6

Mailbox Database Management

In this chapter, we will cover:

* Managing the mailbox databases
* Moving databases and logs to another location
* Configuring the mailbox database limits
* Reporting on mailbox database size
* Finding the total number of mailboxes in a database
* Determining the average mailbox size per database
* Reporting on database backup status
* Restoring data from a recovery database

# Introduction

In this chapter, we will focus on several scenarios in which PowerShell scripting can be used to increase your efficiency when managing databases, which are the most critical resources in your Exchange environment. We will look at how you can add and remove mailbox databases, configure database settings, generate advanced reports on database statistics, and more from within the shell.

## Performing some basic steps

To work with the code samples in this chapter, follow these steps to launch the Exchange Management Shell:

1. Log onto a workstation or server with the Exchange Management Tools installed.
2. You can connect using remote PowerShell if you for some reason don't have Exchange Management Tools installed. Use the following command:

$Session = New-PSSession -ConfigurationName Microsoft.Exchange `

-ConnectionUri http://tlex01/PowerShell/ `

-Authentication Kerberos

Import-PSSession $Session

1. Open the Exchange Management Shell by clicking the windows button and go to Microsoft Exchange Server 2016 | Exchange Management Shell.

If any additional steps are required they will be listed at the beginning of the recipe in the Getting ready section.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Remember to start the Exchange Management Shell using Run As Admin to avoid permission problems.  In the chapter, notice that in the examples of cmdlets, I have used the back tick (`) character for breaking up long commands into multiple lines. The purpose with this is to make it easier to read. The back ticks are not required and should only be used if needed. |  |

# Managing the mailbox databases

The Exchange Management Shell provides a set of cmdlets for mailbox database management. In this recipe, we will take a look at how you can use these cmdlets to create, change, or delete mailbox databases. We will also take a look at how the automatic mailbox distribution works.

## How to do it...

The process for managing mailbox databases is pretty straight forward. We’ll start with creating a new mailbox database:

1. To create a mailbox database, use the New-MailboxDatabase cmdlet, as shown in the following example:

New-MailboxDatabase -Name DB4 `

-EdbFilePath E:\Databases\DB4\Database\DB4.edb `

-LogFolderPath E:\Databases\DB4\Logs `

-Server EX01

1. You can mount the database after it has been created using the   
   Mount-Database cmdlet:

Mount-Database -Identity DB4

1. The name of a database can be changed using the Set-MailboxDatabase cmdlet:

Set-MailboxDatabase -Identity DB4 -Name Database4

And, finally, you can remove a mailbox database using the   
Remove-MailboxDatabase cmdlet:

Remove-MailboxDatabase -Identity Database4 `  
-Confirm:$false

## How it works...

The New-MailboxDatabase cmdlet requires that you provide a name for your database and specify the server name where it should be hosted. In the previous example, you can see that we created the DB4 database on the EX01 server. The –EdbFilePath parameter specifies the location for your database file, however it is not required to use for creating the database. Additionally, you can use the -LogFolderPath variable to identify the directory that should hold the transaction logs for this database. If no value is provided for either of these parameters, the database and log directories will be set to the default location within the Exchange installation directory.

Mounting a database is done as a separate step. If you want to create the database and mount it in one operation, pipe your New-MailboxDatabase command to the   
Mount-Database cmdlet, as shown in the following line of code:

New-MailboxDatabase -Name DB10 -Server EX01 | Mount-Database

The Mount-Database cmdlet can be used with mailbox databases. The same is true for its counterpart, Dismount-Database, which allows you to dismount a mailbox database.

As you saw previously, to rename a mailbox database we used the   
Set-MailboxDatabase cmdlet with the -Name parameter. It's important to note that, while this will change the database name in the Active Directory and therefore in Exchange, it does not change the filename or path of the database.

Before running the Remove-MailboxDatabase cmdlet, you will need to move any regular mailboxes, archive mailboxes, or arbitration mailboxes to another database, using the New-MoveRequest or New-MigrationBatch cmdlet. The arbitration mailboxes can be found by using the Get-Mailbox –Arbitration command, these can be found on the first created database.

Keep in mind that the removal of a database is only done logically in the Active Directory. Later on, you will need to manually delete the files and directories used by the database running the Remove-MailboxDatabase cmdlet.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | There are only two required parameters for creating the mailbox database, which are Name and Server. However, the others like EdbFilePath and LogFolderPath are nice to use for specifying the configuration. |  |

## There's more…

The -Database parameter allows you to omit the -Database parameter when creating or moving a mailbox and an agent determines the most appropriate target database based on a number of factors.

