# Real-World PowerShell for SQL Administration

Brian Hansen
brian@tf3604.com
@tf3604

VIRTUAL

PASS
SQLSATURDAY
CHATTANOOGA | JUN 27 2020

## Welcome to SQL Saturday

- Enjoy this day of learning
- Be sure to visit and thank the sponsors
- Be sure to thank the organizer and volunteers
- Take time to NETWORK with others. That's what this is really all about!
- Act professionally and treat others with respect (like this was a work environment)

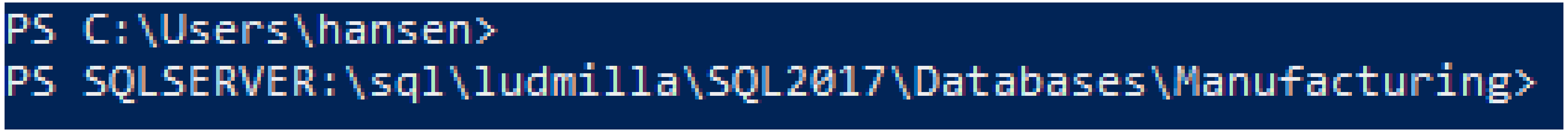
# Agenda

- “Gotchas”
- Tips
- PowerShell and .NET
- Scripts

- Gotchas
- Tips
- PowerShell and .NET
- Scripts

## “Gotchas”: Providers

- Be in the right provider
  - PowerShell has many virtual drives and folder structures



```
PS C:\Users\hansen>
PS SQLSERVER:\sql\ludmilla\SQL2017\Databases\Manufacturing>
```

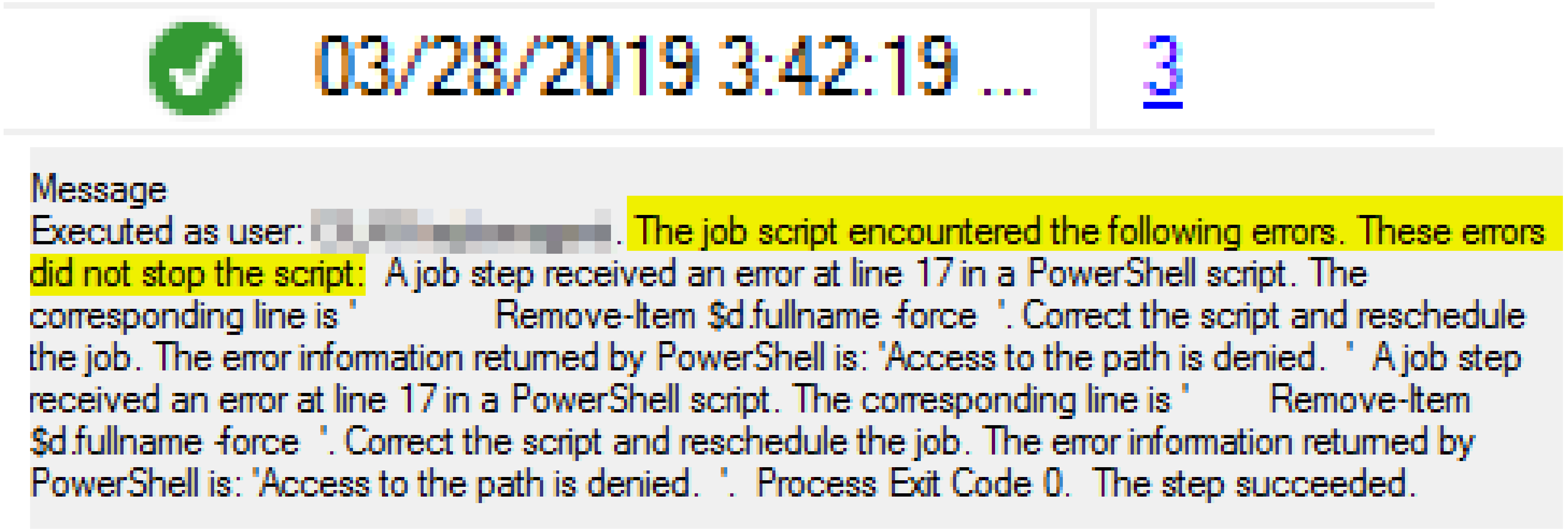
  - What is the default provider in SQL Agent?

