Powershell - Guia Rápido

Anúncios

Página anterior Próxima página

Powershell - Visão Geral

O Windows PowerShell é um **shell de linha de comando** e uma **linguagem de script** projetada especialmente para a administração do sistema. É análogo no Linux é chamado de Bash Scripting. Baseado no .NET Framework, o Windows PowerShell ajuda os profissionais de TI a controlarem e automatizarem a administração do sistema operacional Windows e dos aplicativos executados no ambiente do Windows Server.

Os comandos do Windows PowerShell, chamados de **cmdlets** , permitem gerenciar os computadores a partir da linha de comando. Os provedores do Windows PowerShell permitem acessar os repositórios de dados, como o Registro e o Armazenamento de certificados, tão facilmente quanto você acessa o sistema de arquivos.

Além disso, o Windows PowerShell tem um analisador de expressões avançado e uma linguagem de script totalmente desenvolvida. Então, em palavras simples, você pode concluir todas as tarefas que você faz com GUI e muito mais.

PowerShell ISE

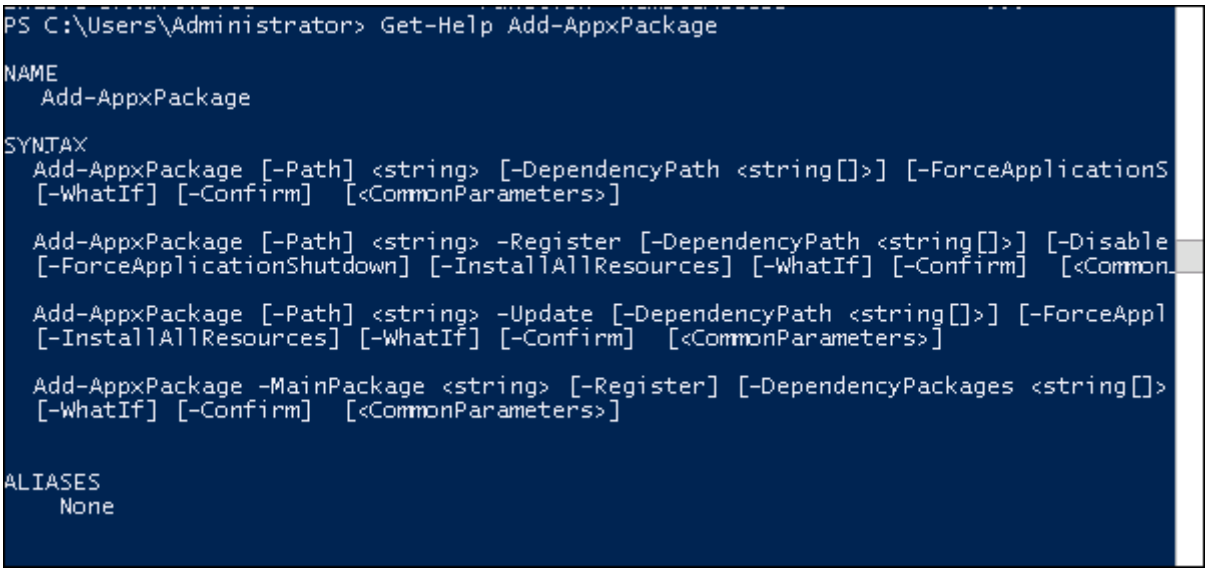
O ISE (**Ambiente de Script Integrado) do** Windows PowerShell é um aplicativo host do Windows PowerShell. No Windows PowerShell ISE, você pode executar comandos e escrever, testar e depurar scripts em uma única interface gráfica do usuário baseada em Windows com edição de múltiplas linhas, preenchimento de guias, coloração de sintaxe, execução seletiva, ajuda sensível ao contexto e suporte Idiomas

Você pode usar itens de menu e atalhos de teclado para executar muitas das mesmas tarefas que executaria no console do Windows PowerShell. Por exemplo, quando você depurar um script no Windows PowerShell ISE, para definir um ponto de interrupção de linha em um script, clique com o botão direito do mouse na linha de código e clique em **Alternar Ponto de Interrupção** .

Comandos básicos do PowerShell

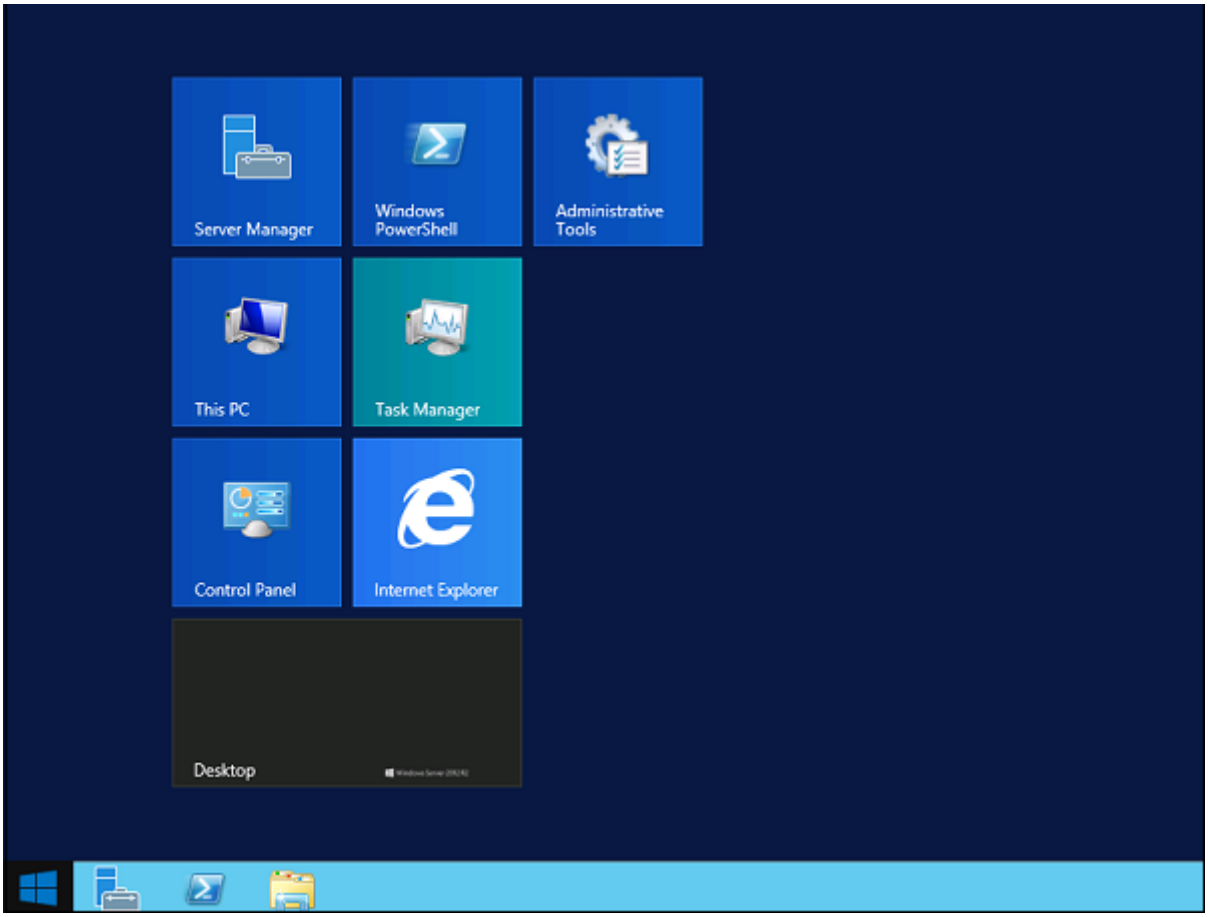
Há muitos comandos do PowerShell e é muito difícil colocar todos esses comandos neste tutorial, vamos nos concentrar em alguns dos comandos mais importantes e básicos do PowerShell.

O primeiro passo é ir ao comando Get-Help, que lhe dá uma explicação sobre como dar um comando e seu parâmetro.