The Mailbox Resources Management Agent, a cmdlet extension agent, is the application that runs in the background that handles this and it is enabled by default. The benefit of this is that if you provision multiple mailboxes or move multiple mailboxes at one time without specifying a target database, the mailboxes will be distributed across all of the available mailbox databases in the current Active Directory site from where you are running the commands.

### Understanding automatic mailbox distribution

Each mailbox database has three properties called IsExcludedFromProvisioning, IsExcludedFromInitialProvisioning and IsSuspendedFromProvisioning. These controls whether or not a database can be used for automatic mailbox distribution. By default, all are set to $false, which means that every mailbox database you create is available for automatic distribution out of the box. If you intend to create a mailbox database used strictly for archive mailboxes or you don't want mailboxes to be placed in a particular database automatically, you can exclude the database from being automatically used. To do so, use the following command syntax after the database has been created:

Set-MailboxDatabase -Identity DB1 -IsExcludedFromProvisioning ` $true

When the IsExcludedFromProvisioning property is set to $true, you can still manually create mailboxes in the database, but it will not be used for automatic distribution.

### Taking it a step further

Let's look at an example of creating mailbox databases in bulk. This can be helpful when creating many databases. The following code can be used for doing the job:

$data = Import-CSV .\DBs.csv

foreach($row in $data) {

$DBName = $row.DBName

$LogPath = ‘E:\Databases\’ + $DBName + ‘\Logs’

$DBPath = ‘E:\Databases\’ + $DBName + ‘\Database\’ + $DBName + ‘.edb’

$Server = $row.Server

New-MailboxDatabase –Name $DBName –Server $Server –Edbfilepath ` $DBPath –Logfolderpath $LogPath

}

foreach($row in $data) {

$DBName = $row.DBName

Mount-Database $DBName

}

In this example, we create an array by importing a CSV file. We start looping through each row in the file, also in the loop we create new variables named DBName, LogPath, DBPath and Server. These variables are then used in the cmdlet   
New-MailboxDatabase. Finally, with this small script we’ll loop through each row again and create a new variable named DBName and try to mount each database in the CSV file.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | The resource called MSFT\_xExchMailboxDatabase in xExchange moduled for Desired State Configuration can be used for creating databases.  When creating databases in larger environments it can be impossible to mount the databases immediately, in these cases let the Active Directory replication finish. When it’s completed the databases can be mounted. |  |

## See also

Reporting on mailbox database size

Moving databases and logs to another location

# Moving databases and logs to another

# location

As your environment grows or changes over time, it may be necessary to move one or more databases and their log streams to another location. This is one of those tasks that’s required to be done from the Exchange Management Shell. In advantage the shell gives you some more flexibility. In this recipe, you will learn how to move database and log files to another location.

## How to do it...

To move the database file and log stream for the DB1 database to a new location, use the following command syntax:

Move-DatabasePath -Identity DB1 `

-EdbFilePath F:\Databases\DB1\Database\DB1.edb `

-LogFolderPath F:\Databases\DB1\Logs `

-Confirm:$false `

-Force

After executing the preceding command, the DB1 database and log files will be moved to the F:\Databases\DB1\Database directory, without prompting you for confirmation.

## How it works...

In this example, you can see that we are moving both the database file and the transaction logs to the same directory. You can use different directories or even separate disk spindles as the locations for the database and log folder paths if needed.

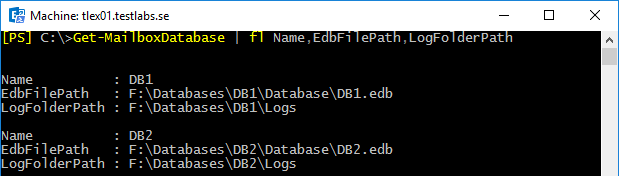
To remove the confirmation prompts, we need to set the -Confirm parameter to $false and also use the -Force switch parameter. This may be an important detail if you are running this cmdlet from an automated script. If not used, the cmdlet will not make any changes until an operator confirms it in the shell.

Obviously, in order to move the database file or the logs, the database will need to be taken offline for the duration of the move. The Move-DatabasePath cmdlet will automatically dismount the database and remount it when the move process is complete. If the database is already dismounted at the time that you initiate a move, the database will not be automatically mounted upon completion of the command and you will need to mount it manually using the Mount-Database cmdlet. Obviously, any users with a mailbox in a dismounted database will be unable to connect to their mailbox. If you need to move a database, ensure that this can be done during a time that will not impact end users.

