

Artigo

Invista em você! Saiba como a DevMedia pode ajudar sua carreira.

Soluções com o Windows PowerShell - Revista Infra Magazine 6

Este artigo retrata uma abordagem simplificada de primeiro contato com o novo shell e linguagem de script da Microsoft, com a finalidade de apresentar e exemplificar sua utilização no dia a dia do administrador de redes.



Anotar



Marcar como concluído

Artigos



Soluções com o Windows PowerShell - Revista Infra Magazine 6

Do que se trata o artigo:

Este artigo retrata uma abordagem simplificada de primeiro contato com o novo shell e linguagem de script da Microsoft, com a finalidade de apresentar e exemplificar sua utilização no dia a dia do administrador de redes.

DEVMEDIA

As novas versões do Windows Server 2008 e produtos Microsoft estão vindo com um gerenciamento via Windows PowerShell. Nestas condições, faz-se necessário o entendimento e aprendizado do mesmo para a administração de uma rede com ambiente Microsoft.

Resumo DevMan:

O Microsoft Windows PowerShell é um novo prompt de comando do Windows, mais poderoso e flexível para administração de servidores, aplicações e scripts. É sobre ele que abordaremos neste artigo.

Por muitos anos certamente administradores gerenciaram seus servidores e produtos Microsoft utilizando GUI (*Graphical User Interface* – Interface Gráfica de Usuário), *cmd.exe*, VBScript e talvez outras tecnologias complementares.

Em função de maior flexibilidade e extensão, a Microsoft vem tentando fazer com que o Microsoft Windows PowerShell seja a ferramenta de gerenciamento de melhor escolha. Para isso, os novos produtos de servidores requerem Windows PowerShell para administração em função da sua facilidade no gerenciamento e na manutenção.

O Microsoft Windows PowerShell é um novo prompt de comando do Windows que comparado ao antigo *cmd.exe* é muito mais poderoso. É voltado ao uso de scripts para manutenção de sistemas por parte de administradores, oferecendo maior controle do sistema e agilidade na automatização de tarefas como backup, transferência de arquivos, parametrização de aplicações como Microsoft Exchange, entre outras.

Criado em uma estrutura do Framework .NET, o Windows PowerShell ajuda os profissionais de Tecnologia da Informação (TI) e usuários a controlar e automatizar a administração do Sistema Operacional Windows e os aplicativos

que são executados no mesmo. É sobre este tema que o conteúdo deste artigo é baseado.

Windows PowerShell Versão 2.0

Até o momento da escrita deste artigo, o Windows PowerShell encontrava-se na versão 2.0, que oferece diversas melhorias, a saber:

1. **Comunicação remota:** Permite executar comandos em um ou mais computadores remotamente, a partir de um computador que esteja executando o Windows PowerShell;
2. **ISE (Integrated Scripting Environment – Ambiente de Script Integrado):** Permite executar comandos interativos, editar e depurar scripts em um ambiente gráfico, incluindo ao ambiente sintaxe realçada por cor, depuração gráfica, suporte a Unicode e ajuda contextual do código;
3. **Módulos:** Permite ao desenvolvedor e ao administrador dividir e organizar o código escrito em Windows PowerShell em unidades reutilizáveis e independentes;
4. **Trabalhos em background:** Permite a execução de comandos PowerShell em segundo plano, sem interação com o console;
5. **Novos *cmdlets*** (pronuncia-se *command-lets*): São comandos integrados que estão incluídos no Windows PowerShell. Assemelham-se aos comandos internos em outros shells, como por exemplo, o **dir**, encontrado no *cmd.exe*. Podem ser chamados diretamente da linha de comando do Shell e executados no contexto desse interpretador, e não como um processo separado.

Quina

A instalação do Windows PowerShell é feita a partir do Windows XP com Service Pack 3, e, também, nas versões superiores do Windows, tendo como pré-requisito a instalação do Microsoft .NET Framework 2.0 com Service Pack 1 ou superior.

Após a instalação, começaremos então a entender como funciona o Windows PowerShell. Para inicialização do mesmo, basta executá-lo a partir do menu *Iniciar*. Para isto, clique em *Iniciar* > *Executar*. Observe que aparecerá uma caixa solicitando a digitação do comando a ser executado, que é *Powershell.exe*. Feito isso, será aberto um *shell* para execução dos comandos e criação de scripts, conforme ilustrado na **Figura 1**.

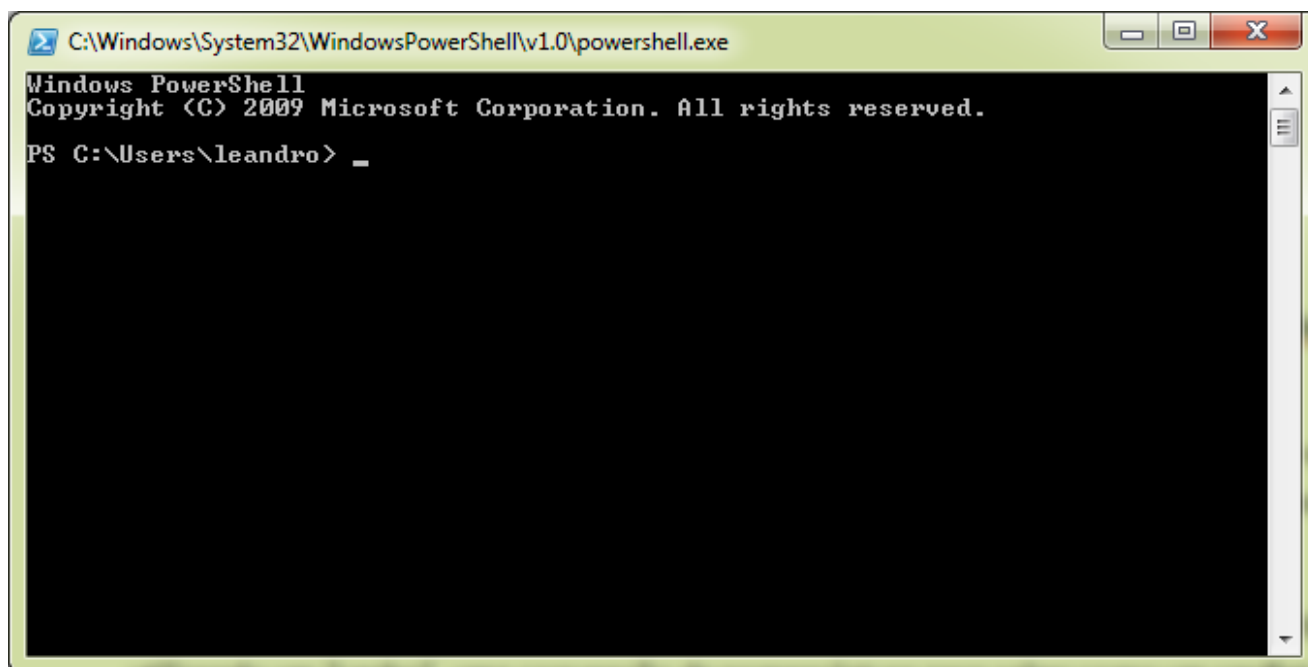


Figura 1. Prompt de comando do Windows PowerShell quando inicializado.