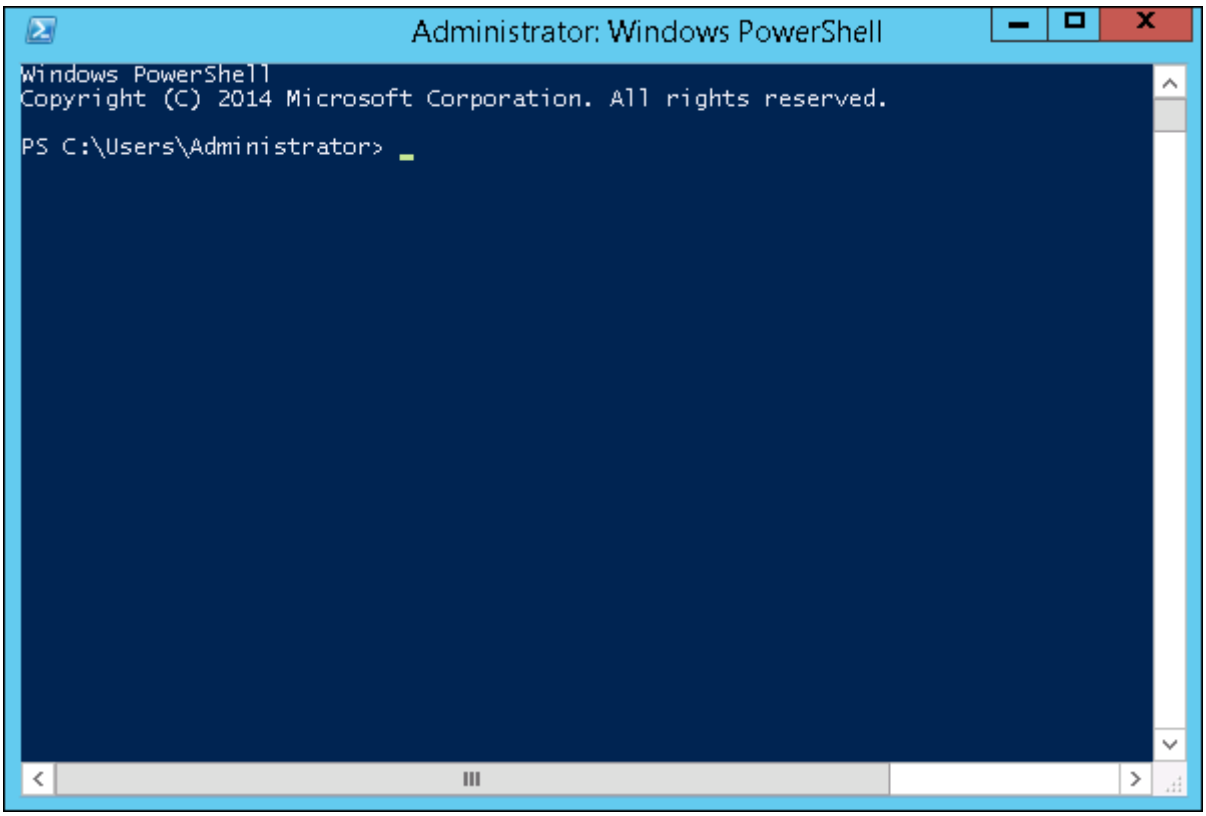


Powershell - Configuração do Ambiente

O ícone do PowerShell pode ser encontrado na barra de tarefas e no menu Iniciar. Apenas clicando no ícone, ele será aberto.

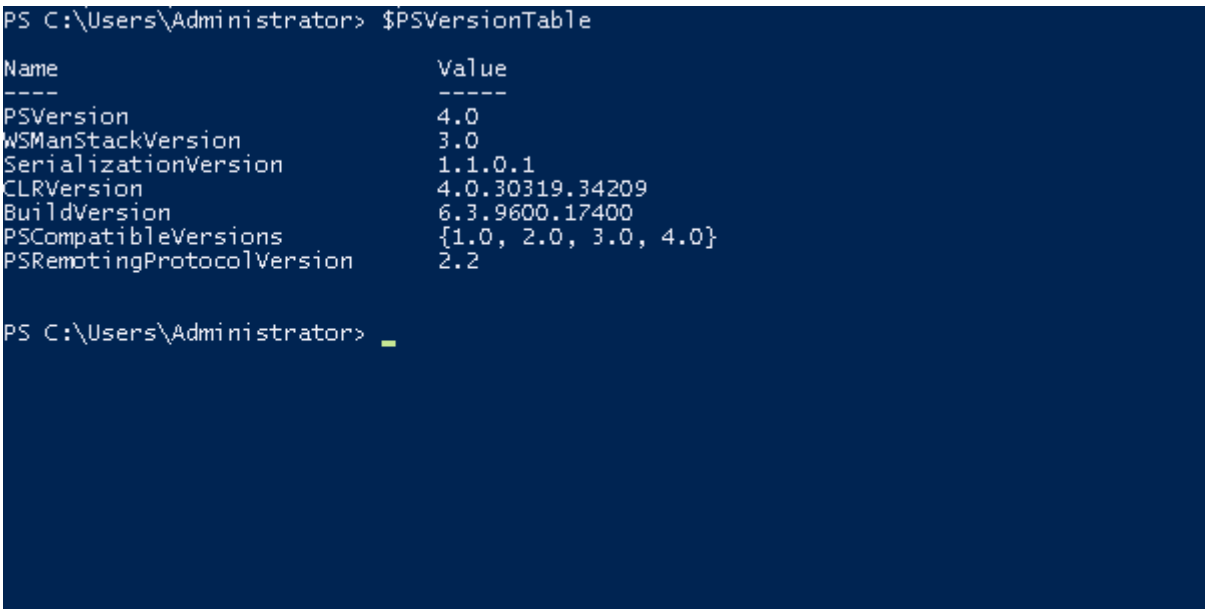


Para abri-lo, basta clicar no ícone e, em seguida, a tela a seguir será aberta e isso significa que o PowerShell está pronto para você trabalhar.

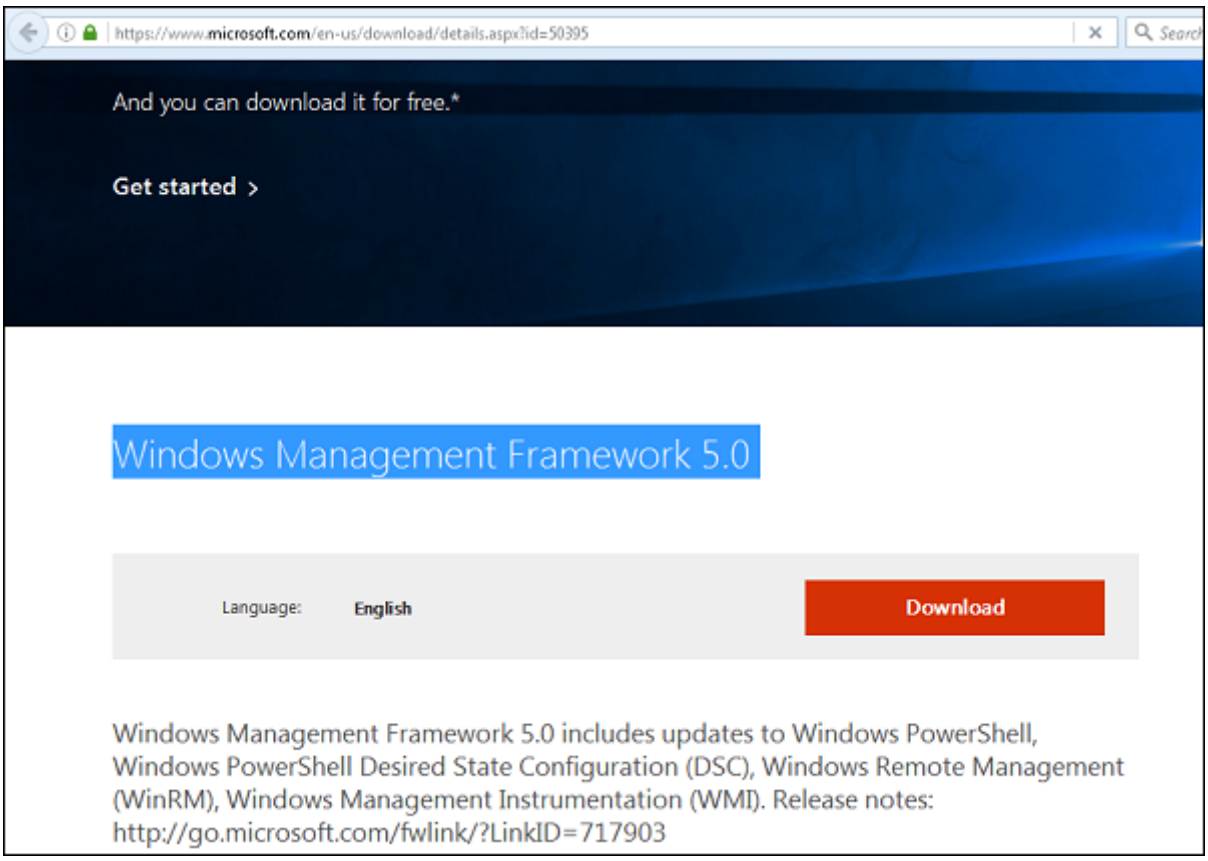


Versão do PowerShell

A versão mais recente do PowerShell é 5.0 e para verificar o que está instalado em nosso servidor, **digitamos** o seguinte comando - **: $ PSVersionTable,** conforme mostrado na captura de tela a seguir e, na tela, também sabemos que temos o PSVersion 4.0



Para atualizar com a versão mais recente, onde há mais cmdlets, baixe o **Windows Management Framework 5.0** no seguinte link - https://www.microsoft.com/pt-br/download/details.aspx?id=50395 e instale-o .

