, Variant)
CreateShortcut	Method	IDispatch CreateShortcut (string)
Exec	Method	IWshExec Exec (string)
ExpandEnvironmentStrings	Method	string ExpandEnvironmentStrings (string)
LogEvent	Method	bool LogEvent (Variant, string, string)
Popup	Method	int Popup (string, Variant, Variant, Variant)
RegDelete	Method	void RegDelete (string)
RegRead	Method	Variant RegRead (string)
RegWrite	Method	void RegWrite (string, Variant, Variant)
Run	Method	int Run (string, Variant, Variant)
SendKeys	Method	void SendKeys (string, Variant)
Environment	Property	IWshEnvironment Environment (Variant) {get}
CurrentDirectory	Property	string CurrentDirectory () {get} {set}
SpecialFolders	Property	IWshCollection SpecialFolders () {get}

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

```
$wshell.Popup("O PowerShell É Muito loco mesmo")
```

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

```
$wshell.Run("Calc")
```

Outro método interessante é o **SendKey**.

```
$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 \$

```
1 # Um script com uma variável simples
2 $variavel = "Hello world"
3 Write-Output $variavel
```

Você pode fazer uso de diversas variáveis:

```
1 # Um script com duas variáveis
2 $variavel1 = "Hello"
3 $variavel2 = "world"
4 Write-Output "$variavel1, $variavel2"
```

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

```
1 # Um script com duas variáveis e sem o cmdlet Write-Output
2 $variavel1 = "Hello"
3 $variavel2 = "world"
4 "$variavel1, $variavel2"
```

DICA – Você aprende muito mais fazendo. Tente executar todos os exemplos.

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.

```
1 # Um script que recebe dados do usuário e armazena em uma variável.
2 $variavel1 = "Hello"
3 $variavel2 = Read-Host "Qual o seu nome ? "
4 "$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.

```
1 #Script que armazena uma lista de servidores em um Array e testa todos.  
2 $computadores = @("server1","server2","server3")  
3 $somacomputadores = $computadores.Count  
4 Write-Host "Esse script verifica se $somacomputadores srv estão online"  
5 $computadores  
6 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.

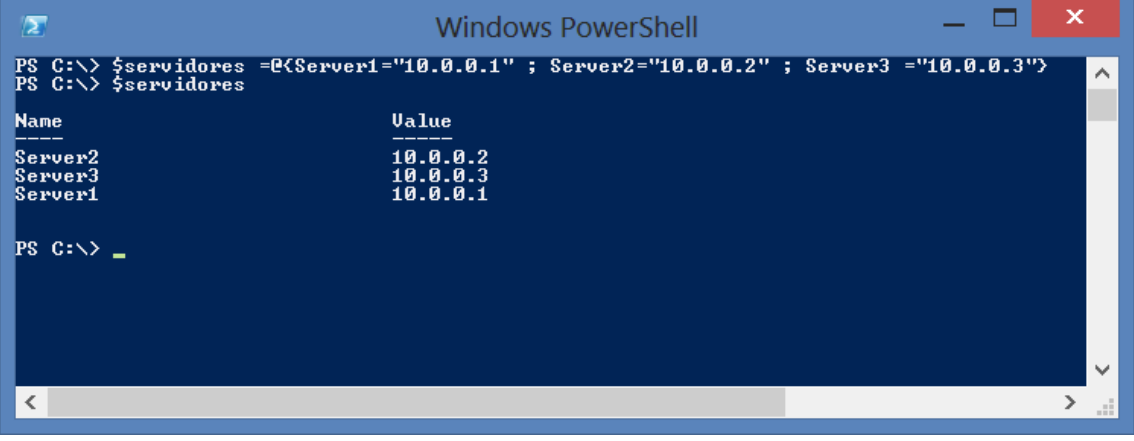
```
1 #Script que armazena uma lista de servidores em um Array.  
2 $computadores = @("server1","server2","server3")  
3 $computadores[0]  
4 $computadores[1]  
5 $computadores[2]
```

Hash Table

Uma tabela de Hash é uma matriz que permite armazenar dados aos pares. Uma "chave-valor".

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
```



```
Windows PowerShell
PS C:\> $servidores = @{"Server1"="10.0.0.1" ; Server2="10.0.0.2" ; Server3 ="10.0.0.3"}
PS C:\> $servidores

Name                Value
-----
Server2             10.0.0.2
Server3             10.0.0.3
Server1             10.0.0.1

PS C:\> _
```

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}
```

PSDefaultParametersValues

Esse novo recurso **PSDefaultParametersValues** 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 e-mails:

```
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"
```

Com o recurso **PSDefaultParameterValues** você salva as configurações

```
$PSDefaultParameterValues = @{
"Send-MailMessage:from" = " seuemail@seudominio.com ";
"Send-MailMessage:SmtpServer"= " smtp.seuservidor.com ";
```

```
"Send-MailMessage:UseSsl"=$true;
"Send-MailMessage:Port"=587;
"Send-MailMessage:Credential" = (Get-Credential seuemail@seudominio.com )
}
```

```
# envio de emails simplificado
Send-MailMessage -to Emaiqueedesejaenvia@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:

```
4 -gt 8
```

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.

Operadores de comparação

Operador	Descrição	Exemplo	Significado e Saída
-lt	Menor que	\$a -lt \$b	A é menor que B? Booleano
-le	Menor ou igual	\$a -le \$b	A for menor ou igual a B? Booleano
-gt	Maior que	\$a -gt \$b	A é maior que B? Booleano
-ge	Maior ou igual	\$a -ge \$b	A é maior ou igual a B? Booleano
-eq	Igual	\$a -eq \$b	A é igua a B? Booleano
-ne	Não igual	\$a -ne \$b	A não é igual a B? Booleano
-like	Como	\$a -like \$b	A inclui um valor como B? Booleano
-notlike	Não como	\$a -notlike \$b	A não inclui um valor como B?
-contains	Contém	\$a -contains \$b	A está contido em B? Booleano
-notcontains	Não Contém	\$a -notcontains \$b	A não está contido em B? Booleano
-match	Coincide	\$a -match \$b	A coincide com B? Booleano
-notmatch	Não coincide	\$a -notmatch \$b	A não coincide com B? Booleano
-replace	Substitui	\$a -replace \$b,c\$	Se A possui strings de B substitua por C

Operadores Aritméticos

O Windows PowerShell também suporte operadores matemáticos.