Comandos internos do PowerShell

Como foi dito, o Windows PowerShell é repleto de comandos chamados *cmdlets*. Existem mais de 250 comandos do Windows PowerShell para aprender. No entanto, veremos que pela estrutura com que o Windows PowerShell foi desenhado, facilmente encontraremos os comandos que precisamos com uma

Ao contrário da maioria das outras interfaces de linha de comando, o Windows PowerShell foi padronizado utilizando verbos auxiliares no início dos comandos. Uma convenção de nomenclatura para saber como um *cmdlet* irá agir. Esta padronização é importante pois simplifica a curva de aprendizado, e a documentação do comando fornece uma descrição do que o *cmdlet* faz e formas de utilizá-lo.

As palavras reservadas dos *cmdlets* do Windows Power Shell são baseadas no inglês, por isto tem-se no início dos comandos verbos como **Get**, **Set**, **Add**, dentre outros. Por exemplo, se desejar ver uma lista de *cmdlets* disponíveis no Windows PowerShell, basta digitar o *cmdlet*:

1 | `Get-Command <enter>`

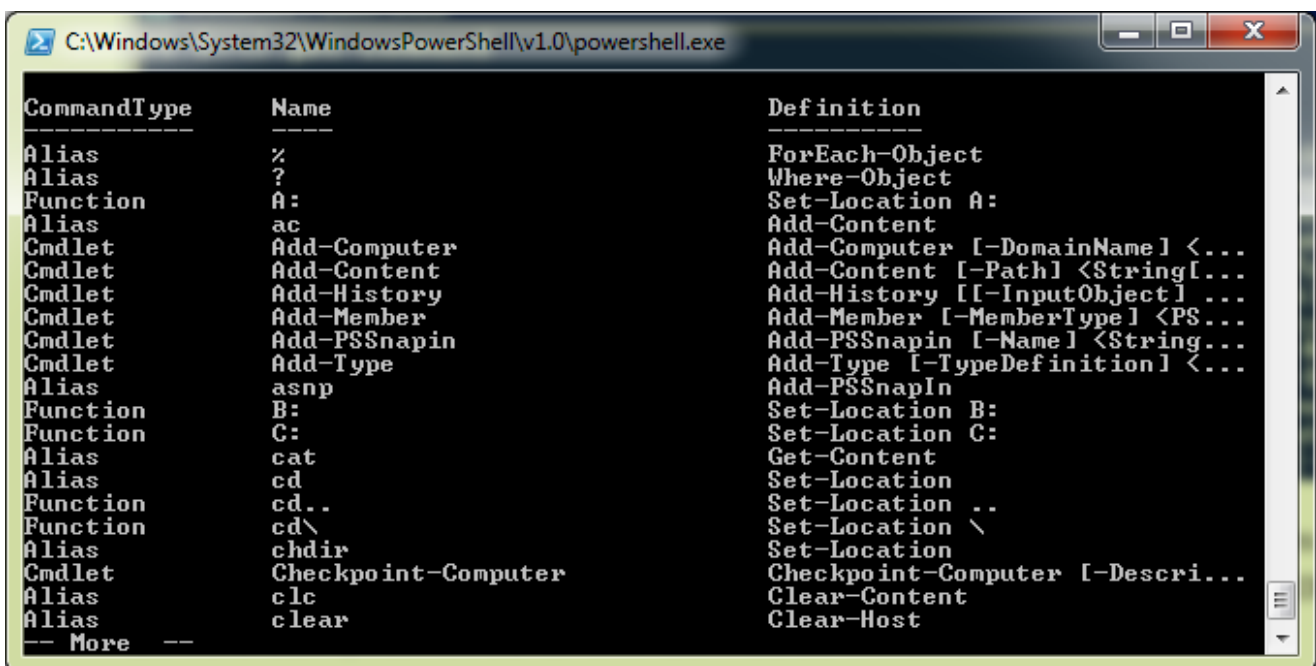


Figura 2. Lista de cmdlets exibidos com o uso do comando Get-Command.

Na saída de **Get-Command**, apresentada na **Figura 2**, é possível notar que existe uma série de outros comandos que começam com **Get**, **Set**, **Clear**, **Add**, **Start**, etc. Isso simplifica bastante a curva de aprendizado em função da padronização dos

