INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA RIO GRANDE DO NORTE

Professor: Macêdo Firmino
Disciplina: Administração de Sistemas Proprietários
Aula 14: Introdução ao PowerShell.

Olá, meu queridos!! Tudo bem??? Na aula de hoje iremos conhecer os fundamentos do PowerShell, incluindo sua interface, comandos básicos e aplicações práticas para administração de sistemas operacionais Windows. Vamos lá!!! Preparados???

PowerShell

O PowerShell é um shell desenvolvida pela Microsoft. O Shell é uma interface entre o usuário e o sistema operacional que permite executar comandos e interagir com os recursos do sistema. O PowerShell é uma ferramenta que, além das funções de um shell, visa permitir automação de tarefas e gerenciamento de sistemas através de comandos e scripts em ambientes Windows. Ele possui uma linguagem de script baseada no .NET.

O PowerShell foi lançado em 2006, inicialmente desenvolvido para suprir as limitações do Prompt de Comando (CMD). Enquanto o CMD trabalha apenas com texto, o PowerShell usa objetos .NET, facilitando o processamento e a manipulação de dados complexos. Além disso, o PowerShell interage nativamente com componentes do Windows, como Active Directory e serviços na nuvem, algo inviável no CMD.

Os comandos do PowerShell (chamados de cmdlet) permitem que você gerencie os computadores da linha de comando. Ele ainda permitem o acesso a vários recursos e configurações do Windows, por exemplo, ao repositórios de dados, como o registro e o repositório de certificados, de forma tão fácil quanto acessar o sistema de arquivos.

O PowerShell é um *software livre* e código-fonte aberto para contribuições da comunidade.

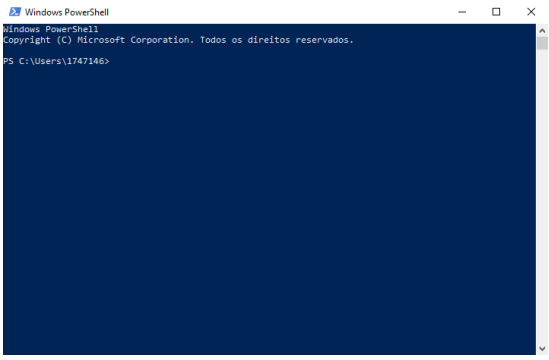
Consoles PowerShell

Para trabalharmos com o Powershell podemos utilizar o Shell diretamente (PS) ou uma interface gráfica o Powershell ISE.

Console PowerShell PS

O PS é a interface de linha de comando tradicional do PowerShell, acessada pelo console padrão no Windows. Ele é ideal para comandos rápidos e scripts curtos. Para abrir, siga os passos:

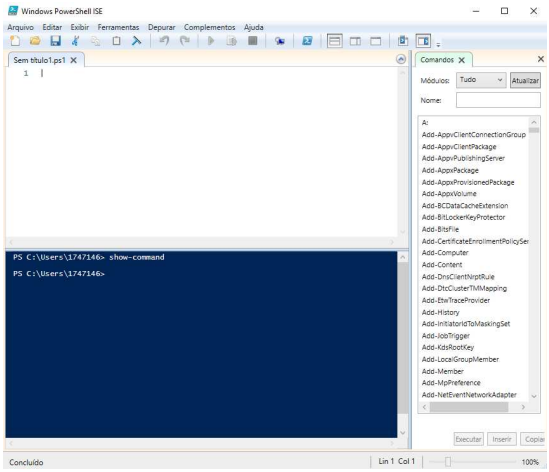
1. Clique em “Iniciar”, digite “PowerShell” e clique em “Windows PowerShell”.



Console PowerShell ISE

O PowerShell ISE (*Integrated Scripting Environment*) é um ambiente de programação do PowerShell que facilita o desenvolvimento de *scripts*. Nele você pode executar comandos, gravar, testar e depurar *scripts* em uma interface de usuário gráfica baseada no Windows. Para iniciar PS ISE:

2. Clique em “Iniciar”, digite “ISE” e clique em “ISE do Windows PowerShell”.



O console Powershell ISE se divide em 3 painéis

- Script: onde você pode editar seus scripts.
- Console: exibe a saída do seus comandos e onde podemos executar comandos diretamente.
- Commands: exibe uma lista dos comandos disponíveis e seus parâmetros.