Keep in mind that databases that are replicated within a Database Availability Group (DAG) cannot be moved. Each database copy in a DAG needs to use the same local path for the database and logs, so you cannot change this after copies have already been created. If you need to change the paths for a replicated database, you will need to remove all database copies and perform the move. Once this process has been completed, you can create new database copies that will use the new path.

## There's more…

Before changing the EdbfilePath or the LogFolderPath locations for a database, you may want to check the existing configuration. To do so, use the Get-MailboxDatabase cmdlet, as shown in the following screenshot:



7081EN\_06\_01

Here you can see that we are piping the Get-MailboxDatabase cmdlet to Format-List (using the fl alias) and selecting the Name, EdbFilePath, and LogFolderPath properties, which will display the relevant information for every database in the organization. You can retrieve this information for a single database by specifying the name of the database using the -Identity parameter.

### Manually moving databases

In certain situations, you may prefer to manually copy or move the database and log files instead of allowing the Move-DatabasePath cmdlet to move the data for you. In this case, you can use the following process:

1. Let's say that you need to move the DB2 database to the F:\ drive. To do this manually, the first thing you will want to do is dismount the database:

Dismount-Database -Identity DB2 -Confirm:$false

1. Next, use whatever method you prefer to copy the data to the new location on the F:\ drive. After the data has been copied, use the Move-DatabasePath cmdlet,   
   as shown next, to update the configuration information in Exchange:

Move-DatabasePath -Identity DB2 `

-EdbFilePath F:\Databases\DB2\Database\DB2.edb `

-LogFolderPath F:\Databases\DB2\Logs `

-ConfigurationOnly `

-Confirm:$false `

-Force

1. The preceding command uses the -ConfigurationOnly switch parameter when running the Move-DatabasePath cmdlet. This ensures that only the configuration of the database paths is updated and that there is no attempt to copy the data files to the new location.

After the files are manually moved or copied and the configuration has been changed, you can re-mount the database, as shown next:

Mount-Database -Identity DB2

At this point, the database will be brought online and the move operation will be complete.

### Taking it a step further

Let's look at an example of how we can use the shell to move databases in bulk. Let's say we have added a new disk to the EX01 server using the S:\ drive letter and all the databases need to be moved to this new disk under the Databases root directory. The following code can be used to perform the move:

foreach($i in Get-MailboxDatabase -Server EX01) {

$DBName = $i.Name

Move-DatabasePath -Identity $DBName `

-EdbFilePath "S:\Databases\$DBName\Database\$DBName.edb" `

-LogFolderPath "S:\Databases\$DBName\Logs" `

-Confirm:$false `

-Force

}

In this example, we use the Get-MailboxDatabase cmdlet to retrieve a list of all the mailbox databases on the EX01 server. As we loop through each mailbox database, we   
move the EDB file and log path under the S:\Database folder in a subdirectory that   
matches the name of the database.

You can type the preceding code straight into the shell or save it in an external .ps1 file   
and execute it as a script.

## See also

* Looping through items in Chapter 1, PowerShell Key Concepts

Working with variables and objects in Chapter 1, PowerShell Key Concepts

# Configuring the mailbox database limits

The Exchange Management Shell provides cmdlets that allow you to configure the storage limits for mailbox databases. This recipe will show you how to set these limits interactively in the shell or in bulk using automated script.

## How to do it...

To configure the storage limits for a mailbox database, use the   
Set-MailboxDatabase cmdlet, for example:

Set-MailboxDatabase -Identity DB1 `

-IssueWarningQuota 2gb `

-ProhibitSendQuota 2.5gb `

-ProhibitSendReceiveQuota 3gb

## How it works...

In the example, we have configured the IssueWarningQuota, ProhibitSendQuota and ProhibitSendRecieveQuota limits for the DB1 mailbox database. These are the storage limits that will be applied to each mailbox that is stored in this database. Based on the values used with the command, you can see that users will receive a warning once their mailbox reaches 2 GB in size. When their mailbox reaches 2.5 GB, they will be unable to send outbound e-mail messages and when they hit the 3 GB limit they will be unable to send or receive e-mail messages.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | You can override the database limits on a per mailbox basis using the  Set-Mailbox cmdlet. |  |

## There's more…

Mailbox databases support deleted item retention, which allows you to recover items that have been removed from the deleted items folder. By default, the retention period for mailbox databases is set to 14 days, but this can be changed using the   
-DeletedItemRetention parameter when using the appropriate cmdlet. For example, to increase the deleted item retention period for the DB1 database,   
use the following command:

Set-MailboxDatabase -Identity DB1 -DeletedItemRetention 30

In this example, we have set the deleted item retention to 30 days. This parameter will also accept input in the form of a time span, and therefore can be specified using the dd.hh:mm:ss format. For example, we could have also used 30.00:00:00 as the parameter value, indicating that the deleted item retention should be 30 days, zero hours, zero minutes, and zero seconds, but that would be pointless in this example. However, this format is useful when you need to be specific about hours or minutes, for instance,   
using 12:00:00 would indicate that deleted items should only be retained for   
12 hours.