Os operadores de Adição e Multiplicação aceitam variáveis e arrays.

Operador	Descrição	Exemplo	Significado e Saída
+	Adição	2 + 2	Retorna a soma
/	Divisão	4 / 2	Retorna a quociente
%	Modulo	5 % 2	Retorna o resto da divisão
*	Multiplicação	7 * 8	Retorna o produto
-	Subtração	7 - 5	Retorna a subtração
-	Negação	-7	Transforma o valor em negativo.

Operadores Lógicos

Os operadores lógicos são amplamente utilizados em scripts.

Operador	Descrição	Exemplo	Significado e Saída
and	Operador lógico AND	\$a –and \$b	Verdade (1) se ambas as variáveis de entrada forem verdade
or	Operador lógico OR	\$a –or \$b	Verdade (1) se e somente se pelo menos uma das variáveis de entrada for verdade
not	Operador lógico NOT	\$a –not \$b	Negação (inverso) da variável atua
xor	Operador lógico XOR	\$a –xor \$b	Verdade (1) quando as variáveis assumirem valores diferentes entre si.

Operadores de atribuição

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

Operador	Descrição	Exemplo	Significado e Saída
=	Atribui/Define/Compara valor	\$a = 2	\$a =2
+=	Adiciona um valor	\$a += \$b	\$a = \$a + \$b
-=	Subtrai um determinado valor	\$a -= \$b	\$a = \$a - \$b
*=	Multiplica o valor	\$a *= \$b	\$a = \$a * \$b
/=	Divide o valor	\$a /= \$b	\$a = \$a / \$b
%=	Resultado da operação Modulo	\$a %= \$b	\$a = \$a % \$b
++	Incrementa em mais 1	\$a++	\$a = \$a + 1
--	Decresce em menos 1	\$a--	\$a = \$a – 1

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?

2TB / 250GB

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:\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:\security.log) {echo OK} Else {Get-ChildItem -
File}
```

FOR...FOREACH...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 (início; condição; proximovalor) {Código de repetição}  
for ($a=1; $a -le 10; $a++) {echo 192.168.1.$a}
```

Dica – Não se esqueça de consultar os operadores ainda neste capítulo.

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

While é bem simples e sua estrutura é While (Condição)

Neste exemplo temos uma variável chamada \$i que tem o valor de 1 e “Enquanto” \$i for menor que 5 será exibido.

O código \$i++ incrementa em 1 e o valor \$i somente exibe o valor da variável na tela

```
$i = 1
while ($i -le 5)
{
    $i++ ; $i
}
```

you will find various other examples in this e-book.

Funções

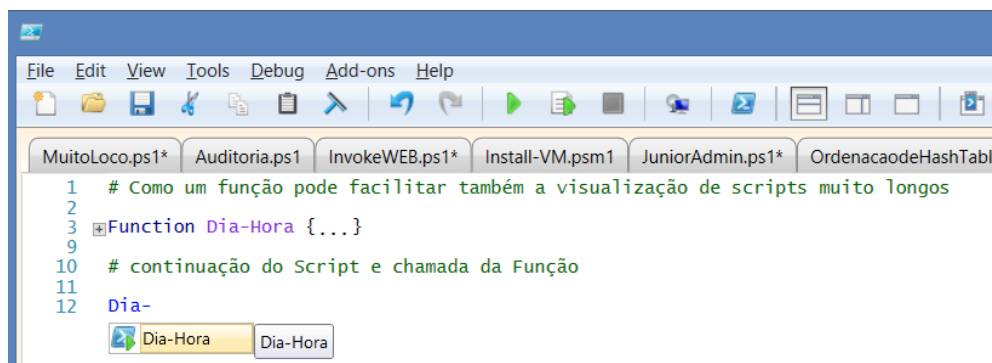
Funções são comandos em um script, que dura apenas durante a sessão em que estiver sendo executado. Quando terminar de usar os comandos em uma função ela não existirá mais na memória e isso é bem simples assim mesmo e fácil de implementar.

```
Function Dia-Hora {
    write-host "Essa função fica na memoria durante a sessão ";
    Get-Date
}
```

Dia-Hora

During the session you can call your functions as many times as necessary, because it remains available as if it were a command. In this example the command is Dia-Hora

Below in the image you can notice that the function was retracted and we can click on the + symbol next to the function to expand and view the code.



The PowerShell ISE auto completes its functions.

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 mesmo:

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
    }
}

#invocar 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 o recurso de Workflow, podemos tomar vantagem do uso da execução paralela. Em scripts sendo executados em vários servidores e que precisam executar várias 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++  
  write-Host $arquivo  
  write-Progress -Activity "Listando Diretorios" -status "Andamento" -  
  PercentComplete (($contagem / $lista.count)*100)  
}
```

Listando Diretorios.
Andamento.