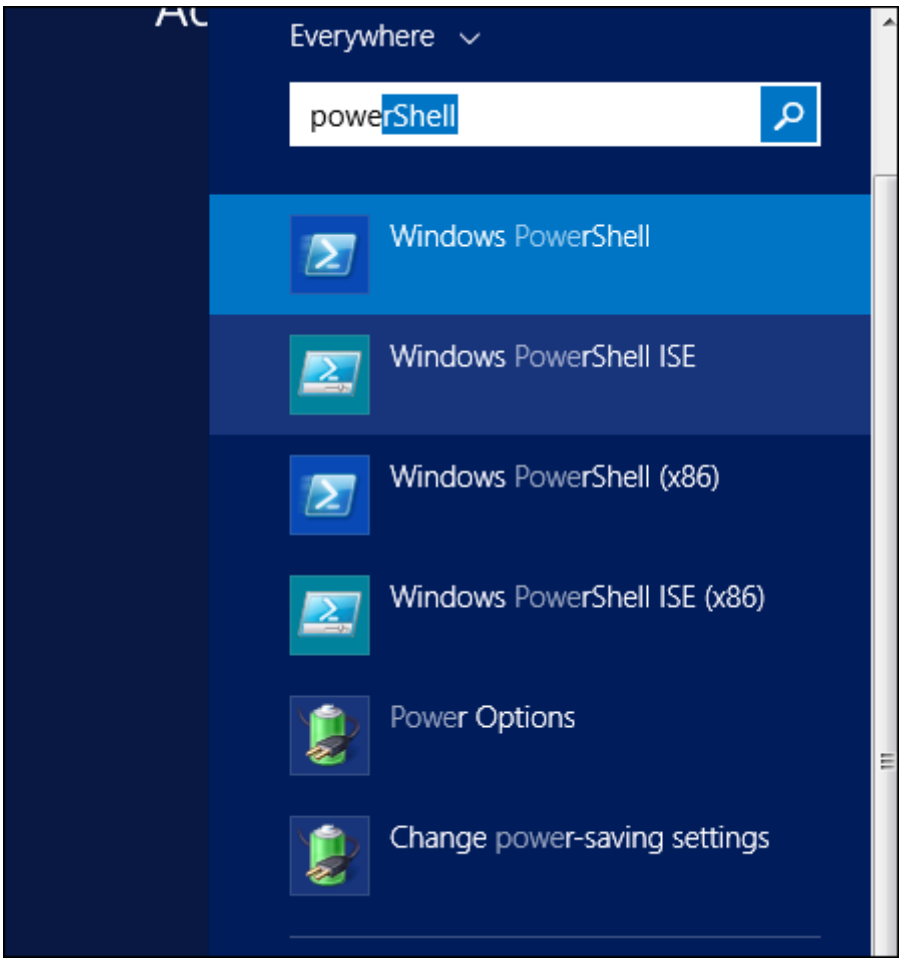


PowerShell ISE

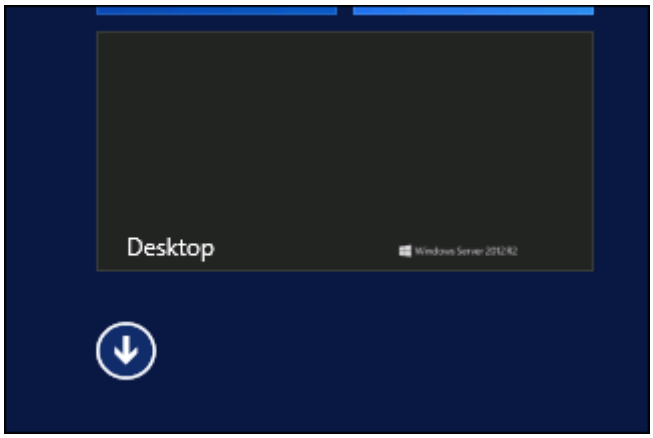
O ISE (**Ambiente de Script Integrado) do** Windows PowerShell é um aplicativo host do Windows PowerShell. No Windows PowerShell ISE, você pode executar comandos e escrever, testar e depurar scripts em uma única interface gráfica do usuário baseada em Windows com edição de múltiplas linhas, preenchimento de guias, coloração de sintaxe, execução seletiva, ajuda sensível ao contexto e suporte Idiomas

Você pode usar itens de menu e atalhos de teclado para executar muitas das mesmas tarefas que executaria no console do Windows PowerShell. Por exemplo, quando você depurar um script no Windows PowerShell ISE, para definir um ponto de interrupção de linha em um script, clique com o botão direito do mouse na linha de código e clique em **Alternar Ponto de Interrupção** .

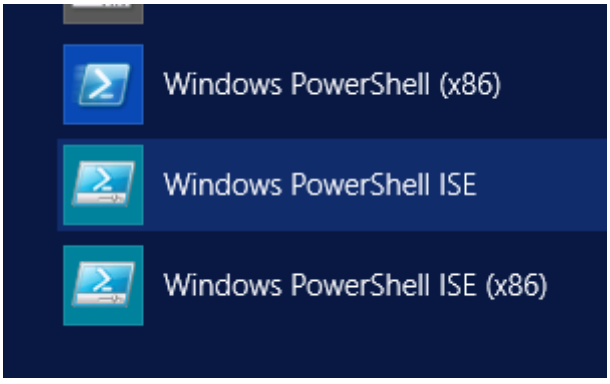
Para abri-lo, basta ir em Iniciar - Pesquisar e, em seguida, Digitar - PowerShell, conforme mostrado na captura de tela a seguir.



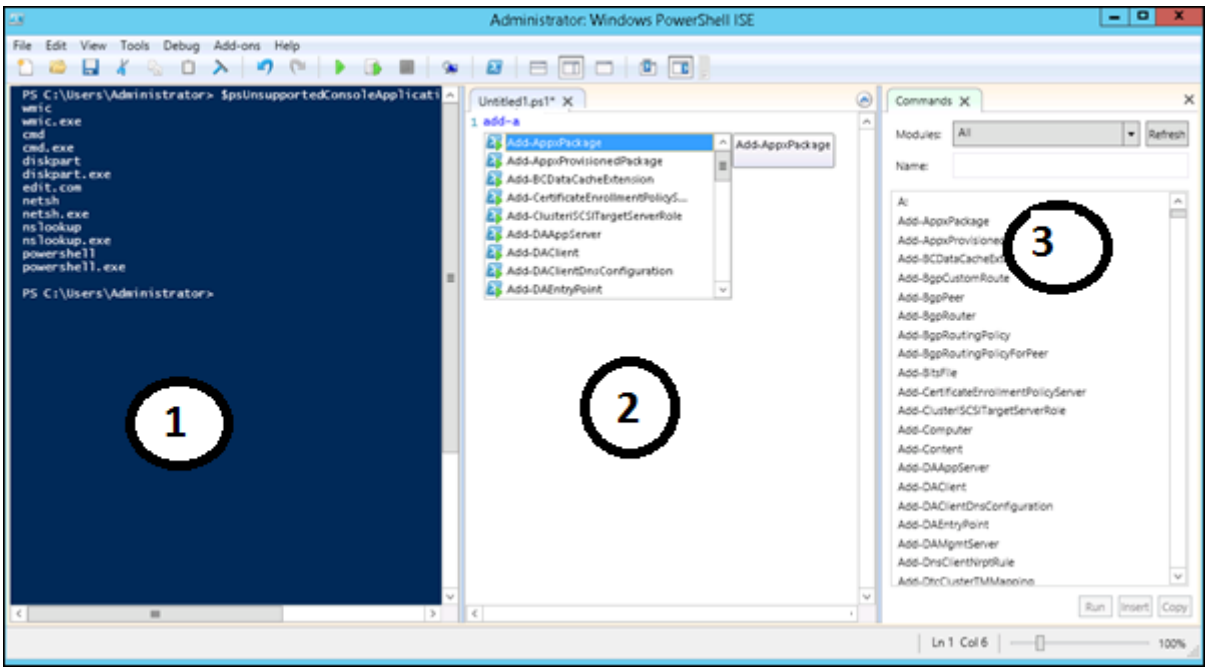
Em seguida, clique no Windows PowerShell ISE. Ou clique na seta para baixo, conforme mostrado na captura de tela a seguir.



Ele listará todos os aplicativos instalados no servidor e, em seguida, clicar no Windows PowerShell ISE.

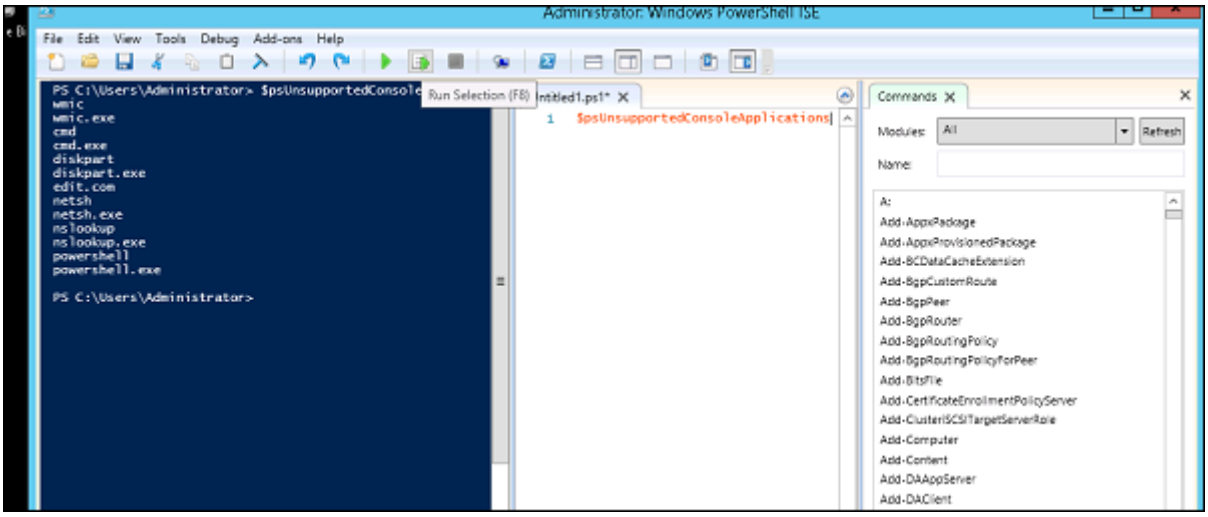


A tabela a seguir será aberta -



Ele tem três seções, que incluem - O **Console do PowerShell** com o número 1, depois o **Arquivo de Script** número 2 e o terceiro é o **Módulo de Comando** onde você pode encontrar o módulo.