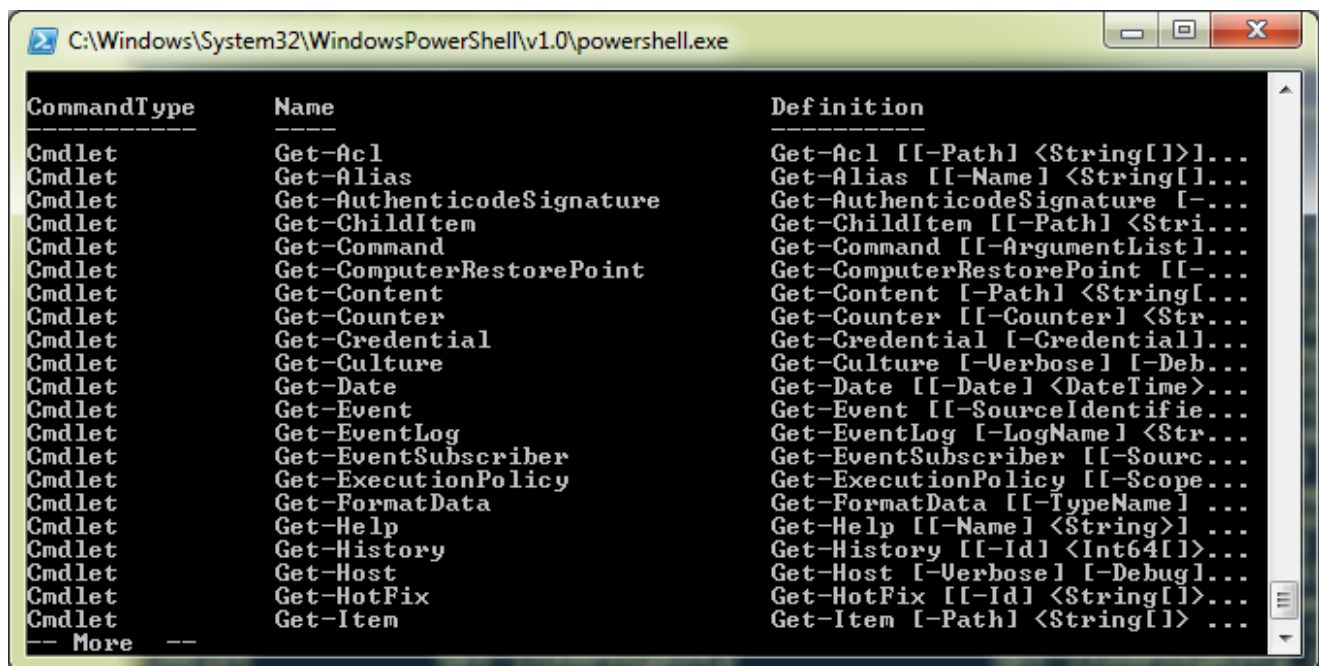
Ainda com a janela aberta, vamos dar sequência ao nosso estudo, familiarizando-se ainda mais com o Windows PowerShell e entendendo melhor esse *cmdlet* chamado **Get-Command**.

Para quem já utilizou outra linguagem de script e está acostumado com comandos, percebe que muitos retornam informações sem a necessidade de parâmetros.

No uso do comando **Get-Command** pode-se incrementá-lo passando, por exemplo, um parâmetro que ajude a listar comandos com verbos específicos, ou seja, se houver necessidade de resgatar uma lista de comandos iniciados com algum verbo, como **Get**, **Set**, **New** ou outro qualquer, basta informá-lo via parâmetro.

Tomando como exemplo que se queira uma lista de comandos que contenham o verbo **Get**, basta executar o mesmo comando utilizando o parâmetro “**-Verb**” e o verbo a ser listado. Por exemplo:

```
1 | Get-Command -Verb Get <enter>
```



CommandType	Name	Definition
Cmdlet	Get-Acl	Get-Acl [[-Path] <String[]>]...
Cmdlet	Get-Alias	Get-Alias [[-Name] <String[]>]...
Cmdlet	Get-AuthenticodeSignature	Get-AuthenticodeSignature [-...]
Cmdlet	Get-ChildItem	Get-ChildItem [[-Path] <Stri...]
Cmdlet	Get-Command	Get-Command [[-ArgumentList]...
Cmdlet	Get-ComputerRestorePoint	Get-ComputerRestorePoint [[-...]
Cmdlet	Get-Content	Get-Content [[-Path] <String[]>]...
Cmdlet	Get-Counter	Get-Counter [[-Counter] <Str...]
Cmdlet	Get-Credential	Get-Credential [-Credential]...
Cmdlet	Get-Culture	Get-Culture [-Verbose] [-Deb...]
Cmdlet	Get-Date	Get-Date [[-Date] <DateTime>]...
Cmdlet	Get-Event	Get-Event [[-SourceIdentifie...]
Cmdlet	Get-EventLog	Get-EventLog [-LogName] <Str...]
Cmdlet	Get-EventSubscriber	Get-EventSubscriber [[-Sourc...]
Cmdlet	Get-ExecutionPolicy	Get-ExecutionPolicy [[-Scope...]
Cmdlet	Get-FormatData	Get-FormatData [[-TypeName] ...]
Cmdlet	Get-Help	Get-Help [[-Name] <String>] ...]
Cmdlet	Get-History	Get-History [[-Id] <Int64[]>]...
Cmdlet	Get-Host	Get-Host [-Verbose] [-Debug]...
Cmdlet	Get-HotFix	Get-HotFix [[-Id] <String[]>]...
Cmdlet	Get-Item	Get-Item [-Path] <String[]> ...]
-- More --		

Como se pode notar na **Figura 3**, com pouco contato e alguns minutos dedicados ao Windows PowerShell, facilmente pode-se identificar uma lista de comandos que anteriormente era difícil de ser encontrada no *cmd.exe*.

Bom, já sabemos como listar comandos e filtrá-los de acordo com nossa necessidade. Deste modo, se em algum momento precisarmos adquirir alguma informação do sistema operacional, como por exemplo, a data, é necessário obter (verbo **Get**) uma informação que neste caso trata-se da data. Então basta procurar na lista de saída do **Get-Command -Verb Get** se tem algo relacionado à data. Nesse caso, veremos na listagem o comando **Get-Date** e facilmente receberemos a informação da data.

Help

Um ponto fundamental e indispensável para o aprendizado e uso da ferramenta é o sistema de ajuda, e neste contexto novamente o Windows PowerShell foi muito bem arquitetado.

Seguindo a linha de raciocínio e uso dos verbos auxiliares empregado no começo do comando, nota-se que para obter (**Get**) ajuda (**Help**) podemos utilizar novamente o comando **Get-Command -Verb Get**. Neste caso já é possível deduzir algo como **Get-Help**, e, sem muita surpresa, verificamos que o comando existe. Para facilitar, podemos digitar **Get-<TAB>** para completar o comando.

Até agora utilizamos o comando **Get-Command** para listar os *cmdlets* internos do PowerShell que contenham o verbo auxiliar **Get**. Aprendemos também que este e outros comandos podem receber parâmetros. Nesse ponto uma dúvida pode ser levantada com relação a quais parâmetros um comando espera receber. Para isto, utiliza-se o *cmdlet* **Get-Help**.

1 | Get-Help

Na saída do mesmo é possível identificar outro padrão interessante do Windows PowerShell que também visa reduzir a sua curva de aprendizado. Ao digitar **Get-Help**, uma saída na tela é exibida com uma série de informações sobre como utilizá-lo.

