Send-MailMessage –SmtpServer “smtpserver” –From [DoNotReply@hotmail.com](mailto:DoNotReply@hotmail.com) –To [testing@hotmail.com](mailto:testing@hotmail.com) –Subject “Testing e-mail” –Body “Body of the message”

$smtpserver = “smtp.lab.com”

$to = [testing@hotmail.com](mailto:testing@hotmail.com)

$from = [sending@hotmail.com](mailto:sending@hotmail.com)

$file = “C:\file.txt” #attachment

$subject = “Testing”

$message = New-Object Net.Mail.MailMessage

$smtp = New-Object Net.Mail.SmtpClient($smtpserver)

$message.From = $from

$message.To.Add($to)

$att = New-Object Net.Mail.Attachment($file)

$message.IsBodyHtml = $False

$message.Subject = $subject

$message.Attachments.Add($att)

$smtp.Send($message)

# Sending a Formatted Message Body:

$smtpserver = “smtp.lab.com”

$to =

$from =

$subject = “Testing”

$message = @”

Hello,

Line1

Line2

Line3

“@

Send-MailMessage –SmtpServer $smtpserver –From $from –To $to

-Subject $subject –Body $message

# Sending HTML

HTML Online Editor:

<https://html-online.com/editor/>

$smtpserver = "smtp.lab.com"

$to = "sukhija@techwizard.cloud"

$from = "DonotReply@labtest.com"

$subject = "Test Subject"

$message = @"

<!-- ####### YAY, I AM THE SOURCE EDITOR! #########-->

<h1>You can edit <span>this demo</span> text!</h1>

<h2>How to use the editor:</h2>

<p>Paste your documents in the visual editor on the left or your HTML code in the source editor in the right. <br />Edit any of the two areas and see the other changing in real time.&nbsp;</p>

<p>&nbsp;</p>

"@

Send-MailMessage -SmtpServer $smtpserver -From $from -To $to -Subject $subject -Body $message –BodyAsHtml

# Sending Errors via Email

$from = "donotreply@lab.com"

$to="vikas@lab.com"

$subject = "Error has occured"

$smtpServer="smtp.lab.com"

if ($error)

{

Send-MailMessage -SmtpServer $smtpserver -From $from -To $to -Subject $subject -Body $error[0].ToString()

$error.clear()

}

# Send-Email Function to Send an $error Array

function Send-Email

{

[CmdletBinding()]

param

(

[Parameter(Mandatory = $true)]

$From,

[Parameter(Mandatory = $true)]

[array]$To,

[array]$bcc,

[array]$cc,

$body,

$subject,

$attachment,

[Parameter(Mandatory = $true)]

$smtpserver

)

$message = New-Object System.Net.Mail.MailMessage

$message.From = $From

if ($To -ne $null)

{

$To | ForEach-Object{

$to1 = $\_

$to1

$message.To.Add($to1)

}

}

if ($cc -ne $null)

{

$cc | ForEach-Object{

$cc1 = $\_

$cc1

$message.CC.Add($cc1)

}

}

if ($bcc -ne $null)

{

$bcc | ForEach-Object{

$bcc1 = $\_

$bcc1

$message.bcc.Add($bcc1)

}

}

$message.IsBodyHtml = $true

if ($subject -ne $null)

{$message.Subject = $subject}

if ($attachment -ne $null)

{

$attach = New-Object Net.Mail.Attachment($attachment)

$message.Attachments.Add($attach)

}

if ($body -ne $null)

{$message.body = $body}

$smtp = New-Object Net.Mail.SmtpClient($smtpserver)

$smtp.Send($message)

}

# Sending $error Array in an Email

$from = "donotreply@lab.com"

$to="vikas@lab.com"

$subject = "Error has occured"

$smtpServer="smtp.lab.com"

if ($error)

{

Send-Email -smtpserver $smtpServer -From $from -To $to -subject $subject -body $error

$error.clear()

}

## Logging Everything Including Errors

There is an built-in PowerShell cmdlet that you can use at the beginning of your script and stop at the end of the script. This is often called transcript logging.