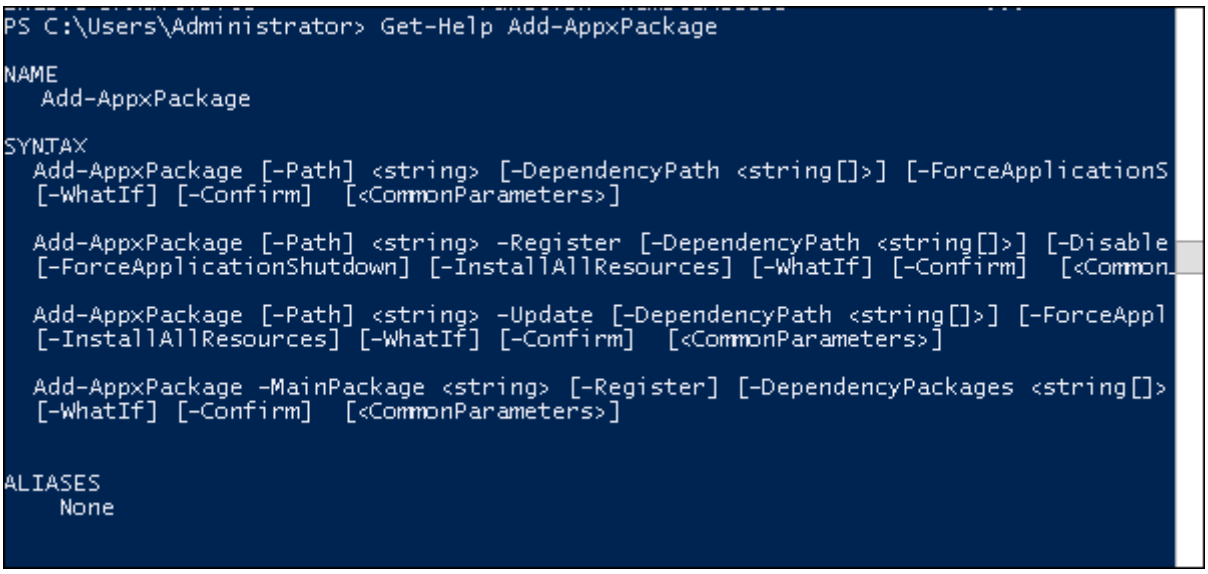
Ao criar o script, você pode executar diretamente e ver o resultado como o exemplo a seguir -



Comandos básicos do PowerShell

Há muitos comandos do PowerShell e é muito difícil colocar todos esses comandos neste tutorial, vamos nos concentrar em alguns dos comandos mais importantes e básicos do PowerShell.

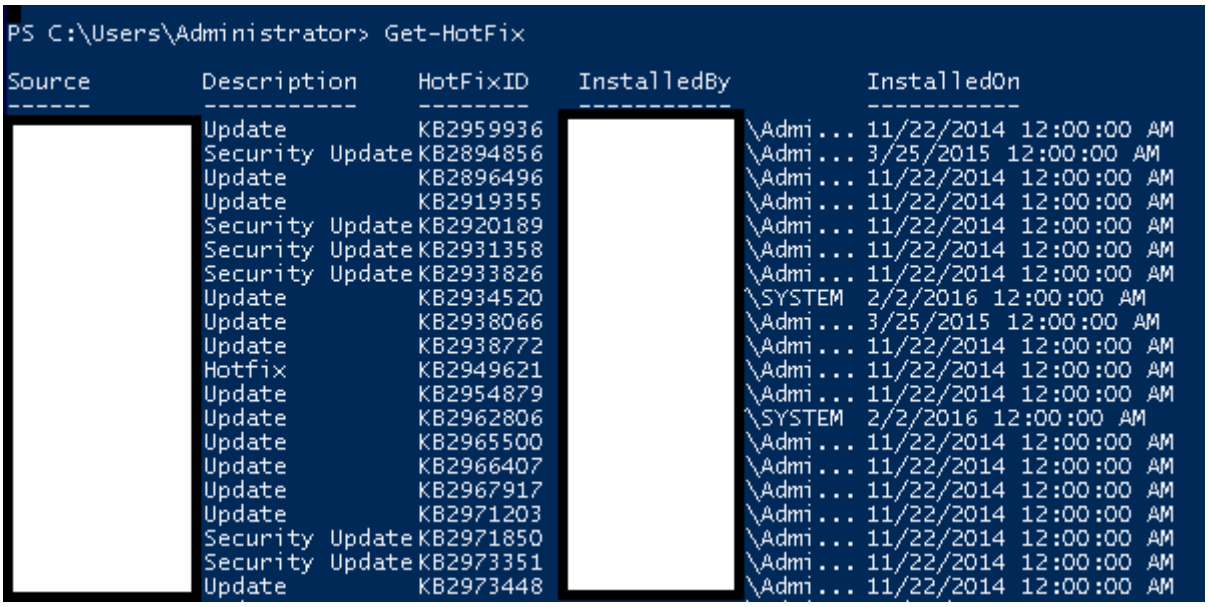
O primeiro passo é ir ao comando Get-Help, que lhe dá uma explicação sobre como dar um comando e seu parâmetro.



Para obter a lista de atualizações -

Get-HotFix e para instalar um hot fix da seguinte forma

Get-HotFix -id kb2741530



Powershell - cmdlets

Um cmdlet ou "Command let" é um comando leve usado no ambiente do Windows PowerShell. O tempo de execução do Windows PowerShell chama esses cmdlets no prompt de comando. Você pode criar e invocá-los programaticamente por meio das APIs do Windows PowerShell.

Cmdlet vs Command

Os cmdlets são muito diferentes dos comandos em outros ambientes de shell de comando das seguintes maneiras:

Cmdlets são objetos de classe do .NET Framework; e não apenas executáveis independentes.

Os cmdlets podem ser facilmente construídos a partir de apenas algumas linhas de código.

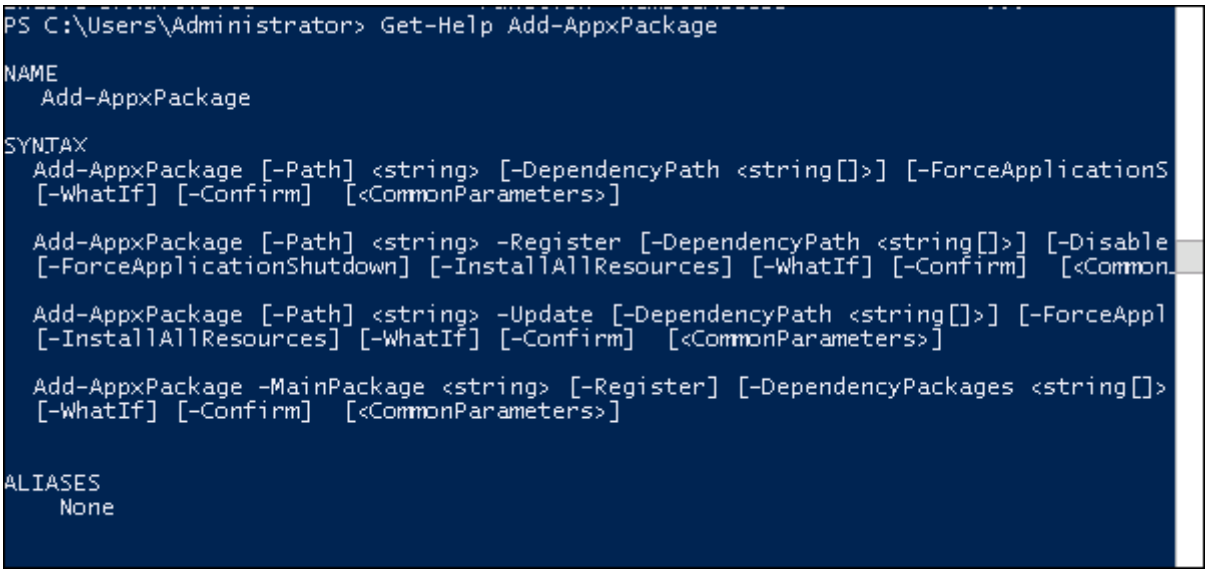
A análise, a apresentação de erros e a formatação de saída não são tratadas pelos cmdlets. Isso é feito pelo tempo de execução do Windows PowerShell.

O processo de cmdlets funciona em objetos que não estão no fluxo de texto e os objetos podem ser transmitidos como saída para o pipelining.

Os cmdlets são baseados em registros à medida que processam um único objeto por vez.

Conseguindo ajuda

O primeiro passo é ir ao comando Get-Help, que lhe dá uma explicação sobre como dar um comando e seu parâmetro.



Powershell - Operações de arquivos e pastas

A seguir estão os exemplos de scripts do PowerShell em Arquivos e Pastas.

Sr. Não.	Operação e Descrição
1	**Criando Pastas** Exemplo de script para mostrar como criar pastas usando scripts do PowerShell.
2	**Criando arquivos** Exemplo de script para mostrar como criar arquivo (s) usando scripts do PowerShell.
3	**Copiando Pastas** Exemplo de script para mostrar como copiar arquivo (s) usando scripts do PowerShell.