In addition to the deleted item retention, mailbox databases also retain deleted mailboxes for 30 days by default. You can change this value using the -MailboxRetention parameter as shown next:

Set-MailboxDatabase -Identity DB1 -MailboxRetention 90

Like the value used for the -DeletedItemRetention parameter, you can specify a time span as the value for the -MailboxRetention parameter. Both of these parameters will accept a maximum of 24,855 days.

Finally, you can configure mailbox databases so that items will not be permanently deleted until a database backup has been performed. This is not enabled by default. To turn it on for a particular database, use the -RetainDeletedItemsUntilBackup parameter with the Set-MailboxDatabase cmdlet. For example:

Set-MailboxDatabase -Identity DB1 `

-RetainDeletedItemsUntilBackup $true

### Taking it a step further

To configure these settings in bulk, we can make use of the pipeline to update the settings for a group of databases. For example, the following command will set the database limits for all mailboxes in the organization:

Get-MailboxDatabase | Set-MailboxDatabase `

-IssueWarningQuota 2gb `

-ProhibitSendQuota 2.5gb `

-ProhibitSendReceiveQuota 3gb `

-DeletedItemRetention 30 `

-MailboxRetention 90 `

-RetainDeletedItemsUntilBackup $true

In this command, we are piping the results of the Get-MailboxDatabase cmdlet to the Set-MailboxDatabase cmdlet and changing the default settings to the desired values   
for all databases in the organization.

## See also

Determining the average mailbox size per database

# Reporting on mailbox database size

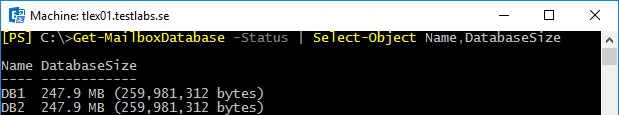
In this recipe, we are going to take a look at how we can verify the size of the mailbox databases. This is fairly simple and the information can easily be retrieved using the Get-MailboxDatabase cmdlet. In this recipe, we will take a look at how to report on mailbox database size using the Exchange Management Shell for Exchange 2016.

## How to do it...

To retrieve the total size for each mailbox database, use the following command:

Get-MailboxDatabase -Status | select-object Name,DatabaseSize

The output from this command might look something like this:



7081EN\_06\_02

## How it works...

When running the Get-MailboxDatabase cmdlet, we can use the -Status switch parameter to receive additional information about the database, such as the mount status, the backup status, and the total size of the database, as shown in the previous example. To generate a report with this information, simply pipe the command to the Export-CSV cmdlet and specify the path and filename, as shown:

Get-MailboxDatabase -Status |

select-object Name,Server,DatabaseSize,Mounted |

Export-CSV –Path c:\databasereport.csv -NoTypeInformation

This time, we have added the server name that the database is currently associated with   
and the mount status for that database.

## There's more...

When viewing the value for the database size, you probably noticed that we see the total size in megabytes and in parenthesis we see the value in bytes, rather than just seeing a single integer for the total size. The DatabaseSize property is of the type ByteQuantifiedSize, and we can use several methods provided by this type to convert the value if all we want to retrieve is a numeric representation of the database size.

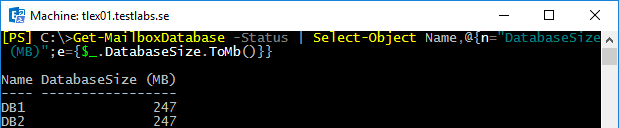
For example, we can use the ToKB, ToMB, ToGB and ToTB methods of the DatabaseSize object to convert the value to kilobytes, megabytes, gigabytes, or terabytes. For example:

Get-MailboxDatabase -Status |

Select-Object Name,

@{n="DatabaseSize (MB)";e={$\_.DatabaseSize.ToMb()}}

As you can see, this time we have created a calculated property for the DatabaseSize and we are using the ToMB method to convert the value of the database. The output we get from the command would look something like this:



7081EN\_06\_03

This technique may be useful if you are looking to generate basic reports and you don't need all of the extra information that is returned by default. For instance, you may already know that your databases will always be in the range of hundreds of gigabytes. You can simply use a calculated property as shown in the previous example and call the ToGB method for each DatabaseSize object.