It is SQLSERVER:\

```
Set-Location c:;
```

## “Gotchas”: Silent Failures

- Silent failures (non-terminating errors)



✓ 03/28/2019 3:42:19 ... | 3

```
Message
Executed as user: [redacted]. The job script encountered the following errors. These errors did not stop the script:  A job step received an error at line 17 in a PowerShell script. The corresponding line is '            Remove-Item $d.fullname -force  '. Correct the script and reschedule the job. The error information returned by PowerShell is: 'Access to the path is denied.  '  A job step received an error at line 17 in a PowerShell script. The corresponding line is '        Remove-Item $d.fullname -force  '. Correct the script and reschedule the job. The error information returned by PowerShell is: 'Access to the path is denied.  '.  Process Exit Code 0.  The step succeeded.
```

## “Gotchas”: Silent Failures

- How to fix?

```
$ErrorActionPreference = "Stop";
```

- But what if we really want to ignore an error?

```
try { ... }
catch { ... }
finally { ... }
```

## “Gotchas”: Silent Failures

- More about try … catch … finally

```
try { $connection.Open(); }
catch [System.Data.SqlClient.SqlException]
{
    $except = $_.Exception;
    $errorMessage = $except.Message;
    $errorNumber = $except.Number;
}
```

## “Gotchas”: Agent compatibility

- Jobs running in agent must comply with the proper version of PowerShell
  - SQL 2008 to 2012 → PowerShell 2.0
  - SQL 2014 to 2019 → PowerShell 5.1
- Cannot use PowerShell features beyond the loaded version!

# “Gotchas”: Agent compatibility

- What can go wrong with this?

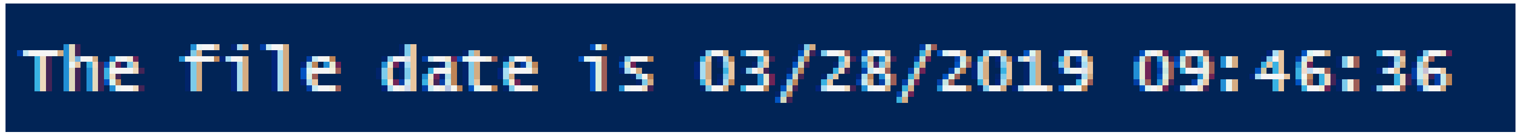
```
$files = Get-ChildItem “c:\temp";
$mostRecent = ($files |
    Sort-Object $_.LastWriteTime -Descending)[0];
```

- What if c:\temp is empty?
- What if c:\temp has one file?
    - Will fail in PS 2.0 (`$mostRecent` is of type FileInfo, not FileInfo[]) Unable to index into an object of type System.IO.FileInfo.
    - Will work in PS 5.1 (still of type FileInfo, but PS allows indexing)

## “Gotchas”: PS Features Not in Agent

```
$file = New-Object System.IO.FileInfo("C:\temp\data.txt");
Write-Output "The file date is $($file.LastWriteTime)";
```

- This works in Standard PowerShell



```
The file date is 03/28/2019 09:46:36
```

- But not in SQL PS (either 2.0 or 5.1)

  Unable to start execution of step 1 (reason: line(2): Syntax error). The step failed.

- This works:

```
Write-Output ("The file date is " + $file.LastWriteTime);
```

## “Gotchas”: Getting just files (or folders)

```
Get-ChildItem "c:\data“ -Recurse;
```

- Returns both files and folders
- What if we just want files?

```
... | Where-Object { -not $_.PSIsContainer };
```

- Or just folders?

```
... | Where-Object { $_.PSIsContainer };
```

## “Gotchas”: Dot-sourcing

- Imagine a PS script with initialization features (variables, functions), and we call that script.

```
C:\data\Initialize.ps1;
```

- Then try to access these features

```
Do-Something;
Do-Something : The term 'Do-Something' is
not recognized as the name of a cmdlet,
function, script file...
```

## “Gotchas”: Dot-sourcing

- The script is loaded into a subshell, which is closed when the script is done
- We need to “dot-source” (prefix the call to the script file is a period and space)

  ```
  . C:\data\Initialize.ps1;
  ```

- This causes the script to be loaded within the scope of the current shell

## “Gotchas”: Syntax oddities

- What do these mean?

