Full credit is given to the above companies including the OS that this PDF file was generated!

PowerShell Get-Help on command 'Use-Transaction'

PS C:\Users\wahid> Get-Help Use-Transaction

NAME

Use-Transaction

SYNOPSIS

Adds the script block to the active transaction.

SYNTAX

Use-Transaction [-TransactedScript] <System.Management.Automation.ScriptBlock>

[-UseTransaction] [<CommonParameters>]

DESCRIPTION

The `Use-Transaction` cmdlet adds a script block to an active transaction.

This enables you to do transacted scripting by using transaction-enabled

Microsoft .NET Framework objects. The script block can contain only

transaction-enabled .NET Framework objects, such as instances of the

Microsoft.PowerShell.Commands.Management.TransactedString class.

The UseTransaction parameter, which is optional for most cmdlets, is required

when you use this cmdlet.

`Use-Transaction` is one of a set of cmdlets that support the transactions feature in Windows PowerShell. For more information, see about_Transactions (../Microsoft.PowerShell.Core/About/about_Transactions.md).

PARAMETERS

-TransactedScript <System.Management.Automation.ScriptBlock>

Specifies the script block that is run in the transaction. Enter any valid script block enclosed in braces (`{}`). This parameter is required.

-UseTransaction <System.Management.Automation.SwitchParameter>

Includes the command in the active transaction. This parameter is valid only when a transaction is in progress. For more information, see about_transactions (../Microsoft.PowerShell.Core/About/about_Transactions.md).

<CommonParameters>

This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see about_CommonParameters (https:/go.microsoft.com/fwlink/?LinkID=113216).

--- Example 1: Script by using a transaction-enabled object ---

```
Start-Transaction
$transactedString = New-Object
Microsoft.PowerShell.Commands.Management.TransactedString
$transactedString.Append("Hello")
Use-Transaction -TransactedScript { $transactedString.Append(", World") }
-UseTransaction
$transactedString.ToString()
```

Hello

Use-Transaction -TransactedScript { $transactedString.ToString() }
-UseTransaction

Hello, World

Complete-Transaction
$transactedString.ToString()

Hello, World

This example shows how to use `Use-Transaction` to script against a transaction-enabled .NET Framework object. In this case, the object is a TransactedString object.

The first command uses the `Start-Transaction` cmdlet to start a transaction.

The second command uses the New-Object command to create a TransactedString object. It stores the object in the `$TransactedString` variable.

The third and fourth commands both use the Append method of the TransactedString object to add text to the value of `$TransactedString`. One command is part of the transaction. The other command is not.

The third command uses the Append method of the transacted string to add Hello to the value of `$TransactedString`. Because the command is not part of the transaction, the change is applied immediately.

The fourth command uses `Use-Transaction` to add text to the string in the transaction. The command uses the Append method to add ", World" to the value of `$TransactedString`. The command is enclosed in braces (`{}`) to make it a script block. The UseTransaction parameter is required in this command.

The fifth and sixth commands use the ToString method of the TransactedString object to display the value of the TransactedString as a string. Again, one command is part of the transaction. The other transaction is not.

The fifth command uses the ToString method to display the current value of the $TransactedString variable. Because it is not part of the transaction, it displays only the current state of the string.

The sixth command uses `Use-Transaction` to run the same command in the transaction. Because the command is part of the transaction, it displays the current value of the string in the transaction, much like a preview of the transaction changes.

The seventh command uses the `Complete-Transaction` cmdlet to commit the transaction.

The final command uses the ToString method to display the resulting value of the variable as a string.

---------- Example 2: Roll back a transaction ----------

```
Start-Transaction
$transactedString = New-Object
Microsoft.PowerShell.Commands.Management.TransactedString
$transactedString.Append("Hello")
Use-Transaction -TransactedScript { $transactedString.Append(", World") }
-UseTransaction
Undo-Transaction
$transactedString.ToString()
```

Hello

This example shows the effect of rolling back a transaction that includes

`Use-Transaction` commands. Like all commands in a transaction, when the transaction is rolled back, the transacted changes are discarded and the data is unchanged.

The first command uses `Start-Transaction` to start a transaction.

The second command uses `New-Object` to create a TransactedString object. It stores the object in the `$TransactedString` variable.

The third command, which is not part of the transaction, uses the Append method to add "Hello" to the value of `$TransactedString`.

The fourth command uses `Use-Transaction` to run another command that uses the Append method in the transaction. The command uses the Append method to add ", World" to the value of `$TransactedString`.

The fifth command uses the Undo-Transaction cmdlet to roll back the transaction. As a result, all commands performed in the transaction are reversed.

The final command uses the ToString method to display the resulting value of `$TransactedString` as a string. The results show that only the changes that were made outside the transaction were applied to the object.

REMARKS

To see the examples, type: "get-help Use-Transaction -examples".
For more information, type: "get-help Use-Transaction -detailed".
For technical information, type: "get-help Use-Transaction -full".
For online help, type: "get-help Use-Transaction -online"