4	**Copiando Arquivos** Exemplo de script para mostrar como criar arquivo (s) usando scripts do PowerShell.
5	**Excluindo Pastas** Exemplo de script para mostrar como excluir pasta (s) usando scripts do PowerShell.
6	**Excluindo arquivos** Exemplo de script para mostrar como excluir arquivo (s) usando scripts do PowerShell.
7	**Movendo Pastas** Exemplo de script para mostrar como mover pasta (s) usando scripts do PowerShell.
8	**Movendo Arquivos** Exemplo de script para mostrar como mover arquivo (s) usando scripts do PowerShell.
9	**Renomear pastas** Exemplo de script para mostrar como renomear pasta (s) usando scripts do PowerShell.
10	**Renomear arquivos** Exemplo de script para mostrar como renomear arquivo (s) usando scripts do PowerShell.
11	**Recuperando Item** Exemplo de script para mostrar como recuperar itens usando scripts do PowerShell.
12	**Verifique a existência da pasta** Exemplo de script para mostrar como verificar a existência de pastas usando scripts do PowerShell.
13	**Verificar a existência do arquivo** Exemplo de script para mostrar como verificar a existência de arquivos usando scripts do PowerShell.

Powershell - Operações de data e hora

A seguir estão os exemplos de scripts do PowerShell na Data e Hora do Sistema.

Sr. Não.	Operação e Descrição
1	**Obter data do sistema** Exemplo de script para mostrar como obter a data do sistema usando scripts do PowerShell.
2	**Definir data do sistema** Exemplo de script para mostrar como definir a data do sistema usando scripts do PowerShell.
3	**Obter hora do sistema** Exemplo de script para mostrar como obter a hora do sistema usando scripts do PowerShell.
4	**Definir hora do sistema** Exemplo de script para mostrar como definir a hora do sistema usando scripts do PowerShell.

Powershell - Operações de E / S de Arquivo

A seguir estão os exemplos de scripts do PowerShell de criação e leitura de diferentes tipos de arquivos.

Sr. Não.	Operação e Descrição
1	**Criar arquivo de texto** Exemplo de script para mostrar como criar um arquivo de texto usando scripts do PowerShell.
2	**Leia o arquivo de texto** Exemplo de script para mostrar como ler um arquivo de texto usando scripts do PowerShell.
3	**Crie um arquivo XML** Exemplo de script para mostrar como criar um arquivo XML usando scripts do PowerShell.
4	**Leia o arquivo XML** Exemplo de script para mostrar como ler um arquivo XML usando scripts do PowerShell.
5	**Criar arquivo CSV**

	Exemplo de script para mostrar como criar um arquivo CSV usando scripts do PowerShell.
6	**Leia o arquivo CSV** Exemplo de script para mostrar como ler um arquivo CSV usando scripts do PowerShell.
7	**Crie um arquivo HTML** Exemplo de script para mostrar como criar um arquivo HTML usando scripts do PowerShell.
8	**Leia o arquivo HTML** Exemplo de script para mostrar como ler um arquivo HTML usando scripts do PowerShell.
9	**Apagando o conteúdo do arquivo** Exemplo de script para mostrar como apagar o conteúdo do arquivo usando scripts do PowerShell.
10	**Anexar dados de texto** Exemplo de script para mostrar como anexar texto a um conteúdo de arquivo usando scripts do PowerShell.

Powershell - Cmdlets avançados

Cmdlets

Um cmdlet ou "Command let" é um comando leve usado no ambiente do Windows PowerShell. O tempo de execução do Windows PowerShell chama esses cmdlets no prompt de comando. Você pode criar e invocá-los programaticamente por meio das APIs do Windows PowerShell. A seguir estão exemplos avançados de uso de cmdlets.

Sr. Não.	Tipo e descrição do cmdlet
1	**Cmdlet Get-Unique** Exemplo de programa para mostrar o cmdlet Get-Unique.
2	**Cmdlet Group-Object** Exemplo de programa para mostrar o Cmdlet Group-Object.
3	**Cmdlet Measure-Object** Exemplo de programa para mostrar o cmdlet Measure-Object.

4	**Cmdlet Comparar Objeto** Exemplo de programa para mostrar o Cmdlet Compare-Object.
5	**Cmdlet Format-List** Exemplo de programa para mostrar o cmdlet Format-List.
6	**Cmdlet Format-Wide** Exemplo de programa para mostrar o cmdlet Format-Wide.
7	**Cmdlet Where-Object** Exemplo de programa para mostrar o cmdlet Where-Object.
8	**Cmdlet Get-ChildItem** Exemplo de programa para mostrar o Cmdlet Get-ChildItem.
9	**Cmdlet ForEach-Object** Exemplo de programa para mostrar o cmdlet ForEach-Object.
10	**Cmdlet Start-Sleep** Exemplo de programa para mostrar o cmdlet Start-Sleep.
11	**Cmdlet Read-Host** Exemplo de programa para mostrar o cmdlet Read-Host.
12	**Cmdlet Select-Object** Exemplo de programa para mostrar o cmdlet Select-Object.
13	**Cmdlet Sort-Object** Exemplo de programa para mostrar o Cmdlet Sort-Object.
14	**Cmdlet Write-Warning** Exemplo de programa para mostrar o cmdlet Write-Warning.
15	**Cmdlet Write-Host** Exemplo de programa para mostrar o Cmdlet Write-Host.
16	**Cmdlet Invoke-Item** Exemplo de programa para mostrar o Cmdlet Invoke-Item.
17	**Cmdlet Invoke-Expression** Exemplo de programa para mostrar o Cmdlet Invoke-Expression.
18	**Cmdlet Measure-Command** Exemplo de programa para mostrar o cmdlet Measure-Command.

19	**Cmdlet Invoke-History** Exemplo de programa para mostrar o cmdlet Invoke-History.
20	**Cmdlet Add-History** Exemplo de programa para mostrar o Cmdlet Add-History.
21	**Cmdlet Get-History** Exemplo de programa para mostrar o cmdlet Get-History.
22	**Cmdlet Get-Culture** Exemplo de programa para mostrar o cmdlet Get-Culture.

Powershell - Scripting

O Windows PowerShell é um **shell de linha de comando** e uma **linguagem de script** projetada especialmente para a administração do sistema. Seu análogo no Linux é chamado de Bash Scripting. Baseado no .NET Framework, o Windows PowerShell ajuda os profissionais de TI a controlarem e automatizarem a administração do sistema operacional Windows e dos aplicativos executados no ambiente do Windows Server.

Os comandos do Windows PowerShell, chamados de **cmdlets** , permitem gerenciar os computadores a partir da linha de comando. Os provedores do Windows PowerShell permitem acessar os repositórios de dados, como o Registro e o Armazenamento de certificados, tão facilmente quanto você acessa o sistema de arquivos.

Além disso, o Windows PowerShell tem um analisador de expressões avançado e uma linguagem de script totalmente desenvolvida. Então, em palavras simples, você pode concluir todas as tarefas que você faz com GUI e muito mais. O Windows PowerShell Scripting é uma linguagem de script totalmente desenvolvida e possui um analisador de expressões

Características

Cmdlets - os cmdlets executam tarefas comuns de administração do sistema, por exemplo, gerenciando o registro, serviços, processos, logs de eventos e usando o Windows Management Instrumentation (WMI).

Orientado a tarefas - a linguagem de script do PowerShell é baseada em tarefas e fornece suporte a scripts e ferramentas de linha de comando existentes.

Design consistente - Como os cmdlets e os armazenamentos de dados do sistema usam uma sintaxe comum e possuem convenções de nomenclatura comuns, o compartilhamento de dados é fácil. A saída de um cmdlet pode ser canalizada para outro cmdlet sem qualquer manipulação.

Simples de usar - A navegação simplificada baseada em comandos permite que os usuários naveguem no registro e em outros armazenamentos de dados, de forma semelhante à navegação do sistema de arquivos.

Baseado em objetos - o PowerShell possui recursos poderosos de manipulação de objetos. Objetos podem ser enviados para outras ferramentas ou bancos de dados diretamente.

Interface extensível. - O PowerShell é personalizável como fornecedores independentes de software e os desenvolvedores corporativos podem criar ferramentas e utilitários personalizados usando o PowerShell para administrar seu software.

Variáveis

Variáveis do PowerShell são objetos nomeados. Como o PowerShell trabalha com objetos, essas variáveis são usadas para trabalhar com objetos.

Criando Variável

O nome da variável deve começar com $ e pode conter caracteres alfanuméricos e sublinhados em seus nomes. Uma variável pode ser criada digitando um nome de variável válido.

Digite o seguinte comando no Console do PowerShell ISE. Supondo que você esteja na pasta D: \ test.

```
$location = Get-Location
```

Aqui, criamos uma variável $ location e atribuímos a saída do cmdlet Get-Location. Agora contém a localização atual.