Nestas informações está contida uma pequena descrição do que faz o comando, a sintaxe, exemplos e comentários, mantendo sempre esta ordem e padrão para qualquer comando.

Como exemplo, digite o código abaixo no prompt do PowerShell:

```
1 | Get-Help Get-Command <enter>
```

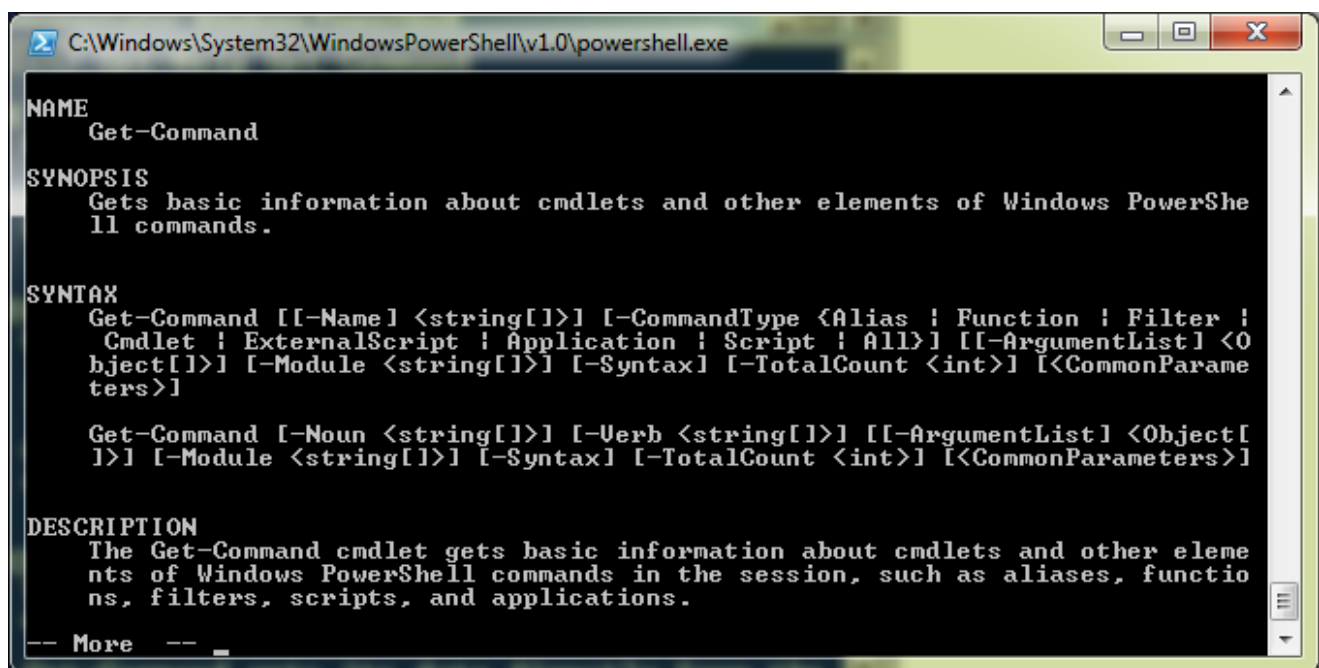


Figura 4. Informações obtidas para ajuda no uso do comando Get-Command.

Com a execução do comando **Get-Help Get-Command**, conforme ilustrado na **Figura 4**, é exibida uma saída com todas as informações relacionadas ao *cmdlet*.

Com apenas dois *cmdlets* observamos que é possível aprender sobre diversos outros sem a necessidade de dedicar muito tempo navegando pela Internet em busca de informações. Com esta estrutura o Windows PowerShell proporciona mais objetividade na busca de comandos que resolvam nossos problemas ou melhore nossos resultados.

Desta forma, quando se deseja obter informações mais detalhadas sobre um *cmdlet*, basta usar o **Get-Help** passando como parâmetro o nome do *cmdlet*.

Para finalizar esta etapa, digite no prompt do PowerShell o comando: **Get-Help Get-Command -Examples**. Com ele pode-se obter diversos exemplos de uso do *cmdlet* especificado. Veja a **Figura 5**.

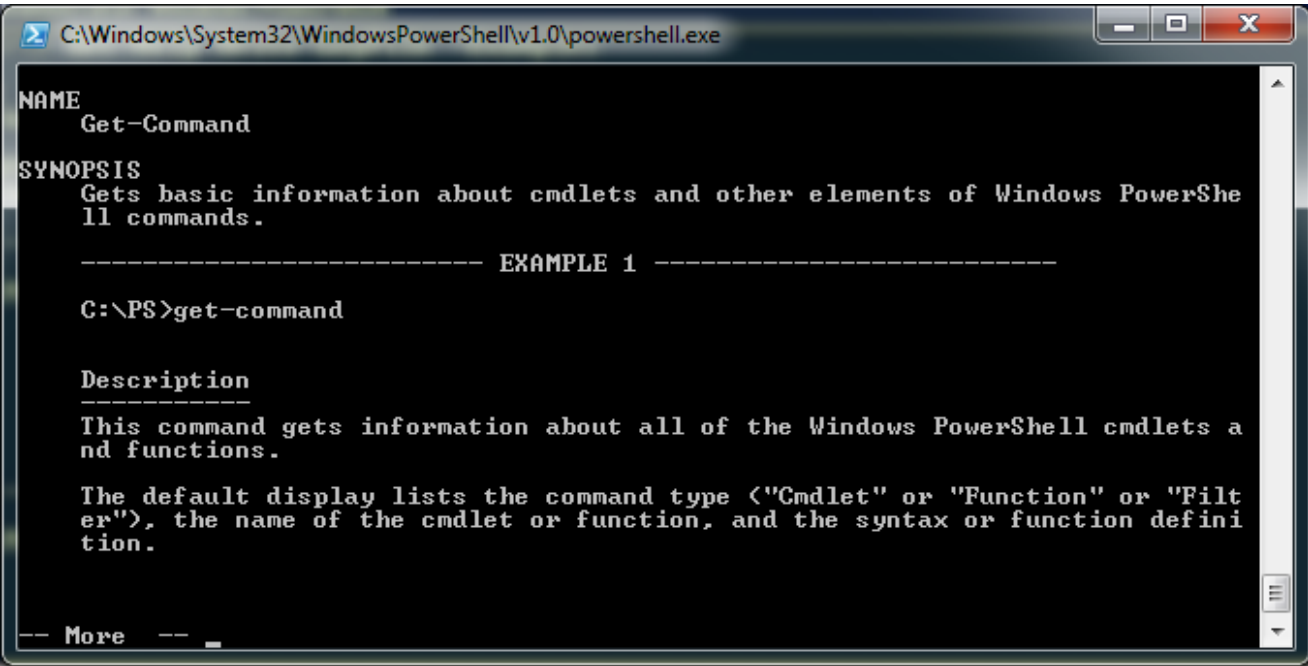


Figura 5. Lista de exemplos para uso do comando Get-Command.