## See also

Formatting output in Chapter 1, PowerShell Key Concepts

# Finding the total number of mailboxes in a database

You can retrieve all kinds of information about a mailbox database using the Exchange Management Shell cmdlets. Surprisingly, the total number of mailboxes in a given mailbox database is not one of those pieces of information. We need to retrieve this data manually. Luckily, PowerShell makes this easy, as you will see in this recipe.

## How to do it...

1. There are two ways that you can retrieve the total number of mailboxes in a database. First, we can use the Count property of a collection of mailboxes:

@(Get-Mailbox -Database DB1).count

1. Another way to retrieve this information is to use the Measure-Object cmdlet using the same collection from the preceding example:

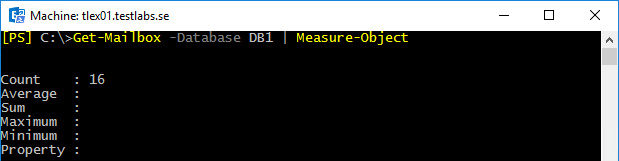
Get-Mailbox -Database DB1 | Measure-Object

## How it works...

In both steps, we use the Get-Mailbox cmdlet and specify the -Database parameter, which will retrieve all of the mailboxes in that particular database. In the first example, we have wrapped the command inside the @() characters to ensure that PowerShell will always interpret the output as an array. The reason for this is that if the mailbox database contains only one mailbox, the resulting output object will not be a collection, and thus will not have a Count property.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Remember, the default result size for  Get-Mailbox is 1000. Set the -ResultSize parameter to Unlimited to override this. |  |

The second step makes use of the Measure-Object cmdlet. You can see that, in addition to the Count property, we also get a number of other details. Consider the output as shown in the following screenshot:



7081EN\_06\_04

To retrieve only the total number of mailboxes, we can extend this command further in   
two ways. First, we can enclose the entire command in parenthesis and access the   
Count property:

(Get-Mailbox -Database DB1 | Measure-Object).Count

In this case, the preceding command would return only the total number of mailboxes in   
the DB1 database.

We can also pipe the command to Select-Object, and use the -ExpandProperty parameter to retrieve only the value of the Count property:

Get-Mailbox -Database DB1 |

Measure-Object |

Select-Object -ExpandProperty Count

This command would again only return the total number of mailboxes in the database.

One of the most common questions that comes up when people see both of these methods is, of course, which way is faster? Well, we can use the Measure-Command cmdlet to determine this information, but the truth is that your results will vary greatly and there probably won't be a huge difference in this case. The syntax to measure the time it takes   
to run a script or command is shown next:

Measure-Command -Expression {@(Get-Mailbox -Database DB1).Count}

Simply supply a script block containing the commands you want to measure and assign it to the -Expression parameter as shown previously. The Measure-Command cmdlet will return a TimeSpan object that reports on the total milliseconds, seconds, minutes that it took to complete the command. You can then compare these values to other commands that produce the same result but use alternate syntax or cmdlets.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | To report on the total number of archive mailboxes, use Get-Mailbox -Filter {ArchiveName -ne $null} | Measure-Object. |  |

## There's more...

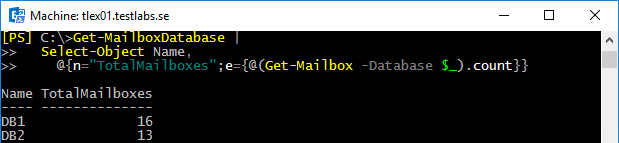
We can easily determine the total number of mailboxes in each database using a single command. The key to this is using the Select-Object cmdlet to create a calculated property. For example:

Get-MailboxDatabase |

Select-Object Name,

@{n="TotalMailboxes";e={@(Get-Mailbox -Database $\_).count}}

This command would generate output similar to the following:



7081EN\_06\_05

This command pipes the output from Get-MailboxDatabase to the Select-Object cmdlet. For each database output by the command, we select the database name and then use the $\_ object when creating the calculated property to determine the total number of mailboxes, using the Get-Mailbox cmdlet. This command can be piped further down to the Out-File or Export-CSV cmdlets that will generate a report saved in an external file.

## See also

* Creating custom objects in Chapter 1, PowerShell Key Concepts

# Determining the average mailbox size

# per database

PowerShell is very flexible and gives you the ability to generate very detailed reports. When generating mailbox database statistics, we can utilize data returned from multiple cmdlets provided by the Exchange Management Shell. This recipe will show you an example of this, and you will learn how to calculate the average mailbox size per database using PowerShell.