Usando variável

Digite o seguinte comando no Console do PowerShell ISE.

```
 $location
```

Saída

Você pode ver a seguinte saída no console do PowerShell.

```
Path
----
D:\test
```

Obtendo informações de variável

O cmdlet Get-Member pode informar o tipo de variável que está sendo usada. Veja o exemplo abaixo.

```
$location | Get-Member
```

Saída

Você pode ver a seguinte saída no console do PowerShell.

```
   TypeName: System.Management.Automation.PathInfo

Name          MemberType   Definition
----          ----------   ----------
Equals        Method       bool Equals(System.Object obj)
GetHashCode   Method       int GetHashCode()
GetType       Method       type GetType()
ToString      Method       string ToString()
Drive         Property     System.Management.Automation.PSDriveInfo Drive {get;}
Path          Property     System.String Path {get;}
Provider      Property     System.Management.Automation.ProviderInfo Provider {get;}
ProviderPath  Property     System.String ProviderPath {get;}
```

Powershell - Variáveis Especiais

Variáveis especiais do PowerShell armazenam informações sobre o PowerShell. Estas também são chamadas de variáveis automáticas. A seguir, a lista de variáveis automáticas -

Operador	Descrição
$$	Representa o último token na última linha recebida pela sessão.
$?	Representa o status de execução da última operação. Ele contém TRUE se a última operação foi bem-sucedida e FALSE se falhou.
$ ^	Representa o primeiro token na última linha recebida pela sessão.
$ _	O mesmo que $ PSItem. Contém o objeto atual no objeto de pipeline. Você pode usar essa variável em comandos que executam uma ação em cada objeto ou em objetos selecionados em um pipeline.
$ ARGS	Representa uma matriz de parâmetros não declarados e / ou valores de parâmetros que são passados para uma função, script ou bloco de script.
$ CONSOLEFILENAME	Representa o caminho do arquivo de console (.psc1) usado mais recentemente na sessão.

$ ERROR	Representa uma matriz de objetos de erro que representam os erros mais recentes.
$ EVENT	Representa um objeto PSEventArgs que representa o evento que está sendo processado.
$ EVENTARGS	Representa um objeto que representa o primeiro argumento de evento que deriva de EventArgs do evento que está sendo processado.
$ EVENTSUBSCRIBER	Representa um objeto PSEventSubscriber que representa o assinante do evento que está sendo processado.
$ EXECUTIONCONTEXT	Representa um objeto EngineIntrinsics que representa o contexto de execução do host do PowerShell.
$ FALSE	Representa FALSE. Você pode usar essa variável para representar FALSE em comandos e scripts, em vez de usar a string "false".
$ FOREACH	Representa o enumerador (não os valores resultantes) de um loop ForEach. Você pode usar as propriedades e métodos de enumeradores no valor da variável $ ForEach.
$ HOME	Representa o caminho completo do diretório inicial do usuário.
$ HOST	Representa um objeto que representa o aplicativo host atual do PowerShell.
$ INPUT	Representa um enumerador que enumera todas as entradas que são passadas para uma função.
$ LASTEXITCODE	Representa o código de saída do último programa baseado no Windows que foi executado.
$ MATCHES	A variável $ Matches trabalha com os operadores -match e -notmatch.
$ MYINVOCATION	$ MyInvocation é preenchido apenas para scripts, funções e blocos de script. As propriedades PSScriptRoot e PSCommandPath da variável automática $ MyInvocation contêm informações sobre o invocador ou script de chamada, não o script atual.
$ NESTEDPROMPTLEVEL	Representa o nível de prompt atual.
$ NULL	$ null é uma variável automática que contém um valor vazio ou NULL. Você pode usar essa variável para representar um valor ausente ou indefinido em comandos e scripts.
$ PID	Representa o identificador de processo (PID) do processo que hospeda a sessão atual do PowerShell.
$ PROFILE	Representa o caminho completo do perfil do PowerShell para o

	usuário atual e o aplicativo host atual.
$ PSCMDLET	Representa um objeto que representa o cmdlet ou a função avançada que está sendo executada.
$ PSCOMMANDPATH	Representa o caminho completo e o nome do arquivo do script que está sendo executado.
$ PSCULTURE	Representa o nome da cultura atualmente em uso no sistema operacional.
$ PSDEBUGCONTEXT	Durante a depuração, essa variável contém informações sobre o ambiente de depuração. Caso contrário, contém um valor NULL.
$ PSHOME	Representa o caminho completo do diretório de instalação do PowerShell.
$ PSITEM	O mesmo que $ _. Contém o objeto atual no objeto de pipeline.
$ PSSCRIPTROOT	Representa o diretório a partir do qual um script está sendo executado.
$ PSSENDERINFO	Representa informações sobre o usuário que iniciou a PSSession, incluindo a identidade do usuário e o fuso horário do computador de origem.
$ PSUICULTURA	Representa o nome da cultura da interface do usuário (IU) que está atualmente em uso no sistema operacional.
$ PSVERSIONTABLE	Representa uma tabela de hash somente leitura que exibe detalhes sobre a versão do PowerShell em execução na sessão atual.
$ SENDER	Representa o objeto que gerou este evento.
$ SHELLID	Representa o identificador do shell atual.
$ STACKTRACE	Representa um rastreamento de pilha para o erro mais recente.
$ THIS	Em um bloco de script que define uma propriedade de script ou método de script, a variável $ This refere-se ao objeto que está sendo estendido.
$ TRUE	Representa TRUE. Você pode usar essa variável para representar TRUE em comandos e scripts.

Powershell - Operadores

O PowerShell fornece um conjunto avançado de operadores para manipular variáveis. Podemos dividir todos os operadores do PowerShell nos seguintes grupos -

Operadores aritméticos

Operadores de atribuição

Operadores de Comparação

Operadores lógicos

Operadores Redirecionais

Operadores de spilled e join

Operadores de tipo

Operadores unários

Os operadores aritméticos

Operadores aritméticos são usados em expressões matemáticas da mesma maneira que são usados em álgebra. A tabela a seguir lista os operadores aritméticos -

Suponha que a variável inteira A detenha 10 e a variável B detenha 20, então -

Mostrar exemplos

Operador	Descrição	Exemplo
+ (Adição)	Adiciona valores em ambos os lados do operador.	A + B dará 30
- (subtração)	Subtrai o operando direito do operando esquerdo.	A - B vai dar -10
* (Multiplicação)	Multiplica valores em ambos os lados do operador.	A * B dará 200
/ (Divisão)	Divide o operando esquerdo pelo operando direito.	B / A vai dar 2
% (Módulo)	Divide o operando esquerdo pelo operando direito e retorna o restante.	B% A dará 0

Os operadores de comparação

A seguir, os operadores de atribuição suportados pelo idioma do PowerShell -

Suponha que a variável inteira A detenha 10 e a variável B detenha 20, então -

Mostrar exemplos

Operador	Descrição	Exemplo
eq (igual)	Compara dois valores para serem iguais ou não.	A -eq B dará falso
ne (não é igual)	Compara dois valores para não serem iguais.	A -ne B dará a verdade

gt (maior que)	Compara o primeiro valor a ser maior que o segundo.	B -gt A dará a verdade
ge (maior que ou igual a)	Compara o primeiro valor a ser maior ou igual ao segundo.	B -ge A dará a verdade
lt (menos de)	Compara o primeiro valor a ser menor que o segundo.	B -lt A dará falso
le (menor que ou igual a)	Compara o primeiro valor a ser menor ou igual ao segundo.	B -le A dará a verdade

Os operadores de atribuição

A seguir, os operadores de atribuição suportados pelo idioma do PowerShell -

Mostrar exemplos

Operador	Descrição	Exemplo
=	Operador de atribuição simples. Atribui valores dos operandos do lado direito ao operando do lado esquerdo.	C = A + B atribuirá valor de A + B a C
+ =	Adicionar operador de atribuição E. Adiciona o operando direito ao operando esquerdo e atribui o resultado ao operando esquerdo.	C + = A é equivalente a C = C + A
- =	Subtrair e operador de atribuição. Subtrai o operando direito do operando esquerdo e atribui o resultado ao operando esquerdo.	C - = A é equivalente a C = C - A

Os operadores lógicos

A tabela a seguir lista os operadores lógicos -

Suponha que as variáveis booleanas A sejam verdadeiras e as variáveis B sejam falsas, então -

Mostrar exemplos

Operador	Descrição	Exemplo
E (lógico e)	Chamado Lógico E operador. Se ambos os operandos forem diferentes de zero, a condição se tornará verdadeira.	(A -AND B) é falso
OU (lógico ou)	Chamado Lógico OU Operador. Se algum dos dois operandos for diferente de zero, a condição se tornará verdadeira.	(A -OR B) é verdadeiro

NÃO (não lógico)	Operador NÃO Lógico Chamado. Use para reverter o estado lógico de seu operando. Se uma condição for verdadeira, o operador Lógico NOT tornará falso.	-NOT (A -AND B) é verdadeiro

Operadores Diversos

A seguir, vários operadores importantes suportados pela linguagem PowerShell -

Mostrar exemplos

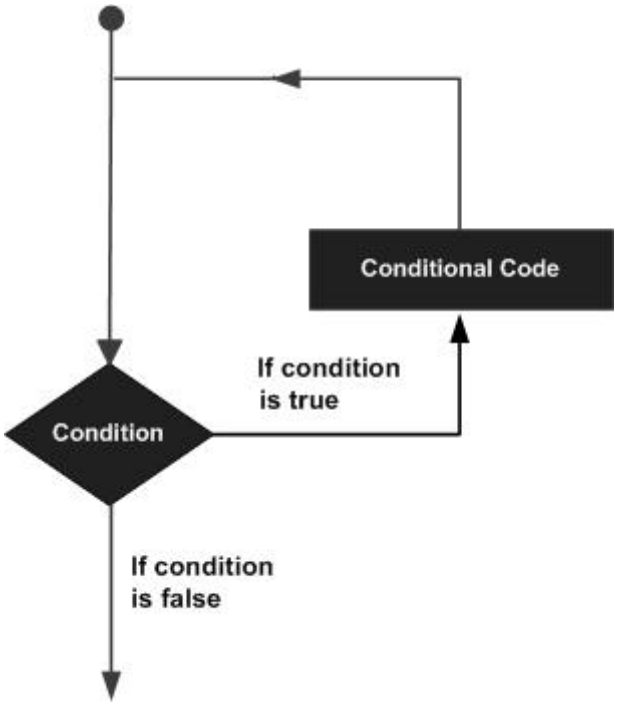
Operador	Descrição	Exemplo
> (Opeator Redirecionado)	Operador Redirecionado. Atribui a saída a ser impressa no dispositivo de arquivo / saída redirecionado.	dir> test.log imprimirá a listagem de diretórios no arquivo test.log

Powershell - looping

Pode haver uma situação em que você precise executar um bloco de código várias vezes. Em geral, as instruções são executadas seqüencialmente: a primeira instrução em uma função é executada primeiro, seguida pela segunda e assim por diante.