| Operator | Meaning | Other languages |
|---|---|---|
| -eq | Equality comparison | = or == |
| -ne | Not-equals comparison | <> or != |
| -gt | Greater-than comparison | > |

- But this does not generate an error. Why?

```
if ($x = 4) { Write-Output "True"; }
$x = 3;
if ($x > 0) { Write-Output "True"; }
```

## “Gotchas”: Syntax oddities

- What is this checking for?
- `if (!$?) { Write-Output "Huh?"; }`
- Last command was unsuccessful
    - `$?` means last command was successful
    - `!$?` is the same as `-not $?`

## “Gotchas”: Syntax oddities

- What does the ampersand do here?

```
& "c:\utils\sleep.exe";
```

- Treats the string as a command rather than just a string object.

```
& "c:\utils\sleep.exe";
```

  - Executes sleep.exe

```
"c:\utils\sleep.exe";
```

  - Returns the string “c:\utils\sleep.exe”

## "Gotchas": Syntax oddities

- What data type is the variable?

  ```
  $variable = @{};
  ```

- Hash table
  - To add records:

    ```
    $variable.Add("key1", "value1");
    $variable.key2 = "value2";
    ```

  - Or initialize as

    ```
    $variable = @{"key1" = "value1"; "key2" = "value2"};
    ```

- Gotchas
- **Tips**
- PowerShell and .NET
- Scripts

## Tips: Customizable Variables

- Place customizable variables at top of script
  - Even if not referenced until much later.

```
set-location c:;
$ErrorActionPreference = "Stop";

$backupPath = "\\FileServer\SQL\LogBackups";
$localPath = "S:\SQL\Backups";
$filePattern = "*.bak";
$databaseName = "OurDatabase";
$logFile = "\\FileServer\AppLogs\BackupLog.txt";
```

## Tips: Aliases

- What is this code doing?

```
gci "c:\data"|?{$_.LastWriteTime -gt
"2019-01-01"}|%{$x+=','+$_.Name};
```

- Aliases in PowerShell
  - `gci` = `Get-ChildItem`
  - `?` = `Where-Object` (or `where`)
  - `%` = `ForEach-Object` (or `foreach`)

# Tips: Aliases

```
gci "c:\data"|?{$_.LastWriteTime -gt
"2019-01-01"}|%{$x+=','+$_.Name};
```

```
Get-ChildItem "c:\data" | Where-Object {
$_.LastWriteTime -gt "2019-01-01" } |
ForEach-Object { $x += ',' + $_.Name };
```

# Tips: Aliases

```
Get-ChildItem "c:\data" | Where-Object { $_.LastWriteTime
-gt "2019-01-01" } | ForEach-Object { $x += ',' + $_.Name
};
```

```
Get-ChildItem "c:\data" | `
  Where-Object { $_.LastWriteTime -gt "2019-01-01" } | `
  ForEach-Object { $x += ',' + $_.Name };
```

## Tips: Debugging

- PowerShell ISE or Visual Studio Code is your friend
  - F8 to run highlighted text (or current line)
  - F5 or Debug → Run/Continue
    - Always starts from beginning of script
    - Goes until done or hits breakpoint
  - F10 or Debug → Step Over to run next statement
    - Also F11 to Step Into and Shift-F11 to Step Out
  - F9 to toggle breakpoint

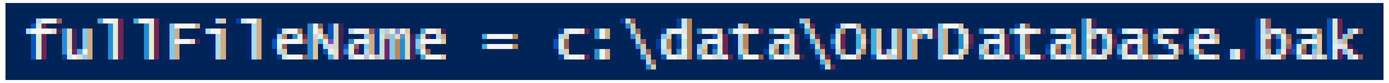
## Tips: Debugging

- Sometimes the script works fine in ISE / VS Code but not in SQL Agent
  - Logging (yeah, it stinks, but …)

Note: custom function; see scripts

```
Write-LogMessage -LogFileName $logName
-Message "About to do something.";
Do-Something;
Write-LogMessage -LogFileName $logName
-Message "Successfully did something.";
```