## How to do it...

To determine the average mailbox size for a given database, use the following one-liner:

Get-MailboxStatistics -Database DB1 |

ForEach-Object {$\_.TotalItemSize.Value.ToMB()} |

Measure-Object -Average |

Select-Object –ExpandProperty Average

## How it works...

Calculating an average is as simple as performing some basic math, but PowerShell gives   
us the ability to do this quickly with the Measure-Object cmdlet. The example uses the Get-MailboxStatistics cmdlet to retrieve all the mailboxes in the DB1 database. We then loop through each one, retrieving only the TotalItemSize property, and inside the ForEach-Object script block we convert the total item size to megabytes. The result from each mailbox can then be averaged using the Measure-Object cmdlet. At the end of the command, you can see that the Select-Object cmdlet is used to retrieve only the value for the Average property.

The number returned here will give us the average mailbox size in total for regular   
mailboxes, archive mailboxes, as well as any other type of mailbox that has been disconnected. If you want to be more specific, you can filter out these mailboxes   
after running the Get-MailboxStatistics cmdlet:

Get-MailboxStatistics -Database DB1 |

Where-Object{!$\_.DisconnectDate -and !$\_.IsArchive} |

ForEach-Object {$\_.TotalItemSize.Value.ToMB()} |

Measure-Object -Average |

Select-Object –ExpandProperty Average

Notice that, in the preceding example, we have added the Where-Object cmdlet to filter   
out any mailboxes that have a DisconnectDate defined or where the IsArchive property is $true.

Another thing that you may want to do is round the average. Let's say the DB1 database contained 16 mailboxes and the total size of the database was around 512 megabytes. The value returned from the preceding command would roughly look something like 4.33333333333333. Rarely are all those extra decimal places of any use. Here are   
a couple of ways to make the output a little cleaner:

$MBAvg = Get-MailboxStatistics -Database DB1 |

ForEach-Object {$\_.TotalItemSize.value.ToMB()} |

Measure-Object -Average |

Select-Object –ExpandProperty Average

[Math]::Round($MBAvg,2)

You can see that this time, we stored the result of the one-liner in the $MBAvg variable. We then use the Round method of the Math class in the .NET Framework to round the value, specifying that the result should only contain two decimal places. Based on the previous information, the result of the preceding command would be 4.35.

We can also use string formatting to specify the number of decimal places to be used:

[PS] "{0:n2}" -f $MBAvg

4.35

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | The -f Format operator is documented in PowerShell's help system in about\_operators. |  |

Keep in mind that this command will return a string, so if you need to be able to sort on this value, cast it to double:

[PS] [double]("{0:n2}" -f $MBAvg)

4.35

## There's more...

The previous examples have only shown how to determine the average mailbox size for a single database. To determine this information for all mailbox databases, we can use the following code (save it to a file called size.ps1):

foreach($DB in Get-MailboxDatabase) {

Get-MailboxStatistics -Database $DB |

ForEach-Object{$\_.TotalItemSize.value.ToMB()} |

Measure-Object -Average |

Select-Object @{n="Name";e={$DB.Name}},

@{n="AvgMailboxSize";e={[Math]::Round($\_.Average,2)}} |

Sort-Object AvgMailboxSize -Desc

}

The result of this command would look something like this:



7081EN\_06\_06

This example is very similar to the one we looked at previously. The difference is that, this time, we are running our one-liner using a foreach loop for every mailbox database in the organization. When each mailbox database has been processed, we sort the output based on the AvgMailboxSize property.

## See also

Creating custom objects in Chapter 1, PowerShell Key Concepts

# Reporting on database backup status

Using the Exchange Management Shell, we can write scripts that will check on the last full backup time for a database that can be used for monitoring and reporting. In this recipe, you will learn how to check the last backup time for each database and use this information to generate statistics and find databases that are not being backed up on a regular basis.

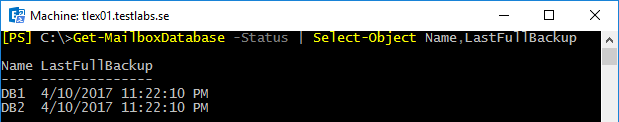
## How to do it...