As linguagens de programação fornecem várias estruturas de controle que permitem caminhos de execução mais complicados.

Uma instrução de **loop** nos permite executar uma instrução ou grupo de instruções várias vezes e seguir é a forma geral de uma instrução de loop na maioria das linguagens de programação -



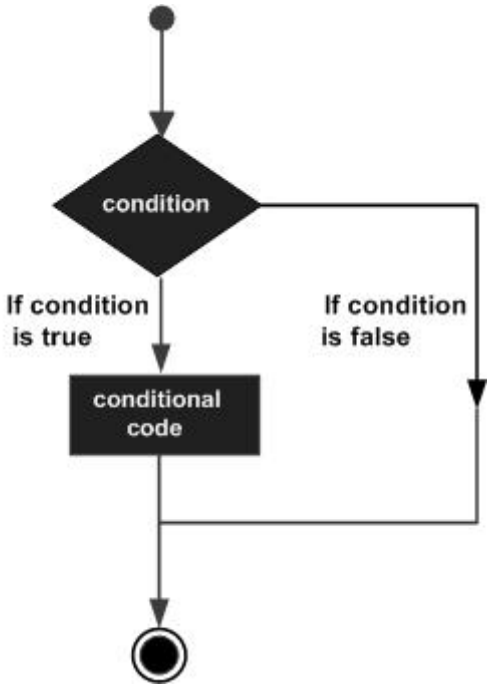
A linguagem de programação do PowerShell fornece os seguintes tipos de loop para lidar com os requisitos de loop. Clique nos links a seguir para verificar seus detalhes.

Sr. Não.	Loop e Descrição
1	**para loop** Execute uma sequência de instruções várias vezes e abrevie o código que gerencia a variável de loop.
2	**loop forEach** Aprimorado por loop. Isso é usado principalmente para percorrer a coleção de elementos, incluindo matrizes.
3	**enquanto loop** Repete uma instrução ou grupo de instruções enquanto uma determinada condição é verdadeira. Ele testa a condição antes de executar o corpo do loop.
4	**fazer ... while loop** Como uma declaração while, exceto que testa a condição no final do corpo do loop.

Powershell - Condições

Estruturas de tomada de decisão têm uma ou mais condições para serem avaliadas ou testadas pelo programa, juntamente com uma declaração ou instruções que devem ser executadas se a condição for determinada como verdadeira e, opcionalmente, outras instruções a serem executadas se a condição for determinada ser falso.

A seguir está a forma geral de uma estrutura de tomada de decisão típica encontrada na maioria das linguagens de programação -



A linguagem de script do PowerShell fornece os seguintes tipos de declarações de tomada de decisão. Clique nos links a seguir para verificar seus detalhes.

Sr. Não.	Declaração & Descrição
1	**se declaração** Uma **instrução if** consiste em uma expressão booleana seguida de uma ou mais instruções.
2	**if ... else statement** Uma **instrução if** pode ser seguida por uma **instrução else** opcional , que é executada quando a expressão booleana é falsa.
3	**aninhado se declaração** Você pode usar um **caso** ou **elseif** declaração dentro de outro **se** ou **elseif** declaração (s).
4	**mudar a indicação** Uma instrução **switch** permite que uma variável seja testada quanto à igualdade em relação a uma lista de valores.

Powershell - matriz

O PowerShell fornece uma estrutura de dados, a **matriz** , que armazena uma coleção seqüencial de elementos de tamanho fixo de qualquer tipo. Uma matriz é usada para armazenar uma coleção de dados, mas geralmente é mais útil pensar em uma matriz como uma coleção de variáveis ou objetos.

Em vez de declarar variáveis individuais, como number0, number1, ... e number99, você declara uma variável de matriz, como números, e usa números [0], números [1] e ..., números [99] para representar variáveis individuais.

Este tutorial apresenta como declarar variáveis de matriz, criar matrizes e matrizes de processo usando variáveis indexadas.

Declarando Variáveis de Matriz

Para usar um array em um programa, você deve declarar uma variável para referenciar o array, e você pode especificar o tipo de array que a variável pode referenciar. Aqui está a sintaxe para declarar uma variável de array -

Sintaxe

```
$A = 1, 2, 3, 4
or
$A = 1..4
```

Nota - Por padrão, o tipo de objetos da matriz é System.Object. O método GetType () retorna o tipo da matriz. Tipo pode ser passado.

Exemplo

Os trechos de código a seguir são exemplos dessa sintaxe -

```
[int32[]]$intA = 1500,2230,3350,4000

$A = 1, 2, 3, 4
$A.getType()
```

Isso produzirá o seguinte resultado -

Saída

```
IsPublic    IsSerial    Name                      BaseType
--------    --------    ----                      --------
True        True        Object[]                  System.Array
```

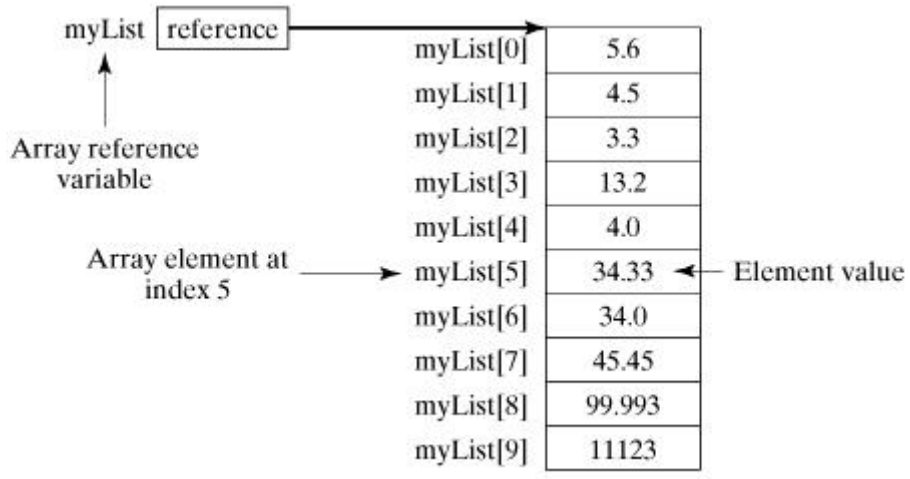
Os elementos da matriz são acessados pelo **índice** . Os índices de matriz são baseados em 0; isto é, eles começam de 0 a **arrayRefVar.length-1** .

Exemplo

A seguinte declaração declara que uma variável de matriz, myList, cria uma matriz de 10 elementos do tipo double e atribui sua referência a myList -

```
$myList = 5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123
```

A figura a seguir representa o array myList. Aqui, myList contém dez valores duplos e os índices são de 0 a 9.



Processando Matrizes

Ao processar elementos de matriz, geralmente usamos loop **de** loop ou **foreach** porque todos os elementos em uma matriz são do mesmo tipo e o tamanho da matriz é conhecido.

Exemplo

Aqui está um exemplo completo mostrando como criar, inicializar e processar matrizes -

```
$myList = 5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123

write-host("Print all the array elements")
$myList

write-host("Get the length of array")
$myList.Length

write-host("Get Second element of array")
$myList[1]

write-host("Get partial array")
$subList = $myList[1..3]

write-host("print subList")
$subList

write-host("using for loop")
for ($i = 0; $i -le ($myList.length - 1); $i += 1) {
   $myList[$i]
}

write-host("using forEach Loop")
foreach ($element in $myList) {
   $element
}

write-host("using while Loop")
$i = 0
while($i -lt 4) {
   $myList[$i];
   $i++
}

write-host("Assign values")
$myList[1] = 10
$myList
```

Isso produzirá o seguinte resultado -

Saída

```
Print all the array elements
5.6
4.5
3.3
13.2
4
34.33
34
45.45
99.993
11123
Get the length of array
10
Get Second element of array
```

```
4.5
Get partial array
print subList
4.5
3.3
13.2
using for loop
5.6
4.5
3.3
13.2
4
34.33
34
45.45
99.993
11123
using forEach Loop
5.6
4.5
3.3
13.2
4
34.33
34
45.45
99.993
11123
using while Loop
5.6
4.5
3.3
13.2
Assign values
5.6
10
3.3
13.2
4
34.33
34
45.45
99.993
11123
```

Os exemplos de métodos de arrays

Aqui está um exemplo completo mostrando operações em arrays usando seus métodos

```
$myList = @(0..4)

write-host("Print array")
$myList

$myList = @(0..4)

write-host("Assign values")
$myList[1]  = 10
$myList
```

Isso produzirá o seguinte resultado -

Saída

```
Clear array
Print array
0
1
2
3
4
Assign values
0
10
2
3
4
```

Powershell - Hashtables

Hashtable armazena pares chave / valor em uma tabela de hash. Ao usar uma Hashtable, você especifica um objeto que é usado como uma chave e o valor que você deseja vincular a essa chave. Geralmente usamos String ou números como chaves.