Start-transcript # at the beginning of the script

Stop-transcript # at the end of the script

This log will by default get stored in the running account's My Documents folder. To store the transcript at a different location, you can specify the path parameter as shown in [Figure 7-1](https://cdn2.percipio.com/1651640039.ab7862f216d102e2482406b4b912b6636c073c64/eod/books/158621/OEBPS/section-36.xhtml#ch07Fig1).

$log = "c:\data\log.txt"

Start-transcript -path $log # at the beginning of the script

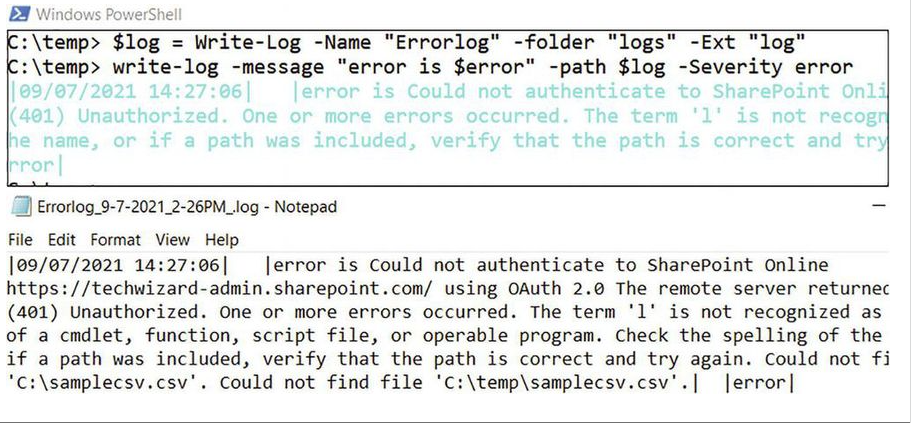
Stop-transcript # at the end of the script

## Logging Errors to a Text File

You can also just log errors in text files. This cheat guide describes a Write-Log function; the same can be utilized to log errors in text files. The result is shown in [Figure 7-2](https://cdn2.percipio.com/1651640039.ab7862f216d102e2482406b4b912b6636c073c64/eod/books/158621/OEBPS/section-37.xhtml#ch07Fig2).

$log = Write-Log -Name "Errorlog" -folder "logs" -Ext "log"

write-log -message "error is $error" -path $log -Severity error

Figure 7-2: Showing the error logging in a text file

# Exchange Server Reporting:

**CSV Report**

Export-CSV is the built-in PowerShell cmdlet that can be used to export the data to a CSV file. You can simply **pipe select and then export** as shown in the following code that shows the export of certain attributes of users' mailboxes from Exchange Server (to be run in the Exchange shell):

Get-Mailbox -ResultSize unlimited | Select Name,identity, WindowsEmailAddress,Database,ProhibitSendQuota,ProhibitSendReceiveQuota,IssueWarningQuota | export-csv c:\mailboxes.csv –notypeinfo

# Exporting to CSV when Fetching from multiple sources:

$collection=@() #array to collect report data

$data = get-content .\users.txt #read samaccountname from text file

$data | foreach-object{

$coll = "" | Select Name,identity,WindowsEmailAddress,Database,ProhibitSendQuota,ProhibitSendReceiveQuota,IssueWarningQuota,employeeid, l,C #values needed in report

$getmbx = get-mailbox -identity $\_

$getaduser = get-aduser -identity $\_ -properties employeeid, l,C

$coll.Name = $getmbx.Name

$coll.identity = $getmbx.identity

$coll.WindowsEmailAddress = $getmbx.WindowsEmailAddress

$coll.Database= $getmbx.Database

$coll.ProhibitSendQuota = $getmbx.ProhibitSendQuota

$coll.ProhibitSendReceiveQuota = $getmbx.ProhibitSendReceiveQuota

$coll.IssueWarningQuota = $getmbx.IssueWarningQuota

$coll.employeeid = $getaduser.employeeid #note difference here

$coll.l = $getaduser.l

$coll.c = $getaduser.c

$collection+=$coll #add the collected values to the collecttion array

}

#now export to CSV file

$collection | Export-Csv .\report.csv –NoTypeInformation

# Another important aspect of CSV reporting is to export multi-valued attributes. Here is an example of extracting recipients (which is a multi-valued attribute) in Exchange tracking logs:

@{Name="Recipents";Expression={$\_.recipients}}

See [Listing 8-2](https://cdn2.percipio.com/1651640039.ab7862f216d102e2482406b4b912b6636c073c64/eod/books/158621/OEBPS/section-40.xhtml#ls8-2) for an example of extracting recipient values from Exchange transport logs:

Get-transportserver | Get-MessageTrackingLog -Start"03/09/2015 00:00:00 AM" -End"03/09/2015 11:59:59 PM" -sender "vikas@lab.com" -resultsize unlimited | '

select-object Timestamp,clientip,ClientHostname,ServerIp,ServerHostname,sender,EventId,MessageSubject, TotalBytes, SourceContext,ConnectorId,Source, '

InternalMessageId, MessageId ,@{Name="Recipents";Expression={$\_.recipients}} | '

export-csv c:\track.csv

## Excel Reporting

Although CSV reports are fine for most purposes, there are situations in which you want to share the data with your managers so converting the CSV file to Excel is a much-needed script. I will share two methods for doing the same.

The first method exists in the vsadmin module that was shared in the modules chapter.

**Note**

Excel should be installed on the machine to use this method.

[Listing 8-3](https://cdn2.percipio.com/1651640039.ab7862f216d102e2482406b4b912b6636c073c64/eod/books/158621/OEBPS/section-41.xhtml#ls8-3) shows the code of the Save-CSV2Excel function in case you do not have the vsadmin module installed or do not want to use it.

**Listing 8-3: Cheat Code for the Save-CSV2Excel Function**

Function Save-CSV2Excel

{

[CmdletBinding()]

Param(

[Parameter(Mandatory = $true,Position = 1)]

[ValidateScript({

if(-Not ($\_ | Test-Path) ){throw "File or folder does not exist"}

if(-Not ($\_ | Test-Path -PathType Leaf) ){throw "The Path argument must be a file. Folder paths are not allowed."}

if($\_ -notmatch "(\.csv)"){throw "The file specified in the path argument must be either of type csv"}

return $true

})]

[System.IO.FileInfo]$CSVPath,

[Parameter(Mandatory = $true)]

[ValidateScript({

if($\_ -notmatch "(\.xlsx)"){throw "The file specified in the path argument must be either of type xlsx"}

return $true

})]

[System.IO.FileInfo]$Exceloutputpath

)

####### Borrowed function from Lloyd Watkinson from script gallery##

Function Convert-NumberToA1

{

Param([parameter(Mandatory = $true)]

[int]$number)

$a1Value = $null

While ($number -gt 0)

{

$multiplier = [int][system.math]::Floor(($number / 26))

$charNumber = $number - ($multiplier \* 26)

If ($charNumber -eq 0) { $multiplier-- ; $charNumber = 26 }

$a1Value = [char]($charNumber + 64) + $a1Value

$number = $multiplier

}

Return $a1Value

}

##################Start converting excel#######################

$importcsv = Import-Csv $CSVPath

$countcolumns = ($importcsv |

Get-Member |

Where-Object{$\_.membertype -eq "Noteproperty"}).count

#################call Excel com object ##############

$xl = New-Object -comobject excel.application

$xl.visible = $false

$Workbook = $xl.workbooks.open($CSVPath)

$Workbook.SaveAs($Exceloutputpath, 51)

$Workbook.Saved = $true

$xl.Quit()

#############Now format the Excel###################

timeout.exe 10 #wait for 10 seconds before saving

$xl = New-Object -comobject excel.application

$xl.visible = $false

$Workbook = $xl.workbooks.open($Exceloutputpath)

$worksheet1 = $Workbook.worksheets.Item(1)

for ($c = 1; $c -le $countcolumns; $c++) {$worksheet1.Cells.Item(1, $c).Interior.ColorIndex = 39}