To check the last full backup time for a database, use the Get-MailboxDatabase cmdlet, as shown:

Get-MailboxDatabase -Identity DB1 -Status | fl Name,LastFullBackup

## How it works...

When you run the Get-MailboxDatabase cmdlet, you must remember to use the -Status switch parameter or else the LastFullBackup property will be $null. In the previous example, we checked the last full backup for the DB1 database and piped the output to the Format-List (using the fl alias) cmdlet. When viewing the LastFullBackup for each database, you might find it helpful to pipe the output to the Select-Object cmdlet, as shown in the following screenshot:



7081EN\_06\_07

In addition to simply checking the date, it may be useful to schedule this script to run daily and report on the databases that have not recently been backed up. For example, the following command will only retrieve databases that have not had a successful full back up in the last 24 hours:

Get-MailboxDatabase -Status |

?{$\_.LastFullBackup -le (Get-Date).AddDays(-1)} |

Select-object Name,LastFullBackup

Here you can see that the Get-MailboxDatabase output is piped to the Where-Object cmdlet (using the ? alias) and we check the value of the LastFullBackup property for each database. If the value is less than or equal to 24 hours ago, the database name and last full backup time are retuned.

## There's more...

Since the LastFullBackup property value is a DateTime object, not only can we use comparison operators to find databases that have not been backed up within a certain time frame, but we can also calculate the number of days since that time. This might be a useful piece of information to add to a reporting or monitoring script. The following code will provide this information:

Get-MailboxDatabase -Status | ForEach-Object {

if(!$\_.LastFullBackup) {

$LastFull = "Never"

}

else {

$LastFull = $\_.LastFullBackup

}

New-Object PSObject -Property @{

Name = $\_.Name

LastFullBackup = $LastFull

DaysSinceBackup = if($LastFull-is [datetime]) {

(New-TimeSpan $LastFull).Days

}

Else {

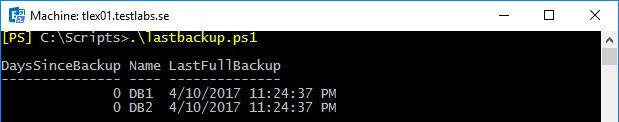
$LastFull

}

}

}

When running this code in the Exchange Management Shell, you would see output similar to the following:



7081EN\_06\_08

As you can see, we are simply looping through each mailbox database and retrieving the LastFullBackup time. If a database has never been backed up, the value will be $null. With that in mind, this code will return the string Never for those databases when reporting on the status. If a value is present for LastFullBackup, we use the New-TimeSpan cmdlet to determine the number of days since the last backup and include that in the data returned.

## See also

Creating custom objects in Chapter 1, PowerShell Key Concepts

# Restoring data from a recovery database

When it comes to recovering data from a failed database, you have several options depending on what kind of backup product you are using or how you have deployed Exchange 2016. The ideal method for enabling redundancy is to use a DAG, which will replicate your mailbox databases to one or more servers and provide automatic failover in the event of a disaster. However, you may need to pull old data out of a database restored from a backup. In this recipe, we will take a look at how you can create a recovery database and restore data from it using the Exchange Management Shell.

## How to do it...

First, restore the failed database using the steps required by your current backup solution. For this example, let's say that we have restored the DB1 database file to E:\Recovery\DB1 and the database has been brought to a clean shutdown state. We can use the following steps to create a recovery database and restore mailbox data:

1. Create a recovery database using the New-MailboxDatabase cmdlet:

New-MailboxDatabase -Name RecoveryDB `

-EdbFilePath E:\Recovery\DB1\DB1.edb `

-LogFolderPath E:\Recovery\DB1 `

-Recovery `

-Server MBX1

1. When you run the preceding command, you will see a warning that the recovery database was created using the existing database file. The next step is to check the state of the database, followed by mounting the database:

Eseutil /mh .\DB1.edb

Eseutil /R E00 /D

Mount-Database -Identity RecoveryDB

Next, Query the recovery database for all mailboxes that resides in the database RecoveryDB:

Get-MailboxStatistics –Database RecoveryDB | fl DisplayName,MailboxGUID,LegacyDN

Lastly, we will use the New-MailboxRestoreRequest cmdlet to restore the data from the recovery database for a single mailbox:

New-MailboxRestoreRequest -SourceDatabase RecoveryDB `

-SourceStoreMailbox "Joe Smith" `

-TargetMailbox joe.smith

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | When running the eseutil commands, make sure to be in the folder where the restored mailbox database and logs are placed.  Also make sure that the name of the recovery database is unique. |  |

## How it works...

When you restore the database file from your backup application, you may need to ensure that the database is in a clean shutdown state. For example, if you are using Windows Server Backup for your backup solution, you will need to use the Eseutil.exe database utility to play any uncommitted logs into the database to get it in a clean shutdown state.

Once the data is restored, we can create a recovery database using the   
New-MailboxDatabase cmdlet, as shown in the first example. Notice that when we ran the command we used several parameters. First, we specified the path to the EDB file and the log files, both of which are in the same location where we restored the files. We have also used the -Recovery switch parameter to specify that this is a special type of database that will only be used for restoring data and should not be used for production mailboxes. Finally, we specified which mailbox server the database should be hosted on using the -Server parameter. Make sure to run the New-MailboxDatabase cmdlet from the mailbox server that you are specifying in the -Server parameter, and then mount the database using the Mount-Database cmdlet.