## Tips: Creating file names



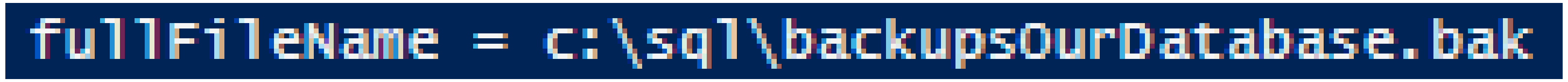
```
fullFileName = c:\data\OurDatabase.bak
```

- Suppose we have the following script

```
## Configure these variables as appropriate
$folderName = "c:\data\";
## End of configuration variables

## Do lots of other stuff, then:
$fileName = "OurDatabase.bak";
$fullFileName = $folderName + $fileName;
```

## Tips: Creating file names



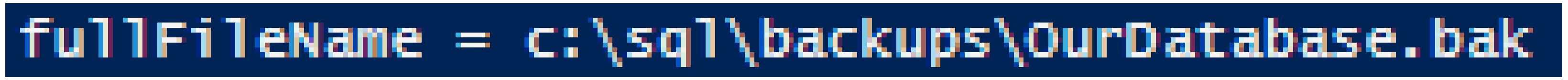
```
fullFileName = c:\sql\backupsOurDatabase.bak
```

- Some time later, we need to change the folder

```
## Configure these variables as appropriate
$folderName = "c:\sql\backups";
## End of configuration variables

## Do lots of other stuff, then:
$fileName = "OurDatabase.bak";
$fullFileName = $folderName + $fileName;
```

# Tips: Creating file names



```
fullFileName = c:\sql\backups\OurDatabase.bak
```

- Use Path.Combine instead

```
## Configure these variables as appropriate
$folderName = "c:\sql\backups";
## End of configuration variables

## Do lots of other stuff, then:
$fileName = "OurDatabase.bak";
$fullFileName = [System.IO.Path]::Combine($folderName, $fileName);
```

- Gotchas
- Tips
- PowerShell and .NET
- Scripts

## PowerShell and .NET

- Knowing a bit of .NET can be really useful
- .NET integrates quite nicely into PS
- Can help to read PS scripts that use .NET
- Exercise great control doing data access
- And file system operations
  - (Though PS wraps file system stuff quite nicely)

# PowerShell and .NET: Data Access

- Common data access objects: `SqlConnection`
- Create from connection string
- Or use `SqlConnectionStringBuilder`

```
$sb = New-Object
    System.Data.SqlClient.SqlConnectionStringBuilder;
$sb["Data Source"] = "server\instance";
$sb["Initial Catalog"] = "AdventureWorks2014";
$sb["Integrated Security"] = $true;

$connection = New-Object
    System.Data.SqlClient.SqlConnection($sb.ToString());
$connection.Open();
```

## PowerShell and .NET: Data Access

- Be sure to close the connection

```
try
{
    $connection.Open();
    #Use the connection
}
finally
{
    $connection.Close();
}
```

## PowerShell and .NET: Data Access

- Common data access objects: `SqlCommand`

```
$sql = "update dbo.AppUser set name = 'Jane' where ID = 4;";
try
{
    $command = New-Object
        System.Data.SqlClient.SqlCommand($sql, $connection);
    $command.CommandTimeout = 3600;

    [void]$command.ExecuteNonQuery();
}
finally
{
    $command.Dispose();
}
```

# PowerShell and .NET: Data Access

- SqlCommand parameters

```
$sql = "update dbo.AppUser set name = @name where ID = @appUserId;";
try
{
    $command = New-Object System.Data.SqlClient.command($sql, $sqlConnection);
    $command.CommandTimeout = 3600;

    $command.Parameters.Add("@name", "Jane");
    $command.Parameters.Add("@appUserId", 4);

    [void]$command.ExecuteNonQuery();
}
finally
{
    $command.Dispose();
}
```

# PowerShell and .NET: Data Access

- `SqlCommand` common methods
  - ExecuteNonQuery (useful for update, insert, delete statements)
  - ExecuteScalar (returns the value of the first column of the first row)
  - ExecuteReader (creates `SqlDataReader` object)
  - ExecuteXmlReader (creates `XmlReader` object)

# PowerShell and .NET: Data Access

- `SqlCommand` and resultsets

```
$sql = "select * from Sales.SalesPerson;";
$table = New-Object System.Data.DataTable;
try
{
    $command = New-Object System.Data.SqlClient.SqlCommand($sql, $connection);
    try
    {
        $adapter = New-Object System.Data.SqlClient.SqlDataAdapter($command);
        [void]$adapter.Fill($table);
    }
    finally
    {
        $adapter.Dispose();
    }
}
finally
{
    $command.Dispose();
}
# Use data in $table
```