Este tutorial apresenta como declarar variáveis hashtable, criar hashtabets e processar hashtable usando seus métodos.

Declarando variáveis hashtable

Para usar uma hashtable em um programa, você deve declarar uma variável para fazer referência à hashtable. Aqui está a sintaxe para declarar uma variável hashtable -

Sintaxe

```
$hash = @{ ID = 1; Shape = "Square"; Color = "Blue"}
or
$hash = @{}
```

Nota - Os dicionários ordenados podem ser criados usando uma sintaxe semelhante. Os dicionários ordenados mantêm a ordem na qual as entradas são adicionadas,

enquanto as hashtabs não.

Exemplo

Os trechos de código a seguir são exemplos dessa sintaxe -

```
$hash = [ordered]@{ ID = 1; Shape = "Square"; Color = "Blue"}
```

Imprima o hashtable.

```
$hash
```

Saída

```
Name                           Value
----                           -----
ID                             1
Color                          Blue
Shape                          Square
```

Os valores de hashtable são acessados por meio das **chaves** .

```
> $hash["ID"]
 1
```

Hashtable de processamento

A notação de ponto pode ser usada para acessar chaves ou valores de hashtables.

```
> $hash.keys
ID
Color
Shape

> $hash.values
1
Blue
Square
```

Exemplo

Aqui está um exemplo completo mostrando como criar, inicializar e processar hashtable -

```
$hash = @{ ID = 1; Shape = "Square"; Color = "Blue"}

write-host("Print all hashtable keys")
$hash.keys

write-host("Print all hashtable values")
$hash.values

write-host("Get ID")
$hash["ID"]

write-host("Get Shape")
```

```
$hash.Number

write-host("print Size")
$hash.Count

write-host("Add key-value")
$hash["Updated"] = "Now"

write-host("Add key-value")
$hash.Add("Created","Now")

write-host("print Size")
$hash.Count

write-host("Remove key-value")
$hash.Remove("Updated")

write-host("print Size")
$hash.Count

write-host("sort by key")
$hash.GetEnumerator() | Sort-Object -Property key
```

Isso produzirá o seguinte resultado -

Saída

```
Print all hashtable keys
ID
Color
Shape
Print all hashtable values
1
Blue
Square
Get ID
1
Get Shape
print Size
3
Add key-value
Add key-value
print Size
5
Remove key-value
print Size
4
sort by key

Name                           Value
----                           -----
Color                          Blue
Created                        Now
ID                             1
```

```
Shape
Square
```

Powershell - Expressão Regular

Uma expressão regular é uma seqüência especial de caracteres que ajuda você a encontrar ou encontrar outras strings ou conjuntos de strings, usando uma sintaxe especializada mantida em um padrão. Eles podem ser usados para pesquisar, editar ou manipular texto e dados.

Aqui está a tabela listando toda a sintaxe do metacaractere de expressão regular disponível no PowerShell -

Subexpressão	Fósforos
^	Corresponde ao início da linha.
$	Corresponde ao final da linha.
.	Corresponde a qualquer caractere único, exceto a nova linha. Usando **m** opção permite coincidir com a nova linha também.
[...]	Corresponde a qualquer caractere único entre colchetes.
[^ ...]	Corresponde a qualquer caractere único que não esteja entre colchetes.
\UMA	Começo da cadeia inteira.
\ z	Fim da string inteira.
\ Z	Fim da seqüência inteira, exceto o terminador de linha final permitido.
ré*	Corresponde a 0 ou mais ocorrências da expressão anterior.
re +	Corresponde a 1 ou mais da coisa anterior.
ré?	Corresponde a 0 ou 1 ocorrência da expressão anterior.
re {n}	Corresponde exatamente ao número de ocorrências da expressão anterior.
re {n}	Corresponde a n ou mais ocorrências da expressão anterior.
re {n, m}	Corresponde pelo menos n e no máximo m ocorrências da expressão anterior.
a | b	Corresponde a ou b.
(ré)	Agrupa expressões regulares e lembra o texto correspondente.
(?: re)	Agrupa expressões regulares sem lembrar do texto correspondente.

(?> re)	Corresponde ao padrão independente sem retroceder.
\W	Corresponde aos caracteres da palavra.
\W	Corresponde aos caracteres nonword.
\ s	Corresponde ao espaço em branco. Equivalente a [\ t \ n \ r \ f].
\ S	Corresponde ao não-branco.
\ d	Corresponde aos dígitos. Equivalente a [0-9].
\ D	Corresponde aos não-dígitos.
\UMA	Corresponde ao início da string.
\ Z	Corresponde ao final da string. Se houver uma nova linha, ela corresponderá logo antes da nova linha.
\ z	Corresponde ao final da string.
\ G	Corresponde ao ponto em que a última partida terminou.
\ n	Voltar referência para capturar o número do grupo "n".
\ b	Corresponde os limites da palavra quando fora dos colchetes. Corresponde ao backspace (0x08) quando dentro dos colchetes.
\ B	Corresponde aos limites nonword.
\ n, \ t, etc.	Corresponde a novas linhas, retornos de carro, tabulações, etc.
\ Q	Escape (quote) todos os caracteres até \ E.
\ E	Termina a citação iniciada com \ Q.

Aqui está um exemplo completo mostrando como usar o regex no PowerShell;

Sr. Não.	Correspondência e Descrição
1	**Caracteres do jogo** Exemplo de caracteres de expressão regular suportados.
2	**Classes de Caracteres de Jogo** Exemplo de classes de caracteres suportadas.
3	**Quantificadores de correspondência** Exemplo de quantificadores suportados.

Powershell - Backtick

O operador Backtick (`) também é chamado de operador word-wrap. Permite que um comando seja escrito em várias linhas. Pode ser usado para nova linha (`n) ou tab (` t) em sentenças também. Veja os exemplos abaixo -

Exemplo 1

```
Get-Service * | Sort-Object ServiceType `
| Format-Table Name, ServiceType, Status -AutoSize
```

Se tornará

```
Get-Service * | Sort-Object ServiceType | Format-Table Name, ServiceType, Status -AutoSize
```

Verifique a saída como

```
Name                                           ServiceType  Status
----                                           -----------  ------
MSSQLServerADHelper100                       Win32OwnProcess Stopped
ntrtscan                                     Win32OwnProcess Running
...
```

Exemplo 2

Uso de nova linha e aba.

```
> Write-host "Title Subtitle"
Title Subtitle

> Write-host "Title `nSubtitle"
Title
Subtitle

> Write-host "Title `tSubtitle"
Title   Subtitle
```

Powershell - suportes

Powershell suporta três tipos de colchetes.

Parêntese suportes. - ()

Suportes de aparelho. - {}

Colchetes. - []

Parêntese suportes

Este tipo de parênteses é usado para

passar argumentos

coloque vários conjuntos de instruções

resolver ambiguidade

criar matriz

Exemplo

```
> $array = @("item1", "item2", "item3")

> foreach ($element in $array) { $element }
item1
item2
item3
```

Suportes de aparelho

Este tipo de parênteses é usado para

inclua declarações

comandos de bloco

Exemplo

```
$x = 10

if($x -le 20){
   write-host("This is if statement")
}
```

Isso produzirá o seguinte resultado -

Saída

```
This is if statement.
```

Colchetes

Este tipo de parênteses é usado para

acesso ao array

acesso a hashtables

filtrar usando expressão regular

Exemplo

```
> $array = @("item1", "item2", "item3")

> for($i = 0; $i -lt $array.length; $i++){ $array[$i] }
item1
item2
item3

>Get-Process [r-s]*
 Handles    NPM(K)     PM(K)    WS(K)   VM(M)   CPU(s)     Id    ProcessName
-------    ------     -----    -----   -----   ------     --    -----------
```

```
   320        72     27300     33764    227     3.95    4028    SCNotification
  2298        77     57792     48712    308             2884    SearchIndexer
  ...
```

Powershell - Alias

O alias do PowerShell é outro nome para o cmdlet ou para qualquer elemento de comando.

Criando alias

Use o cmdlet **New-Alias** para criar um alias. No exemplo abaixo, criamos uma ajuda de alias para o cmdlet Get-Help.

```
New-Alias -Name help -Value Get-Help
```

Agora invoque o alias.

```
help Get-WmiObject -Detailed
```

Você verá a seguinte saída.

```
NAME
   Get-WmiObject

SYNOPSIS
   Gets instances of Windows Management Instrumentation (WMI) classes or information about the availab

SYNTAX
   Get-WmiObject [
...
```

Obtendo o alias

Use o cmdlet **get-alias** para obter todo o alias presente na sessão atual do PowerShell.

```
Get-Alias
```

Você verá a seguinte saída.

```
CommandType     Name                    Definition
-----------     ----                    ----------
Alias           %                       ForEach-Object
Alias           ?                       Where-Object
Alias           ac                      Add-Content
Alias           asnp                    Add-PSSnapIn
...
```

© Copyright 2018. Todos os direitos reservados.

Enter email for newsletter vai