The last step is to restore data from one or more mailboxes. As we saw in the previous example, the New-MailboxRestoreRequest is the tool to use for this task. This cmdlet was introduced in Exchange 2010 SP1, so if you have used this process in the past, the procedure is the same with Exchange 2013.

## There's more…

When you run the New-MailboxRestoreRequest cmdlet, you need to specify the identity of the mailbox you wish to restore using the -SourceStoreMailbox parameter. There are three possible values you can use to provide this information: DisplayName, MailboxGuid and LegacyDN. To retrieve these values, you can use the Get-MailboxStatistics cmdlet once the recovery database is online and mounted:

Get-MailboxStatistics -Database RecoveryDB | `

fl DisplayName,MailboxGUID,LegacyDN

Here we have specified that we want to retrieve all three of these values for each mailbox in the RecoveryDB database.

### Understanding target mailbox identity

When restoring data with the New-MailboxRestoreRequest cmdlet, you also need to provide a value for the -TargetMailbox parameter. The mailbox needs to already exist before running this command. If you are restoring data from a backup for an existing mailbox that has not changed since the backup was done, you can simply provide the typical identity values for a mailbox for this parameter.

If you want to restore data to a mailbox that was not the original source of the data, you need to use the -AllowLegacyDNMismatch switch parameter. This will be useful if you are restoring data to another user's mailbox, or if you've recreated the mailbox since the backup was taken.

### Learning about other useful parameters

The New-MailboxRestoreRequest cmdlet can be used to granularly control how data is restored out of a mailbox. The following parameters may be useful to customize the behaviour of the restores:

* ConflictResolutionOption: This parameter specifies the action to take if   
  multiple matching messages exist in the target mailbox. The possible values are KeepSourceItem, KeepLatestItem or KeepAll. If no value is specified, KeepSourceItem will be used by default.
* ExcludeDumpster Use this switch parameter to indicate that the dumpster should not be included in the restore.
* SourceRootFolder: Use this parameter to restore data only from a root folder of a mailbox.
* TargetIsArchive: You can use this switch parameter to perform a mailbox restore to a mailbox archive.

TargetRootFolder: This parameter can be used to restore data to a specific folder in the root of the target mailbox. If no value is provided, the data is restored and merged into the existing folders, and, if they do not exist, they will be created in the target mailbox.

These are just a few of the useful parameters that can be used with this cmdlet, but there are more. For a complete list of all the available parameters and full details on each one, run Get-Help New-MailboxRestoreRequest -Detailed

### Understanding mailbox restore request cmdlets

There is an entire cmdlet set for mailbox restore requests in addition to the   
New-MailboxRestoreRequest cmdlet. The remaining available cmdlets   
are outlined as follows:

* Get-MailboxRestoreRequest: Provides detailed status of mailbox   
  restore requests
* Remove-MailboxRestoreRequest: Removes fully or partially completed restore requests
* Resume-MailboxRestoreRequest: Resumes a restore request that was suspended or failed
* Set-MailboxRestoreRequest: Can be used to change the restore request options after the request has been created

Suspend-MailboxRestoreRequest: Suspends a restore request any time after the request was created but before the request reaches the status of Completed

For complete details and examples for each of these cmdlets, use the Get-Help cmdlet with the appropriate cmdlet using the -Full switch parameter.

### Taking it a step further

Let's say that you have restored your database from backup, you have created a recovery database, and now you need to restore each mailbox in the backup to the corresponding target mailboxes that are currently online. We can use the following script to accomplish this:

$mailboxes = Get-MailboxStatistics -Database RecoveryDB

foreach($mailbox in $mailboxes) {

New-MailboxRestoreRequest -SourceDatabase RecoveryDB `

-SourceStoreMailbox $mailbox.DisplayName `

-TargetMailbox $mailbox.DisplayName

}

Here you can see that first we use the Get-MailboxStatistics cmdlet to retrieve all the mailboxes in the recovery database and store the results in the $mailboxes variable. We then loop through each mailbox and restore the data to the original mailbox. You can track the status of these restores using the Get-MailboxRestoreRequest cmdlet and the Get-MailboxRestoreRequestStatistics cmdlet.

## See also

Managing disconnected mailboxes in Chapter 4, Managing Mailboxes

Importing and exporting mailboxes in Chapter 4, Managing Mailboxes