Comandos

Os Cmdlets (pronuncia-se "command-lets") são comandos do PowerShell. Eles não são case sensitive (tanto faz estar em maiúsculo ou minúsculo) e usam uma convenção simples de nomes, baseada em Verbo-Substantivo (em inglês), que descreve claramente a ação e o objeto que será manipulado. Por exemplo, o comando:

```
Get-Process
```

obter informações sobre processos em execução e o comando

```
Get-Help Test-Connection
```

paar obter auxilio sobre o comando "Test-Connection".

Com a padronização da nomeclatura dos comandos, buscou-se facilitar o seu entendimento. Outro exemplo, o comando para desligar um computador é *Stop-Computer*. O comando para listar todos os computadores em uma rede é *Get-Computer*. O comando para obter a data do sistema é *Get-Date*.

Na sequência iremos mostrar mais alguns comandos.

Comandos de Auxílio

- Para ter uma lista de todos os comandos disponíveis no PowerShell:

```
get-Command
```

Por exemplo, Get-Command -Verb Get (lista todos os cmdlets que começam com "Get").

- Comando para fornece informações detalhadas sobre um cmdlet:

```
Get-Help
```

Por exemplo, Get-Help Get-Process para obtermos ajuda relacionada ao comando Get-Process.

- O alias cria atalhos para comandos. Usar alias permite que usuários com experiência em outros shells (por exemplo, Linux) utilizem nomes de comando comuns que já conhecem para operações semelhantes no PowerShell. Para listar os atalhos, digite:

```
Get-Alias
```

Através dos alias podemos utilizar comandos como: cd , cat, cp, dir, echo, ls, mount, man, kill, move, mv, ps, rm, entre outros.

- Na dúvida sobre os parâmetros de um determinado cmdlet:

```
Show-Command
```

Por exemplo, Show-Command Get-Date retorna os parâmetros do comando Get-Date.

Manipulação de Arquivos e Pastas

Com o PowerShell você pode navegar pelo Windows e manipular os arquivos e pastas. Na sequência será apresentado os principais comandos sobre manipulação de arquivos e pastas.

- Para listar todos os arquivos e pastas dentro de uma pasta:

```
Get-ChildItem
```

Por exemplo, Get-ChildItem -Path C:\Users\.

- Cria um novo arquivo ou pasta:

```
New-Item
```

Por exemplo, New-Item -Path C:\Temp\novo_arquivo.txt -ItemType File.

- Para exclui arquivos ou pastas:

```
Remove-Item
```

Por exemplo, Remove-Item -Path C:\Temp\arquivo.txt.

- Para copiar arquivos ou pastas:

```
Copy-Item:
```

Por exemplo, Copy-Item -Path C:\arquivo.txt -Destination D:\.

Se o arquivo de destino já existir, a tentativa de cópia falhará. Para substituir um destino pré-existente, use o parâmetro Force:

```
Copy-Item -Path C:\boot.ini -Destination C:\boot.bak -Force
```

Copiar a pasta funciona da mesma maneira. Este comando copia a pasta c:\test1 na nova pasta c:\DeleteMe recursivamente:

```
Copy-Item C:\test1 -Recurse C:\DeleteMe
```

Você também pode copiar uma seleção de itens. Por exemplo, o comando a seguir copia todos os arquivos .txt contidos em qualquer lugar em c:\data em c:\temp\text:

```
Copy-Item -Filter *.txt -Path c:\data -Recurse -Destination C:\temp\text
```

- Para modificar a sua loalização poderá utilizar o comando Set-Location e a pasta de destino.

```
Set-Location
```

Por exemplo, Set-Location Music para entrar na pasta Music.

- Para mover arquivos ou pastas

```
Move-Item
```

Por exemplo, Move-Item -Path C:\arquivo.txt -Destination D:\.

Configurações de Redes

Na sequência teremos os principais comandos (cmdlets) de PowerShell relacionado a configurações de redes.