A **Tabela 1** apresenta um breve comparativo entre *cmdlets* e outros comandos internos do *cmd.exe* e do Unix Shell.

Power Shell	Power Shell	Cmd.exe	Unix	Descrição
-------------	-------------	---------	------	-----------

Get-ChildItem	gci, dir, ls	dir	ls	Lista todos os arquivos do diretório atual.
Get-Content	gc, type, cat	type	cat	Lista o conteúdo do arquivo.
Get-Command	gcm	help	help	Lista os comandos disponíveis.
Get-Help	help, man	help	man	Comando de ajuda.
Clear-Host	cls, clear	cls	clear	Limpa a tela.
Copy-Item	cpi, copy, cp	copy	cp	Copia um ou mais arquivos.
Move-Item	mi, move, mv	move	mv	Move um arquivo/diretório para um novo local.
Remove-Item	ri, del, erase, rmdir, rd, rm	del, erase, rmdir, rd	rm, rmdir	Deleta um arquivo/diretório.
Rename-Item	rni, ren, mv	ren, rename	mv	Renomeia um arquivo/diretório.
Get-Location	gl, pwd	cd	pwd	Mostra o diretório

Set-Location	sl, cd, chdir	cd, chdir	cd	Troca o diretório atual.
Write-Output	echo, write	echo	echo	Mostra o valor de uma variável no prompt de comando.
Get-Process	gps, os	tlist, tasklist	ps	Lista todos os atuais processos em execução.
Stop-Process	spp, kill	kill, taskkill	kill	Finaliza um processo em execução.
Select-String	n/a	find, findstr	grep	Encontra uma linha utilizando parâmetros comparativos.

Tabela 1. Cmdlets comparados com outros comandos internos do cmd.exe e do Unix Shell.

Uma linguagem inteligente

Até o momento foi apresentado bem pouco do que o PowerShell pode fazer para os administradores. Vimos apenas dois *cmdlets* que nos mostraram uma lista gigante de outros *cmdlets*. De qualquer maneira, ainda não interagimos com muita inteligência, ou seja, apenas obtemos informações. No dia-a-dia da

Scripts

O Windows PowerShell além de possuir comandos internos é uma linguagem de script do tipo dinâmica, em que não é necessário declarar o tipo de variável que será utilizada. Com isto, permite programar complexas operações usando *cmdlets*.

Por se tratar de uma poderosa linguagem de script, o Windows PowerShell suporta o uso de variáveis, constantes, funções, verificações (**if-then-else**), iterações (**while**, **do**, **for** e **foreach**), tratamentos de erros/exceções e interação com o Framework .NET.

Com base nisso, serão apresentados a seguir alguns aspectos de desenvolvimento para melhor interação e aproveitamento do Windows PowerShell na criação e execução de scripts.

Variáveis

Antes de iniciar a criação de scripts, é importante destacar que uma variável tem a funcionalidade de armazenar um determinado valor (não necessariamente numérico) durante algum tempo ou até o final da execução do script. Isto vai depender da finalidade com que a mesma foi criada.

Para criar uma variável basta acrescentar no início do nome o caractere “\$”. Desta maneira a mesma armazenará um valor. Considerando que nossa variável se chama nome, devemos criá-la como “\$nome”.

Para criar e atribuir um valor à variável **\$nome**, por exemplo, digite dentro do prompt de comando do Windows PowerShell o seguinte:

Para testarmos o valor armazenado na variável, digite no prompt do Windows PowerShell o nome da variável (“\$nome”). O valor obtido é uma string (conjunto de caracteres) contendo o nome “Leandro”. Na prática poderia conter, por exemplo, o nome de *login* de um usuário.

```
1 | $nome
```

Para atribuir a data do dia de hoje à variável **\$data** utilizando o *cmdlet* **Get-Date**, digite no prompt do Windows PowerShell o seguinte:

```
1 | $data = Get-Date
```

Como já informado, o Windows PowerShell não é uma linguagem fortemente tipada, em que se faz necessário declarar a variável antes de ser utilizada, ou seja, não é necessário informar que a variável **\$nome** é do tipo string. Vê-se claramente que em nenhum momento foi necessário declarar a variável e informar qual será o seu tipo (**int**, **string**, **date**, etc.). Nesse caso, foi executado o *cmdlet* **Get-Date**, atribuindo a saída à variável **\$data**, deixando que o Windows PowerShell defina que tipo será a variável dinamicamente.

Se desejar escrever na tela o valor armazenado na variável, pode-se utilizar o *cmdlet* **Write-Output**, da seguinte forma:

```
1 | Write-Output $data
```

Até então estamos manipulando valores sem nenhum tratamento antes de atribuir o mesmo à variável.

Uma atividade comum na criação de scripts é fazer a verificação de um resultado antes de armazená-lo e/ou manipulá-lo. Para isto, veremos declarações que

ajudam a comparar o valor obtido com o valor desejado e então tomar alguma ação, antes de atribuir o valor à variável.

If, Then e Else

Para tratarmos a informação comparando-a com algum outro valor desejado, utilizam-se as declarações **If**, **Then** e **Else**, que podem ser entendidas como **Se**, **Então** e **Senão**, respectivamente. Com essas declarações é possível comparar um resultado obtido com um resultado esperado e então armazená-lo.

Utilizando a variável criada anteriormente chamada `$data` (onde se encontram armazenadas informações relacionadas à data de hoje, que foi atribuída através do `cmdlet` **Get-Date**), pode-se comparar o valor contido na variável com um dia da semana para identificar se os dias são iguais. Em caso afirmativo, pode ser exibida na tela uma mensagem dizendo que o dia é igual, caso contrário nada será exibido.

