**Automate file cleanup on SQL Server drives using PowerShell**

By: [Tim Smith](https://www.mssqltips.com/sqlserverauthor/92/tim-smith/)

FROM: <https://www.mssqltips.com/sqlservertip/4147/automate-file-cleanup-on-sql-server-drives-using-powershell/?utm_source=dailynewsletter&utm_medium=email&utm_content=headline&utm_campaign=20160216>

**Problem**

We work with multiple drives on each server and divide responsibility of some drive management to different teams and individuals. Are there ways to make this manageable for automation with PowerShell, where each individual can define further rules from something centrally managed?

**Solution**

One approach to solving the above problem, if the goal is to minimize the amount of daily work, is use a configuration table (or file) for all the servers, drives, SQL instances, teams, and possibly individuals. From that table, teams (or individuals) can get the files from the drives and proceed with the next action.

When people monitor drives on servers, especially through automation, they first want to outline the next steps of what they want to do. For an example:

* We might archive old CSV files, which have been forgotten on a server.
* We might delete old backup files over a year old that have been left.
* We might delete old SQLIO or IOMeter files that have been forgotten.
* We might shrink a log of the size is too large, or taking up too much space on the server.

Those are possible "next steps" and they vary significantly by environment. In this tip, we'll look at how to obtain information, and more often than not, if there are only a combination of next steps (such as deletes, migrations, shrinks, etc.), then those further steps can also be automated. One should be careful though, as automation will do those next steps *every time* without regards to exceptions, if no exceptions are coded. As an example, if I code a next step to remove every file over a month old, even if I want to keep one old file, without an exception written, automation will remove the file.

With this approach, we'll use a configuration table because we may or may not monitor all the drives on a server or computer. You may maintain the full environment, meaning you'll need to report on all the drives, you may maintain only parts of the environment, such as drives for database files, or some drives (like the OS) may be configured for application purposes only and rarely experience additional files. Using a configuration table will help because we can add or remove values if changes occur in the environment (imagine another data drive being removed, or a new drive added for sql files). In the configuration table, I store the SQL instance name, as well as the server name because I may run a further script against the SQL instance, relative to the information I receive from the drives. The table that I use here can be added to, if other steps might also be necessary (such as a share to move a file to, if certain files need to be archived).

Here is the PowerShell script.  This example will return all files that are on the drives that are in the configuration table.

Function Get-Files {

Param(

[string]$fileserver

, [string]$driveletter

)

Process

{

$fulllist = Get-ChildItem "\\$fileserver\$driveletter$\." -Recurse | Select-Object FullName | Sort Length -Descending

foreach ($fileitem in $fulllist)

{

$filename\_big = $fileitem.FullName

#Write-Host $filename\_big

### If we had specific rules for next steps, we would add them here

}

}

}

<#

### Read-FullDriveList reads from a table like this

CREATE TABLE Configuration.dbo.tbDrives(

ServerName VARCHAR(30),

InstanceName VARCHAR(30),

DriveLetter VARCHAR(1),

Environment VARHCAR(1) -- Production, QA, Dev, which may be useful if these are responsibilities per team

)

#>

Function Read-FullDriveList {

Param(

[ValidateLength(4,30)][string]$server

)

Process

{

$drive\_con = New-Object System.Data.SqlClient.SqlConnection

$drive\_con.ConnectionString = "Data Source=$server;Initial Catalog=master;Integrated Security=true;"

$drive\_cmd = New-Object System.Data.SqlClient.SqlCommand

$drive\_cmd.Connection = $drive\_con

$drive\_cmd.CommandText = "SELECT \* FROM Configuration.dbo.tbDrives"

$drive\_cmd.CommandTimeout = 0

try

{

$drive\_con.Open()

$readdrive = $drive\_cmd.ExecuteReader()

while ($readdrive.Read())

{

[string]$drive\_servername = $readdrive["ServerName"]

[string]$drive\_serverletter = $readdrive["DriveLetter"]

Get-Files -fileserver $drive\_servername -driveletter $drive\_serverletter

}

}

catch [Exception]

{

Write-Warning "Read-FullDriveList"

Write-Warning $\_.Exception.Message

}

finally

{

$drive\_cmd.Dispose()

$drive\_con.Dispose()

}

}

}

Let's look at an example where we remove any file that is over thirty days old. Start with the above code and change the **Get-Files** code inside the **Process** section with the following:

### added LastWriteTime property to check for files over 30 days old

$fulllist = Get-ChildItem "\\$fileserver\$driveletter$\." -Recurse | Where-Object {$\_.LastWriteTime -gt (Get-Date).AddDays.(-30) }

foreach ($fileitem in $fulllist)

{

$filename\_big = $fileitem.FullName

### The what if prevents an actual delete from taking place

Remove-Item $filename\_big -WhatIf

}

If we have a drive that we allocate for SQL Server processes, we might run a transaction against SQL Server - for instance, shrinking a log, so storing the instance name is helpful *if maintaining a SQL instance on the server*.

One final reminder: for every interruption, you lose energy. The more you can automate to avoid interruptions, *especially* on defined rules, the more energy you'll save for other more important work.

**Next Steps**

* Since the "next step" scripts vary by environment, define what these need to do.
* Use and change the above scripts to meet your environments needs: consider that the configuration may need to contain more columns for define rules, like a DeleteFile column where a file that hasn't been written to in the number of days stored in the column is removed from the server.

Last Update: 2/16/2016