

A PowerShell Cookbook for DBAs

Comparing PowerShell and T-SQL Syntax Trevor Barkhouse

Introduction

Since this presentation is designed specifically for database professionals (SQL Server administrators and developers), I created this document to help translate your current expertise with Transact-SQL into proficiency with PowerShell.

Variables

- Instead of starting with the At Sign ("@"), PowerShell variables begin with a Dollar Sign ("-\$")
- Data types
 - By default, PowerShell variables are loosely typed, meaning that they can hold any type of data (similar to the SQL_Variant data type in T-SQL)
 - Unlike T-SQL, PowerShell variables can be used without be declared first
 - However, for scripts, it is usually a good idea to declare them before usage
 - To enforce declaring variables before usage, call the [Set-PSDebug](#) Cmdlet with the "-Strict" switch:
 - `Set-PSDebug -Strict;`
 - PowerShell variables can be strongly typed by specifying the data type when declaring the variable
 - In T-SQL, variable declaration consists of the DECLARE keyword, the name of the variable, the AS keyword and then the data type name
 - `DECLARE @Birthdate AS DateTime;`
 - In PowerShell, the (strongly typed) variable declarations consist of the data type name, within square brackets, followed immediately by the data type name
 - `[DateTime]$BirthDate;`
 - Just like T-SQL (with SQL Server 2008 or later), PowerShell variables can be initialized in the same statement as they are declared
 - T-SQL:
 - `DECLARE @Birthdate AS DateTime = '1970-01-01';`
 - PowerShell:
 - `[DateTime]$Birthdate = '1970-01-01';`

- Similar to T-SQL, PowerShell allows both implicit and explicit casting of values to different data types
 - The variable declaration/assignment statements above each cast from text data types (string literals) to the respective DateTime data types
 - Examples of explicit casting are:
 - T-SQL:
 - `SET @ProductNumber = CAST('4321' AS Int);`
 - PowerShell:
 - `$ProductNumber = [Int]('4321');`
- Unlike T-SQL, all PowerShell variables are technically objects
 - In addition to storing data, they can also have behaviors (events and methods)
 - This is similar to the newer, CLR-based data types introduced with SQL Server 2005, such as the XML data type
- All PowerShell data types have a corresponding class/object type in the .NET Frameworks
- In T-SQL, variables are scoped to batches or routines. PowerShell variable's have a [multitude of possible scopes](#).

Line Terminators

- Just like T-SQL, PowerShell uses semi-colons (";") for line terminators
- Also like T-SQL, hardly anyone uses them

Aliases

- Aliases are shortcuts for PowerShell commands, to reduce the amount that you have to type, filling the same purpose as the use of aliases for column or table names in a T-SQL query

- T-SQL:

```
SELECT [UserDBs].[database_id] AS [DatabaseID],
       [UserDBs].[name] AS [DatabaseName],
       (
         SELECT (Sum(Cast([DatabaseFiles].[size] AS BigInt)) * 8192)
         FROM [sys].[master_files] AS [DatabaseFiles]
         WHERE
           (
             ([DatabaseFiles].[type] = 1)
             AND
             ([DatabaseFiles].[database_id] = [UserDBs].[database_id])
           )
       ) AS [TransactionLogSize_Bytes]
FROM [sys].[databases] AS [UserDBs]
WHERE ([UserDBs].[database_id] > 4)
ORDER BY [DatabaseID] ASC;
```

- PowerShell:

- `gps | ? {$_.Name -like 'p*'};`
 - Instead of:
 - `Get-Process | Where-Object {$_.Name -like 'p*'};`
- There are many built-in aliases, but you can create your own too
 - For example:
 - “dir” is the alias for the [Get-ChildItem](#) Cmdlet
 - “cls” is the alias for the [Clear-Host](#) function
 - “gwi” is the alias for the [Get-WMIObject](#) Cmdlet
- Aliases can be listed by accessing the [Alias provider](#):
 - `Get-ChildItem -Path Alias:d*;`
- They can also be listed via the [Get-Alias](#) Cmdlet:
 - `Get-Alias;`
- As shown above, aliases aren’t limited to being letters
 - “%” is the alias for the [ForEach-Object](#) Cmdlet
 - “?” is the alias for the [Where-Object](#) Cmdlet
- Aliases can be created via the [New-Alias](#) Cmdlet:
 - `New-Alias -Name 'drop' -Value 'Remove-Item';`