Para isso, digite no prompt de comando do Windows PowerShell o seguinte:

```
1 | $data = Get-Date <enter>  
2 | if ($data.DayOfWeek -eq (Get-Date).DayOfWeek) {Write-Output "O dia
```

Verifica-se que se trata de algo bem simples, porém identificamos algumas particularidades na linguagem do Windows PowerShell muito interessantes.

Pelo fato das variáveis serem do tipo dinâmico, `$data` passa a ser do tipo **Date** ao receber o retorno do `cmdlet` **Get-Date**. Se porventura atribuíssemos uma string, `$data` seria do tipo string, ou seja, dinamicamente se modificaria para atender o valor recebido. Como se trata de uma variável do tipo **Date**, é possível manipulá-la como tal, chamando assim propriedades como **DayOfWeek** que informa o dia da

Se por acaso explorarmos a variável **\$nome**, que recebeu uma string, veremos que esta, de forma semelhante, contém métodos e propriedades que nos permitem manipulá-la como uma string.

Para exemplificar, entre com o seguinte comando no prompt do Windows PowerShell:

```
1 | $nome.ToUpper()
```

Tem-se como resultado o valor da variável com todas as letras em maiúsculo.

Por acaso, se você colocar um ponto depois do nome da variável **\$data** e pressionar o TAB, poderá verificar que a mesma possui uma série de propriedades, tais como: **Day**, **Hour**, **Year**, entre outras.

O mesmo vale para o *cmdlet* **Get-Date**, que desta vez foi chamado entre parênteses para garantir que o Windows PowerShell obtenha a data e mostre apenas a propriedade **DayOfWeek**.

De posse das duas informações, podemos comparar se elas são iguais. No Windows PowerShell tem-se uma lista de operadores, conforme destacado na **Tabela 2**, que fazem essas comparações baseadas em palavras reservadas. No exemplo apresentado, foi comparado se os valores das variáveis são iguais, utilizando para isto a palavra reservada **-eq**.

Para entender o uso destas palavras reservadas, na **Tabela 2** será atribuído o valor 5 a uma variável chamada **\$variavel**, e esta será comparada a um valor com o objetivo de se obter Verdadeiro ou Falso como resultado em várias situações.

Parâmetro	Ação	Exemplo	Resultado
-----------	------	---------	-----------

		\$variavel -eq 4	
-ne	not equal (diferente)	\$ variavel = 5 ; \$variavel -ne 4	Verdadeiro
-gt	greater than (maior)	\$ variavel = 5 ; \$variavel -gt 4	Verdadeiro
-ge	greater than or equal to (maior ou igual)	\$ variavel = 5 ; \$variavel -ge 5	Verdadeiro
-lt	less than (menor)	\$ variavel = 5 ; \$variavel -lt 5	Falso
-le	less than or equal to (menor ou igual)	\$ variavel = 5 ; \$variavel -le 5	Verdadeiro
-like	wildcard comparison	\$ variavel = "This is Text" ; \$variavel -like "Text"	Falso
-notlike	wildcard comparison	\$ variavel = "This is Text" ; \$variavel - notlike "Text"	Verdadeiro
-match	regular expression comparison	\$ variavel = "Text is Text" .	Verdadeiro

-notmatch	regular expression comparison	\$ variavel = "This is Text" ; \$variavel - notmatch "Text\$"	Falso
-----------	----------------------------------	---	-------

Tabela 2. Exemplos de uso de operadores lógicos.

Embora esta seja uma particularidade do Windows PowerShell, vimos novamente que a arquitetura da linguagem permite facilmente entender e aprender o que estamos fazendo.

Até este ponto vimos como comparar uma única vez a variável, esperando um resultado falso ou verdadeiro. No entanto é muito comum interagir ao ponto de fazer mais que duas comparações. Para isso devemos avaliar o uso do **else** e do **then**.

Em nosso exemplo, a data armazenada na variável *\$data* é comparada ao dia da semana para identificar se são iguais. Em caso afirmativo, uma mensagem é exibida dizendo: “O dia armazenado na variável é o mesmo dia da data de hoje”. Caso contrário, nada é informado.

Vamos então aprender o que podemos fazer para tratar o caso da condição ser falsa. Para isso utiliza-se a declaração “**else**”. Neste caso dizemos que se (**if**) o dia armazenado na variável *\$data* for igual a hoje, devemos emitir uma mensagem, senão, (**else**) emitir outra mensagem.

Para tanto, dentro do prompt de comando do Windows PowerShell, entre com o código:

```
1 | $data = Get-Date 02/26/1985 <enter>  
2 | if ($data.DayOfWeek -eq (Get-Date).DayOfWeek) {Write-Output "O dia da
```

É interessante notar que desta vez foi enviado como parâmetro ao *cmdlet* **Get-Date** uma data, e a partir de então se utilizou esta data para obter informações, como o dia da semana da mesma.

Neste caso a data armazenada na variável foi diferente da data de hoje e por isso o comando emitiu a mensagem: “O dia armazenado na variável não é o mesmo dia da data de hoje”.

Dadas essas condições, é possível fazer diversas comparações a fim de se obter um resultado esperado, ou tratar uma condição não prevista.

Ambiente de Script Integrado (ISE)

Certamente ficou claro que se acrescentarmos mais informações em nossa linha de comando começa a ficar mais complexo o entendimento e consequentemente a manutenção dos scripts. Além disso, caso o prompt do Windows PowerShell seja fechado, as informações serão perdidas, pois não são persistidas ou salvas em nenhum arquivo. Para evitar este problema conheceremos o Ambiente de Script Integrado (ISE), que permite a criação de scripts de forma organizada e com ferramentas de teste e manipulação.

O ISE do Windows PowerShell permite criar, executar e depurar comandos e scripts através de um ambiente integrado completo. Veja a **Figura 6**.