$colvalue = (Convert-NumberToA1 $countcolumns) + "1"

$headerRange = $worksheet1.Range("a1", $colvalue)

$null = $headerRange.AutoFilter()

$null = $headerRange.entirecolumn.AutoFit()

$worksheet1.rows.item(1).Font.Bold = $true

$Workbook.Save()

$Workbook.Close()

$xl.Quit()

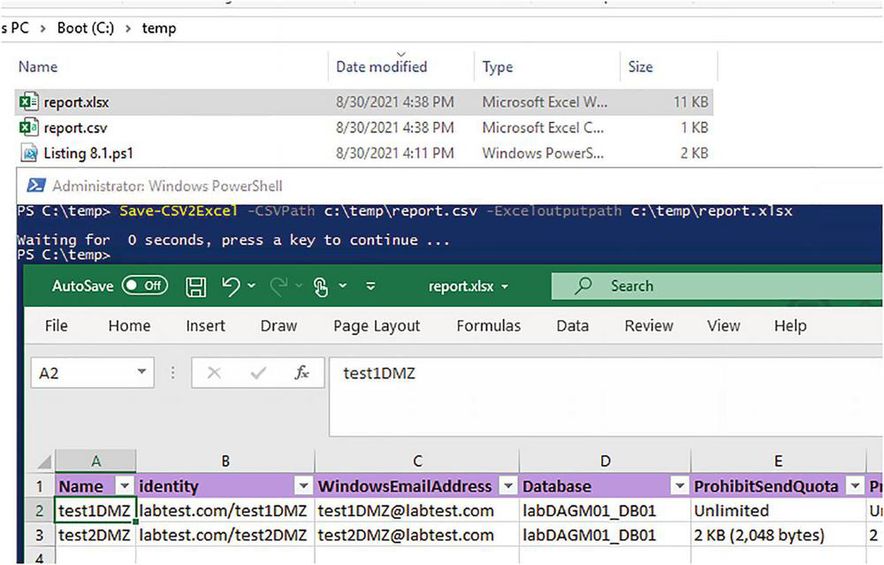
$Null = [System.Runtime.Interopservices.Marshal]::ReleaseComObject($xl)

#######################################################################

}#Write-CSV2Excel

Let's use the CSV report from [Listing 8-1](https://cdn2.percipio.com/1651640039.ab7862f216d102e2482406b4b912b6636c073c64/eod/books/158621/OEBPS/section-40.xhtml#ls8-1) and convert it to Excel using Save-CSV2Excel**.** See [Figure 8-2](https://cdn2.percipio.com/1651640039.ab7862f216d102e2482406b4b912b6636c073c64/eod/books/158621/OEBPS/section-41.xhtml#ch08Fig2).

Save-CSV2Excel -CSVPath c:\temp\report.csv -Exceloutputpath c:\temp\report.xlsx

Figure 8-2: Showing a CSV-to-Excel conversion

There is a module named ImportExcel. It is one of the most popular modules in the PowerShell Gallery. You can utilize this module to directly convert variables to Excel. Get it from [www.powershellgallery.com/packages/ImportExcel](http://www.powershellgallery.com/packages/ImportExcel). Install the module on your machine and then import it to utilize it:

Install-Module -Name ImportExcel

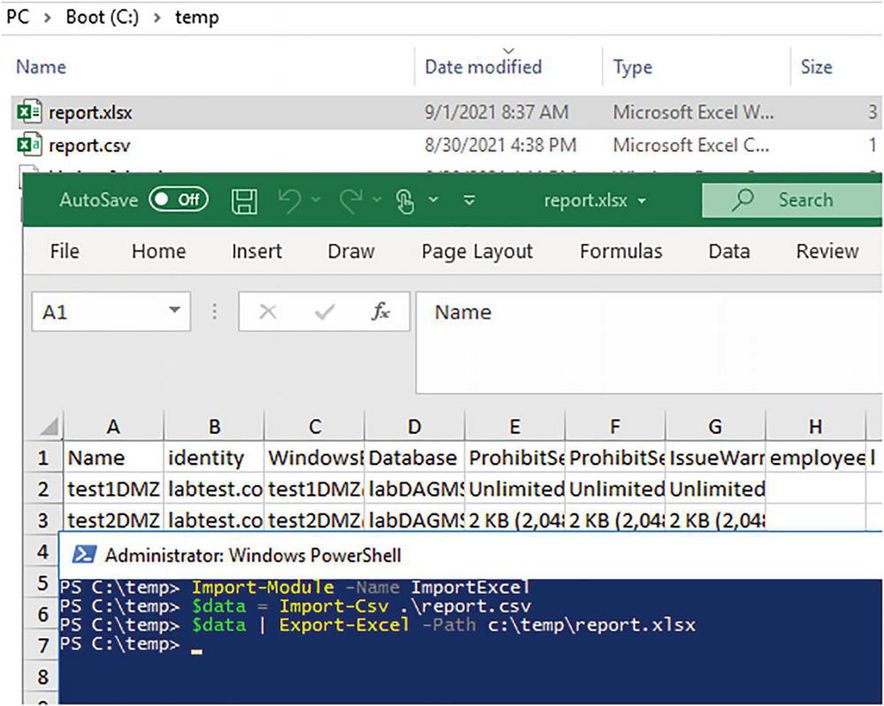
Let's use the same report and use this new module to convert it into Excel. The advantage of using this module is that it does not require Excel to be installed on the machine. See [Figure 8-3](https://cdn2.percipio.com/1651640039.ab7862f216d102e2482406b4b912b6636c073c64/eod/books/158621/OEBPS/section-41.xhtml#ch08Fig3).

Import-Module -Name ImportExcel

$data = Import-Csv .\report.csv

$data | Export-Excel -Path c:\temp\report.xlsx

Figure 8-3: Using the Import-Excel module



## HTML Reporting

It would be wonderful if we could create HTML dashboards with PowerShell ☺ that can show traffic light-type signals. For example, if a service is down, it shows red. Otherwise, it shows green. See [Figure 8-4](https://cdn2.percipio.com/1651640039.ab7862f216d102e2482406b4b912b6636c073c64/eod/books/158621/OEBPS/section-42.xhtml#ch08Fig4).

Figure 8-4: An HTML table report

[Listing 8-4](https://cdn2.percipio.com/1651640039.ab7862f216d102e2482406b4b912b6636c073c64/eod/books/158621/OEBPS/section-42.xhtml#ls8-4) is a template for HTML coding that you can use inside scripts and do traffic light-type operations based on conditions.

**Listing 8-4: Template for HTML Coding**

$report = $reportpath

Clear-Content $report

#################HTml Report Content############################

Add-Content $report "<html>"

Add-Content $report "<head>"

Add-Content $report "<meta http-equiv='Content-Type' content='text/html; charset=iso-8859-1'>"

Add-Content $report '<title>Exchange Status Report</title>'

add-content $report '<STYLE TYPE="text/css">'

add-content $report "<!--"

add-content $report "td {"

add-content $report "font-family: Tahoma;"

add-content $report "font-size: 11px;"

add-content $report "border-top: 1px solid #999999;"

add-content $report "border-right: 1px solid #999999;"

add-content $report "border-bottom: 1px solid #999999;"

add-content $report "border-left: 1px solid #999999;"

add-content $report "padding-top: 0px;"

add-content $report "padding-right: 0px;"

add-content $report "padding-bottom: 0px;"

add-content $report "padding-left: 0px;"

add-content $report "}"

add-content $report "body {"

add-content $report "margin-left: 5px;"

add-content $report "margin-top: 5px;"

add-content $report "margin-right: 0px;"

add-content $report "margin-bottom: 10px;"

add-content $report ""

add-content $report "table {"

add-content $report "border: thin solid #000000;"

add-content $report "}"

add-content $report "-->"

add-content $report "</style>"

Add-Content $report "</head>"