```
fc.exe  
fdBth.dll  
fdBthProxy.dll  
FdDevQuery.dll  
fde.dll  
fddeploy.dll  
fdPHost.dll  
fdPnp.dll  
fdprint.dll  
fdProxy.dll  
FDResPub.dll  
fdSSDP.dll  
fdWCN.dll  
fdWNet.dll  
fdWSD.dll  
feclient.dll  
fhautoplay.dll  
fhcat.dll  
fhcfg.dll  
fhcleanup.dll
```

Running script / selection. Press Ctrl+Break to stop.

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 por falhas na rede. O PowerShell suporta sessões persistentes ou resilientes (**PSSessions**). Uma sessão persistente 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`

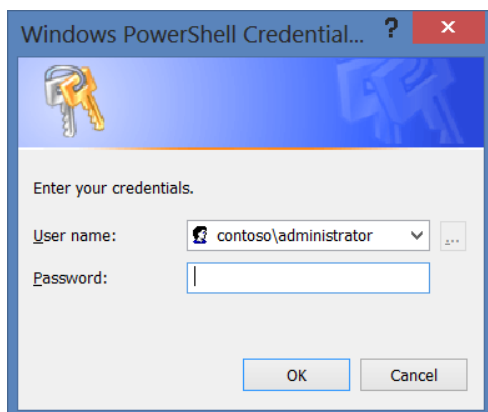
Solicitando credenciais

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

```
$cred = Get-Credential
```

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

```
$admin = Get-Credential contoso\administrator
```



Você pode criar scripts que não solicitam a senha, mas tenha em mente que isso pode não estar em conformidade com certos padrões de segurança.

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 {Get-Process Notepad}
```

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

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

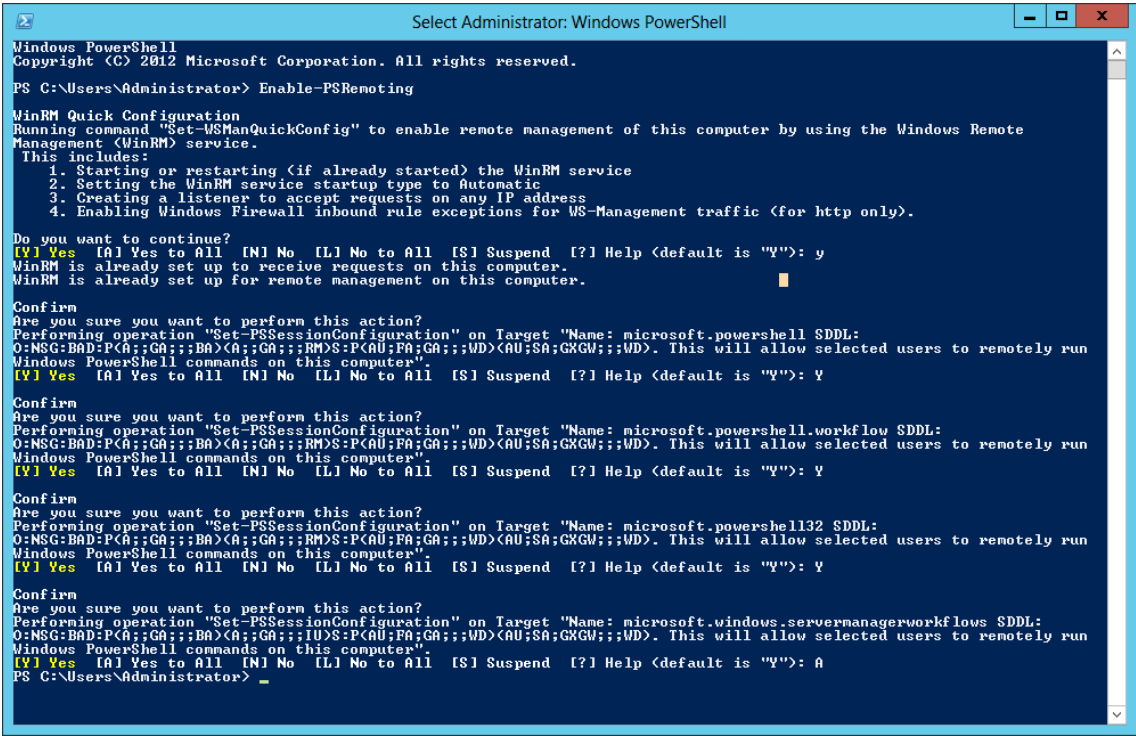
Habilitando gerenciamento Remoto

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).



```
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> Enable-PSRemoting

WinRM Quick Configuration
Running command "Set-WSManQuickConfig" to enable remote management of this computer by using the Windows Remote
Management (WinRM) service.
This includes:
  1. Starting or restarting (if already started) the WinRM service
  2. Setting the WinRM service startup type to Automatic
  3. Creating a listener to accept requests on any IP address
  4. Enabling Windows Firewall inbound rule exceptions for WS-Management traffic (for http only).

Do you want to continue?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y
WinRM is already set up to receive requests on this computer.
WinRM is already set up for remote management on this computer.

Confirm
Are you sure you want to perform this action?
Performing operation "Set-PSSessionConfiguration" on Target "Name: microsoft.powershell SDDL:
O:NSG:BAD:P(A;;GA;;;BA)A;;GA;;;RM)S:P(AU;FA;GA;;;WD)AU;SA;GXGW;;;WD". This will allow selected users to remotely run
Windows PowerShell commands on this computer".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): Y

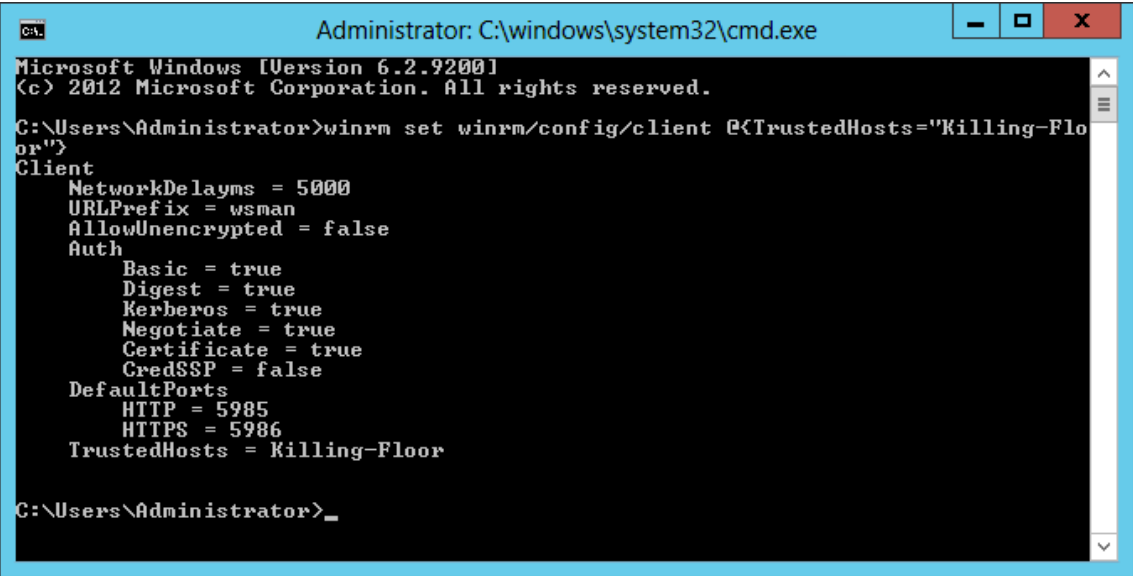
Confirm
Are you sure you want to perform this action?
Performing operation "Set-PSSessionConfiguration" on Target "Name: microsoft.powershell.workflow SDDL:
O:NSG:BAD:P(A;;GA;;;BA)A;;GA;;;RM)S:P(AU;FA;GA;;;WD)AU;SA;GXGW;;;WD". This will allow selected users to remotely run
Windows PowerShell commands on this computer".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): Y

Confirm
Are you sure you want to perform this action?
Performing operation "Set-PSSessionConfiguration" on Target "Name: microsoft.powershell132 SDDL:
O:NSG:BAD:P(A;;GA;;;BA)A;;GA;;;IU)S:P(AU;FA;GA;;;WD)AU;SA;GXGW;;;WD". This will allow selected users to remotely run
Windows PowerShell commands on this computer".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): Y

Confirm
Are you sure you want to perform this action?
Performing operation "Set-PSSessionConfiguration" on Target "Name: microsoft.windows.servermanagerworkflows SDDL:
O:NSG:BAD:P(A;;GA;;;BA)A;;GA;;;IU)S:P(AU;FA;GA;;;WD)AU;SA;GXGW;;;WD". This will allow selected users to remotely run
Windows PowerShell commands on this computer".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): A
PS C:\Users\Administrator>
```