- Testes de Conectividade

```
Test-Connection
```

Equivalente ao ping que envia mensagens icmp.

```
Test-NetConnection
```

Este comando realiza teste de conexão. Se não for passado nenhum endereço, ele testará um endereço da microsoft. Mas é possível informar o hostname ou IP para que o teste seja feito. Por padrão o teste é feito utilizando ICMP, e no output temos o status (True/False), o RTT, o IP de origem e IP de destino. Para mostrar maiores detalhes utilizar a opção -InformationLevel “Detailed”.

O Test-NetConnection também nos permite testar portas específicas. Neste caso usamos o parâmetro -Port. O status True para TcpTestSucceeded indica que a porta está aberta no host de destino.

- Resolução de Domínio

```
Resolve-DnsName
```

Equivalente ao nslookup, com o objetivo de resolução de endereços DNS.

- Tabela de Roteamento

O comando Get-NetRoute mostra a tabela de rotas do Windows.

```
Get-NetRoute
```

- Configuração de Interface

O comando Get-NetAdapter é utilizado para obter informações sobre as placas de rede (adaptadores de rede) em um sistema Windows. Por exemplo, para listar todas as placas de rede:

```
Get-NetAdapter
```

Cada placa de rede recebe um identificador, chamado de InterfaceIndex. Com ele poderemos mostrar as configurações e configura-la. Por exemplo, para mostrar as informações da placa 16, utilizamos:

```
Get-NetAdapter -InterfaceIndex 13
```

O comando Get-NetRoute fornece informações sobre a tabela de roteamento do computador. Digite e observe os resultados.

```
Get-NetRoute
```

Para obtermos o endereço do ser vidor DNS configurado.

```
Get-DnsClientServerAddress
```

O comando New-NetIPAddress é utilizado para configurar um endereço IP fixo. Por exemplo, para configurarmos a interface Ethernet, observe abaixo.

```
New-NetIPAddress -InterfaceAlias ”Ethernet” -IPAddress 192.168.1.10 -PrefixLength 24 -DefaultGateway 192.168.1.1
```

Para alterar o servidor DNS de uma interface de rede no PowerShell, podemos usar o cmdlet Set-DnsClientServerAddress. Por exemplo, para configurarmos os servidores DNS do Google (8.8.8.8 e 8.8.4.4) na interface Ethernet usamos o comando:

```
Set-DnsClientServerAddress -InterfaceIndex 13 -ServerAddresses 8.8.8.8, 8.8.4.4
```

Scripts no PowerShell

Um script no PowerShell é um arquivo de texto contendo uma sequência de comandos que podem ser executados automaticamente para realizar tarefas específicas. Os scripts geralmente possuem a extensão .ps1 e são usados para:

- Automatizar tarefas administrativas ou repetitivas.
- Configurar sistemas e ambientes.
- Gerenciar redes, aplicativos e outros recursos.

Scripts permitem incluir estruturas como loops (for, while) e condições (if, else), tornando-os altamente versáteis para automatização e desenvolvimento.

Criando Scripts

Para criar um script inicial utilizaremos o editor de texto PowerShell ISE. Para isso:

1. Clique em "Iniciar", digite "ISE" e clique em "ISE do Windows PowerShell".

2. Escreva os seguintes comandos:

```
Write-Host "Bem-vindo ao PowerShell!"
Get-Date
```

3. Salve o arquivo com a extensão .ps1, como meu_script.ps1.

4. Execute o script digitando seu nome precedido por .\:

```
.\meu_script.ps1
```

Variáveis

No PowerShell, as variáveis armazenam dados temporariamente durante a execução de comandos e scripts. Elas são criadas com o prefixo $ seguido pelo nome da variável. Para criar uma variável, basta atribuir um valor a ela usando o operador = (igual). Por exemplo:

```
$nome = "João"
$idade = 25
```

Neste exemplo: $nome armazena o texto "João" e $idade armazena o número 25.