Add-Content $report "<body>"

add-content $report "<table width='100%'>"

add-content $report "<tr bgcolor='Lavender'>"

add-content $report "<td colspan='7' height='25' align='center'>"

add-content $report "<font face='tahoma' color='#003399' size='4'><strong>DAG Active Manager</strong></font>"

add-content $report "</td>"

add-content $report "</tr>"

add-content $report "</table>"

add-content $report "<table width='100%'>"

Add-Content $report "<tr bgcolor='IndianRed'>"

Add-Content $report "<td width='10%' align='center'><B>Identity</B></td>"

Add-Content $report "<td width='5%' align='center'><B>PrimaryActiveManager</B></td>"

Add-Content $report "<td width='20%' align='center'><B>OperationalMachines</B></td>"

Add-Content $report "</tr>"

##############################Report Template##################################

add-content $report "<tr bgcolor='Lavender'>"

add-content $report "<td colspan='7' height='25' align='center'>"

add-content $report "<font face='tahoma' color='#003399' size='4'><strong>DAG Database Backup Status</strong></font>"

add-content $report "</td>"

add-content $report "</tr>"

add-content $report "</tr>"

add-content $report "</table>"

add-content $report "<table width='100%'>"

Add-Content $report "<tr bgcolor='IndianRed'>"

Add-Content $report "<td width='10%' align='center'><B>Database</B></td>"

Add-Content $report "<td width='5%' align='center'><B>BackupInProgress</B></td>"

Add-Content $report "<td width='10%' align='center'><B>SnapshotLastFullBackup</B></td>"

Add-Content $report "<td width='5%' align='center'><B>SnapshotLastCopyBackup</B></td>"

Add-Content $report "<td width='10%' align='center'><B>LastFullBackup</B></td>"

Add-Content $report "<td width='5%' align='center'><B>RetainDeletedItemsUntilBackup</B></td>"

$dbst= Get-MailboxDatabase | where{$\_.MasterType -like "DatabaseAvailabilityGroup"}

$dbst | foreach{$st=Get-MailboxDatabase $\_ -status

$dbname = $st.Name

$dbbkprg = $st.BackupInProgress

$dbsnpl = $st.SnapshotLastFullBackup

$dbsnplc= $st.SnapshotLastCopyBackup

$dblfb = $st.LastFullBackup

$dbrd = $st.RetainDeletedItemsUntilBackup

Add-Content $report "<tr>"

Add-Content $report "<td bgcolor= 'GainsBoro' align=center> <B>$dbname</B></td>"

Add-Content $report "<td bgcolor= 'GainsBoro' align=center> <B>$dbbkprg</B></td>"

Add-Content $report "<td bgcolor= 'GainsBoro' align=center> <B>$dbsnpl</B></td>"

Add-Content $report "<td bgcolor= 'GainsBoro' align=center> <B>$dbsnplc</B></td>"

if($dblfb -lt $hrs)

{

Add-Content $report "<td bgcolor= 'Red' align=center> <B>$dblfb</B></td>"

}

else

{

Add-Content $report "<td bgcolor= 'Aquamarine' align=center> <B>$dblfb</B></td>"

}

Add-Content $report "<td bgcolor= 'GainsBoro' align=center> <B>$dbrd</B></td>"

Add-Content $report "</tr>"

}

##############################################################

Add-content $report "</table>"

Add-Content $report "</body>"

Add-Content $report "</html>"

See examples at the following links where this template has been successfully utilized for the Exchange Health Check, AD Health Check, and Monitor Remote services:

[https://techwizard.cloud/exchange-2010-health-check/](http://techwizard.cloud/exchange-2010-health-check/)

[https://techwizard.cloud/adhealthcheck/](http://techwizard.cloud/adhealthcheck/)

[https://techwizard.cloud/monitor-windows-services-status-remotely/](http://techwizard.cloud/monitor-windows-services-status-remotely/)

As mentioned, you can use the HTML Online Editor to create HTML and use it in your PowerShell scripts ([https://html-online.com/editor/](http://html-online.com/editor/)).