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"}
```

A screenshot of a Windows Command Prompt window titled "Administrator: C:\windows\system32\cmd.exe". The window shows the execution of the command `winrm set winrm/config/client @{TrustedHosts="Killing-Floor"}`. The output displays the configuration for the WinRM client, including network delays, URL prefix, authentication methods (Basic, Digest, Kerberos, Negotiate, Certificate, CredSSP), and default ports (HTTP = 5985, HTTPS = 5986). The TrustedHosts are set to "Killing-Floor".

```
Administrator: C:\windows\system32\cmd.exe
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>winrm set winrm/config/client @{TrustedHosts="Killing-Floor"}
Client
  NetworkDelays = 5000
  URLPrefix = wsman
  AllowUnencrypted = false
  Auth
    Basic = true
    Digest = true
    Kerberos = true
    Negotiate = true
    Certificate = true
    CredSSP = false
  DefaultPorts
    HTTP = 5985
    HTTPS = 5986
  TrustedHosts = Killing-Floor

C:\Users\Administrator>_
```

Sessão Remota

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

Esse comando inicia uma **sessão remota interativa** com o computador remoto.

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`

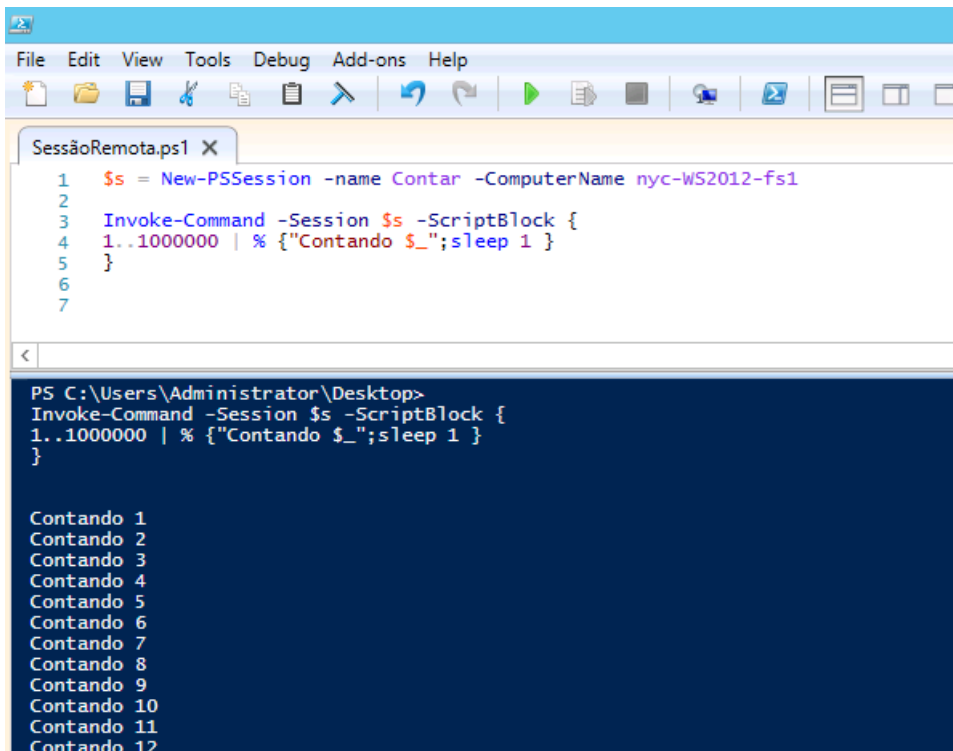
Caso você esteja utilizando scripts ou funções que sejam de longas durações você pode desconectar da sessão que o trabalho continua em execução remota:

```
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 }
}
```



```
File Edit View Tools Debug Add-ons Help
SessãoRemota.ps1 X
1 $s = New-PSSession -name Contar -ComputerName nyc-WS2012-fs1
2
3 Invoke-Command -Session $s -ScriptBlock {
4 1..1000000 | % {"Contando $_";sleep 1 }
5 }
6
7