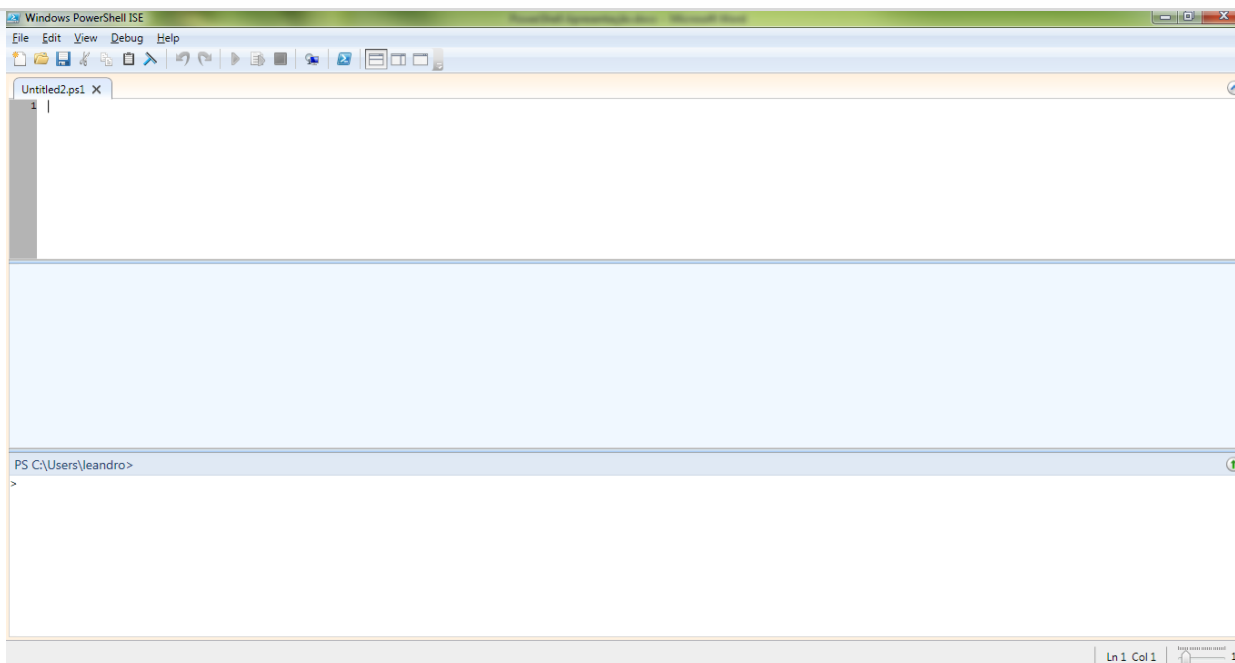


Figura 6. Ambiente de Script Integrado quando inicializado.

Note que o ambiente está organizado em três seções horizontais, sendo a primeira seção o arquivo onde será criado o script, a segunda é onde será apresentado o resultado da execução do mesmo e a última seção representa o prompt do Windows PowerShell.

Agora, ao invés de utilizar o prompt de comando do Windows PowerShell, vamos criar nosso script dentro do ISE. Assim sendo, dentro do arquivo *Untitled1.ps1*, que foi aberto ao executar o ISE, escreva o código da **Listagem 1**.

Listagem 1. Exemplo de criação de um script no ISE.

```
1 | $data = Get-Date 02/26/1985
2 |     if ($data.DayOfWeek -eq (Get-Date).DayOfWeek)
3 |     {
4 |         Write-Output "O dia armazenado na variável é o mesmo dia da data c
5 |     } else {
6 |         Write-Output "O dia armazenado na variável não é o mesmo dia da da
7 |     }
```

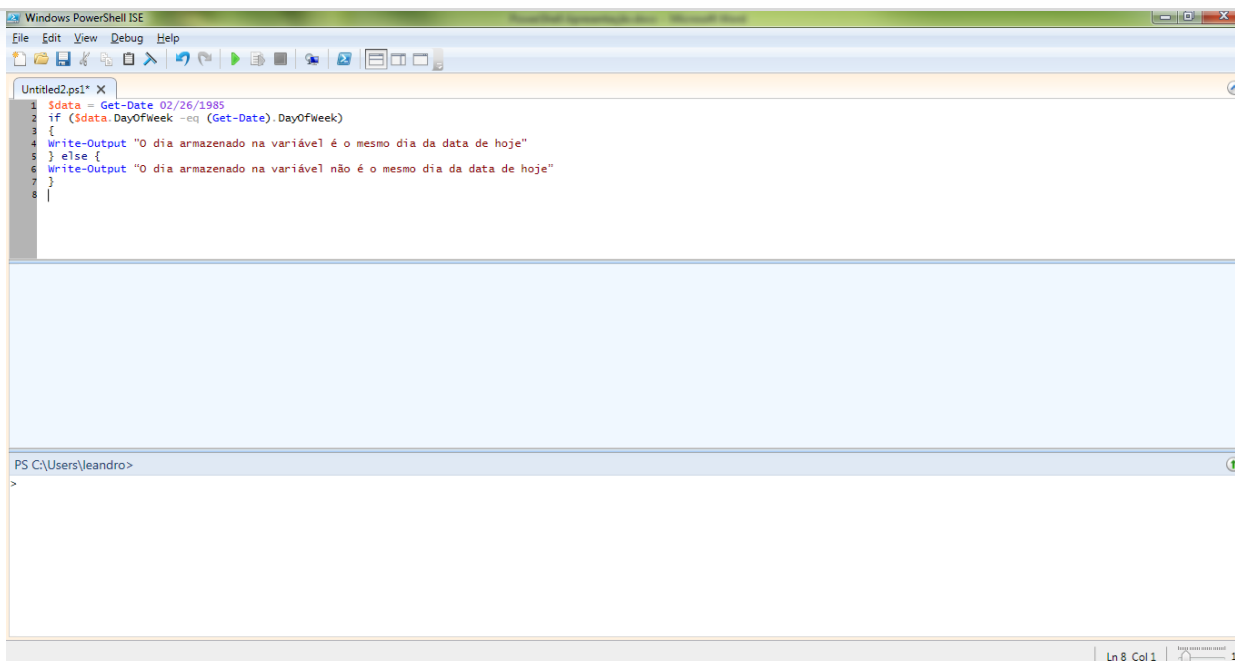


Figura 7. Ambiente de Script Integrado utilizado para criação de um script.

Logo é possível notar (na **Figura 7**) que as palavras ganham destaque por meio de cores. Isto ajuda a identificar palavras reservadas e nomes de variáveis. Outro ponto interessante dentro do ambiente é o uso da tecla <TAB>, que assim como no PowerShell, facilita a interação dos comandos com o usuário.

Após a digitação do código, clique no botão *Run Script* ou simplesmente aperte a tecla *F5*.