For multiple result sets, use:

```
$dataSet = New-Object System.Data.DataSet;
```

And reference `$dataSet` here

## PowerShell and .NET: Data Access

- Accessing resultset data

```
foreach ($table in $dataSet.Tables)
{
    foreach ($row in $table.Rows)
    {
        $businessEntityId = $row["BusinessEntityID"];
        $bonus = $row["Bonus"];
        $salesYtd = $row["SalesYTD"];
        # Use row data here
    }
}
```

# PowerShell and .NET: Data Access

- Accessing resultset data

```
foreach ($table in $dataSet.Tables)
{
    foreach ($row in $table.Rows)
    {
      $businessEntityId = $row.BusinessEntityID;
      $bonus = $row.Bonus;
      $salesYtd = $row.SalesYTD;
      # Use row data here
    }
}
```

## NULL Values in .NET

- PowerShell and .NET null value is `$null`
- SQL null value is `[System.DBNull]::Value`
- Not the same!
- Be sure to use SQL null when setting parameters or checking column values

- Gotchas
- Tips
- PowerShell and .NET
- Scripts (Usage Scenarios)

## Real-World Scripts: Copying files

- Very useful to copy files around (especially backups)
- Make sure agent account has sufficient rights
  - Probably at least “Modify” access
- Many options in PowerShell to copy files

## Real-World Scripts: Copying files

- Copy-Item
  - Built in to PowerShell
  - Very generic (will copy more than files)
  - Limited flexibility

# Real-World Scripts: Copying files

- FileInfo.CopyTo
  - Also not much flexibility

```
$file = New-Object System.IO.FileInfo
   ("c:\temp\original.log");
$file.CopyTo("d:\temp\copy.txt", $true);
```

- File.Copy

```
[System.IO.File]::Copy("c:\temp\original.log",
"d:\temp\copy.txt", $true);
```

# Real-World Scripts: Copying files

- robocopy
  - Adds robustness to copy process (lots of switches)
  - Always use /R (retry) and /W (wait) switches
  - Consider /NP (no progress) and /Z (restartability)
  - Check $LastExitCode greater than 7 for error
  - Can be slow for very large files

```
robocopy c:\temp d:\temp ssis.log /R:3 /W:30;
if ($LastExitCode -gt 7)
{
    throw "Error using ROBOCOPY.";
}
```

## Real-World Scripts: Copying files

- Custom copying scripts
    - Note logic to cleanup old backups

## Real-World Scripts: Reaching Across Servers

- Light-weight and maintainable code to execute SQL across servers
  - Easier to maintain than SSIS package
  - Avoids linked server issues
  - Presumes agent account has rights on remote system
  - Examples: start a job, run a stored procedure, backup Express Edition

- Gotchas
- Tips
- PowerShell and .NET
- Scripts (Real-World Scripts)

## Real-World Scripts

- See the Initialize.ps1 script
    - Invoke using dot-sourcing
    - Consider prefixing with your organization name (XYZ_Initialize.ps1)
- Intended as a source of ideas, not necessarily to use as-is
    - Lots of assumptions
        - only one backup per file
        - backups not split into multiple files
        - certain naming conventions for backup files, etc.

## My Common Usage Job Step

```
set-location c:\;
. \\FileServer\Share\Initialize.ps1;

Backup-Database `
    -InstanceName "SQLHost\oltp" `
    -DatabaseList "ProdDatabase" `
    -BackupPath "s:\SQL\OLTP\Backup" `
    -AppendInstanceToBackupPath $false `
    -UseCompression $true `
    -FileExtension "bak";
```

## Wrap-Up

## Wrap-Up: Summary

- PowerShell is a practical, useful way to automate SQL administration.
- What we've covered today is only the beginning.
  - The power of PowerShell lets us tackle a wide variety of tasks

## Wrap-Up: How about you?

- What other tasks do you accomplish via PowerShell?

## Thank You

- This presentation and supporting materials can be found at www.tf3604.com/poshadmin.
  - Slide deck
  - Scripts

brian@tf3604.com • @tf3604