PS C:\Users\Administrator\Desktop>
Invoke-Command -Session $s -ScriptBlock {
1..1000000 | % {"Contando $_";sleep 1 }
}

Contando 1
Contando 2
Contando 3
Contando 4
Contando 5
Contando 6
Contando 7
Contando 8
Contando 9
Contando 10
Contando 11
Contando 12
```

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

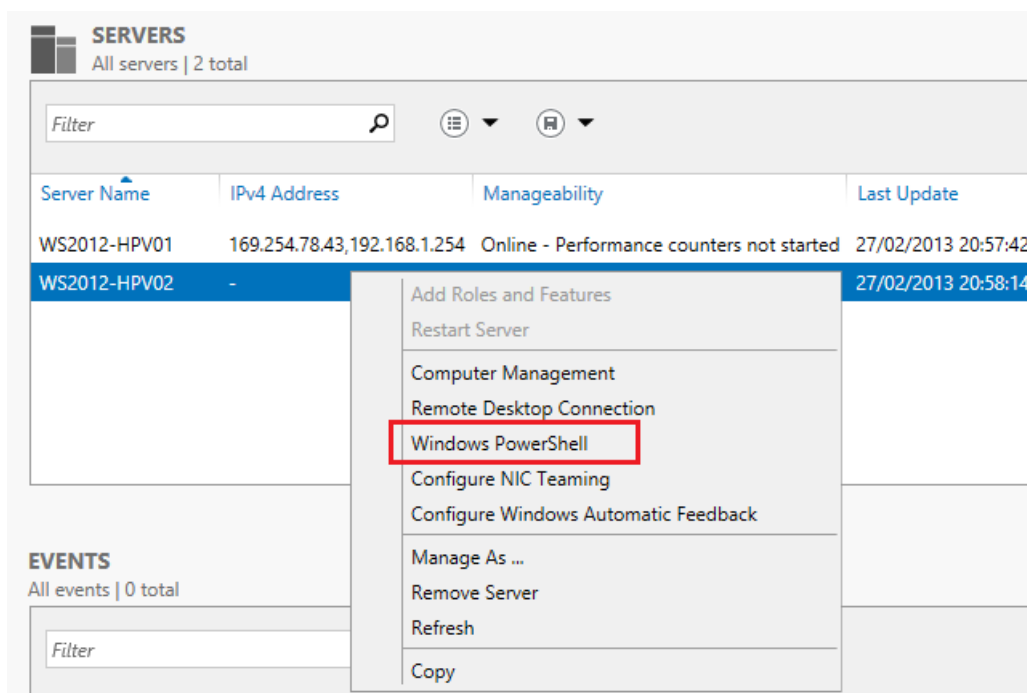
`Connect-PSSession $s`

Depois é só terminar a sessão remota


`Remove-PSSession -ComputerName localhost`

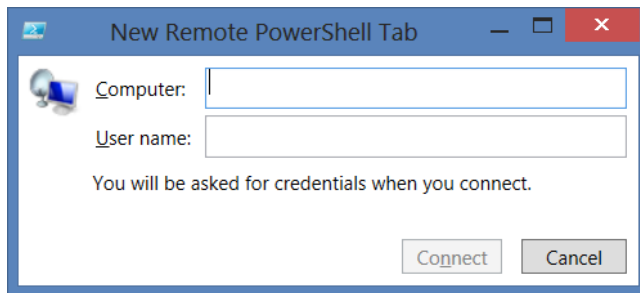
Sessão Remota Server Manager do Server 2012

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

Tenho certeza que você irá gostar desse novo recurso.

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 você 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 **uninstall-PswaWebApplication** e em seguida instale novamente.

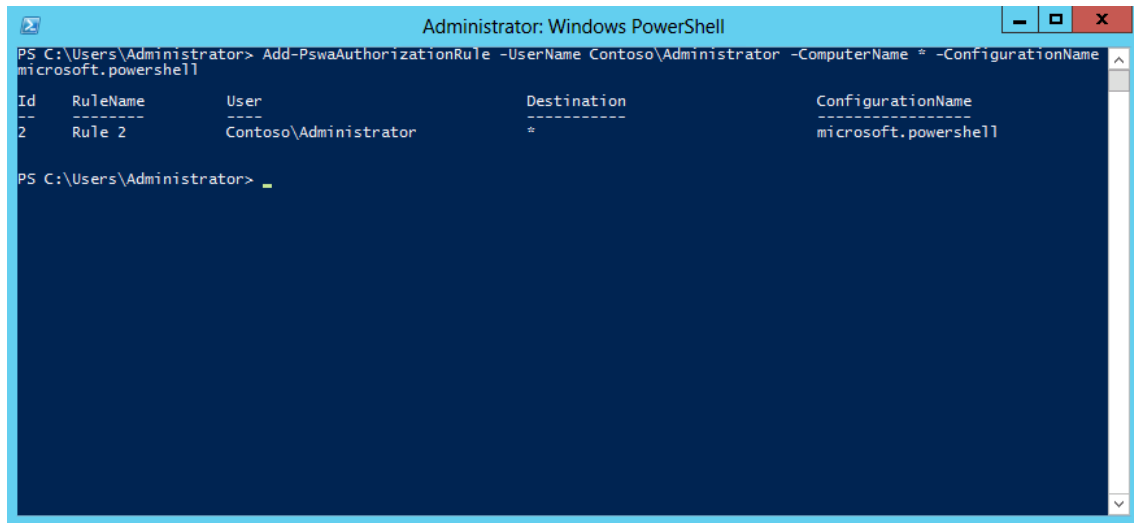
```
Install-PswaWebApplication -webApplicationName PSWA -useTestCertificate
```

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
```