Discoverability

- PowerShell has a lot of function for exploring the language as well as any extensions to it
- This is analogous to being able to query the [system catalog](#) and [DMVs/DMFs](#) in SQL Server
- The main Cmdlets for this type of discovery are:
 - [Get-Command](#)
 - [Get-Help](#)
 - [Get-Member](#)
- Most PowerShell tutorials start by introducing these Cmdlets, as they are invaluable for teaching yourself more about PowerShell

Grouping Symbols

- Parentheses, by themselves, are basically equivalent in PowerShell and T-SQL
- Curly braces (“{” and “}”), by themselves, are equivalent to the BEGIN and END keywords in T-SQL. They indicate the boundaries of a script block. There are more uses for script blocks in PowerShell than in T-SQL, but they also overlap a lot, such as with:
 - Function definitions
 - If ... Else statements
 - Looping constructs (Do, For, While)

- Square brackets ("[" and "]") have two different purposes:
 - As index operators
 - To indicate a data type
- Grouping symbols that are prefixed with other symbols, have special meanings:
 - The "@(" and ")" combination forces its contents to be evaluated as an array of values, even if there is only one element
 - The "@{" and "}" combination indicates that its contents are a key-value pair in a [System.Collections.HashTable](#) object
 - The "\$(" and ")" combination causes its contents to be evaluated as PowerShell code and then be replaced by the result of the expression

Operators

- Other than bitwise operators, PowerShell contains all of the operators that T-SQL does, plus many others
- Logical and comparison operators begin with a dash, so that PowerShell can maintain backward compatibility with the legacy command shell (Cmd.exe)
 - Comparison operators:
 - T-SQL
 - =
 - >=
 - >
 - <=
 - <
 - LIKE
 - <>
 - PowerShell
 - -eq
 - -ge
 - -gt
 - -le
 - -lt
 - -like
 - -ne
 - Logical operators:
 - T-SQL
 - AND
 - NOT
 - OR
 - PowerShell
 - -and

- -not (or !)
- -or
- This tends to be one of the harder things to get used to when learning PowerShell

Changes Made During a PowerShell Session

- For the most part, changes made during a PowerShell session will not persist once the session has ended
- The PowerShell session can be thought of as the [tempdb] database:
 - The PowerShell session is created when the PowerShell console is launched
 - Creating functions in session is analogous to created functions or stored procedures in the [tempdb] database
 - Creating variables in the session is like creating tables in the [tempdb] database
 - The objects can be created, used, and dropped as desired
 - When the PowerShell console is closed, the objects cease to exist (as objects created in the [tempdb] database are destroyed when the SQL Server service restarts)
- In order for objects to be available across sessions, they should be added to a [profile script](#)
- This is similar to added a stored procedure or table to the [model] system database... the next time that SQL Server restarts, the objects will be available in the [tempdb] database (since it is recreated based on the [model] database)

Custom Properties When Outputting Data

- In a T-SQL query, a calculate/derived column is achieved by putting an expression in the SELECT clause and using the AS keyword to name the new column (see above for the rest of the query):
 - `SELECT (Sum(Cast([DatabaseFiles].[size] AS BigInt)) * 8192) [...]`
- In PowerShell, the mechanism is a bit more cryptic
- The ability to add custom properties (calculated columns) is provided by the following Cmdlets:
 - [Add-Member](#)
 - While this Cmdlet has a lot of power, it is less frequently used and is a bit more advanced, so I won't cover it any further in this document
 - [Format-Custom](#)
 - [Format-List](#)
 - [Format-Table](#)
 - [Format-Wide](#)

- [Select-Object](#)
- For the latter five Cmdlets, Format-* and Select-Object, the calculated column is achieved via a special hash table (dictionary)
 - There should be an element with a key of "Label" (for the Format-* Cmdlets), or "Name" for the Select-Object Cmdlet, and a value with the column name
 - As of PowerShell version 2.0, the "Label" and "Name" keys are interchangeable for all five Cmdlets
 - There should also be an element with a key of "Expression" and a value of a script block
- Hash tables can also be used to create customer properties for the [Sort-Object](#) Cmdlet (see its help for more information)