As variáveis no PowerShell são dinamicamente tipadas, ou seja, você não precisa definir explicitamente o tipo de dado. Alguns exemplos:

- Texto (String): $mensagem = "Olá, mundo!"
- Número Inteiro: $quantidade = 10
- Número Decimal: $preco = 19.99
- Booleano: $ativo = $true

Você pode verificar o tipo de uma variável com o método .GetType():

```
$variavel = 42
$variavel.GetType()
```

Podemos utilizar o cmdlet Read-Host para receber a entrada de dados e armazenar em forma de variável para ser utilizada na estrutura do seu script. Por exemplo:

```
$nome = Read-Host "Qual o seu nome ?"
Write-Output "Oi!, $nome tudo bem?"
```

IF

O operador if é usado para executar blocos de código com base em condições especificadas. Ele permite que os scripts tomem decisões de acordo com valores ou estados. A estrutura básica de um comando if no PowerShell é:

```
if (<condição>) {
  # Código a ser executado se a condição
for verdadeira
}
```

É possível adicionarmos condições alternativas com os operadores elseif e else:

```
if (<condição1>) {
  # Código se condição1 for verdadeira
} elseif (<condição2>) {
  # Código se condição2 for verdadeira
} else {
  # Código se nenhuma das condições
anteriores for verdadeira
}
```

Por exemplo, o código abaixo verifica se a variável $numero é maior que 5.

```
$numero = 10
if ($numero -gt 5) {
  Write-Host "O número é maior que 5."
}
```

For

O operador for é uma estrutura de controle de iteração usada no PowerShell para repetir a execução de um bloco de código um número definido de vezes. A estrutura básica do for consiste em três partes dentro dos parênteses, separadas por ponto e vírgula (;):

```
for (<inic>; <cond>; <increm>){
  Código a ser executado
}
```

onde: inici define e inicializa a variável de controle do loop. Cond: especifica a condição que será avaliada antes de cada iteração. O loop continuará enquanto essa condição for verdadeira. Increm: atualiza a variável de controle após cada iteração.

Por exmplo, o loop abaixo faz uma contagem regressiva de 10 até 1.

```
for ($i = 10; $i -ge 1; $i–){
  Write-Host “Contagem regressiva: $i”
}
```

Funções

Em PowerShell, uma função é um bloco de código que pode ser reutilizado em seu script. A sintaxe básica para definir uma função é a seguinte:

```
Function NomeDaFuncao {
  Param(
    [tipo]$parametro1,
    [tipo]$parametro2
  )
  # Código da função
  Write-Output “Resultado: $parametro1,
$parametro2”
}
```

Para definir a função use a palavra-chave *Function* seguida do nome que você deseja para a função. O nome da função deve ser único. Os parametros são opcionais e é atrbiuido com a palavra-chave *Param*. Cada parâmetro é declarado com o tipo de dado entre colchetes ([tipo]) seguido do nome do parâmetro. O código que você deseja executar quando a função for chamada vai dentro das chaves . Como retorno você pode usar o comando Write-Output ou simplesmente retornar o valor diretamente.

Por exemplo, a função a seguir recebe dois parâmetros (numero1 e numero2), soma os dois números e retorna o resultado.

```
Function Somar {
  Param(
    [int]$numero1,
    [int]$numero2
  )
  $soma = $numero1 + $numero2
  Write-Output $soma
}
# Chamando a função
$resultado = Somar -numero1 5 -numero2
3
Write-Output “Resultado da soma:
$resultado”
```

Atividades

1. Crie um script que gera uma tabuada para um número fornecido. Por exemplo: passando o número 10, ele me retorna: 10 x 1 = 10; 10 x 2 = 20; ...; 10 x 10 = 100.

2. Crie um script que leia um número e diga se ele é par ou ímpar.

3. Escreva uma função que crie uma pasta especificada pelo usuário.

4. Crie uma função ao qual passamos como parâmetros um endereço (IP ou DNS) e seja verificado a coneção via comando Test-Connection.

5. Escreva um script para exibir a configuração atual de rede do sistema, como o endereço IP, máscara de sub-rede, gateway padrão e servidores DNS. Pesquise os comandos Get-NetAdapter, Get-NetIPAddress e Get-NetRoute.

6. Crie uma função que receber um diretório como parâmetro e listar os arquivos presentes neste diretório.