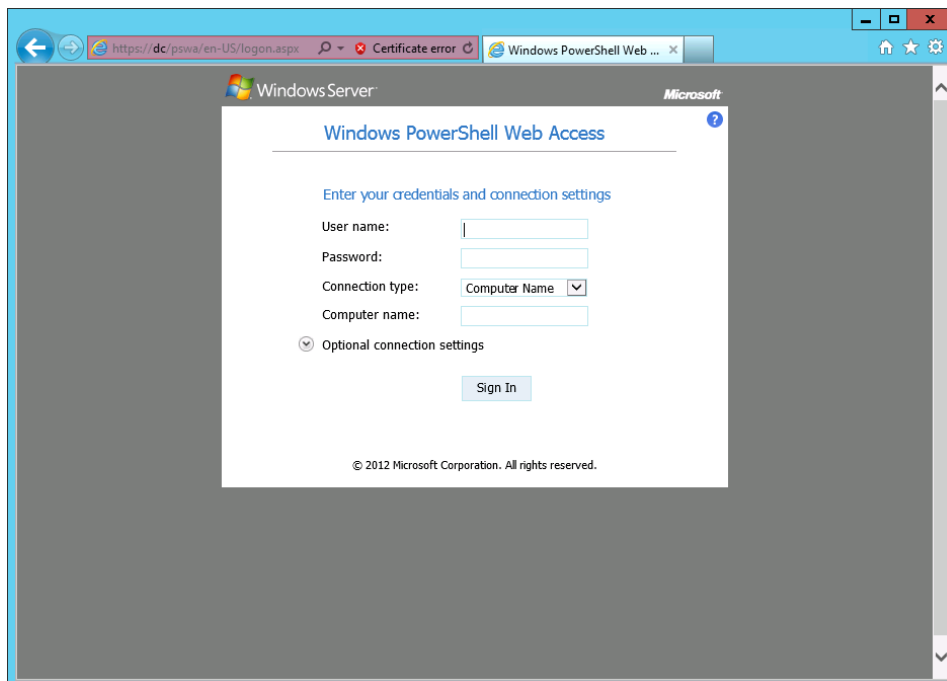


Para maiores detalhes sobre os parâmetros do cmdlet **Add-PswaAuthorizationRule** use o comando:

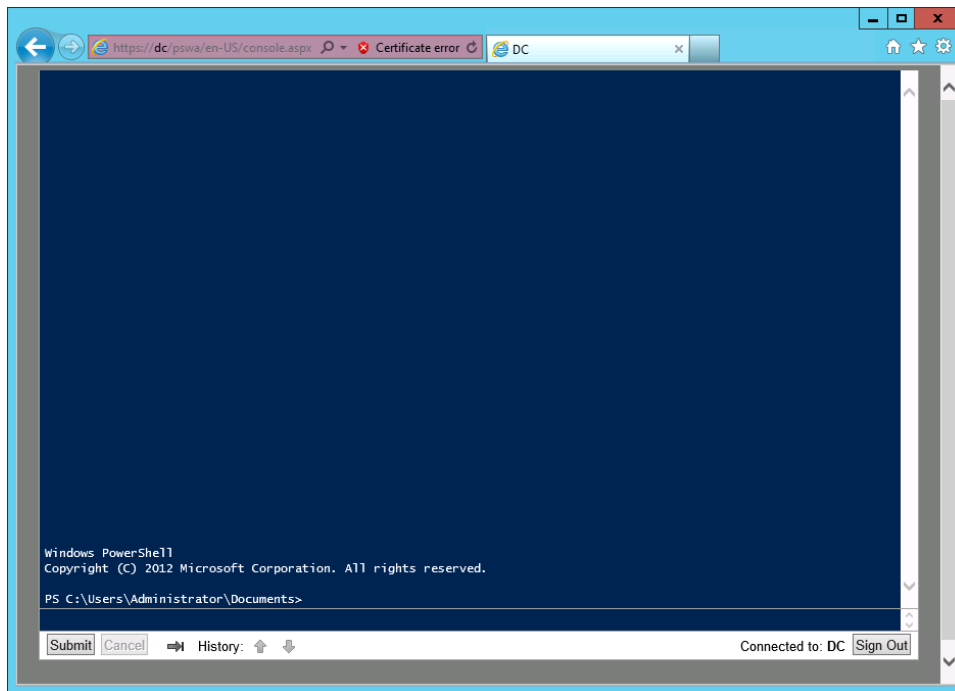
```
show-Command Add-PswaAuthorizationRule.
```

Após as configurações você pode acessar o PSWA usando a seguinte URL:

https://nome_do_servidor/pswa



Forneça suas credenciais de usuário e sua senha.



E você terá o acesso via WEB ao seu PowerShell.

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/10jW1u0>

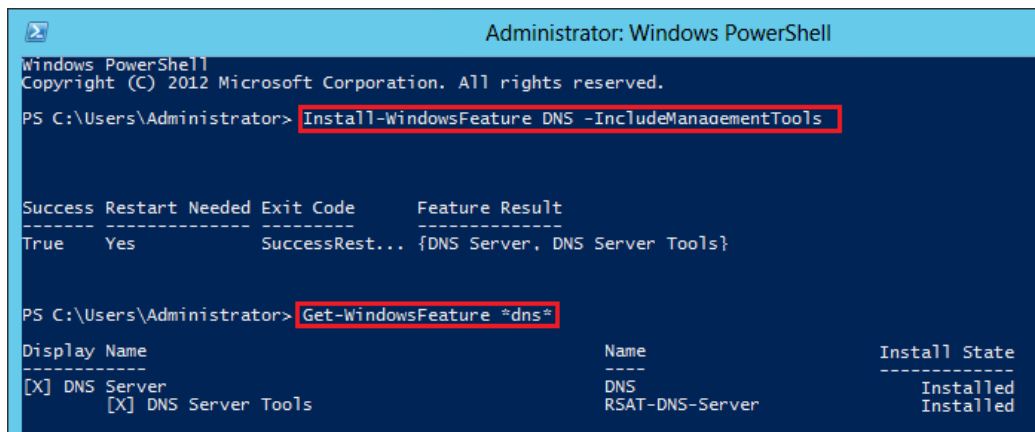
Gerenciamento de Features do Windows Server 2012.