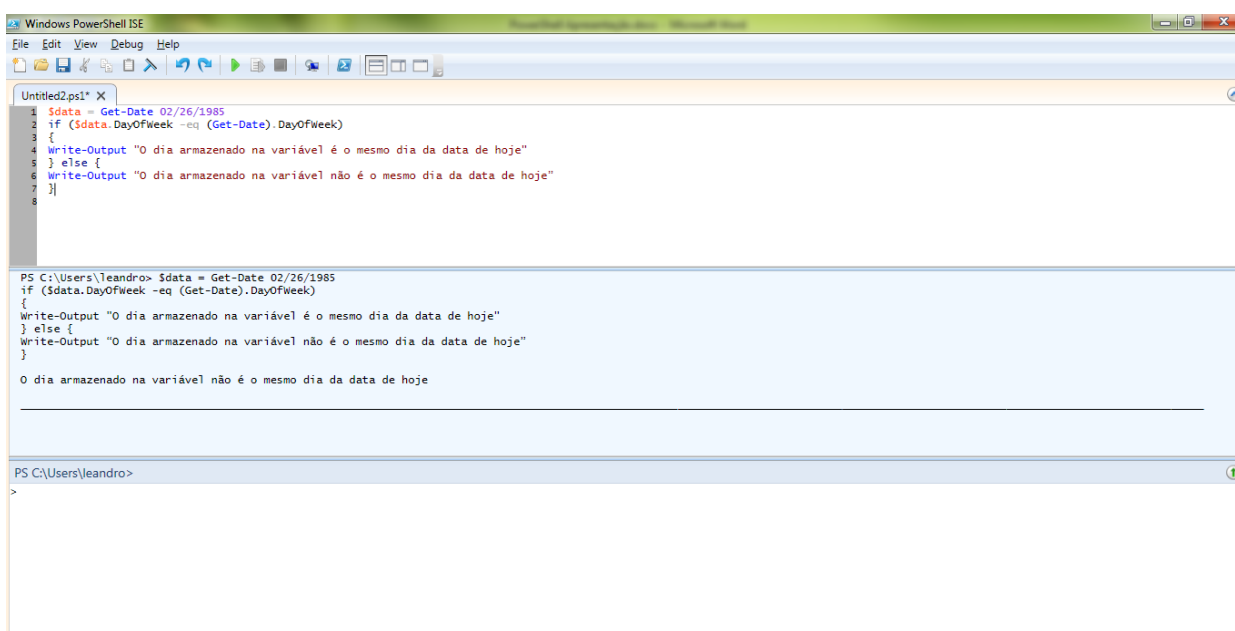


Figura 8. Ambiente de Script Integrado utilizado para execução de um script.

Pode-se observar na **Figura 8** que na segunda parte horizontal a saída do comando foi impressa. Já na terceira parte horizontal, pode-se utilizar um prompt para manipular a variável e/ou comandos sem a necessidade de alterar o script, permitindo assim fazer melhores testes.

Para finalizar, o script pode ser salvo em um arquivo para ser executado sempre que necessário.

Diferentemente de quando utilizávamos apenas o prompt do Windows PowerShell, com a ferramenta ISE é possível salvar o script com a extensão *.ps1*. Para isso, basta acessar o menu *File*, clicar na opção *Save* e dar um nome ao arquivo.

Antes de tentar executar o script fora do Ambiente de Script Integrado, é importante saber que por padrão o Windows PowerShell possui a política de execução de script restrita ou desabilitada, ou seja, diferente dos comandos digitados ao longo do artigo, o Windows PowerShell não permite executar scripts (arquivos preparados com uma sequência de comandos). Com isso, mesmo que você esteja como administrador, não será possível executar o script.

Para verificar a configuração da política de execução de script, digite no Windows PowerShell o *cmdlet* **Get-ExecutionPolicy**. Por padrão esta vem parametrizada como **Restricted** (desabilitada). Para modificá-la basta usar o comando **Set-ExecutionPolicy** passando como parâmetro a nova regra para execução dos scripts. A seguir são destacadas as regras mais utilizadas:

- **Restricted:** Não permite a execução de nenhum script;
- **Unrestricted:** Permite rodar todos os scripts e arquivos de configuração;

- **AllSigned**: Requer que todos os scripts e, inclusive arquivos de configuração, sejam assinados por um autor de confiança com um certificado digital;
- **RemoteSigned**: Requer que todos os scripts baixados da Internet sejam assinados por um autor de confiança.

Como estamos trabalhando em um ambiente de teste, podemos utilizar a regra de execução **Unrestricted**, que permite executar qualquer script sem fazer a verificação de confiança. Assim, dentro do prompt de comando do Windows PowerShell, informe:

```
1 | Set-ExecutionPolicy Unrestricted.
```

Agora, basta executar o script criado. Para isso, no prompt do Windows PowerShell, digite o comando:

```
1 | powershell.exe MeuScriptPS.ps1
```

Conclusão

Como dito no artigo, o Windows PowerShell é hoje a ferramenta escolhida pela Microsoft para gerenciamento dos servidores e também dos novos produtos como Microsoft Exchange, Microsoft Lync Server, SharePoint, Windows Server Core, entre outros.

Ao longo do artigo foi possível observar que o Windows PowerShell nos permite um aprendizado muito rápido, em função do mesmo utilizar verbos auxiliares no início dos *cmdlets*. Com isto os comandos aproximam-se da nossa linguagem, facilitando o entendimento dos mesmos para a criação e manutenção de scripts.

ajudam no entendimento de outros comandos, facilitando o conhecimento e exploração da linguagem. Abordou também o uso de scripts para automatização de tarefas e apresentou uma ferramenta capaz de ajudar na confecção e gerenciamento dos scripts.

Tecnologias:

ShellScript

Windows Server



Anotar



Marcar como concluído

Inicie agora sua carreira de programador por apenas R\$ 54,90/mês

Ainda está em dúvida? Experimente a plataforma durante 3 dias sem cartão. **Faça um teste grátis**

BENEFÍCIOS

Suporte em tempo real

Certificado de autoridade

Exercícios para praticar

Estudo gamificado

Planos de estudo para

Saiba mais



Por **Leandro**
Em 2012

RECEBA NOSSAS NOVIDADES

Informe o seu e-mail

Receber Newsletter

Suporte ao aluno

Minhas dúvidas

[Cursos](#)[Artigos](#)[Revistas](#)[Quem Somos](#)[Fale conosco](#)[Plano para Instituição de ensino](#)[Assinatura para empresas](#)[Assine agora](#)

Hospedagem web por Porta 80 Web Hosting



6