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`



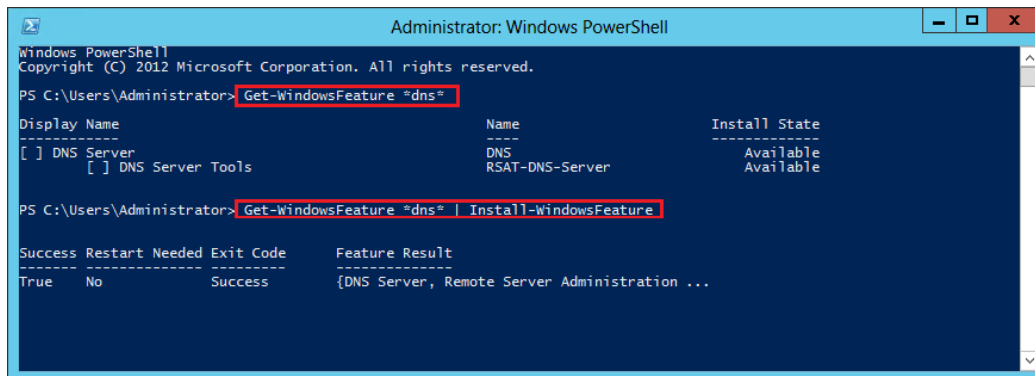
```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> Install-WindowsFeature DNS -IncludeManagementTools

Success Restart Needed Exit Code      Feature Result
-----
True      Yes          SuccessRest... {DNS Server, DNS Server Tools}

PS C:\Users\Administrator> Get-WindowsFeature *dns*

Display Name                                Name                Install State
-----
[X] DNS Server                               DNS                  Installed
[X] DNS Server Tools                         RSAT-DNS-Server     Installed
```



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

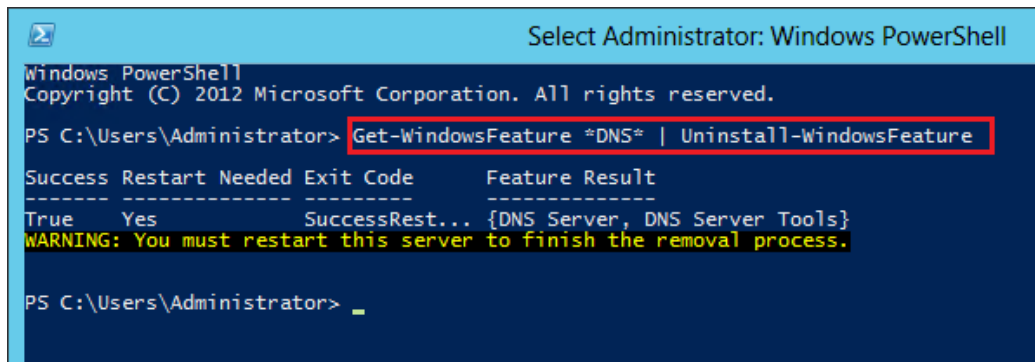
PS C:\Users\Administrator> Get-WindowsFeature *dns*

Display Name                                Name                Install State
-----
[ ] DNS Server                               DNS                  Available
[ ] DNS Server Tools                         RSAT-DNS-Server     Available

PS C:\Users\Administrator> Get-WindowsFeature *dns* | Install-WindowsFeature

Success Restart Needed Exit Code      Feature Result
-----
True      No           Success      {DNS Server, Remote Server Administration ...}
```

O processo de desinstalação pode ser feito da mesma forma.



```
Select Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> Get-WindowsFeature *DNS* | Uninstall-WindowsFeature

Success Restart Needed Exit Code      Feature Result
-----
True      Yes          SuccessRest... {DNS Server, DNS Server Tools}
WARNING: You must restart this server to finish the removal process.

PS C:\Users\Administrator> _
```

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.

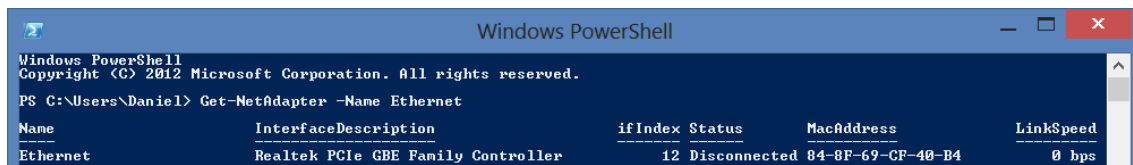
Get-NetAdapter – Exibe informações a respeito dos seus adaptadores de rede.

Listando as placas de redes

Get-NetAdapter

Você pode filtrar o resultado com parâmetros.

Get-NetAdapter -Name Ethernet



```
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

PS C:\Users\Daniel> Get-NetAdapter -Name Ethernet

Name                InterfaceDescription      ifIndex Status      MacAddress      LinkSpeed
-----                -
Ethernet            Realtek PCIe GBE Family Controller 12 Disconnected 84-8F-69-CF-40-B4 0 bps
```

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 fazer uso aqui nesse e-book. Mas você encontrar uma lista completa dos cmdlets em [http://technet.microsoft.com/en-us/library/hh848559\(v=wps.620\).aspx](http://technet.microsoft.com/en-us/library/hh848559(v=wps.620).aspx)

Mesmo assim, vamos enumerar alguns cmdlets que podem ser úteis 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 máquinas virtuais

```
Export-VM -Name "Virtual Machine 1" -Path C:\VM  
Import-VM -Name "VM1" -Path 'C:\VM\413BADFD-E712-4373-8C06-68285BEEE794.XML'
```

Dicas e Truques

Adicionar “Tile” Shutdown/Restart/Logoff no Windows 8

Esse script foi criado para permitir adicionar um botão (Tile) no Windows 8 para aqueles que desejam desligar, reiniciar ou fazer logoff ao alcance de um clique.

Para isso, baixe o módulo **CreateWindowsTil.zip** na Galeria TechNet : <http://bit.ly/10jW1u0>

Importe o módulo com o comando: `Import-Module C:\Script\CreateWindowsTile.psm1`

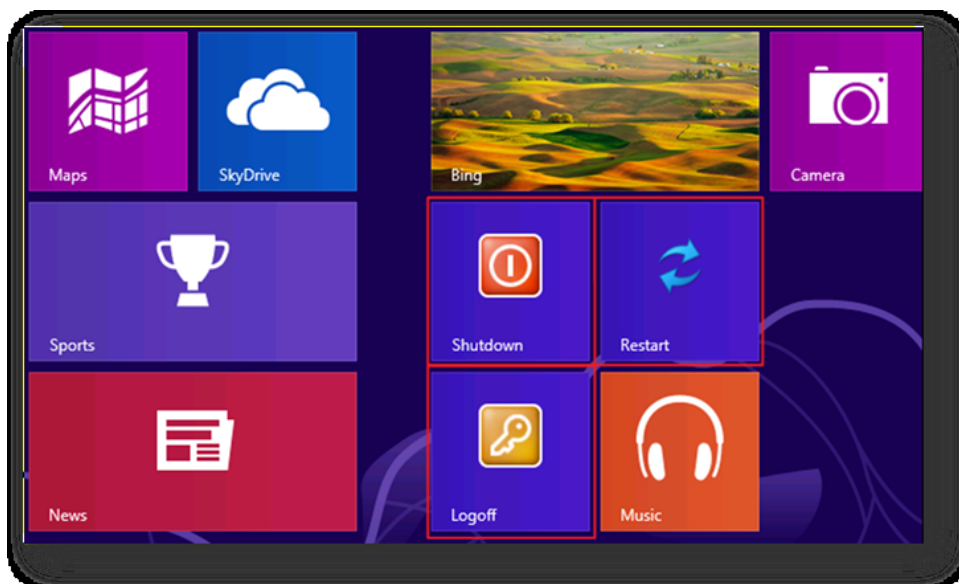
Digite `New-OSCWindowsTile` para instalar as 3 opções: Shutdown, Restart e Logoff.

Ou

`New-OSCWindowsTile –ShutdownTile` para instalar o botão Shutdown

`New-OSCWindowsTile –RestartTile` para instalar o botão Restart

`New-OSCWindowsTile –LogoffTile` para instalar o botão Logoff



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("PowerShell Rocks")
```

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 específico 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 "hk1m:\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 "hk1m:\software\microsoft\windows nt\currentversion\winlogon"  
shell powershell.exe
```

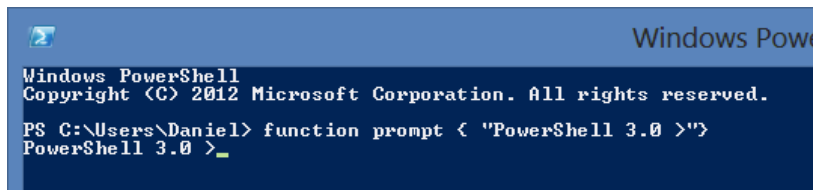
Para voltar o Shell que você salvou:

```
$defaultshell=get-content .\shellpadrao.txt  
set-itemproperty "hk1m:\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 { "O que você quiser > " }
```



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 aniversário 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 notícias de um determinado site com RSS habilitado:

```
$irm = Invoke-RestMethod http://www.mcseolution.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.

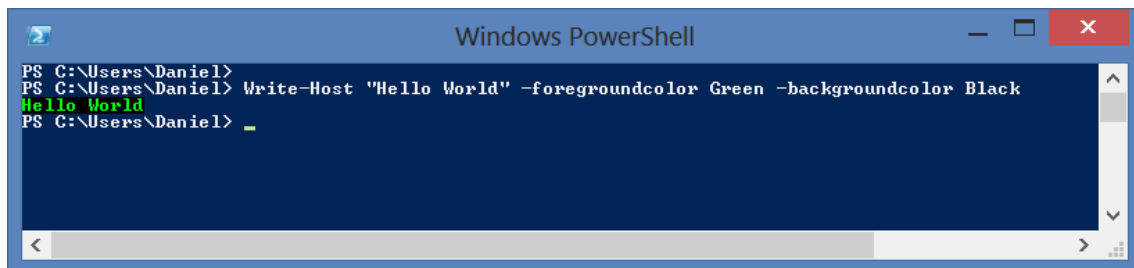
```
Send-MailMessage -from seuemail@seudominio.com `
-SmtpServer smtp.seuservidor.com `
-UseSsl `
-Port 587 `
-Credential (Get-Credential seuemail@seudominio.com) `
-To seuemail@seudominio.com `
-Subject "PowerShell Rocks"
```

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:

Tabela de cores	
Black	DarkGray
DarkBlue	Blue
DarkGreen	Green
DarkCyan	Cyan
DarkRed	Red
DarkMagenta	Magenta
DarkYellow	Yellow
Gray	White

Referências Bibliográficas

Microsoft Windows PowerShell 3.0 First Look - Adam Driscoll – Editora Packt Publishing
Windows PowerShell Cookbook – Segunda edição - Lee Holmes – Editora O'Reilly
Windows PowerShell In Action – Segunda Edição – Bruce Payette – Editora Manning

[10 cool things you can do with Windows PowerShell | TechRepublic](#)
[Beginning Windows Powershell Scripting](#)
[Dmitry's PowerBlog: PowerShell and beyond](#)
<http://bit.ly/X4KAeA>

Sobre o autor

Daniel Donda é MVP na competência Windows Expert- IT Pro.
Possui diversas certificações na área de Infraestrutura:
MCP,MCT,MCITP-EA, MCSA, MCSA+Security, MCSE+Security, MCSE+Messaging
EC-CEH V8 | EC-CEI
BIO online <http://www.mcsesolution.com/curriculo-daniel-donda.html>

Líder do grupo de usuários Microsoft UGSS Mcsesolution (www.mcsesolution.com)
CEO & Founder na empresa Infosec Brasil (www.infosecbrasil.org)
Autor de livros e diversos artigos técnicos (Technet Wiki e Technet Library)

twitter: @danieldonda

Saiba mais sobre o programa MVP <http://mvp.microsoft.com/pt-br/>